UNIVERSITY OF NIŠ
FACULTY OF PHILOSOPHY
Department of Mathematics

FILOMAT

9:2 (1995)

Conference Filomat '94
GEOMETRY & COMPUTER SCIENCE

Guest Editors: S. Minčić and Lj. Kocić

УНИВЕРЗИТЕТ У НИШУ - ФИЛОЗОФСКИ ФАКУЛТЕТ

NIŠ

FILOMAT

a continuation of
ZBORNIK RADOVA FILOZOFSKOG FAKULTETA U NIŠU
SERIJA MATEMATIKA

This publication was supported by the Faculty of Philosophy, University of Niš

Published by the Faculty of Philosophy, University of Niš

Printed by "Prosveta", Niš

UNIVERZITET U NIŠU
FILOZOFSKI FAKULTET

FILOMAT

9:2 (1995)

Konferencija Filomat '94

Geometrija. Računarstvo i informatika

Urednici: S. Minčić i Lj. Kocić

NIŠ, 1995

Conference FILOMAT '94

(October 22-24, 1994, Niš)

Department of Mathematics of the Philosophical Faculty, University of Niš, organized the mathematical conference „Filomat '94 " from October 22 to October 24, 1994, with two sections:

1. Geometry (The 10th Yugoslav Meeting of Geometricians),
2. Computer Science (Geometric Modeling, Numerical and System Software, Networks)

About 80 mathematicians from Yugoslavia and other countries have taken part on the conference, with about 60 talks. Some of them are published in this volume of „Filomat".

Chairman of the Organizing Committee
Svetislav Minčić

The organization of the conference „FILOMAT '94" was helped by the sponsors:

-Ministry of Science and Technology of Serbia

-EI Sigraf d.o.o.

-Libis Computers

-STN Slavia Trade, Niš

-WB Osiguranje

-Yugoduvan, Niš

GEOMETRY

FILOMAT (Niš) 9:2(1995), 117–120
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

AN INEQUALITY FOR THE TRIANGLE

Momčilo Bjelica

ABSTRACT. *Inequalities for the triangle in the most of cases become equalities for the equilateral triangle* [2], [5]. *In this article is given an inequality with unique property that it becomes equality for isoscales and rectangular triangles. Also, an inequality connected with Karamata's inequality is given.*

Theorem 1. *Let a, b, c, α, β, γ are the sides and angles of a triangle respectively and R the radius of its circumcircle. Then*

$$2R \geq \frac{b^2+c^2}{\sqrt{2b^2+2c^2-a^2}}, \tag{1}$$

equality holds if and only if $b = c$ or $\alpha = \frac{\pi}{2}$.

$$2R \geq \max\left\{\frac{b^2+c^2}{\sqrt{2b^2+2c^2-a^2}}, \frac{c^2+a^2}{\sqrt{2c^2+2a^2-b^2}}, \frac{a^2+b^2}{\sqrt{2a^2+2b^2-c^2}}\right\}, \tag{2}$$

equality holds if and only if the triangle is isosceles or rectangular.

Lemma.

$$\begin{gathered} |\cos\alpha| \geq \frac{|b^2+c^2-a^2|}{b^2+c^2}, \quad \sin\alpha \leq a\frac{\sqrt{2b^2+2c^2-a^2}}{b^2+c^2}, \\ |\tan\alpha| \leq a\frac{\sqrt{2b^2+2c^2-a^2}}{|b^2+c^2-a^2|}. \end{gathered} \tag{3}$$

If $\alpha \leq \frac{\pi}{2}$, then

$$\begin{gathered} \cos\frac{\alpha}{2} \geq \frac{\sqrt{2b^2+2c^2-a^2}}{\sqrt{2b^2+2c^2}}, \quad \sin\frac{\alpha}{2} \leq \frac{a}{\sqrt{2b^2+2c^2}}, \\ \tan\frac{\alpha}{2} \leq \frac{a}{\sqrt{2b^2+2c^2-a^2}}, \end{gathered} \tag{4}$$

1991 *Mathematics Subject Classification.* 51M16.

and conversely for $\alpha \geq \frac{\pi}{2}$. Equalities hold if and only if $b = c$ or $\alpha = \frac{\pi}{2}$.

Proof. Inequality (3.1) is equivalent to $|\cos\alpha|(b-c)^2 \geq 0$, which becomes equality if and only if $b = c$ or $\alpha = \frac{\pi}{2}$. (3.2) is equivalent to (3.1), and (3.3) is their consequence. Using $2R\sin\alpha = a$ one obtains that (1) is equivalent to (3.2). Since $\cos\alpha \geq (b^2+c^2-a^2)/(b^2+c^2)$ if $\alpha \leq \frac{\pi}{2}$, and conversely if $\alpha \geq \frac{\pi}{2}$, inequalities (4) follow. $\square$

The inequality of I. J. Schoenberg [4] for the two-dimensional euclidean space reads as follows: If λ_1, λ_2, λ_3 are real numbers, then

$$(\lambda_1+\lambda_2+\lambda_3)^2R^2 \geq \lambda_2\lambda_3a^2+\lambda_3\lambda_1b^2+\lambda_1\lambda_2c^2. \tag{5}$$

Introduce the functional

$$f(\lambda_1,\lambda_2,\lambda_3) = \lambda_2\lambda_3a^2+\lambda_3\lambda_1b^2+\lambda_1\lambda_2c^2$$

and consider now the inequality (5) with two equal parameters. The functional $f(\lambda_1,\lambda_2,\lambda_2)$, $\lambda_1+2\lambda_2 = \text{const.}$ has a maximum if

$$\lambda_1(b^2+c^2)+2\lambda_2(a^2-b^2-c^2) = 0,$$

$$\lambda_1 = 2\lambda_2\frac{b^2+c^2-a^2}{b^2+c^2}.$$

For this value (5) becomes

$$\left(2\frac{b^2+c^2-a^2}{b^2+c^2}+2\right)^2R^2 \geq a^2+2(b^2+c^2-a^2),$$

as $2b^2+2c^2-a^2 > (b-c)^2 > 0$, follows (1).

We now give the necessary and sufficient conditions for parameters in Schoenberg's inequality for holding equality, what led to the given thorem. Let

$$\lambda_1+\lambda_2+\lambda_3 = \lambda. \tag{6}$$

The functional f has a maximum, with the condition $\lambda_2+\lambda_3 = \text{const.}$, similarly $\lambda_3+\lambda_1 = \text{const.}$ and $\lambda_1+\lambda_2 = \text{const.}$, if

$$(\lambda_2-\lambda_3)a^2+(b^2-c^2)\lambda_1 = 0, \qquad \text{cycl.} \tag{7}$$

The system of linear equations (6-7) has solution

$$\mu_1 = ka^2(b^2+c^2-a^2), \qquad \text{cycl.,}$$

where

$$k = \lambda\left(2b^2c^2 + 2c^2a^2 + 2a^2b^2 - a^4 - b^4 - c^4\right)^{-1}.$$

Using formulas for area F of a triangle, Heron's and $4FR = abc$, we get equality

$$f(\mu_1, \mu_2, \mu_3) = k^2a^2b^2c^2\left(2b^2c^2 + 2c^2a^2 + 2a^2b^2 - a^4 - b^4 - c^4\right) = \lambda^2R^2.$$

By special values of the λ's several inequalities for the triangle, including the well-known formulas of Weitzenböck, Finsler and Hadwiger, can be deduced [4], [2].

Remark. *Equality in (5) holds if and only if* $\sin 2\alpha = r\lambda_1$, $\sin 2\beta = r\lambda_2$, $\sin 2\gamma = r\lambda_3$, $r \in R$, [1], *also*

$$\mu_1 = ka^2 2bc\cos\alpha = \lambda\frac{R^2}{2F}\sin 2\alpha, \qquad \textit{cycl.}$$

Theorem 2.

$$\text{(8)}\qquad \frac{a}{\sqrt{2b^2+2c^2-a^2}} + \frac{b}{\sqrt{2c^2+2a^2-b^2}} + \frac{c}{\sqrt{2a^2+2b^2-c^2}} \geq \sqrt{3},$$

equality holds if and only if the triangle is equilateral.

Proof. The inequality of J. Karamata [3]

$$\tan\frac{\alpha}{2} + \tan\frac{\beta}{2} + \tan\frac{\gamma}{2} \geq \sqrt{3}$$

and the third formula in (4) for either an acute or a rectangular triangle induce given inequality. Let $A = \sqrt{2b^2+2c^2-a^2}$, cycl. and f — the left-hand side of (8). Then

$$f'_a = (b^2+c^2)A^{-3} - abB^{-3} - caC^{-3} = 0, \qquad \text{cycl.}$$

implies

$$a : b : c = A^{-3} : B^{-3} : C^{-3}.$$

Therefore,

$$a = b \qquad \text{or} \qquad 2(a^2+b^2+c^2) = 3\frac{a^{8/3}-b^{8/3}}{a^{2/3}-b^{2/3}}, \qquad \text{cycl.}$$

and either $a = b = c$ or e. g. $a = b$, $c = \left(\sqrt[3]{2}-1\right)^{3/2}a$. Also $f > 2$ if $a = b + c$. □

References

[1] O. BOTTEMA, *An inequality for the triangle*, Simon Stevin **33** (1959), 97–100.
[2] O. BOTTEMA ET AL., *Geometric inequalities*, Wolters—Noordhoff Publishing, Groningen, 1969.
[3] J. KARAMATA, *Problem 119*, Glasnik matematičko—fizički i astronomski **3** (1948), 223.
[4] O. KOOI, *Inequalities for the triangle*, Simon Stevin **32** (1958), 97–101.
[5] D.S. MITRINOVIĆ ET AL., *Recent Advances in Geometric Inequalities*, Dordrecht, Boston–London, 1989.

"M. PUPIN", UNIVERSITY OF NOVI SAD, ZRENJANIN 23000, YUGOSLAVIA

FILOMAT (Niš) 9:2 (1995), 121–129

Filomat '94, Niš, October 22–24, 1994
Geometry. Computer Sciences

THE COMPLETE LIST OF $F(2)$ TYPE STRUCTURES IN THE COMPLEX FINSLER SPACE

Irena Čomić

ABSTRACT. *The complex Finsler space E' is formed in such a way, that its tangent space $T(E')$ is equal to $T(F_1) \oplus iT(F_2)$, where F_1 and F_2 are two 2n-dimensional Finsler spaces. Using the nonlinear connections N and $\overline{N}$ of F_1 and F_2 respectively, the adapted basis B' of $T(E')$ is formed. There is given the complete list of $F(2)$ type structures. Some of them for different values of parameters are almost complex, almost product or tangent structures.*

1. Complex Finsler spaces

Let us consider two n-dimensional Finsler spaces $F_1(x, \dot{x})$ and $F_2(y, \dot{y})$. The allowable coordinate transformations in F_1 and F_2 are given by

$$(1.1)\qquad \begin{array}{ll} x^{a'} = x^{a'}(x) & y^{i'} = y^{i'}(y) \\ \dot{x}^{a'} = A^{a'}_a(x)\,\dot{x}^a & \dot{y}^{i'} = B^{i'}_i(y)\dot{y}^i \\ A^{a'}_a = \frac{\partial x^{a'}}{\partial x^a} & B^{i'}_i = \frac{\partial y^{i'}}{\partial y^i}, \end{array}$$

where

$$rank[A^{a'}_a] = n, \quad rank[B^{i'}_i] = n,$$

so the inverse transformations exist.

The adapted basis of $T(F_1)$ is $B_1 = \{\frac{\delta}{\delta x^a}, \frac{\partial}{\partial \dot{x}^a}\}$ and the adapted basis of $T(F_2)$ is $B_2 = \{\frac{\delta}{\delta y^i}, \frac{\partial}{\partial \dot{y}^i}\}$, where

$$\frac{\delta}{\delta x^a} = \frac{\partial}{\partial x^a} - N^b_a(x, \dot{x})\frac{\partial}{\partial \dot{x}^b}, \quad \frac{\delta}{\delta y^i} = \frac{\partial}{\partial y^i} - \overline{N}^j_i(y, \dot{y})\frac{\partial}{\partial \dot{y}^j}.$$

$N^b_a(x, \dot{x})$ and $\overline{N}^j_i(y, \dot{y})$ are coefficients of the non-linear connections, which satisfy the usual transformation law with respect to (1).

The complex Finsler space $E'(x, \dot{x}, y, \dot{y})$ is formed in such a way that B', the adapted basis of $T(E')$, is given by $B' = B_1 \cup iB_2$.

1991 *Mathematics Subject Classification.* 53B40, 53C56, 53C60.

For the further exploration we shall use five kinds of indices

$a, b, c, d, e, f, g = 1, 2, \dots, n,$ $i, j, h, k, l, m, p, q = n+1, \dots, 2n$
$A, B, C, D, E, F, G = 2n+1, .., 3n,$ $I, J, H, K, L, M, P, Q = 3n+1, .., 4n$
$\alpha, \beta, \gamma, \delta, \kappa, \nu, \mu = 1, 2, \dots, 4n.$

The following equalities are valid

$$a = i = A = I (mod\, n), \qquad b = j = B = J (mod\, n) \tag{1.2}$$
$$c = h = C = H (mod\, n).$$

Using these indices, B' and its dual B'^* can be written in the form

(1.3)
$$\text{(a)} \quad B' = \{\partial_\alpha\} = \{\frac{\delta}{\delta x^a}, i\frac{\delta}{\delta y^i}, \frac{\partial}{\partial \dot{x}^A}, i\frac{\partial}{\partial \dot{y}^I}\}$$
$$\text{(b)} \quad B'^* = \{d^\alpha\} = \{dx^b, -idy^j, \delta\dot{x}^B, -i\delta\dot{y}^J\},$$

where

$$\delta\dot{x}^B = d\dot{x}^B + N^B_c(x, \dot{x})dx^c, \quad \delta\dot{y}^J = d\dot{y}^J + N^J_i(y, \dot{y})dy^i.$$

If we introduce the notations

(1.4)
$$\text{(a)} \quad R = [\frac{\delta}{\delta x^a} \ \ i\frac{\delta}{\delta y^i} \ \ \frac{\partial}{\partial \dot{x}^A} \ \ i\frac{\partial}{\partial \dot{y}^I}]$$

$$\text{(b)} \quad D = \begin{bmatrix} A^a_{a'}(x') & 0 & 0 & 0 \\ 0 & B^i_{i'}(y') & 0 & 0 \\ 0 & 0 & A^A_{A'}(x') & 0 \\ 0 & 0 & 0 & B^I_{I'}(y') \end{bmatrix}$$

$$\text{(c)} \quad K' = \begin{bmatrix} dx^{a'} \\ -idy^{i'} \\ \delta\dot{x}^{A'} \\ -i\delta\dot{y}^{I'} \end{bmatrix},$$

then the following relations are valid.

$$R' = RD \qquad K = DK'. \tag{1.5}$$

R' is obtained from R if indices a, i, A and I are substituted by a', i', A' and I' respectively, similarly K is obtained from K' if in K' the sign "′" over all indices is dropped. D is regular matrix, so exists D^{-1}. From (4a) we have

$$R = R'D^{-1} \qquad K' = D^{-1}K, \tag{1.6}$$

where

$$D^{-1}=\begin{bmatrix} A_b^{b'}(x) & 0 & 0 & 0 \\ 0 & B_j^{j'}(y) & 0 & 0 \\ 0 & 0 & A_B^{B'}(x) & 0 \\ 0 & 0 & 0 & B_J^{J'}(y) \end{bmatrix}.$$

2. The $F(2)$ type structures defined on E'

Definition 2.1. *The tensor field F of type (1) defined on E' is the structure of $F(k)$ type if in the basis B' its matrix can be decomposed on 4×4 blocks of format $n\times n$, such that in each row and each column are k scalar matrices and $4-k$ zero blocks.*

Notation. Every one of the scalar fields a, b, c, d, e, f, g, h denotes the corresponding real or complex scalar matrix of type $n\times n$ (for example $a=a(x,\dot{x},y,\dot{y})I$).

Theorem 2.1. *There exist 90 $F(2)$ type structures on E'. They are:*

$$\begin{bmatrix} a & 0 & e & 0 \\ b & 0 & f & 0 \\ 0 & c & 0 & g \\ 0 & d & 0 & h \end{bmatrix}\begin{bmatrix} a & 0 & e & 0 \\ b & 0 & 0 & g \\ 0 & c & f & 0 \\ 0 & d & 0 & h \end{bmatrix}\begin{bmatrix} a & 0 & e & 0 \\ b & 0 & 0 & g \\ 0 & c & 0 & h \\ 0 & d & f & 0 \end{bmatrix}\begin{bmatrix} a & 0 & 0 & g \\ b & 0 & e & 0 \\ 0 & c & f & 0 \\ 0 & d & 0 & h \end{bmatrix}$$

$$\begin{bmatrix} a & 0 & 0 & g \\ b & 0 & e & 0 \\ 0 & c & 0 & h \\ 0 & d & f & 0 \end{bmatrix}\begin{bmatrix} a & 0 & 0 & g \\ b & 0 & 0 & h \\ 0 & c & e & 0 \\ 0 & d & f & 0 \end{bmatrix}\begin{bmatrix} a & 0 & e & 0 \\ 0 & c & f & 0 \\ b & 0 & 0 & g \\ 0 & d & 0 & h \end{bmatrix}\begin{bmatrix} a & 0 & e & 0 \\ 0 & c & 0 & g \\ b & 0 & f & 0 \\ 0 & d & 0 & h \end{bmatrix}^{*}_{(1)}$$

$$\begin{bmatrix} a & 0 & e & 0 \\ 0 & c & 0 & g \\ b & 0 & 0 & h \\ 0 & d & f & 0 \end{bmatrix}\begin{bmatrix} a & 0 & 0 & g \\ 0 & c & e & 0 \\ b & 0 & f & 0 \\ 0 & d & 0 & h \end{bmatrix}\begin{bmatrix} a & 0 & 0 & g \\ 0 & c & e & 0 \\ b & 0 & 0 & h \\ 0 & d & f & 0 \end{bmatrix}\begin{bmatrix} a & 0 & 0 & g \\ 0 & c & 0 & h \\ b & 0 & e & 0 \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} a & 0 & e & 0 \\ 0 & c & f & 0 \\ 0 & d & 0 & g \\ b & 0 & 0 & h \end{bmatrix}\begin{bmatrix} a & 0 & e & 0 \\ 0 & c & 0 & g \\ 0 & d & f & 0 \\ b & 0 & 0 & h \end{bmatrix}\begin{bmatrix} a & 0 & e & 0 \\ 0 & c & 0 & g \\ 0 & d & 0 & h \\ b & 0 & f & 0 \end{bmatrix}\begin{bmatrix} a & 0 & 0 & g \\ 0 & c & e & 0 \\ 0 & d & f & 0 \\ b & 0 & 0 & h \end{bmatrix}^{*}_{(2)}$$

$$\begin{bmatrix} a & 0 & 0 & g \\ 0 & c & e & 0 \\ 0 & d & 0 & h \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} a & 0 & 0 & g \\ 0 & c & 0 & h \\ 0 & d & e & 0 \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & c & e & 0 \\ a & 0 & f & 0 \\ b & 0 & 0 & g \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} 0 & c & e & 0 \\ a & 0 & 0 & g \\ b & 0 & f & 0 \\ 0 & d & 0 & h \end{bmatrix}$$

$$\begin{bmatrix} 0 & c & e & 0 \\ a & 0 & 0 & g. \\ b & 0 & 0 & h \\ 0 & d & f & 0 \end{bmatrix}^{*}_{(3)} \begin{bmatrix} 0 & c & 0 & g \\ a & 0 & e & 0 \\ b & 0 & f & 0 \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} 0 & c & 0 & g \\ a & 0 & e & 0 \\ b & 0 & 0 & h \\ 0 & d & f & 0 \end{bmatrix} \begin{bmatrix} 0 & c & 0 & g \\ a & 0 & 0 & h \\ b & 0 & e & 0 \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & c & e & 0 \\ a & 0 & f & 0 \\ 0 & d & 0 & g \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & c & e & 0 \\ a & 0 & 0 & g \\ 0 & d & f & 0 \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & c & e & 0 \\ a & 0 & 0 & g \\ 0 & d & 0 & h \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & c & 0 & g \\ a & 0 & e & 0 \\ 0 & d & f & 0 \\ b & 0 & 0 & h \end{bmatrix}$$

$$\begin{bmatrix} 0 & c & 0 & g \\ a & 0 & e & 0 \\ 0 & d & 0 & h \\ b & 0 & f & 0 \end{bmatrix}^{*}_{(4)} \begin{bmatrix} 0 & c & 0 & g \\ a & 0 & 0 & h \\ 0 & d & e & 0 \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & c & e & 0 \\ 0 & d & f & 0 \\ a & 0 & 0 & g \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & c & e & 0 \\ 0 & d & 0 & g \\ a & 0 & f & 0 \\ b & 0 & 0 & h \end{bmatrix}$$

$$\begin{bmatrix} 0 & c & e & 0 \\ 0 & d & 0 & g \\ a & 0 & 0 & h \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & c & 0 & g \\ 0 & d & e & 0 \\ a & 0 & f & 0 \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & c & 0 & g \\ 0 & d & e & 0 \\ a & 0 & 0 & h \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & c & 0 & g \\ 0 & d & 0 & h \\ a & 0 & e & 0 \\ b & 0 & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} a & c & 0 & 0 \\ b & 0 & e & 0 \\ 0 & d & 0 & g \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ b & 0 & 0 & g \\ 0 & d & e & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ b & 0 & e & 0 \\ 0 & 0 & f & g \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ b & 0 & 0 & g \\ 0 & 0 & e & h \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} a & c & 0 & 0 \\ 0 & d & e & 0 \\ b & 0 & 0 & g \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & d & 0 & g \\ b & 0 & e & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & 0 & e & g \\ b & 0 & f & 0 \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & 0 & e & g \\ b & 0 & 0 & h \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} a & c & 0 & 0 \\ 0 & d & e & 0 \\ 0 & 0 & f & g \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & d & 0 & g \\ 0 & 0 & e & h \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & 0 & e & g \\ 0 & d & f & 0 \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & 0 & e & g \\ 0 & d & 0 & h \\ b & 0 & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} b & 0 & e & 0 \\ a & c & 0 & 0 \\ 0 & d & 0 & g \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} b & 0 & 0 & g \\ a & c & 0 & 0 \\ 0 & d & e & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} b & 0 & e & 0 \\ a & c & 0 & 0 \\ 0 & 0 & f & g \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} b & 0 & 0 & g \\ a & c & 0 & 0 \\ 0 & 0 & e & h \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & d & e & 0 \\ a & c & 0 & 0 \\ b & 0 & 0 & g \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} 0 & d & 0 & g \\ a & c & 0 & 0 \\ b & 0 & e & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ a & c & 0 & 0 \\ b & 0 & f & 0 \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ a & c & 0 & 0 \\ b & 0 & 0 & h \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & d & e & 0 \\ a & c & 0 & 0 \\ 0 & 0 & f & g \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & d & 0 & g \\ a & c & 0 & 0 \\ 0 & 0 & e & h \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ a & c & 0 & 0 \\ 0 & d & f & 0 \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ a & c & 0 & 0 \\ 0 & d & 0 & h \\ b & 0 & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} b & 0 & e & 0 \\ 0 & d & 0 & g \\ a & c & 0 & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} b & 0 & 0 & g \\ 0 & d & e & 0 \\ a & c & 0 & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} b & 0 & e & 0 \\ 0 & 0 & f & g \\ a & c & 0 & 0 \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} b & 0 & 0 & g \\ 0 & 0 & e & h \\ a & c & 0 & 0 \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & d & e & 0 \\ b & 0 & 0 & g \\ a & c & 0 & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} 0 & d & 0 & g \\ b & 0 & e & 0 \\ a & c & 0 & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ b & 0 & f & 0 \\ a & c & 0 & 0 \\ 0 & d & 0 & h \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ b & 0 & 0 & h \\ a & c & 0 & 0 \\ 0 & d & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & d & e & 0 \\ 0 & 0 & f & g \\ a & c & 0 & 0 \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & d & 0 & g \\ 0 & 0 & e & h \\ a & c & 0 & 0 \\ b & 0 & f & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ 0 & d & f & 0 \\ a & c & 0 & 0 \\ b & 0 & 0 & h \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ 0 & d & 0 & h \\ a & c & 0 & 0 \\ b & 0 & f & 0 \end{bmatrix}$$

$$\begin{bmatrix} b & 0 & e & 0 \\ 0 & d & 0 & g \\ 0 & 0 & f & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} b & 0 & 0 & g \\ 0 & d & e & 0 \\ 0 & 0 & f & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} b & 0 & e & 0 \\ 0 & 0 & f & g \\ 0 & d & 0 & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} b & 0 & 0 & g \\ 0 & 0 & e & h \\ 0 & d & f & 0 \\ a & c & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & d & e & 0 \\ b & 0 & 0 & g \\ 0 & 0 & f & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & d & 0 & g \\ b & 0 & e & 0 \\ 0 & 0 & f & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ b & 0 & f & 0 \\ 0 & d & 0 & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ b & 0 & 0 & h \\ 0 & d & f & 0 \\ a & c & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & d & e & 0 \\ 0 & 0 & f & g \\ b & 0 & 0 & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & d & 0 & g \\ 0 & 0 & e & h \\ b & 0 & f & 0 \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ 0 & d & f & 0 \\ b & 0 & 0 & h \\ a & c & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ 0 & d & 0 & h \\ b & 0 & f & 0 \\ a & c & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} a & c & 0 & 0 \\ b & d & 0 & 0 \\ 0 & 0 & e & g \\ 0 & 0 & f & h \end{bmatrix}^{*}_{(5)} \begin{bmatrix} a & c & 0 & 0 \\ 0 & 0 & e & g \\ b & d & 0 & 0 \\ 0 & 0 & f & h \end{bmatrix} \begin{bmatrix} a & c & 0 & 0 \\ 0 & 0 & e & g \\ 0 & 0 & f & h \\ b & d & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ a & c & 0 & 0 \\ b & d & 0 & 0 \\ 0 & 0 & f & h \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & e & g \\ a & c & 0 & 0 \\ 0 & 0 & f & h \\ b & d & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & e & g \\ 0 & 0 & f & h \\ a & c & 0 & 0 \\ b & d & 0 & 0 \end{bmatrix}^{*}_{(6)} .$$

The first 36 matrices are formed in such a way that in the first two columns the chosen elements are always in different rows; in the next 48 matrices the first two columns have once two elements in the same row (ac) and two elements in different rows; in the last 6 matrices the first and second columns have two times, two elements in the same row.

Definition 2.2. *The tensor field F of type (1,1) defined on E' is almost complex structure (a.c.s.) iff $F^2 = -I$, almost product structure (a.p.s.) iff $F^2 = I$, or tangent structure (t.s.) iff $F^2 = 0$.*

Theorem 2.2. *The $F(2)$ type structure, which in the former list do not have the sign "*" can not be a.c.s., or o.p.s., or t.s.*

Proof. Some $F_i (i = 1, \dots, 90)$ from the above list of $F(2)$ type structures can be a.c.s., or a.p.s., or t.s. if F_i^2 has the property, that all elements, which are not on the main diagonal are equal to zero. All F_i's, which do not have the sign "*" (there are 84) are such, that F_i^2 has at least on one place, which is not on the main diagonal, product of two elements. This product is zero if at least one of the factor is equal to zero, but in this case F_i is not $F(2)$ type structure.

Theorem 2.3. *There are only six $F(2)$ type structures defined on E', which for some special values of parameters can be a.c.s., or a.p.s., or t.s. They are denoted by "*" in the above list of $F(2)$ type structures.*

Proof. For special values of parameters we have

$$F_1 = \begin{bmatrix} a & 0 & b & 0 \\ 0 & c & 0 & d \\ e & 0 & -a & 0 \\ 0 & g & 0 & -c \end{bmatrix} \quad F_2 = \begin{bmatrix} a & 0 & 0 & b \\ 0 & c & d & 0 \\ 0 & e & -c & 0 \\ f & 0 & 0 & -a \end{bmatrix}$$

$$F_3 = \begin{bmatrix} 0 & a & b & 0 \\ -ce & 0 & 0 & cd \\ cd & 0 & 0 & -ca \\ 0 & d & e & 0 \end{bmatrix} \quad F_4 = \begin{bmatrix} 0 & a & 0 & -b \\ ce & 0 & bc & 0 \\ 0 & d & 0 & e \\ -cd & 0 & ac & 0 \end{bmatrix}$$

$$F_5 = \begin{bmatrix} a & b & 0 & 0 \\ c & -a & 0 & 0 \\ 0 & 0 & e & f \\ 0 & 0 & d & -e \end{bmatrix} \quad F_6 = \begin{bmatrix} 0 & 0 & ae & -ab \\ 0 & 0 & -ac & d \\ a^{-1}b & b & 0 & 0 \\ c & e & 0 & 0 \end{bmatrix}.$$

By direct calculation we obtain

$$\begin{aligned} F_1^2 &= diag[a^2+be, c^2+dg, a^2+be, c^2+dg] \\ F_2^2 &= diag[a^2+bf, c^2+de, c^2+de, a^2+bf] \\ F_3^2 &= c(bd-ae)I \\ F_4^2 &= c(bd+ae)I \\ F_5^2 &= diag[a^2+bc, a^2+bc, e^2+df, e^2+df] \\ F_6^2 &= (de-abc)I. \end{aligned}$$

From Theorem 2.3 follows

Theorem 2.4. *The $F(2)$ type structures $F_1 - F_6$ are a.c.s. if*

$$\begin{aligned} &in \quad F_1 \quad a^2+be = c^2+dg = -1, \\ &in \quad F_2 \quad a^2+bf = c^2+de = -1, \\ &in \quad F_3 \quad c(bd-ae) = -1, \\ &in \quad F_4 \quad c(bd+ae) = -1, \\ &in \quad F_5 \quad a^2+bc = e^2+df = -1, \\ &in \quad F_6 \quad de-abc = -1. \end{aligned}$$

If in the above equations -1 is everywhere replaced by 1, the structures $F_1 - F_6$ become a.p.s.; if -1 is everywhere replaced by 0, the structures $F_1 - F_6$ become t.s.

3. The tensor character of $F(2)$ type structures

Theorem 3.1. *All 90 $F(2)$ type structures from the list in paragraph 2 determine tensor fields of type $(1,1)$ in the basis B', with respect to the coordinate transformation (1).*

Proof. As the proof is the same for all structures, we shall give it for F_1. The structure F_1 in the basis B' determines the following transformation:

$$\begin{aligned}
F_1(\tfrac{\delta}{\delta x^a}) &= a\tfrac{\delta}{\delta x^a} && +e\tfrac{\partial}{\partial \dot{x}^A} \\
F_1(i\tfrac{\delta}{\delta y^i}) &= c(i\tfrac{\delta}{\delta y^i}) && +g(i\tfrac{\partial}{\partial \dot{y}^I}) \\
F_1(\tfrac{\partial}{\partial \dot{x}^A}) &= b\tfrac{\delta}{\delta x^a} && -a\tfrac{\partial}{\partial \dot{x}^A} \\
F_1(i\tfrac{\partial}{\partial \dot{y}^I}) &= d(i\tfrac{\delta}{\delta y^i}) && -c(i\tfrac{\partial}{\partial \dot{y}^I}).
\end{aligned}$$

The precise form of F_1 is the matrix

$$F_1 = \begin{bmatrix} a\delta^b_a & 0 & b\delta^B_a & 0 \\ 0 & c\delta^j_i & 0 & d\delta^J_i \\ e\delta^b_A & 0 & -a\delta^B_A & 0 \\ 0 & g\delta^j_I & 0 & -c\delta^J_I \end{bmatrix}.$$

The tensor F_1, which is determined by the matrix F_1 can be written in the following way:

$$F_1 = RF_1 \otimes K.$$

In the basis R' and K' F_1 has the form (see (4)):

$$F_1 = R'D^{-1}F_1D \otimes K' = R'F'_1 \otimes K',$$

where

$$F'_1 = D^{-1}F_1D = \begin{bmatrix} a\delta^b_a A^a_{a'}A^{b'}_b & 0 & b\delta^B_a A^a_{a'}A^{B'}_B & 0 \\ 0 & c\delta^j_i B^i_{i'}B^{j'}_j & 0 & d\delta^J_i B^i_{i'}B^{J'}_J \\ e\delta^b_A A^{b'}_b A^A_{A'} & 0 & -a\delta^B_A A^A_{A'}A^{B'}_B & 0 \\ 0 & g\delta^j_I B^I_{I'}B^{j'}_j & 0 & -c\delta^J_I B^I_{I'}B^{J'}_J \end{bmatrix}.$$

For F'_1 we have

$$F'^2_1 = (D^{-1}F_1D)(D^{-1}F_1D) = D^{-1}F^2_1D.$$

From the above relation follows:

$$\begin{aligned}
&\text{if } F^2_1 = -I \Rightarrow F'^2_1 = -I, \\
&\text{if } F^2_1 = I \Rightarrow F'^2_1 = I, \\
&\text{if } F^2_1 = 0 \Rightarrow F'^2_1 = 0.
\end{aligned}$$

Refrences

[1] Bejancu, A., *Geometry of CR-Submanifolds*, D Reider Publishing Company, 1986.

[2] Čomić, I., *Generalized Connection in the Complex Finsler Space*, (to be published).

[3] Čomić I., Nikić J., *Some Hermite metrics in the complex Finsler spaces*, Publ. Inst. Math. Beograd **55 (69)** (1994), 89–97.

[4] Ichiyio, Y., *Almost complex structures of tangent bundles and Finsler metrics*. J. Math. Kyoto Univ. 6-3 (1967) 419–452.

[5] Kobayashi, Sh., Nomizu, K., *Foundations of Differential Geometry*, Interscience Publishers, New York, London 1963.

[6] Prakash, N., *Kaehlerian Finsler Manifolds*, The Math. Student. Vol. 30, No. 1,2, (1962), 1–11.

[7] Rizza, G.B., *Structure di Finsler di tipo quasi hermitiano*, Riv. Mat. Univ. Parma, 4 (1963) 83–106.

[8] Shimada, H., *Remarks on the almost complex structures of tangent bundles*, Research Report Kushiro Tech. Coll. No. 21, (1987) 169–176.

[9] Yano, K., *Differential Geometry on Complex and Almost Complex Spaces*, A Pergamon Press Book, New York, 1965.

Faculty of Technical Sciences, 21000 Novi Sad, Yugoslavia

FILOMAT (Niš) 9:2 (1995), 131–142
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

GEODESIC TUBES AND JACOBI VECTOR FIELDS ON COMPLEX SPACE FORMS

Mirjana Djorić

ABSTRACT. *Studying geodesic variations and associated Jacobi vector fields is very useful for examining the theory of curvature in local and global Riemannian geometry. This is directly connected with the investigation of the geometry of small geodesic spheres and tubes, so it can be used in the analysis of the curvature of the ambient space. In this paper, the explicit expressions for the Jacobi vector fields on complex space forms will be used for calculating the matrix of the shape operator of tubes about geodesics on complex space forms.*

1. Introduction

The study of the curvature of a Riemannian manifold is one of the most interesting topics in Riemannian geometry. As it is well-known, the study of variations of geodesics and the associated Jacobi vector fields is very useful in treating curvature theory in local and global Riemannian geometry. This is directly related to the investigation of the geometry of small geodesic spheres and tubes about curves and submanifolds. The properties of the extrinsic and intrinsic geometry of these geometric objects may be used to study the curvature of the ambient space, as it was done in [1]-[9]. On this occasion we consider only the converse situation, namely, it is quite clear and well-known that when the Riemannian manifold is of a special type (for example, if it has special curvature), then the properties of geometric objects on it are strongly influenced. In [4] the author gave the explicit expressions for the shape operator of tubes about φ-geodesics on Sasakian space forms, while in this paper the special case when the ambient space is a complex

1991 *Mathematics Subject Classification*: 53B20, 53B35, 53C22, 53C55
Key words: Kähler manifold, complex space form, geodesic tube, shape operator, Jacobi vector field.
Supported by Grant 0401A of RFNS through Math. Inst. SANU

space form is considered. Working with Jacobi vector fields, since this falls among the best ways of analyzing the geometry of tubular neighborhoods, the matrix of the shape operator of tubes about geodesics on complex space form is obtained.

We refer to [11] and [14] for a study of tubular neighborhoods and [2] where a more detailed and more complete developement may be found, with an extensive list of references. The article is organized in the following way: Section 2 is devoted to a brief survey of the concepts used throughout the paper and in Section 3 the main results are treated.

2. Preliminaries

Let M be a complex analytic manifold of complex dimension m. By means of charts we may transfer the complex structure of complex m-dimensional Euclidean space C^m to M to obtain an almost complex structure J on M, i.e., a tensor field J on M of type $(1,1)$ such that $J^2 = -I$, where I is the tensor field which is the identity transformation on each tangent space of M. A Riemannian metric g on M is a Hermitian metric if $g(JX, JY) = g(X,Y)$ for any vector fields X and Y on M; M is then called a *Hermitian manifold*. If moreover the almost complex structure J is parallel with respect to the Riemannian connection of g, then J (resp. g) is called a *Kähler structure* (resp. *Kähler metric*); M is then called a *Kähler manifold.* We call a plane which is tangent to M and is invariant by J a *holomorphic plane.* If M is a Kähler manifold, the sectional curvature of a plane p tangent to M will be denoted by $K(p)$ and the sectional curvature of the holomorphic plane generated by a unit tangent vector X will be denoted by $K(X)$. M is said to be of constant holomorphic sectional curvature c if the sectional curvature of every holomorphic tangent plane is equal to c. As a *complex space form* we shall understand a complete Kähler manifold of constant holomorphic sectional curvature and its curvature tensor $R_{XY}Z = \nabla_{[X,Y]}Z - [\nabla_X, \nabla_Y]Z$ is completely determined and given by ([15]):

$$\begin{aligned}R_{XY}Z = \frac{c}{4}\Big(&g(X,Z)Y - g(Y,Z)X + g(JX,Z)JY - g(JY,Z)JX \\ &+ 2g(JX,Y)JZ\Big).\end{aligned} \tag{1}$$

As is well known, any simply-connected complex space form M is (after multiplying the metric of M by a suitable positive constant) holomorphically isometric to a complex projective space, a complex Euclidean space or a complex hyperbolic space, in dependence of M being of positive, zero or negative holomorphic sectional curvature, respectively ([15]).

We finish these preliminaries by repeating some general facts about tubes. We refer to [2], [11] and [14] for more details and references.

Therefore, let $\sigma : [a,b] \to M$ be a smooth, embedded unit speed curve in a Riemannian manifold M of dimension n and denote by $\sigma^\perp$ the normal bundle of σ and by $\exp_\sigma$ the exponential map of this normal bundle, i.e.,

$$\exp_\sigma(\sigma(t), v) = \exp_{\sigma(t)} v$$

for any $t \in [a,b]$ and all $v \in \sigma(t)^\perp$. Here $\sigma(t)^\perp$ denotes the fiber of $\sigma^\perp$ over $\sigma(t)$. Further, let $\mathcal{U}_\sigma(r)$ be the (open) tubular neighborhood or the (open) solid tube of radius r about σ, i.e., the set defined by

$$\mathcal{U}_\sigma(r) = \{\exp_{\sigma(t)} v \mid v \in \sigma(t)^\perp, \| v \| < r, t \in [a,b]\}$$

and denote by $N_\sigma(r)$ the (open) solid tube of radius r about the zero section of the normal bundle $\sigma^\perp$ of σ. In further text, we shall always assume that the radius r of the tubular neighborhood is smaller than the distance from σ to its nearest focal point. In this case, the exponential map $\exp_\sigma$ is a diffeomorphism between $\mathcal{U}_\sigma(r)$ and $N_\sigma(r)$ and consequently, the set

$$\mathcal{P}_\sigma(s) = \{p \in \mathcal{U}_\sigma(r) \mid d(\sigma, p) = s\},$$

for some $s < r$, is a (smooth) hypersurface in M, called the tube of radius s about σ. If σ is a geodesic on M, the tubes $\mathcal{P}_\sigma$ are called *geodesic tubes* about σ.

For the purpose of describing the geometry of a Riemannian manifold M in the neighborhood of a curve σ we use Fermi coordinates. The *Fermi coordinate system* $(x_1, \dots, x_n)$ with respect to $\sigma(a)$ and relative to a given orthonormal frame field $\{F_1, \dots, F_n\}$ along the curve σ for which $\dot\sigma(t) = (F_1)_{\sigma(t)}$ is defined by

$$\begin{aligned} x_1\left(\exp_{\sigma(t)}\left(\sum_{j=2}^n t_j F_j\right)\right) &= t - a\,, \\ x_i\left(\exp_{\sigma(t)}\left(\sum_{j=2}^n t_j F_j\right)\right) &= t_i\,, \quad i = 2, \dots, n, \end{aligned} \tag{2}$$

provided that the numbers $t_2, \dots, t_n$ are small enough in order to have a diffeomorphic $\exp_\sigma$.

Further, if γ is a unit speed geodesic of M normal to σ with $\gamma(0) = m = \sigma(t)$ and $v = \gamma'(0)$, then there is a system of Fermi coordinates $(x_1, \dots, x_n)$ such that for small s we have

$$\left(\frac{\partial}{\partial x_1}\right)_m = \dot\sigma(t), \quad \left(\frac{\partial}{\partial x_i}\right)_m \in \{\dot\sigma(t)\}^\perp, \, i = 2, \dots, n-1,$$

$$\left(\frac{\partial}{\partial x_n}\right)_{\gamma(s)} = \gamma'(s) .$$

Since $\exp_{\sigma(t)}$ is diffeomorphism on $\mathcal{U}_\sigma(r)$, the equations (2) define a coordinate system near m. It is known ([11]) that the restrictions of the coordinate vector fields $\left\{\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right\}$ to σ are orthonormal. In what follows we shall relate the coordinate frame field to a frame field obtained by considering a special set of Jacobi vector fields along γ with a view to obtaining the expression for the shape operator of $\mathcal{P}_\sigma(r)$.

In this aim, let $p = \exp_{\sigma(t)}(rv)$, $v \in \sigma(t)^\perp$, $\|v\| = 1$ be a point of $\mathcal{P}_\sigma(r)$ and let $\gamma : s \mapsto \exp_{\sigma(t)}(sv)$ be the (unique) unit speed geodesic connecting $\sigma(t)$ and p (and cutting σ orthogonally). Denote by $\{E_1, \ldots, E_n\}$ the frame field along γ obtained by parallel translation of $\{F_1(t), \ldots, F_n(t)\}$ with respect to the Levi Civita connection ∇. Next, if $R = R(s)$ denotes the endomorphism $u \mapsto R_{\gamma'(s)\,u}\gamma'(s)$ of the vector space $\{\gamma'(s)\}^\perp \subset T_{\gamma(s)}M$, then a vector field Y along a geodesic γ is called a *Jacobi vector field* if it satisfies the following second order differential equation- the *Jacobi equation*:

$$Y'' + RY = 0 , \tag{3}$$

where the prime $'$ denotes covariant differentiation along γ. Next, let $Y_i, i = 1, \ldots, n-1$ be the $n-1$ Jacobi vector fields along γ, satisfying the initial conditions

$$\begin{cases} Y_1(0) = F_1(t) , & Y_1'(0) = \left(\nabla_{\gamma'} \dfrac{\partial}{\partial x_1}\right)(\sigma(t)) , \\ Y_i(0) = 0 , & Y_i'(0) = F_i(t) , \qquad i = 2, \ldots, n-1 \end{cases} \tag{4}$$

and define

$$Y_i(s) = (B\, E_i)(s) , \quad i = 1, \ldots, n-1 . \tag{5}$$

The vector fields $Y_i(s)$ determine a basis for the space $\{\gamma'(s)\}^\perp$ for sufficiently small s and $s \mapsto B(s)$ is an endomorphism-valued function. Then, each $B(s)$ is an endomorphism of the space $\{\gamma'(s)\}^\perp$ and all these spaces may be identified via the parallel translation along γ by using the basis $\{E_i, i = 1, \ldots, n\}$. We shall do this at several places without mentioning it explicitly.

Now, from (3), (5) and the initial conditions (4) it follows that B satisfies the Jacobi equation

$$B'' + R \circ B = 0 \tag{6}$$

with the initial conditions

$$B(0)=\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}, \quad B'(0)=\begin{pmatrix}0 & 0\\ 0 & I\end{pmatrix}, \tag{7}$$

since we shall focus our attention only to tubes along geodesics.

Finally, we shall write down the matrix of the shape operator S^σ of geodesic tube $\mathcal{P}_\sigma(r)$, using Jacobi vector fields along geodesics orthogonal to σ. Since $\frac{\partial}{\partial s}(p)$ is a unit normal vector of $\mathcal{P}_\sigma(r)$ at $p=\exp_{\sigma(t)}(rv)$, the shape operator S^σ of $\mathcal{P}_\sigma(r)$ at p is defined by

$$(S^\sigma X)(p)=\left(\nabla_X \frac{\partial}{\partial s}\right)(p) \tag{8}$$

for any vector X tangent to $\mathcal{P}_\sigma(r)$ at p. Hence, it is easy to see, by using (5), that the shape operator $S^\sigma(p)$ takes the form

$$S^\sigma(p)=\left(B'B^{-1}\right)(r). \tag{9}$$

3. The main results

In this section we consider complex space forms and we compute the explicit expressions for the shape operator of geodesic tubes in these manifolds. To obtain our results we use here one of the most convenient methods for analyzing the geometry of small geodesic spheres and tubes about curves and submanifolds, by studying the Jacobi vector fields on complex space forms. It is quite natural that the Jacobi vector fields play an important role in this research since it is a well-known fact that the curvature of a Riemannian manifold is geometrically reflected by the behavior of one-parameter families of neighboring geodesics and they are analytically described by Jacobi vector fields. When the manifold is of a special type, the consideration of Jacobi vector fields results in the study of the Jacobi differential equation which has a relatively simple form.

Let m be a point on a complete Kähler manifold M^n of constant holomorphic sectional curvature c and let $\mathcal{P}_\sigma(r)$ be a tube of radius r about a geodesic σ tangent to a unit vector field u. Further, let γ be a geodesic through $m=\gamma(0)$, parametrized by arc length s, with initial velocity vector $\gamma'(0)=v$ and meeting σ orthogonally at $m=\sigma(t)$, with $u=\dot\sigma$ at m. Hereafter we shall also write $\gamma'(s)=v$ at any point of γ. For a vector field v the Jacobi equation

$$\nabla_v\nabla_v X+R_{vX}v=0 \tag{10}$$

for a given complex space form M becomes by virtue of (1)

$$\nabla_v \nabla_v X + \frac{c}{4}\Big(X - 3\,g(JX,v)Jv\Big) = 0\,. \tag{11}$$

Further, we shall distinguish three cases, depending on the position of the point $p = \exp_{\sigma(t)}(rv)$, $v \in \sigma(t)^{\perp}$, $\|v\| = 1$, in the forthcoming three theorems.

First, consider the special points p of the geodesic tubes $\mathcal{P}_\sigma(r)$ on a complex space form M^n, such that $p = \exp_{\sigma(t)}(rv)$, $v \in \sigma(t)^{\perp}$, $v(\sigma(t)) = Ju(\sigma(t))$. As this case has already been investigated in [2] and [6], we give here only the final expression for the matrix of the shape operator. Namely, the following theorem holds:

Theorem 1. ([6]) *Let (M,g,J) be a Kähler manifold of constant holomorphic sectional curvature c. Then, at a point $p = \exp_{\sigma(t)}(rv)$, $v \in \sigma(t)^{\perp}$, $v(\sigma(t)) = Ju(\sigma(t))$ of the tube $\mathcal{P}_\sigma(r)$ (along a geodesic $\sigma(t)$ tangent to a vector u), the shape operator $S^\sigma(p)$ can be represented by the following matrix:*

$$S^\sigma(p) = \begin{bmatrix} A(r) & 0 & \dots & 0 \\ 0 & B(r) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B(r) \end{bmatrix} \tag{12}$$

with respect to the basis $\{E_1, \dots, E_{n-1}\}$ defined in Section 2. The explicit expressions for the entries are as follows:

$A(r) = 0,$	$B(r) = \dfrac{1}{r},$	for $c = 0;$
$A(r) = -\sqrt{c}\tan\sqrt{c}r,$	$B(r) = \dfrac{\sqrt{c}}{2}\cot\dfrac{\sqrt{c}}{2}r,$	for $c > 0;$
$A(r) = \sqrt{\lvert c\rvert}\tanh\sqrt{\lvert c\rvert}r,$	$B(r) = \dfrac{\sqrt{\lvert c\rvert}}{2}\coth\dfrac{\sqrt{\lvert c\rvert}}{2}r,$	for $c < 0.$

Now, let us consider sufficiently small tube $\mathcal{P}_\sigma(r)$ about the geodesic σ embedded in a Kähler manifold of constant holomorphic sectional curvature c. Let γ denote the unit-speed geodesic meeting σ orthogonally at $m = \sigma(t)$ and tangent to a vector v such that $g(u(m), Jv(m)) = a$, where $\dot\sigma = u$ at m. To obtain the matrix of the shape operator of $\mathcal{P}_\sigma(r)$ at points $p =$

$\exp_{\sigma(t)}(rv)$, we first choose an orthonormal basis $\{e_1, \ldots, e_n\}$ for the tangent space $T_m M$, such that $e_1 = u(m), e_2 = (Jv(m) - au(m))/b, e_n = v(m)$, where $a^2 + b^2 = 1$. Further, let $\{E_1, \ldots, E_n\}$ be the basis obtained by parallel translation of the basis $\{e_1, \ldots, e_n\}$ along γ. Then it follows that any vector field X orthogonal to the geodesic γ can be written as

$$X = f_1\, E_1 + f_2 E_2 + \sum_{i=3}^{n-1} f_i\, E_i\,. \tag{13}$$

Since, using (1) we obtain

$$\begin{cases} R_{vE_1}v = \dfrac{c}{4}\Big((3a^2+1)E_1 + 3abE_2\Big) \\ R_{vE_2}v = \dfrac{c}{4}\Big(3abE_1 + (3b^2+1)E_2\Big) \\ R_{vE_i}v = \dfrac{c}{4}\, E_i\,, \quad i = 3, \ldots, n-1\,, \end{cases} \tag{14}$$

we see that (11) is equivalent to the following system of differential equations:

$$\begin{aligned} 4\, f_1'' + c\,(3a^2+1)f_1 + 3abc\, f_2 &= 0\,, \\ 4\, f_2'' + 3abc\, f_1 + c\,(3b^2+1)f_2 &= 0\,, \end{aligned} \tag{15}$$

$$4\, f_i'' + c\, f_i = 0\,, \quad i = 3, \ldots, n-1\,. \tag{16}$$

Now, consider the substitution

$$\begin{aligned} z_1 &= a\, f_1 + b\, f_2\,, \\ z_2 &= b\, f_1 - a\, f_2\,. \end{aligned} \tag{17}$$

Then the equations (15) take the form

$$\begin{aligned} a\, z_1'' + b\, z_2'' + c\, a\, z_1 + \frac{c}{4}\, b\, z_2 &= 0\,, \\ b\, z_1'' - a\, z_2'' + c\, b\, z_1 - \frac{c}{4}\, a\, z_2 &= 0\,. \end{aligned} \tag{18}$$

In this way, by multiplying the first equation in (18) by a and the second by b and adding the obtained results, we arrive at a differential equation

$$z_1'' + c\, z_1 = 0\,,$$

which is easy to integrate, having in mind that the solutions will depend on the sign of c. Finally, using the standard solutions of the $n-3$ equations (16), we can derive the complete solutions in the three cases we shall need.

Case 1: $\mathbf{c} = 0$

Here we find

$$\begin{aligned} f_1 &= (aA + bC)s + aB + bD\,, \\ f_2 &= (bA - aC)s + bB - aD\,, \\ f_i &= A_i s + B_i\,, \quad i = 3, \dots, n-1\,, \end{aligned}$$

with $A\,, B\,, C\,, D\,, A_i\,, B_i$ being constant along γ.

Case 2: $\mathbf{c} > 0$

In this case, putting $k = \sqrt{c}$, we obtain

$$\begin{aligned} f_1 &= aF\,\cos ks + aG\,\sin ks + bH\,\cos\tfrac{ks}{2} + bI\,\sin\tfrac{ks}{2}\,, \\ f_2 &= bF\,\cos ks + bG\,\sin ks - aH\,\cos\tfrac{ks}{2} - aI\,\sin\tfrac{ks}{2}\,, \\ f_i &= F_i\,\cos\tfrac{ks}{2} + G_i\,\sin\tfrac{ks}{2}\,, \quad i = 3, \dots, n-1\,, \end{aligned}$$

with $F\,, G\,, H\,, I\,, F_i\,, G_i$ being constant along γ.

Case 3: $\mathbf{c} < 0$

This time we put $k = \sqrt{-c}$. Repeating the same computations, we obtain

$$\begin{aligned} f_1 &= a(Ke^{ks} + Le^{-ks}) + b(Me^{\frac{ks}{2}} + Ne^{-\frac{ks}{2}})\,, \\ f_2 &= b(Ke^{ks} + Le^{-ks}) - a(Me^{\frac{ks}{2}} + Ne^{-\frac{ks}{2}})\,, \\ f_i &= K_i e^{\frac{ks}{2}} + L_i e^{-\frac{ks}{2}}\,, \quad i = 3, \dots, n-1\,, \end{aligned}$$

with $K\,, L\,, M\,, N\,, K_i\,, L_i$ being again constant along γ.

Moreover, we shall need the form of the Jacobi vector fields along a geodesic γ satisfying the following initial conditions:

$$X_1(0) = E_1(0)\,, \quad X_1'(0) = 0\,, \tag{19}$$

$$X_i(0) = 0\,, \qquad X_i'(0) = E_i(0)\,, \quad i = 3, \dots, n-1. \tag{20}$$

We shall therefore compute these special Jacobi fields in the three above-described cases, using the notation $k = \sqrt{c}$ if $c > 0$ and $k = \sqrt{-c}$ when $c < 0$.

Case 1: $\mathbf{c} = 0$

$$X_1(s) = E_1(s)\,, \quad X_2(s) = s\,E_2(s)\,, \quad X_i(s) = s\,E_i(s),\ i = 3,\ldots,n-1\,.$$

Case 2: $\mathbf{c} > 0$

$$X_1(s)= \left(a^2\cos ks + b^2\cos \tfrac{ks}{2}\right) E_1 + a\,b\left(\cos ks - \cos\tfrac{ks}{2}\right) E_2\,,$$
$$X_2(s)= \frac{a\,b}{k}\left(\sin ks - 2\sin\tfrac{ks}{2}\right) E_1 + \frac{1}{k}\left(b^2\sin ks + 2a^2\sin\tfrac{ks}{2}\right) E_2\,,$$
$$X_i(s)= \frac{2}{k}\sin\tfrac{ks}{2}\,E_i(s)\,,\ i = 3,\ldots,n-1.$$

Case 3: $\mathbf{c} < 0$

$$X_1(s)= \left(a^2\cosh ks + b^2\cosh \tfrac{ks}{2}\right) E_1 + a\,b\left(\cosh ks - \cosh\tfrac{ks}{2}\right) E_2\,,$$
$$X_2(s)= \frac{a\,b}{k}\left(\sinh ks - 2\sinh\tfrac{ks}{2}\right) E_1 + \frac{1}{k}\left(b^2\sinh ks + 2a^2\sinh\tfrac{ks}{2}\right) E_2\,,$$
$$X_i(s)= \frac{2}{k}\sinh\tfrac{ks}{2}\,E_i(s)\,,\ i = 3,\ldots,n-1.$$

Finally, using relations (4)-(9) and computed Jacobi vector fields, it follows that the shape operator S^σ can be represented by the following quasi-diagonal matrix:

$$S^\sigma(p) = \begin{bmatrix} A(r) & B(r) & 0 & \ldots & 0 \\ B(r) & C(r) & 0 & \ldots & 0 \\ 0 & 0 & D(r) & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & D(r) \end{bmatrix} \tag{21}$$

with respect to the basis $\{bE_1 - aE_2, Jv, E_3, \ldots, E_{n-1}\}$. The explicit expressions for the entries are as follows:

Case 1: $\mathbf{c} = 0$

$$A(r)= B(r) = 0\,,$$
$$C(r)= D(r) = \frac{1}{r}\,.$$

Case 2: $\mathbf{c} > 0$

$$A(r)= \tfrac{1}{2\,\omega}\left(-b^2\sin\tfrac{kr}{2}\sin kr + 2a^2\cos\tfrac{kr}{2}\cos kr\right)\,,$$
$$B(r)= -\tfrac{1}{\omega}\,a\,b\,,$$
$$C(r)= \tfrac{1}{\omega}\left(-2a^2\sin\tfrac{kr}{2}\sin kr + b^2\cos\tfrac{kr}{2}\cos kr\right)\,,$$
$$D(r)= \tfrac{k}{2}\cot\tfrac{kr}{2}\,,$$

where $\omega = \frac{1}{k}\left(2a^2 \sin\frac{kr}{2}\cos kr + b^2 \sin kr \cos\frac{kr}{2}\right)$.

Case 3: $\mathbf{c} < 0$

$$\begin{aligned}
A(r) &= \tfrac{1}{2\theta}\left(b^2 \sinh\tfrac{kr}{2}\sinh kr + 2a^2 \cosh\tfrac{kr}{2}\cosh kr\right),\\
B(r) &= -\tfrac{1}{\theta}\, a\, b,\\
C(r) &= \tfrac{1}{\theta}\left(2a^2 \sinh\tfrac{kr}{2}\sinh kr + b^2 \cosh\tfrac{kr}{2}\cosh kr\right),\\
D(r) &= \tfrac{k}{2}\coth\tfrac{kr}{2},
\end{aligned}$$

where $\theta = \frac{1}{k}\left(2a^2 \sinh\frac{kr}{2}\cosh kr + b^2 \sinh kr \cosh\frac{kr}{2}\right)$.

Therefore, this proves that the following theorem holds:

Theorem 2. *Let (M^n, g, J) be a Kähler manifold of constant holomorphic sectional curvarure c and let $P^\sigma(r)$ be a sufficiently small geodesic tube of radius r around a geodesic σ tangent to a vector u on M^n. Then the shape operator S^σ of tube $P^\sigma(r)$ at points $p = \exp_{\sigma(t)}(rv)$, such that $v(\sigma(t)) \perp u(\sigma(t))$, $g(Jv(\sigma(t)), u(\sigma(t))) = a$, can be represented by the matrix (21).*

Finally, since the case $v(\sigma(t)) \perp Ju(\sigma(t))$ is slightly more difficult than the case $v(\sigma(t)) = Ju(\sigma(t))$, but easier than the general case, where $g(Jv(\sigma(t)), u(\sigma(t))) = a$, we give here only the final result, i.e., the matrix of the shape operator in this case.

Theorem 3. *Let (M^n, g, J) be a Kähler manifold of constant holomorphic sectional curvature c and let $P^\sigma(r)$ be a sufficiently small geodesic tube of radius r around a geodesic σ tangent to a vector u on M^n. Then the shape operator S^σ of tube $P^\sigma(r)$ at points $p = \exp_{\sigma(t)}(rv)$, such that $v(\sigma(t)) = Ju(\sigma(t))$, is given by the following matrix:*

$$S^\sigma(p) = \begin{bmatrix} A(r) & 0 & 0 & \dots & 0 \\ 0 & B(r) & 0 & \dots & 0 \\ 0 & 0 & C(r) & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & C(r) \end{bmatrix} \tag{22}$$

with respect to the basis $\{E_1, E_2, \dots, E_{n-1}\}$ defined in Section 2, such that $E_2(\sigma(t)) = Jv(\sigma(t))$. The explicit expressions for the entries are as follows:

Case 1: $\mathbf{c} = 0$

$$A(r) = 0\,,\ B(r) = C(r) = \frac{1}{r}\,.$$

Case 2: $\mathbf{c} > 0$

$$\begin{aligned} A(r) &= -\tfrac{k}{2}\tan\tfrac{kr}{2}\,,\\ B(r) &= k\cot kr\,,\\ C(r) &= \tfrac{k}{2}\cot\tfrac{kr}{2}\,. \end{aligned}$$

Case 3: $\mathbf{c} < 0$

$$\begin{aligned} A(r) &= \tfrac{k}{2}\tanh\tfrac{kr}{2}\,,\\ B(r) &= k\coth kr\,,\\ C(r) &= \tfrac{k}{2}\coth\tfrac{kr}{2}\,. \end{aligned}$$

It is evident that the last result follows either directly from Theorem 2 (by replacing $a = 0\,, b = 1$ in (21)), or following the similar procedure as in Theorem 1 and Theorem 2 (i.e., solving the Jacobi equation (10) and computing the Jacobi vector fields). The author first used the latter method, and then checked the results after having proved Theorem 2.

Remark. After having completed this work, the author was informed by L. Vanhecke, that L. Gheysens derived the complete formulas for S^σ in his dissertation [10] and that the needed material is given in [12].

References

[1] P. Bueken, *Reflections and rotations in contact geometry*, doctoral dissertation, Katholi- eke Universiteit Leuven, 1992.

[2] M. Djorić, *Geometrija geodezijskih sfera i cevi*, doctoral dissertation, Faculty of Mathematics, University of Belgrade, 1994.

[3] M. Djorić, On characterizations of Sasakian space forms and locally φ-symmetric spaces, *Publ. Math. Debrecen* 46(1995), 1-23.

[4] M. Djorić, Geometry of geodesic tubes on Sasakian manifolds, *Proc. Colloquium on Differential Geometry, Debrecen*, 1995, Kluwer Ac. Publ., to appear.

[5] M. Djorić, Geometry of tubes about φ-geodesics on Sasakian manifolds, submitted.

[6] M. Djorić, Characterizations of complex space forms and locally Hermitian symmetric spaces by geodesic tubes, submitted.

[7] M. Djorić and L. Vanhecke, Almost Hermitian geometry, geodesic spheres and symmetries, *Math. Okayama Univ.* 32 (1990), 187-206.

[8] M. Djorić and L. Vanhecke, Geometry of geodesic spheres on Sasakian manifolds, *Rend. Sem. Mat. Univ. Pol. Torino* 49 (1991), 329-357.

[9] M. Djorić and L. Vanhecke, Geometry of tubes about characteristic curves on Sasakian manifolds, *Rend. Circ. Mat. Palermo* XLI (1992), 111-122.

[10] L. Gheysens, *Riemannse differentiaalmeetkunde van buisvormige omgevingen*, doctoral dissertation, Catholic University Leuven, 1981.

[11] A. Gray, *Tubes*, Addison-Wesley Publ. Co., Reading, (1990).

[12] A. Gray and L. Vanhecke, The volumes of tubes in a Riemannian manifold, *Rend. Sem. Mat. Univ. Politec. Torino* 39 (1981), 1-50.

[13] S. Kobayashi and K. Nomizu, *Foundations of Differential geometry*, I,II, Interscience Publ., New York, 1963, 1969.

[14] L. Vanhecke, Geometry in normal and tubular neighborhoods, *Rend. Sem. Fac. Sci. Univ. Cagliari, Supplemento al* Vol.58 (1988), 73-176.

[15] K. Yano and M. Kon, *Structures on manifolds*, Series in Pure Mathematics, 3, World Scientific Publ. Co., Singapore, 1984.

Faculty of Mathematics, University of Belgrade, Studentski trg 16, p.b. 550, 11000 Belgrade, Yugoslavia

E-mail: edjoric@ubbg.etf.bg.ac.yu

FILOMAT (Niš) 9:2(1995), 143–148
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

CURVES GENERATED BY MIRROR REFLECTIONS

Slavik V. Jablan

ABSTRACT. *Curves generated by mirror reflections are discussed from theory of symmetry, combinatorial geometry and knot theory point of view.*

The imitation of the three-dimensional arts of plaiting, weaving and basketry was the origin of interlacing and knotwork interlacing ornaments. Their highlights are the Celtic interlacing knotworks [1,2] (Fig.1a), Islamic layered patterns and Moorish floor and wall decorations.

The common geometrical construction principle for all such decorations is the use of (two-sided) mirrors incident to the edges of a square, triangular or hexagonal regular plane tiling, or perpendicular to its edges in their midpoints (Fig.1a). In the ideal case, after the series of consecutive reflections, the ray of light reaches its beginning point, defining a single closed curve [3]. In other cases, the result consists of several such curves.

The construction of such curves was occupied the attention of two most greatest painters–mathematicians: Leonardo and Dürer [1]. Some interesting geometrical and arithmetical properties of the curves mentioned are discovered by Paulus Gerdes [3,4,5]. Let us notice one more beautiful geometrical property: such curves can be obtained using only few different prototiles. For the construction of all the curves with internal mirrors incident to the edges, they are sufficient three prototiles in the case of a regular triangular tiling, five in the case of square, and 11 in the case of hexagonal regular tiling. We may also use their combinations occuring in the 11 uniform Archimedean tilings [6] (Fig.1b).

The symmetry of such curves is used for the reconstruction of Tamil designs [4], as well as for the classification of the Celtic frieze designs [1]. From the ornamental heritage, at first glance it looks that the symmetry is the mathematical basis for their construction and possible classification. But, the existence of such asymmetrical curves suggests the other approach. Trying to discover their common mathematical background, they appear two questions: how to construct such a perfect curve (this means, how to arrange the set of mirrors generating it), and how to classify the

1991 *Mathematics Subject Classification*: Primary 20H15, 57M25
Supported by Grant 0401 of FNS through Math. Inst. SANU

curves obtained. Our consideration we will restrict to the curves derived from the square tilings.

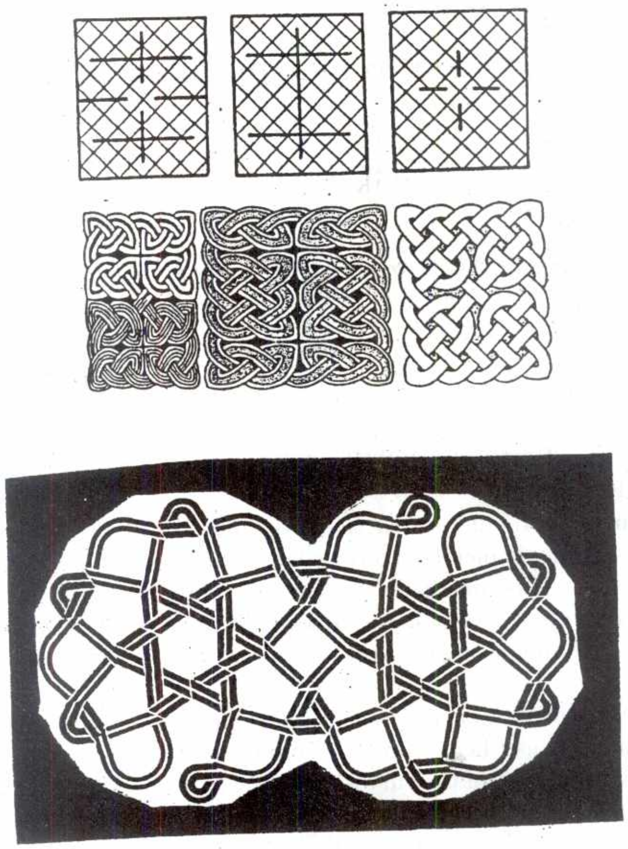


Figure 1

In principle, any polyomino [6] with mirrors on its border, and two-sided mirrors between cells or perpendicular on the internal edges in their midpoints, can be used for the creation of the corresponding curves. First, we construct all the different curves without use of internal mirrors, starting from different edge midpoints and ending in them, till the polyomino is exausted, i.e. uniformly covered by k curves. After that, we use "curve surgery" in order to obtain a single curve, according to the following rules: (a) any mirror introduced in a crossing point of two distinct curves connects them into one curve; (b) depending on the position of a mirror, a mirror introduced into a self-crossing point of an (oriented) curve makes no change, or breaks it into two closed curves. In every polyomino we may introduce $k-1$, k, $k+1$, $\ldots$, $2A-P/2$ internal two-sided mirrors, where A is area and P perimeter of the polyomino. Introducing minimal number of mirrors $k-1$ we first obtain a single curve, and in the next steps we try to preserve that result.

There is also a simple way to preserve such single closed curve: to add on the border of a polyomino a cell with three mirror-edges and one empty edge, or delete such a cell. This way, any such polyomino with a single curve can be transformed into a rectangle. Unfortunately, they are rectangular mirror-schemes which cannot be derived that way.

In the case of a rectangle with the sides a, b, the initial number of curves, obtained without use of internal mirrors, is $k = \gcd(a,b)$, so in order to obtain a single curve, the possible number of internal two-sided mirrors is $k-1$, k, $\ldots$, $2ab-a-b$. According to the rules for introduction of internal mirrors, we have the algorithm for the production of designs consisting of a simple closed curve: each from the first internal $k-1$ mirrors must be introduced in crossing points belonging to different curves. After that, when they are conected and transformed into a single line, we may introduce other mirrors, taking care about the number of lines, according to the rules mentioned. The next question is the classification of the curves obtained. First criterion we may use is the geometrical: two curves are equal iff there is a similarity transforming one into the other. Instead of considering the curves, we may consider the equal mirror arrangements defined in the same way. Having the algorithm for the construction of such perfect curves and the criterion for their equality, we may try to enumerate them: to find the number of all the different curves (i.e. mirror arrangements) which can be derived from a rectangle with the sides a, b, for a given number of internal mirrors m ($m \in \{k-1, k, \ldots, 2ab-a-b\}$). Unfortunately, we are very far from the general solution of this problem. Reasons for this are: every introduction of an internal mirror changes the whole structure, so it behaves like some kind of "Game of Life" or cellular automata.

Till this time, we have only few combinatorial results, obtained by non-standard use of Pólya enumeration theory [7,8]. Let a rectangle with sides a, b, $k = \gcd(a,b)$, be given, with the minimal number $k-1$ of two-sided internal mirrors incident to the edges of its square tiling. If $t = (ab - \mathrm{lcm}(a,b)) : (k(k-1))$, $x = a : (2k)$, $y = b : (2k)$, we have, for example, for $k = 5$, $a = 0 (\mathrm{mod}\ 10)$ and $b = 5 (\mathrm{mod}\ 10)$, the formula $14720t^4 - 576t^3 + 80t^2 + 32tx - 4xy - x$, giving the number of such curves.

The other point of view on the classification of such perfect curves is that of the knot theory. As it is mentioned before, every such curve can be simply transformed into an interlacing knotwork design, this means, a projection of some alternating knot. In the history of ornamental art, such curves occured most frequently as knotworks, then as plane curves. Even the name *Brahma-mudi* (Brahma's knot) [4] denoting such Tamil curves refers us to the knot theory [9,10,11]. In order to classify them, we will first transform every such knot projection into a proper (reduced) knot projection [11]- a knot projection without loops, by deleting cells with loops.

This way, we will obtain proper knot projections with the minimal number of crossings. Two such projections or knot diagrams are equal *iff* they are isotopic in projection plane as graphs, where the isotopy is required to respect overcrossing respectectively undercrossing [9]. For the classification of knots they are used different kinds of knot invariants: Alexander polynomials [9,10,11], Jones polynomials [11], Conway polynomials [10], etc. In order to classify the knot projections [12] we will define a new invariant of knot (or link) projections. Let consider a proper oriented knot diagram D with generators $g_1, \dots, g_n$. If the meeting point of generators g_i, g_j, g_k is "right", then $a_{ii} = t$, $a_{ij} = 1$, $a_{ik} = -1$; if it is "left", then $a_{ii} = -t$, $a_{ij} = 1$, $a_{ik} = -1$; in all the other cases $a_{ij} = 0$. The determinant $d(t) = |a_{ij}|$ is the polynomial invariant of D.

The writhe of D, denoted by $w(D)$, is the sum of signs of all the crossing points in D, where the sign is $+1$ if the crossing point is "right", and -1 if it is "left" [11]. The writhe is the most simple visible property of every knot projection: $|w(D)|$ is the type of the knot projection.

By the use of a computer program developed by Vesna Veličković, based on the algorithm of Dowker and Thistlethwaite [12], it is derived the complete list of non-isomorphic alternating knot projections for $3 \leq n \leq 11$, where n is the number of crossing points. For example, for $n = 8$ there are 27 non-isomsorphic projections of the 18 alternating prime knots. They are given by the ordering number of knot [9,10], by the sequences [12] and by the series of coefficients $(c_n, \dots, c_1)$ of $d(t)$.

8_1	4 10 16 14 12 2 8 6	1 0 -6 -2 10 8 -4 -4	
8_2	4 10 12 14 16 2 6 8	1 0 3 1 -1 1 -4 -4	
8_3	6 12 10 16 14 4 2 8	1 0 -6 0 10 0 -4 0	(e)
8_4	6 10 12 16 14 4 2 8	1 0 -1 -2 -6 1 4 0	
8_5	6 8 12 2 14 16 4 10	1 0 3 2 0 2 -4 -4	
$8'_6$	4 10 14 16 12 2 8 6	1 0 0 -2 -7 -5 4 4	
$8''_6$	4 10 16 12 14 2 8 6	1 0 -1 -1 -5 -5 4 4	
8_7	4 10 12 14 2 16 6 8	1 0 4 1 5 2 4 2	
$8'_8$	4 8 12 2 16 14 6 10	1 0 0 -1 -2 -3 -4 -2	
$8''_8$	4 10 8 14 2 16 6 12	1 0 1 -1 -2 -2 -4 -2	
8_9	6 10 12 14 16 4 2 8	1 0 4 0 2 0 -4 0	(e)
8_{10}	4 8 12 2 14 16 6 10	1 0 4 2 7 4 4 2	
$8'_{11}$	4 10 12 14 16 2 8 6	1 0 -3 1 -2 -4 4 4	
$8''_{11}$	4 10 12 16 14 2 6 8	1 0 -2 0 -3 -4 4 4	
$8'_{12}$	4 8 14 10 2 16 6 12	1 0 -3 -3 3 3 0 0	
$8''_{12}$	4 8 16 12 2 14 6 10	1 0 -4 0 8 0 -4 0	(e)
$8'''_{12}$	4 10 8 14 12 2 16 6	1 0 -2 0 -6 0 4 0	(e)
$8'_{13}$	4 10 12 14 2 16 8 6	1 0 -1 0 -2 -1 -4 -2	
$8''_{13}$	4 10 14 12 2 16 6 8	1 0 -1 -1 -1 -1 -4 -2	
$8'_{14}$	4 8 10 14 2 16 6 12	1 0 -1 -2 -2 -1 4 4	
$8''_{14}$	4 8 12 16 2 14 6 10	1 0 -1 1 -2 -2 4 4	
$8'''_{14}$	4 10 8 14 16 2 6 12	1 0 0 -1 -2 -1 4 4	
$8'_{15}$	4 8 12 2 14 6 16 10	1 0 1 1 -4 -3 -4 -8	
$8''_{15}$	4 8 14 2 12 16 6 10	1 0 1 2 -6 -2 -4 -8	
8_{16}	6 8 14 12 4 16 2 10	1 0 2 3 3 1 4 2	
8_{17}	6 8 12 14 4 16 2 10	1 0 2 0 1 0 -4 0	(e)
8_{18}	6 8 10 12 14 16 2 4	1 0 0 0 -2 0 -4 0	(e)

There are some important properties of the integer polynomial invariant $d(t) =$

$c_nt^n+\cdots+c_1t$: (a) for every alternating knot projection, the degree of $d(t)$ is n and $|c_n|=1$; (b) for every knot projection $|c_1|$ is equal to the type of the knot projection (i.e. $|c_1|=|w(D)|$); (c) $d(t)$ and $d(-t)$ correspond to the obverse (enantiomorphic, mirror symmetrical) knot diagrams; (d) for $n=0(\text{mod }2)$, a change of the orientation of an alternating knot projection results in the change of $d(t)$ to $d(-t)$; (e) for $n=1(\text{mod }2)$ a change of orientation of an alternating knot projection results in the change of $d(t)$ to $-d(-t)$. According to (c), (d) and (e), in the set of all the knot invariants $d(t)$ we may distinguish even functions ($d(t)=d(-t)$), containing only even degrees of t, corresponding to amphichiral knot projections (denoted by e), and odd functions ($d(t)=-d(-t)$), containing only odd degrees of t, which are invariant to a change of orientation of the knot projection. Let us also notice that invariant introduced makes distinction between non-isomorphic knot projections of composite knots (i.e. direct products of prime knots).

This invariant may be simply transferred to the alternating link projections. In this case, the result is the polynomial invariant of the form: $d(t)=c_nt^n+\cdots+c_kt^k$, where n is the number of crossing points, and k is the number of the link components. For every link, $|c_n|=1$. If a_i are the link components, $a_{ii}=w(a_i)$, and if $a_{ij}=\text{lk}(a_i,a_j)$ denotes the linking number of the components a_i, a_j, then $|c_k|=|\det(a_{ij})|$. For example, there are two non-isomorphic non-oriented projections of the link 6_3^2 (Figure 2).

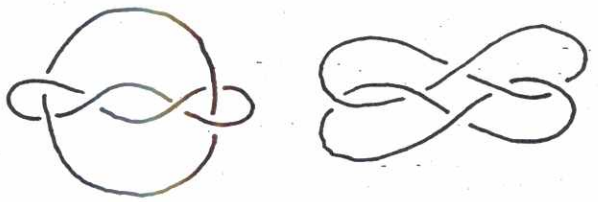


Figure 2

The problem exposed shows how the same (old) structures– perfect *pavitram* curves [3,4], may be regarded from the three different points of view: that of the theory of symmetry, combinatorial geometry and topology, taking us to a trip through mathematics, and introducing a new class of mirror-structures: curves generated by mirror reflections.

References

[1] Bain G., *Celtic art– the methods of construction*, Dover, New York, 1973.

[2] Cromwel P.R., *Celtic knotwork: Mathematical art*, The Math. Intelligencer, Vol. 15, No. 1 (1993), 36-47.

[3] Gerdes P.P.J., *On ethnomathematical research and symmetry*, Symmetry: Culture and Science, Vol. 1, No. 2 (1990), 154-170.

[4] Gerdes P., *Reconstruction and extension of lost symmetries: examples from the Tamil of South India*, Computers Math. Applic. Vol. 17, No. 4-6 (1989), 791-813.

[5] Gerdes P., *Geometria Sona*, Vol. 1, Instituto Superior Pedagogico, Maputo, 1993.
[6] Grünbaum B., Shephard G.C., *Tilings and Patterns*, Freeman, New York, 1979.
[7] Aigner M., *Combinatorial Theory*, Springer Verlag, Berlin, Heidelberg, New York, 1979.
[8] Pölya G, Read R.C., *Combinatorial enumeration of groups, graphs and chemical compounds*, Springer Verlag, New York, 1987.
[9] Burde G., Zieschang H., *Knots*, Walter de Greyter, 1985.
[10] Kauffman L.H., *On knots*, Princeton University Press, Princeton, 1987.
[11] Kohno T. (ed.), *New developments in the theory of knots*, World Scientific, Singapoore, New Jersey, London, Hong Kong, 1989.
[12] Dowker C.H., Thistlethwaite M.B., *Classification of knot projections*, Topology Appl., **16** (1983), 19-31.

Department of Mathematics, Phylosophical Faculty, 18000 Niš Ćirila i Metodija 2, Yugoslavia

E-mail: eslavik@ubbg.etf.bg.ac.yu

FILOMAT (Niš) 9:2 (1995), 149–159
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ON A FAMILY OF TENSOR FIELDS IN A GENERALIZED RIEMANNIAN SPACE

Svetislav M. Minčić

ABSTRACT. *In a subspace GR_M of a generalized Riemannian space GR_N we observe a familly of tensor fields* (1.1), *which contains as particular cases tangent and normal vectors of the subspace as well the curvature vector q^α of a curve in the subspace. Because of non-symmetry of Cristoffel symbols we define four kinds of derivational formulas of the above mentioned familly, as well six integrability conditions of these formulas. As particular cases one obtains Gauss-Codazzi eqations of the subspace and corensponding eqations for q^α. In this manner derivational formulas of Riemannian space are generalized, as well as their integrability conditions, i.e. the Gauss-Codazzi equations.*

0. Introduction

A generalized Riemannian space GR_N in the sense of Eisenhart [1], [2] is a differentiable manifold in which a non-symmetric basic tensor $a_{\alpha\beta} \neq a_{\beta\alpha}$ is introduced. If in the GR_N the coordinates are y^α $(\alpha = 1, \dots, N)$, then by the equations

$$y^\alpha = y^\alpha(x^1, \dots, x^M) \ (M < N) \tag{0.1}$$

a subspace GR_M of the space GR_N is defined. If g_{ij} is the basic tensor of this subspace, then in general $g_{ij} \neq g_{ji}$. In every point of the subspace we can observe $N - M$ unit, mutually orthogonal vectors $N^\alpha_{(\rho)}$ $(\rho = M+1, \dots, N)$, which are also orthogonal to GR_M, i.e. to the vectors (for fixed i)

$$t^\alpha_i = y^\alpha_{,i} = \partial y^\alpha / \partial x^i, \tag{0.2}$$

where the comma (,) signifies a partial derivative. We remark that in this work the Greek indices take values $1, \dots, N$ and the Latin values $1, \dots, M$ $(M < N)$, except indices in brackets, which take values $M+1, \dots, N$.

Let $g_{\underline{ij}}$ signify the symmetrisation, and $g_{\underset{\vee}{ij}}$ antisymmetrsation with respect to i, j and analogically in other cases. Then ([7], [8]):

$$a_{\alpha\beta} t^\alpha_i t^\beta_j = g_{ij}, \tag{0.3}$$

$$a_{\underline{\alpha\beta}} t^\alpha_i t^\beta_j = g_{\underline{ij}}, \qquad a_{\underset{\vee}{\alpha\beta}} t^\alpha_i t^\beta_j = g_{\underset{\vee}{ij}}, \tag{0.4 a,b}$$

$$a_{\underline{\alpha\pi}} a^{\underline{\pi\beta}} = \delta^\beta_\alpha, \qquad g_{\underline{ip}} g^{\underline{pj}} = \delta^j_i, \tag{0.5 a,b}$$

$$a_{\underline{\alpha\beta}} N^\alpha_{(\rho)} N^\beta_{(\sigma)} = e_{(\rho)} \delta_{(\rho\sigma)} \qquad (e_{(\rho)} = \pm 1), \quad a_{\underline{\alpha\beta}} N^\alpha_{(\rho)} t^\beta_j = 0. \tag{0.6 a,b}$$

Supported by Grant 0401A of RFNS through Math. Inst. SANU

The Cristoffel symbols of the GR_N are given by

$$\Gamma_{\alpha.\beta\gamma} = \frac{1}{2}(a_{\beta\alpha,\gamma} - a_{\beta\gamma,\alpha} + a_{\alpha\gamma,\beta}), \quad \Gamma^{\alpha}_{\beta\gamma} = a^{\underline{\alpha\pi}}\Gamma_{\pi.\beta\gamma} \tag{0.7a, b}$$

and analogically for GR_M by g_{ij}. Then we have, for example,

$$\Gamma_{\alpha.\beta\gamma} \neq \Gamma_{\alpha.\gamma\beta}, \quad \Gamma^{\alpha}_{\beta\gamma} \neq \Gamma^{\alpha}_{\gamma\beta}.$$

Because of non-summetry of Cristoffel symbols, we can define 4 kinds of covariant derivative [3], [4]. For example,

$$\begin{aligned}
t^{\alpha}_{i\underset{1}{|}m} &= t^{\alpha}_{i,m} + \Gamma^{\alpha}_{\pi\mu}t^{\pi}_{i}t^{\mu}_{m} - \Gamma^{p}_{im}t^{\alpha}_{p},\\
t^{\alpha}_{i\underset{2}{|}m} &= t^{\alpha}_{i,m} + \Gamma^{\alpha}_{\mu\pi}t^{\pi}_{i}t^{\mu}_{m} - \Gamma^{p}_{mi}t^{\alpha}_{p},\\
t^{\alpha}_{i\underset{3}{|}m} &= t^{\alpha}_{i,m} + \Gamma^{\alpha}_{\pi\mu}t^{\pi}_{i}t^{\mu}_{m} - \Gamma^{p}_{mi}t^{\alpha}_{p},\\
t^{\alpha}_{i\underset{4}{|}m} &= t^{\alpha}_{i,m} + \Gamma^{\alpha}_{\mu\pi}t^{\pi}_{i}t^{\mu}_{m} - \Gamma^{p}_{im}t^{\alpha}_{p},
\end{aligned} \tag{0.8a}$$

$$\begin{aligned}
N^{\alpha}_{(\rho)\underset{1}{|}m} &= N^{\alpha}_{(\rho)\underset{3}{|}m} = N^{\alpha}_{(\rho),m} + \Gamma^{\alpha}_{\pi\mu}N^{\pi}_{(\rho)}t^{\mu}_{m},\\
N^{\alpha}_{(\rho)\underset{2}{|}m} &= N^{\alpha}_{(\rho)\underset{4}{|}m} = N^{\alpha}_{(\rho),m} + \Gamma^{\alpha}_{\mu\pi}N^{\pi}_{(\rho)}t^{\mu}_{m}
\end{aligned} \tag{0.8b}$$

We also obtain 4 kinds of derivational formulas (see (16) and (37′) in [5]):

$$t^{\alpha}_{i\underset{\theta}{|}m} = \underset{\theta}{\Phi}{}^{p}_{im}t^{\alpha}_{p} + \sum_{\rho}\underset{\theta}{\Omega}_{(\rho)im}N^{\alpha}_{(\rho)}, \tag{0.9a}$$

$$N^{\alpha}_{(\sigma)\underset{\theta}{|}m} = -e_{(\sigma)}g^{\underline{ps}}\underset{\theta}{\Omega}_{(\sigma)sm}t^{\alpha}_{p} + \sum_{\rho}\underset{\theta}{\psi}_{(\rho\sigma)m}N^{\alpha}_{(\rho)}, \quad \underset{\theta}{\psi}_{(\sigma\sigma)m} = 0, \tag{0.9b}$$

where $\theta = 1, 2, 3, 4$ signifies the kind of covariant derivative.

On the base of $(0.8b)$ it is

$$\underset{1}{\Omega}_{(\rho)ij} = \underset{3}{\Omega}_{(\rho)ij}, \quad \underset{2}{\Omega}_{(\rho)ij} = \underset{4}{\Omega}_{(\rho)ij}, \tag{0.10a}$$

$$\underset{1}{\psi}_{(\rho\sigma)m} = \underset{3}{\psi}_{(\rho\sigma)m}, \quad \underset{2}{\psi}_{(\rho\sigma)m} = \underset{4}{\psi}_{(\rho\sigma)m}, \tag{0.10b}$$

and with respect to (48′, 24′) in [5] we have

$$\underset{2}{\Phi}{}^{h}_{im} = -\underset{1}{\Phi}{}^{h}_{im},\ \underset{3}{\Phi}{}^{h}_{im} = \underset{1}{\Phi}{}^{h}_{im} + 2\Gamma^{h}_{\underset{\vee}{im}},\ \underset{4}{\Phi}{}^{h}_{im} = -\underset{1}{\Phi}{}^{h}_{im} - 2\Gamma^{h}_{\underset{\vee}{im}} \tag{0.11}$$

1. Derivational formula of the field and integrability conditions

Suppouse that in the points of the GR_M a family of tensor fields is defined:

$$\lambda^{\alpha}_{(\tau)i} = b^{s}_{(\tau)i} t^{\alpha}_{s} + \sum_{\rho=M+1}^{N} c_{(\rho\tau)i} N^{\alpha}_{(\rho)}. \tag{1.1}$$

Applying to (1.1) the covariant derivative of kind $\mu \in \{1,2,3,4\}$ with respect to x^m and using (0.9), we obtain

$$\lambda^{\alpha}_{(\tau)i\underset{\mu}{|}m} = \underset{\mu}{q}{}^{p}_{(\tau)im} t^{\alpha}_{p} + \sum_{\rho} \underset{\mu}{r}_{(\rho\tau)im} N^{\alpha}_{(\rho)}, \tag{1.2}$$

where

$$\underset{\mu}{q}{}^{p}_{(\tau)im} = b^{p}_{(\tau)i\underset{\mu}{|}m} + b^{s}_{(\tau)i} \underset{\mu}{\Phi}{}^{p}_{sm} - \sum_{\rho} e_{(\rho)} c_{(\rho\tau)i} g^{\underline{ps}} \underset{\mu}{\Omega}_{(\rho)sm}, \tag{1.3a}$$

$$\underset{\mu}{r}_{(\rho\tau)im} = b^{s}_{(\tau)i} \underset{\mu}{\Omega}_{(\rho)sm} + c_{(\rho\tau)i\underset{\mu}{|}m} + \sum_{\sigma} c_{(\sigma\tau)i} \underset{\mu}{\psi}_{(\rho\sigma)m}. \tag{1.3b}$$

The formula (1.2) is *derivational formula of the field* $\lambda^{\alpha}_{(\tau)i}$.

Applying to (1.2) covariant derivative of kind $\nu \in \{1,2,3,4\}$ with respect to x^n and using (0.9) repeatedly, we get

$$\begin{aligned} \lambda^{\alpha}_{(\tau)i\underset{\mu}{|}m\underset{\nu}{|}n} = {} & (\underset{\mu}{q}{}^{p}_{(\tau)im\underset{\nu}{|}n} + \underset{\mu}{q}{}^{s}_{(\tau)im} \underset{\nu}{\Phi}{}^{p}_{sn} - \sum_{\sigma} e_{(\sigma)} g^{\underline{ps}} \underset{\mu}{r}_{(\sigma\tau)im} \underset{\nu}{\Omega}_{(\sigma)sn}) t^{\alpha}_{p} + \\ & + \sum_{\rho} (\underset{\mu}{q}{}^{p}_{(\tau)im} \underset{\nu}{\Omega}_{(\rho)pn} + \underset{\mu}{r}_{(\rho\tau)im\underset{\nu}{|}n} + \sum_{\sigma} \underset{\mu}{r}_{(\sigma\tau)im} \underset{\nu}{\psi}_{(\rho\sigma)n}) N^{\alpha}_{(\rho)}, \end{aligned} \tag{1.4}$$

where the tensors $\underset{\mu}{q}{}^{p}_{(\tau)im}$, $\underset{\mu}{r}_{(\rho\tau)im}$ are given by (1.3). From here we obtain

$$\begin{aligned} \lambda^{\alpha}_{(\tau)i\underset{\mu}{|}m\underset{\nu}{|}n} - \lambda^{\alpha}_{(\tau)i\underset{\nu}{|}n\underset{\mu}{|}m} = {} & [\underset{\mu}{q}{}^{p}_{(\tau)im\underset{\nu}{|}n} - \underset{\nu}{q}{}^{p}_{(\tau)in\underset{\mu}{|}m} + \underset{\mu}{q}{}^{s}_{(\tau)im} \underset{\nu}{\Phi}{}^{p}_{sn} - \underset{\nu}{q}{}^{s}_{(\tau)in} \underset{\mu}{\Phi}{}^{p}_{sm} - \\ & - \sum_{\sigma} e_{(\sigma)} g^{\underline{ps}} (\underset{\mu}{r}_{(\sigma\tau)im} \underset{\nu}{\Omega}_{(\sigma)sn} - \underset{\nu}{r}_{(\sigma\tau)in} \underset{\mu}{\Omega}_{(\sigma)sm})] t^{\alpha}_{p} + \\ & + \sum_{\rho} [\underset{\mu}{q}{}^{p}_{(\tau)im} \underset{\nu}{\Omega}_{(\rho)pn} - \underset{\nu}{q}{}^{p}_{(\tau)in} \underset{\mu}{\Omega}_{(\rho)pm} + \underset{\mu}{r}_{(\rho\tau)im\underset{\nu}{|}n} - \underset{\nu}{r}_{(\rho\tau)in\underset{\mu}{|}m} + \\ & + \sum_{\sigma} (\underset{\mu}{r}_{(\sigma\tau)im} \underset{\nu}{\psi}_{(\rho\sigma)n} - \underset{\nu}{r}_{(\sigma\tau)in} \underset{\mu}{\psi}_{(\rho\sigma)m})] N^{\alpha}_{(\rho)}. \end{aligned} \tag{1.5}$$

Applying the identities of Ricci-type (7), (11), (56) from [3] and (12), (13), (46) from [4] to the family of tensor fields $\lambda^{\alpha}_{(\tau)i}$, we get

$$(1.6)\qquad \lambda^{\alpha}_{(\tau)i\underset{1}{|}m\underset{1}{|}n}-\lambda^{\alpha}_{(\tau)i\underset{1}{|}n\underset{1}{|}m}=\underset{1}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{1}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}-2\Gamma^{p}_{\underset{\vee}{mn}}\lambda^{\alpha}_{(\tau)i\underset{1}{|}p},$$

$$(1.7)\qquad \lambda^{\alpha}_{(\tau)i\underset{2}{|}m\underset{2}{|}n}-\lambda^{\alpha}_{(\tau)i\underset{2}{|}n\underset{2}{|}m}=\underset{2}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}+2\Gamma^{p}_{\underset{\vee}{mn}}\lambda^{\alpha}_{(\tau)i\underset{2}{|}p},$$

$$(1.8)\qquad \lambda^{\alpha}_{(\tau)i\underset{1}{|}m\underset{2}{|}n}-\lambda^{\alpha}_{(\tau)i\underset{2}{|}n\underset{1}{|}m}=\underset{3}{R}{}^{\alpha}_{\pi mn}\lambda^{\pi}_{(\tau)i}-\underset{3}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p},$$

$$(1.9)\qquad \lambda^{\alpha}_{(\tau)i\underset{3}{|}m\underset{3}{|}n}-\lambda^{\alpha}_{(\tau)i\underset{3}{|}n\underset{3}{|}m}=\underset{1}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}+2\Gamma^{p}_{\underset{\vee}{mn}}\lambda^{\alpha}_{(\tau)i\underset{3}{|}p},$$

$$(1.10)\qquad \lambda^{\alpha}_{(\tau)i\underset{4}{|}m\underset{4}{|}n}-\lambda^{\alpha}_{(\tau)i\underset{4}{|}n\underset{4}{|}m}=\underset{2}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{1}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}-2\Gamma^{p}_{\underset{\vee}{mn}}\lambda^{\alpha}_{(\tau)i\underset{4}{|}p},$$

$$(1.11)\qquad \lambda^{\alpha}_{(\tau)i\underset{3}{|}m\underset{4}{|}n}-\lambda^{\alpha}_{(\tau)i\underset{4}{|}n\underset{3}{|}m}=\underset{4}{R}{}^{\alpha}_{\pi mn}\lambda^{\pi}_{(\tau)i}+\underset{3}{R}{}^{p}_{inm}\lambda^{\alpha}_{(\tau)p},$$

where

$$(1.12)\qquad \underset{1}{R}{}^{i}_{jmn}=\Gamma^{i}_{jm,n}-\Gamma^{i}_{jn,m}+\Gamma^{p}_{jm}\Gamma^{i}_{pn}-\Gamma^{p}_{jn}\Gamma^{i}_{pm},$$

$$(1.13)\qquad \underset{2}{R}{}^{i}_{jmn}=\Gamma^{i}_{mj,n}-\Gamma^{i}_{nj,m}+\Gamma^{p}_{mj}\Gamma^{i}_{np}-\Gamma^{p}_{nj}\Gamma^{i}_{mp},$$

$$(1.14)\qquad \underset{3}{R}{}^{i}_{jmn}=\Gamma^{i}_{jm,n}-\Gamma^{i}_{nj,m}+\Gamma^{p}_{jm}\Gamma^{i}_{np}-\Gamma^{p}_{nj}\Gamma^{i}_{pm}+\Gamma^{p}_{nm}(\Gamma^{i}_{pj}-\Gamma^{i}_{jp}),$$

$$(1.15)\qquad \begin{aligned}\underset{3}{R}{}^{\alpha}_{\beta mn}=&(\Gamma^{\alpha}_{\beta\mu,\nu}-\Gamma^{\alpha}_{\nu\beta,\mu}+\Gamma^{\pi}_{\beta\mu}\Gamma^{\alpha}_{\nu\pi}-\Gamma^{\pi}_{\nu\beta}\Gamma^{\alpha}_{\pi\mu})t^{\mu}_{m}t^{\nu}_{n}+\\&+2\Gamma^{\alpha}_{\underset{\vee}{\beta\mu}}(y^{\mu}_{,mn}-\Gamma^{p}_{nm}t^{\mu}_{p}),\end{aligned}$$

$$(1.16)\qquad \begin{aligned}\underset{4}{R}{}^{\alpha}_{\beta mn}=&(\Gamma^{\alpha}_{\beta\mu,\nu}-\Gamma^{\alpha}_{\nu\beta,\mu}+\Gamma^{\pi}_{\beta\mu}\Gamma^{\alpha}_{\nu\pi}-\Gamma^{\pi}_{\nu\beta}\Gamma^{\alpha}_{\pi\mu})t^{\mu}_{m}t^{\nu}_{n}+\\&+2\Gamma^{\alpha}_{\underset{\vee}{\beta\mu}}(y^{\mu}_{,mn}-\Gamma^{p}_{mn}t^{\mu}_{p}),\end{aligned}$$

The magnitudes $\underset{t}{R}{}^{\alpha}_{\beta\mu\nu}$, $\underset{t}{R}{}^{i}_{jmn}$, $(t=1,2,3)$ are tensors and we call them curvature tensors of the space, and of the subspace, respectively the 1st, the 2nd and the 3rd kind. The magnitudes $\underset{3}{R}{}^{\alpha}_{\beta mn}$, $\underset{4}{R}{}^{\alpha}_{\beta mn}$ are tensors too and we call them curvature tensors of the 3rd, of the 4th kind of the space GR_N in relation to the subspace GR_M.

1.1. Taking in (1.5) $\mu=\nu=1$, in (1.6) replacing $\lambda^{\alpha}_{(\tau)i\underset{1}{|}p}$ by virtue of (1.2) and equating the obtained results, we obtain the *first integrability condition of the* *derivational formula*(1.2):

$$\begin{aligned}
&\underset{1}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{1}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}-2\Gamma^{p}_{\underset{\vee}{mn}}(\underset{1}{q}{}^{s}_{(\tau)ip}t^{\alpha}_{s}+\sum_{\rho}\underset{1}{r}{}_{(\rho\tau)ip}N^{\alpha}_{(\rho)})=\\
&=[\underset{1}{q}{}^{p}_{(\tau)im\underset{1}{|}n}-\underset{1}{q}{}^{p}_{(\tau)in\underset{1}{|}m}+\underset{1}{q}{}^{s}_{(\tau)im}\underset{1}{\Phi}{}^{p}_{sn}-\underset{1}{q}{}^{s}_{(\tau)in}\underset{1}{\Phi}{}^{p}_{sm}-\\
&-\sum_{\sigma}e_{(\sigma)}g^{\underline{ps}}(\underset{1}{r}{}_{(\sigma\tau)im}\underset{1}{\Omega}{}_{(\sigma)sn}-\underset{1}{r}{}_{(\sigma\tau)in}\underset{1}{\Omega}{}_{(\sigma)sm})]t^{\alpha}_{p}+\\
&+\sum_{\rho}[\underset{1}{q}{}^{p}_{(\tau)im}\underset{1}{\Omega}{}_{(\rho)pn}-\underset{1}{q}{}^{p}_{(\tau)in}\underset{1}{\Omega}{}_{(\rho)pm}+\underset{1}{r}{}_{(\rho\tau)im\underset{1}{|}n}-\underset{1}{r}{}_{(\rho\tau)in\underset{1}{|}m}+\\
&+\sum_{\sigma}(\underset{1}{r}{}_{(\sigma\tau)im}\underset{1}{\psi}{}_{(\rho\sigma)n}-\underset{1}{r}{}_{(\sigma\tau)in}\underset{1}{\psi}{}_{(\rho\sigma)m})]N^{\alpha}_{(\rho)}.
\end{aligned}\tag{1.17}$$

Multiplying (1.17) by $a_{\underline{\alpha\beta}}t^{\beta}_{h}$ and using (0.4a), (0.6b), (0.5b), we obtain

$$\begin{aligned}
&\underset{1}{R}{}_{\beta\pi\mu\nu}t^{\beta}_{h}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{1}{R}{}^{p}_{imn}a_{\underline{\alpha\beta}}t^{\beta}_{h}\lambda^{\alpha}_{(\tau)p}-2\Gamma^{p}_{\underset{\vee}{mn}}\underset{1}{q}{}^{s}_{(\tau)ip}g_{\underline{hs}}=\\
&=(\underset{1}{q}{}^{p}_{(\tau)im\underset{1}{|}n}-\underset{1}{q}{}^{p}_{(\tau)in\underset{1}{|}m}+\underset{1}{q}{}^{s}_{(\tau)im}\underset{1}{\Phi}{}^{p}_{sn}-\underset{1}{q}{}^{s}_{(\tau)in}\underset{1}{\Phi}{}^{p}_{sm})g_{\underline{hp}}-\\
&-\sum_{\sigma}e_{(\sigma)}(\underset{1}{r}{}_{(\sigma\tau)im}\underset{1}{\Omega}{}_{(\sigma)hn}-\underset{1}{r}{}_{(\sigma\tau)in}\underset{1}{\Omega}{}_{(\sigma)hm}).
\end{aligned}\tag{1.17'}$$

If we multiply (1.17) by $a_{\underline{\alpha\beta}}N^{\beta}_{(\varphi)}$ and take into consideration (0.6), we get

$$\begin{aligned}
&\underset{1}{R}{}_{\beta\pi\mu\nu}N^{\beta}_{(\varphi)}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{1}{R}{}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}N^{\beta}_{(\varphi)}-2\Gamma^{p}_{\underset{\vee}{mn}}\underset{1}{r}{}_{(\varphi\tau)ip}e_{(\varphi)}=\\
&=e_{(\varphi)}[\underset{1}{q}{}^{p}_{(\tau)im}\underset{1}{\Omega}{}_{(\varphi)pn}-\underset{1}{q}{}^{p}_{(\tau)in}\underset{1}{\Omega}{}_{(\varphi)pm}+\underset{1}{r}{}_{(\varphi\tau)im\underset{1}{|}n}-\underset{1}{r}{}_{(\varphi\tau)in\underset{1}{|}m}+\\
&+\sum_{\sigma}(\underset{1}{r}{}_{(\sigma\tau)im}\underset{1}{\psi}{}_{(\varphi\sigma)n}-\underset{1}{r}{}_{(\sigma\tau)in}\underset{1}{\psi}{}_{(\varphi\sigma)m})].
\end{aligned}\tag{1.17''}$$

1.2. Taking $\mu=\nu=2$ in (1.5) and equating with (1.7), using (1.2), we obtain the *second integrability condition of the derivational formula*(1.2):

$$\begin{aligned}
&\underset{2}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}+2\Gamma^{p}_{\underset{\vee}{mn}}(\underset{1}{q}{}^{s}_{(\tau)ip}t^{\alpha}_{s}+\sum_{\rho}\underset{2}{r}{}_{(\rho\tau)ip}N^{\alpha}_{(\rho)})=\\
&=[\underset{2}{q}{}^{p}_{(\tau)im\underset{2}{|}n}-\underset{2}{q}{}^{p}_{(\tau)in\underset{2}{|}m}+\underset{2}{q}{}^{s}_{(\tau)im}\underset{2}{\Phi}{}^{p}_{sn}-\underset{2}{q}{}^{s}_{(\tau)in}\underset{2}{\Phi}{}^{p}_{sm}-\\
&-\sum_{\sigma}e_{(\sigma)}g^{\underline{ps}}(\underset{2}{r}{}_{(\sigma\tau)im}\underset{2}{\Omega}{}_{(\sigma)sn}-\underset{2}{r}{}_{(\sigma\tau)in}\underset{2}{\Omega}{}_{(\sigma)sm})]t^{\alpha}_{p}+\\
&+\sum_{\rho}[\underset{2}{q}{}^{p}_{(\tau)im}\underset{2}{\Omega}{}_{(\rho)pn}-\underset{2}{q}{}^{p}_{(\tau)in}\underset{2}{\Omega}{}_{(\rho)pm}+\underset{2}{r}{}_{(\rho\tau)im\underset{2}{|}n}-\underset{2}{r}{}_{(\rho\tau)in\underset{2}{|}m}+\\
&+\sum_{\sigma}(\underset{2}{r}{}_{(\sigma\tau)im}\underset{2}{\psi}{}_{(\rho\sigma)n}-\underset{2}{r}{}_{(\sigma\tau)in}\underset{2}{\psi}{}_{(\rho\sigma)m})]N^{\alpha}_{(\rho)}.
\end{aligned}\tag{1.18}$$

Multiplying (1.18) by $a_{\underline{\alpha\beta}}t^{\beta}_{h}$, we obtain

$$\begin{aligned}
&\underset{2}{R}_{\beta\pi\mu\nu}t^{\beta}_{h}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}^{p}_{imn}a_{\underline{\alpha\beta}}t^{\beta}_{h}\lambda^{\alpha}_{(\tau)p}+2\Gamma^{p}_{\underset{\vee}{mn}}\underset{2}{q}^{s}_{(\tau)ip}g_{\underline{hs}}=\\
(1.18')\quad &=(\underset{2}{q}^{p}_{(\tau)im\underset{2}{|}n}-\underset{2}{q}^{p}_{(\tau)in\underset{2}{|}m}+\underset{2}{q}^{s}_{(\tau)im}\underset{2}{\Phi}^{p}_{sn}-\underset{2}{q}^{s}_{(\tau)in}\underset{2}{\Phi}^{p}_{sm})g_{\underline{hp}}-\\
&-\sum_{\sigma}e_{(\sigma)}(\underset{2}{r}_{(\sigma\tau)im}\underset{2}{\Omega}_{(\sigma)hn}-\underset{2}{r}_{(\sigma\tau)in}\underset{2}{\Omega}_{(\sigma)hm}),
\end{aligned}$$

and multiplying the same equation by $a_{\underline{\alpha\beta}}N^{\beta}_{(\varphi)}$:

$$\begin{aligned}
&\underset{2}{R}_{\beta\pi\mu\nu}N^{\beta}_{(\varphi)}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}N^{\beta}_{(\varphi)}+2\Gamma^{p}_{\underset{\vee}{mn}}\underset{2}{r}_{(\varphi\tau)ip}e_{(\varphi)}=\\
(1.18")\quad &=e_{(\varphi)}[\underset{2}{q}^{p}_{(\tau)im}\underset{2}{\Omega}_{(\varphi)pn}-\underset{2}{q}^{p}_{(\tau)in}\underset{2}{\Omega}_{(\varphi)pm}+\underset{2}{r}_{(\varphi\tau)im\underset{2}{|}n}-\underset{2}{r}_{(\varphi\tau)in\underset{2}{|}m}+\\
&+\sum_{\sigma}(\underset{2}{r}_{(\sigma\tau)im}\underset{2}{\psi}_{(\varphi\sigma)n}-\underset{2}{r}_{(\sigma\tau)in}\underset{2}{\psi}_{(\varphi\sigma)m})].
\end{aligned}$$

1.3. If in (1.5) we replace $\mu=1$, $\nu=2$ and use (1.8), we get *the third integrability condition of the derivational formula* (1.2):

$$\begin{aligned}
&\underset{3}{R}^{\alpha}_{\pi mn}\lambda^{\pi}_{(\tau)i}-\underset{3}{R}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}=\\
&=[\underset{1}{q}^{p}_{(\tau)im\underset{2}{|}n}-\underset{2}{q}^{p}_{(\tau)in\underset{1}{|}m}+\underset{1}{q}^{s}_{(\tau)im}\underset{2}{\Phi}^{p}_{sn}-\underset{2}{q}^{s}_{(\tau)in}\underset{1}{\Phi}^{p}_{sm}-\\
(1.19)\quad &-\sum_{\sigma}e_{(\sigma)}g^{\underline{ps}}(\underset{1}{r}_{(\sigma\tau)im}\underset{2}{\Omega}_{(\sigma)sn}-\underset{2}{r}_{(\sigma\tau)in}\underset{1}{\Omega}_{(\sigma)sm})]t^{\alpha}_{p}+\\
&+\sum_{\rho}[\underset{1}{q}^{p}_{(\tau)im}\underset{2}{\Omega}_{(\rho)pn}-\underset{2}{q}^{p}_{(\tau)in}\underset{1}{\Omega}_{(\rho)pm}+\underset{1}{r}_{(\rho\tau)im\underset{2}{|}n}-\underset{2}{r}_{(\rho\tau)in\underset{1}{|}m}+\\
&+\sum_{\sigma}(\underset{1}{r}_{(\sigma\tau)im}\underset{2}{\psi}_{(\rho\sigma)n}-\underset{2}{r}_{(\sigma\tau)in}\underset{1}{\psi}_{(\rho\sigma)m})]N^{\alpha}_{(\rho)}.
\end{aligned}$$

Multiplying the pervious equation by $a_{\underline{\alpha\beta}}t^{\beta}_{h}$, we obtain

$$\begin{aligned}
&\underset{3}{R}_{\beta\pi mn}t^{\beta}_{h}\lambda^{\pi}_{(\tau)i}-\underset{3}{R}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}t^{\beta}_{h}=\\
(1.19')\quad &=(\underset{1}{q}^{p}_{(\tau)im\underset{2}{|}n}-\underset{2}{q}^{p}_{(\tau)in\underset{1}{|}m}+\underset{1}{q}^{s}_{(\tau)im}\underset{2}{\Phi}^{p}_{sn}-\underset{2}{q}^{s}_{(\tau)in}\underset{1}{\Phi}^{p}_{sm})g_{\underline{hp}}-\\
&-\sum_{\sigma}e_{(\sigma)}(\underset{1}{r}_{(\sigma\tau)im}\underset{2}{\Omega}_{(\sigma)hn}-\underset{2}{r}_{(\sigma\tau)in}\underset{1}{\Omega}_{(\sigma)hm}).
\end{aligned}$$

where $\underset{3}{R}_{\beta\pi mn}=a_{\underline{\alpha\beta}}\underset{3}{R}^{\alpha}_{\pi mn}$, and $\underset{3}{R}^{\alpha}_{\pi mn}$ is given by (1.15).

Multipling (1.19) by $a_{\underline{\alpha\beta}}N^{\beta}_{(\varphi)}$, we get

$$\begin{aligned}(1.19")\quad &\underset{3}{R}_{\beta\pi mn}N^{\beta}_{(\varphi)}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{3}{R}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}N^{\beta}_{(\varphi)}=\\ &=e_{(\varphi)}[\underset{1}{q}^{p}_{(\tau)im}\underset{2}{\Omega}_{(\varphi)pn}-\underset{2}{q}^{p}_{(\tau)in}\underset{1}{\Omega}_{(\varphi)pm}+\underset{1}{r}_{(\varphi\tau)im\underset{2}{|}n}-\underset{2}{r}_{(\varphi\tau)in\underset{1}{|}m}+\\ &+\sum_{\sigma}(\underset{1}{r}_{(\sigma\tau)im}\underset{2}{\psi}_{(\varphi\sigma)n}-\underset{2}{r}_{(\sigma\tau)in}\underset{1}{\psi}_{(\varphi\sigma)m})].\end{aligned}$$

1.4. By replacing $\mu=\nu=3$ in (1.5) and applying (1.9), we obtain *the fourth integrability condition of the derivational formula* (1.2):

$$\begin{aligned}&\underset{1}{R}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}^{p}_{imn}\lambda^{\alpha}_{(\tau)p}+2\Gamma^{p}_{\underset{\vee}{mn}}(\underset{3}{q}^{s}_{(\tau)ip}t^{\alpha}_{s}+\sum_{\rho}\underset{3}{r}_{(\rho\tau)ip}N^{\alpha}_{(\rho)})=\\ &[\underset{3}{q}^{p}_{(\tau)im\underset{3}{|}n}-\underset{3}{q}^{p}_{(\tau)in\underset{3}{|}m}+\underset{3}{q}^{s}_{(\tau)im}\underset{3}{\Phi}^{p}_{sn}-\underset{3}{q}^{s}_{(\tau)in}\underset{3}{\Phi}^{p}_{sm}-\\ (1.20)\quad &-\sum_{\sigma}e_{(\sigma)}g^{\underline{ps}}(\underset{3}{r}_{(\sigma\tau)im}\underset{3}{\Omega}_{(\sigma)sn}-\underset{3}{r}_{(\sigma\tau)in}\underset{3}{\Omega}_{(\sigma)sm})]t^{\alpha}_{p}+\\ &+\sum_{\rho}[\underset{3}{q}^{p}_{(\tau)im}\underset{3}{\Omega}_{(\rho)pn}-\underset{3}{q}^{p}_{(\tau)in}\underset{3}{\Omega}_{(\rho)pm}+\underset{3}{r}_{(\rho\tau)im\underset{3}{|}n}-\underset{3}{r}_{(\rho\tau)in\underset{3}{|}m}+\\ &+\sum_{\sigma}(\underset{3}{r}_{(\sigma\tau)im}\underset{3}{\psi}_{(\rho\sigma)n}-\underset{3}{r}_{(\sigma\tau)in}\underset{3}{\psi}_{(\rho\sigma)m})]N^{\alpha}_{(\rho)}.\end{aligned}$$

If we multiply the pervious equation by $a_{\underline{\alpha\beta}}t^{\beta}_{h}$, we have

$$\begin{aligned}&\underset{1}{R}_{\beta\pi mn}t^{\beta}_{h}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}t^{\beta}_{h}+2\Gamma^{p}_{\underset{\vee}{mn}}\underset{3}{q}^{s}_{(\tau)ip}g_{\underline{hs}}=\\ (1.20')\quad &=(\underset{3}{q}^{p}_{(\tau)im\underset{3}{|}n}-\underset{3}{q}^{p}_{(\tau)in\underset{3}{|}m}+\underset{3}{q}^{s}_{(\tau)im}\underset{3}{\Phi}^{p}_{sn}-\underset{3}{q}^{s}_{(\tau)in}\underset{3}{\Phi}^{p}_{sm})g_{\underline{hp}}-\\ &-\sum_{\sigma}e_{(\sigma)}(\underset{3}{r}_{(\sigma\tau)im}\underset{3}{\Omega}_{(\sigma)hn}-\underset{3}{r}_{(\sigma\tau)in}\underset{3}{\Omega}_{(\sigma)hm}).\end{aligned}$$

Multiplying (1.20) by $a_{\underline{\alpha\beta}}N^{\beta}_{(\varphi)}$, we get

$$\begin{aligned}&\underset{1}{R}_{\beta\pi\mu\nu}N^{\beta}_{(\varphi)}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n}-\underset{2}{R}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}N^{\beta}_{(\varphi)}+2\Gamma^{p}_{\underset{\vee}{mn}}\underset{3}{r}_{(\varphi\tau)ip}e_{(\varphi)}=\\ (1.20")\quad &=e_{(\varphi)}[\underset{3}{q}^{p}_{(\tau)im}\underset{3}{\Omega}_{(\varphi)pn}-\underset{3}{q}^{p}_{(\tau)in}\underset{3}{\Omega}_{(\varphi)pm}+\underset{3}{r}_{(\varphi\tau)im\underset{3}{|}n}-\underset{3}{r}_{(\varphi\tau)in\underset{3}{|}m}+\\ &+\sum_{\sigma}(\underset{3}{r}_{(\sigma\tau)im}\underset{3}{\psi}_{(\varphi\sigma)n}-\underset{3}{r}_{(\sigma\tau)in}\underset{3}{\psi}_{(\varphi\sigma)m})].\end{aligned}$$

1.5 By equalizing the right sides in (1.10) and (1.5) for $\mu=\nu=4$, we get *the* *fifth integrability condition of the derivational formula* (1.2):

$$\begin{aligned}
&\underset{2}{R}{}^{\alpha}_{\pi\mu\nu}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n} - \underset{1}{R}{}^{p}_{imn}\lambda^{\alpha}_{(\tau)p} - 2\Gamma^{p}_{\underset{\vee}{mn}}(\underset{4}{q}{}^{s}_{(\tau)ip}t^{\alpha}_{s} + \sum_{\rho}\underset{4}{r}{}_{(\rho\tau)ip}N^{\alpha}_{(\rho)}) = \\
&[\underset{4}{q}{}^{p}_{(\tau)im\underset{4}{|}n} - \underset{4}{q}{}^{p}_{(\tau)in\underset{4}{|}m} + \underset{4}{q}{}^{s}_{(\tau)im}\underset{4}{\Phi}{}^{p}_{sn} - \underset{4}{q}{}^{s}_{(\tau)in}\underset{4}{\Phi}{}^{p}_{sm} - \\
(1.21)\qquad &-\sum_{\sigma}e_{(\sigma)}g^{\underline{ps}}(\underset{4}{r}{}_{(\sigma\tau)im}\underset{4}{\Omega}{}_{(\sigma)sn} - \underset{4}{r}{}_{(\sigma\tau)in}\underset{4}{\Omega}{}_{(\sigma)sm})]t^{\alpha}_{p} + \\
&+\sum_{\rho}[\underset{4}{q}{}^{p}_{(\tau)im}\underset{4}{\Omega}{}_{(\rho)pn} - \underset{4}{q}{}^{p}_{(\tau)in}\underset{4}{\Omega}{}_{(\rho)pm} + \underset{4}{r}{}_{(\rho\tau)im\underset{4}{|}n} - \underset{4}{r}{}_{(\rho\tau)in\underset{4}{|}m} + \\
&+\sum_{\sigma}(\underset{4}{r}{}_{(\sigma\tau)im}\underset{4}{\psi}{}_{(\rho\sigma)n} - \underset{4}{r}{}_{(\sigma\tau)in}\underset{4}{\psi}{}_{(\rho\sigma)m})]N^{\alpha}_{(\rho)}.
\end{aligned}$$

Multiplying the pervious equation by $a_{\underline{\alpha\beta}}t^{\beta}_{h}$, we obtain

$$\begin{aligned}
&\underset{2}{R}{}_{\beta\pi\mu\nu}t^{\beta}_{h}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n} - \underset{1}{R}{}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}t^{\beta}_{h} - 2\Gamma^{p}_{\underset{\vee}{mn}}\underset{4}{q}{}^{s}_{(\tau)ip}g_{\underline{hs}} = \\
(1.21')\qquad &= (\underset{4}{q}{}^{p}_{(\tau)im\underset{4}{|}n} - \underset{4}{q}{}^{p}_{(\tau)in\underset{4}{|}m} + \underset{4}{q}{}^{s}_{(\tau)im}\underset{4}{\Phi}{}^{p}_{sn} - \underset{4}{q}{}^{s}_{(\tau)in}\underset{4}{\Phi}{}^{p}_{sm})g_{\underline{hp}} - \\
&-\sum_{\sigma}e_{(\sigma)}(\underset{4}{r}{}_{(\sigma\tau)im}\underset{4}{\Omega}{}_{(\sigma)hn} - \underset{4}{r}{}_{(\sigma\tau)in}\underset{4}{\Omega}{}_{(\sigma)hm}).
\end{aligned}$$

and multiplying the same equation by $a_{\underline{\alpha\beta}}N^{\beta}_{(\varphi)}$:

$$\begin{aligned}
&\underset{2}{R}{}_{\beta\pi\mu\nu}N^{\beta}_{(\varphi)}\lambda^{\pi}_{(\tau)i}t^{\mu}_{m}t^{\nu}_{n} - \underset{1}{R}{}^{p}_{imn}a_{\underline{\alpha\beta}}\lambda^{\alpha}_{(\tau)p}N^{\beta}_{(\varphi)} - 2\Gamma^{p}_{\underset{\vee}{mn}}\underset{4}{r}{}_{(\varphi\tau)ip}e_{(\varphi)} = \\
(1.21'')\qquad &= e_{(\varphi)}[\underset{4}{q}{}^{p}_{(\tau)im}\underset{4}{\Omega}{}_{(\varphi)pn} - \underset{4}{q}{}^{p}_{(\tau)in}\underset{4}{\Omega}{}_{(\varphi)pm} + \underset{4}{r}{}_{(\varphi\tau)im\underset{4}{|}n} - \underset{4}{r}{}_{(\varphi\tau)in\underset{4}{|}m} + \\
&+\sum_{\sigma}(\underset{4}{r}{}_{(\sigma\tau)im}\underset{4}{\psi}{}_{(\varphi\sigma)n} - \underset{4}{r}{}_{(\sigma\tau)in}\underset{4}{\psi}{}_{(\varphi\sigma)m})].
\end{aligned}$$

1.6 Finally, if we equilaize the right sides in (1.11) and (1.5) for $\mu = 3\ \nu = 4$, we obtain *the sixth integrability condition of the derivational formula* (1.2):

$$\begin{aligned}
&\underset{4}{R}{}^{\alpha}_{\pi mn}\lambda^{\pi}_{(\tau)i} + \underset{3}{R}{}^{p}_{inm}\lambda^{\alpha}_{(\tau)p} = \\
&[\underset{3}{q}{}^{p}_{(\tau)im\underset{4}{|}n} - \underset{4}{q}{}^{p}_{(\tau)in\underset{3}{|}m} + \underset{3}{q}{}^{s}_{(\tau)im}\underset{4}{\Phi}{}^{p}_{sn} - \underset{4}{q}{}^{s}_{(\tau)in}\underset{3}{\Phi}{}^{p}_{sm} - \\
(1.22)\qquad &-\sum_{\sigma}e_{\sigma}g^{\underline{ps}}(\underset{3}{r}{}_{(\sigma\tau)im}\underset{4}{\Omega}{}_{(\sigma)sn} - \underset{4}{r}{}_{(\sigma\tau)in}\underset{3}{\Omega}{}_{(\sigma)sm})]t^{\alpha}_{p} + \\
&+\sum_{\rho}[\underset{3}{q}{}^{p}_{(\tau)im}\underset{4}{\Omega}{}_{(\rho)pn} - \underset{4}{q}{}^{p}_{(\tau)in}\underset{3}{\Omega}{}_{(\rho)pm} + \underset{3}{r}{}_{(\rho\tau)im\underset{4}{|}n} - \underset{4}{r}{}_{(\rho\tau)in\underset{3}{|}m} + \\
&+\sum_{\sigma}(\underset{3}{r}{}_{(\sigma\tau)im}\underset{4}{\psi}{}_{(\rho\sigma)n} - \underset{4}{r}{}_{(\sigma\tau)in}\underset{3}{\psi}{}_{(\rho\sigma)m})]N^{\alpha}_{(\rho)}.
\end{aligned}$$

If we multiply this equaton by $a_{\underline{\alpha\beta}} t^{\beta}_{h}$, we obtain

(1.22')
$$\begin{aligned} &\underset{4}{R}_{\beta\pi mn} t^{\beta}_{h} \lambda^{\pi}_{(\tau)i} + \underset{3}{R}{}^{p}_{inm} a_{\underline{\alpha\beta}} \lambda^{\alpha}_{(\tau)p} t^{\beta}_{h} = \\ &= (\underset{3}{q}{}^{p}_{(\tau)im\underset{4}{|}n} - \underset{4}{q}{}^{p}_{(\tau)in\underset{3}{|}m} + \underset{3}{q}{}^{s}_{(\tau)im} \underset{4}{\Phi}{}^{p}_{sn} - \underset{4}{q}{}^{s}_{(\tau)in} \underset{3}{\Phi}{}^{p}_{sm}) g_{\underline{hp}} - \\ &- \sum_{\sigma} e_{(\sigma)} (\underset{3}{r}_{(\sigma\tau)im} \underset{4}{\Omega}_{(\sigma)hn} - \underset{4}{r}_{(\sigma\tau)in} \underset{3}{\Omega}_{(\sigma)hm}). \end{aligned}$$

where $\underset{4}{R}_{\beta\pi mn} = a_{\underline{\alpha\beta}} \underset{4}{R}{}^{\alpha}_{\pi mn}$ and $\underset{4}{R}{}^{\alpha}_{\pi mn}$ is given in relation to (1.16).

Multiplying (1.22) by $a_{\underline{\alpha\beta}} N^{\beta}_{(\varphi)}$, we have

(1.22")
$$\begin{aligned} &\underset{4}{R}_{\beta\pi mn} N^{\beta}_{(\varphi)} \lambda^{\pi}_{(\tau)i} + \underset{3}{R}{}^{p}_{inm} a_{\underline{\alpha\beta}} \lambda^{\alpha}_{(\tau)p} N^{\beta}_{(\varphi)} = \\ &= e_{(\varphi)} [\underset{3}{q}{}^{p}_{(\tau)im} \underset{4}{\Omega}_{(\varphi)pn} - \underset{4}{q}{}^{p}_{(\tau)in} \underset{3}{\Omega}_{(\varphi)pm} + \underset{3}{r}_{(\varphi\tau)im\underset{4}{|}n} - \underset{4}{r}_{(\varphi\tau)in\underset{3}{|}m} + \\ &+ \sum_{\sigma} (\underset{3}{r}_{(\sigma\tau)im} \underset{4}{\psi}_{(\varphi\sigma)n} - \underset{4}{r}_{(\sigma\tau)in} \underset{3}{\psi}_{(\varphi\sigma)m})]. \end{aligned}$$

2. Some special cases

For some fixed values of coeffitients b, c we obtain from (1.1) special cases, some of which are very importtant.

2.1. If

(2.1)
$$b^{s}_{(\tau)i} = \delta^{s}_{i} \quad \& \quad c_{(\rho\tau)i} = 0, \quad \text{then} \quad \lambda^{\alpha}_{(\tau)i} = t^{\alpha}_{i} = y^{\alpha}_{,i}.$$

In this case from (1.3) we obtain

$$\underset{\mu}{q}{}^{p}_{(\tau)im} = \delta^{s}_{i\underset{\mu}{|}m} + \delta^{s}_{i} \underset{\mu}{\Phi}{}^{p}_{sm} = \underset{\mu}{\Phi}{}^{p}_{im}, \quad \underset{\mu}{r}_{(\rho\tau)im} = \delta^{s}_{i} \underset{\mu}{\Omega}_{(\rho)sm} = \underset{\mu}{\Omega}_{(\rho)im},$$

and (1.2) gives (0.9a) i.e. the first derivational formula of a subspace of a generalized Riemannian space. In this case integrability conditions of the first derivational formula of the field (1.1) reduce to integrability conditions of the first derivational formula, from which we obtain several equations of the Gauss-Codazzi type. For example, in this case (1.19') becomes

(2.2)
$$\begin{aligned} &\underset{3}{R}_{\beta\pi mn} t^{\beta}_{h} t^{\pi}_{i} - \underset{3}{R}_{himn} = \\ &= (\underset{1}{\Phi}{}^{p}_{im\underset{2}{|}n} - \underset{2}{\Phi}{}^{p}_{in\underset{1}{|}m} + \underset{1}{\Phi}{}^{s}_{im} \underset{2}{\Phi}{}^{p}_{sn} - \underset{2}{\Phi}{}^{s}_{in} \underset{1}{\Phi}{}^{p}_{sn}) g_{\underline{hp}} - \\ &- \sum_{\sigma} e_{(\sigma)} (\underset{1}{\Omega}_{(\sigma)im} \underset{2}{\Omega}_{(\sigma)hn} - \underset{2}{\Omega}_{(\sigma)in} \underset{1}{\Omega}_{(\sigma)hm}), \end{aligned}$$

and this is the third kind of the Gauss equation of the subspace GR_M. Now from (1.19") we obtain

$$\begin{aligned}&\underset{3}{R}_{\beta\pi mn} N^{\beta}_{(\varphi)} t^{\pi}_{i} = \\ (2.3)\qquad &= e_{(\varphi)}[\underset{1}{\Phi}{}^{p}_{im}\underset{2}{\Omega}_{(\varphi)pn} - \underset{2}{\Phi}{}^{p}_{in}\underset{1}{\Omega}_{(\varphi)pm} + \underset{1}{\Omega}_{\varphi)im\underset{2}{|}n} - \underset{2}{\Omega}_{\varphi in\underset{1}{|}m} + \\ &+\sum_{\sigma}(\underset{1}{\Omega}_{(\sigma)im}\underset{2}{\psi}_{(\varphi\sigma)n} - \underset{2}{\Omega}_{(\sigma)in}\underset{1}{\psi}_{(\varphi\sigma)m})].\end{aligned}$$

and this is the first Codazzi eqation of the third kind.

2.2. The next special case is

$$(2.4)\qquad b^{s}_{(\tau)i} = 0 \quad \& \quad c_{(\rho\tau)i} = \delta_{\rho\tau} \quad \Rightarrow \quad \lambda^{\alpha}_{(\tau)i} = N^{\alpha}_{(\tau)}.$$

Now, it is

$$\underset{\mu}{q}{}^{p}_{(\tau)im} = -\sum_{\rho} e_{(\rho)}\delta_{\rho\tau} g^{\underline{ps}}\underset{\mu}{\Omega}_{(\rho)sm} = -g^{\underline{ps}}\underset{\mu}{\Omega}_{(\tau)sm}e_{(\tau)}$$

$$\underset{\mu}{r}_{(\rho\tau)im} = \sum_{\sigma}\delta_{\sigma\tau}\underset{\mu}{\psi}_{(\rho\sigma)m} = \underset{\mu}{\psi}_{(\rho\tau)m},$$

and (1.2) results in (0.9b), i.e. in the second derivational formula of the subspace . In this case from (1.19') one obtains the equation which is equivalent to (2.3). The equation (1.19") in this case becomes

$$\begin{aligned}\underset{3}{R}_{\beta\pi mn} N^{\beta}_{(\varphi)} N^{\pi}_{(\tau)} &= e_{(\varphi)}[-g^{\underline{ps}}\underset{1}{\Omega}_{(\tau)sm}e_{(\tau)}\underset{2}{\Omega}_{(\varphi)pn} + g^{\underline{ps}}\underset{2}{\Omega}_{(\tau)sn}e_{(\tau)}\underset{1}{\Omega}_{(\varphi)pm} + \\ &+\underset{1}{\psi}_{(\varphi\tau)m\underset{2}{|}n} - \underset{2}{\psi}_{(\varphi\tau)n\underset{1}{|}m} + \sum_{\sigma}(\underset{1}{\psi}_{(\sigma\tau)m}\underset{2}{\psi}_{(\varphi\sigma)n} - \underset{2}{\psi}_{(\sigma\tau)n}\underset{1}{\psi}_{(\varphi\sigma)m})]\end{aligned}$$

and this is the second Codazzi equation of the third kind.

2.3. Curvature vector of a curve C in a subspace GR_M of a space GR_N is determined in the same way [6], as in the usual Riemannian space, i.e.

$$(2.5)\qquad q^{\alpha} = t^{\alpha}_{i}p^{i} + \sum_{\rho}K_{(\rho)}N^{\alpha}_{(\rho)}$$

where $K_{(\rho)}$ is normal curvature of C, which corresponds to the normal $N^{\alpha}_{(\rho)}$. Now from (1.1) one obtains

$$(2.6)\qquad b^{s}_{(\tau)i} = p^{s} \quad \& \quad c_{(\rho\tau)i} = K_{(\rho)} \quad \Rightarrow \quad \lambda^{\alpha}_{(\tau)i} = q^{\alpha}.$$

So the field q^{α} is a special case of the field $\lambda^{\alpha}_{(\tau)i}$ (1.1). Therefore, from integrability conditions of derivational formula of the field $\lambda^{\alpha}_{(\tau)i}$ one obtains corensponding equations for q^{α}.

References

[1] L. P. Eisenhart, *Generalized Riemannian spaces I*, Proc. Nat. Acad. Sci. USA, 37(1951), 311-315.

[2] L. P. Eisenhart, *Generalized Riemannian spaces II*, Proc. Nat. Acad. Sci. USA, 38(1952), 505-508

[3] S. M. Minčić, *Ricci type identities in a subspace of a space of non-summetric affine connexion*, Publ. Inst. Math. (Beograd), 18(32)(1975), 137-148.

[4] С.М. Минчич, *Новые тождества типа Риччи в подпространстве пространства несимметричной аффинной связаности*, Известия ВУЗ, Математика, 4(203) (1979), 17-27.

[5] S. M. Minčić, *Derivational formulas of a subspace of a generalized Riemannian space*, Publ. Inst. Math. (Beograd), 34(48)(1983), 125-135.

[6] С.М. Минчич, *О векторе кривизны кривой в подпространстве обобщенного риманова пространства*,
Facta Universitatis, Ser. Math. Inform 2(1987), 75-89.

[7] R. S. Mishra, *Subspaces of a generalized Riemannian space*, Bull. Acad. Roy. Belgique, 1954, 1058-1071.

[8] M. Prvanović, *Equations de Gaus d'un sous-espace plongé dans l'espace Riemannien generalisé* , Bull. Acad. Roy. Belgique, 1955, 615-621.

Department of Mathematics, Faculty of Philosophy, Ćirila i Metodija 2, 18 000 Niš, Yugoslavia.

FILOMAT (Niš) 9:2 (1995), 161–167
Filomat '94, Niš, October 22–24, 1994
Geometry. Computer Sciences

$F(2k+1,1)$-STRUCTURE ON THE LAGRANGIAN SPACE

Jovanka Nikić

ABSTRACT. *If almost product P or almost complex structure J on the tangent space $T(E)=T_V(E)+T_H(E)$ of Lagrangian $2n$ dimensional manifold E are defined, and if $f_v(2k+1,1)$-structure on $T_V(E)$ is defined, then $f_p(2k+1,1)$ and $f_j(2k+1,1)$-structures on $T_H(E)$ are defined in the natural way. We can define $F_p(2k+1,1)$, $F_j(2k+1,1)$-structures on $T(E)$. The condition is given for the reduction of the structural group of such manifolds.*

1. Introduction

Let M be an n dimensional and E $2n$ dimensional differentiable manifold and let $\eta=(E,\pi,M)$ be vector bundles and $\pi E=M$. The differential structures (U,ϕ,R^{2n}) are vector charts of the vector bundles η. Hence the canonical coordinates on $\pi^{-1}(U)$ are $(x^1,\dots,x^n,y^1,\dots,y^n)=(x^i,y^a)$, $i=1,2,\dots,n$ $a=1,\dots,n$. Transformation maps on E are

$$x^{i'}=x^{i'}(x^1,x^2,\dots,x^n)$$
$$y^{a'}=M_a^{a'}(x^1,\dots,x^n)y^a=M_a^{a'}(x^i)y^a$$
$$\operatorname{rank}\left[\frac{\partial x^{i'}}{\partial x^i}\right]=n,\quad \operatorname{rank}\left[\frac{\partial y^{a'}}{\partial y^a}\right]=\operatorname{rank} M_a^{a'}=n.$$

The inverse transformations are

$$x^i=x^i(x^{1'},x^{2'},\dots,x^{n'})$$
$$y^a=M_{a'}^a(x^{i'},\dots,x^{n'})y^{a'}$$

where $M_{a'}^a M_a^{a'}=\delta_b^a$.

1991 *Mathematics Subject Classification.* 53B40, 53C60.

The local natural bases of the tangent space $T(E)$ are $\{\partial_i, \partial_a\}$

$$\partial_a = \frac{\partial}{\partial y^a} = M_a^{a'}(x^i)\partial_{a'}$$
$$\partial_i = \frac{\partial}{\partial x^i} = \frac{\partial x^{i'}}{\partial x^i}\partial_i + (\partial_i M_b^{a'}(x))y^b\partial_{a'}.$$

The nonlinear connection on E is distribution

$$N : u \in E \to N_u \subset T_u(E)$$

which is supplementary to the distribution V,

(1.1) $$T_u(E) = N_u \oplus V_u, \quad \forall_u \in E.$$

They are localy determined by $\delta_i = \partial_i - N_i^a\partial_a$. The local bases adapted to decompositions in (1.1) is $\{\delta_i, \partial_a\}$.

It is easy to prove that on $\{\delta_i, \partial_a\}$

$$\delta_{i'} = \delta_i \frac{\partial x^i}{\partial x^{i'}}, \quad \partial_{a'} = \frac{\partial y^a}{\partial y^{a'}}\partial_a.$$

The subspace of $T(E)$ spaned by $\{\delta_i\}$ will be denoted by $T_H(E)$ and the subspace spaned by $\{\partial_a\}$ will be denoted by $T_V(E)$, $T(E) = T_H(E) \oplus T_V(E)$, $\dim T_H(E) = n = \dim T_V(E)$.

Definition 1.1. *If the Riemannian metrical structure on $T(E)$ is given by $G = g_{ij}(x^i, y^a)dx^i \otimes dx^j + g_{ab}(x^i, y^a)\delta y^a \otimes \delta y^b$ where $g_{ij}(x^i, y^a) = g_{ij}(x^i)$, $g_{ab} = \frac{1}{2}\partial_a\partial_b L(x^i, y^a)$ and $L(x^i, y^a)$ is a Lagrange function, then such a space we call Lagrangian space.*

Let $X \in T(E)$, then $X = X^i\delta_i + \bar{X}^a\partial_a$ and the automorphism $P : \mathcal{X}(T(E)) \to \mathcal{X}(T(E))$ defined by

$$PX = \bar{X}^i\delta_i + X^a\partial_a$$

is the natural almost product structure on $T(E)$. i.e, $P^2 = I$. If we denote by v and h the projection morphism of $T(E)$ to $T_V(E)$ and $T_H(E)$ respectively, we have

$$P \circ h = v \circ P.$$

The automorphism

$$JX = -\bar{X}^i\delta_i + X^a\partial_a$$

is the natural almost complex structure on $T(E)$.

2. $f(2k+1,1)$-structures

Definition 2.1. *We call Lagrange vertical $f_v(2k+1,1)$-structure of rank r on $T_V(E)$ a non-null tensor field f_v of type (1,1) and of class C^∞ such that $f_v^{2k+1}+f_v=0$, $k\in N$, and rank $f_v=r$, where r is constant everywhere.*

Definition 2.2. *We call Lagrange horizontal $f_h(2k+1,1)$-structure on $T_H(E)$ a non-null tensor field f_h on $T_H(E)$ of type (1,1) of class C^∞ satisfying $f_h^{2k+1}+f_h=0$, $k\in N$, rank $f_h=r$, where r is constant everywhere.*

An $F(2k+1,1)$-structure on $T(E)$ is a non-null tensor field F of type $\binom{11}{11}$ such that $F^{2k+1}+F=0$, $k\in N$, rank $F=2r=$const.

For our study it is very convenient to consider f_v or f_h as morphism of vectors bundles.

$$f_v:\mathcal{X}T_V(E)\to\mathcal{X}T_V(E)$$
$$f_h:\mathcal{X}T_H(E)\to\mathcal{X}T_H(E).$$

Let f_v be a Lagrange vertical $f_v(2k+1,1)$-structure of rank r. We define the morphisms

$$l=-f_v^{2k}\ \text{ and }\ m=f_v^{2k}+I_{T_V(E)}$$

where $I_{T_V(E)}$ denotes the identity morphism on $T_V(E)$.

It is clear thet $l+m=I$. Also we have

$$l\,m=m\,l=-f_v^{4k}-f_v^{2k}=-f_v^{2k-1}(f_v^{2k+1}+f_v)=0,$$
$$m^2=m,\quad l^2,=l.$$

Hence the morphisms l, m applied to the $\mathcal{X}(T_V(E))$ are complementaly projection morphisms, then there exist complementary distributions VL and VM corresponding to the projection morphisms l and m respectively such that dim $VL=r$ and dim $VM=n-r$.

It is easily to see that

$$lf_v=f_vl=f_v,\quad mf_v=f_vm=0,\quad f_v^{2k}m=0, \tag{2.1}$$

$$f_v^{2k}l=-l.$$

Proposition 2.1. *If a Lagrange $f_v(2k+1,1)$-structure of rank r defined on $T_V(E)$, then the horizontal $f_h(2k+1,1)$-structure of rank r is defined on $T_H(E)$ by the natural almost product structure of $T(E)$, as f_p, or by the almost complex natural complex structure of $T(E)$, as f_j.*

Proof. If we put

$$f_p X = P f_v P X, \ \ \forall X \in T_H(E) \tag{2.2}$$

$$f_j X = -J f_v J X, \ \ \forall X \in T_H(E) \tag{2.3}$$

it is easy to see that

$$f_p^{2k+1} X = P f_v^{2k+1} P X, \ \ f_j^{2k+1} X = -J f_v^{2k+1} J X$$

and

$$f_p^{2k+1} + f_p = 0, \ \ f_j^{2k+1} + f_j = 0$$

and rank f_p = rank $f_j = r$. It is easy to see that $f_p = f_j = f_h$.

Proposition 2.2. *If a Lagrange $f_v(2k+1,1)$-structure of rank r is defined on $T_V(E)$, then an $F_p(2k+1,1)$-structure or $F_j(2k+1,1)$-structure are defined on $T(E)$ by the natural almost product or natural almost complex structure of $T(E)$.*

Proof. We put

$$F_p = f_p h + f_v v,$$
$$F_j = f_j h + f_v v,$$

where f_p, f_j are defined by (2.2), (2.3) and h, v are the projection morphisms of $T(E)$ to $T_H(E)$ and $T_V(E)$. Then it is easy to check that

$$F_p^2 = f_p^2 h + f_v^2 v, \ \ F_p^{2k+1} = f_p^{2k+1} h + f_v^{2k+1} v.$$

Thus $F_p^{2k+1} + F_p = 0$. Similary $F_j^{2k+1} + F_j = 0$. It is clear that rank F_p =rank$F_j = 2r$.

If l_p, m_p are complementary projection morphisms of the horizontal $f_p(2k+1,1)$-structure, which is defined by the natural almost product structure of $T(E)$, we have

$$l_p X = -f_p^{2k} X = -P f_v^{2k} P X = P l P X, \forall X \in T_H(E)$$

$$m_p X = f_p^{2k} + I_{T_V(E)} X = P f_v^{2k} P X + P I_{T_V(E)} P X = P m P X, \forall X \in T_H(E).$$

If L_p, M_p are complementary projection morphism of the $F_p(2k+1,1)$ structure on $T(E)$, then we have

$$\begin{aligned} L_p = -F_p^{2k} &= -f_p^{2k} h - f_v^{2k} v = l_p h + l v \\ M_p = F_p^{2k} + I_{T(E)} &= f_p^{2k} h + f_v^{2k} + I_{T_H(E)} h + I_{T_V(E)} v = \\ &= m_p h + m v. \end{aligned} \tag{2.4}$$

Thus, if there is given a Lagrange $f_v(2k+1,1)$-structure on $T_V(E)$ of rank r, then there exist complementary distributions HL_p, HM_p of $T_H(E)$, corresponding to the morphisms l_p, m_p such that

$$HL_p = PVL, HM_p = PVM. \tag{2.5}$$

Thus we have the decompositions

$$T(E) = T_H(E) \oplus T_V(E) = PVL \oplus PVM \oplus VL \oplus VM.$$

If TL_p, TM_p denote complementary distributions corresponding to the morphisms L_p, M_p respectively, then from (2.4) and (2.5) we have

$$TL_p = PVL \oplus VL \quad , \quad TM_p = PVM \oplus VM.$$

Let $\bar{g}$ is a pseudo-Riemannian metric tensor, which is symmetric, bilinear and non-degenerate on $T_V(E)$.

$$\bar{g} : \mathcal{X}(T_V(E)) \times \mathcal{X}(T_V(E)) \to \mathcal{F}(T(E)).$$

(for examples $\bar{g}$ can be the vertical part of Lagrange metric structure).

The mapping

$$a : \mathcal{X}(T_V(E)) \times \mathcal{X}(T_V(E) \to \mathcal{F}(T(E))$$

which is defined by

$$a(X,Y) = \frac{1}{2}[\bar{g}(lX,lY) + \bar{g}(mX,mY)] \;\; \forall X,Y \in \mathcal{X}T_V(E)$$

is a pseudo-Riemannian structure on $T(E)$ such that $a(X,Y)=0$, $\forall X \in \mathcal{X}(T(VL))$, $Y \in \mathcal{X}(T(VM))$.

Theorem 2.1. *If a Lagrange $f_v(2k+1,1)$-structure $k \geq 1$ of rank r is defined on $T_V(E)$ then there exist a pseudo-Riemannian structure of $T_V(E)$ with respect to the complementary distributions VL and VM are orthogonal and f_v is an isometry on $T_V(E)$.*

Proof. If we put

$$g(X,Y) = \frac{1}{2k}[a(X,Y) + a(f_vX, f_vY) + \cdots + a(f_v^{2k-1}X, f_v^{2k-1}Y)]$$

it is easy to see that

$$g(X,Y) = 0 \quad \forall X \in \mathcal{X}(VL), \quad Y \in \mathcal{X}(VM).$$

Using (2.1) we get

$$g(f_vX, f_vY) = \frac{1}{2k}[a(f_vX, f_vY) + a(f_v^2X, f_v^2Y) + \cdots + a(X,Y)].$$

Thus f_v is an isometry with respect to g.

Let $X \in \mathcal{X}(T(VL))$ then $f_v X, f_v^2 X, \dots, f^{2k} X \in \mathcal{X}(T(VL))$ and

$$g(X, f_v^k X) = g(f_v X, f_v^{k+1} X) = \dots = g(f_v^k X, f_v^{2k} X) = -g(f_v^k X, X).$$

Consequently

$$g(X, f_v^k X) = g(f_v X, f_v^{k+1} X) = \dots = g(f_v^k X, f_v^{2k} X) = 0$$

and $r = 2km$.

Thus we can chose in $\mathcal{X}(T(VL))$ $r = 2km$ mutualy orthogonal unit vector fields such that

$$\begin{aligned} &f(X_a) = X_{\alpha+m} && \alpha = 1, 2, \dots, (2k-1)m, \\ &f(X_\alpha) = -X_{-(2k-1)m+\alpha}, && \alpha = (2k-1)m + 1, \dots, 2km. \end{aligned}$$

An adapted frame of the Lagrange $f_v(2k+1,1)$-structure on $T_V(E)$ is the orthogonal frame $R = \{X_\alpha, X_\beta\}$, where X_β is an orthogonal frame of $\mathcal{X}(T(VM))$.

Let $\bar{R} = \{\bar{X}_\alpha, \bar{X}_\beta\}$ be another adapted frame of the Lagrange $f_v(2k+1,1)$-structure, and $\bar{R} = AR$, then orthogonal matrix A is an element of the group $U_{(km)} \times O_{(n-2km)}$.

Theorem 2.2. *A necessary and suficient condition for $T_V(E)$ to admit Lagrange $f_v(2k+1,1)$-structure, $k \geq 1$ of rank r is that $r = 2km$ and the structure group of the tangent bundle of the manifold be reduced to the group $U_{(km)} \times O_{(n-2km)}$.*

We can define a maping g_p:

$$g_p(X,Y) = g(PX, PY), \quad \forall X, Y \in \mathcal{X}(T_H(E))$$

g_p is a metric structure on $T_H(E)$. Using (2.5) the distributions HL_p, HM_p are orthogonal with respect to g_p and the horizontal $f_p(2k+1,1)$-structure which is define by $f_p X = P f_v P X, \forall X \in \mathcal{X}(T_H(E))$ is an isometry on $T_H(E)$ with respect to g_p.

Proposition 2.3. *If $\{X_\alpha, X_\beta\}$ is an adapted frame of a given Lagrange $f_v(2k+1,1)$-structure f_v on $T_V(E)$ with respect to g, then the frame $\{PX_\alpha, PX_\beta\}$ is an adapted frame of the horizontal $f_p(2k+1,1)$-structure with respect to g_p.*

It is clear that the frames $\{PX_\alpha, PX_\beta, X_\alpha, X_\beta\}$ are adapted frames to the decomposition

$$T(E) = HL_p \oplus HM_p \oplus VL \oplus VM.$$

Theorem 2.3. *If a Lagrange $f_v(2k+1,1)$-structure is defined on $T_V(E)$ with pseudo-Riemannian structure g, then the structure group of the tangent bundle on $T(E)$ be reduced to $U_{(km)} \times O_{(n-2km)} \times U_{(km)} \times O_{(n-2km)}$.*

Refrences

[1] M. Atanasiu, *Modes in Finsler and Lagrange geometry*, Proc. IVth Nat. Sem. on Finsler and Lagrange geometry, Brasov, (1986), 43–56.

[2] F. G. Andreou, *On a structure defined by a tensor field f of type (1,1) satisfying* $f^5 + f = 0$, Tensor, N.S. **36**(1982), 79–84.

[3] S. Ishihara, K. Yano, *On integrability conditions of a structure satisfying* $f^3 + f = 0$, Quart-J. Math. **15**(1964), 217–222.

[4] J. Nikić, I. Čomić, $f(2 \cdot 2^k + 1, -1)$*-structure in* $(k+1)$*-Lagrangian Space*, Review of Research Faculty of Science, Mathematics Series, (toappear).

[5] J. Nikić, *On a structure defined by a tensor field f of the type (1,1) satisfying* $f^{2\cdot 2k+1} - f = 0$, Review of Research Faculty of Science-University of Novi Sad, **12**(1982), 369–377.

Faculty of Technical Sciences, University of Novi Sad, Trg D. Obradovića 6, 21000 Novi Sad, Yugoslavia

FILOMAT (Niš) 9:2(1995), 169–185
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ON WARPED PRODUCT MANIFOLDS

Mileva Prvanović

ABSTRACT. *This is a survey article on warped product manifolds and contains: applications in some relativistic theories (Schwarzschild spacetime and Robertson-Walker spacetime), subprojective spaces, the invariant way characterizing warped products and the geometry of warped product in terms of warping function and the geometies of the base and the fiber.*

1. Definition and the first example

Let $(\overline{M}, \overline{g})$ and $(\overset{*}{M}, \overset{*}{g})$ be two Riemannian manifolds such that $dim\,\overline{M} = q$, $dim\,\overset{*}{M} = n - q$, $1 < q < n$. Let F be a positive C^∞ function on $\overline{M}$.

Definition. *([23],p.204). The warped product $M = \overline{M} \times_F \overset{*}{M}$ of $(\overline{M}, \overline{g})$ and $(\overset{*}{M}, (\overset{*}{g})$ is the manifold $M = \overline{M} \times \overset{*}{M}$ whith the metric $g = \overline{g} \times_F \overset{*}{g}$. More precisely*

$$g = \overline{g} \times F\overset{*}{g} = \pi_1^*\overline{g} + (F \circ \pi_1)\pi_2^*\overset{*}{g},$$

where $\pi_1 : \overline{M} \times \overset{*}{M} \to \overline{M}$, $\pi_2 : \overline{M} \times \overset{*}{M} \to \overset{*}{M}$ are natural projections. The manifold $\overline{M}$ is caled the base manifold, while $\overset{*}{M}$ is the fiber.

For each $(\overline{m}, \overset{*}{m}) \in M$ the subset $\overline{M} \times \overset{*}{m}$ is a totally geodesic submanifold of warped product and all such submanifolds are isometrically related; the submanifolds $\overline{m} \times \overset{*}{M}$ are totally umbilic and the map $\pi_2|_{\overline{m} \times \overset{*}{M}}$ is a positive homotety onto $\overset{*}{M}$ scale factor $\dfrac{1}{F(\overline{m})}$. For each $(\overline{m}, \overset{*}{m}) \in M$, the submanifolds $\overline{M} \times \overset{*}{m}$ and $\overline{m} \times \overset{*}{M}$ are orthogonal at $(\overline{m}, \overset{*}{m})$. The converse is also true, that is we have the following theorem:

Theorem. *([23], [44]). A Riemannian space is a warped product manifold if and only if it can be decomposed into two families of mutualy orthogonal submanifolds, one family consisting of totally geodesic and the other of totally umbilic submanifolds.*

If $F = const.$ then F can be incorporate in the metric $\overset{*}{g}$ and $M = \overline{M} \times \overset{*}{M}$ reduces to a productmanifold, both $\overline{M} \times \overset{*}{m}$ and $\overline{m} \times \overset{*}{M}$ being totally geodesic.

Thus, the class of warped products contains the class of product manifolds and is its generalization.

The class of warped product contains all Riemannian manifolds of constant curvature. In fact, for each point of such a manifold there exsist a neighbourhood in which, with respect to the polar coordinats $r, \phi^1, ..., \phi^{n-1}$ the metric is

$$ds^2 = dr^2 + sin^2\sqrt{k}r\, d\overset{*}{s}^2(\phi^1, ..., \phi^{n-1}) \text{ for } k > 0$$

and

$$ds^2 = dr^2 + sin^2\sqrt{-k}r\, d\overset{*}{s}^2(\phi^1, ..., \phi^{n-1}) \text{ for } k < 0,$$

where $k \neq 0$ is the constant curvature of M and $d\overset{*}{s}^2$ is the metric of the unit $(n-1)$–dimensional sphere S^{n-1}. We note that the manifold of constant curvature $k \neq 0$ can not be a product manifold. If $k = 0$, then the manifold can be represented as a product manifold on many ways. But for example, for $R^3\backslash 0$, with respect to the spherical coordinates, we have

$$ds^2 = dr^2 + r^2(d\theta^2 + sin^2\theta\, d\varphi^2).$$

It means that $R^3\backslash 0$ can be identified with warped $R^+ \times_r S^2$ with a ray from the origin as a basis and the spheres $S^2(r)$, $r > 0$ as the fibers.

The surface of revolution is an other example of Warped products. Let C be the curve in R^3 whose parametric representation is

$$x = g(u), \qquad y = 0, \qquad z = F(u).$$

If z is the axis of revolution and v is the angle of rotation, then we have

$$ds^2 = [(g'(u))^2 + (F')^2]du^2 + F^2dv^2.$$

Thus, the surface of revolution is warped product $C \times_F S^1$, where the curve C is the basis manifold and the circles of revolution the fibers.

The four dimensional warped products are very important in the construction of simple models of some relativistic theories.Thus Schwartzschild spacetime is the simplest relativistic model of a universe containing a single star. The star is assumed to be static and spherically symmetric and to be the only source of gravitation for the spacetime. It follows from these assumptions that Schwartzschild spacetime is warped product $P \times_r S^2$, where the fiber S^2 is the unit sphere and the base space $P = R \times R^+$ is a half-plane $r > 0$ in the rt-plane endowed with the metric

$$-(1 - \frac{2m}{r})dt^2 + (1 - \frac{2m}{r})^{-1}dr^2, \tag{1.1}$$

where m is a constant identified with the mass of the star. The function $1 - \dfrac{2m}{r}$ increases from limit $-\infty$ at $r = 0$ toward limit 1 at $r = \infty$. But $1 - \dfrac{2m}{r} = 0$

at $r = 2m$, that is the metric (1.1), and therefore the metric of Warped product $M = P \times_r S^2$, degenerates at $r = 2m$. So, we have to consider two Schwartzschild spacetimes:

1) Schwartzschild exterior spacetime $M = P_I \times_r S^2$, where P_I is the region $r > 2m$;

2) Schwartzschild black hole $M = P_{II} \times_r S^2$, where P_{II} is the region $0 < r < 2m$.

The star is characterized by its mass m and its radius R. For the spacetime around the star we have $r > R$. For an ordinary star we have $R > 2m$, that is the surface of the star is in Schwarzschild exterior spacetime. But if $R < 2m$, then R can be only 0; the star dissapears and the warped product $P_{II} \times_r S^2$ becomes black hole. ([23],Chapter 13).

According to the astronomical evidences, the universe can be modeled as a spacetime containing a perfect fluid whose "molecules" are the galaxies. Also, the galaxies, taking into account the large scale appropiate to cosmology, appear to be distributed the same in all directions. Starting with this isotropy condition and using the physical assumptions about the galactic flow, it is possible to construct a simple cosmological model, so called Robertson-Waker spacetime ([23]). This model is the warpedproduct

$$M = M(k, F) = I \times_F S, \tag{1.2}$$

where I is an open interval in R^1 and S is a three-dimensional manifold of constant curvature $k = -1$, 0 or 1. The metric of the manifold (1.2) is

$$ds^2 = -(dt)^2 + F\,d\overset{*}{s}{}^2,$$

where $d\overset{*}{s}{}^2$ is the metric of the mabifold S. It can be proved that the Ricci curvature for Robertson-Walker spacetime $M(k, F)$ with flow vector field $\mathcal{U} = \partial_t$ is given by

$$Ric(\mathcal{U}, \mathcal{U}) = -\frac{3F''}{F}, \qquad Ric(\mathcal{U}, \mathcal{X}) = 0,$$
$$Ric(\mathcal{X}, \mathcal{Y}) = \left[2\left(\frac{F'}{F}\right) + \frac{2k}{F^2} + \frac{F''}{F}\right] < \mathcal{X}, \mathcal{Y} > \text{ if } \mathcal{X}, \mathcal{Y} \perp \mathcal{U}$$

([23], p.345).

Also, if $\mathcal{U}$ is the flow vector field a Robertson-Walker spacetime $M(k, F)$, then $(\mathcal{U}, p, \rho)$ is a perfect fluid with energy density ρ and the pressure p given by

$$\frac{8\pi}{3}\rho = \left(\frac{F'}{F}\right) + \frac{k}{F^2}, \quad -8\pi p = 2\frac{F''}{F} + \left(\frac{F'}{F}\right) + \frac{k}{F^2}$$

(see [23],p.345).

Acording the astronomical estimates, the spaces $S(t)$ are expanding, i.e. currently F has positive derivative. The following theorem considers the past and the future.

Theorem. *([23],p.348). Let* $m(k, F) = I \times_F SandH(t) = \frac{F'(t)}{F(t)}$*. If* $H_0 = H_0(t_0) > 0$ *for some* T_0*, and* $\rho + 3p > 0$*, then* I *has an initial endpoint* t_* *with*

$$t_0 - H_0^{-1} < t_* n < t_0$$

and eighter (1) $F' > 0$ *or (2)* F *has a maximum point after* t_0 *and* I *is a finite interval* (t_*, t_{**}).

It means thet the universe had the definite beginning and eighter continues expanding, or after conctracting for a while, comes to an end. Using some additional dates, it can be concluded that our universe began in a colossal explosoin.

2. Subprojective spaces

The warped product appears also in the investigations of the subprojective and generalized subprojective spaces.

The subprojective spaces were first defined and investigated by V.F.Kagan ([20],-[21],[39]). With respect to the projective properties, these spaces are a natural generalization of the Riemannian spaces of constant curvature. Namely, according the well known Beltrami's theorem, the spaces of constant curvature and only such spaces, admit a mapping on an euclidean space such that the geodesics corenspond to the streit lines. But if the space allows mapping on the flat space such that each of its geodesic corresponds to a plane curve and all such planes contain the same point or are parallel, then we say that the space is subprojective one. A geodesic can also be considered as an autoparallel line, i.e. it is an object defined by the connection only. Thus a subprojective space need not to be Riemannian. It is sufficient that it is a diferentiable manifold endowed with an affine connection. As for Riemannian subprojective spaces, all of them are known in the sense that their metrics are known ([32],[39],[43]). In fact, with respect to the special local coordinates, the metric of the subprojective space has the form

$$ds^2 = (dx^1)^2 + F(x^1)\overset{*}{ds}{}^2\,(x^2, ...x^n), \tag{2.1}$$

where $\overset{*}{ds}{}^2$ is the metric of (n-1)-dimensional space of constant curvature, or the form

$$ds^2 = 2dx^1dx^2 + F(x^1)\overset{*}{ds}{}^2\,(x^3, ...x^n), \tag{2.2}$$

where $\overset{*}{ds}{}^2$ is the (n-2)-dimensional euclidean metric. The metric (2.1) is positive definite, and (2.2) is not.

We see from (2.1) and (2.2) that every subprojective space is a Warped product manifold.

It is intresting to compare subprojective Riemannian spaces to the spaces of constant curvature with respect to the group of motions. It is wel known (see for example [15] or [38]) that the group of motions of an n-dimensional Riemannian space has at most $\frac{n(n+1)}{2}$ parametres. Such a group is transitive and a space admits a group of motions of maximum order $\frac{n(n+1)}{2}$ if and only if it is a space of constant curvature.

The intransitive group has a most $\frac{n(n-1)}{2}$ parametres and all Riemannian spaces admiting such group of motions are subprojective (see [41],[45]). Conversely, every subprojective space admits intransitive group of motions of order $\frac{n(n-1)}{2}$.In both cases (2.1) and (2.2), this group acts as the transitive group on the hypersurfaces $x^1 = const.$ In the case (2.2), they are isotropic. In some casees, this intransitive group of motions becomes the transitive group of order $\frac{1}{2}n(n-1)+1$. In the case (2.1) this happens only if $F = const.$, that is if the subprojective space is decomposable. Namely, $\frac{1}{2}n(n-1)+1$ is the order of the transitive group of motions of $(n-1)$-dimensional space of constant curvature. Then we add one parameter group of motions along the curve x^1

The subprojective space of type (2.2) also admits, in some special cases, the transitive group of motions of order $\frac{1}{2}n(n-1)+1$. First, we note that, with respect to the conformally euclidean coordinates, (2.2) can be rewritten as follows

$$ds^2 = e^{-2\mu(x^1)}\left[2dx^1dx^2 + \epsilon_i(dx^1)^2\right],\ i = 3, ..., n;\ \epsilon_i = \pm 1. \tag{2.3}$$

It was proved in [41] that that the manifold endowed with metric (2.3) admits the transitive group of motions of order $\frac{1}{2}n(n-1)+1$ if and only if the function $\mu = \mu(x^1)$ satisfies

$$\frac{d\mu}{dx^1} = \frac{Ax^1 + B}{A(x^1)^2 + Cx^1 + D},$$

where A,B,C and D are constants.

Thus, while the Riemannian space of constat curvature are characterized by the property that they admit the transitive group of motions of maximum order, the subprojective spaces are characterized by the property that they admit the intransitive group of motions of maximum order.

The group of motions of general warped products are investigated in [40]. Here, we cite the following theorem.

Theorem. *([40]). If the intransitive group of motions of a Riemannian space M is of order $\frac{1}{2}q(q+1)$, $q \geq 2$ and has q-dimensional nonisotropic surfaces of the transitivity, then M is the warped product $M = \overline{M} \times_F \overset{*}{M}$ such that $dim\overset{*}{M} = q$.*

This is one of the theorems used for proving the above properties of the subprojective spaces.

We say that a manifold endowed with an affine connection is a generalized subprojective manifold if it admits a mapping on an euclidean space such that every autoparallel corresponds to the curve belonging to a $(q+1)$-dimensional plane $(1 \leq q \leq n-2)$, all this planes containing finite or infinite $(q-1)$-dimensional plane. For $q=1$, this definition of the subprojective spaces.

G.Vranceanu ([34],[35]) proved that a generalized Riemannian subprojective space with positive definite metric is the warped product $M = \overline{M} \times_F \overset{*}{M}$, where $\overset{*}{M}$ is the space of constant curvature. Conversely, each such a warped product is a generalized subprojective space. But, in the case of indefinite metric, a generalized subprojective space need not be a warped product ([36]).

3. The invariant way characterizing the warped product manifolds

Let $\overline{U} : x^1, ..., x^q$ be a local chart for the manifold $\overline{M}$ and $\overset{*}{U} : x^{q+1}, ..., x^n$ that for $\overset{*}{M}$. Then $\overline{U} \times \overset{*}{U} : x^1, ..., x^n$ is a local chart for the warped product $M = \overline{M} \times_F \overset{*}{M}$. With respect to this local chart, we have

$$\overline{g}_{ab} = \overline{g}_{ab}(x^c), \quad \overset{*}{g}_{\alpha\beta} = \overset{*}{g}_{\alpha\beta}(x^\gamma), \quad F = F(x^a), \tag{3.1}$$

while for the metric tensor g of warped product $M = \overline{M} \times_F \overset{*}{M}$, we have

$$g_{ij} = \begin{cases} \overline{g}_{ab} & \text{for} \quad i=a,\ j=b; \\ F\overset{*}{g}_{\alpha\beta} & \text{for} \quad i=\alpha,\ j=\beta; \\ 0 & \text{for all other cases.} \end{cases} \tag{3.2}$$

Here and the seqel the letters a, b, c range over the indices $1, ..., q$, greak letters α, β, γ over the indices $q+1, ..., n$ and letters i, j, k - over the indices $1, ..., n$.

The definition given in §1 shows that a Riemannian manifold is a warped product if the coordinates can be chosen such that the metric tensor takes the form (3.2) where (3.1) is satisfied. We shall see in §4 that many interesting properties of the warped product manifolds can be obtained using so adapted local coordinates.

There exsist also tensor equations, that is an invariant way, characterizing the warped products. They are contained in the following theorem.

Theorem. *([44]).- A Riemannian manifold is warped product if and only if there exsists a symmetric tensor A_{ij}, not proportional to the metric tensor, and gradient vector field $u_i$such that*

$$\nabla_k A_{ij} = \frac{1}{2}(u_i A_{kj} + u_j A_{ik}), \quad A_{ij} A^j{}_k = A_{ik}. \tag{3.3}$$

then $u_i = \dfrac{\partial}{\partial x^i} log\, F$.

Here and in the sequel, ∇ is the operator of covariant differentatiion with respect to Levi-Civita connection.

If $F = const.$, then (3.3) reduces to

$$\nabla_k A_{ij} = 0, \qquad A_{ij} A^j{}_k = A_{ik},$$

and this conditions, given by P.A.Schirokov ([46]), for a Riemannian space to be decomposable.

For a warped product manifold with l-dimensional base or with l-dimensional fiber, we have theorems:

Theorem. *([37],[44]). A Riemannian manifold is a warped product with l-dimensional base if and only if the equations*

$$\nabla_j f_i = \varphi g_{ij}, \qquad f_i = \frac{\partial f}{\partial x^i}, \qquad \varphi = \varphi(f)$$

admit solution $f \neq const.$

Theorem. *([44]).-A Riemannian manifold is a warped product with l-dimensional fiber if and only if there exists a nonisotropic vector field A_i which, together with the gradient $u_i = \dfrac{\partial}{\partial x^i} log\ F$ satisfies*

$$\nabla_j = \frac{1}{2}(A_i u_j - A_j u_i).$$

4. The geometry of the warped product in terms of warping function F and the geometries of $\overline{M}$ and $\overset{*}{M}$

There are many papers dedicated to the investigation of the geometry of the warped product $M = \overline{M} \times_F \overset{*}{M}$ in terms of warping function F and the geometries of $\overline{M}$ and $\overset{*}{M}$. In this section we quote some examples.

4.1 From now on, we suppose $F = const.$ and we use the local coordinates with respect to which the relation (3.1) and (3.2) are satisfied. We assume that each object denoted by a dash is formed $\overline{g}_{ab}$ and each object denoted by a star using $\overset{*}{g}_{\alpha\beta}$. Then the local components Γ^h_{ij} of the Levi-Civita connection on $\overline{M} \times_F \overset{*}{M}$ are the following (see for example [11], [17], [44]):

$$(4.1) \qquad \begin{cases} \Gamma^a_{bc} = \overline{\Gamma}^a_{bc}, & \Gamma^a_{\beta\gamma} = -\dfrac{1}{2}\overline{g}^{ab} F_b \overset{*}{G}_{\alpha\beta}, \qquad \Gamma^\alpha_{\beta\gamma} = \overset{*}{\Gamma}^\alpha_{\beta\gamma}, \\ \Gamma^\alpha_{a\beta} = \dfrac{1}{2F} F_a \delta^\alpha_\beta, & \Gamma^a_{\alpha b} = \Gamma^\alpha_{ab} = 0, \qquad F_a = \dfrac{\partial F}{\partial x^a}. \end{cases}$$

The local components R_{hijk} of the curvature tensor of $M = \overline{M} \times_F \overset{*}{M}$ which in general do not vanish identicaly, are the following

$$(4.2) \qquad \begin{cases} R_{abcd} = \overline{R}_{abcd}, & R_{\alpha ab\beta} = -\dfrac{1}{2} T_{ab} \overset{*}{G}_{\alpha\beta}, \\ R_{\alpha\beta\gamma\delta} = F \overset{*}{R}_{\alpha\beta\gamma\delta} - \dfrac{1}{4}\Delta_1 F \overset{*}{G}_{\alpha\beta\gamma\delta}, \end{cases}$$

where T is the (0.2) tensor with the local components T_{ij} defined by

$$T_{\alpha\beta} = T_{\alpha a} = 0, \qquad T_{ab} = \overline{\nabla}_b F_a - \frac{1}{2F} F_a F_b, \tag{4.3}$$

and

$$\Delta_1 F = \overline{g}^{ab} F_a F_b,$$
$$\overset{*}{G}_{\alpha\beta\gamma\delta} = \overset{*}{g}_{\alpha\delta}\overset{*}{g}_{\beta\gamma} - \overset{*}{g}_{\alpha\gamma}\overset{*}{g}_{\beta\delta}.$$

In view of (4.1) and (4.2), we get

$$\left\{\begin{array}{l}
\nabla_e R_{abcd} = \overline{\nabla}_e \overline{R}_{abcd},\\
\nabla_\alpha R_{abcd} = \nabla_d R_{abc\alpha} = \nabla_\gamma R_{\alpha ab\beta} = 0,\\
\nabla_c R_{ab\alpha\beta} = \nabla_\gamma R_{ab\alpha\beta} = \nabla_b R_{\alpha\beta\gamma a} = 0,\\
\nabla_\beta R_{\alpha abc} = \frac{1}{2}\left[F_e \overline{R}^e{}_{abc} - \frac{1}{2F}(F_b T_{ac} - F_c T_{ab})\right]\overset{*}{g}_{\alpha\beta},\\
\nabla_b R_{\alpha ac\beta} = -\frac{1}{2}(\nabla_b T_{ac})\overset{*}{g}_{\alpha\beta},\\
\nabla_\delta R_{\alpha\beta\gamma a} = -\frac{1}{2}\left[F_a \overset{*}{R}_{\alpha\beta\gamma\delta} + \frac{1}{2}(F_e T^e{}_a - \frac{1}{2F} F_a \Delta_1 F)\overset{*}{G}_{\alpha\beta\gamma\delta}\right],\\
\nabla_a R_{\alpha\beta\gamma\delta} = -F_a \overset{*}{R}_{\alpha\beta\gamma\delta} + \frac{1}{2}\left[\frac{F_a}{F}\Delta_1 F - \frac{1}{2}\partial_a(\Delta_1 F)\right]\overset{*}{G}_{\alpha\beta\gamma\delta},\\
\nabla_\rho R_{\alpha\beta\gamma\delta} = F\overset{*}{\nabla}_\rho \overset{*}{R}_{\alpha\beta\gamma\delta}.
\end{array}\right. \tag{4.4}$$

The local components $S_{ij} = R^r{}_{ijr}$ of the Ricci tensor of $\overline{M} \times_F \overset{*}{M}$,which in general do not vanish identically, are the following

$$\left\{\begin{array}{l}
S_{ab} = \overline{S}_{ab} - \frac{n-q}{2F} T_{ab},\\
\overset{*}{S}_{\alpha\beta} = \overset{*}{S}_{\alpha\beta} - \frac{1}{2}\left[tr(T) + \frac{n-q-1}{2F}\Delta_1 F\right]\overset{*}{g}_{\alpha\beta}, \quad tr9T) = \overline{g}^{ab} T_{ab}.
\end{array}\right. \tag{4.5}$$

The scalar curvature R of the metric $\overline{g} \times_F \overset{*}{g}$ satisfies the eqation

$$R = \overline{R} + \frac{1}{R}\overset{*}{R} - \frac{n-q}{F}\left(tr(T) + \frac{n-q-1}{4F}\Delta_1 F\right). \tag{4.6}$$

Therefore, Weyl conformal curvature tensor

$$C_{hijk} = R_{hijk} - \frac{1}{n-2}(g_{ij}S_{hk} - g_{ik}S_{hj} + g_{hk}S_{ij} - g_{hj}S_{ik}) +$$
$$+ \frac{R}{(n-1)(n-2)}(g_{ij}g_{hk} - g_{ik}g_{hj})$$

has the following components

$$
(4.7)\quad \begin{cases}
C_{abcd} = \overline{R}_{abcd} - \dfrac{1}{n-2}(\overline{g}_{ad}\overline{S}_{bc} - \overline{g}_{ac}\overline{S}_{bd} + \overline{g}_{bc}\overline{S}_{ad} - \overline{g}_{bd}\overline{S}_{ac}) + \\
\quad + \dfrac{n-q}{2(n-2)F}(\overline{g}_{ad}T_{bc} - \overline{g}_{ac}T_{bd} + \overline{g}_{bc}T_{ad} - \overline{g}_{bd}T_{ac}) + \\
\quad + \dfrac{R}{(n-1)(n-2)}\overline{G}_{abcd}, \\
C_{\alpha ab\beta} = -\dfrac{1}{n-2}\left(\dfrac{q-2}{2}T_{ab} + F\overline{S}_{ab}\right)\overset{*}{g}_{\alpha\beta} - \dfrac{1}{n-2}\overline{g}_{ab}\overset{*}{S}\alpha\beta + \\
\quad + \dfrac{1}{(n-1)(n-2)}\left[F\overline{R} + \overset{*}{R} - \dfrac{n-2q+1}{2}tr(T) + \right. \\
\quad \left. + \dfrac{(q-1)(n-q-1)}{4F}\Delta_1 F\right]\overline{g}_{ab}\overset{*}{g}_{\alpha\beta}, \\
C_{\alpha\beta\gamma\delta} = F\overset{*}{R}_{\alpha\beta\gamma\delta} - \dfrac{F}{n-2}(\overset{*}{g}_{\alpha\delta}\overset{*}{S}_{\beta\gamma} - \overset{*}{g}_{\alpha\gamma}\overset{*}{S}_{\beta\delta} + \overset{*}{g}_{\beta\gamma}\overset{*}{S}_{\alpha\delta} - \overset{*}{g}_{\beta\delta}\overset{*}{S}_{\alpha\gamma}) + \\
\quad + \dfrac{F}{n-2}\left[\dfrac{FR}{n-1} + tr(T) + \dfrac{n-2q}{4F}\Delta_1 F\right]\overset{*}{G}_{\alpha\beta\gamma\delta}, \\
C_{abc\beta} = C_{ab\alpha\beta} = C_{a\beta\gamma\delta} = 0.
\end{cases}
$$

Moreover, from (4.1) and (4.5), we find

$$
(4.8)\quad \begin{cases}
\nabla_c S_{ab} = \overline{\nabla}_c \overline{S}_{ab} - (n-q)\nabla_c\left(\dfrac{1}{2F}T_{ab}\right), \\
\nabla_\alpha S_{ab} = \nabla_b S_{a\alpha} = 0, \\
\nabla_\beta S_{a\alpha} = -\dfrac{F_a}{2F}\left[\overset{*}{S}_{\alpha\beta} - \dfrac{1}{2}\left(tr(T) + \dfrac{n-q-1}{2F}\nabla_1 F\right)\overset{*}{g}_{\alpha\beta}\right] + \\
\quad + \dfrac{1}{2}F_e\left[\overline{S}^{e}{}_{a} - \dfrac{n-q}{2F}T^{e}{}_{a}\right]\overset{*}{g}_{\alpha\beta}, \qquad T^{E}{}_{a} = \overline{g}^{eb}T_{ab}, \\
\nabla_a S_{\alpha\beta} = -\dfrac{F_a}{F}\overset{*}{S}_{\alpha\beta} + \dfrac{1}{2}\dfrac{F_a}{F}\left[tr(T) + \dfrac{n-q-1}{2F}\nabla_1 F\right]\overset{*}{g}_{\alpha\beta} - \\
\quad - \dfrac{1}{2}\partial_a\left[tr(T) + \dfrac{n-q-1}{2F}\Delta_1 F\right]\overset{*}{g}_{\alpha\beta}, \\
\nabla_\delta S_{\alpha\beta} = \overset{*}{\nabla}_\delta \overset{*}{S}_{\alpha\beta}.
\end{cases}
$$

Using (4.4), (4.6), (4.7) and (4.8) we can get the local components of $\nabla_r C_{ijkh}$ and $\nabla_r\nabla_s C_{ijkh}$.

4.2 In view of (4.4) we can state

Theorem. *([22], Th.1)-If a warped product $M = \overline{M} \times_F \overset{*}{M}$ with $n \neq q+1$ is Cartan-symmetric (i.e. if $\nabla_r R_{ijkh} = 0$). then $\overline{M}$ is Cartan-symmetric and $\overset{*}{M}$ is of a constant curvature.*

Simmilary, it follows from (4.7) that if M is conformally flat (i.e. if $C_{ijkh}=0$), then $\overset{*}{M}$ is a space of constant curvature. Conversely is not true, but we have

Theorem. *-Let $\overline{M}$ be an open interval of R with metric $g_{11}=\epsilon$, $\epsilon\in\{-1,\ 1\}$. Let F be a positive C^∞ function on $\overline{M}$ and let $dim\,\overset{*}{M}\geq 2$. Then the warped product $M=\overline{M}\times_F\overset{*}{M}$ is conformally flat if and only if $\overset{*}{M}$ is a manifold of constant curvature.*

If a recurrent space ($\nabla_r R_{ijkh}=A_r R_{ijkh}$) is a locally decomposable, then one of the decomposition space is flat and other is a recurrent space ([31,p164). The non-decomposable recurrent Riemannian spaces are all known. Some of them are warped products. For 2-recurrent ($\nabla_r\nabla_s R_{ijkh}=A_{rs}R_{ijkh}$), conformally symmetric ($\nabla_r C_{ijkh}$), conformally birecurrent ($\nabla_r\nabla_s C_{ijkh}=A_{rs}C_{ijkh}$) Riemannian spaces, we have

Theorem. *([13], [19], [22]).- If a 2-recurent (conformally symmetric, conformally recurrent, conformally birecurrent) Riemannian space is a warped product and $dim\,\overline{M}>3$, then $\overline{M}$ is 2-recurent (conformally symmetric, conformally recurrent, conformally birecurrent) and $\overset{*}{M}$ is a space of constant curvature.*

M.C.Chaki and G.Kumer ([6]) generalized this theorem for the space satisfying

$$\nabla_r\nabla_s C_{ijkh}=A_r\nabla_s C_{ijkh}+B_{rs}C_{ijkh}.$$

One generalization of a recurrent space is the Riemannian manifold satisfying ([2],-[4],[5],[16],[25],[26],[33]):

$$\nabla_r R_{ijkh}=A_r R_{ijkh}+B_i R_{rjkh}+B_j R_{irkh}+B_k R_{ijrh}+B_h R_{ijkr}. \tag{4.9}$$

Here, we shall give an example of warped product manifolds satisfying (4.9).

Let $\overline{M}$, $dim\,\overline{M}\geq 2$, be equiped with the metric

$$\overline{g}_{ab}dx^a dx^b=\sum_{a=1}^{q}e_a(dx^a)^2,\qquad e_a=\pm 1$$

and let

$$F=(C_0=C_1x^1+,\cdots,C_qx^q)^2,$$

where C_0, C_1,...,C_q are constants such that

$$\sum_{a=1}^{q}e_a(C_a)^2=0.$$

Then

$$\left\{\begin{array}{l} F_a=2C_a(C_0=C_1x^1+,\cdots,C_qx^q),\quad \overline{\nabla}_bF_a=2C_aC_b,\\ T_{ab}=0,\qquad \nabla_1F=0.\end{array}\right. \tag{4.10}$$

In view of (4.2), (4.4) and (4.10), it follows that the only components of R_{ijkh} and $\nabla_r R_{ijkh}$ not identically equal to zero are those related to

$$\begin{aligned}
&R_{\alpha\beta\gamma\delta} = F\overset{*}{R}_{\alpha\beta\gamma\delta},\\
&\nabla_\rho R_{\alpha\beta\gamma\delta} = F\overset{*}{\nabla}_\rho \overset{*}{R}_{\alpha\beta\gamma\delta},\\
&\nabla_a R_{\alpha\beta\gamma\delta} = -F_a\overset{*}{R}_{\alpha\beta\gamma\delta}, \quad \nabla_\rho R_{\alpha\beta\gamma a} = -\frac{1}{2}F_a\overset{*}{R}_{\alpha\beta\gamma\rho}.
\end{aligned}$$

Now, if $\overset{*}{M}$ is a recurrent space, i.e. if $\overset{*}{\nabla}_\rho \overset{*}{R}_{\alpha\beta\gamma\delta} = A_\rho \overset{*}{R}_{\alpha\beta\gamma\delta}$, then

$$\begin{aligned}
&\nabla_\rho R_{\alpha\beta\gamma\delta} = A_\rho R_{\alpha\beta\gamma\delta},\\
&\nabla_a R_{\alpha\beta\gamma\delta} = -\frac{F_a}{F}R_{\alpha\beta\gamma\delta}, \quad \nabla_\rho R_{\alpha\beta\gamma a} = -\frac{1}{2}\frac{F_a}{F}R_{\alpha\beta\gamma\rho},
\end{aligned}$$

and warped product$M = \overline{M} \times_F \overset{*}{M}$ satisfies (4.9), the vector fields A and Bhaving the localcomponents

$$(4.11) \qquad \begin{cases} A:\ A_\alpha, \quad A_a = -\dfrac{F_a}{F};\\ B:\ B_\alpha = 0, \quad B_a = -\dfrac{1}{2}\dfrac{F_a}{F}. \end{cases}$$

If $\overset{*}{M}$ is Cartan-symmetric, $A_\alpha = 0$, and $A = 2B$. The Ricci tensor of considered warped product satisfies

$$(4.12) \qquad \nabla_k S_{ij} = A_k S_{ij} + B_i S_{kj} + B_j S_{ik},$$

or (in the case $\overset{*}{M}$ is Cartan-symmetric)

$$(1.13) \qquad \nabla_k S_{ij} = 2B_k S_{ij} + B_i S_{kj} + B_j S_{ik}.$$

Indeed, in view of (4.10), we can reduce the relations (4.5) and (4.8) as follows

$$\begin{aligned}
&S_{ab} = S_{a\beta} = 0, \qquad S_{\alpha\beta} = \overset{*}{S}_{\alpha\beta};\\
&\nabla_c S_{ab} = \nabla_\beta S_{ab} = \nabla_b S_{a\alpha} = 0,\\
&\nabla_\beta S_{a\alpha} = -\frac{1}{2}\frac{F_a}{F}\overset{*}{S}_{\alpha\beta}, \quad \nabla_a S_{\alpha\beta} = -12\frac{F_a}{F}\overset{*}{S}_{\alpha\beta},\\
&\nabla_\delta S_{\alpha\beta} = \overset{*}{\nabla}_\delta \overset{*}{S}_{\alpha\beta}.
\end{aligned}$$

If $\overset{*}{M}$ is a reccurent manifold, then $\overset{*}{\nabla}_\rho \overset{*}{S}_{\alpha\beta} = A_\rho \overset{*}{S}_{\alpha\beta}$, i.e. it is also Ricci-recurrent and taking into account (4.11), we have (4.12).

$\overset{*}{M}$ can be Ricci-reccurent and not recurrent. For example, W.Roter determined in [29] and [30] the metrics of conformally symmetric and conformally recurrent

Ricci recurrent manifolds which are not recurrent. In this way we obtain new examples of Riemannian manifolds satisfying (4.12). The Riemannian manifolds satisfying (4.13) was introduced by M.C.Chaki ([3]) and further investigated in [4] and [7].

S.Ewert-Krzemieniewski ([16]) determined the subprojective spaces satisfying (4.9) with $A = 2B$. More precisely, he determined the function F in (2.1) such that the condition (4.9) is fulfilled for $A = 2B$.

N.Pušić ([27],[28]) investigated Ricci-recurrent warped product manifolds. Among others, she proved that if $M = \overline{M} \times_F \overset{*}{M}$ is Ricci-recurrent, then $\overset{*}{M}$ is an Einstein space.

4.3 An n-dimensional ($n \geq 4$) Riemannian manifold is said to have harmonic Weyl conformal curvature tensor ([1],p440) or to be nearly conformally symmetric ([17]) if its Ricci tensor satisfies the condition

$$\nabla_k S_{ij} - \nabla_j S_{ik} = \frac{1}{2(n-1)}(g_{ij}\nabla_k R - g_{ik}\nabla_j R). \tag{4.14}$$

Namely, it is easy to check that for every conformally symmetric manifold the condition (4.14) holds.

If the Ricci tensor satisfies

$$\nabla_k S_{ij} = \frac{n}{(n-1)(n+2)} g_{ij}\nabla_k R + \frac{n-2}{2(n-1)(n+2)}(g_{kj}\nabla_i R + g_{ik}\nabla_j R), \tag{4.15}$$

then it satisfies the condition (4.14), too.

Finaly, for Einstein manifold ($S_{ij} = \frac{R}{n} g_{ij}$), if $n > 2$ then $R = const.$ and both conditions (4.14) and (4.15) are identically satisfied.

If the warped product manifold $M = \overline{M} \times_F \overset{*}{M}$ satisfies (4.14), then $\overset{*}{M}$ is an Einstein space with a constant scalar curvature. The converse is not true. But if $dim\,\overline{M} = 1$, we have

Theorem. *([17]).-Let* $dim\,\overline{M} = 1$ *and* $g_{11} = 1$. *then the warped product* $M = \overline{M} \times_F \overset{*}{M}$ *satisfies (4.14) if and only if* $\overset{*}{M}$ *is an Einstein space and its scalar curvature is constant.*

Furthermore, if the function $f^2 = \frac{1}{F}$ is a solution of ordinary differential equation

$$\frac{d^2 f}{(dx^1)^2} - \frac{2\overset{*}{R} f^3}{(n-1)(n-2)} = const.,$$

the Ricci tensor satisfies the condition (4.15). (This is the example of manifolds satisfying (4.15), given in [1],p.433)

But, if F is given by one of the following formulas

$$\text{if } \overset{*}{R} > 0, \quad F^2 = \begin{cases} \dfrac{4}{a}\dfrac{\overset{*}{R}}{(n-1)(n-2)}\, sh^2 \dfrac{\sqrt{a}(x^1+b)}{2}, & a > 0, \\ \dfrac{\overset{*}{R}}{(n-1)(n-2)}(x^1+b)^2, & \\ -\dfrac{4}{a}\dfrac{\overset{*}{R}}{(n-1)(n-2)}\, sin^2 \dfrac{\sqrt{-a}(x^1+b)}{2}, & a < 0; \end{cases}$$

$$\text{if } \overset{*}{R} = 0, \quad F^2 = be^{ax^1}, \quad a \neq 0;$$

$$\text{if } \overset{*}{R} < 0, \quad F^2 = -\frac{4}{a}\frac{\overset{*}{R}}{(n-1)(n-2)}\, cos\,h^2 \frac{\sqrt{a}(x^1+b)}{2}, \quad a > 0;$$

where a and b are constans, then warped product $M = \overline{M} \times_F \overset{*}{M}$ is an Einstein space ([18]).

It is interesting to note that warped product manifold provided with the metric

$$ds^2 = -(dx^1)^2 + F\overset{*}{g}_{\alpha\beta}dx^\alpha dx^\beta$$

satisfies (4.14) if and only if $\overset{*}{g}_{\alpha\beta}dx^\alpha dx^\beta$ is the metric of an Einstein space with constant scalar curvature and the function F has the form $F = e^{bx+a}$, where a and b are constants.

4.4 The Riemannian space is said to be semi-symmetric if its curvature tensor satisfies

$$R \cdot R = 0, \tag{4.16}$$

where the first tensor acts on the second as a derivation.

There are many various possibilities to obtain curvature conditions weaker that (4.16). To expres them, let $\widetilde{R}(X, Y)$ and $X \wedge_A Y$ be defined by

$$\widetilde{R}(X, Y)Z = \nabla_X\nabla_Y Z - \nabla_Y\nabla_X Z - \nabla_{[X,Y]}Z,$$
$$(X \wedge_A Y)Z = A(Y, Z)X - A(X, Z)Y,$$

respectively, where X, Y, Z are vector fields and A is an $(0, 2)$ tensor field on (M, g). For (0, k)-tensor field P on M, $k \geq 1$, we define tensors $R \cdot P$ and $Q(A, P)$ by the formulas

$$\begin{aligned}(R \cdot P)(X_1, ..., X_k; X, Y) =& -P(\widetilde{R}(X, Y)X_1, \cdots, X_k) - \cdots - \\ & P(X_1, \cdots, X_{k-1}, \widetilde{R}(X, Y)X_k), \\ Q(A, P)(X_1, ..., X_k; X, Y) =& P((X \wedge_A Y)X_1, \cdots, X_k) + \cdots + \\ & P(X_1, \cdots, X_{k-1}, (X \wedge Y)X_k).\end{aligned}$$

Then, the desired conditions weaker then (4.16) are

$$R \cdot R = \mathcal{L}Q(g,\, R), \tag{4.18}$$

$$R \cdot R = Q(S,\, R), \tag{4.19}$$

$$C \cdot C = \mathcal{L}Q(g,\, C), \qquad (4.19)$$

$$R \cdot R = Q(S,\, R) + \mathcal{L}Q(g,\, C), \tag{4.20}$$

where C is the conformal curvature tensor and $\mathcal{L}$ is a function on M.

There exsists many examples of warped product manifolds satisfying one of these conditions. We cite some of them.

Theorem. *([11]).- Let M be an open interval of $\mathbf{R}$ with the metric $g_{11} = \epsilon$, $\epsilon \in \{-1,\, 1\}$, F a positive C^∞ function on $\overline{M}$ and na $* M$ a manifold of constant curvature. Then the warped prodoct $M = \overline{M} \times_F \overset{*}{M}$ satisfied (4.17).*

Theorem. *([11]).- Let $\overline{M}$ be an open subset of $\mathbf{R}^q \setminus \{0, ..., 0\}$, $q \geq 2$, eqiped with the metric $\overline{g}_{ab} = \delta_{ab}$, $F(x^1, ..., x^q) = \frac{1}{4}[(x^1)^2 + ... + (x^q)^2]^2$ and na $* M$ $(dim\, \overset{*}{M} \geq 2)$ a localy flat manifold. Then the warped prodoct $M = \overline{M} \times_F \overset{*}{M}$ satisfied (4.17).*

Theorem. *([8]).- Let $\overline{M} = \{(x^1,\, x^2) \in \mathbf{R}^2$ and $x^1 > 0,\, x^2 > 0\}$ be a 2-dimensional manifold with the metric $\overline{g}$ defined by $\overline{g}_{ab} = \epsilon_a$, $\epsilon_a = \pm 1$. Let $\overset{*}{M}$, $dim\, \overset{*}{M} \geq 2$ be a manifold of constsnt curvature and let*

$$F(x^1,\, x^2) = (x^1)^{\frac{c+1}{c}} \cdot (x^2)^{\frac{c-1}{c}},$$

where c is nonzero constant. Then the warped product $M = \overline{M} \times_F \overset{*}{M}$ satisfied (4.18).

Theorem. *([14]).- Let $\overline{M}$ be a l-dimensional manifold and let $\overset{*}{M}$ be a 3-dimensional manifold or (if $dim\, M \geq 4$) conformally flat. Then the warped product $M = \overline{M} \times_F \overset{*}{M}$ satisfied (4.19) if and only if*

$$\overset{*}{S}_{\alpha\beta} = \mu \overset{*}{g}_{\alpha\beta} + \nu u_\alpha u_\beta,$$

where μ and ν are function and u_α is a vector field on $\overset{*}{M}$.

Theorem. *([10]).- Let $\overline{M}$, $dim\, \overline{M} = q \geq 2$ and $\overset{*}{M}$, $dim\, \overset{*}{M}$ be two Riemannian manifolds of constant curvature and F a positive smoth function on $\overline{M}$. Then the* *warped product* $M = \overline{M} \times_F \overset{*}{M}$ *satisfied (4.20) with* $\mathcal{L} = -\dfrac{n-2}{q(q-1)}\overline{R}$ *if and only if, at every point of* $\overline{M}$, *the condition*

$$rank\left(\frac{1}{2}T_{ab} - \frac{F\mathcal{L}}{n-2}\overline{g}_{ab}\right) \leq 1$$

is satisfied. (The (0, 2) tensor T *is defined by (4. 3) and* $\overline{R}$ *is the scalar curvature of* $\overline{M}$.)

Theorem. *([10]).- Let* F *be a positive smooth function on 2-dimensional Riemannian manifold* $\overline{M}$ *such that the tensor* T *is proportional to* $\overline{g}$. *Moreover, let* $\overset{*}{M}$, $\dim \overset{*}{M} \geq 2$, *be a manifold of constant curvature. Let the function* $\mathcal{L}$ *defined by*

$$\mathcal{L} = \frac{n-3}{4}\frac{tr\,(T)}{F} - \frac{\overline{R}}{2}$$

satisfied $\mathcal{L} = -\dfrac{n-2}{2}\overline{R}$. *Then* $M = \overline{M} \times_F \overset{*}{M}$ *is a manifold fulfilling (4.20).*

Theorem. *([9]).- An Einstein manifold* (M, g), $\dim M \geq 4$, *satisfying (4.17), satisfies the condition (4.19), too.*

References

[1] BEES A.L., *Einstein manifolds*, Springer-Verlag, 1987.

[2] CSHAKI M.C., *On pseudo pseudo-symmetric manifolds*, An.Stiint.Univ."Al.I.Cuza, Iasi, Ser. Ia Mat. **33** (1987), 53-58.

[3] CSHAKI M.C., *On pseudo Ricci-symmetric manifolds*, Bulgar.J.Phys. **15** (1988), 526-531.

[4] CSHAKI M.C., BARNA B., *On a new type of Riemannian manifolds and its aplication to general relativity,*, Mahavishva **4** (1991), 63-65.

[5] CHAKI M.C., DE U.C, *On pseudo-symmetric spaces*, Acta Math. Hungar. **54(3-4)** (1989), 185-190.

[6] CHAKI M.C.,KUMAR G., *On semi-decomposable generalized conformally 2-recurrent space*, Mathematica, Revue d'Analyse numerique et de la théorie de l'approximation, T.30 **53, No 1** (1988), 11-18.

[7] CHAKI M.C., TARAFDAR M., *On conformally flat pseudo-Ricci symmetric manifolds*, Period. Math. Hungar. **19(30)** (1988), 209-215.

[8] DEFEVER F., DESZCZ R., *On warped product manifolds satisfying a certain curvature condition*, Atti Academia Peloritana dei Pericolonti, Clase I di Sci. Fiz. Math. e nat. **69** (1991), 213-236.

[9] DEFEVER F., DESZCZ R., *On Riemannian manifolds satisfying a certain curvature condition imposed on the Weyl curvature tensor*, Acta univ. Palackianea Olomucensis facultas rerum naturalum, Mathematica, 32 **110** (1993), 27-34.

[10] DEFEVER F., DESZCZ R., PRVANOVIĆ M., *On warped product manifolds satisfying some curvature condition of pseudosymmetry type*, submited for publication.

[11] DEPREZ J., DESZCZ R., VERSTRAELEN L., *Examples of pseudo-symmetric flat warped products*, Chinese J. Math. No 1 **17** (1989), 51-65.

[12] DERDZINSKI A., ROTER W., *Some theorems on conformally symmetric manifolds*, Tensor **32** (1978), 11-13.

[13] Deszcz R., *On semi decomposable conformally recurrent and conformally birecurrent Riemannian spaces,*, Scientific papers, Inst.Math.Wroclaw Tech.Univ. **No 16** (1976), 27-32.

[14] Deszcz R., Verstraelen L. Yaprak S., *Warped products realizing a certain condition of pseudometric type imposed on the Weylcurvature tensor*, Chinese J.Math. **22 No3** (1994), 139-157.

[15] Eisenhart L.P., *Riemannian geometry*, Princeton University Press, 1949.

[16] Eweret-Krzemieniewski S., *On some generalization of recurrent manifolds*, Mathematica Pannonica **4/2** (1993), 191-203.

[17] Gebarowski A., *Nearly conformally symmetric warped product manifolds*, Bull. Inst. Acad. Sinica **20, No 4** (1992), 359-377.

[18] Gebarowski A., *On Einstein warped product*, Tensor **52** (1993), 204-207.

[19] Grucak W., *On semi-decomposable 2-recurent Riemannian space*, Scientific papers, Inst. Math. Wroclaw Tech. Univ. **No 16** (1976), 15-25.

[20] Kagan B., *Über eine Erweiterung des Begriffes vom projectiven Raume und dem zugehörigen Absolut*, Abhandlungen aus dem Seminar für Vektor-und Tensoranalysis, Lieferung I, Moskau (1933), 12-101.

[21] Kagan B., *Der Ausnahmefall in der Theorie der subprojectiven Räume, Abhandlungen aus dem Seminar für Vektor-und Tensoranalysis*, Lieferung II-III, Moskau (1935), 151-170.

[22] Krawczyk A., *Some theorems on semi-decomposable conformally symmetric spaces*, Scientific papers, Inst. Math. Wroclaw Tech. Univ. **No 16** (1976), 3-10.

[23] O'Neill B., *Semi-Riemanian geometry with application to relativity*, Academic Pres (1983).

[24] Prvanović M., *Poludekomponovani rekurentni Rimanovi prostori*, Godišnjak Filozofskog fakulteta u Novom Sadu **XI/2** (1968), 717-720.

[25] Prvanović M., *Generalized recurrent Riemannian manifold*, An. Stinit. Univ. Al. I. Cuza, Iasi Ser. Ia Math. **38** (1992), 423-434.

[26] Prvanović M., *On weakly symmetric Riemannian manifold*, Publications Mathematicae Debrecin, in print.

[27] Pušić N., *On Ricci recurrent semi-decomposable Riemanian spaes*, Zb. Rad. PMF u Novom Sadu, Ser. Mat **21,2** (1991), 49-59.

[28] Pušić N., *On Ricci recurrent semi-decomposable Riemanian spaes with vanishing scalar curvature*, Zb. Rad. PMF u Novom Sadu, Ser. Mat; in print **23,1** (1993).

[29] Roter W., *On conformally symmetric Ricci-recurent spaces*, Colloq. Math. **XXXI** (1974), 87-96.

[30] Roter W., *On the existence of conformally symmetric Ricci-recurent spaces*, Bull. Acad. Polonaise Sci., Ser. Math. Ast. Phys. **XXIV, No11** (1976), 973-979.

[31] Ruse H.S., Walker A.G., Willmore T.J., *Harmonic spaces*, Ed.Cremonese, Roma, 1961.

[32] Schapiro H., *Über die Metrik der subprojective Raume*, Abhandlungen aus dem Seminar für Vektor -und Tensoranalysis, Lieferung I, Moskau (1935), 102-125.

[33] Tamássy L., Binh T.Q., *On weakly symmetric and projective symmetric Riemannian manifolds*, Colloq. Math. Soc. János Bolyai **56** (1992), 663-670.

[34] Vránceanu G., *Aspura spatiilor lui Kagan metrice*, Bull. Stiint. mat. fiz. chim **No 6** (1950), 503-508.

[35] Vránceanu G., *Lecon de géométrie differentielle*, Ed. Acad. Roumanie, Buchharest, 1957.

[36] Vránceanu G., *Espaces de Riemann partiellement projectives á metrique indéfinie*, Math. Nachr. **18 No 1-6** (1958), 123-126.

[37] Yano K., *Concircular geometry*, Proc. Japan Acad. **16** (1940), 195-200, 354-360, 442-448, 505-511.

[38] Yano K., *The theory of the Lie derivatives and its applications*, North-Holand Publisching, 1955.

[39] Каган В.Ф. , *Субпројективные пространства* , Гос. из. физ. мат. литер., Москва , 1961.

[40] Кручкович Г.И. , *О движениях в полупроводимых Риманових пространствах* , Успехи мат. наук, т XII **6 (78)** (1957), 149-156.

[41] ———, *О движениях в субпроективных пространствах В.Ф.Кагана*, Научн. док. выс. школ., физ.мат. науки **No 1** (1958), 43-47.

[42] ———, *О Риманобых пространствах с достаточно большой группой движений*, ДАН, том 133 **No 6** (1960), 1283-1286.

[43] ———, *О пространствах В.Ф. Кагана, в книге: Каган В.Ф., Субпроективные пространства* , Гос. изд. физ. мат. литер., Москва , 1961.

[44] ———, *Об одном классе Риманобых пространств* , Труды сем. вектор. тенсор. анал., вып **XI** (1961), 103-128.

[45] ———, *Пространства Кагана и нетранзитивхые группы движения* , Труды сем. вектор. тенсор. анал., вып **XIV** (1968), 144-153.

[46] ШИРОКОВ П.А. , *Симетрические конформно-евклидовые пространства* , Изд. Казансск. физ. матем. об-ва сер **3,11** (1938), 9-27.

FILOMAT (Niš) 9:2 (1995), 187–195

Filomat '94, Niš, October 22–24, 1994
Geometry. Computer Sciences

HOLOMORPHICALLY-PROJECTIVE CONNECTIONS OF A HYPERBOLIC KAEHLERIAN SPACE

Nevena Pušić

ABSTRACT. *We consider the set of connections on a hyperbolic Kaehlerian space which are in holomorphically-projective correspodence to the Levi-Civita connection. We find an invariant tensor of curvature type for all these connections.*

1. About hyperbolic Kaehlerian space

A hyperbolic Kaehlerian space M_n $(n = 2m)$ is a differentiable manifold with indefinite metrics

(1.1) $$ds^2 = g_{ij}dx^i dx^j$$

and so-called strusture$(F_j{}^i)$(which is itself a linear transformation of the tangent space, in every point), which satisfies

(1.2) $$F_j{}^s F_s{}^i = \delta^i_j$$

The metrics and the structure are conected in the following way

(1.3) $$F_{ij} = g_{is}F^s{}_j = g_{js}F_i{}^s = -F_{ji}$$

(1.4) $$\check{\nabla}_k F_j{}^i = 0.$$

The tensor (F_{ij}), appearing in (1. 3), which we have formally got from the structure tensor by lowering the upper index, is the covariant structure tensor. The symbol $\check{\nabla}$ denotes the ooperator of covariant differentiation towards the Levi-Civita connection. (1. 4) means that the structure tensor is parallel regarding to the Levi-Civita connection. It is clear that the covariant structure tensor is also parallel. (1. 2) means that the structure is involutive as a linear transformation of the tangent space in every point.

The structure tensor is a real nondegenerate tensor and it has n linearly independent eigenvectors; its matrix has a diagonal expression.

There holds

Lemma 1. *(A) Every tangent vector of a hyperbolic Kaehlerian space is transformed by the structure into an orthogonal vector.*
(B) The scalar square of a vector-original is opposite to the scalar square of the vector-image.

Proof. (A) $a_j F_i{}^j = b_i$

$$a_j b^j = a_j a_s F_t{}^s g^{tj} = a_j a_s F^{js} = -a_j a_s F^{sj} = -a_j b^j = 0$$

$$(B) b_s b^s = b_s b_t g^{ts} = b_s a_j F_t{}^j g^{ts} = b_s a_j F^{sj} =$$
$$= -a_j b_s F^{js} = -a_j a^j . \square$$

The fact that the structure has eigenvectors is enabled by the fact that the metrics is indefinite. We shall give here some features of eigenvalues and eigevectors of the structure.

Lemma 2. *For two different eigenvectors of the structure on a hyperbolic Kaehlerian space either the eigenvalues are mutually iopposite or the eigenvectors are mutually orthogonal.*

Proof. Suppose that u and v are two different eigenvectors for the structure, with eigenvalues λ and κ respectively. Then

$$u_a v^a = u_j v_k g^{jk} = \frac{1}{\kappa} u_j F_k{}^s v_s g^{jk}$$
$$= \frac{1}{\kappa} u_j v_s F^{js} = -\frac{1}{\kappa} v_s u_j F^{sj} = -\frac{\lambda}{\kappa} v_s u^s ,$$

and

$$u_a v^a (1 + \frac{\lambda}{\kappa}) = 0 \tag{1.5}$$

and the Lemma is proved. $\square$

Lemma 3. *If on a hyperbolic Kaehlerian space the vector u is an eigenvector for the structure, then Fu is also an eigenvector for the structure.*

Proof. $v = Fu$, $v_i = F_i{}^a u_a = \lambda u_i$

$$F_j{}^i v_i = F_j{}^i F_i{}^a u_a = u_j = \lambda^2 u_j = \lambda v_j$$

and the Lemma is proved. $\square$

It is obvious from the proof of the Lemma 3. that only eigenvalues of the structure are $\lambda = \pm 1$.

According to the Lemma 1, eigenvectors of the structure tensor are self-orthogonal, i. e. their scalar square vanishes. As the structure has n (dimension of the manifold) linearly independent eigenvectors, there exists a basis of the tangent space which consists of isotropic vectors. We call such a basis an **adapted basis**. Such a basis shows in the simplest way the geometry of a hyperbolic Kaehlerian space. We can construct an adapted basis in the following way: we put on the first $m = \frac{n}{2}$ places those eigenvectors with corresponding eigenvalue 1; on the second m places we put those m eigenvectors with corresponding eigenvalue -1. According to the Lemma 3, there is no other eigenvalues. According to the Lemma 2, in every of these subspaces every basic vector is orthogonal to the every other basic vector; on every of these subspaces induced metrics vanishes identically. Besides, every of these subspaces is invariant under structure isomorphism. This means that a hyperbolic Kaehlerian space is decomposed in very natural way into two totaly geodesic subspaces of same dimension.

We have to mention that, according to the Lemma 1, there exist vectors with positive scalar square (space-like vectors) and those with negative scalar sqare (time-like vectors).

2. Holomorphically planer curves

A two-dimensional submanifold of the manifold M_n with a tangent subspace of the tangent space on M_n, generated by vectors u, Fu we call a holomorphic section of a hyperbolic Kaehlerian space.

A curve $\xi^h(t)$ on M_n satisfying the differential equation

$$\frac{d^2\xi^h}{dt^2} + \Lambda^h_{ji}\frac{d\xi^j}{dt}\frac{d\xi^i}{dt} == \alpha(t)\frac{d\xi^h}{dt} + \beta(t)F_s{}^h\frac{d\xi^s}{dt} \tag{2.1}$$

where $\alpha(t)$ and $\beta(t)$ are functions depending of the parameter t, we call a **holomorphically planer curve**. It can be seen from (2. 1) that a curve is holomorphically planer if and only if holomorphic sections generated by tangent vectors are parallel along the curve.

Two F-connections (satisfying $\nabla_k F = 0$) are said to be mutually holomorphically projective if and only if they have holomorhically planer curves in common.

It is easy to prove that there holds

Proposition 1. *Two symmetric F-connections with coefficients Λ^i_{jk} and $\bar{\Lambda}^i_{jk}$ are holomorphically projective if and only if*

$$\begin{aligned}\bar{\Lambda}^i_{jk} &= \Lambda^i_{jk} + p_j\delta^i_k + p_k\delta^i_j \\ &+ p_sF_j{}^sF_k{}^i + p_jF_k{}^sF_j{}^i\end{aligned} \tag{2.2}$$

for some vector field (p_j).

In this article, we shall investigate connections which are holomorphically projective to the Levi-Civita connection.

3. Holomorphically-projective connections

We say that a connection with coefficients Λ^i_{jk} on a hyperbolic Kaehlerian space is holomorphically-projective if its coefficients have the form:

$$\Lambda^i_{jk} = \check{\Gamma}^i_{jk} + p_j\delta^i_k + p_k\delta^i_j + q_jF_k{}^i + q_kF_j{}^i, \tag{3.1}$$

where (p_j) are components of a gradient vector field and (q_j) are components of a vector which is an image of (p_j) under the structure. $\check{\Gamma}$ stands for Christoffel symbols. It is obvious that a holomorphically-projective connection has holomorphically planer curves in common with Levi-Civita connection.

The curvature tensor of the connection (3. 1) has the form

$$R_{ijkl} = K_{ijkl} + g_{ki}p_{lj} - g_{li}p_{kj} + F_{ki}q_{lj} - F_{li}q_{kj} + F_{ji}(q_{kl} - q_{lk}) \tag{3.2}$$

where

$$p_{lj} = \check{\nabla}_l p_j - p_lp_j - q_lq_j \tag{3.3}$$

$$q_{lj} = \check{\nabla}_l q_j - p_lq_j - q_lp_j \tag{3.4}$$

and

$$q_{lj} = F_j{}^a p_{la} \tag{3.5}$$

By K_{ijkl} we denote Riemann-Christoffel tensor of the hyperbolic Kaehlerian space.

In order to eliminate p_{lj} and q_{lj} from the expression (3. 2), we shall suppose that the curvature tensor of the holomorphically-projective connection is invariant under change of places of the first and second pair of indices:

$$R_{ijkl} = R_{klij} \tag{3.6}$$

By (3. 2), we obtain from (3. 6)

$$g_{jk}p_{il} - g_{li}p_{kj} + F_{ki}(q_{lj} + q_{jl}) - F_{li}q_{kj} + F_{jk}q_{il} + F_{ji}(q_{kl} - q_{lk}) - F_{lk}(q_{ij} - q_{ji}) = 0$$

After transvection the upper equality by F^{jk}, we obtain

$$q_{li} + (1-n)q_{il} = F_{il}p^s_s \tag{3.7}$$

where p^s_s stands for $p_{ij}g^{ij}$.

As the covariant structure tensor is skew-symmetric, then the left-hand side of (3. 7) is also skew-symmetric and there holds

$$q_{li} + (1-n)q_{il} = -q_{il} - (1-n)q_{li}$$

what means

$$q_{il} = -q_{li}. \tag{3.8}$$

Now, the curvature tensor of holomorphically-projective connection on a hyperbolic Kaehlerian space has the form

$$R_{ijkl} = K_{ijkl} + g_{ki}p_{lj} - g_{li}p_{kj} + F_{ki}q_{lj} - F_{li}q_{kj} + 2F_{ji}q_{kl}. \tag{3.9}$$

4. HP-curvature tensor

Taking into account equalities (3. 7) and (3. 8), one can easily get

$$q_{li} = -\frac{p_s{}^s}{n}F_{li} \tag{4.1}$$

and by the relation $p_{li} = F_i{}^a q_{la}$,

$$p_{li} = \frac{p_s{}^s}{n}g_{li} \tag{4.2}$$

Using (3. 9), we can find the Ricci tensor of the holomorphically-projective connection

$$p_{li} = \frac{R_{li} - K_{li}}{2-n} \tag{4.3}$$

and

$$p_s{}^s = \frac{R-K}{2-n} \tag{4.4}$$

Then, we have

$$q_{li} = -\frac{R-K}{n(2-n)}F_{li}, \tag{4.5}$$

$$p_{li} = \frac{R-K}{n(2-n)}g_{li}. \tag{4.6}$$

If we substitute (4. 5) and (4. 6) into (3. 9), we obtain

$$R_{ijkl} - \frac{R}{n(2-n)}(g_{ki}g_{lj} - g_{li}g_{kj} - F_{ki}F_{lj} + F_{li}F_{kj} - 2F_{ji}F_{kl}) =$$

$$= K_{ijkl} - \frac{K}{n(2-n)}(g_{ki}g_{lj} - g_{li}g_{kj} - F_{ki}F_{lj} + F_{li}F_{kj} - 2F_{ji}F_{kl}).$$

The tensor on the right-hand side of the upper equality we call the **holomorphically-projective curvature tensor** of a hyperbolic Kaehlerian space. We have proved

Theorem 1. *The tensor*

(4.7)
$$HP_{ijkl} =$$
$$K_{ijkl} - \frac{K}{n(2-n)}(g_{ki}g_{lj} - g_{li}g_{kj} - F_{ki}F_{lj} + F_{li}F_{kj} - 2F_{ji}F_{kl})$$
is an invariant tensor of holomorphically-projective connections on the hyperbolic Kaehlerian space.

We can also prove that there holds

Theorem 2. *The curvature tensor of a holomorphically-projective connection on a hyperbolic Kaehlerian space is skew-symetric in first two indices, but it does not satisfy the first Bianchi identity, except of some special cases.*

Proof. One can easily check, using (3. 9), that R_{ijkl} is skew-symmetric in first two indices.

If we suppose that R_{ijkl} satisfies the first Bianchi identity, then, by (3. 9), we obtain
$$\begin{aligned} 0 = K_{ijkl} + K_{iklj} + K_{iljk} + \\ +g_{ki}p_{lj} - g_{li}p_{kj} + F_{ki}q_{lj} - F_{li}q_{kj} + 2F_{ji}q_{kl} \\ +g_{li}p_{jk} - g_{ji}p_{lk} + F_{li}q_{jk} - F_{ji}q_{lk} + 2F_{ki}q_{lj} \\ +g_{ji}p_{kl} - g_{ki}p_{jl} + F_{ji}q_{kl} - F_{ki}q_{jl} + 2F_{li}q_{jk} = \\ = 4(F_{ki}q_{lj} - F_{li}q_{kj} + F_{ji}q_{kl}), \end{aligned}$$
and, taking into account (4. 1)
$$\frac{p_s{}^s}{n}(F_{ki}F_{lj} - F_{li}F_{kj} + F_{ji}F_{kl} = 0).$$
If we suppose that the expression in parentheses vanishes, then, after contraction by F^{ik},
$$(n-2)F_{lj} = 0,$$
what is senseless. Then $p_s{}^s = 0$ and, regarding to (4. 4), $K = R$, what is a special case. □

Also, we can prove

Theorem 3. *The holomorphically-projective curvature tensor of a hyperbolic Kaehlerian space satisfies the following relations*
$$(a) HP_{ijkl} = -HP_{ijlk}; HP_{ijkl} = -HP_{jikl}; HP_{ijkl} = HP_{klij}$$
$$(b) HP_{ijkl} + HP_{iklj} + HP_{iljk} = -4(F_{ki}F_{lj} - F_{li}F_{kj} + F_{ji}F_{kl})$$
$$(c) HP^t{}_{jkt} = K_{jk} - \frac{K}{n}g_{jk}$$
$$(d) HP^i{}_{tkl}F_j{}^t - HP^t{}_{jkl}F_t{}^i = 0.$$

One can easily prove all these properties using the expression (3. 9).

5. Some special cases

There always holds

$$p_s{}^s = \check{\nabla}_s p^s \tag{5.1}$$

according to the Lemma 1. Then also holds

$$R - K = (2-n)\check{\nabla}_s p^s;\ p_{li} = \frac{\check{\nabla}_s p^s}{n} g_{li};\ q_{li} = -\frac{\check{\nabla}_s p^s}{n} F_{li}. \tag{5.2}$$

As the first special case we shall consider that one when the vector field (p^i) generating holomorphically-projective connection is a harmonic vector field, that is

$$\check{\nabla}_s p^s = 0.$$

Then, according to (5. 2), there holds

$$R = K;\ p_{li} = 0;\ q_{li} = 0 \tag{5.3}$$

and then

$$R_{ijkl} = K_{ijkl}$$

and the curvature tensor of the holomorphically-projective connection in this special case will satisfy the first Bianchi identity.

The other special case which we are going to consider here is that one when the generating vector field for the holomorphically-projective connection is an eigenvector for the structure; then the holomorphic section is invariant for the structure. As the only eigenvalues for the structure are ± 1, then holds

$$q_{lj} = \pm p_{li} \tag{5.4}$$

As the tensor (p_{lj}) is symmetric and the tensor (q_{lj}) is skew-symmetric, there will hold

$$q_{lj} = p_{lj} = 0. \tag{5.5}$$

This means that

$$p_s{}^s = \check{\nabla}_s p^s = 0$$

i. e. that the generating vector field is a harmonic one.

If the vector field (p_i) is harmonic or isotropic, then

$$\check{\nabla}_j p_i = p_i p_j + q_i q_j;\ \check{\nabla}_j q_i = p_j q_i + q_j p_i.$$

According to the Ricci identity, there holds

$$\check{\nabla} j \check{\nabla}_k p_i - \check{\nabla}_k \check{\nabla}_j p_i = -K^t{}_{ikj} p_t = 0$$

After contraction by g^{ik}, we obtain

$$-K^{t}{}_{j}p_t = 0$$

or, consequently

$$g^{jk}\check{\nabla}_j\check{\nabla}_k p_i = 0.$$

There holds

Theorem 4. *If the vector which is generating a holomorphically-projective connection of the hyperbolic Kaehlerian space is a harmonic vector field, then the curvature tensor of the hyperbolic Kaehlerian space is equal to the curvature tensor of the holomorphically-projective connection. An example of generating harmonic vector field is a structure eigenvector field. For such a vector field there holds*

$$K^{t}{}_{j}p_t = 0 \textit{ and } g^{jk}\check{\nabla}_j\check{\nabla}_k p_i = 0.$$

If the generating vector field has constant scalar square, then the difference between R and K is constant.

Proof. We shall prove just the last statement.

$$\frac{\partial}{\partial x^k}(p_s p^s) = \check{\nabla}_k p_s p^s = p_s\check{\nabla}_k p^s + p^s\check{\nabla}_k p_s.$$

But

$$(5.6) \qquad \check{\nabla}_k p_s = p_{ks} + p_k p_s + q_k q_s \quad \text{and} \quad \check{\nabla}_k p^s = p_k^s + p_k p^s + q_k q^s$$

As $p_{ks} = \frac{p_s^s}{n} g_{ks}$ and $p_k^s = \frac{p_s^s}{n}\delta_k^s$, then

$$0 = \frac{\partial}{\partial x^k}(p_s p^s) = 2(\frac{p_s^s}{n} + p_s p^s)p_k$$

and, consequently,

$$p_s p^s = -\frac{p_s^s}{n}.$$

But,

$$R - K = (2-n)p_s^s = n(n-2)p_s p^s$$

and the proof is completed. □

Refrences

[1] M. Prvanović, *Holomorphically-projective transformations in a locally product space*, Mathematica Balcanica, **1**(1971) 193–213

[2] N. Pušić, *On invariant tensor of a conformal transformation of a hyperbolic Kaehlerian space*, Zbornik radova Filozofskog fakulteta u Nišu, Serija Matematika, **4**(1990) 55–64

[3] K. Yano, *Differential geometry of complex and almost complex spaces*, Pergamon Press, New York, 1965.

Institut za matematiku PMF, 21000 Novi Sad, dr Ilije Djuričića 4, Yugoslavia

FILOMAT (Niš) 9:2 (1995), 197–204
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ON INFINITESIMAL DEFORMATIONS OF A TOROID ROTATIONAL SURFACE GENERATED BY A QUADRANGULAR MERIDIAN

Ljubica Velimirović

ABSTRACT. *In this paper we consider a toroid rotational surface with a quadrangular meridian and obtain a necessary and sufficient condition for infinitesimal deformations of such a surface (eq.(1.18)). It is determined the field of deformations too.*

0. Introduction

In the paper [1] K.M. Belov gave necessary and sufficient condition for infinitesimal deformations of a toroid surface of rotation generated by a special case of the meridian.

One puts question of considering infintensimal deformations, i.e. of the rigidity of a surface with any quadrangular meridian.

In the plane of the meridian which rotates around the u-axis let's introduce Descartes' orthogonal coordinate system $uO\rho$ and let $\rho = \rho(u)$ be the equation of the meridian. If $\bar{e}$ is unit vector of the axis of rotation, $\bar{a}(v)$ unit vector of the ρ-axis, where v is the angle between the plane of initial position of the meridian and $\bar{a}(v)$ then $\bar{a}'(v) \perp \bar{a}(v)$ and $\bar{a}'(v) \perp \bar{e}$ (see [2], page 90, or [3] page 253).

The equation of a surface of rotation, in the coordinate system with the base $\bar{e}, \bar{a}, \bar{a}'$ is

$$\bar{r}(u,v) = u\bar{e} + \rho(u)\bar{a}(v). \tag{0.1}$$

As it is known ([2],page 91.) for every $k \in \{2, 3, ...\}$ there is a field of infinitesimal deformations

$$\begin{aligned} \bar{z}(u,v) &= [\varphi_k(u)e^{ikv} + \tilde{\varphi}_k(u)e^{-ikv}]\bar{e} \\ &+ [\psi_k(u)e^{ikv} + \tilde{\psi}_k(u)e^{-ikv}]\bar{a}(v) \\ &+ [\chi_k(u)e^{ikv} + \tilde{\chi}_k(u)e^{-ikv}]\bar{a}'(v) \end{aligned} \tag{0.2}$$

1991 *Mathematics Subject Classification*: 53A05
Supported by Grant 0401A of RFNS through Math. Inst. SANU

of a surface (0.1),where e.g. $\tilde{\varphi}_k(u)$ is the conjugated value for $\varphi_k(u)$. The functions $\psi_k(u)$, $\chi_k(u)$ satisfy differential equation in the form of

$$\rho(u)\lambda''(u)+(k^2-1)\rho''(u)\lambda(u)=0, \tag{0.3}$$

where $\lambda(u)$ is unknown function,and also satisfy the system

$$\begin{gathered}\varphi_k'(u)+\rho'(u)\psi_k'(u)=0, \quad \psi_k(u)+ik\chi_k(u)=0\\ ik\varphi(u)+\rho'(u)[ik\psi_k(u)-\chi_k(u)]+\rho(u)\chi_k'(u)=0.\end{gathered} \tag{0.4}$$

In the vertexes $u=\sigma$ of the meridian, $\psi_k(u)$ satisfy the equation ([2],page 112)

$$\rho(\sigma)[\psi_k'(\sigma+0)-\psi_k'(\sigma-0)]+(k^2-1)\psi_k(\sigma)[\rho'(\sigma+0)-\rho'(\sigma-0)]=0, \tag{0.5}$$

supposing the function $\varphi_k(u),\chi_k(u)$ to be continuous in this points. Analogously,the equation (0.5) is satisfied for $\chi_k(u)$, if $\varphi_k(u),\psi_k(u)$ are continuous.

1. Condition for the existence of infinitesimal deformations

Suppose that quadrangle $A_i(u_i,\rho_i)$ $(i=1,2,3,4;\ \rho_i>0)$ rotates around the u-axis. If $\rho_{(1)}$ is value of ρ on the A_1A_2, $\rho_{(2)}$ on A_2A_3 etc., we get the equations of the sides of the meridian

$$\begin{gathered}A_iA_{i+1}:\rho_{(i)}=\rho_i+\frac{\rho_{i+1}-\rho_i}{u_{i+1}-u_i}(u-u_i),\\ (\ i=1,2,3,4;A_5\equiv A_1,\quad \rho_5\equiv\rho_1,\quad u_5\equiv u_1\ ),\end{gathered} \tag{1.1}$$

from where

$$\rho_{(i)}'=k_i=\frac{\rho_{i+1}-\rho_i}{u_{i+1}-u_i}. \tag{1.1'}$$

Dropping index k, let's designate with $\psi_{(i)}$ $(i=1,2,3,4)$ the values of the function ψ on the sides $A_1A_2,\dots,A_4A_1$ respectively. If we replace $\lambda(u)$ with $\psi_{(i)}(u)$ at (0.3) according to (1.1), we can see that the functions $\psi_{(i)}$ are linear, i.e.

$$\psi_{(i)}=M_iu+N_i \quad (i=1,2,3,4) \tag{1.2}$$

Supposing that the functions $\psi_{(i)}(u)$ are continuous at the points $u=\sigma$ of the meridian $\rho=\rho(u)$, where $\rho'(\sigma-0)\neq\rho'(\sigma+0)$, we get the system

$$\begin{aligned}\psi_{(1)}(u_1)&=\psi_{(4)}(u_1)=\psi_{(41)}(u_1)\\ \psi_{(2)}(u_2)&=\psi_{(1)}(u_2)=\psi_{(12)}(u_2)\\ \psi_{(3)}(u_3)&=\psi_{(2)}(u_3)=\psi_{(23)}(u_3)\\ \psi_{(4)}(u_4)&=\psi_{(3)}(u_4)=\psi_{(34)}(u_4)\end{aligned} \tag{1.3}$$

According to (1.2) we have the system

$$
\begin{aligned}
M_1u_1 + N_1 &= M_4u_1 + N_4\\
M_2u_2 + N_2 &= M_1u_2 + N_1\\
M_3u_3 + N_3 &= M_2u_3 + N_2\\
M_4u_4 + N_4 &= M_3u_4 + N_3
\end{aligned}
\tag{1.4}
$$

i.e., if we consider this system as a system on N_i:

$$
\begin{array}{cccccl}
N_1 & & & -N_4 & = & -M_1u_1 + M_4u_1\\
N_1 & -N_2 & & & = & -M_1u_2 + M_2u_2\\
 & N_2 & -N_3 & & = & -M_2u_3 + M_3u_3\\
 & & N_3 & -N_4 & = & -M_3u_4 + M_4u_4
\end{array}
\tag{1.5}
$$

At the apeces of the meridian the condition (0.5) gives the equations:

$$
\begin{aligned}
A_1:&\quad \rho_1(M_1 - M_4) + (k^2-1)(M_1u_1 + N_1)(k_1 - k_4) = 0\\
A_2:&\quad \rho_2(M_2 - M_1) + (k^2-1)(M_2u_2 + N_2)(k_2 - k_1) = 0\\
A_3:&\quad \rho_3(M_3 - M_2) + (k^2-1)(M_3u_3 + N_3)(k_3 - k_2) = 0\\
A_4:&\quad \rho_4(M_4 - M_3) + (k^2-1)(M_4u_4 + N_4)(k_4 - k_3) = 0
\end{aligned}
$$

or:

$$
\begin{aligned}
[\rho_1 + (k^2-1)u_1(k_1-k_4)]M_1 - \rho_1M_4 + (k^2-1)(k_1-k_4)N_1 &= 0\\
-\rho_2M_1 + [\rho_2 + (k^2-1)u_2(k_2-k_1)]M_2 + (k^2-1)(k_2-k_1)N_2 &= 0\\
-\rho_3M_2 + [\rho_3 + (k^2-1)u_3(k_3-k_2)]M_3 + (k^2-1)(k_3-k_2)N_3 &= 0\\
-\rho_4M_3 + [\rho_4 + (k^2-1)u_4(k_4-k_3)]M_4 + (k^2-1)(k_4-k_3)N_4 &= 0
\end{aligned}
\tag{1.6}
$$

Necessary and surfficient condition for the compatibility of system (1.5) is $rank\, M = rank\, P$, where M is the matrix of the system and P is extended matrix of the system.In order to explore the system, we are making elementary transformations of the matrices M and P.

According to (1.5) :

$$
P = \begin{array}{c}
\begin{array}{cccc} N_1 & N_2 & N_3 & N_4 \end{array}\\
\left[\begin{array}{cccc|c}
1 & 0 & 0 & -1 & -M_1u_1 + M_4u_1\\
1 & -1 & 0 & 0 & -M_1u_2 + M_2u_2\\
0 & 1 & -1 & 0 & -M_2u_3 + M_3u_3\\
0 & 0 & 1 & -1 & -M_3u_4 + M_4u_4
\end{array}\right]
\end{array}
\tag{1.7}
$$

Applying Gauss'algorithm for matrix P, let's realize following elementary transformations successively:

1) $-I \to II$, 2) $II \to III$, 3) $III \to IV$,

which means: 1) the first row is transcribed, it's elements are multiplied with -1 and added to corresponding elements of the second row,

2) the elements of the second row obtained in 1) we add to the corresponding elements of the third row , etc.. Thus we obtain

$$(1.8)\qquad P \sim \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_4 \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & -1 & \vdots & m_1 \\ 0 & -1 & 0 & 1 & \vdots & m_2 \\ 0 & 0 & -1 & 1 & \vdots & m_3 \\ 0 & 0 & 0 & 0 & \vdots & m_4 \end{array}\right] \end{array}$$

where

$$\begin{aligned} m_1 &= -M_1u_1 + M_4u_1 \\ m_2 &= M_1u_1 - M_4u_1 - M_1u_2 + M_2u_2 \\ m_3 &= M_1u_1 - M_4u_1 - M_1u_2 + M_2u_2 - M_2u_3 + M_3u_3 \\ m_4 &= M_1u_1 - M_4u_1 - M_1u_2 + M_2u_2 - M_2u_3 + M_3u_3 - M_3u_4 + M_4u_4 \end{aligned}$$

Hence, the system is compatible if

$$m_4 = M_1(u_1 - u_2) + M_2(u_2 - u_3) + M_3(u_3 - u_4) + M_4(u_4 - u_1) = 0.$$

When $u_i = u_{i+1}$ $(i = 1, 2, 3, 4, u_5 = u_1)$ the meridian contains a side which is orthogonal on the axis of rotation, generated surface contains a plane part and it is non rigid (see[4]). We omit this case in following consideration.

Suppose that $u_4 \neq u_1$. Then

$$(1.9)\qquad M_4 = \frac{1}{u_1 - u_4}[(u_1 - u_2)M_1 + (u_2 - u_3)M_2 + (u_3 - u_4)M_3].$$

Reduced system (according to (1.8)) is:

$$(1.10)\qquad \begin{aligned} N_1 - N_4 &= -M_1u_1 + M_4u_1 \\ -N_2 + N_4 &= M_1(u_1 - u_2) + M_2u_2 - M_4u_1 \\ -N_3 + N_4 &= M_1(u_1 - u_2) + M_2(u_2 - u_3) + M_3u_3 - M_4u_1 \end{aligned}$$

From (1.9,10) we have

$$(1.11)\qquad \begin{aligned} N_1 &= N_4 + \frac{u_1(u_4 - u_2)}{u_1 - u_4}M_1 + \frac{u_1(u_2 - u_3)}{u_1 - u_4}M_2 + \frac{u_1(u_3 - u_4)}{u_1 - u_4}M_3 \\ N_2 &= N_4 + \frac{u_4(u_1 - u_2)}{u_1 - u_4}M_1 + \frac{u_2u_4 - u_1u_3}{u_1 - u_4}M_2 + \frac{u_1(u_3 - u_4)}{u_1 - u_4}M_3 \\ N_3 &= N_4 + \frac{u_4(u_1 - u_2)}{u_1 - u_4}M_1 + \frac{u_4(u_2 - u_3)}{u_1 - u_4}M_2 + \frac{u_4(u_3 - u_1)}{u_1 - u_4}M_3 \end{aligned}$$

By the equations (1.9,11) unknowns M_4, N_1, N_2, N_3 are expressed by M_1, M_2, M_3 and N_4. Substituting (1.9) and (1.11) at (1.6) and designating

$$u_i - u_j = u_{ij}$$
$$k_i - k_j = k_{ij} \tag{1.12}$$

we get the system

$$\begin{aligned}
&[\rho_1 u_{24} + (k^2-1)k_{14}u_1u_{12}]M_1 + [(k^2-1)k_{14}u_1 - \rho_1]u_{23}M_2 + \\
&\quad + [(k^2-1)k_{14}u_1 - \rho_1]u_{34}M_3 + (k^2-1)k_{14}u_{14}N_4 = 0 \\
&[\rho_2 u_{41} + (k^2-1)k_{21}u_{12}u_4]M_1 + [\rho_2 u_{14} + (k^2-1)k_{21}u_{23}u_1]M_2 + \\
&\quad + (k^2-1)k_{21}u_{34}u_1M_3 + (k^2-1)k_{21}u_{14}N_4 = 0 \\
&(k^2-1)k_{32}u_{12}u_4M_1 + [\rho_3 u_{41} + (k^2-1)k_{32}u_{23}u_4]M_2 + \\
&\quad + [\rho_3 u_{14} + (k^2-1)k_{32}u_{34}u_1]M_3 + (k^2-1)k_{32}u_{14}N_4 = 0 \\
&[\rho_4 u_{12} + (k^2-1)u_{12}k_{43}u_4]M_1 + [\rho_4 u_{23} + (k^2-1)u_{23}k_{43}u_4]M_2 + \\
&\quad + [\rho_4 u_{31} + (k^2-1)u_{34}k_{43}u_4]M_3 + (k^2-1)u_{14}k_{43}N_4 = 0.
\end{aligned} \tag{1.13}$$

Necessary and sufficient condition for this system of linear homogeneous equations to have nontrivial solutions is the *rank* of matrix

$$H = \begin{array}{c} \begin{array}{cccc} N_4 & M_3 & M_2 & M_1 \end{array} \\ \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \end{array} \tag{1.14}$$

of the system to be less then 4. We have to find a condition under which it is valid. According to (1.13) we have

$$\begin{aligned}
&A_{11} = (k^2-1)k_{14}u_{14} && A_{12} = \rho_1 u_{43} + (k^2-1)k_{14}u_{34}u_1 \\
&A_{13} = \rho_1 u_{32} + (k^2-1)k_{14}u_{23}u_1 && A_{14} = \rho_1 u_{24} + (k^2-1)k_{14}u_{12}u_1 \\
&A_{21} = (k^2-1)k_{21}u_{14} && A_{22} = (k^2-1)k_{21}u_{34}u_1 \\
&A_{23} = \rho_2 u_{14} + (k^2-1)k_{21}u_{23}u_1 && A_{24} = \rho_2 u_{41} + (k^2-1)k_{21}u_{12}u_4 \\
&A_{31} = (k^2-1)k_{32}u_{14} && A_{32} = \rho_3 u_{14} + (k^2-1)k_{32}u_{34}u_1 \\
&A_{33} = \rho_3 u_{41} + (k^2-1)k_{32}u_{23}u_4 && A_{34} = (k^2-1)k_{32}u_{12}u_4 \\
&A_{41} = (k^2-1)k_{43}u_{14} && A_{42} = \rho_4 u_{31} + (k^2-1)k_{43}u_{34}u_4 \\
&A_{43} = \rho_4 u_{23} + (k^2-1)k_{43}u_{23}u_4 && A_{44} = \rho_4 u_{12} + (k^2-1)k_{43}u_{12}u_4
\end{aligned} \tag{1.14'}$$

Evidently,it is always $k_{ii+1} \neq 0$,as on contrary the meridian will not be quadrangular. Applying at the same time following operations to the matrix (1.14)

$$I\frac{k_{12}}{k_{14}} \to II, \qquad I\frac{k_{23}}{k_{14}} \to III, \qquad II\frac{k_{34}}{k_{21}} \to IV,$$

we obtain

$$H \sim [B_{ij}], \tag{1.15}$$

where

$$B_{1j} = A_{1j}, B_{21} = 0, B_{22} = \rho_1 u_{43}\frac{k_{12}}{k_{14}}, B_{23} = \rho_1 u_{32}\frac{k_{12}}{k_{14}} + \rho_2 u_{14}$$
$$B_{24} = \rho_1 u_{24}\frac{k_{12}}{k_{14}} + \rho_2 u_{41} + (k^2-1)u_{12}u_{14}k_{12}, B_{31} = 0$$
$$B_{32} = \rho_1 u_{43}\frac{k_{23}}{k_{14}} + \rho_3 u_{14}, B_{33} = \rho_1 u_{32}\frac{k_{23}}{k_{14}} + \rho_3 u_{41} + (k^2-1)u_{23}k_{23}u_{14},$$
$$B_{34} = \rho_1 u_{24}\frac{k_{23}}{k_{14}} + (k^2-1)u_{12}k_{23}u_{14}, B_{41} = 0, B_{42} = \rho_4 u_{31} + (k^2-1)u_{34}u_{14}k_{34}$$
$$B_{43} = \rho_2 u_{14}\frac{k_{34}}{k_{21}} + \rho_4 u_{23} + (k^2-1)u_{23}k_{34}u_{14}, B_{44} = \rho_2 u_{41}\frac{k_{34}}{k_{21}} + \rho_4 u_{12}.$$

Further, we apply the operations

$$II\frac{k_{41}}{\rho_1 u_{43}k_{12}}(\rho_1 u_{43}\frac{k_{23}}{k_{14}} + \rho_3 u_{14}) \to III$$
$$II\frac{k_{41}}{\rho_1 u_{43}k_{12}}[\rho_4 u_{31} + (k^2-1)u_{34}u_{14}k_{34}] \to IV$$

and obtain

$$H \sim [C_{ij}], \tag{1.16}$$

where

$$C_{1j} = B_{1j} = A_{1j}, C_{2j} = B_{2j}, C_{31} = C_{32} = 0$$
$$C_{33} = \rho_2 u_{14}\frac{k_{32}}{k_{12}} + \rho_3\frac{u_{14}u_{24}}{u_{43}} + \frac{\rho_2\rho_3(u_{14})^2k_{41}}{\rho_1 u_{43}k_{12}} + (k^2-1)u_{23}u_{14}k_{23},$$
$$C_{34} = \rho_2 u_{41}\frac{k_{32}}{k_{12}} + \rho_3 u_{41}\frac{u_{24}}{u_{43}} + \frac{\rho_3(k^2-1)u_{12}(u_{14})^2k_{41}}{\rho_1 u_{43}} + \frac{\rho_2\rho_3(u_{14})^2k_{14}}{\rho_1 u_{43}k_{12}},$$
$$C_{41} = C_{42} = 0$$
$$C_{43} = \rho_2 u_{14}\frac{k_{34}}{k_{12}} + \rho_4 u_{23}\frac{u_{41}}{u_{43}} + \frac{\rho_2\rho_4 u_{14}u_{31}k_{41}}{\rho_1 u_{43}k_{12}} + \frac{\rho_2(k^2-1)(u_{14})^2k_{14}k_34}{\rho_1 k_{12}} \tag{1.16'}$$
$$C_{44} = \frac{\rho_4 u_{14}u_{23}}{u_{43}} + \frac{\rho_2 u_{41}k_{34}}{k_{21}} + \frac{\rho_2(k^2-1)(u_{14})^2k_{41}k_{34}}{\rho_1 k_{12}} +$$
$$+\frac{\rho_4(k^2-1)u_{12}u_{14}u_{31}k_{41}}{\rho_1 u_{43}} + \frac{\rho_2\rho_4 u_{41}u_{31}k_{41}}{\rho_1 u_{43}k_{12}} +$$
$$+\frac{(k^2-1)^2u_{12}(u_{14})^2k_{14}k_{34}}{\rho_1} + (k^2-1)u_{24}u_{14}k_{34}.$$

By transformation $III\left(-\frac{C_{43}}{C_{33}}\right) \to IV$ the matrix (1.16) take a form

$$H \sim [D_{ij}], \tag{1.17}$$

where

$$\text{(1.17')}\quad \begin{aligned} &D_{1j}=C_{1j}=B_{1j}=A_{1j}, \quad D_{2j}=C_{2j}=B_{2j}, \quad D_{3j}=C_{3j} \\ &D_{41}=D_{42}=D_{43}=0, \quad D_{44}=-\frac{C_{43}C_{34}}{C_{33}}+C_{44}=\frac{1}{C_{33}}(C_{33}C_{44}-C_{43}C_{34}), \end{aligned}$$

and C_{ij} are given by (1.16'). The rank of the matrix N will be less then 4 for $D_{44}=0$,i.e. $C_{33}C_{44}-C_{43}C_{34}=0$,what gives

$$\text{(1.18)}\quad \begin{aligned} &[\rho_1\rho_2u_{43}k_{32}+\rho_1\rho_3u_{24}k_{12}+\rho_2\rho_3u_{14}k_{41}+\rho_1(k^2-1)u_{23}u_{43}k_{12}k_{23}]\times \\ &[\rho_4u_{12}u_{31}k_{41}+(k^2-1)u_{12}u_{43}u_{14}k_{14}k_{34}+\rho_1u_{43}u_{24}k_{34}]- \\ &-(\rho_1u_{23}u_{43}k_{23}+\rho_3u_{12}u_{14}k_{41})\times \\ &[\rho_1\rho_2u_{34}k_{34}+\rho_1\rho_4u_{32}k_{12}+\rho_2\rho_4u_{31}k_{41}+\rho_2(k^2-1)u_{14}u_{43}k_{14}k_{34}]=0, \end{aligned}$$

where u_{ij}, k_{ij} are given by (1.12) and (1.1').
Thus, we have

Theorem. *Necessary and sufficient condition for infinitesimal deformations of a toroid rotational surface,which is generated by a quadrangular meridian with apeces $A_i(u_i,\rho_i)$ ($\rho_i>0$, $u_{i+1}\neq u_i$, $u_5\equiv u_1$, $i=1,2,3,4$), around the Ou axis is the relation (1.18) where u_{ij}, k_{ij} are given by (1.12) and (1.1').*

Remark.. *If we apply (1.18) to the quadrangle of Belov $A_1(-1,b)$, $A_2(0,b+c_1)$, $A_3(1,b)$, $A_4(0,b-c_2)$ we obtain the relation $1/c_2-1/c_1=k^2/b$, which Belov obtained in other manner. So, the previous theorem is a generalization of the result of Belov.*

2. Determination of the field of infinitesimal deformation

Above applied method makes possible to determine the field of infinitesimal deformation.From (1.17) one obtains reduced system

$$\text{(2.1)}\quad \begin{aligned} &D_{11}N_4+D_{12}M_3+D_{13}M_2+D_{14}M_1=0 \\ &D_{22}M_3+D_{23}M_2+D_{24}M_1=0 \\ &D_{33}M_2+D_{34}M_1=0, \end{aligned}$$

from where

$$\text{(2.2a)}\quad M_2=-\frac{D_{34}}{D_{33}}M_1$$

$$\text{(2.2b)}\quad M_3=\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)M_1$$

$$\text{(2.2c)}\quad N_4=\left[-\frac{D_{12}}{D_{11}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)+\frac{D_{13}D_{34}}{D_{11}D_{33}}-\frac{D_{14}}{D_{11}}\right]M_1$$

Further, from (1.9) we have

$$\text{(2.2d)}\quad M_4=\frac{1}{u_{14}}\left[u_{12}-\frac{u_{23}D_{34}}{D_{33}}+u_{34}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)\right]M_1,$$

$$N_1=\left\{\left[-\frac{D_{12}}{D_{11}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)+\frac{D_{13}D_{34}}{D_{11}D_{33}}-\frac{D_{14}}{D_{11}}\right]+\right.$$

$$\tag{2.2e}\left.+\frac{u_1u_{34}}{u_{14}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)-\frac{u_1u_{23}D_{34}}{u_{14}D_{33}}+\frac{u_1u_{12}}{u_{14}}-u_1\right\}M_1,$$

$$N_2=\left\{\left[-\frac{D_{12}}{D_{11}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)+\frac{D_{13}D_{34}}{D_{11}D_{33}}-\frac{D_{14}}{D_{11}}\right]-u_1+u_2+\right.$$

$$\tag{2.2f}\left.+\frac{D_{34}u_2}{D_{33}}+\frac{u_1u_{12}}{u_{14}}-\frac{u_1u_{23}D_{34}}{u_{14}D_{33}}+\frac{u_1u_{34}}{u_{14}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)\right\}M_1,$$

$$N_3=\left\{\left[-\frac{D_{12}}{D_{11}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)+\frac{D_{13}D_{34}}{D_{11}D_{33}}-\frac{D_{14}}{D_{11}}\right]-u_1+u_2+\right.$$

$$\tag{2.2g}+\frac{D_{34}u_2}{D_{33}}-\frac{D_{34}u_3}{D_{33}}-\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)u_3+\frac{u_1u_{12}}{u_{14}}-$$

$$\left.-\frac{D_{34}u_1u_{23}}{D_{33}u_{14}}+\frac{u_1u_{34}}{u_{14}}\left(\frac{D_{23}D_{34}}{D_{22}D_{33}}-\frac{D_{24}}{D_{22}}\right)\right\}M_1.$$

By the equations $(2.2.a-g)$ we expressed $M_i,\ N_i\ (i=1,2,3,4)$ by M_1 (indefinit const.). Further, we obtain $\psi_{(i)}(u)$ on the base of (1.2).In this manner, we get the field $\overline{z}(u,v)$ of infinitesimal deformations, given by (0.2).

Finaly, I wish to thank prof. Milica Ilić-Dajović for introducing me into the matter treated in the present work.

References

[1] К.М. Белов, *О бесконечно малых изгибаниях торообразной поверхности вращения*, Сиб. мат. журнал, т Н3 (IX)(1968), 490-494.
[2] С.Э. Кон-Фоссен, Некоторые вопросы дифференциальной геометрии в целом, Физматгиз, Москва, 1959.
[3] В.Ф. Каган, Основы теории поверхностей, Т.И, ОГИЗ, Москва-Ленниниград, 1947.
[4] Н.В. Ефимов, *Качественные вопросы теории деформаций поверхностей*, УМН 3,2(1948), 47-158.

Faculty of Civil Engineering, Beogradska 14, 18000 Niš, Yugoslavia

COMPUTER SCIENCE

FILOMAT (Niš) 9:2 (1995), 207–231
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

FRACTALS AND THEIR APPLICATIONS IN COMPUTER GRAPHICS

Ljubiša M. Kocić

ABSTRACT. *The paper presents elements of fractal geometry and its application in computer graphics and geometric modeling. A connection with chaos dynamics, mostly from historical angle of view, is stressed. Two fundamental algorithms for computing fractal attractors are described. Barnsley affine iterated function systems (IFS) are described as means of constructing deterministic fractals. It is pointed out how to introduce parameters in IFS, via Bernstein polynomials, to produce different natural forms. Variety of applications: in animation, data compressing, rendering objects and modeling phenomena in physics and biology are described.*

1. Introduction: Physics and History

At the end of 19. century, physicists considered Physics as a mainly finished discipline, with everything of any importance in the field being already explained and known. Everything, except a couple of unimportant loose ends. Making efforts to remove them, Schrödinger discovered quantum mechanics, while Einstein invented the relativity theory. The new physics required new mathematical techniques, and before all, new geometry. So, the non-euclidean and projective geometries became topics of interest. Just when it seemed that relativity theory would finally finish the job of completing the great book of nature being opened by the Newton physics, the strange properties of simple oscillators have been noticed. Actually, under certain circumstances, they began to exhibit irregular, chaotic behaviour. Thus, in the last quarter of 20-th century, scientists have faced the new revolution called *chaos*.

1991 *Mathematics Subject Classification.* 58F13.
Key words and phrases. Fractal sets, chaos, computer graphics.
This research was partly supported by Science Fund of Serbia, grant number 0401A, through Matematički institut

The word 'chaos' ($\chi\alpha o\varsigma$) stems from the old Greek *hainō* which means 'open widely'. In Aristotel's works, 'chaos' is used to denote an 'empty space'. Later, in the history, this word becomes the synonym for 'mess' and 'lack of order'.

Chaos cannot be successfully described neither by Euclid geometry nor by non-euclidean or projective geometries. Except in some crystal forms, nature rarely exhibits regularity and geometric order. Natural forms and structures are irregular and chaotic: clouds, moss, trees, coastlines, feathers, rocks, surface of the sea, network of neurons etc. These are forms that *fractal geometry* deals with.

Phenomenology of chaos appears in, at least three planes. The plane of *morphology* is the most accessible for studying due to huge amount the empirical, factual material which is collected during the time. The plane of *logic* is much more complicated, so that only the partial breakthrough has been done (for ex. in information theory). The *causality* plane is still in domain of hypothesis and till now, it is beyond the experimental confirmation. Typical example is the hypothetical 'quantum chaos'.

Much earlier before the physicists started coping with chaos, there were hints of it in mathematical thinking.

So, at the beginning of 19. century Laplace introduced the new discipline to describe unruliness and disorder – *probability theory.* Contrary to deterministic theories, probability theory states that future depends randomly on the past.

Then, Weierstrass defined the function

$$f(x) = \sum_{i=1}^{\infty} \frac{\sin(\lambda^i x)}{\lambda^{\varepsilon i}} \ , \ x \in [0, 2\pi] \ ,$$

where $\lambda > 1$ and $0 < \varepsilon < 1$ are real parameters. Being continuous but nowhere differentiable [12 , p. 53], this function was unlike the things that mathematicians had ever seen before. Although bounded, its graph has infinite length (Figure 1-a).

Actually, Weierstrass function belongs to the 'fractal' class $\mathbf{C}^{\varepsilon}$. *Topological dimension* of this curve is $\mathrm{Dim}_T = 1$, while its *Hausdorff* (also called *Hausdorff–Besicovitch* or *geometric*, see Section 4.) dimension is apparently $\mathrm{Dim}_H = 2 - \varepsilon$, but this has not been proved rigorously [15].

The most famous fractal set is probably the Cantor set which is equipotent to the interval $[0, 1]$ but of zero measure. Its Hausdorff dimension is $\mathrm{Dim}_H = \ln 2 / \ln 3 = 0.6309...$ and topological dimension $\mathrm{Dim}_T = 0$ (Fig. 1-b). So, the Cantor set can not be reduced to a set of isolated points in which case its H-dimension should be $\mathrm{Dim}_H = 0$.

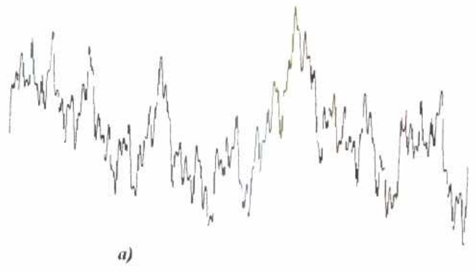
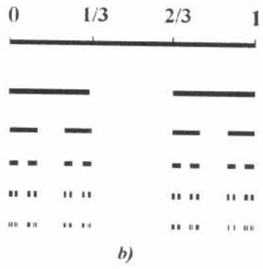



Figure 1. a) The Weierstrass function for $\lambda = 1.9$, $\varepsilon = 0.3$; b) Generating of the Cantor set

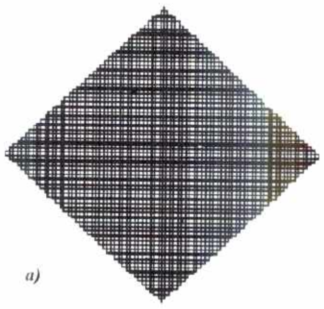
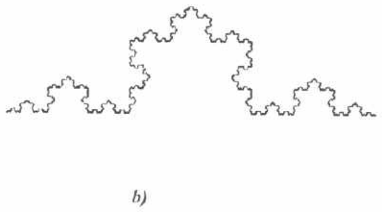



Figure 2. a) Peano curve; b) von Koch curve

In 1890. Peano [24] published a construction of a curve that fills in the unit square without self-intersections. *Peano* curve has Hausdorff dimension $\mathrm{Dim}_H = 2$, while $\mathrm{Dim}_T = 1$ (Figure 2-a). A year later, Hilbert [16] announced its own construction of such, so called *space filling curve.*

The next important construction is the von Koch curve from 1904 ([19]), known from calculus textbooks as an example of a simple continuous curve without tangents. Its geometrical dimension is $\mathrm{Dim}_H = \ln 4 / \ln 3 = 1.2619\ldots$, ($\mathrm{Dim}_T = 1$), see Figure 2-b.

These 'early birds' were named 'monsters' and 'pathological cases' by other mathematicians, and they refused to deal with them at all. In spite of lacking the tools (like modern computers), first systematic study of chaos and irregular structures starts with works of Kowalewska and continues with works of Lyapunov.

Making efforts to describe the chaotic phenomena as accurate as it is possible, Poincaré introduces *topology* and considers the physical chaos on model of *orbits* of mapping $f : \mathbf{X} \to \mathbf{X}$:

$$x, \ f(x) \ f^2(x) \ \ldots, \quad x \in \mathbf{X},$$

and intersection of dynamic trajectory in m-dimensional *phase space* and transversal $(m-1)$-dimensional hyperplane, now known as the *Poincaré section.*

The ideas of Poincaré have been further developed by Gaston Julia and Pierre Fatou during 20-ies of this century. Their work drew attention of physicists due to its applicability to the simple dynamical systems called *oscillators.* A typical model is a pendulum, but there were other interesting oscillators.

So, B. van der Pol in Holland studied the oscillating model of an electronic tube, while the mathematician V. Arnold made detailed analysis of the mathematical model of the human heart, which is an oscillator by himself.

In 1950–es, ecologists have studied so called logistic equation which describes variations in population of different zoological forms

$$x_{n+1} = r\,x_n(1-x_n)\,, \quad n \in N\,, \quad x_1 \in \mathbf{R}\,, \tag{1}$$

where $r \in \mathbf{R}$ is a parameter. The Sequence $\{x_n\}$ represents the orbit of a simple quadratic map

$$f : \ x \mapsto r\,x(1-x)\,, \quad r \in \mathbf{R}\,,$$

which exhibits unexpected dynamical properties. For $r < 3$ the corresponding dynamical system $(f, \mathbf{R})$ is stable, ie. f is a contractive mapping with a unique fixed point $x = lim_{n\to+\infty} x_n$. For $3 < r < 3.5699456\ldots$, the system has periodical behaviour with successively doubling of the period, whilst for the bigger values of r it goes to chaos. The graph of x as function of r is known as *bifurcation diagram* The sequence of branching points (bifurcations) $\{r_n\}$ has an accumulating point $3.5699456\ldots$, which marks the limit of stability. The ratio $\Delta r_n/\Delta r_{n+1} = 4.6692016091\ldots$ is invariant for all mappings with 'parabolic' maximum and is referred as the *Figenbaum number.*

This type of mapping describes 'explosions' in biological population like the famous locust flood every seven years, unexpected starting and spreading

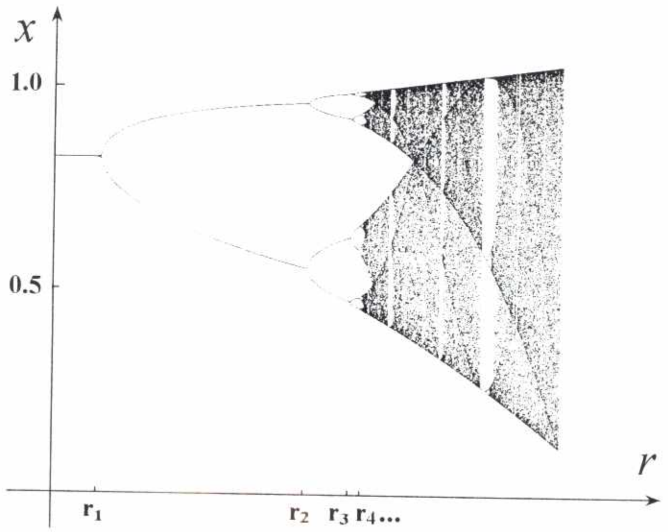


Figure 3. The bifurcation diagram for logistic equation.

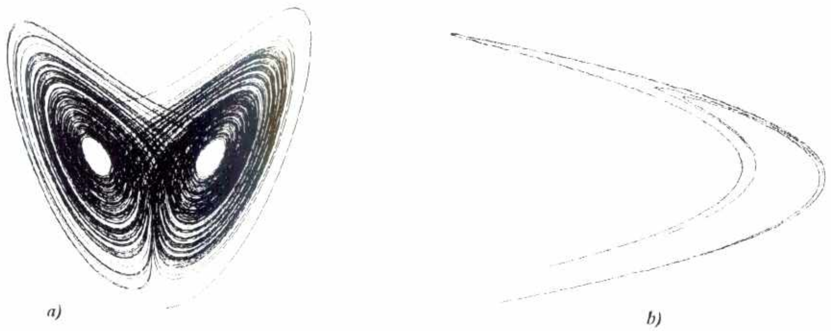


Figure 4. a) Lorentz attractor; b) Hénon attractor

of diseases, but it also describes fluctuation of the money value on the market, where chaos means the economical breakdown.

In 1962., Edward Lorenz made a mathematical model of meteorological variations of weather, being described by the set of differential equations

$$x'(t) = a(y - x)\ , \quad y'(t) = bx - y - xz\ , \quad z'(t) = xy - cz\ , \tag{2}$$

in time domain. It comes out that the model gives good results, but it is very sensitive on initial conditions for some parameter values. It implies its sensitivity
on the error which is accumulated during the numerical integration process. The exact solution of (2) is a trajectory in $\mathbf{R}^3$ which has very complicated form. (For its XY-projection, see Figure 4-a). If converging, the numerical solution approaches (in the Hausdorff metric) this trajectory, the reason led Lorenz in name it *strange attractor.* In 1963., Michelle Hénon, a French astronomer, used Poincaré's ideas and include chaos in mechanical model of stars motion. This helped him to overcome a many years standstill in the problem, caused by the classical newtonian approach. The Hénon model can be reduced to the system of difference equations

$$x_{n+1} = 1 - \alpha x_n^2 + y_n \ , \ y_{n+1} = \beta x_n \ ,$$

whose attractor, for $\alpha = 1.4$ and $\beta = 0.3$ has a remarkable self–similar '3–2–1 pattern' structure (Figure 4-b).

2. Deterministic fractals

Two important observations lead to the fractal geometry.

1° The Nature is permeated with something that scientists call *deterministic chaos*. This is the common name of the behaviour of the huge number of fairly simple physical systems that are governed by deterministic law, but, in spite of this, they behave unpredictably.

2° There is a *hierarchical structure* in the Universe. Details resembles to the whole: it can be easily noticed in forms of crystals and plants, in the relief of Earth surface, in the structure of stellar clusters and in variation of market prices.

During sixties, the physics of chaos becomes more and more attractive field. The remarkable oscillatory chemical reaction of *Belousov-Žabotinski* is explained by using chaos. It was discovered that there are three 'scenarios' for a system to pass to chaos. These types can be described by purely geometrical language, depending on the type of bifurcation of dynamical system.

In seventies, Benoit Mandelbrot from IBM-a, made, by the help of computer, first fractal images. These are graphical 'portraits' of dynamics of simple mappings with astonishing degree of disorder, but this disorder was systematic and unusually complex. The most popular among these pictures is probably the *Mandelbrot set* (Figure 5). It represents the dynamical chart of mapping $z \to z^2 + C$, $z \in \mathbf{C}$, for fixed value of complex constant C. Orbits are given by the sequence $\{z_n\}$, which is the solution of difference equation

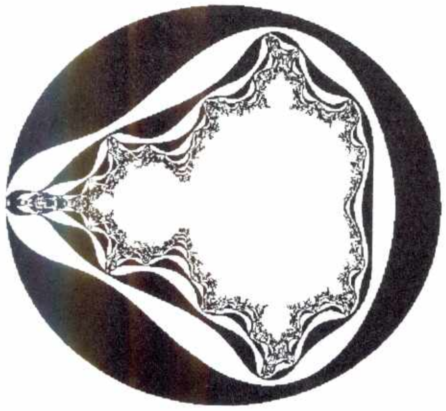


Figure 5. The Mandelbrot set.

$z_{n+1} = z_n^2 + C$, $z_0 = 0$. For a given value C, the behaviour of sequence $\{z_n\}$ has been examined. If it diverges, the point C in the complex plane are 'painted' in, for ex., black color. If converges, it will be painted in some lighter color, as lighter as faster the convergence is (in Fig. 5, this is white). The Mandelbrot set has an important role in fractal geometry, as for ex., circle in Euclid's geometry, and it is studied out exhaustively [5].

Mandelbrot coined the word "fractal" from lat. *fractus* which means a stone, broken and having irregular form. In collaboration with other scientists, he studies a large variety of phenomena being connected with fractals: stochastic form of a coastline and its relationship with Brownian motion, turbulence in fluids, statistical distribution of telephone calls, Nile floodings, branching neurons in the neural tissue etc. By the way, Mandelbrot noticed that fractal images possess aesthetical values.

In 1982. the fundamental Mandelbrot book [22] appeared. The new discipline was born. It includes fractal geometry as its most important part. After the book has been issued, the interest on fractal exploded. Many definitions of fractal sets and their dimensions appeared ([5],[7],[8], [11],[13],[15],[30]).

In this section, a large class of fractal sets will be introduced. This very class has application in computer graphics in modeling natural phenomena.

The notation $(\mathbf{X}, d)$ throughout the text will denote the complete metric space. Also, $\mathcal{H}(\mathbf{X})$ will denote the space whose points are compact subsets of $\mathbf{X}$.

Definition 1. A map $F : \mathbf{X} \to \mathbf{X}$ of a metric space $(\mathbf{X}, d)$ is a *Lipschitz map* if there is a number σ such that $d(F(x), F(y)) \leq \sigma d(x, y)$, for all $x, y \in \mathbf{X}$.

The least such number, $s(F) = \min\{\sigma\}$ is the *Lipschitz constant* of F. If $s(F) < 1$ then F is called a *contraction.*

Definition 2. Let $'\circ'$ be a usual composition of mappings. Then, $F^{\circ n}$ denotes n-th iteration of a mapping F, i.e.

$$F^{\circ n} = F(F^{\circ(n-1)}) \ , \quad F^1 = F \ .$$

Theorem 1 (Contraction principle). *If F is a contraction $\mathbf{X} \to \mathbf{X}$, then the sequence $\{n \mapsto F^{\circ n}(x)\}_{n=0}^{+\infty}$ converges to a fixed point $a \in \mathbf{X}$ of F. The fixed point is unique. Moreover, if s is the Lipschitz constant of F, then*

$$d(F^{\circ n}(x), a) \leq \frac{s^n}{1-s} d(F(x), x) \ .$$

Proof. For the proof, see any textbook in functional analysis. □

Definition 3. For any $x \in \mathbf{X}$ and $B \in \mathcal{H}(\mathbf{X})$, the distance from the point x to the set B is

$$d(x, B) = \min_{b \in B}\{d(x, b)\} \ .$$

Definition 4. For any $A, B \in \mathcal{H}(\mathbf{X})$, the distance from A to B is

$$\rho(A, B) = \max_{x \in A}\{d(x, B)\} \ .$$

It is easy to see that ρ is not symmetric, i.e. $\rho(A, B) \neq \rho(B, A)$, so ρ does not provide a metric on $\mathcal{H}(\mathbf{X})$.

Definition 5. For any $A, B \in \mathcal{H}(\mathbf{X})$, the Hausdorff distance between A and B (induced by the metric d) is

$$h(A, B) = \max\{\rho(A, B), \rho(B, A)\} \ .$$

Theorem 2. *The Hausdorff distance h is a metric on the space $\mathcal{H}(\mathbf{X})$. The space $(\mathcal{H}(\mathbf{X}), h)$ is a complete metric space.*

Proof. See [1]. □

Definition 6. *A (hyperbolic) function system* (IFS) is a set $S = \{\mathbf{X}; f_i\}_{i=1}^m$, where f_i is a contraction of $(\mathbf{X}, d)$ into itself. If s_i is a Lipschitz constant for f_i, then $s = \max_i\{s_i\}$ is the Lipschitz contractive constant for the IFS.

Theorem 3 (Hutchinson). *Let S, be an IFS, with contractive constant s. For $A \in \mathcal{H}(\mathbf{X})$, define $F(A) = \cup_{i=1}^{m} f_i(A)$. Then,*

$$h\{F(A),\ F(B)\} \leq sh(A,\ B)\ ,\ \ A, B \in \mathcal{H}(\mathbf{X})\ ,$$

Proof. See [17]. □

Definition 7. A set $A \in \mathcal{H}(\mathbf{X})$ is an attractor of IFS $\{\mathbf{X}, f_i, i = 1, \dots m\}$ if $F(A) = A$. Sometimes attractors are called *deterministic fractals.*

Theorem 4 (Hutchinson). *There is a unique closed bounded attractor A for S. Moreover, if B is any set from $\mathcal{H}(\mathbf{X})$, then*

$$A = \lim_{n \to \infty} F^{\circ n} B\ .$$

Proof. See [17]. □

3. Algorithms, affine IFS and Bernstein polynomials

The fractal geometry is a discipline of computer ages. Without computers, exploration of fractal sets would not be possible. For the mutual benefit, fractals contribute in picture synthesis as an mighty tool. There are many algorithms for computing and visualizing fractals, but all are variations of two basic ones:

a) *Hutchinson's algorithm* [10]. This algorithm is based on Theorem 4. It starts from an initial set $B \in \mathbf{R}^2$ and transforms by the IFS recursively until graphical details become smaller than a pixel. This algorithm is also known as *deterministic algorithm.*

b) *Algorithm of Barnsley and Demko (random algorithm* [1],[2],[3]). It uses a positive sequence $\{p_i\}_{i=1}^{m}$ of probabilities so that $\sum p_i = 1$, where p_i is probability of application of contraction f_i in given iteration. Choose a point $x_0 \in \mathbf{X}$ and then, for $n = 1, 2, \dots, m$, calculate

$$x_n = f_i(x_{n-1})\ ,$$

where index i is chosen randomly from $\{1, 2, \dots, m\}$ with probability p_i. This procedure forms the sequence $\{x_n,\ n = 0, 1, \dots\} \subset \mathbf{X}$ which approximates the attractor A of IFS. There are no precise rules for choosing the probabilities p_i, which allows flexibility in choosing the sequence $\{p_i\}$ which may be useful in modeling, as it will be shown in Section 4.

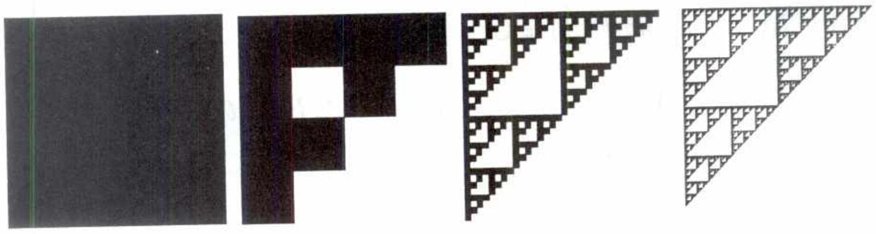


Figure 6. Four iterations of Hutchinson algorithm: Sierpinski gasket

An important class of deterministic fractals is defined by the IFS $\{\mathbf{R}^2; \phi_i\}$, (Euclidean metric), when ϕ_i are affine contractions, i.e.

$$\phi_i(\mathbf{x}) = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \mathbf{x} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} , \tag{3}$$

where $\mathbf{x} = [x\ y]^T \in \mathbf{R}^2$, and $a_i, b_i, c_i, d_i, e_i, f_i$ are real constants, chosen so that ϕ_i is a contraction. In this case, the probabilities p_i in Barnsley-Demko's algorithm can be calculated as

$$p_i = \frac{D_i}{\sum_{j=1}^{m} D_j} ,$$

where $D_i = |a_i c_i - b_i d_i|$ is the determinant of the matrix in (3).

It is hardly understandable how many different forms can be produced by an affine IFS. Let us see some examples.

1. *Sierpinski gasket.* It is a 'triangular extension' of Cantor set defined by the IFS $\{\mathbf{R}^2; \phi_1, \phi_2, \phi_3\}$, where $a_i = d_i = 0.5, b_i = 0, i = 1, 2, 3$, $c_1 = e_3 = 0.5$, $c_2 = c_3 = e_1 = f_1 = f_3 = 0$, $e_2 = 0.25$, $f_2 = \sqrt{3}/4$. Applying Hutchinson algorithm on the initial set–the unit square (leftmost in Figure 6), an approximation of the attractor is obtained through seven iterations (rightmost in Fig. 6). Figure 6 also shows third and fifth iteration. The same result will be obtained if any bounded initial set is taken. It is interesting that the Pascal triangle of binomial coefficients has fractal structure of Sierpinski triangle [31].

2. *Takagi function.* This is yet another fractal function that fits to the line of Weierstrass and von Koch construction. It was published in 1903 by Teiji Takagi [28]. It can be described by the affine IFS $(\mathbf{R}^2; \phi_1, \phi_2)$ so that

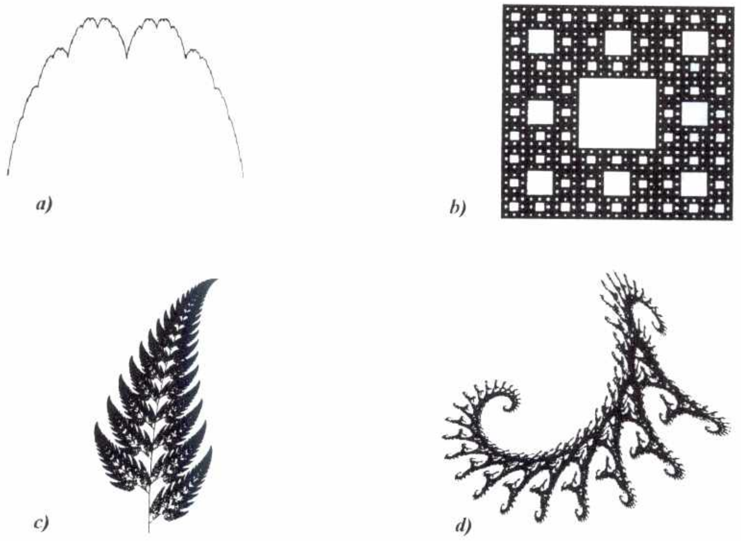


Figure 7. a) Takagi function; b) Sierpinski carpet; c) Barnsley fern; d) The dragon–like attractor

ϕ_1 and ϕ_2 are given by $a_1 = a_2 = c_1 = -c_2 = d_1 = d_2 = e_2 = f_2 = 0.5$, $b_1 = b_2 = e_1 = f_1 = 0$ with probabilities $p_1 = p_2 = 0.5$. The attractor, obtained by Barnsley-Demko algorithm is shown in Figure 7-a.

3. *Sierpinski carpet.* Yet another two-dimensional variation on Cantor theme. It is defined in $\mathbf{R}^2$ by eight affine transformations with coefficients: $a_1 = a_3 = a_6 = a_8 = -b_2 = b_4 = b_5 = -b_7 = c_2 = -c_4 = -c_5 = c_7 = d_1 = d_3 = d_6 = d_8 = e_2 = e_4 = e_5 = f_2 = f_5 = f_7 = 1/3$, $e_6 = e_8 = f_3 = f_8 = 2/3$, $e_7 = f_4 = 1$. Other entries are zero. The attractor is rendered by the random algorithm (Fig. 7-b).

4. *Barnsley fern.* An interesting fern-like attractor (Fig. 7-c) is found by M. Barnsley [1]. Four affine contractions are given by the coefficients

a	b	c	d	e	f	p
0.0	0.0	0	0.16	0.0	0.0	0.01
0.85	0.04	−0.04	0.85	0.0	1.6	0.85
0.2	−0.26	0.23	0.22	0.0	1.6	0.07
−0.15	0.28	0.26	0.24	0.0	0.44	0.07

The attractor, produced by random algorithm is shown in Figure 7-c.

5. *Dragon.* The dragon-like set (Fig. 7-d) is defined by IFS data

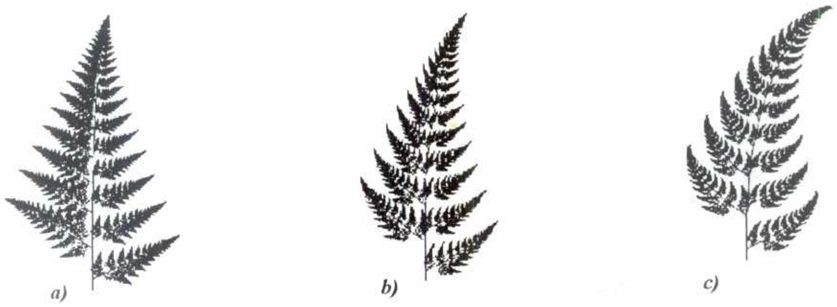


Figure 8. Wind in fractal plants: the parametric IFS

a	b	c	d	e	f	p
0.824074	0.281482	−0.212346	0.864198	−1.882290	−0.110607	0.787473
0.088272	0.520988	−0.463889	−0.377778	0.785360	8.095795	0.212527

It is of special importance to introduce one or several parameters into IFS, so that the form of attractor set depends on them. Parameters should be incorporated in such a way that affine mappings in IFS maintain their contractive properties for a whole range of parameter changing. Actually, the following theorem takes place.

Theorem 4 (Barnsley). *Let $(\mathbf{X}, d)$ be a metric space, and $\{\mathbf{X}; f_1, \dots, f_N\}$ be a hyperbolic IFS of contractivity s. Let f_n depend continuously on a parameter $p \in P$, where P is a compact metric space. Then, the attractor $A(p) \in \mathcal{H}(\mathbf{X})$ depends continuously on $p \in P$, with respect to the Hausdorff metric $h(d)$.*

Proof. See [1]. □

This theorem provides a way of controlling the shape of IFS in continuous way. It can be used very effectively in modeling motion of fractal objects. For example, the fern swung by the wind may be obtained by simple changing a parameter in IFS-fern data so to get a plant without wind (Figure 8-a), under breeze (Fig. 8-b) and stronger wind (Fig. 8-c). Such effects are especially important in animation.

An elegant way to introduce parameters in iterated function systems is to put one-variable real functions as IFS coefficients in (3). Author made some experiments using cubic Bernstein polynomials

$$B_i(t) = \binom{3}{i} t^i (1-t)^{3-i}\ ,\ t \in [0,1]\ , \tag{4}$$

the choice approved by the known property of the Bernstein polynomials to be bounded over the unit interval, i.e. $0 \leq B_i(t) \leq 1$. Further, numerical computation of Bernstein polynomials is fast enough through de Casteljau algorithm [20]. It is considered an IFS with only two mappings $\{\mathbf{R}^2; \phi_1, \phi_2\}$, with two parameters, t_1 and t_2. In the tables below the coefficients of four different IFS's are given.

IFS1

a	b	c	d	e	f
$b_0(t_1)$	$-b_1(t_2)$	$b_2(t_1)$	$b_0(t_1)$	0.0	0.35
$b_0(t_2)$	$b_1(t_2)$	$-b_2(t_2)$	$b_0(t_2)$	0.6	0.1

IFS2

a	b	c	d	e	f
$b_0(t_1)$	$-b_1(t_2)$	$b_2(t_2)$	$b_0(t_1)$	0.0	0.45
$b_0(t_2)$	$b_1(t_1)$	$-b_2(t_1)$	$b_0(t_2)$	0.6	0.1

IFS3

a	b	c	d	e	f
$b_0(t_1)$	$-b_2(t_2)$	$b_2(t_2)$	$b_0(t_1)$	0.5	0.0
$b_0(t_2)$	$b_1(t_1)$	$-b_1(t_1)$	$b_0(t_2)$	2.5	1.5

IFS4

a	b	c	d	e	f
$b_0(t_1)$	$-b_1(t_1)$	$b_2(t_1)$	$b_0(t_1)$	0.0	0.55
$-b_0(t_2)$	$b_1(t_2)$	$-b_2(t_2)$	$b_0(t_2)$	0.4	0.1

As it can be seen from Figure 9, in spite of using only two contractions, exciting results are obtained. For the attractor *'feather'* (a) it was used IFS1, with parameters $t_1 = 0.1, t_2 = 0.22$. The *'cloud formation'* (b) uses IFS2 with $t_1 = 0.18, t_2 = 0.22$; Next three attractors are produced using IFS3: *'star'* (c) with $t_1 = 0.042, t_2 = 0.75$, *'Nautilus spiral'* (d) with $t_1 = 0.05, t_2 = 0.45$ and *'sunflower seed'* (e) with $t_1 = 0.033, t_2 = 0.52$. Finally, a *'fir twig'* (f) is formed by IFS4 with $t_1 = 0.09, t_2 = 0.18$.

Continuous variation of parameters t_1 and t_2 will reflect in continuous changing of the forms of attractors. In this way, it is possible to animate the sequence when the 'Nautilus spiral' transforms into 'sunflower' by running t_1 from 0.05 to 0.033, and t_2, from 0.45 to 0.52.

4. Modeling of natural phenomena

Describing a way of reproducing the wind, and a variety of natural forms by using parametric IFS, opens the question of using fractal sets in modeling

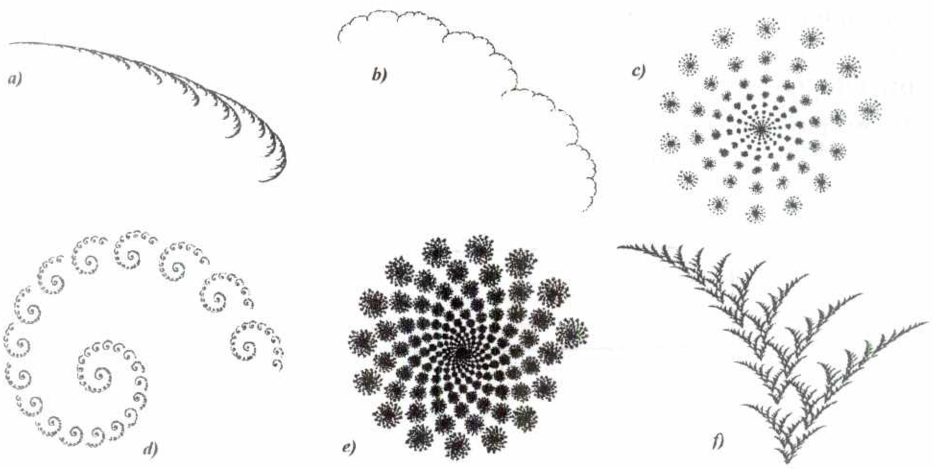


FIGURE 9. SIX ATTRACTORS GENERATED BY TWO-PARAMETER IFS CONTAINING BERNSTEIN POLYNOMIALS

wider class of forms and processes in the Universe. These forms are not easy to measure with classical Euclidean tools. So, a more sophisticated technique is developed. One of the important numbers associated with fractals is their Hausdorff dimension. The meaning of dimension is the 'density' with which the fractal set occupies the metric space in which it lies. It can be used for comparing fractals. It is an important parameter for modeling natural objects.

Definition 8. Let $K \in \mathcal{H}(\mathbf{X})$ be a nonempty set and $(\mathbf{X}, d)$ be a metric space. The *diameter* of K is

$$|K| = \sup_{x,y\in K} \{d(x,y)\} .$$

Definition 9. Let $\mathcal{K} = \{K_i\}_{i=0}^{+\infty}$ be a collection of sets in $\mathcal{H}(\mathbf{X})$ such that $0 < |K_i| \leq \varepsilon$, for each i. If $A \subset \cup_i K_i$, then $\mathcal{K}$ is ε-cover of A.

Definition 10. Let m be a positive integer and let A be a bounded subset of the metric space $(\mathbf{R}^m, d)$, where d is Euclidean metric. The function $p \mapsto \mu(A,p)$ defined as

$$\mu(A,p) = \sup_{\varepsilon>0}\{\inf\{\sum_{i=0}^{+\infty} |K_i|^p\}\} ,$$

where infimum is taken over all ε/covers of A, is called *Hausdorff p-dimensional measure* of A.

Definition 11. Let A be a bounded subset of the metric space $(\mathbf{R}^m, d)$. Then the real number $Dim_H(A)$ defined as

$$Dim_H(A) = \inf_{\mu(A,p)=0} \{p\},$$

is *Hausdorff dimension* of A. It is also called *Hausdorff–Besicovitch* or *geometrical dimension* of A.

Theorem 5. *Let A be a bounded subset of the space $(\mathbf{R}^m, d)$. $Dim_H(A)$ is a unique real number which satisfies $0 \le Dim_H(A) \le m$.*

Proof. See [1]. □

Hausdorff dimension of a typical fractal set may not be an integer number. This is 0.6309... for Cantor set, 1.2619... for von Koch curve, 1.5849... for Sierpinski gasket, 1.8927... for Sierpinski carpet or 2 for Peano curve. Fractal dimension may characterize type of relief and roughness of terrain or physical process. Hausdorff dimension of the coastline of Britain is ≈ 1.2 [18], while it is ≈ 1.5 for jet flame laboratory data [1]. It is possible to determine Hausdorff dimension of the chain of human DNA from genetic code [4], or for fractured metal surfaces [9]. Maybe painters or sculptors can be characterized by Hausdorff dimension of their masterpieces?

Practical determination of fractal dimension is not an easy task. It may depend on scaling. Mandelbrot gives an interesting example in [22] trying to answer to the question: 'What is the dimension of a ball of yarn?' From a great distance it is effectively a point, and appears zero dimensional; on approach it becomes a three-dimensional solid; moving closer discern the one-dimensional threads, which then become three dimensional again; the threads are again composed of fibers, etc. These different scaling regimes would produce rather extreme oscillations in a numerical estimate of dimension. Typically, when we are computing dimension we are interested in a given scaling range, but it may be very difficult to discern.

But there is a class of affine IFS whose attractors' dimension can be calculated without much trouble.

Definition 12. If the hyperbolic IFS $\{\mathbf{R}^m; \phi_i\ i = 1, \ldots, N\}$ has the following properties:

a) ϕ_i , $i = 1, \ldots, N$ are *similitudes*;

b) $\mu(\phi_i(B) \cap \phi_j(B)) = 0$, for $i \neq j$, and any $B \subset \mathbf{R}^m$,

then attractor A is *self-similar*.

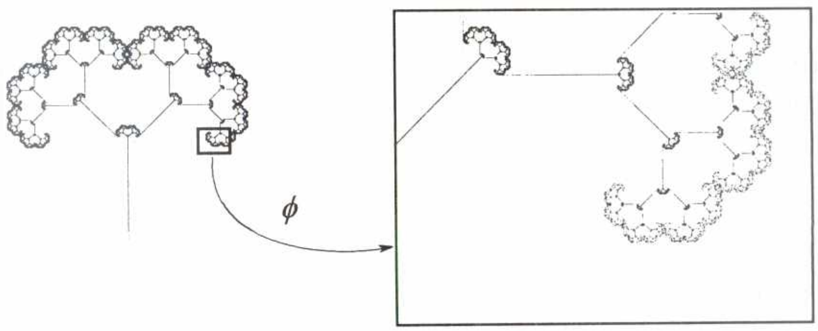


Figure 10. A self–similar set. Expanded area contains set that is identical to the whole set

Theorem 6. *Let A be self–similar attractor, generated by a hyperbolic IFS with s_i being a contractivity factor of ϕ_i. Then $D = Dim_H(A)$ is the unique solution of*

$$\sum_{i=1}^{N} |s_i|^D = 1, \quad D \in [0, m] \ .$$

Proof. See [1]. □

An example of self–similar fractal set is given in Figure 10. It means that a small portion of the set is identical to the whole set.

The property of self–similarity is important for modeling natural objects. Namely, natural scenes are organized in hierarchical structures. For example a forest is made of trees; a tree is a collection of boughs and limbs along a trunk; on each branch there are clusters of leaves; a leaf is filled with veins and covered with hairs. Similar hierarchy one can find in the structure of rocks, mountains, live forms... In every case, the object is built up from numerous near repetitions of some smaller structure. Although the natural entities have more complex kind of self–similarity, so called *statistical self–similarity*, the iterated function systems with similitudes
can be used for modeling approximations of such entities. Also, one can use some fractal set representing dynamics of some simple mapping, like the fractal set known as *Barnsley-3m*, being displayed in the left part of Figure 11. If specify the domain of mapping to be a rectangle denoted by 'A', the dynamical mapping will produce a magnified picture that resembles the 'wave' (right part of Fig. 11, above). Further multiplication reveals self–

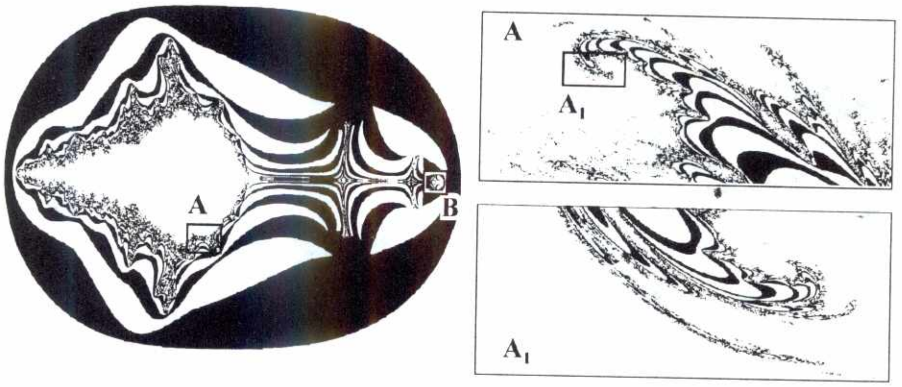


Figure 11. Fractal 'Barnsley-3m' and repeated magnification of its detail

similar structure of this fractal (Fig. 11, below). The fractal set 'Barnsley-3m' obtained by Barnsley [1] by applying the same principle as in the case of Mandelbrot set to more general functions of two complex variables. This set has connection with polarized-light microphotograph of some minerals. It reveals patterns that are less organic and more crystalline than those of the Mandelbrot and Julia sets. The dynamical system for 'Barnsley-3m' is given by

$$z_{n+1} = \begin{cases} Re^2 z_n - Im^2 z_n - 1 + i(2Re z_n Im z_n) \, , & Re z_n > 0 \, , \\ Re^2 z_n - Im^2 z_n - 1 + \lambda Re z_n + i(2Re z_n Im z_n + \lambda Re\, z_n) \, , & Re z_n \leq 0 \, , \end{cases}$$

where λ is a real parameter.

Besides self-similarity, the simplicity of IFS is the next attractive property of modeling natural scenes by fractals. It results in a tremendous compression of the data. Instead of keeping the whole picture in the computer's memory one can save only IFS code which gives compression ratio up to one hundred! To illustrate this, let us compare byte-length of an IFS file with that of the **pcx** format of the corresponding attractor picture: Barnsley fern 123 : 11732; von Koch curve 270 : 14368; Peano curve 586 : 25884 etc. For the **bmp** format the compression ratio is even larger.

Look at the fractal 'Barnsley-3m' from Figure 11. Computer–aided magnification of some part of the fractal set can be performed in two ways:

1. *By some graphical software;* The framed detail B is magnified using standard graphical package (for. ex. Corel-Draw). The result is shown in

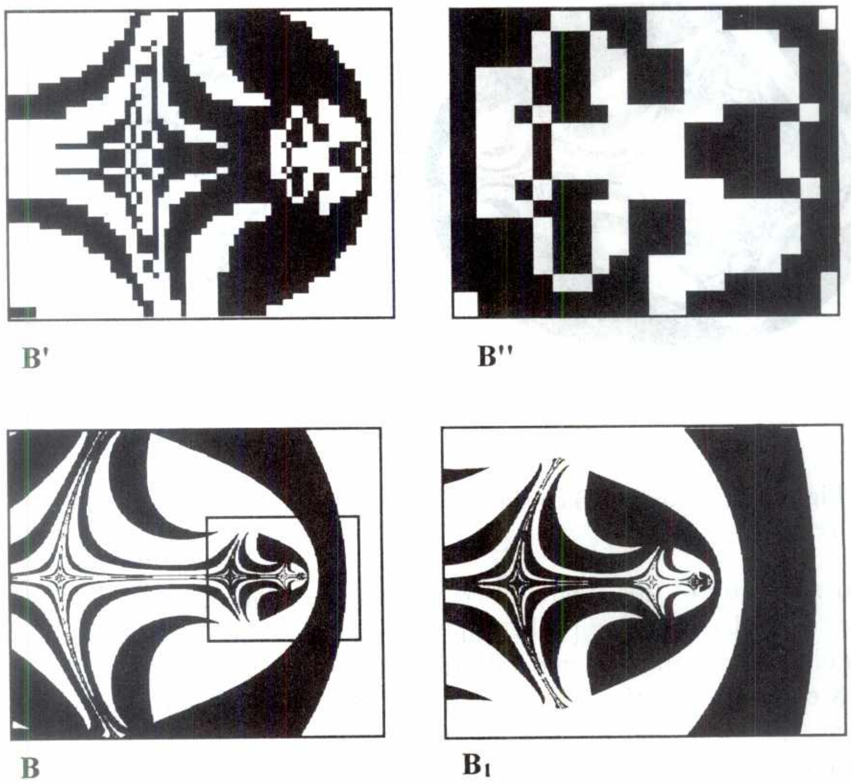


Figure 12. Magnification with or without loosing of details.

Figure 12, frame B'. Repeated magnification will cause further loosing of graphical information (frame B'').

2. *By fractal software;* If the same rule is used in the window B of *Barnsley3m* the picture framed by B in Figure 12 is produced. Repeated magnification of the frame B_1 is shown as the rightmost below frame. Fractal images can be magnified endlessly, without loosing of details.

So, fractal attractors are convenient for modeling different natural forms. Is it important to know how one can define an IFS to produce exactly the image that he wants? The answer is in the collage theorem:

Theorem 7 (Barnsley). *Let L be a nonempty compact subset of* $\mathbf{X}$, *and*

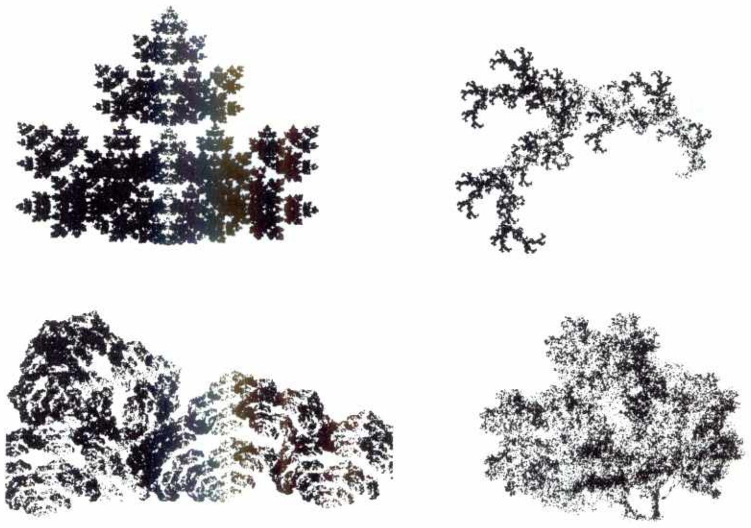


Figure 13. Fractals as natural forms: a leaf, a coral branch, rocks formation, and a tree crown

$\varepsilon \geq 0$ be given. Choose an IFS $\{\mathbf{X}; \phi_0, \dots, \phi_N\}$ with contractivity factor $0 \leq s < 1$, so that $h(L, F(L)) \leq \varepsilon$, where $F(L) = \cup_{i=1}^{N} \phi_i(L)$, and $h(\cdot,\cdot)$ is Hausdorff metric. Then, $h(L, A) \leq \frac{\varepsilon}{1-s}$, where A is the attractor of the IFS.

Proof. See [1], [2] or [3]. □

The value of this theorem is in its practical side. It gives the procedure of constructing IFS, once the fractal attractor is given. Take, for example the leaf form in Figure 13. This is a subset in $(\mathbf{R}^2, d)$. Cover this figure by four smaller copies of this subset, as in making collage. Pieces do not fit quite perfect – some holes and overlappings will occur. These four copies are obtained by performing four affine contractions in $\mathbf{R}^2$: ϕ_1, ϕ_2, ϕ_3 and ϕ_4. This IFS is being used in generating the 'leaf' in Figure 13 by the random algorithm. The 'holes' in the leaf structure appear due to the holes in the collage. But, it is clear that the leaf form is obtained.

In the similar way the 'coral branch', ' formation of rocks' or 'tree crown' are obtained (Fig. 13).

Now, let say something about probabilities p_i that appear in Barnsley–Demko's random algorithm. Take the IFS $\{\mathbf{R}^2; \phi_1, \phi_2, \phi_3, \phi_4\}$ described above. The leaf image in Figure 13 is produced by the random algorithm

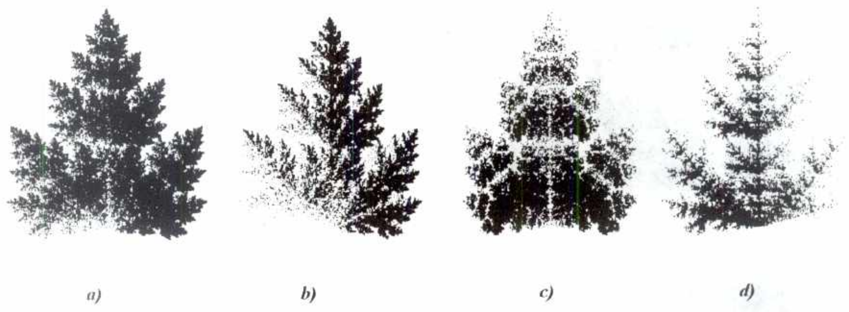


Figure 14. Leaves with different distribution of measure

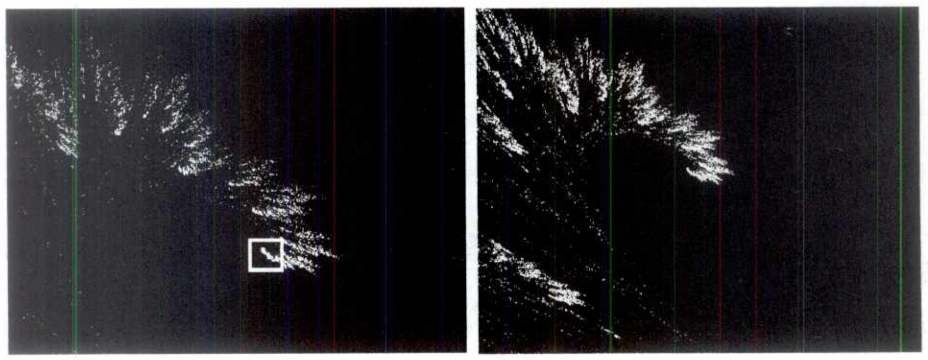


Figure 15. Waves from affine IFS

with uniformly distributed probabilities (0.25, 0.25, 0.25, 0.25). If this vector changes into (0.36, 0.16, 0.34, 0.34) the leaf a) in Figure 14 is obtained. Slight variation: (0.46, 0.16, 0.34, 0.04) brings in an effect as though the leaf was lighted from the left (Fig. 14-b). The choice (0.16, 0.16, 0.34, 0.34) will result into more rounded leaf (Fig. 14-c), while (0.16, 0.56, 0.04, 0.04) gives a fir–tree (Fig. 14-d).

Phenomena in water, wrinkled surface, turbulences and streams can be nicely modeled by fractal sets. The magnifying details of *Barnsley3m* fractal

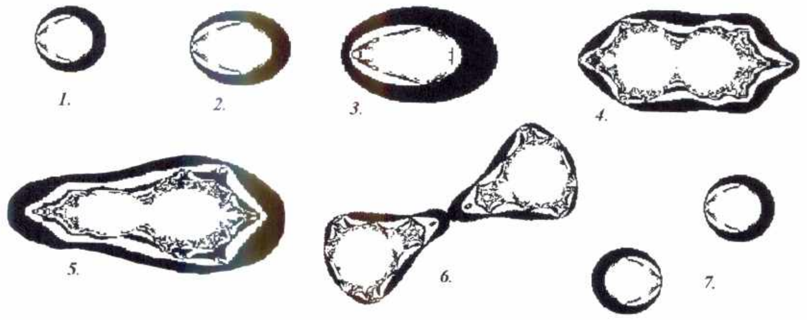


Figure 16. Modeling process of cell division

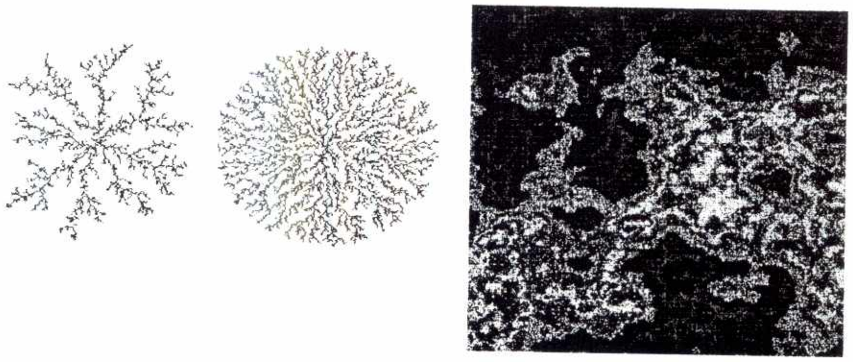


Figure 17. Crystal growth and a plasmatic cloud

from Figure 11 are good background for modeling waves. But they can be made using a simple affine IFS obtained by varying Takagi function (Fig 7-a). This 'wave' together with a magnification is shown in Figure 15.

The fractal, known from dynamical systems as *Mandellambda*, can be of help in modeling a biological process of cell division, Figure 16 (Stages are marked by numbers).

An important type of fractals are obtained by the physical processes known as diffusion–limited aggregation (DLA) [23]. This process is modeled by the use of random generator. Fractal forms obtained resembles tree root, and stand for models of growth of crystal structures (Figure 17).

If some fractal surface is intersected with a plane, a fractal level–line is obtained. These level–lines can be used to model clouds of different particles, which includes plasmatic cloud or intergalactic dust (see the rightmost illustration in Figure 17).

A large class of self–similar fractals are obtained by varying Peano and von Koch curves. The *Hilbert curve* , already mentioned in Section 1, has important application for digital halftoning. It has geometric dimension $Dim_H = 2$, the same as *Peano curve*, which means that it fills the square in the plane. An approximation of Hilbert curve is shown in Figure 18-a. Actually, all space filling curves are mappings $c : \mathbf{I} = [0,1] \to \mathbf{R}^2$. The graph of c covers the square $\mathbf{I}^2$, so that it 'visits' all points of the square in ordered way if parameter t runs from 0 to 1. If $c_n : \mathbf{I} \to \mathbf{I}^2$ is an approximation of the space filling curve c. Subdivision of the interval $\mathbf{I}$ into n subintervals $\mathbf{I}_1, \dots, \mathbf{I}_n$ will result in dividing square $\mathbf{I}^2$ into n subregions $R_1, \dots, R_n$. The size of each subregion R_i varies proportionally with the length of the corresponding subinterval $\mathbf{I}_i$. The curve c_n visits all subregions R_i, actually each point of it. The restriction $c_i : \mathbf{I}_i \to \mathbf{R}_i$ is itself a space filling curve due to the self–similar property. Such restrictions will be used for selecting clusters of pixels (so called dithering) which results in different halftoning effects. This method of dithering using space filling fractal curves has an advantage over standard scan–image methods. Actually, it minimizes the grid effect, which results in better shadow textures [29]. Many variations of space filling curves include curves of Sierpinski [13], Lebesgue and Schoenberg [27] and others [21]. Two curves that fill space in different ways are shown in Figure 18-b and c.

This type of fractal curves inspired Prusinkiewitz and Lindenmayer to introduce L-systems for modeling plants and trees [25]. The central idea is that, in all cases, plants are defined by a small number of rules, applied repetitively to produce complex structures. L-system is a graph–rewriting mechanism, which operates on axial trees, and operates in parallel. The result is a fractal on graph alias *graftal.* Authors of [26] presented a model of tree synthesis which integrates botanical knowledge of the architecture of the trees.

5. Conclusion

This paper offers a short information on fractal geometry and its application in computer graphics and geometric modeling. This geometrz was born as a child of computer era, trying to explain some unsolved problems in mathematics, physics and related sciences. A great insight was given by the books of Mandelbrot [22] and Barnsley [1]. Fractals are sets having, in general, very complicated structure. The shortest definition of the class of

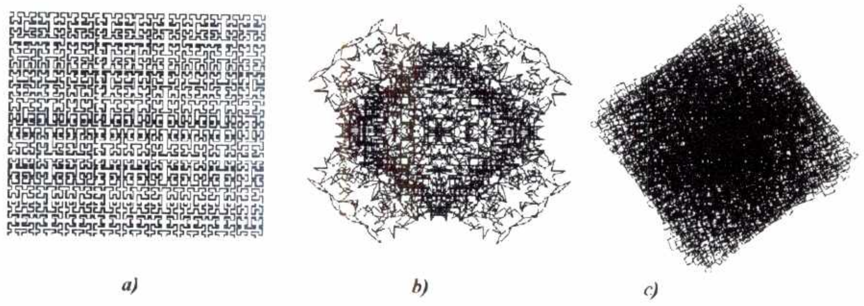


Figure 18. a) The Hilbert curve; b) and c) Two variations on Peano theme

so called *deterministic fractals* is that this is a subset of a compact metric space being invariant under the collection of contractive mappings. A simple but important example of such contractions are affine functions that map a plane into itself. This leads to the most important feature of deterministic fractals: self–similarity. Using this property, one can use fractal sets to model many natural forms having hierarchical self–similar structure: plants, rocks, water dynamics, clouds, foam, neural cells etc.

Text is illustrated with examples of fractal sets, together with some applications. All pictures, except Fig. 17, are produced by the software created by the author. Figure 17 was rendered by using the software *Fractint* by Bert Tyler.

References

[1] M. F. BARNSLEY, *Fractals Everywhere*, Academic Press, 1988.

[2] M. F. BARNSLEY, A. JACQUIN, F. MALASSENET, L. REUTER, A. D. SLOAN, *Harnessing Chaos for Image Synthesis*, Comput. Graph. **22** (1988), 131–140.

[3] M. F. BARNSLEY, *Lecture Notes on Iterated Function Systems*, Chaos and Fractals. The Mathematics Behind the Computer Graphics (R. Devaney and L. Keen, eds.), Amer. Math. Soc., 1989, pp. 127–144.

[4] C. L. BERTHELSEN, J. A. GLAZIER, M. H. SKOLNICK, *Global fractal dimension of human DNA sequences treated as pseudorandom walks*, Phys. Rew. A **45** (1992), 8902–8913.

[5] B. BRANNER, *The Mandelbrot Set*, Chaos and Fractals. The Mathematics Behind the Computer Graphics (R. Devaney and L. Keen, eds.), Amer. Math. Soc., 1989, pp. 75–105.

[6] S. D. Casey, N. F. Reingold, *Self-Similar Fractal Sets: Theory and Procedure*, IEEE CG&A **14** (1994), 73–82.

[7] A. J. Cole, *Compaction Techniques for Raster Scan Graphics using Space-filling Curves*, Computer J. **30** (1987), 87–92.

[8] R. M. Corless, *Continued Fractions and Chaos*, Amer. Math. Monthly **99** (1992), 203–215.

[9] R. H. Dauskardt, F. Haubensak, R. O. Ritchie, *On the Interpretation of the Fractal Character of fracture surfaces*, Acta metall. **38** (1990), 143–159.

[10] S. Dubuc, A. Elqortobi, *Approximation of fractal sets*, J. Comput. Appl. Math. **29** (1990), 79–89.

[11] S. Eubank, D. Farmer, *An Introduction to Chaos and Randomness*, Lectures in Complex Systems, SFI Studies in the Sciences of Complexity (E. Jean, eds.), Addison-Wesley, 1990, pp. 75–190.

[12] B. R. Gelbaum, J. M. H. Olmsted, *Counterexamples in Analysis*, Mir, Moskva, 1967. (Russian)

[13] J. G. Griffiths, *Table-driven algorithms for generating space-filling curves*, Coput. Aided Design **17** (1985), 37–41.

[14] J. Harrison, *Continued Fractals and the Seifert Conjecture*, Bull. Amer. Math. Soc. **13** (1985), 147–153.

[15] J. Harrison, Chaos and Fractals. The Mathematics Behind the Computer Graphics (R. Devaney and L. Keen, eds.), Amer. Math. Soc., 1989, pp. 107–126.

[16] D. Hilbert, *Uber stetige Abbildung einer Linie auf ein Flächenstück*, Math. Annl. **38** (1891), 459–468.

[17] J. E. Hutchinson, *Fractals and Self Similarity*, Indian. J. Math **30** (1981), 713–747.

[18] J. Kappraff, *The Geometry of Coastlines: A Study in Fractals*, Comput. Math. Appl. **12B** (1986), 655–671.

[19] H. Von koch, *Sur une courbe continue sans tangente obtenue par une construction géométrique élémentaire*, Ark. Mat. Astr. Fys. **1** (1904), 681–704.

[20] Lj. M. Kocic, *Affine shape control of cubics*, PU.M.A. **3** (1992), 207–229.

[21] T. Lance, E. Thomas, *Arcs with Positive Measure and Space-Filling Curve*, Amer. Math. Monthly **98** (1991), 124–127.

[22] B. Mandelbrot, *The fractal geometry of nature*, Freeman, San Francisco, 1982.

[23] P. Meakin, J. Feder, T. Jøssang, *Radially biased diffusion-limited aggregation*, Physical Review A (1991), 1952–1964.

[24] G. Peano, *Sur une courbe, qui remplit toute une aire plane*, Math. Annl. **36** (1980), 157–160.

[25] P. Prusinkiewicz, A. Lindenmayer, J. Hanan, *Developmental Models of Herbaceous Plants for Computer Imagery Purposes*, Computer Graph. **22** (1988), 141–150.

[26] P. De Reffye, C. Edelin, J. Françon, M. Jaeger, C. Puech, *Plant Models Faithful to Botanical Structure and Development*, Computer Graph. **22** (1988), 151–158.

[27] H. Sagan, *Approximating Polygons for Lebesgue's and Schoenberg's Space Filling Curves*, Amer. Math. Monthly **93**, 361–368.

[28] T. Takagi, *A simple example of the continuous function without derivative*, Proc. Phys. Math. Soc. Japan **1** (1903), 176–177.

[29] L. Velho, J. M. Gomes, *Digital Halftoning with Space Filling Curves*, Computer Graph. **25** (1991), 81–90.

[30] M. T. Weiss, *An Early Introduction to Dynamics*, Amer. Math. Monthly **98** (1991), 635–641.

[31] S. Wolfram, *Geometry of Binomial Coefficients*, Amer. Math. Monthly **91** (1984), 566–570.

Department of Mathematics, Faculty of Electronic Engineering, P.O.Box 73, 18000 Niš

FILOMAT (Niš) 9:2 (1995), 233–242
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

LINES OF CURVATURE OF FREE FORM SURFACES TRACING

Dušan M. Milošević and Ljubiša M. Kocić

ABSTRACT. *An level-line tracing algorithm, recently developed by the authors is used for Bézier surface interrogation. Namely, for Bézier triangular patches the algorithm is modified so as to trace the lines of constant Gaussian and mean curvature. The map of these lines can be used for better understanding the shape of these patches. The efficacy of the method is illustrated through several examples.*

1. Introduction

The aim of this paper is to obtain curvature level sets of Bernstein-Bézier triangle fragment. Particularly, it gives level sets of Gaussian and mean curvature. This problem is solved by using the algorithm for implicit function graph tracing. Since the, analytic form for Gaussian and mean curvature involve derivatives of two degree, it is necessary to have at least thread order Bézier's fragment.

As far as the applications is concerning, it is enough to mention Computer Aided Geometric Design and Data Visualization. In both topics, the sets of curvature level sets is applied for Bézier surface interrogation. From the curvature level sets one can easily seen the monotonicity, convexity, the existence of saddle points, locations of extrema and gradient intensity of curvature lines. This means having more information about surfaces them self. For example zero Gaussian curvature line share surface on three parts: elliptic (greater than zero) , hyperbolic (smaller) and parabolic (equal). The points of extrema of mean curvature is very important for example in industry because this point of surface is critical in mean of tension.

1991 *Mathematics Subject Classification.* Primary 65D17.
This work was supported in part by the Science Fund of Serbia under grant #0401F.

2. Gaussian end mean curvature

For parametric defined surfaces

$$\vec{x} = \vec{x}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}; \quad \vec{u} = \begin{bmatrix} u \\ v \end{bmatrix} \in [a,b] = T \subset \mathbb{R}^2,$$

where x, y, z are differentiable functions and $T = [a,b]$ is triangle in u, v plane, Gaussian (K) and mean curvature (G) are defined as

$$K = \frac{LN - M^2}{EG - F^2}, \qquad H = \frac{NE - 2MF + LG}{2(EG - F^2)}, \tag{1}$$

where $L,\ M\ , N\ , E\ , F$ and G fit standard Gauss notation

$$L = L(u,v) = \vec{n} \cdot \vec{x}_{uu}, \quad M = M(u,v) = \vec{n} \cdot \vec{x}_{uv}, \quad N = N(u,v) = \vec{n} \cdot \vec{x}_{vv}$$
$$E = E(u,v) = \vec{x}_u \cdot \vec{x}_u, \quad F = F(u,v) = \vec{x}_u \cdot \vec{x}_v, \quad G = G(u,v) = \vec{x}_v \cdot \vec{x}_v.$$

Bézier surfaces are defined implicitly $B(x,y) = 0$. To use (1) it was necessary to express Bézier surfaces in parametric form

$$\vec{x} = \vec{x}(u,v) = \begin{bmatrix} u \\ v \\ B(u,v) \end{bmatrix}; \qquad (u,v) \in T \subset \mathbb{R}^2,$$

which makes

$$\begin{aligned} E(u,v) &= \vec{x}_u \cdot \vec{x}_u = 1 + z_u^2 \\ F(u,v) &= \vec{x}_u \cdot \vec{x}_v = z_u z_v \\ G(u,v) &= \vec{x}_v \cdot \vec{x}_v = 1 + z_v^2. \end{aligned} \tag{2}$$

The normal vector $\vec{n}$ is

$$\vec{n} = \frac{\vec{x}_u \times \vec{x}_v}{||\vec{x}_u \times \vec{x}_v||} = \frac{\begin{bmatrix} -z_u \\ -z_v \\ 1 \end{bmatrix}}{\sqrt{1 + z_u^2 + z_v^2}},$$

and because of that

$$L = \frac{z_{uu}}{\sqrt{1 + z_u^2 + z_v^2}}, \quad M = \frac{z_{uv}}{\sqrt{1 + z_u^2 + z_v^2}}, \quad N = \frac{z_{vv}}{\sqrt{1 + z_u^2 + z_v^2}}. \tag{3}$$

Using (2), (3) and equalities $u = x$ and $v = y$ the analytic form for Gaussian and mean curvature for Bézier surfaces on triangular domain can be obtained.

$$\begin{aligned} K &= \frac{z_{xx}z_{yy} - z_{xy}^2}{(1 + z_x^2 + z_y^2)^2}, \\ H &= \frac{z_{yy}(1 + z_x^2) - 2z_{xy}z_xz_y + z_{xx}(1 + z_y^2)}{(1 + z_x^2 + z_y^2)^{3/2}}. \end{aligned} \tag{4}$$

For finding $m - th$ derivative of Bézier's surface in $l - th$ direction, the following formula [1] is used:

$$\frac{\partial^m}{\partial l^m} B_n(f,t) = \frac{n!}{(n-m)!} \sum_{i+j+k=m} P_{ijk}^{n-m}(t) b_{ijk}^m(l). \tag{5}$$

For $m = 1$ one obtains from (5)

$$\frac{\partial}{\partial l} B_n(f,t) = n \sum_{i+j+k=1} P_{ijk}^{n-1}(t) b_{ijk}^1(l).$$

For $l = (-1, 1, 0)$ (which specifies x-axis direction) this yields

$$\frac{\partial}{\partial x} B_n(f,t) = n(P_{010}^{n-1} - P_{100}^{n-1}),$$

while in y-axis direction $l = (-1, 0, 1)$,

$$\frac{\partial}{\partial y} B_n(f,t) = n(P_{001}^{n-1} - P_{100}^{n-1}).$$

For $m = 2$ (5) becomes

$$\frac{\partial^2}{\partial l^2} B_n(f,t) = n(n-1) \sum_{i+j+k=2} P_{ijk}^{n-2}(t) b_{ijk}^2(l).$$

By using this formula one can find

$$\begin{aligned} \frac{\partial^2}{\partial x^2} B_n(f,t) &= n(n-1)(P_{200}^{n-2} + P_{020}^{n-2} - 2P_{110}^{n-2}), \\ \frac{\partial^2}{\partial y^2} B_n(f,t) &= n(n-1)(P_{200}^{n-2} + P_{002}^{n-2} - 2P_{101}^{n-2}), \end{aligned}$$

$$\frac{\partial^2}{\partial x \partial y} B_n(f,t) = n(n-1)(4P_{200}^{n-2} + P_{020}^{n-2} + P_{002}^{n-2} - 4P_{110}^{n-2} + 2P_{011}^{n-2} - 4P_{101}^{n-2}).$$

3. Curvature level-set

Algorithm for tracing graph of a function given implicitly by $f(x,y)=0$ in some domain is important and attractive problem. Many authors have given important contribution to this problem (see references in [3]). Majority of these methods make use of two stages.

1. Fixing seed points;
2. Tracing the curve.

Fixing seed points

In this stage, the seed points (starting points) are determined for each branch of the curve. This can be done by solving the double sequence of equations $f(x_i,y)=0,\ i=0,\ldots,N_x$ and $f(x,y_j)=0,\ j=0,\ldots,N_y$ where x_i and y_j are uniformly distributed along the interval $[a,b]$. The density of "hunting mesh" is controlled by the numbers N_x and N_y. It is recommended to use a predictor-corrector method for solving of each equation above. First, the coarse subdivision of an interval is performed to locate the root and then an iterative method is applied (here modified Regula falsi method is used).

Tracing the curve

In this stage, starting from the seed points, the algorithm traces branches of the curve until some of them leaves the domain T, or until the branch closes up to form a loop. Tracing of each branch is performed by joining the sequence of points $(x_i,y_i),\ i=0,\ldots,m$, where (x_0,y_0) is the seed point for the corresponding branch. The problem of finding next point on the curve can be solved by using derivatives of $f(x,y)$ or without that. If we choose to use derivatives in tracing implicit graph function $f(x,y)=0$, we must calculate derivatives of (K) and (H) in (4). It mean that we must calculate 3–th derivatives of Bézier's surfaces. It is possible to do provided that we have at least 4–th order Béziers fragment.

a) Algorithms without derivatives

- *Four-point algorithm*

For each point (x,y), the next point in the sequence is calculated by evaluating four neighbour points $(x,y\pm h)$ and $(x\pm h,y)$ and selecting this one which minimize $|f(x,y)|$. In the case when the branch of the curve is closed loop, lying entirely in D, the terminating criteria employes closeness to the starting point. So, each point in the sequence is tested whether or not it is in the ϵ_2 vicinity of the starting seed point. The accuracy may be controlled by testing the inequality $|f(x,y)|<\epsilon$ for each point. If it is not satisfied, the step h is halving until it is.

- *Eight-point algorithm*

This algorithm is similar to the previous one, except that the function is evaluated at eight points $(x, y+h), \dots, (x+h, y+h)$, and the next point is choosing among them so that $|f(x,y)|$ is minimal.

b) Algorithms with derivatives

- *Algorithm with initial value problem solver*

This stage is consist of M iterations to product the sequence $\{(x_0, y_0), \dots, (x_M, y_M)\}$. Connecting these points results in a polygonal line being an approximation of the implicit curve. Each point $p_i = (x_i, y_i)$ is tested for being in ϵ_1-vicinity of a singular point i.e.

$$|F'_x(x_i, y_i)| + |F'_y(x_i, y_i)| < \epsilon_1. \tag{6}$$

The logical value of (6) is the main switch in this stage of the algorithm. If it is true, i.e. if p_i is close enough to the singularity, the next point $p_{i+1} = (x_{i+1}, y_{i+1})$ calculates by linear extrapolation, i.e. $p_i = (p_{i-1} + p_{i+1})/2$. Of course, the case when the seed point $p_0 = (x_0, y_0)$ is also the singular point has to be considered separately. Since the preceding point, say p_{-1} is missing, it is taken $x_{-1} = x_0 \pm h$, $y_{i-1} = y_0 \pm h$ where $h > 0$ is the given step. The signs $\pm$ should be chosen arbitrarily if $p_0 \in intD$. But, if $p_0 \in \partial D$ (the border of D), signs should be chosen so that $p_{-1} \in extD$, which gives $p_1 \in intD$.

If (6) is false, p_{i+1} is found by two-stage predictor-corrector method. Then, one solves

$$y' + \frac{F'_x(x_i, y_i)}{F'_y(x_i, y_i)} = 0, \quad y(x_0) = y_0, \tag{7}$$

whenever

$$F'_y(x_i, y_i) \geq F'_x(x_i, y_i),$$

or

$$x' + \frac{F'_y(x_i, y_i)}{F'_x(x_i, y_i)} = 0, \quad x(y_0) = x_0, \tag{8}$$

otherwise. (Note that it can not be $F'_x(x_i, y_i) = 0$ and $F'_y(x_i, y_i) = 0$ at the same time as the consequence of the singular point being far enough). Equations (7) or (8) are solved by Euler method:

$$x_{i+1} = x_i + S_x h, \quad y_{i+1} = y_i - S_x h \frac{F'_x(x_i, y_i)}{F'_y(x_i, y_i)},$$

where $S_x = \text{sgn}(x_i - x_{i-1})$ when $\delta_i = |F'_x(x_i, y_i)| - |F'_y(x_i, y_i)| < 0$, and

$$y_{i+1} = y_i + S_y h, \quad x_{i+1} = x_i - S_y h \frac{F'_y(x_i, y_i)}{F'_x(x_i, y_i)},$$

where $S_y = \text{sgn}(y_i = y_{i-1})$;

So, the point p_{i+1} is obtained and it is corrected by the Newton-Raphson method,

$$\begin{aligned} y_{j+1} =& y_j - \frac{F(x_j, y_j)}{F'_y(x_j, y_j)}, \quad x_{j+1} = x_j \ (\delta_j < 0), \\ x_{j+1} =& x_j - \frac{F(x_j, y_j)}{F'_x(x_j, y_j)}, \quad y_{j+1} = y_j \ (\delta_j \geq 0), \end{aligned}$$

until

$$|F(x_j, y_j)| < \epsilon_2.$$

This completed the algorithm.

Algorithm with initial value problem solver is better in aspect of accuracy and speed (see [4]), but in the case of 3-th order Bézier path it is necessary to use some of the previous algorithms.

4. Examples

The algorithm is tested through many examples and two of them will be presented here. The arrangement of the control points of n-th order Bernstein-Bézier polynomial is accepted to be

$$\begin{matrix} P_{n00} & \cdots & P_{0n0} \\ \vdots & \cdots & \\ P_{00n} & & \end{matrix}$$

Example 1. For the triangular patch given by the control points

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & \\ 0 & & \end{matrix},$$

the corresponding level-lines map is given in Figure 1.A (level-lines map is obtain by using algorithm develop in [2]). Figures 1.B and Figure 1.C presents the level-lines map of Gaussian and mean curvature respectively.

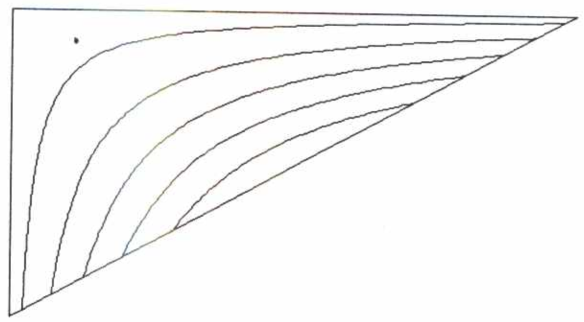


Figure 1.A

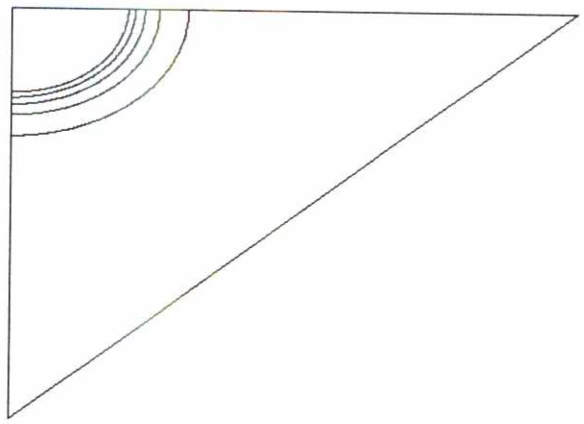


Figure 1.B

Example 2. For the control points

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \\ 0 & 0 & & \\ 0 & & & \end{array},$$

the level-lines map for the corresponding patch is given in Figure 2.A. As in the previous example, the Gaussian and mean curvature are shown by Figure 2.B. and 2.C respectively.

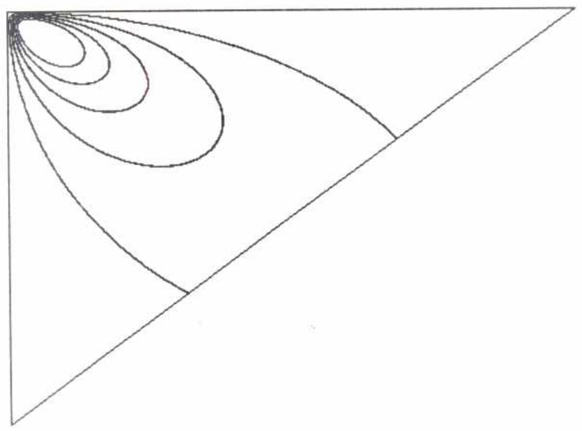


Figure 1.C

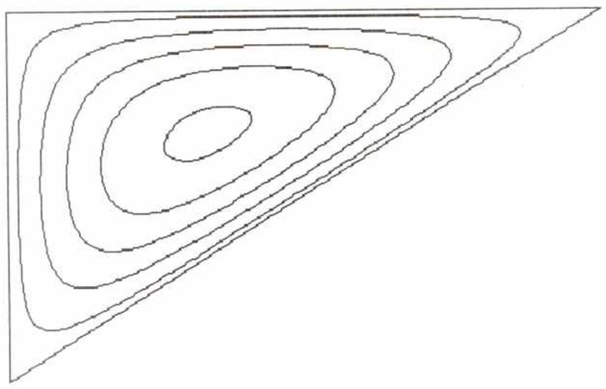


Figure 2.A.

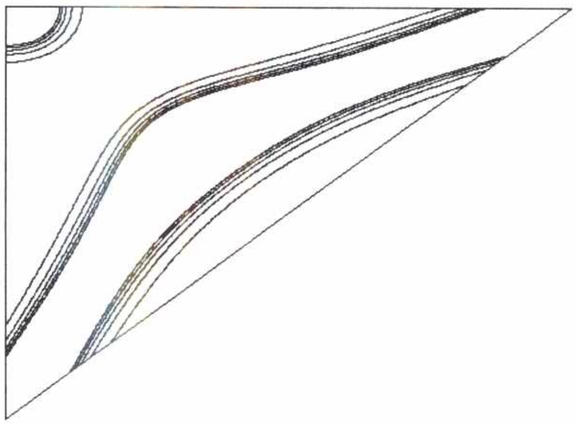


Figure 2.B.

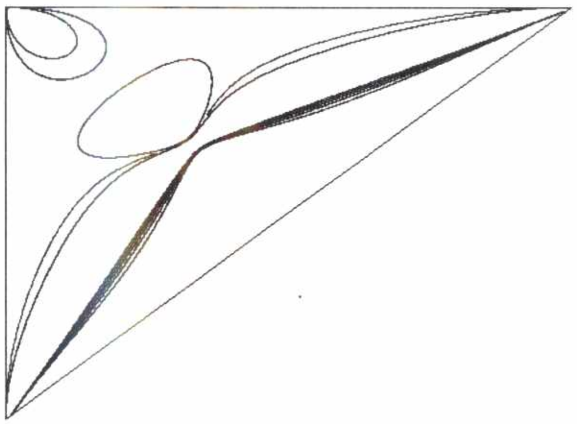


Figure 2.C.

References

[1] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design.*, Academic press, 1988.

[2] Lj. Kocić, D. Milošević, *On level sets of Bernstein-Bézier operators.*, Zbornik radova Filozofskog fakulteta u Nišu, Serija Matematika **6** (1992), 19–25.

[3] Lj. Kocić, D. Milošević, *Numerical Characteristics of Algorithm for Implicit Curve tracing*, Facta Univ. Ser. Mathematics and Informatics **8** (1993), 97–109.

[4] D. Milošević, Lj. Kocić, *Comparison of some algorithms for implicit function graph tracing.*, IX Conference on applied mathematics, Budva, 30 May – 1 Jun 1994, (D. Herceg, Lj. Cvetković, eds.), Institut of Mathematics, Novi Sad, 1995, pp. 65–70.

Department of Mathematics, Faculty of Electronic Engineering, P.O.Box 73, 18000 Niš

FILOMAT (Niš) 9:2 (1995), 243–250
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

MODELING OF RATIONAL CURVES BY INTERPOLATION

Nenad V. Blagojević and Ljubiša M. Kocić

ABSTRACT. *The algorithm for modeling shapes with (n,n)-rational curves is proposed. It is based on interpolation by rational functions using continued fraction numerical technique. The converse algorithm for transformation of rational curve into a parametric continued fraction form is also given. The direct algorithm is illustrated through several examples.*

1. Introduction

The Bézier curve of degree n is defined by the control points $B_0, \dots, B_n$ trough

$$P_n(t) = \sum_{i=0}^{n} B_i b_i^n(t), \quad t \in [0,1],$$

where $b_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$ are Bernstein basis polynomials. An example of a third order Bézier curve is shown in Figure 1(a).

A natural generalization of this model is the rational Bézier curve (of degree n) that, besides the control points $B_0, \dots, B_n$ involves the weights $\omega_0, \dots, \omega_n$ as shape parameters

$$R_n(t) = \frac{\sum_{i=0}^{n} B_i \omega_i b_i^n(t)}{\sum_{i=0}^{n} \omega_i b_i^n(t)}, \quad t \in [0,1]. \tag{1}$$

1991 *Mathematics Subject Classification.* Primary 65D05; Secondary 41A20.
Key words and phrases. modeling, rational curve, interpolation, continued fraction, inverted differences.
This work was supported in part by the Science Fund of Serbia under grant #0401A.

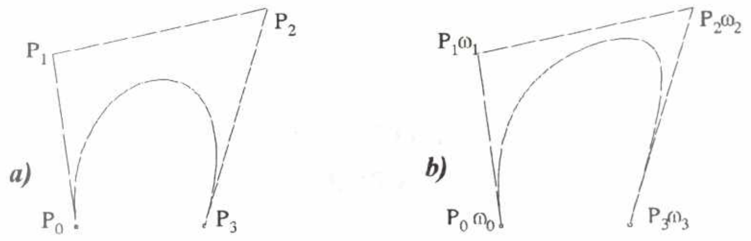


Figure 1. Bézier curve (a) and rational Bézier curve (b)

If some weight is relatively large comparing to others, the corresponding control point "pulls" the curve toward it. Figure 1(b) shows the rational curve with $\omega_0 = \omega_3 - 1$, $\omega_1 = 3$ and $\omega_2 = 6$.

The rational scheme reveals many useful properties. The most important of them are:

- the possibility of exact modeling of conic sections;

- continuous changing of weights results in continuous adjustment of curve form.

On the other hand, all good properties of the polynomial Bézier curves maintains, except subdivision which can not be carried over without weights being changed.

It is customary in free form curve modeling to use some interpolation model as an initiator. The Bézier curve modeling is preceded by the Lagrange or spline interpolation model. For the rational Bézier curve, it is recommendable to start with rational interpolant. The most natural approach is to represent such (n,n)-rational interpolation curve via the Bernstein basis, for each coordinate axis separately. For example for x-axis:

$$(2) \qquad R_n(x) = \frac{\sum_{i=0}^{n} B_i \omega_i b_i^n(x)}{\sum_{i=0}^{n} \omega_i b_i^n(x)}, \quad x \in [0,1],$$

where the ordinates B_i and weights ω_i are to be determined so that $R_n(x)$ interpolates the data $\{(x_i, y_i)\}_{i=0}^{\nu}$, i.e.

$$(3) \qquad R_n(x_i) = y_i, \quad i = 0, \ldots, \nu,$$

with ν chosen so that there are enough equations to determine B_i, ω_i in (2), with one arbitrary weight.

A variant of this problem is considered by Piegl [8], but for piecewise cubic rational curve.

By introducing (3) in (2), the following linear system is obtained

$$\begin{bmatrix} b_0^n(x_0) & \cdots & b_n^n(x_0) & -y_0 b_1^n(x_0) & \cdots & -y_0 b_n^n(x_0) \\ b_0^n(x_1) & \cdots & b_n^n(x_1) & -y_1 b_1^n(x_1) & \cdots & -y_1 b_n^n(x_1) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_0^n(x_\nu) & \cdots & b_n^n(x_\nu) & -y_\nu b_1^n(x_\nu) & \cdots & -y_\nu b_n^n(x_\nu) \end{bmatrix} \begin{bmatrix} B_0\omega_0 \\ B_1\omega_1 \\ \vdots \\ \omega_n \end{bmatrix} = \begin{bmatrix} y_0 b_0^n(x_0) \\ y_1 b_0^n(x_1) \\ \vdots \\ y_\nu b_0^n(x_\nu) \end{bmatrix}.$$

Unfortunately, this system has no such nice behaviour as in the case of Lagrange interpolation (Vandermonde determinant $\neq 0$). Here, the singularity can occur. Next, the interpolant may not exist in spite of regularity of the system (see Mayers [6]).

In this paper only the case when interpolant exists is considered.

For interpolant construction, the inverted differences are used while the interpolant has continued fraction form

(4) $$R_n(x) = c_0 + \frac{x - x_0}{c_1} \underset{+}{} \frac{x - x_1}{c_2} \underset{+\cdots+}{} \frac{x - x_{\nu-1}}{c_\nu},$$

or more conveniently

(5) $$R_n(x) = \left[c_0; \frac{x - x_0}{c_1}, \frac{x - x_1}{c_2}, \ldots, \frac{x - x_{\nu-1}}{c_\nu}\right],$$

where

(6) $$c_i = \phi(x_0, \ldots, x_i), \quad i = 0, \ldots, \nu,$$

are inverted differences given by

(7)
$$\phi(x_0) = y_0, \quad \phi(x_0, x_1) = \frac{x_1 - x_0}{y_1 - y_0},$$
$$\phi(x_0, \ldots, x_i) = \frac{x_i - x_{i-1}}{\phi(x_0, \ldots, x_{i-2}, x_i) - \phi(x_0, \ldots, x_{i-2}, x_{i-1})}, \quad i = 2, 3, \ldots$$

Now, the continued fraction $R_n(x)$ can be expressed in the form $\frac{P_n(x)}{Q_n(x)}$. By the using of the known transformation from monomial to Bernstein basis, the rational form (2) can be obtained. But, this transformation is numerically unstable (Farouki, Rajan [3]). So it is better to find the algorithm for direct expression of the continued fraction in Bernstein form.

2. Algorithms

Here, two algorithms are proposed. One, for transformation the continued fraction in rational Bézier form and, the second, for inverse transformation back to the continued fraction form.

For continued fraction $R_n(x)$, given by (4), the rational function

(8)
$$r_k(x) = \frac{P_k(x)}{Q_k(x)} = \left[c_0; \frac{x-x_0}{c_1}, \frac{x-x_1}{c_2}, \ldots, \frac{x-x_{k-1}}{c_k}\right], \quad k = 0, \ldots, \nu - 1,$$

refers to as k-th convergent of $R_n(x)$. Obviously, $r_n(x) = R_n(x)$.

It is known that polynomials P_k and Q_k satisfy three term recurrence relation

$$z_k = c_k z_{k-1} + (x - x_{k-1}) z_{k-2}, \quad k = 1, \ldots, n,$$

where the sequence $\{P_k\}$ is initialized by $P_{-1}(x) = 1$, $P_0(x) = c_0$ and $\{Q_k\}$ by $Q_{-1}(x) = 0$, $Q_0(x) = 1$, see [2], [4], [5], [7].

The Algorithm 1 is given by the following theorem:

Theorem 1. *Let the set of points in the plane $\{(x_i, y_i)\}_{i=0}^n$ be given sash that $0 = x_0 < x_1 < \cdots < x_n = 1$. Let the k-th convergent of $R_n(x)$ be given in Bézier form*

$$r_k(x) = \frac{\sum_{i=0}^k p_i^k b_i^k(x)}{\sum_{i=0}^k p_i^k b_i^k(x)}, \tag{9}$$

by the coefficients p_i^k and q_i^k $(i = 0, \ldots, k)$ satisfy the recurrence relation

$$\begin{aligned} s_i^k &= A_0 s_i^{k-1} + A_1 s_{i-1}^{k-1} + A_2 s_i^{k-2} + A_3 s_{i-1}^{k-2} + A_4 s_{i-2}^{k-2}, \\ &i = 0, \ldots, k, \quad k = 2, 3, \ldots, n, \end{aligned} \tag{10}$$

with initial conditions for $\{P_k\}$, given by $p_0^0 = c_0$, $p_0^1 = c_0 c_1 - x_0$, $p_1^1 = c_0 c_1 - x_0 + 1$, and $q_0^0 = 1$, $q_0^1 = q_1^1 = c_1$, for $\{Q_k\}$. The constants A_i in (10) are given by

$$A_0 = \frac{(k-i)c_k}{k}, \quad A_1 = \frac{ic_k}{k}, \quad A_2 = -\frac{(k-i)(k-i-1)x_{k-1}}{k(k-1)},$$
$$A_3 = -\frac{i(k-1)(1-2x_{k-1})}{k(k-1)}, \qquad A_4 = -\frac{i(i-1)(1-x_{k-1})}{k(k-1)}.$$

In practical calculations the coefficients A_i are replaced with

$$A_0 = (k-1)(k-i)c_k, \quad A_1 = (k-1)ic_k, \quad A_2 = -(k-i)(k-i-1)x_{k-1},$$
$$A_3 = -i(k-1)(1-2x_{k-1}), \qquad A_4 = -i(i-1)(1-x_{k-1}).$$

In this way, one can avoid operation of division, which results in improving the numerical stability of the algorithm.

Finally, the control points B_i and weights ω_i can be determined from the system

(11) $$B_i\omega_i = p_i^n, \quad \omega_i = q_i^n, \quad i = 0, \ldots, n.$$

In the case $\omega_i = 0$, the control point B_i is an arbitrary constant.

The truncation error estimates as (see [4])

(12) $$R_n(x) - r_k(x) = K[r_k(x) - r_{k-1}(x)], \quad x \in [0,1],$$

with $K = -d_k(x)/\big(1 + d_k(x)\big)$, where

(13) $$d_k(x) = \frac{(x - x_k)Q_{k-1}(x)}{\phi_{k+1}(x_0, \ldots, x_k, x)Q_k(x)},$$

and ϕ_{k+1} are inverted differences given by (7) with y_i being replaced by $R_n(x_i)$. Note that the constant K in (12) can be easily approximated using extended de Casteljau algorithm [1], which allows to compute both $Q_k(x)$ and ϕ_{k+1}.

Converse algorithm

Conversely, the Bézier curve (2) can be transformed in the form

$$R_n(x) = \left[a_0; \frac{1}{\beta_1(x)}, \frac{1}{\beta_2(x)}, \dots, \frac{1}{\beta_n(x)}\right], \tag{14}$$

where

$$\beta_k(x) = C_{n-k+1} b_0^1(x) + D_{n-k+1} b_1^1(x), k = 1, \dots, n, \tag{15}$$

are the Bernstein polynomials of first order. This procedure of the Algorithm 2 is given by the following theorem:

Theorem 2. *In* (14), *the constants* a_0, C_k *and* D_k, $k = 1, \dots, n$ *are given by*

$$\begin{cases} p_0^n = a_0 q_0^n + p_0^{n-1}, \\ p_i^n = a_0 q_i^n + \frac{i}{k} p_{i-1}^{n-1} + \left(1 - \frac{i}{k}\right) p_i^{n-1}, \quad i = 1, 2, \dots, n-1, \\ p_n^n = a_0 q_i^n + p_{n-1}^{n-1}, \end{cases} \tag{16}$$

$$\begin{cases} q_i^k = \frac{k-1}{k} C_{n-k+1} p_i^{k-1} + \frac{i}{k} D_{n-k+1} p_{i-1}^{k-1} + \frac{(k-1)(k-i-1)}{k(k-1)} r_i^{k-2} \\ + \frac{2i(k-i)}{k(k-1)} r_{i-2}^{k-2} + \frac{i(i-1)}{k(k-1)} r_{i-2}^{k-2}, \quad i = 0, \dots, k \quad k = n, n-1, \dots, 1. \end{cases} \tag{17}$$

Proof. After division in $R_n(x)$ one obtains

$$R_n(t) = \frac{\sum_{i=0}^n p_i^n b_i^n(x)}{\sum_{i=0}^n q_i^n b_i^n(x)} = a_0 + \frac{\sum_{i=0}^{n-1} p_i^{n-1} b_i^{n-1}(x)}{\sum_{i=0}^n q_i^n b_i^n(x)}, \tag{18}$$

where, according to [3 eq. 48], constants a_0, p_i^n, q_i^n and p_i^{n-1} are connected as in (16). Note that a_0 can be expressed explicitly as

$$a_0 = \frac{\sum_{i=0}^n (-1)^{n-1} \binom{n}{i} p_i^n}{\sum_{i=0}^n (-1)^{n-1} \binom{n}{i} q_i^n}.$$

After k-th division in (18), by the rule of continued fraction, one gets

$$\begin{aligned}R_n(x) &= a_0 + \frac{1}{\beta_1(x)}{}_+ \frac{\sum_{i=0}^{k} p_i^{k-1} b_i^{k-1}(x)}{\sum_{i=0}^{k} q_i^k b_i^k(x)} \\ &= a_0 + \frac{1}{\beta_1(x)}{}_{+\cdots+} \frac{1}{\beta_n(x) + \frac{\sum r_i^{k-2} b_i^{k-2}(x)}{\sum p_i^{k-1} b_i^{k-1}(x)}},\end{aligned}$$

i.e.

$$(19)\qquad \sum_{i=0}^{k} q_i^k b_i^k(x) = \beta_k(x) \sum_{i=0}^{k-1} p_i^{k-1} b_i^{k-1}(x) + \sum_{i=0}^{k-2} r_i^{k-2} b_i^{k-2}(x).$$

Relations (17) follow from (19) by comparing coefficients after replacing $\beta_k(x)$ by (15), then by using identities $(1-x)b_i^{n-1}(x) = \frac{n-i}{n} b_i^n(x)$, $x b_i^{n-1}(x) = \frac{i+1}{n} b_{i+1}^n(x)$, and, after that, elevating degree of $\sum_{i=0}^{k-2} r_i^{k-2} b_i^{k-2}(x)$ for two. □

3. Applications and examples

The main application of Algorithm 1 is in modeling. Using interpolating points, the initial interpolation model is found by calculating the control points B_i and weights ω_i and the one can continue the modeling process by the standard interactive technique.

Second application is in recognition the parameters for same free form curve. For example, if one knows that some curve is rational but does not know its control points or weights they can be retrieved by the Algorithm 1.

The Algorithm 2 carries over the Bézier rational curve into continued fraction form. It may be important for further processing of such curve, like for approximation or data reduction.

The following examples illustrate our Algorithm 1

Example 1. The data (x_i, y_i):$(2.5, 6.875)(5.0, 2.23)(20.0, 0.283)(40.0, 0.143)$ results the curve in Figure 2(a), while the more complete data (x_i, y_i) : $(2.5, 6.875)(5.0, 2.23)(10.0, 0.751)$ $(15.0, 0.416)$ $(20.0, 0.283)$ $(25.0, 0.219)$ $(30.0, 0.182)$ $(40.0, 0.143)$ gives curve in Figure 2(b).

Example 2. Here, the data (7.99, 0.0) (8.09, 0.000027643) (8.19, 0.0437488) (8.7, 0. 169183) (9.2, 0.46428) (10.0, 0.943740) (12, 0.998636) (15.0, 0.999919) (20.0, 0.9999 94) are used. The corresponding rational curve is shown in Figure 2(c).

Example 3. Figure 2(d) shows the result of applying Algorithm 1 on the data (−4.0, −1.0) (−3.0, −1.0) (−2.0, −1.0) (−1.0, −1.0) (0.0, 0.0) (1.0, 1.0) (2.0, 1.0) (3.0, 1.0) (4.0, 1.0).

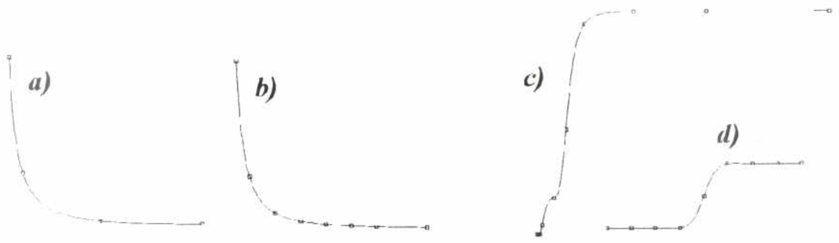


Figure 2. Rational Bézier interpolants for various data

References

[1] N. V. Blagojević, *Racionalni modeli krivih i površi i primene u račuarskoj grafici*, Magistarska teza, Univerzitet u Nišu, 1993..

[2] J. D. P. Donnely, *Continued Fractions*, Methods of Numerical Approximation (D. C. Handscomb, eds.), Pergamon Press, Oxford, 1965.

[3] R. T. Farouki and V. T. Rajan, *Algorithms for polynomials in Bernstein form*, Comput. Aided Geom. Design **5** (1988), 1-26.

[4] F. B. Hilderbrand, *Introduction to Numerical Analysis*, McGraw Hill, New York, 1956.

[5] W. B. Jones and W. J. Thron, *Continued Fractions, Analytic Theory and Applications*, Addison-Wesley Publ., London, 1980; Russian transl., Mir, Moskva, 1985.

[6] D. F. Mayers, *Interpolation by Rational Function*, Methods of Numerical Approximation (D.C. Handscomb, eds.), Pergamon Press, Oxford, 1965.

[7] G. V. Milovanović, *Numerička Analiza.* I, Naučna Knjiga, Beograd, 1991.

[8] L. Piegl, *Interactive Data Interpolation by the Rational Bézier Curves*, IEEE Comput. Graph. Applic. **1987**, no. 9(7), 485-498.

[9] H. Werner, *Algorithm 51 - A Reliable and Numerically Stable Program for Rational interpolation of Lagrange Data*, Computing **1983**, no. 5(31), 269-286.

Ei SiGraf, Trg Bratstva i jedinstva, 2, 18000 Niš.
E-mail address: `nenad@ban.junis.ni.ac.yu`

Dept. of Mathematics, Faculty of Electronic Engineering, P.O.Box 73, 18000 Niš.
E-mail address: `kole@gauss.elfak.ni.ac.yu`

FILOMAT (Niš) 9:2 (1995), 251–259
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

"EXACT" DISPLAY OF OBJECTS WITH REAL VALUED POSITIONS AND DIMENSIONS

Siniša N. Hristov, Miomir S. Stanković and Vesna I. Veličković

ABSTRACT. *In this paper we consider the correct method for the "exact" display of objects with arbitrary forms, having positions and dimensions expressed as arbitrary real numbers. We also consider advantages of such an approach over the usual methods which do not produce "exact" picture, or can "exactly" display only some forms of objects which must have integer positions and dimensions. We also consider some difficulties that might arise in an implementation.*

1. Introduction

This paper deals with methods for generation of an image from an internal description of a scene.

An **image** is a two-dimensional array of numbers, held in computer memory, from which the **actual picture** on the screen is produced by some suitable hardware. A single element of this array is known as a **pixel**, short for "picture element".

A **scene** is some internal description of the **desired picture**. We leave the particular form of the description undefined, but assume that it describes every detail of the desired picture with the complete precision.

We emphasize the distinction between the desired picture, represented by the scene description, and the actual one, represented by the image and presented on the screen.

1991 *Mathematics Subject Classification.* 68U05; 68U10.
Key words and phrases. computer graphics, signal processing, filters, aliasing.

2. Usual "nonexact" methods

Image generation methods used in most computer graphics packages fall into the following three categories:

1. turn-on fully all pixels that have their centers covered by the object;
2. turn-on fully all pixels that have at least half of their area covered by the object;
3. set the intensity of a pixel in proportion with that part of its area which is covered by the object.

It is known that each of these methods suffers from one or more of the following imperfections:

1. object edges appear "ragged";
2. all dimensions must be expressed as integer multiples of the pixel size;
3. objects are not displayed accurately enough – there is significant distortion of object's shape and position.

Some graphics package implementors have recognized the first disadvantage as a serious one and have provided an option to use some form of "anti-aliasing", so that objects appear to have more "smooth" edges. As the "anti-aliasing" is usually performed by some semi-empirical procedure, the resulting picture may appear "smooth", but it is nevertheless inaccurate. And inaccurate picture, having either "ragged" or "smooth" edges, has, as we shall see, serious practical deficiences.

Let us note that most computer graphics applications involve presentation of some scene which is generally defined in a continuous two-dimensional space. There are some applications, circuit board design, for example, which place objects on a predefined grid, but when comes to the image generation, the grid does not help much. Therefore, we shall restrict our discussion to continuos space only.

Some basic graphics packages allow only integer values of object coordinates and dimensions. They force the programmer to write explicit conversions from the continuous space, be it rounding or whatever. In this way, the programmer has full control over the actual picture, which she uses to create some clever arrangements of objects, disguising aforementioned imperfections as much as he can, [1]. Besides placing enormous burden on the programmer, such an approach suffers from a phenomenon common to all "singular" designs: small changes in input data can completely invalidate all she has achieved.

Numerous graphics packages allow specification of real values for coordinates and dimensions. But, to specify is one thing, and to display is quite another. Some "less sophisticated" graphics packages simply round the real values to the nearest integers. The scene is effectively converted into a "similar" one, from which the image is generated. Many have noticed that the rounding errors introduced in this process are by no means insignificant. Other, "more sophisticated" graphics packages attempt to draw approximations of objects without rounding coordinates first. Some clever algorithms are employed to determine pixel values, and in specific cases acceptable results are produced, [1]. However, it is our impression that all such methods rely too much on clever tricks, without having solid theoretical background, and can, therefore, produce acceptable results only in limited cases.

Contrary to the popular belief, we find that an error of "a pixel or two" is by no means negligible, at least given the resolution of today's equipment. Here is a short list of most common consequences of such "small" errors.

1. A uniform set of objects from the continuous space appears as non-uniform, and vice versa.
2. Parts of objects or entire objects disappear.
3. Shape of a small object appears very distorted.
4. The original proportions of object positions and dimensions are not retained.
5. A small change in object's position or size can sometimes cause significant effect on the picture, while in some other case a much bigger change produces no effect.

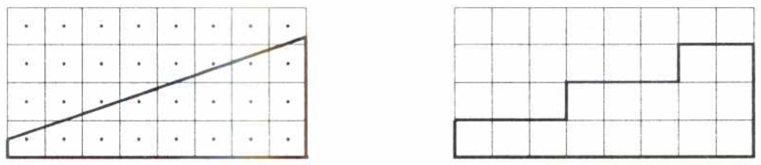


Figure 1. "Ragged" appearance of an object edge.

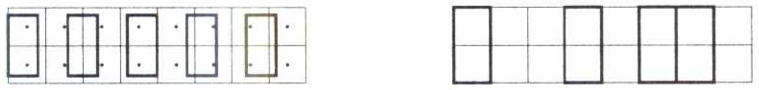


Figure 2. A uniform set of objects appears as non-uniform.

Increasing the image resolution makes such errors somewhat less noticeable, but still visible. The right way to fight this problem is certainly not to increase the image resolution. This is very expensive and quite limited by

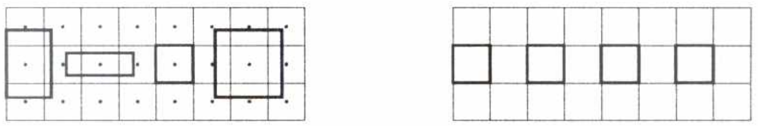


Figure 3. A non-uniform set of objects appears as uniform.

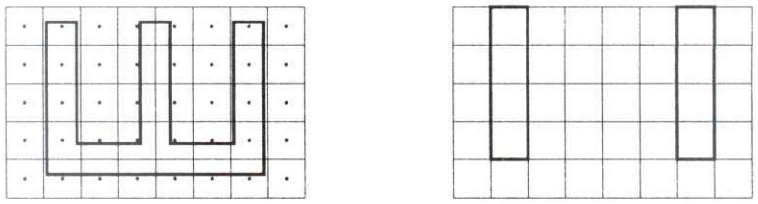


Figure 4. Some parts of the object disappear.
The shape appears very distorted.

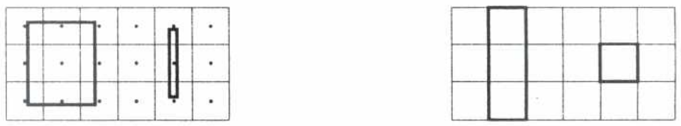


Figure 5. The relative proportions are not retained.

the state of technology, and does not even touch the heart of the problem which is simply the improper sampling.

3. Sampling and reconstruction

The Shannon sampling theorem (see, for example [3]) says that a signal can be properly reconstructed if its spectrum is non-zero only at frequencies less than a half of the sampling rate.

If there is a signal component not satisfying this condition, it will be sampled, but the samples will look exactly as if they came from a component at some frequency less than a half of the sampling rate. In this case the reconstructed signal will have a component not originally present. This phenomenon is known as "aliasing".

As it was mentioned above, the scene is defined in a continuous space, and therefore shall be regarded as a continuous signal. The "image generation" process is, in fact, sampling. The scene may or may not satisfy the sampling theorem condition. The picture is produced from the image by reconstruction, or interpolation, which is performed by the graphics hardware.

Now we can define the correct method for image generation.

If the scene satisfies the sampling condition, everything is well – the aforementioned procedure **will** produce the exact picture. But, if the scene does not satisfy the sampling condition, the exact picture **cannot** be produced. Instead, a "correct" replacement shall be provided.

The question of the "correct" replacement is more philosophical and aesthetical one, rather than technical. We choose the following line of thought: if a component of the scene can be displayed exactly, then do so, and if it cannot be displayed exectly, then suppress it, rather than displaying it as something else that did not exist in the scene.

In other words, we do not attempt to sample the scene that does not satisfy the sampling condition. Instead, we transform that scene into a "similar" one satisfying the sampling condition. This is done by filtering the scene with well chosen low-pass filter. Please note that the scene is a continuous signal – we **cannot** apply a digital filter for this purpose because we do not have a digital signal.

To further substantiate our choice of the "correct" replacement, we note that the components that we have suppressed carry the structure too fine to be displayed by the hardware and/or noticed by the viewer. Therefore, we hope that the absence of those components will not do much harm neither. Had we done otherwise, those components would "alias" to lower frequencies, translating to much larger structure which will be displayed by the hardware and noticed by the viewer.

4. Practical advantages of exact display

Practical advantages of exact display of objects follow from the fact that dimensions and positions do not have to be unnaturally restricted to integer multiples of the pixel size.

An object may have arbitrary size and can be placed anywhere on the screen with the resolution determined by the precision of the floating point numbers used. The uniformity of a set of objects is preserved, as is the non-uniformity. The proportions are retained. A small change in object position or size produces the corresponding small effect on the pucture. In animation, objects do not jump irregularly from pixel to pixel, instead they move in uniform steps. The object shape is not distorted, although very small objects can be smeared or even completely invisible.

The programmer does not have to care about screen resolution and round off errors. And she gets exactly the picture she specified, unlike some modern graphics packages which do allow such a freedom of expression, but distort the picture and leave no possibility for the user to control the picture quality.

The quality of the computer graphics equipment is usually specified by the resolution in the sense of the size of the array holding the image. This is in contrast to the quality specification method used all other visual and optical devices, where the resolution denotes the size of the smallest object that can be reliably reproduced. The resolution of the graphics equipment with exact display of objects can be also given as the size of the smallest object that can be reliably reproduced, regardless of its allignment relative to the pixel grid.

5. How to produce exact pictures systematically?

A new graphics package must be written in order to enable application programmers to use exact pictures within their programs regularly. Although at the moment we do not have the complete proposition for such a package, we can state some basic requirements.

The scene shall be defined in a continuous space and shall be represented in the computer as a set of objects (not to be confused with the so-called "object oriented programming").

There shall be a predefined repertoire of parametrized primitive objects and the user will generate required number of instances and supply actual values for parameters, e.g. size, position, color, etc.

There shall be a systematic way of building complex objects from more primitive ones. Complex objects constructed in this way could also be parametrized, and any number of instances could be generated, with possibility to include them in still more complex objects.

Notions of a point and a line shall be defined in the mathematical sense, i.e. having no area. Therefore, they will not itself be objects, but will be used to build primitive objects.

Some set of predefined primitive objects shall be provided. It is important to select them very carefully, as it must be possible to draw them very efficiently, and, at the same time, to effectively use them in building complex objects and constructing typical scenes.

The package shall include basic geometric transforms, such as translation, rotation, scaling, etc. It shall be possible to apply those transforms uniformly to any kind of object, and to define complex transforms in terms of simpler ones.

Finally, when the complete scene is defined, a drawing procedure will be invoked to produce the image array by filtering the scene and sampling the filter output. Filtering must be performed analyticaly because a numerical approximation will involve sampling and result in aliasing. As long as all objects in the scene are disjoint, filtering can be performed object by object, with all outputs summed – the filtering is a linear process.

Note that the filter output must be known only at sampling points. Therefore, filtering and sampling can be combined into a conceptually simple procedure of centering filter's impulse response at the sampling point and computing the convolution integral. The result is recorded as the pixel value.

6. Desirable filter properties

The most critical thing in implementation of the proposed approach is certainly the choice of the low-pass filter, which has crucial impact both on the quality of the picture and on implementation efficiency. We'll present now our preliminary view of desirable properties of such a filter.

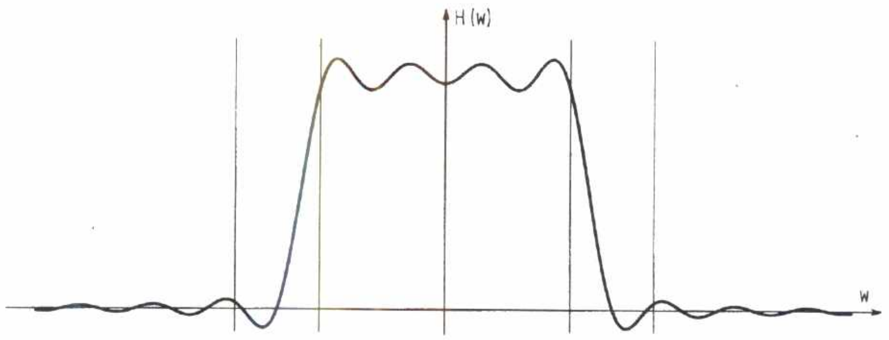


Figure 6. A filter shape in the frequency domain.

Desirable filter properties in the frequency domain are:

1. Relative intensity of components with different frequencies must not be considerably distorted, that is, the variation of the $|H(\omega)|$ in the passband shall be from 0.5% to 2%.
2. Components not satisfying the sampling theorem condition should be sufficiently supressed, that is, peaks of $|H(\omega)|$ in the stopband shall be from 0.2% to 1%.
3. In order to use as wide frequency band as possible, which means as much scene details as possible, the transition band shall be as narrow as possible, that is

$$\frac{\text{upper limit of the passband}}{\text{lower limit of the stopband}} = \quad \text{from } 0.3 \text{ to } 0.9.$$

Desirable filter properties in the spatial domain are:

1. To enable computationally efficient filtering, the filter's impulse response shall be non-zero only over a finite interval, and that interval shall be as narrow as possible. This allows us to consider only a relatively small number of neighbouring objects while computing the value of the filtered scene at a given point.
2. To prevent apperance of a fine structure which did not exist in the scene, the filter's step response should be monotonic, or at least the amplutude of oscillations should not exceed 0.05% to 2%. Also, the filter's impulse response should not have negative values.
3. The transition from one light intensity to another should be as fast as possible, that is, the filter's rise time should be as short as possible.
4. The filter's delay should not depends on frequency or, even better, the delay should be zero. This will be satisfied if the filter's impulse response is an even function.
5. If the scene is rotated, the displayed picture shall appear rotated, but otherwise unchanged. This will be satisfied if the filter's impulse response is rotationally symmetric.

7. Expected implementation difficulties

A relatively complex calculation must be performed in order to obtain the value of a single pixel. The same procedure must be performed about a milion times to complete the image. We expect that achieving a reasonable drawing speed will be the major problem in the implementation.

A rough estimate shows that commonly available processors such as 486, 68040 and T805 permit only experimentation with the proposed approach. For practical applications processing must be faster for at least one order of magnitude. As we have to work with continuous signals and relatively complex data structures and algorithms, currently available digital signal processors do not seem to be particularly useful – they are very diffucult to use for anything outside their intended application area.

Our attention is directed towards the T9000, if it becomes regularly available. It seems that a single T9000 might satisfy basic application requirements. Even more important is its capability for parallel operation, which far exeeds capabilities of all other commercial processors. The single PPC 604 also seems to have enough power for basic applications, but with much less hope for efficient parallel operation.

Another serious difficulty arises from the possibility that a single filter may not be conveniently applicable to all necessary types of objects.

8. Conclusion

We have described some initial results of our work is this area. Currently we are investigating various classes of continuous finite-response filters in order to select viable candidates for an experimental implementation of a small graphics package exploiting the principles set in this paper.

References

[1] Foley J.D., van Dam A., Feiner S.K., Hughes J.F., *Computer Graphics - Principles and practice*, Addison-Wesley, 1990.

[2] Corner, J.B., *Return of the Jaggy*, IEEE Computer Graphics **9 No 2** (March 1989), 82 – 89.

[3] Jerri A.J., *The Shannon sampling theorem – its various extensions and application: A tutorial review*, Proc. IEEE **65 No 11** (Nov. 1977), 1565 – 1596.

[4] Castleman, K.R., *Digital Image Processing*, Prentice-Hall, Inc, New Jersey, 1979.

[5] Max N.L., *Antialiasing scan-line data*, IEEE Computer Graphics and Aplications **10 No 1** (January 1990), 18 – 30.

[6] INMOS Limited, *The T9000 Transputer Instruction Set Manual* (1993), Bristol.

[7] INMOS Limited, *The T9000 Transputer Hardware Reference Manual* (1993), Bristol.

Božidara Adžije 19/I-11, 18000 Niš.
E-mail address: sike@unitop.elfak.ni.ac.yu

Faculty of occupational safety, Čarnojevića 10, 18000 Niš.

Filozofski fakultet, Ćirila i Metodija 2, 18000 Niš.
E-mail address: vesna@archimed.filfak.ni.ac.yu

FILOMAT (Niš) 9:2(1995), 261–271
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

HALLEY-LIKE ASYNCHRONOUS METHODS FOR POLYNOMIAL ROOTS

M. Trajković, S. Tričković and M. Petković

ABSTRACT. *In this paper we present the asynchronous implementation of Halley-like method for the simultaneous approximation of polynomial roots on a distributed memory multicomputer. It is shown that the lower bound of the order of convergence of asynchronous Halley-like method with the delay r is at least $\eta_A > 3$, where η_A is the unique positive root of the equation $\eta^{r+1} - 3\eta^r - 1 = 0$. The computational efficiency of the synchronous and asynchronous versions are studied in the case of hypercube topology.*

1. Some preliminary results

Simultaneous methods for the determination of polynomial roots run in several identical versions so that they are very convenient for the implementation on parallel computers (see, e.g., [**4,5,6,8,9,10**]). All n roots are found simultaneously, n versions of the same algorithm can be run on a distributed memory multicomputer consisting of k ($\leq n$) processors. The main advantage of parallel implementation comes from the fact that a great deal of computation can be performed simultaneously. The details concerning an application of simultaneous methods on parallel computers may be found in [**4,5,6,7**].

In practical implementation of simultaneous methods on parallel computers three standard network topologies are usually applied: rings, torus and hypercubes. The models assume k processors connected through a regular graph of diameter D and degree d. The efficiency of these methods depends on three parameters: the computation time of any arithmetical operation modeled by τ_a, the communication start up β_c and the throughput of the

1991 *Mathematics Subject Classification.* 65H05, 65W05.

links τ_c. Typical values of τ_c, τ_a and β_c for several types of multiprocessors can be found in the paper [**3**]. Since in our analysis we neglect the times for computing the starting points and checking the stopping criteria, the computational cost of algorithms is the sum of a computation time with a communication time. Besides, the number of basic arithmetical operations of the applied method appears as an additional parameter in the analysis of the total computational cost. This number for a wide class of simultaneous iteration methods can be given in the form (see [**12**, Ch. **6**])

$$N(n) = \alpha n^2 + \beta n + \gamma,$$

where n is the polynomial degree. If n is sufficiently large, then we can take approximately that $N(n) = \alpha n^2 + o(n^2)$.

Following [**6**], the total time for the synchronous implementation can be expressed as the sum of the computation time $N(n)\tau_a/k$ and the communication time $D\beta_c + O(n/d)\tau_c$, that is

$$T_{\text{syn}} = \Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a + D\beta_c + O\big(\frac{n}{d}\big)\tau_c. \tag{1}$$

The communication time cannot be neglected in a synchronous implementation; moreover, it has a great influence on the total execution time and appears to be a major drawback of this parallelization of the simultaneous methods. In order to decrease the communicate time the following strategy can be applied [**2,7,10**]: In each iteration, a processor does not have to wait at predetermined points, for example, the end of the total-exchange, for predetermined messages to become available. This type of algorithms is called **asynchronous** by Baudet [**1**] indicating that, at each step, the local computation is performed using only a part of the global information.

Let $m = 0, 1, 2, \ldots$ be the iteration index and let us assume that the new approximation $z_i^{(m+1)}$ is calculated by a processor P_h, $h \in \{1, \ldots, k\}$. Evidently, to force the convergence, this processor *must know* the value of $z_i^{(m)}$. The improved approximation $z_i^{(m+1)}$ is calculated by a general iteration formula

$$z_i^{(m+1)} = F_i(\mathbf{z}^{(m^*)}), \quad \text{where} \quad \mathbf{z}^{(m^*)} = (z_1^{(m-r(1,m,h))}, \ldots, z_n^{(m-r(n,m,h))}). \tag{2}$$

In (2) $\mathbf{z}^{(m^*)}$ is the vector of the last values z_j known by the processor P_h at step m, represented by $z_j^{(m-r(j,m,h))}$. Here $r(j,m,h)$ is a **delay** depending on j, m and h and indicating that the processor P_h only knows the value of z_j computed at step $m - r(j,m,h)$. The maximum delay will be denoted by r, that is, $r = \max_{j,m,h} r(j,m,h)$.

The implementation of an asynchronous method is executed in such a way that, at each iteration step, a processor sends the most recently computed entries to its neighbors only, decreasing the communication time. As it was presented in [**6**], the total time per one iteration step is

$$T_{\text{asy}} = \Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a + \beta_c + O\big(\frac{n}{k}\big)\tau_c. \tag{3}$$

Comparing with (1), we obtain **one** start up instead of D and a propagation time of $O(n/k)$ instead of $O(n/d)$.

Let N_{syn} and N_{asy} be respectively the number of iteration steps of a synchronous and an asynchronous method. Evidently, the asynchronous method will be more efficient if $N_{\text{asy}}T_{\text{asy}} < N_{\text{syn}}T_{\text{syn}}$. By virtue of (1) and (3) this inequality may be written as follows:

$$\begin{aligned} N_{\text{asy}}\Big[\Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a + \beta_c + O\big(\frac{n}{k}\big)\tau_c\Big] < N_{\text{syn}}\Big[\Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a \\ + D\beta_c + O\big(\frac{n}{d}\big)\tau_c\Big]. \end{aligned} \tag{4}$$

Let us suppose that the inequality

$$\frac{\beta_c}{\tau_a} < \frac{\alpha n^2}{k} \tag{5}$$

holds. Namely, if $\beta_c/\tau_a > \alpha n^2/k$, then it could be faster to use less processors in the synchronous implementation (see [**6**]). Furthermore, since the relation $\beta_c \gg \tau_c$ generally occurs ([**3**]) on distributed memory computers, the inequality (4) becomes

$$\frac{N_{\text{asy}}}{N_{\text{syn}}} < \frac{\dfrac{\alpha n^2}{k} + D\dfrac{\beta_c}{\tau_a}}{\dfrac{\alpha n^2}{k} + \dfrac{\beta_c}{\tau_a}}. \tag{6}$$

Eventual validity of the inequality (6) can be suitable verified by a graphical interpretation in the plane $\big(N_{\text{asy}}/N_{\text{syn}}, \beta_c/\tau_a\big)$. Let $R = \beta_c/\tau_a$ denote a realistic parametric ratio depending on the applied network topology. For the hypercube topology this ratio usually belongs to the interval $[10^2, 10^3]$ (see [**3**]). Conditions for the dominance of asynchronous implementation has been discussed in [**13**]. Dominant area is bounded above by the curve

$$\frac{\beta_c}{\tau_a} = \frac{\dfrac{\alpha n^2}{k}\Big(1 - \dfrac{N_{\text{asy}}}{N_{\text{syn}}}\Big)}{\dfrac{N_{\text{asy}}}{N_{\text{syn}}} - D}, \tag{7}$$

which are obtained from (6) taking the sign "=" instead of "<", and the dashing line $\beta_c/\tau_a = R$.

Let V is the **critical ratio** given by the abscissa of the intersection of the curve (7) and the dashing line $\beta_c/\tau_a = R$, that is, $V = 1+(D-1)/(\frac{\alpha n^2}{kR}+1)$. As pointed out in [**13**], an asynchronous algorithm will be more efficient if the realistic ratio R is smaller than the bound value $\beta_c/\tau_a = \alpha n^2/k$ and, in addition, if the ratio of iteration steps $N_{\mathbf{asy}}/N_{\mathbf{syn}}$ can be realized in practice, that is, if $N_{\mathbf{asy}}/N_{\mathbf{syn}} < V$. We note that the following estimate for the ratio $N_{\mathbf{asy}}/N_{\mathbf{syn}}$ has been derived in [**13**]:

$$\frac{N_{\mathbf{asy}}}{N_{\mathbf{syn}}} \cong \frac{\log \eta_S}{\log \eta_A}. \tag{8}$$

2. Convergence analysis of asynchronous Halley-like method

Let $\epsilon_m^{(m)} = z_i^{(m)} - \zeta_i$ be the error of an asynchronous method of the form (2) which generates the sequences $(z_i^{(m)})$ of approximations to the roots $\zeta_1, \dots, \zeta_n$. As mentioned in [**13**], for a wide class of iteration methods for the simultaneous determination of polynomial roots the following relation can be derived:

$$\epsilon_i^{(m+1)} = \alpha_i \left(\epsilon_i^{(m)}\right)^q \sum_{\substack{j=1 \\ j\neq i}}^{n} \beta_{ij}\epsilon_j^{(m-r(j,m,h))} \quad (i=1,\dots,n), \tag{9}$$

where α_i and β_{ij} are complex constants and $q \geq 1$ is integer. In that case following general convergence theorem has been proved in [**13**]:

Theorem 1. *Suppose that a polynomial P has only simple roots $\zeta_1, \dots, \zeta_n$ and starting approximations $z_1^{(0)}, \dots, z_n^{(0)}$ are reasonably close to these roots. Further, assume that $r(j,m,h)$ is bounded for all $j = 1, \dots, n$ and all $h = 1, \dots, k$. Then the asynchronous algorithm (2) for which the relations (9) are valid is locally convergent with the order of convergence at least $\eta_A(q) > q$, where $\eta_A(q)$ is the unique positive root of the equation*

$$\eta^{r+1} - q\eta^r - 1 = 0, \quad r = \max_{j,m,h} r(j,m,h). \tag{10}$$

In this section we give a convergence analysis of the asynchronous Halley-like root finding method and its efficiency compared to the synchronous implementation. Hypercube topology will be considered as the most efficient network topology for this kind of problems.

Halley-like method for the simultaneous approximation of all zeros of a polynomial P of degree n has been considered in [14] and [11]. For simplicity, the approximations $z_j^{(m-r)}$ to the roots $\zeta_1, \dots, \zeta_n$ at the iteration step m will be shortly denoted with z_j if $r = 0$ and z_j^* if $r > 0$. According to this notation we introduce the errors $\epsilon_i = z_i - \zeta_i$ and $\epsilon_j^* = z_j^* - \zeta_j$. The new approximation $z_i^{(m+1)}$ will be denoted with $\hat{z}_i$ and the corresponding error with $\hat{\epsilon}_i = \hat{z}_i - \zeta_i$. Besides, we define the sums

$$S_{\lambda,i} = \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{1}{(z_i - z_j^*)^\lambda} \quad (i = 1, \dots, n;\ \lambda = 1, 2),$$

$$\Sigma_{1,i} = \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{1}{z_i - \zeta_j} \quad (i = 1, \dots, n),$$

the abbreviations

$$a_{ij} = (z_i - \zeta_j)(z_i - z_j^*), \quad b_{ij} = 2z_i - z_j^* - \zeta_j,$$

and the function

$$f(z) = \frac{P'(z)}{P(z)} - \frac{P''(z)}{2P'(z)}.$$

Then Halley-like method reads:

$$\hat{z}_i = z_i - \frac{1}{f(z_i) - \dfrac{P(z_i)}{2P'(z_i)}\left[S_{1,i}^2 + S_{2,i}\right]} \quad (i = 1, \dots, n) \tag{11}$$

or in the form

$$\hat{z}_i = z_i - \frac{\dfrac{2P'(z_i)}{P(z_i)}}{\left[\dfrac{P'(z_i)}{P(z_i)}\right]^2 + \dfrac{P'(z_i)^2 - P''(z_i)P(z_i)}{P(z_i)^2} - S_{1,i}^2 - S_{2,i}}. \tag{12}$$

By the way, we observe that the function f in the denominator of (11) appears in the well-known Halley iteration formula

$$\hat{z} = z - \frac{1}{f(z)}$$

for the determination of a single root.

In the following we will show that the asynchronous Halley-like method (11) belongs to the class of methods for which the relation (9) holds, which means that Theorem 1 can be also applied to this method. For that purpose we use the identities

$$\frac{P'(z)}{P(z)} = \sum_{j=1}^{n} \frac{1}{z-\zeta_j} \tag{13}$$

and

$$\frac{P'(z)^2 - P''(z)P(z)}{P(z)^2} = \sum_{j=1}^{n} \frac{1}{(z-\zeta_j)^2}, \tag{14}$$

which can be easily derived by logarithmic differentiation.

First, from (13) we have

$$\frac{2P'(z_i)}{P(z_i)} = 2\Big(\frac{1}{z_i-\zeta_i} + \sum_{j\neq i} \frac{1}{z_i-\zeta_j}\Big) = 2\Big(\frac{1}{\epsilon_i} + \Sigma_{1,i}\Big)$$

and

$$\begin{aligned}\Big(\frac{P'(z_i)}{P(z_i)}\Big)^2 - S_{1,i}^2 &= \left(\sum_{j=1}^{n} \frac{1}{z_i-\zeta_j}\right)^2 - \left(\sum_{j\neq i} \frac{1}{z_i-z_j^*}\right)^2 \\ &= \Big(\frac{1}{\epsilon_i} - \sum_{j\neq i} \frac{\epsilon_j^*}{a_{ij}}\Big)\Big(\frac{1}{\epsilon_i} + S_{1,i} + \Sigma_{1,i}\Big).\end{aligned}$$

Using the identity (14) we find

$$\frac{P'(z_i)^2 - P''(z_i)P(z_i)}{P(z_i)^2} - S_{2,i} = \sum_{j=1}^{n} \frac{1}{(z_i-\zeta_j)^2} - \sum_{j\neq i} \frac{1}{(z_i-z_j^*)^2} = \frac{1}{\epsilon_i^2} - \sum_{j\neq i} \frac{b_{ij}\epsilon_j^*}{a_{ij}^2}.$$

According the two last relations we find from (12)

$$\hat{z}_i - \zeta_i = z_i - \zeta_i - \frac{2\Big(\frac{1}{\epsilon_i} + \Sigma_{1,i}\Big)}{\Big(\frac{1}{\epsilon_i} - \sum_{j\neq i} \frac{\epsilon_j^*}{a_{ij}}\Big)\Big(\frac{1}{\epsilon_i} + S_{1,i} + \Sigma_{1,i}\Big) + \frac{1}{\epsilon_i^2} - \sum_{j\neq i} \frac{b_{ij}\epsilon_j^*}{a_{ij}^2}},$$

that is,

$$\hat{\epsilon}_i = \epsilon_i - \frac{2\epsilon_i(1+\epsilon_i\Sigma_{1,i})}{\Big(1-\epsilon_i\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}}\Big)\big(1+\epsilon_i S_{1,i}+\epsilon_i\Sigma_{1,i}\big)+1-\epsilon_i^2\sum_{j\neq i}\frac{b_{ij}\epsilon_j^*}{a_{ij}^2}}$$

$$= \epsilon_i - \frac{2\epsilon_i + 2\epsilon_i^2\Sigma_{1,i}}{2+\epsilon_i\Big(S_{1,i}+\Sigma_{1,i}-\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}}\Big)-\epsilon_i^2\Big[(S_{1,i}+\Sigma_{1,i})\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}}+\sum_{j\neq i}\frac{b_{ij}\epsilon_j^*}{a_{ij}^2}\Big]}.$$

Let G_{ij} denotes the denominator in the last relation. After short rearrangement of the previous relation we obtain

$$\hat{\epsilon}_i = \frac{\epsilon_i^2}{G_{ij}}\Big\{S_{1,i}-\Sigma_{1,i}-\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}}-\epsilon_i\Big[(S_{1,i}+\Sigma_{1,i})\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}}+\sum_{j\neq i}\frac{b_{ij}\epsilon_j^*}{a_{ij}^2}\Big]\Big\}. \quad (15)$$

Since

$$S_{1,i}-\Sigma_{1,i}-\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}} = \sum_{j\neq i}\Big(\frac{1}{z_i-z_j^*}-\frac{1}{z_i-\zeta_j}-\frac{z_j^*-\zeta_j}{(z_i-z_j^*)(z_i-\zeta_j)}\Big)=0,$$

from (15) there follows

$$\hat{\epsilon}_i = -\frac{\epsilon_i^3}{G_{ij}}\Big[(S_{1,i}+\Sigma_{1,i})\sum_{j\neq i}\frac{\epsilon_j^*}{a_{ij}}+\sum_{j\neq i}\frac{b_{ij}\epsilon_j^*}{a_{ij}^2}\Big]$$

or in the form

$$\hat{\epsilon}_i = \epsilon_i^3\sum_{j\neq i}c_{ij}\epsilon_j^*, \quad \text{with } c_{ij} = -\frac{a_{ij}(S_{1,i}+\Sigma_{1,i}))+b_{ij}}{a_{ij}^2 G_{ij}}. \quad (16)$$

The quantities $a_{ij}, b_{ij}, S_{1,i}$ and $\Sigma_{1,i}$ are bounded, namely $a_{ij}\to(\zeta_i-\zeta_j)^2$, $b_{ij}\to 2(\zeta_i-\zeta_j)$, while $S_{1,i}$ and $\Sigma_{1,i}$ tend to $\Sigma_{j\neq i}(\zeta_i-\zeta_j)^{-1}$. Also, assuming that the starting approximations are sufficiently close to the exact zeros, the quantities ϵ_i will be small enough so that there exists a positive number $\mu<2$ such that $|G_{ij}|>2-\mu$. Therefore, c_{ij} is bounded in modulus; hence, the relation (16) is of the form (9) with $q=3$ so that we can directly applied the assertion of Theorem 1. Thus, the order of convergence of the asynchronous Halley-like method is at least η_A, where $\eta_A>3$ is the unique positive root of the equation $\eta^{r+1}-3\eta^r-1=0$. Since we assume reasonably good starting approximations, because of the very fast convergence the total number of iteration steps will be rather small (2 or 3 steps in practice). For these reason, greater values of r should not be expected.

3. Comparison of asynchronous and synchronous version

In this section we present a theoretical implementation of Halley-like method (11) on 4-dimensional hypercube with $k = 2^4 = 16$ processors where the diameter is $D = 4$. We take a realistic parameters ratio [**3**] $\beta_c/\tau_a \cong 10^3/6$ (dashing horizontal line). The total arithmetic cost of Halley-like method is $42n^2\tau_a + o(n^2)$, that is, $\alpha = 42$. Polynomials of the degrees $n = 16$ (the so-called full parallelization when the number of processors is equal to the polynomial degree) and $n = 30$ have been considered. The bound values $\beta_c/\tau_a = \frac{21}{8}n^2$ for these values of n are represented by the full horizontal lines in Fig. 1.

Realistic areas where the asynchronous Halley-like method can be more efficient are given by light shaded area for $n = 16$ and darker shaded area (partially invisible) for $n = 30$. The critical values which determine the necessary upper bound ratio $N_{\mathrm{asy}}/N_{\mathrm{syn}}$ are given by V_1 for $n = 16$ and V_2 for $n = 30$. Obviously, a more stronger requirement for the needed ratio $N_{\mathrm{asy}}/N_{\mathrm{syn}}$ appears in the case of the higher degree; namely, this ratio is closer to 1 when the degree n is higher, which is more difficult to realize in practice. Following (8) we find for the worst case model ($\eta_A = 3$) that the ratio of the number of iteration steps $N_{\mathrm{asy}}/N_{\mathrm{syn}}$ which provides a greater efficiency of asynchronous implementation must be smaller than $\log 4/\log 3 \cong 1.26$. For the considered ratio $\beta_c/\tau_a = 10^3/6$ this is available (theoretically) if $n < 26$. On the other side, the higher n permits the topology with the greater ratio β_c/τ_a (see Fig. 1).

Finally, we wish to consider a more general problem. We recall that *Durand-Kerner* method (with a quadratic convergence) in a synchronous implementation have the best performances in a wide class of simultaneous methods although it possesses relatively low convergence rate (see [**5,7**]). The following question arises: *What is the influence of the convergence rate of applied methods in a practical realization when the parallel implementation is performed asynchronously?* In other words, we wish to investigate the case when (from (8))

$$\lambda_q = \frac{N_{\mathrm{asy}}}{N_{\mathrm{syn}}} \cong \frac{\log \eta_S}{\log \eta_A} = \frac{\log(q+1)}{\eta_A(q)} < V, \tag{17}$$

in dependendence on the parameter q which defines the convergence orders of synchronous and asynhronous versions. Here $\eta_A(q)$ is the unique positive root of the equation (10) and V is the critical ratio which is the upper bound of the possible area of dominance of the asynchronous version (see Section 1 and Fig. 1). For this purpose, we have solved the equation (10) and found the ratio λ_q for the delay $r = 0, 1, 2, 3, 4$ and the entries $q = 1, 2, 3, 4$ which are of a practical importance. The dependence λ_q against q with the delay r as a parameter is displayed in Fig. 2.

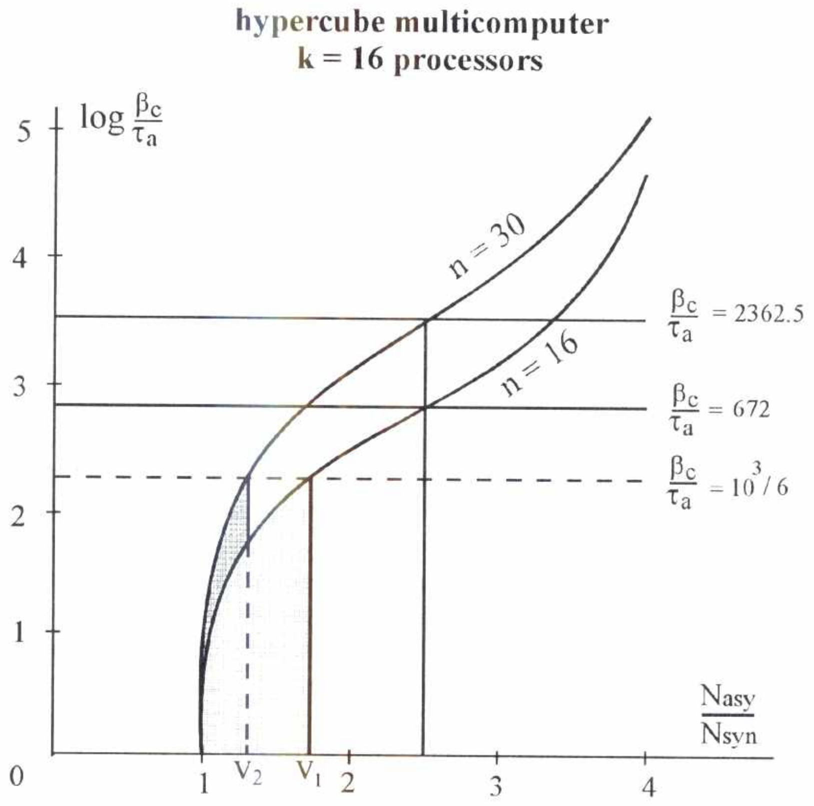


Fig. 1 Dominant areas of Halley-like asynchronous method

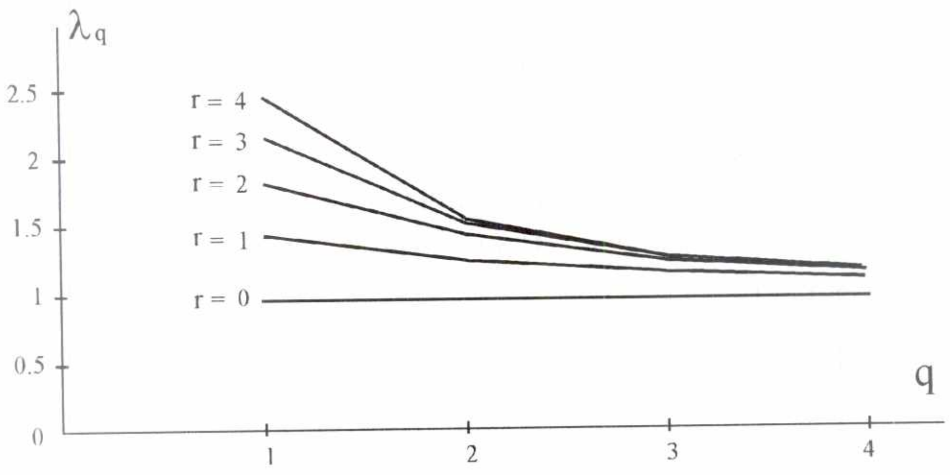


Fig. 2 The ratio of the iteration steps as a function of the convergence order

From Fig. 2 we observe that, for all r, that the ratio λ_q of iteration steps is smaller for a greater q, that is, in the case of methods of higher order. But, under fixed real network performances, a smaller ratio $\lambda = N_{\text{asy}}/N_{\text{syn}}$ means that the inequality (8) (and, accordingly, (17)) is feasible much easier. Hence, the possibility that an asynchronous algorithm be more efficient than the corresponding synchronous algorithm is greater if the basic method has a higher convergence order. This fact gives a slight advantage of Halley-like method (which is of the fourth order) compared to Durand-Kerner method (quadratic convergence) and Ehrlich-Aberth method (cubic convergence) but only in the case of the asynchronous implementation.

References

[1] G.M. Baudet, *Asynchronous iterative methods for multiprocessors.* J. of ACM **2** (1978), 226-244.

[2] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and distributed computation - numerical methods.* Prentice-Hall Inc. 1989.

[3] L. Bomans and D. Roose, *Communication benchmarks for the iPSC/2.* Hypercube and Distributed Computers (Proc. I European Workshop on hypercube and Distributed Computers, eds. F. Andre and J. P. Verjus), North Holland, Amsterdam 1989, pp. 93-104.

[4] M. Cosnard and P. Fraigniaud, *Asynchronous Durand-Kerner and Aberth polynomial root finding methods on a distributed memory multicomputer.* Parallel Computing **9** (1989) 79-84.

[5] M. Cosnard and P. Fraigniaud, *Finding the roots of a polynomial on an MIMD multicomputer.* Parallel Computing **15** (1990) 75-85.

[6] M. Cosnard and P. Fraigniaud, *Asynchronous polynomial root finding methods.* Research report 90-21, LIP-IMAG, Ecole Normale Supérieure de Lyon, France 1990.

[7] M. Cosnard and P. Fraigniaud, *Analysis of asynchronous polynomial root finding methods on a distributed memory multicomputer.* IEEE Transaction on Parallel and Distributed Systems (to appear).

[8] P. Fraigniaud, *Performance analysis of broadcasting in hypercubes.* Hypercube and Distributed Computers (Proc. I European Workshop on hypercube and Distributed Computers, eds. F. Andre and J. P. Verjus), North Holland, Amsterdam 1989, pp. 311-328.

[9] T.L. Freeman, *Calculating polynomial zeros on a local memory parallel computer.* Parallel Computing **12** (1989) 351-358.

[10] T.L. Freeman and M.K. Bane, *Asynchronous polynomial zero-finding algorithms.* Parallel Computing **17** (1991) 673-681.

[11] M.S. Petković, *On Halley-like algorithms for simultaneous approximation of polynomial complex zeros.* SIAM J. Numer. Anal. **3** (1989), 740-763.

[12] M.S. Petković, *Iterative methods for simultaneous inclusion of polynomial zeros.* Springer-Verlag, Berlin-Heidelberg-New York 1989.

[13] S. Tričković, M. Trajković and M. Petković, *Asynchronous methods for simultaneous determination of polynomial roots* (submitted).

[14] X. Wang and S. Zheng, *A family of parallel and interval iterations for finding all roots of a polynomial with rapid convergence (I).* J. Comput. Math. **1** (1984), 70-76.

Faculty of Electronic Engineering, P.O. Box 73, 18 000 Niš

FILOMAT (Niš) 9:2(1995), 273–284
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ASYNCHRONOUS METHODS FOR SIMULTANEOUS DETERMINATION OF POLYNOMIAL ROOTS

S. Tričković, M. Trajković and M. Petković

ABSTRACT. *In this paper we present the implementation of simultaneous method for the determination of polynomial roots on a distributed memory multicomputer. The total cost of such a parallelization per iteration is the sum of a computation time and a communication time needed for a total exchange of the data at each iteration step. In order to decrease the communication time, an asynchronous implementation is considered. The computation of the root approximations is still shared among processors but the updating is performed using only nearest neighbor communications. The price to be paid to decrease this time consists in reducing the order of convergence of asynchronous methods. A general theorem which consider the lower bound of the order of convergence is given. Also, the computational efficiency of the synchronous and asynchronous versions are studied in the case of hypercube topology.*

1. Introduction

Mathematical models in scientific engineering including digital signal processing or automatic control reduce to the problem of finding roots of polynomials with degree 100 and higher [**15,16**]. In these cases the parallel processing becomes of great interest to speed up the determination of roots.

In practice, all methods for finding polynomial roots can be divided (although not strictly) in three classes: analytic, geometric and algebraic. Parallel implementation of geometric or algebraic methods often requires fine grain parallelism (see, e.g. [**2,17**]). On the other side, the multicomputers are rather composed of a network of processors with distributed memory which assumes their coarse grain parallelism. For this reason, we are interested here in analytic methods and their application on a distributed MIMD

1991 *Mathematics Subject Classification.* 65H05, 65W05.

machine. Moreover, we will restrict our study to iteration methods for the simultaneous calculation of all roots of a polynomial. As it is well known, these methods run in several identical versions so that they are very suitably for the implementation on parallel computers (see, e.g., [**6,7,8,9,11,12,13**]). All n roots are found simultaneously, n versions of the same algorithm can be run on a Multiple Instruction/Multiple Data (MIMD) parallel computer consisting of k ($\leq n$) processors. The main advantage of parallel implementation is that a great deal of computation can be done simultaneously. The details concerning an application of simultaneous methods on parallel computers, including an analysis of total running time of a parallel iteration, the determination of the optimal number of processors as well as experimentations, may be found in [**6,7,8,9**].

Many of the simultaneous methods can be written in the form

$$\mathbf{z}^{(m+1)} = F(\mathbf{z}^{(m)}), \tag{1}$$

where F is an operator in $\mathbb{C}^n$, $\mathbf{z}^{(m)} = (z_1^{(m)}, ..., z_n^{(m)})$ is a vector of approximations to the roots $\zeta_1, ..., \zeta_n$ of a given polynomial P of degree n with any initial vector $\mathbf{z}^{(0)}$. In this paper we will always assume that the initial vector $\mathbf{z}^{(0)}$ is chosen so that all $z_i^{(m)}$'s tend to the ζ_i's $(i = 1, ..., n)$. For the construction and a detailed study of simultaneous methods see the book [**18**].

As we have noted, we are concerned with the distributed memory multicomputers. Such parallel computers are modeled by a connected graph. The vertices of this graph are the processors and its edges are the communication links. The exchange of data between two nodes which are not directly connected must pass through different other nodes. Hence, the communication strategy has a great influence to the efficiency of the applied method. Our aim is to establish such a strategy which will decrease the communication time and, at the same time, preserve the computational efficiency of the implemented method.

In practice, the implementation of simultaneous methods on parallel computers is usually performed by three standard network topologies: rings, torus and hypercubes. In short, the model is as follows: k is the number of processors connected through a regular graph of diameter D and degree d. The exchange cost of length L messages between two neighbor processors is the sum of a start up β_c and a propagation time proportional to the message length $L\tau_c$, that is $T_{\text{One-to-one}} = \beta_c + L\tau_c$. Furthermore, one assumes that a processor can communicate simultaneously with all its neighbors (link-bound model), and that the links are full duplex. The arithmetic cost is modeled by the computation time τ_a, where τ_a is usually the mean of a floating point addition and the floating point multiplication. Typical values of τ_c, τ_a and β_c for several types of multiprocessors can be found in the paper [4]. We emphasize that in our analysis the times for computing the starting points and checking the stopping criteria will be neglected. Accordingly, the computational cost of algorithms is the sum of a computation time with a communication time.

The investigation based on the parameters τ_c, τ_a and β_c shows that the network topology has a great influence on the global cost of parallel methods. It has been shown in [7] that, among three mentioned standard topology, the hypercube is the best topology and the relation

$$T_{\text{total}}^{\text{hypercube}} \leq T_{\text{total}}^{\text{torus}} \leq T_{\text{total}}^{\text{ring}}$$

holds. Another advantage of the hypercube topology appears when full parallelism (the degree of a polynomial is equal to the number of processors) is available. Namely, the communication time of an iteration grows linearly with the degree on a ring of processors, with the square root of the degree for a torus, but only logarithmically on a hypercube.

2. Implementation of synchronous methods

Before demonstrating the strategy which decreases the communication time, we present the implementation of so-called **synchronous parallel methods** (like (1)) [8,9]. The term **"synchronous"** does not refer here to the control mode of the multicomputer, but refer to the structure of the algorithm. Actually, considering the iteration formula (1), the next approximate vector $\mathbf{z}^{(m+1)}$ is calculated using the most recent components of $\mathbf{z}^{(m)}$.

In the parallelization of the parallel algorithm (1) we assume that the number of processors k $(\leq n)$ is given in advance. The starting vector $\mathbf{z}^{(0)}$ is computed by all the processors $P_1, \dots, P_k$ using some suitable search procedure (see, e.g., [5,10,14]). Furthermore, each step of the algorithm consists in sharing the computation of n improved approximations $z_1^{(m)}, \dots, z_n^{(m)}$ among the processors and in updating their data $\mathbf{z}^{(m)}$ through a broadcast procedure (shorter $BCAST(\mathbf{z}^{(m)})$). As in [7], let $I_1, \dots, I_k$ be disjunctive partitions of the set $\{1, \dots, n\}$ where $\cup I_j = \{1, \dots, n\}$. To obtain good load balancing between the processors, the index sets $I_1, \dots, I_k$ are chosen so that the number of their components $\omega(I_j)$ $(j = 1, \dots, k)$ is determined as $\omega(I_j) \leq \left[\frac{n}{k}\right]$. At the m-th iteration step the processor P_j $(j = 1, ..., k)$ computes $z_i^{(m)}$ for all $i \in I_j$ by the iteration formula (1) and then it transmits these values to all other processors using a broadcast procedure (referred to as $BCAST(\mathbf{z}^{(m)})$. The program terminates when some stopping criterion (referred to as $STOP(\mathbf{z}^{(m)})$) is fulfilled, for instance, if

$$\max_{1\le i\le n}\left|P\big(z_i^{(m)}\big)\right| < \delta$$

for a given sufficiently small δ. According to the previous we give a program in pseudocode for a parallel implementation of a simultaneous method (1) (following [7]):

```
     Program SYNCHRONOUS SIMULTANEOUS METHOD
     begin
          for all $j = 1,\dots,k$ do determination of the starting
          approximations $\mathbf{z}^{(0)}$;
          $m := 0$
          do
               for all $j = 1,\dots,k$ do in parallel
               begin
(*)                 Compute $z_i^{(m+1)} := F_i(\mathbf{z}^{(m)})$, $i \in I_j$
(**)                Communication: $BCAST\big(\mathbf{z}^{(m+1)}\big)$;
               end
               $m := m + 1$
          until $STOP\big(\mathbf{z}^{(m)}\big)$ holds true;
          OUTPUT $\mathbf{z}^{(m)}$
     end
```

As it was presented in [**18**, Ch. **6**], the number of basic arithmetic operations for a wide class of simultaneous iteration methods can be given in the form

$$N(n) = \alpha n^2 + \beta n + \gamma,$$

where n is the polynomial degree and α, β and γ are integers. Dealing with sufficiently large n we can take approximately that $N(n) = \alpha n^2 + o(n^2)$. Following [**8**], the total time for the presented implementation of the synchronous method can be expressed as the sum of the computation time $N(n)\tau_a/k$ and the communication time $D\beta_c + O(n/d)\tau_c$, that is

$$T_{\text{syn}} = \Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a + D\beta_c + O\big(\frac{n}{d}\big)\tau_c. \tag{2}$$

From (2) we see that the communication time cannot be neglected; moreover, it has a great influence on the total execution time and appears to be a major shortcoming of this parallelization of the simultaneous methods.

3. Asynchronous simultaneous methods

In order to decrease the communicate time the following strategy can be applied [**3**,**9**,**13**]: In each iteration, a processor does not have to wait at predetermined points, for example, the end of the total-exchange, for predetermined messages to become available. This type of algorithms is called **asynchronous** by Baudet [**1**]. The term **"asynchronous"** only refers to the fact that, at each step, the local computation is performed using only a part of the global information. An asynchronous algorithm can be modeled as follows:

Assume that the new approximation $z_i^{(m+1)}$ is calculated by a processor P_h, $h \in \{1,...,k\}$. Evidently, this processor *must know* the value of $z_i^{(m)}$ and $z_i^{(m+1)}$ is calculated by the formula

$$z_i^{(m+1)} = F_i(\mathbf{z}^{(m^*)}), \quad \text{where} \quad \mathbf{z}^{(m^*)} = (z_1^{(m-r(1,m,h))}, \dots, z_n^{(m-r(n,m,h))}). \tag{3}$$

In (3) $\mathbf{z}^{(m^*)}$ is the vector of the last values z_j known by the processor P_h at step m, represented by $z_j^{(m-r(j,m,h))}$. Here $r(j,m,h)$ is a **delay** depending on j, m and h and indicating that the processor P_h only knows the value of z_j computed at step $m - r(j,m,h)$. In the sequel, the maximum delay will be denoted by r, that is, $r = \max_{j,k,h} r(j,m,h)$.

The presented asynchronous algorithm (3) will run if the following strategy of distribution of the indices is chosen:

1. $z_i^{(m+1)}$ is calculated by only one processor for all $i = 1,...,n$;

2. $r(i,m,h) = 0$, that is, the processor P_h must know $z_i^{(m)}$, $i \in I_h$.

Hypothesis 1 insures that there is no redundancy in the computation. Hypothesis 2 has already been discussed and it must be satisfied to provide the convergence of the sequence $(z_i^{(m)})$. Besides, in regard to Hypothesis 1, this implies that at step m, if $z_i^{(\mu)}$ $(0 \le \mu \le m,\ i \in \{1,\dots,n\})$ is known by a set of processors, its value is the same for all these processors.

A short analysis given in [**9**] shows that, excepting hypotheses 1 and 2, some additional conditions must be satisfied. Namely, if each processor always updates the same components, then each processor will know the most recent entries of the components which are updated in its neighborhood, but the other components will never be updated. This causes that, for each $h \in \{1,...,k\}$ and at each iteration step m, there exists j such that the processor P_h knows only the value of $z_j^{(0)}$, that is, $r(j,m,h) = m$. This fact implies that the convergence is not insured. For this reason the strategy of the implementation of asynchronous methods should provide such indices distribution that I_h be different in each iteration step and the delay $r(j,m,h)$ is bounded above by some $\rho < m$. If $I(h,m)$ denotes the set of indices of the components updated by the processor P_h at step m, then the mentioned conditions can be expressed as follows:

(i) $I(h,m)$ $(h = 1,\dots,k)$ form a partition of $(1,\dots,n)$ at each step m;

(ii) If $i \in I(h,m)$, then the processor P_h knows $z_i^{(m)}$, $i \in I_h$;

(iii) For each processor P_h and for each component i there exists an integer ρ such that the sets $I(h,m)$ have the property that the number of steps separating two evaluations of this component in the neighborhood of h does not exceed the delay ρ.

In this way, (i) and (ii) imply that the hypotheses 1 and 2 respectively are satisfied, while (iii) provides the convergence with $r = \rho$. If these conditions are fulfilled, then a a program in pseudocode for a parallel implementation of an asynchronous simultaneous method (3) is as follows:

```
    Program ASYNCHRONOUS SIMULTANEOUS METHOD
    begin
        for all $j = 1,\dots,k$ do determination of the starting
        approximations $\mathbf{z}^{(0)}$;
        $m := 0$
        do
            for all $j = 1,\dots,k$ do in parallel
            begin
(0)             Compute $I(h,m)$;
(1)             Compute $z_i^{(m+1)} := F_i(\mathbf{z}^{(m)})$, $i \in I(h,m)$
(2)             Send $z_i$, $i \in I(h,m)$, to neighbors;
            end
            $m := m + 1$
(3)     until $STOP(\mathbf{z}^{(m)})$ holds true;
        OUTPUT $\mathbf{z}^{(m)}$
    end
```

The checking of the stoping criteria (3) is more difficult in the case of the asynchronous implementation since there are the possibility that some processors verify the stop condition but not the other ones. For more details about the detection of termination see [**3**]. We only note that the step (3) has to include such a strategy which synchronize the processors, namely, the first processors which terminate may signal the end of the execution to the others.

The implementation of an asynchronous method is executed in such a way that, at each iteration step, a processor sends the most recently computed entries to its neighbors only. As in the case of synchronous algorithm, the computation time is again $N(n)\tau_a/k$, but the communication time becomes $\beta_c + O(n/k)\tau_c$ since it corresponds to sending n/k values from each nodes to their neighbors in parallel. Therefore, the total time per one iteration step is

$$T_{\text{asy}} = \Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a + \beta_c + O\Big(\frac{n}{k}\Big)\tau_c. \tag{4}$$

Comparing with (2), we obtain **one** start up instead of D and a propagation time of $O(n/k)$ instead of $O(n/d)$. Thus, the communication cost is decreased by $(D-1)\beta_c + O(n/d - n/k)\tau_c$. However, the order of convergence of the asynchronous method is reduced (see the next section), which could increase the number of iteration steps. If these two (contradictory) features can be balanced in a satisfactory way (by the choice of a suitable network topology, a good strategy for the indices distribution and synchronization of stop test, and an efficient iteration algorithm), then we can hope that the implemented asynchronous algorithm be more efficient that the corresponding synchronous algorithm.

4. R-order of convergence of asynchronous methods

Let $\epsilon_m^{(m)} = z_i^{(m)} - \zeta_i$ be the error of an asynchronous method of the form (3) which generates the sequences $(z_i^{(m)})$ of approximations to the roots $\zeta_1, \dots, \zeta_n$. For a wide class of iteration methods for the simultaneous determination of polynomial roots (see the book [**18**]), the following relation can be derived:

$$\epsilon_i^{(m+1)} = \alpha_i\Big(\epsilon_i^{(m)}\Big)^q \sum_{\substack{j=1\\ j\neq i}}^{n} \beta_{ij}\epsilon_j^{(m-r(j,m,h))} \quad (i = 1, ..., n), \tag{5}$$

where α_i and β_{ij} are complex constants and $q \geq 1$ is integer. Then we have the following assertion:

Theorem 1. *Suppose that a polynomial P has only simple roots $\zeta_1, \dots, \zeta_n$* and starting approximations $z_1^{(0)}, \dots, z_n^{(0)}$ are reasonably close to these roots. Further, assume that $r(j, m, h)$ is bounded for all $j = 1, \dots, n$ and all $h = 1, ..., k$. Then the asynchronous algorithm (3) for which the relations (5) are valid is locally convergent with the order of convergence at least η_A, where η_A is the only positive root of the equation

$$\eta^{r+1} - q\eta^r - 1 = 0, \quad r = \max_{j,m,h} r(j,m,h). \tag{6}$$

Proof. Under the conditions of theorem we can find the constant A and B so that $|\alpha_i| \le A$ and $|\beta_{ij}| \le B$. If at the iteration step m we define the absolute error ϵ_m by $\epsilon_m = ||\mathbf{z}^{(m)} - \zeta||_\infty$, then from (5) we obtain

$$\epsilon_{m+1} \le C\epsilon_m^q \epsilon_{m-r} \quad \text{with } C = (n-1)AB. \tag{7}$$

In regard to the assumed closeness of root approximations we can adopt that $\epsilon_0 < 1$. Then from (7) it follows that the sequence (ϵ_m) tends to zero.

Let $\epsilon_0 = O(E)$, where $0 < E < 1$ and let the order of convergence of the sequence (ϵ_m) be η_A, that is, $\epsilon_{m+1} = O\big(\epsilon_m^{\eta_A}\big)$. Then

$$\epsilon_m = \big(E^{\eta_A^m}\big), \quad \epsilon_{m-r} = O\big(E^{\eta_A^{m-r}}\big), \quad (r = 0, 1, \dots, m).$$

From (7) we obtain

$$\epsilon_{m+1} = O\Big(\epsilon_m^q \cdot \epsilon_{m-r}\Big) = O\Big(E^{q\eta_A^m + \eta_A^{m-r}}\Big).$$

According to the last relation and the fact that $\epsilon_{m+1} = O\big(E^{\eta_A^{m+1}}\big)$, by the comparision of the exponents it follows

$$\eta_A^{r+1} = q\eta_A^r + 1.$$

Hence, $\eta_A > 0$ should satisfy the equation (6). □

Remark 1. Let $y(\eta) = \eta^{r+1} - q\eta^r - 1$. Since $y(q) = -1 < 0$ and $y(q+1) = (q+1)^r - 1 \ge 0$, and taking into account that the equation $y(\eta) = 0$ has the unique positive root, it follows that the order of convergence η_A of the asynchronous method belongs to the interval $(q, q+1]$. Particularly, if $r = 0$ for all $i = 1, ..., n$ and $m = 0, 1, ...$, which means that we have a synchronous method, one obtains the order of this method $\eta_S = q + 1$.

The lower bound of the order of convergence of asynchronous methods as the function of the delay r is given in Table 1. We observe that r has a very strong influence on the value of the convergence rate so that the main problem which has to be solved in the implementation consists of the minimization of the delay r.

	r				
q	0	1	2	3	4
1	2	1.618	1.466	1.380	1.325
2	3	2.414	2.206	2.107	2.056
3	4	3.303	3.104	3.036	3.012

Table 1
The lower bounds of the order of convergence of asynchronous methods in function of the delay r

5. Efficiency of implementation

In this section we will compare the implementation of an asynchronous method and corresponding synchronous method on a hypercube multicomputer. Let N_{syn} and N_{asy} be respectively the number of iteration steps of a synchronous and an asynchronous method. Evidently, the asynchronous method will be more efficient if $N_{\text{asy}}T_{\text{asy}} < N_{\text{syn}}T_{\text{syn}}$. By virtue of (2) and (4) this inequality may be written as follows:

$$N_{\text{asy}}\Big[\Big(\frac{\alpha n^2+o(n^2)}{k}\Big)\tau_a+\beta_c+O\big(\frac{n}{k}\big)\tau_c\Big] < N_{\text{syn}}\Big[\Big(\frac{\alpha n^2+o(n^2)}{k}\Big)\tau_a+D\beta_c+O\big(\frac{n}{d}\big)\tau_c\Big]. \quad (8)$$

Since the relation $\beta_c \gg \tau_c$ generally occurs ([4]) on distributed memory computers, the inequality (8) becomes

$$\frac{N_{\text{asy}}}{N_{\text{syn}}} < \frac{\dfrac{\alpha n^2}{k}+D\dfrac{\beta_c}{\tau_a}}{\dfrac{\alpha n^2}{k}+\dfrac{\beta_c}{\tau_a}}. \quad (9)$$

This inequality will be considered together with the condition

$$\frac{\beta_c}{\tau_a} < \frac{\alpha n^2}{k}. \quad (10)$$

Namely, if $\beta_c/\tau_a > \alpha n^2/k$, then it could be faster to use less processors in the synchronous implementation (see [8]).

Eventual validity of the inequality (9) can be suitable verified by a graphical interpretation in the plane $\Big(\dfrac{N_{\text{asy}}}{N_{\text{syn}}}, \dfrac{\beta_c}{\tau_a}\Big)$. A typical graph is displayed in

Fig. 1. The intersection of the curve

$$\frac{\beta_c}{\tau_a} = \frac{\dfrac{\alpha n^2}{k}\Big(1 - \dfrac{N_{\text{asy}}}{N_{\text{syn}}}\Big)}{\dfrac{N_{\text{asy}}}{N_{\text{syn}}} - D}, \tag{11}$$

and the horizontal bound line $\beta_c/\tau_a = \alpha n^2/k$, which are obtained from (9) and (10) taking the sign "=" instead of "<", gives the area where the asynchronous implementation could be faster (shaded area). Of course, this area will be feasible only if the network topology is such that the ratio β_c/τ_a (dashing line on Fig. 1) is smaller than the bound value $\alpha n^2/k$ (full horizontal line). In fact, the realistic area where the asynchronous algorithm could be faster is bounded by the curve (11) and the realistic parametric ratio $R := \beta_c/\tau_a$ (darker shaded area). This ratio usually belongs to the interval $[10^2, 10^3]$ (see [**4**]). But, these conditions are not still sufficient. Let V is the **critical ratio** which is given by the absissa of the intersection of the curve (11) and the dashing line, that is (from (9) for $\beta_c/\tau_a = R$),

$$V = \frac{\dfrac{\alpha n^2}{k} + DR}{\dfrac{\alpha n^2}{k} + R}.$$

Then the asynchronous implementation will be more efficient only if the ratio $N_{\text{asy}}/N_{\text{syn}}$ can be realized in practice, that is, if $N_{\text{asy}}/N_{\text{syn}} < V$.

We give a short analysis for a theoretical value of the ratio $N_{\text{asy}}/N_{\text{syn}}$ taking into account the accuracy of the initial errors $|z_i^{(0)} - \zeta_i|$, the required accuracy δ and the orders of convergence η_A and η_S of the asynchronous and synchronous methods respectively. Besides, we assume that (complex) roots of tested polynomials are normalized to lie in the unit disk. In that case, a stopping criterion can be given by

$$\max_{1\le i\le n} |z_i^{(m)} - \zeta_i| < \delta = 10^{-\nu},$$

where m is the iteration index and ν is the number of significant decimal digits at the approximations $z_1^{(m)}, \dots, z_n^{(m)}$. If $|z_i^{(0)} - \zeta_i| = O(10^{-1})$ and η is the order of convergence of applied simultaneous method, then the (theoretical) number of iteration steps, necessary for obtaining the accuracy δ, can be determined approximately as $m \cong \log\nu/\log\eta$ (following from $10^{-\nu} = 10^{-\eta^m}$). According to this we could expect that the ratio $N_{\text{asy}}/N_{\text{syn}}$ be approximately

$$\frac{N_{\text{asy}}}{N_{\text{syn}}} \cong \frac{\log \eta_S}{\log \eta_A}. \tag{12}$$

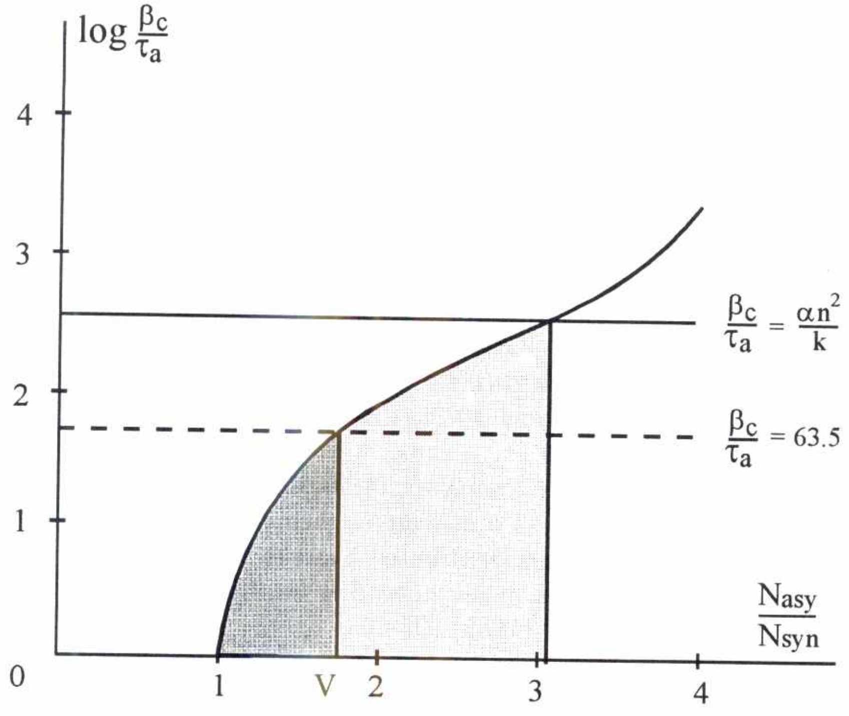


Fig. 1 Dominant area of asynchronous implementation

Finally, from a theoretical point of view, an asynchronous method will be more efficient than the corresponding synchronous method if

$$\log \eta_S / \log \eta_A < V = \frac{\frac{\alpha n^2}{k} + DR}{\frac{\alpha n^2}{k} + R} = 1 + \frac{D-1}{\frac{\alpha n^2}{kR} + 1}.$$

References

[1] G.M. Baudet, *Asynchronous iterative methods for multiprocessors.* J. of ACM **2** (1978), 226-244.

[2] M. Ben-Or, E. Freig, D. Kozen and P. Tiwari, *A fast parallel algorithm for determining all roots of a polynomial with real roots.* Proc. ACM (1989), 340-349.

[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and distributed computation - numerical methods.* Prentice-Hall Inc. 1989.

[4] L. Bomans and D. Roose, *Communication benchmarks for the iPSC/2.* Hypercube and Distributed Computers (Proc. I European Workshop on hypercube and Distributed Computers, eds. F. Andre and J. P. Verjus), North Holland, Amsterdam 1989, pp. 93-104.

[5] D. Braess and K. Hadeler, *Simultaneous inclusion of the zeros of a polynomial.* Numer. Math. **21** (1973) 161-165.

[6] M. Cosnard and P. Fraigniaud, *Asynchronous Durand-Kerner and Aberth polynomial root finding methods on a distributed memory multicomputer.* Parallel Computing **9** (1989) 79-84.

[7] M. Cosnard and P. Fraigniaud, *Finding the roots of a polynomial on an MIMD multicomputer.* Parallel Computing **15** (1990) 75-85.

[8] M. Cosnard and P. Fraigniaud, *Asynchronous polynomial root finding methods.* Research report 90-21, LIP-IMAG, Ecole Normale Supérieure de Lyon, France 1990.

[9] M. Cosnard and P. Fraigniaud, *Analysis of asynchronous polynomial root finding methods on a distributed memory multicomputer.* IEEE Transaction on Parallel and Distributed Systems (to appear).

[10] M.R. Farmer and G. Loizou, *Locating multiple zeros interactively.* Comput. Math. Appl. **11** (1985) 595-603.

[11] P. Fraigniaud, *Performance analysis of broadcasting in hypercubes.* Hypercube and Distributed Computers (Proc. I European Workshop on hypercube and Distributed Computers, eds. F. Andre and J. P. Verjus), North Holland, Amsterdam 1989, pp. 311-328.

[12] T.L. Freeman, *Calculating polynomial zeros on a local memory parallel computer.* Parallel Computing **12** (1989) 351-358.

[13] T.L. Freeman and M.K. Bane, *Asynchronous polynomial zero-finding algorithms.* Parallel Computing **17** (1991) 673-681.

[14] H. Guggenheimer, *Initial approximations in Durand-Kerner's root finding method.* BIT **26** (1986) 537-539.

[15] L.H. Jamieson and T.A. Rice, *A highly parallel algorithms for root extraction.* IEEE Trans. on Comp. **28** (1989), 443-449.

[16] J.L. Nicolas and A. Schinzel, *Localisation des zéros de polynomes intervenant end théorie du signal.* Reserach report, University of Lyon 1, 1988.

[17] V. Pan, *Sequential and parallel complexity of approximate evaluation of polynomial zeros.* Comput. Math. Appls **14** (1987), 591-622.

[18] M.S. Petković, *Iterative methods for simultaneous inclusion of polynomial zeros. Springer-Verlag, Berlin-Heidelberg-New York 1989.*

Faculty of Electronic Engineering, P.O. Box 73, 18 000 Niš

FILOMAT (Niš) 9:2 (1995), 285–294

Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

COMPUTING PSEUDOINVERSES USING MINORS OF AN ARBITRARY MATRIX

Predrag Stanimirović

Abstract. *In this paper we establish a general determinantal representation of generalized inverses in terms of minors of an arbitrary matrix of an adequate order. Then we obtain a general algorithm for exact computation of different classes of pseudoinverses: Moore-Penrose inverse, group inverse, left, right inverses and Radić's and Stojaković's inverse. In this way, this paper is a generalization of an earlier paper [12], where an algorithm for computing of the Moore–Penrose inverse, Radić's and Stojaković's inverse is described. We also give some examples which illustrate our results.*

1. Introduction

Let $\mathbb{C}_r^{m\times n}$ be the set of $m\times n$ complex matrices whose rank is r. Conjugate, transpose and conjugate-transpose matrix of A will be denoted by $\overline{A}$, A^T and A^* respectively. Submatrix and minor of A containing rows $\alpha_1,\dots,\alpha_t$ and columns $\beta_1,\dots,\beta_t$ will be denoted by $A\begin{bmatrix}\alpha_1\dots\alpha_t\\ \beta_1\dots\beta_t\end{bmatrix}$ and $A\begin{pmatrix}\alpha_1\dots\alpha_t\\ \beta_1\dots\beta_t\end{pmatrix}$ respectively, and the *algebraic complement* corresponding to the element a_{ji} is defined by

$$A_{ij}\begin{pmatrix}\alpha_1 & \dots & \alpha_{p-1} & i & \alpha_{p+1} & \dots & \alpha_t\\ \beta_1 & \dots & \beta_{q-1} & j & \beta_{q+1} & \dots & \beta_t\end{pmatrix}=(-1)^{p+q}A\begin{pmatrix}\alpha_1 & \dots & \alpha_{p-1} & \alpha_{p+1} & \dots & \alpha_t\\ \beta_1 & \dots & \beta_{q-1} & \beta_{q+1} & \dots & \beta_t\end{pmatrix}.$$

For any matrix $A\in\mathbb{C}^{m\times n}$, consider the following equations in X:

(1) $AXA=A$ (2) $XAX=X$ (3) $(AX)^*=AX$ (4) $(XA)^*=XA$

and if $m=n$, also

$$(5)\qquad AX=XA\,.$$

1991 *Mathematics Subject Classification.* 68C05, 15A09, 65F05.

For a subset $\mathcal{S}$ of $\{1,2,3,4,5\}$, the set of matrices G obeying the conditions represented in $\mathcal{S}$ will be denoted by $A\{\mathcal{S}\}$. A matrix $G \in A\{\mathcal{S}\}$ is called an $\mathcal{S}$-inverse of A and is denoted by $A^{(\mathcal{S})}$. In particular, for any $A \in \mathbb{C}^{m\times n}$, the set $A\{1,2,3,4\}$ consists of a single element, the *Moore-Penrose inverse* of A, denoted by $A^{\dagger}$ [9]. In the case $m = n$, the *group inverse*, denoted as $A^{\#}$, of A is the unique $\{1,2,5\}$ inverse, and exists if and only if $ind(A) = \min\{k:\ k > 0 \text{ and } rank(A^{k+1}) = rank(A^k)\} = 1$.

The starting point of the investigations of this paper is the determinantal representation of *Moore-Penrose inverse*, studied in [1], [2], [3], [4], [8]. The main result of these papers is:

Theorem 1.1. *Element $a_{ij}^{\dagger}$ lying on the i-row and j-column of the Moore-Penrose pseudoinverse of a given matrix $A \in \mathbb{C}_r^{m\times n}$ is given by*

$$a_{ij}^{\dagger} = \frac{\displaystyle\sum_{\substack{1\le\beta_1<\dots<\beta_r\le n\\ 1\le\alpha_1<\dots<\alpha_r\le m}} \overline{A}\begin{pmatrix}\alpha_1 & \dots & j & \dots & \alpha_r\\ \beta_1 & \dots & i & \dots & \beta_r\end{pmatrix} A_{ji}\begin{pmatrix}\alpha_1 & \dots & j & \dots & \alpha_r\\ \beta_1 & \dots & i & \dots & \beta_r\end{pmatrix}}{\displaystyle\sum_{\substack{1\le\delta_1<\dots<\delta_r\le n\\ 1\le\gamma_1<\dots<\gamma_r\le m}} \overline{A}\begin{pmatrix}\gamma_1 & \dots & \gamma_r\\ \delta_1 & \dots & \delta_r\end{pmatrix} A\begin{pmatrix}\gamma_1 & \dots & \gamma_r\\ \delta_1 & \dots & \delta_r\end{pmatrix}}, \quad \begin{pmatrix}1\le i\le n\\ 1\le j\le m\end{pmatrix}.$$

Determinantal representation of the *Group inverse* of a singular n by n matrix is introduced in [7]:

Theorem 1.2. *The group inverse $A^{\#} = \left(a_{ij}^{\#}\right)$ of $A \in \mathbb{C}_r^{n\times n}$ has the following determinantal representation:*

$$a_{ij}^{\#} = \frac{\displaystyle\sum_{\substack{1\le\alpha_1<\dots<\alpha_r\le n\\ 1\le\beta_1<\dots<\beta_r\le n}} A^T\begin{pmatrix}\alpha_1 & \dots & j & \dots & \alpha_r\\ \beta_1 & \dots & i & \dots & \beta_r\end{pmatrix} A_{ji}\begin{pmatrix}\alpha_1 & \dots & j & \dots & \alpha_r\\ \beta_1 & \dots & i & \dots & \beta_r\end{pmatrix}}{\displaystyle\sum_{\substack{1\le\gamma_1<\dots<\gamma_r\le n\\ 1\le\delta_1<\dots<\delta_r\le n}} A^T\begin{pmatrix}\gamma_1 & \dots & \gamma_r\\ \delta_1 & \dots & \delta_r\end{pmatrix} A\begin{pmatrix}\gamma_1 & \dots & \gamma_r\\ \delta_1 & \dots & \delta_r\end{pmatrix}}.$$

For the sake of completeness, in the following definition we unify the definitions of generalized inverses introduced by M. Radić [10], [11], M. Stojaković [13] and V.N. Joshi [5].

Definition 1.1. *Let i, j be integers, $1 \le i \le n$, $1 \le j \le m$. Then the (i,j)-th entry of Radić's, Stojaković's and Joshi's generalized inverse $A \in \mathbb{C}_r^{m\times n}$ is defined by*

$$a_{ij}^{\epsilon} = \frac{\displaystyle\sum_{\substack{1\le j_1<\dots<j<\dots<j_r\le n\\ 1\le i_1<\dots<i<\dots<i_r\le m}} \epsilon^{(i_1+\dots+i_r)+(j_1+\dots+j_r)} A_{ji}\begin{pmatrix}i_1 & \dots & j & \dots & i_r\\ j_1 & \dots & i & \dots & j_r\end{pmatrix}}{\displaystyle\sum_{\substack{1\le\alpha_1<\dots<\alpha_r\le m\\ 1\le\beta_1<\dots<\beta_r\le n}} \epsilon^{(\alpha_1+\dots+\alpha_r)+(\beta_1+\dots+\beta_r)} A\begin{pmatrix}\alpha_1 & \dots & \alpha_r\\ \beta_1 & \dots & \beta_r\end{pmatrix}}, \quad \epsilon \in \{-1,1\}.$$

For $\epsilon = 1$, we get Stojaković's definition, and for $\epsilon = -1$, we get Radić's definition.

Now, we describe the main results of the paper. First we define a general determinantal representation for the *Moore-Penrose, group inverse* , and the class of *left* and *right* generalized inverses. Later we describe algorithms for exact computation of generalized inverses based on the introduced determinantal representation. Finally, we give several examples which illustrate presented theory and algorithms.

2. General determinantal representation

According to Theorem 1.1, Theorem 1.2 and Definition 1.1, we define a general determinantal representation which includes the determinantal representations of the *Moore-Penrose* pseudoinverse and the *group* inverse. Also, this determinantal representation represents the class of *left* and *right* inverses for full-rank matrices and generalized inverses introduced by M. Stojaković, M. Radić and V.N. Joshi.

Theorem 2.1. *For $A \in \mathbb{C}_r^{m\times n}$ determinantal representation of an (i,j)-element of an arbitrary $left$ and right inverse, the Moore-Penrose pseudoinverse, the group inverse, Radić's and Stojaković's inverse is*

$$g_{ij} = \frac{\sum\limits_{\substack{1\le\beta_1<\ldots<\beta_t\le n\\ 1\le\alpha_1<\ldots<\alpha_t\le m}} \overline{R}\begin{pmatrix}\alpha_1 & \ldots & j & \ldots & \alpha_t\\ \beta_1 & \ldots & i & \ldots & \beta_t\end{pmatrix} A_{ji}\begin{pmatrix}\alpha_1 & \ldots & j & \ldots & \alpha_t\\ \beta_1 & \ldots & i & \ldots & \beta_t\end{pmatrix}}{\sum\limits_{\substack{1\le\delta_1<\ldots<\delta_t\le n\\ 1\le\gamma_1<\ldots<\gamma_t\le m}} \overline{R}\begin{pmatrix}\gamma_1 & \ldots & \gamma_t\\ \delta_1 & \ldots & \delta_t\end{pmatrix} A\begin{pmatrix}\gamma_1 & \ldots & \gamma_t\\ \delta_1 & \ldots & \delta_t\end{pmatrix}}, \tag{2.1}$$

where $R \in \mathbb{C}_r^{m\times n}$ and $t = r_c(A) \le r \le \min\{m,n\}$ is the greatest integer which ensures $DET_{(R,t)}(A) \neq 0$.

For the briefness sake, we denote the numerator of the expression (2.1) by $A_{ij}^{(R,t)}$ and call it the generalized algebraic complement corresponding to element a_{ij}. The denominator is shortly denoted by $DET_{(R,t)}(A)$, and it is called the generalizd determinant.

Proof. Consider the following cases:

1. Suppose that $t = m \le n$. Using the Laplace's development for the square minors $A\begin{pmatrix}1 & \ldots & m\\ j_1 & \ldots & j_m\end{pmatrix}$, we get

$$DET_{(R,m)}(A) = \sum_{j_1<\dots<j_m} \overline{R}\begin{pmatrix} 1 & \dots & m \\ j_1 & \dots & j_m \end{pmatrix} \left[\sum_{k=1}^{r} a_{ij_k} A_{ij_k} \begin{pmatrix} 1 & \dots & m \\ j_1 & \dots & j_m \end{pmatrix}\right] =$$

$$= \sum_{l=1}^{n} a_{il} \left[\sum_{j_1<\dots<j_m} \overline{R}\begin{pmatrix} 1 & \dots & \dots & \dots & m \\ j_1 & \dots & l & \dots & j_m \end{pmatrix} A_{il}\begin{pmatrix} 1 & \dots & \dots & \dots & m \\ j_1 & \dots & l & \dots & j_m \end{pmatrix}\right] = \sum_{l=1}^{n} a_{il} A_{li}^{(R,m)}.$$

For two integers $p \neq q$, $1 \leq p, q \leq m$, substituting in the minors of A the q-th row by the p-th row, and using

$$DET_{(R,m)}(A) = \sum_{j_1<\dots<j_m} \overline{R}\begin{pmatrix} 1 & \dots & m \\ j_1 & \dots & j_m \end{pmatrix} A \begin{pmatrix} 1 & \dots & m \\ j_1 & \dots & j_m \end{pmatrix} = 0,$$

The relation $\sum_{l=1}^{n} a_{pl} A_{lq}^{(R,m)} = 0$ can be proved in the same way. Hence, $g_{ij} = \delta_{ij} DET_{(R,m)}(A)$, and consequently $A \cdot A_{(R,m)}^{-1} = I_m$, for arbitrary R. It means that $A_{(R,m)}^{-1}$ represents the class of *right inverses* of the full-rank matrix A.

On the other hand, it can be proved that $A_{(R,n)}^{-1}$, in the case $t = n \leq m$, represents the class of *left inverses* of A. Now, it is obvious that (2.1) represents the general determinantal representation of *right/left inverses* of a full rank matrix A.

2. For $R = A$, we obtain determinantal representation of $A^{\dagger}$, presented in Theorem 1.1. In this case, $r_c(A) = r$, which represents the known result in [4].

3. If $m = n$, $ind(A) = 1$ and $R = A^*$ the determinantal representation of the *group inverse* is obtained (Theorem 1.2).

4. If $r = r_c(A)$ and a matrix R satisfies condition

(1) $\overline{R}\begin{pmatrix} i_1 & \dots & i_r \\ j_1 & \dots & j_r \end{pmatrix} = K \cdot \epsilon^{(i_1+\dots+i_r)+(j_1+\dots+j_r)}$, where $K \in \mathbb{C}$, $\epsilon \in \{-1, 1\}$,
for all combinations $1 \leq i_1 < \dots < i_r \leq m;\ 1 \leq j_1 < \dots < j_r \leq n$,

then, in the case $\epsilon = 1$, the inverse $A_{(R,r)}^{-1}$ is equal to the Stojaković's inverse and reduces to the Radić's inverse in the case $\epsilon = -1$ (Definition 1.1).

5. If A is regular square matrix, then (2.1) reduces into the well known inversion of regular square matrices, for an arbitrary regular matrix R of an adequate size. □

Note that the partial cases 4. and 2. are studied in [12].

3. Algorithms

In this section we give a high-level description of the algorithms for computing generalized inverses. Theoretical base of these algorithms is contained in Theorem 2.1.

In all presented algorithms complex and rational numbers are represented by an adequate union in programming language C, called the internal form of numbers. The internal form of a given matrix A is the two-dimensional array or the binary tree of the internal forms of the elements of A. Addition, subtraction, multiplication and division of complex or rational numbers in the internal forms are denoted by $\oplus$ $\ominus$ $\otimes$ $\oslash$, respectively (so called makrooperations).

Various implementation details about the generating combinations are presented in [6].

Here presented procedures receive the following global parameters:

◇ S: the actual value of $DET_{(R,t)}(A)$.

◇ $p(1:n)$, $q(1:n)$: The sequences representing combinations of rows and columns of A respectively.

Now, we describe algorithm for computation of $DET_{(R,k)}(A)$ of a rectangular matrix $A \in \mathbb{C}_k^{n\times k}$, such that $r_c(A)=k$. In the algorithm a combination $1 \le q_1 < \ldots < q_k \le n$ of rows or columns of A is fixed.

```
procedure $D_1(n,k,x,y,lg)$
   ◇ $n, k \le n$: The number of rows and number of columns.
   ◇ $x$, $y$: The internal forms of $A$ and $R$ respectively.
   ◇ $lg$: The indikator.
begin
   Step 1: $p(1:k) \overset{\cdot}{\longleftarrow} (1:k)$ ;
   Step 2: A while cycle which terminates when all the combinations
           $1 \le p_1 < \ldots < p_k \le n$ are generaed.
      Step 2.1: Compute $det(M)$ and $det(M_1)$, using $x$ and $y$, where
```

$$M = \begin{cases} A\left[\begin{smallmatrix} p_1 & \dots & p_k \\ 1 & \dots & k \end{smallmatrix}\right], & lg=1 \\ A\left[\begin{smallmatrix} 1 & \dots & k \\ p_1 & \dots & p_k \end{smallmatrix}\right], & lg=2 \\ A\left[\begin{smallmatrix} p_1 & \dots & p_k \\ q_1 & \dots & q_k \end{smallmatrix}\right], & lg=3 \end{cases}, \quad M_1 = \begin{cases} R\left[\begin{smallmatrix} p_1 & \dots & p_k \\ 1 & \dots & k \end{smallmatrix}\right], & lg=1 \\ R\left[\begin{smallmatrix} 1 & \dots & k \\ p_1 & \dots & p_k \end{smallmatrix}\right], & lg=2 \\ R\left[\begin{smallmatrix} p_1 & \dots & p_k \\ q_1 & \dots & q_k \end{smallmatrix}\right], & lg=3. \end{cases}$$

```
      Step 2.2: $S \longleftarrow S \oplus det\left(\overline{M_1}\right) \otimes det(M)$.
      Step 2.3: Generate a new combination $1 \le p_1 < \ldots < p_k \le n$.
end $D_1$
```

In the following procedure D_2 is described the algorithm for computation of $DET_{(R,l)}(A)$, where $A \in \mathbb{C}^{m\times n}$ is a matrix, such that $l = r_c(A) < \min\{m,n\}$. The main part of this algorithm is a cycle generating all combinations $1 \le q_1 < \ldots < q_l \le n$ and calling the procedure $D_1(m,l,x,y,3)$.

```
procedure  $D_2(m,n,l,x,y)$
  ◇  $m,n$ : The number of rows and the number of columns respectively.
  ◇  $l = r_c(A) < \min\{m,n\}$.
  ◇  $x$, $y$: The internal forms of $A$ and $R$, respectively.
begin
  Step 1:  $q(1:l) \longleftarrow (1:l)$ ;
  Step 2:  An while cycle, which terminates when all of the combinations
           $1 \le q_1 < \ldots < q_l \le n$ are formed. In the cycle perform:
      Step 2.1:  $D_1(m,l,x,y,3)$;
      Step 2.2:  Generate a new combination $1 \le q_1 < \ldots < q_l \le n$.
    end $D_2$
```

Finally, the algorithms D_1 and D_2 are used in the following procedure D, which computes $DET_{(R,t)}(A)$, for $t = r_c(A)$.

```
procedure  $D(l,m,n,x,y)$
  ◇  $l = r_c(A)$: Dimensions of square submatrices of $A$ and $R$.
  ◇  $m,n$ : Dimensions of the given matrix $A$.
begin
  $S \longleftarrow 0$
  if  $l = n < m$ then      $D_1(m,n,x,y,1)$
  else if  $l = m < n$ then      $D_1(n,m,x,y,2)$
  else      $D_2(m,n,l,x,y)$
end $D$
```

In the following procedure I, we describe the algorithm for exact computation of generalized inverses.

```
procedure  $I(m,n,x,y,G)$ { Computing the generalized inverse $G$ of $A$.}
  ◇  $m,n$ : The number of rows and number of columns of $A$, respectively.
  ◇  $x$, $y$: The internal forms of $A$ and $R$, respectively.
  ◇  $G = (g_{ij})$ : The internal form of computed generalized inverse of $A$.
begin
  Step 1:  $t \longleftarrow rank(A) + 1$
           repeat
               $t \longleftarrow t - 1$;      $D(t,m,n,x,y)$
           until  $S \neq 0$
  Step 2:  $p(1:t) \longleftarrow (1:t)$ ;   $q(1:t) \longleftarrow (1:t)$ ;
Step 3:
  for $w = 1:n$ do
   for $v = 1:m$ do
     Step 3.1: $suma \longleftarrow 0$
     Step 3.2:
       A while cycle over the combinations $1 \le p_1 < \ldots < p_t \le m$
         A while cycle over the combinations $1 \le q_1 < \ldots < q_t \le n$
       In the while cycles perform step a, step b and step c.
       Step a: if $(q[k] = w)$ and $(p[l] = v)$ then
                  $\{1 \le k \le t,\ 1 \le l \le t\}$
         Step a.1: Form $\overline{R}\begin{bmatrix} p_1 & \cdots & p_t \\ q_1 & \cdots & q_t \end{bmatrix}$, $A_{vw}\begin{bmatrix} p_1 & \cdots & p_t \\ q_1 & \cdots & q_t \end{bmatrix}$, using $y$ and $x$.
         Step a.2: $suma \longleftarrow suma \oplus \overline{R}\begin{pmatrix} p_1 & \cdots & p_t \\ q_1 & \cdots & q_t \end{pmatrix} \otimes A_{vw}\begin{pmatrix} p_1 & \cdots & p_t \\ q_1 & \cdots & q_t \end{pmatrix}$
       Step b: Form a new combination $1 \le q_1 < \ldots q_t \le n$
       Step c: Form a new combination $1 \le p_1 < \ldots p_t \le m$
           Step 3.3: $g_{wv} \longleftarrow suma \oslash S$
end $I$
```

4. Numerical examples

If a matrix R runs over the set of m by n matrices, in (2.1) we get various definitions of generalized inverses.

1. If $r = r_c(A)$ and a matrix R satisfies condition (1), then $A^{-1}_{(R,r)}$ is equal to the *Stojaković's inverse*, i.e. the equivalent *Joshi's inverse*, in the case $\epsilon = 1$ and the *Radić's inverse*, in the case $\epsilon = -1$.

For example, consider the matrix $A = \begin{pmatrix} \frac{11}{2} & \frac{23}{15} & 1 \\ \frac{3}{20} & -\frac{2}{7} & \frac{234}{233} \end{pmatrix}$. Using $R = \begin{pmatrix} 2 & 0 & -2 \\ 1 & 1 & 0 \end{pmatrix}$ we get the following *Stojaković's inverse* of A:

$$A^{-1}_{(R,2)} = \begin{pmatrix} \frac{58600}{440191} & \frac{619780}{1320573} \\ \frac{139335}{880382} & \frac{366975}{440191} \\ \frac{22135}{880382} & \frac{1720705}{1320573} \end{pmatrix}.$$

Using fixed point representation for the elements in A, i.e.

$$A = \begin{pmatrix} 5.50000000000000000 & 1.53333333333333344 & 1.00000000000000000 \\ 0.14999999999999999 & -0.28571428571428569 & 1.00429184549356232 \end{pmatrix},$$

and the same matrix R we get the following *Stojaković's inverse* of A:

$$A^{-1}_{(R,2)} = \begin{pmatrix} 0.1331240302505049270 & -0.4693265726317288330 \\ 0.1582665252129189510 & 0.8336722013853078430 \\ 0.0251424949624140422 & 1.3029987740170365700 \end{pmatrix}.$$

2. Furthermore, if $R = A$ satisfies (1), then $A^{-1}_{(R,r)} = A^{\dagger}$, and both generalized inverses are identical to the *Stojaković's* or the *Radić's generalized inverse*.

Concertly, for $R = A = \begin{pmatrix} \frac{5729}{327} & \frac{5729}{327} & 0 \\ 0 & \frac{5729}{327} & \frac{5729}{327} \\ -\frac{5729}{327} & 0 & -\frac{5729}{327} \end{pmatrix}$ we get the following *Moore-Penrose inverse* of A, which is identical to the *Stojaković's inverse* of A:

$$A^{-1}_{(R,2)} = A^{\dagger} = \begin{pmatrix} \frac{2008044837}{256295929} & 0 & -\frac{2008044837}{256295929} \\ \frac{2008044837}{256295929} & \frac{2008044837}{256295929} & 0 \\ 0 & \frac{2008044837}{256295929} & \frac{2008044837}{256295929} \end{pmatrix},$$

3. If $A \in \mathbb{C}^{m\times n}_r$ and $R = A$ we get $A^{-1}_{(R,r)} = A^{\dagger}$.

For example, if we use $R = A = \begin{pmatrix} \frac{175}{23} & 0 & \frac{175}{23} \\ 0 & \frac{1}{13} & \frac{175}{23} \\ \frac{175}{46} & \frac{1}{13} & \frac{525}{46} \\ 0 & \frac{1}{13} & \frac{175}{23} \end{pmatrix}$, then is obtained

$$A^{-1}_{(R,2)} = A^{\dagger} = \begin{pmatrix} \frac{192878339}{497627891} & -\frac{201395239}{995255782} & -\frac{4258450}{497627891} & -\frac{201395239}{995255782} \\ \frac{1684865000}{497627891} & -\frac{1263648750}{497627891} & -\frac{421216250}{497627891} & -\frac{2263648750}{497627891} \\ -\frac{655721205}{497627891} & -\frac{1075979571}{995255782} & \frac{1281633260}{995255782} & -\frac{1075979571}{995255782} \end{pmatrix}.$$

4. For a square matrix A, such taht $ind(A) = 1$ and $R = A^*$ we get $A^{-1}_{(R,m)} = A^{\#}$. For example, let

$$A = \begin{pmatrix} 21.93 - 3i & 4. & \frac{275}{35917} & 9.13570 + 2950.84725i \\ 11.35 & 35.75 - 2i & 0 & 1257420 \\ 257384 & \frac{91584}{23} & 12 + 15i & \frac{213574}{5762403} \\ 159384 - 135i & 109825.23 & \frac{183294}{7359} & 0.000579 \end{pmatrix}, \quad i = \sqrt{-1}.$$

Using $R = A^*$, we get the following group inverse, approximately:

$$A^{\#} = \begin{pmatrix} -0.006 - 0.011i & -0.00003 + 0.00001i & 0.000004 + 0.000001i & 0 \\ -0.029 + 0.011i & 0.00002 + 0.00007i & -0.000004 + 0.000001i & 0.00001 \\ 167.245 - 23.231i & 0.05330 - 0.39265i & -0.0109 + 0.0004i & -0.0057 - 0.00073i \\ 0.000001 & 0.000001 & 0 & 0 \end{pmatrix}.$$

5. For $A \in \mathbb{C}^{m\times n}$ using $R = \begin{pmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} = \begin{pmatrix} \mathbb{I}_r & \mathbb{O} \\ \mathbb{O} & \mathbb{O} \end{pmatrix} \in \mathbb{C}^{m\times n}$, we

obtain

$$DET_{(R,r)}(A) = A\begin{pmatrix} 1 & \dots & r \\ 1 & \dots & r \end{pmatrix},$$

and the following *algebraic complement* of the element a_{ij}:

$$A_{ij}^{(R,r)} = \begin{cases} 0\,, & \text{for } j > r \text{ or } i > r \\ A_{ji}\begin{pmatrix} 1 & \dots & i & \dots & r \\ 1 & \dots & j & \dots & r \end{pmatrix}, & \text{for } j, i \le r\,. \end{cases}$$

Generalized inverse of A is equal to

$$A_{(R,r)}^{-1} = \frac{1}{A\begin{pmatrix} 1 & \dots & r \\ 1 & \dots & r \end{pmatrix}} \cdot \begin{pmatrix} A_{11}\begin{pmatrix} 1 & \dots & r \\ 1 & \dots & r \end{pmatrix} & \dots & A_{r1}\begin{pmatrix} 1 & \dots & r \\ 1 & \dots & r \end{pmatrix} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ A_{1r}\begin{pmatrix} 1 & \dots & r \\ 1 & \dots & r \end{pmatrix} & \dots & A_{rr}\begin{pmatrix} 1 & \dots & r \\ 1 & \dots & r \end{pmatrix} & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}.$$

Concretly, for $A = \begin{pmatrix} \frac{13}{56} & 115 & \frac{476}{13} \\ \frac{1}{3} & -372 & \frac{23}{26} \\ -3 & \frac{14}{3} & \frac{21}{17} \\ \frac{12}{13} & 1 & 0 \end{pmatrix}$ and $R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ the following *right generalized inverse* of A is obtained :

$$A_{(R,2)}^{-1} = \begin{pmatrix} \frac{10652600}{6188751} & -\frac{8558144}{80453763} & -\frac{1364947612}{26817921} & 0 \\ \frac{35980}{2062917} & -\frac{3615752}{8939307} & -\frac{7448669}{160907526} & 0 \\ \frac{76388480}{18566253} & \frac{7857808}{6188751} & -\frac{1097248}{6188751} & 0 \end{pmatrix}.$$

5. Conclusion

The memory requirements of the above presented procedures for $A \in \mathbb{C}_r^{m\times n}$ are two $r_c(A) \times r_c(A)$ matrices. The advantage of the presented algorithms is in their generality, induced by theoretical weight of Theorem 2.1. The efficiency of these algorithms is identical to the efficiency of the algorithms presented in [12].

References

[1] Arghiriade, E. Dragomir, A., *Une nouvelle definition de l'inverse generalisée d'une matrice*, Lincei - Rend. Sc. fis. mat. e nat. XXXV **35** (1963), 158–165.

[2] Ben-Israel, A., *Generalized inverses of matrices: a perspective of the work of Penrose*, Math. Proc. Camb. Phil. Soc. **100** (1986), 407–425.

[3] Gabriel, R., *Extinderea complementilor algebrici generalizati la matrici oarecare*, Studii si cercetari matematice **17 -Nr. 10** (1965), 1566–1581.

[4] Gabriel, R., *Das verallgemeinerte Inverse einer Matrix, über einem beliebigen Körper - analytish betrachtet*, J. Rewie Ansew Math. **244(V)**, (1970), 83–93.

[5] Joshi, V.N., *A determinant for rectangular matrices*, Bull. Australl. Math. Soc. **21** (1980), 137–146.

[6] Lipski, W, *Combinatorics for Programmers*, Mir, Moscow, 1988. (in Russian)

[7] Manjunatha, K.P., Bhaskara Rao and Bapat, R.B., *Generalized inverses over integral domains. II. Group inverses and Drazin inverses*, Linear Algebra Appl. **146** (1991), 31–47.

[8] Moore, E.H., *General Analysis, Part I. The Algebra of Matrices*, (compiled and edited by R.W. Barnard, The Amer. Philos. Soc., 1935.

[9] Penrose, R., *A generalized inverse for matrices*, Proc. Cambridge Philos. Soc. **51** (1955), 406–413.

[10] Radić, M., *Inverzija Pravokutnih Matrica*, Doktorska disertacija, 1964.

[11] Radić, M., *A definition of the determinant of a rectangular matrix*, Glasnik matematički **1(21)-No. 1** (1966), 17–22.

[12] Stanimirović, P., Stanković, M., *Computing pseudoinverses of rectangular matrices in terms of square submatrices*, $VIII$ Conference on Applied Mathematics, Tivat (1993), 207–216.

[13] Stojaković, M. , *Determinante nekvadratnih matrica* , Vesnik D.M.N.R.S. **1-2** (1952), 9–21.

University of Niš, Faculty of Philosophy, Departement of Mathematics, Ćirila i Metodija 2, 18000 Niš, Yugoslavia

FILOMAT (Niš) 9:2 (1995), 295–302
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ON TRANSLATING MODULA-2 PROGRAMS TO C: LOCAL PROCEDURES AND MODULES

Lehel Szarapka and Dragan Mašulović

ABSTRACT. *This paper demonstrates techniques that enable efficient translation of Modula-2 programs to C. It focuses on a key problem that appears during translation: local procedures and modules. The techniques are presented via examples. For the sake of readability, instead of C a subset of Modula-2 (called Flat Modula-2) is used as a target language.*

Introduction

Modula-2 [1] is a high-level programming language designed by Prof. Niklaus Wirth at the ETH, Zürich. Its key design goal was simple and elegant support of modular programming which is the most important step towards programming in the large.

C [2] is a low-level programming language designed to help reimplementing UNIX[1]. In spite of its consistent inconsistency and poor design, C is a wide spread programming language. Because of that, it has been recognized lately as a *platform independent assembly language* thus giving rise to a slightly different approach to compilation: translation to C as a target language. Such compilers are more portable than "classic" compilers, even those which choose a form of pseudo code as a target language.

Following the tradition of Algol-like languages, Modula-2 admits declaration of procedures and modules local to other procedures. On the other hand, C does not allow declarations of functions local to other functions. Thus local modules and procedures present a key problem that a translator to C has to take care of [3, 4]. This paper presents a set of techniques that solve the problem.

1991 *Mathematics Subject Classification.* 68N20.
Key words and phrases. translation, nested procedures, nested modules.
[1]UNIX is a trade mark of AT&T Bell Labs

For the sake of readability, instead of C a subset of Modula-2 (called *Flat Modula-2*) is used as a target language. Flat Modula-2 does not allow declarations of modules, procedures, types and constants local to other procedures. Thus, translation of Flat Modula-2 programs to C is an one-one mapping and shall not be discussed here because there are several public domain translators from restrictions of Modula-2 (similar to Flat Modula-2) to C (e.g. [4]). Some examples of translating Flat Modula-2 to C can be found in Appendix A.

The rest of the paper is organized as follows: Section 1 describes two major techniques upon which the translation process is based. Section 2 handles constant and type declarations, Section 3 handles local procedures, while Section 4 discusses local modules. Section 5 concludes the paper.

1. Two major techniques

The translation process is based on two major techniques:

(1) globalization of local entities that are not local variables and
(2) systematic renaming.

The basic idea is simple: all the local entities (except local variables) are taken out of the procedure declaration and are declared globally. This is the only way to take care of local procedures and modules. At the same time, globalization gives an elegant solution to local type and constant declarations. In order to prevent name clashes, a systematic renaming is performed. Because names are of no relevance to the compiler, a brute force approach can be employed. We would like to stress that the resulting code can be compiled as efficiently as the original code.

Naturally, several well known techniques are used to support the basic ideas: symbol table, extension of procedure signatures, dependency analysis ... Instead of a formal treatment, the fundamental ideas shall be presented through examples.

2. Constant And Type Declarations

Constants and types declared in a procedure are taken out of the procedure and are declared as global entities. For example, see Figure 1 (the identifier X' is a renamed identifier X). This is an example that gives also a motivation for the approach. The translation process makes procedure `Q` a global procedure. Thus, both constant C_1 and type T_1 (modulo renaming) have to be declared as global entities.

3. Local Procedures

Let us recall that all local procedures are taken out of the procedure

```
(* Modula-2 code *)
MODULE M;
  CONST C0 = ...;
  TYPE  T0 = ...;

  PROCEDURE P(...);
    CONST C1 = "A";
    TYPE  T1 = ...;
    VAR   x : T1;
    PROCEDURE Q(a : T1;
                n : CHAR);
    BEGIN
    ...
    END Q;
  BEGIN
    Q(x, C1);
  END P;
  ...
END M.
```

→

```
(* Flat Modula-2 code *)
MODULE M;
  CONST C0' = ...;
  TYPE  T0' = ...;
  CONST C1' = "A";
  TYPE  T1' = ...;

  PROCEDURE Q(a : T1'; n: CHAR);
  BEGIN
  ...
  END Q;

  PROCEDURE P(...);
  VAR   x : T1';
  BEGIN
    Q(x, C1');
  END P;
  ...
END M.
```

Fig. 1: Constant and type declarations

```
(* Modula-2 code *)
MODULE M;
  VAR x : ...;

  PROCEDURE P;
  VAR y : ...;
    PROCEDURE Q(z:...);
    BEGIN
      y := ...;
      x := ...;
    END Q;
  BEGIN
    Q(7)
  END P;
  ...
END M.
```

→

```
(* Flat Modula-2 code *)
MODULE M;
  VAR x : ...;

  PROCEDURE Q(z:...; VAR y : ...);
  BEGIN
    y := ...;
    x := ...;
  END Q;

  PROCEDURE P;
  VAR y : ...;
  BEGIN
    Q(7, y)
  END P;
  ...
END M.
```

Fig. 2: Side effects

```
(* Modula-2 code *)
MODULE M;
  VAR x : ...;

  PROCEDURE P;
  VAR y : ...;

    PROCEDURE Q1(z : ...);
    BEGIN
      y := ...
    END Q1;

    PROCEDURE Q2(u : ...);
    BEGIN
      Q1(10)
    END Q2;

  BEGIN
    Q1(6);
    Q2(7)
  END P;
  ...
END M.
```

→

```
(* Flat Modula-2 code *)
MODULE M;
  VAR x : ...;

  PROCEDURE Q1(z : ...;
               VAR y : ...);
  BEGIN
    y := ...
  END Q1;

  PROCEDURE Q2(u : ...;
               VAR y : ...);
  BEGIN
    Q1(10, y)
  END Q2;

  PROCEDURE P;
  VAR y : ...;
  BEGIN
    Q1(6, y);
    Q2(7, y)
  END P;
  ...
END M.
```

Fig. 3: More side effects

they are declared in and are made global entities. This raises a couple of problems:

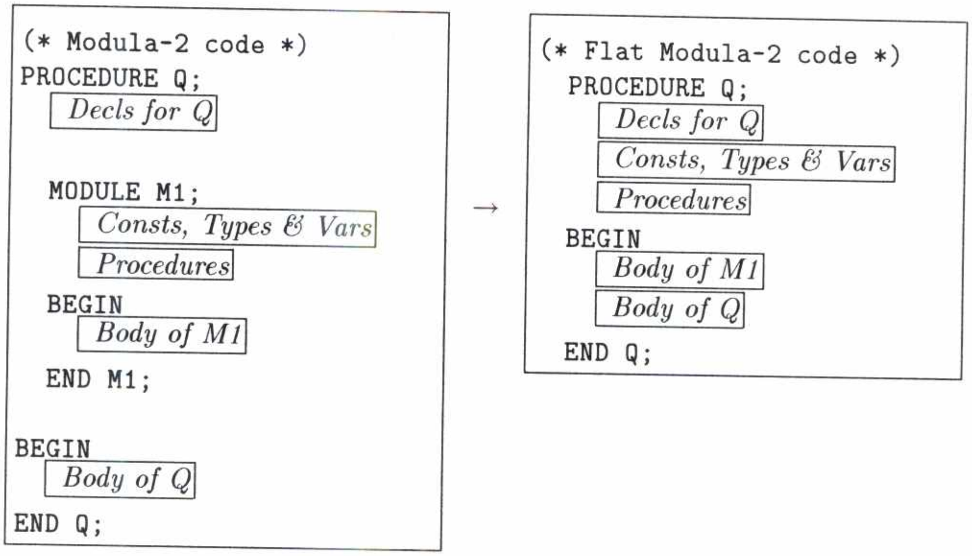
(1) (mutually) recursive procedures and
(2) side effects.

3.1. (Mutually) Recursive Procedures.

Recursive procedures are handled easily, because C supports recursive procedure calls. Mutually recursive procedures are detected using standard dependency analysis algorithm and are translated to a sequence of C procedures preceeded by a set of prototypes.

3.2. Side Effects.

Local variables, of course, remain local. The problem arises when nested procedure uses local variable whose nesting level is less then or equal to



```
(* Modula-2 code *)
PROCEDURE Q;
   Decls for Q

  MODULE M1;
     Consts, Types & Vars
     Procedures
  BEGIN
     Body of M1
  END M1;

BEGIN
   Body of Q
END Q;
```

→

```
(* Flat Modula-2 code *)
  PROCEDURE Q;
     Decls for Q
     Consts, Types & Vars
     Procedures
  BEGIN
     Body of M1
     Body of Q
  END Q;
```

Fig. 4: Local modules

the nesting level of the procedure itself (this situation is known as a side effect). After globalization of a nested procedure, local variable declared in the surrounding procedure is no longer available to the globalized procedure. Consider an example given in Figure 2. Procedure `Q` changes the variable `y`. After globalization, procedure `Q` does not have access to variable `y`.

The best solution is to extend the signature of (previously) nested procedure and to pass the variables as `VAR` parameters. This change is recorded in symbol table in order to extend the signature in procedure calls as well. In our example this means that after globalization another formal parameter has to be introduced to procedure `Q`.

Unfortunately, the problem is not as simple as it has just been presented. There are situations in which a local procedure does not have side effects, but it depends upon other local procedures which do have side effects. Signatures of such local procedures have to be extended, too, in order to obtain correct translation. These situations are easily discovered (an unavoidable dependency analysis does the job), and are recorded in symbol table. As an example, consider the module and its translation given in Figure 3: although procedure `Q2` does not change variable `y`, it calls procedure `Q1` that does change the variable. This is why the signature of procedure `Q1` has to be extended, too.

4. Local Modules

Local modules serve only one purpose: to regulate the visibility and accessibility of identifiers. Systematic renaming and symbol table book-keeping during the translation process can take care of these tasks. Therefore, the translation of local modules is straightforward: the module bounds are broken, the identifiers renamed (having in mind the `IMPORT`/`EXPORT` lists) and the declarations are included in the surrounding environment. The body of the local module is moved to the beginning of the body of the surrounding entity (another module or procedure). Thus, the semantics of the initialization part of the module is preserved, as well as the initialization order. After all the local modules are removed, previous procedures can be applied to flatten the code. All these ideas are demonstrated in the example in Figure 4. It shows a procedure and its translation.

5. Conclusion

The paper has presented basic ideas upon which a translator of Modula-2 programs to C can be based. It has payed attention to translation of procedures and modules local to other procedures and modules, because other Modula-2 language constructs are easily translated to equivalent C constructs. Systematic renaming and globalization of local entities have appeared as key techniques in the process of translation.

The translation process requires two passes. In the first pass the symbol table has to be constructed and all the dependency analyses performed. The first pass can also break local modules and take care of renaming. After the first pass has been completed, the code can be generated in the second pass. Since all the checkings and analyses have been performed in the first pass, the second pass can be carried out very quickly.

Appendix A: Translating Flat Modula-2 to C

In this appendix some Flat Modula-2 programs are translated to a C equivalent just to give the reader a raw idea how the task can be performed.

```
(* Flat Modula-2 code *)
MODULE M;
  CONST $C_0'$ = ...;
  TYPE  $T_0'$ = ...;

  PROCEDURE P(...);
  BEGIN
    $\boxed{B_1}$
  END P;
  ...
END M.
```

→

```
/* C code */
#define $C_0'$      ...
typedef ... $T_0'$;

void P(...) {
      $\boxed{B_1^C}$
}

int main() { ... }
```

```
(* Flat Modula-2 code *)
MODULE M;
  VAR x : ...;

  PROCEDURE Q1(z:...; VAR y:...);
  BEGIN
    y := ...
  END Q1;

  PROCEDURE Q2(u:...; VAR y:...);
  BEGIN
    Q1(10, y)
  END Q2;

  PROCEDURE P;
  VAR y : ...;
  BEGIN
    Q1(6, y);
    Q2(7, y)
  END P;
  ...
END M.
```

→

```
/* C code */
... x;

void Q1(... z; ... *y) {
  *y = ...;
}

void Q2(... u; ... *y) {
  Q1(10, y)
}

void P(void) {
  ... y;

  Q1(6, &y);
  Q2(7, &y);
}

int main() { ... }
```

References

[1] N. Wirth, *Programming in Modula-2*, 4th Ed., Springer-Verlag, Berlin, 1988.

[2] B. W. Kernighan and D. M. Ritchie, *The C Programming language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[3] N. Sundaresan, *Translation of Nested Pascal Routines to C*, SIGPLAN Notices, May 1990, pp. 69–81.

[4] M. Martin, *Entwurf und Implementierung eines Übersetzers von Modula-2 nach C*, Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, 1990.

Institute of Mathematics, University of Novi Sad, Trg Dositeja Obradovića 4, Novi Sad, Yugoslavia
E-mail address: ilehel@unsim.ns.ac.yu

Institute of Mathematics, University of Novi Sad, Trg Dositeja Obradovića 4, Novi Sad, Yugoslavia
E-mail address: masul@unsim.ns.ac.yu

FILOMAT (Niš) 9:2 (1995), 303–313
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

DETERMINING MODULE DEPENDENCIES IN MODULAR PROGRAMS

Lehel Szarapka and Zoran Budimac

ABSTRACT. *A short and precise algorithm for determining a module initialization order in modular programming languages is described. This algorithm is compared with a classical technique of dependency analysis of module names. It is also shown how an algorithm for determining a module compilation order is drawn from a given algorithm.*

1. Introduction

Modular programming languages enable division of a program into a set of *modules* that limit the scope of their identifiers. In this way is the design and maintenance of large programs easier, especially in team projects.

An identifier from module A is visible in module B if it is *exported* from module A and *imported* in module B. Exporting and importing of identifiers is achieved by specialized programming language constructs. Among the most popular modular languages (Ada, Modula-2, Modula-3, ...) the definition of (every) module M consists of (at least) two parts:

(1) an *interface* (or *definition module*, *package specification*, ...) of the module M, which lists all identifiers which module M exports,
(2) an *implementation* (or simply *module*, *package*, ...) of the module M, which implements (i.e., defines exported identifiers) the interface of module M.

In the rest of the paper we shall assume that a main (i.e., program) module contains a "dummy" interface. In this way *all* modules have interface and implementation parts. An implementation of a module can have (possibly

1991 *Mathematics Subject Classification.* 68N20.

The second author is supported by Science Fund of Serbia, Grant #0403 through Mathematical Institute, Belgrade.

empty) initialization part: the sequence of statements to be executed before the main program starts its execution.

Both interface and its implementation can import identifiers from other modules, via the special language constructs (import lists). The scope of the imported identifiers is only the module that has imported them.

Implementations of truly modular programming languages (Ada, Modula-2, Modula-3, Oberon, Oberon-2) should keep a complete "bookkeeping" of their modules to correctly maintain a module *compilation order* and *initialization order.* Implementations of other programming languages that only enable independent compilation (C, C++) are usually supported by separate utilities (for example *make*) to do the same task.

In this paper the algorithms for both activities are presented. It is shown how the algorithm for determining compilation order can be successfully drawn from an algorithm for determining initialization order, thus merging two activities into only one. The main contribution of the paper however is a construction of a small and efficient algorithm that can be included into a compiler of a modular language, which *precisely* determine the module initialization order. This is especially important in the presence of circular dependencies among modules, where many programming languages allow uncertainty.

The rest of the paper is organized as follows. The second section emphasizes the importance of the module compilation and initialization order and different approaches to its determination. The third section describes dependency analysis - a "classical" technique for determination of dependency order. The fourth section introduces the new algorithm, while the fifth section compares two approaches. The sixth section extends the algorithm for initialization order to an algorithm for compilation order. The last section concludes the paper.

2. Definitions and previous work

2.1 Initialization order.

According to definitions of all modular programming languages, an initialization part of every implementation of a module M is to be executed:

(1) exactly once,
(2) after initialization parts of all modules (in arbitrary order) which module M imports, and
(3) before execution of the main program.

Definition 1. Initialization order *is the order in which all modules constituting the program are initialized, such that the above three conditions are fulfilled.*

Example 1. Let module A import modules B, C and D, and modules B, C and D import nothing (which means, that they are independent of other modules.) The initialization order is the following: (B, C, D), A, where the order of B, C and D is arbitrary.

In languages where mutual imports (i.e., circular dependency) of module implementations is allowed, the initialization order is undefined (see for example [7] and [3] for Modula-2 and Modula-3 respectively.)

Example 2. Let module A import modules B and D, module B import module C, module C import module E and module D is independent. The structure of the modules is the same as in Figure 1 except that modules C and D are not connected. The possible initialization orders are the following: E, C, B, D, A, or E, C, D, B, A, or E, D, C, B, A, or D, E, C, B, A. Note that module E is always initialized before module C, and module C is always initialized before module B. Module D must be initialized before module A.

In real programming projects, circular dependencies among module implementations are not rare. If in such cases the initialization order is not defined, the programmer alone must take care of the correct and explicit initialization, to produce reliable and portable code (which is not in the "spirit" of modular programming languages.)

Example 3. Let module A import module B, module B import module C and module C import module B (circular dependency among modules B and C.) Initialization order is the following: (B, C), A, where initialization order of modules B and C is *not defined.* Note the difference between this example and Example 1, where initialization order of modules B, C and D was *arbitrary* (and always correct.) In this example the order chosen by the target compiler might be "incorrect", i.e. different from the programmer's intentions.

2.2. Compilation order.

According to definitions of all modular programming languages, an *interface* of module M is to be compiled:

(1) before compilation of implementation(s) of module M and
(2) after compilation of all interfaces that the interface of M imports.

Similarly, an *implementation* of module M is to be compiled:

(1) after compilation of interface(s) of module M and
(2) after compilation of all interfaces that the implementation of M imports.

Definition 2. Compilation order *is the order in which all modules constituting the program are compiled, such that the above four (two + two) conditions are fulfilled.*

Example 4. Let module A import modules B and D, module B import module C, module C import module D and module D is independent. The compilation order is the following: D, C, B, A.

In order to make a compilation a deterministic process, circular dependencies among interfaces are *not* allowed. Note that circular dependencies among module implementations are allowed.

Example 5. Let module A import module B and module B import module A. In this case the compilation order can not be established, because it is not clear which module should be compiled first.

2.3. Previous work.
Most of the research and publicly available compilers of modular programming languages:

(1) rely on external tools to establish compilation order and
(2) separately deal with compilation order and initialization order.

For example, **MOCKA** Modula-2 compiler [4] provides a separate utility which maintains the dependency graph to establish correct compilation order of modules. Modula-2* compiler [6] uses also dependency graph to establish compilation order, but later relies on Unix *make* utility to maintain it. Modula-2 implementation described in [5] leaves the responsibility of the compilation order to the programmer.

Almost all publicly available compilers implement initialization parts of modules as separate procedures which are called in order of their appearance in import lists. To avoid multiple calls, an indication variable for every module is set to `TRUE`, when initialization procedure is called.

Algorithms and strategies for determining compilation and initialization order are not publicly available in commercial implementations of modular programming languages.

In the next section we describe in more detail dependency analysis, as a tool for establishing correct dependencies. Up to the sixth section, we concentrate only on algorithm for determining initialization order.

3. Dependency analysis

Dependency analysis is a classical technique applied when exact dependency between some entities is to be determined. As stated in [2], it consists of the following three steps:

(1) construct a directed graph (dependency graph), such that a node a

is connected to a node b if and only if the entity a in a real world domain depends on entity b,

(2) find all strongly connected components of the dependency graph,

(3) sort strongly connected components of the dependency graph into dependency order. This is usually achieved by coalescing all components into single nodes and by sorting them topologically.

The graph transformed in the described manner shows dependencies between entities represented as graph nodes, where all nodes coalesced into one node are mutually (i.e., circularly) dependent. More details about the construction of dependency order can be found for example in [1] p. 221.

The graph for determining module dependencies consists of module names (as nodes of the graph) and links between them. More formally, the initial graph (step 1 of the above algorithm) is constructed in the following way:

(1) Construct a graph which consists of isolated nodes $M_i, i = 1, ..., n$, where $M_i, i = 1, ..., n$ are module names,

(2) Connect M_i to M_j if and only if module M_i imports module M_j.

Circularly dependent module names can be initialized in any order (which is in accordance with definitions of most modular languages.)

Dependency analysis is time and space consuming, no matter how efficiently graphs are represented (see the fifth section for details.) Dependency analysis is therefore not very suitable in direct inclusion in compilers. In the next section we proceed with a description of a more efficient solution.

4. Another solution

Basic design decisions of the new solution are:

(1) In the absence of circular dependencies, dependency analysis is equivalent to (much more efficient) depth-first search of a graph.

(2) Therefore, circular dependencies have to be resolved in a deterministic way, if possible.

The most natural way to resolve circular dependencies in a deterministic way is to establish a precise initialization order. A natural solution is to initialize modules in the order in which they appear in import lists. However, the rule that initialization of a module must be executed *after* initialization of all imported modules, must be obeyed. For example, if main module M imports module A, module A imports module B and module B imports module A, then the initialization of module B is to be executed prior to initialization of module A.

Once accepting this principle, an algorithm is simple and straightforward. The Modula-2 (pseudo-)procedure `Analyze1` implements the proposed algorithm. The following variables and procedures are used:

(1) `initialization` - a list of module names in the initialization order,
(2) `InsertBefore(M,Module,List)` - a procedure which inserts module name `M` before the module name `Module` in a list `List`,
(3) `Member(M,List)` - a function procedure which returns logical truth value `TRUE` if name `M` belongs to a list `List`,
(4) `IntfOf(M)` - a function procedure which returns the name(s) of the interface(s) of module M, and
(5) `ImplOf(M)` - a function procedure which returns the name(s) of the implementation(s) of module `M`.

We shall assume that `IntfOf (Impl (module))` is undefined, i.e. returns a null value, and that `Impl ( Impl (module)) = Impl (module)`.

Prior to calling `Analyze1`, the list `initialization` contains only the name of a (main) module implementation. The procedure is as follows.

```
PROCEDURE Analyze1(module: ARRAY OF CHAR);
  FOR every import list of module DO
    FOR every module name M in import list DO
      IF NOT Member(ImplOf(M), initialization) THEN
         InsertBefore(ImplOf(M), module, initialization);
         Analyze1(ImplOf(M))
      END
    END
  END
```

After procedure `Analyze1` returns, the list `initialization` contains the list of module names in their initialization order. Note that initialization order depends only on module implementations, and not on module interfaces.

5. Comparison of two algorithms

Our algorithm gives the same initialization order as dependency analysis, if circular dependencies are not present among modules. However, when circular dependencies are present, our algorithm sometimes gives a different initialization order than dependency analysis.

In the example of module dependencies displayed in Figure 1, A imports B and D (in that order), B imports C, C imports D and E (in that order), and D imports C. Dependency analysis gives the following initialization order: E, (C, D), B, A, where the order of C and D is arbitrary. Our algorithm however, gives the slightly different, but deterministic initialization order as follows: D, E, C, B, A. If the order of imported modules of module C were

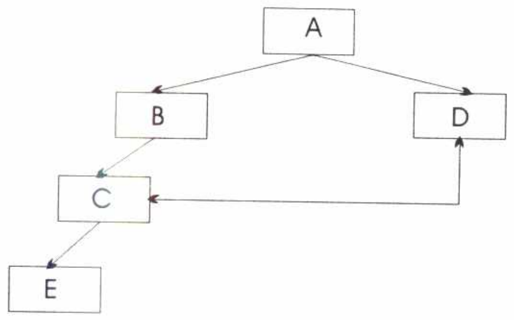


Figure 1. An example of module dependencies

changed into E and D (instead of D and E, as displayed), our algorithm would give the following order: E, D, C, B, A, which is same as the result of dependency analysis, but is deterministic. If A would import D before B, the initialization order would be E, C, D, B, A, no matter what C imports first, which is again the same as a result of dependency analysis.

Dependency analysis has a computational complexity of $O(n^2)$, where n is a total number of modules. Our algorithm has complexity proportional to hn, where h is the deepest possible nesting level of modules constituting the program. Since in most cases $h \ll n$, the complexity of our algorithm is $O(n)$. Only in degenarate cases (when $h = n$), the complexity of our algorithm is equal to the complexity of dependency analysis.

However, in reality (i.e., when n is finite) performances of our algorithm are much better than those of dependency analisys (and better than computational complexity shows.) In the following table some characteristics of both approaches are compared. Compile-time sizes of implementations of both approaches are given in the number of lines of source (Modula-2) code, while the run-time size is given in bytes. Graphs in dependency analysis are implemented as adjacency matrices of static size, but appropriate dynamic implementation would not be much smaller.

Feature	Dependency Analysis	Our Algorithm
Speed (29 modules)	4.40 sec	0.72 sec
Speed (21 modules)	3.41 sec	0.55 sec
Code size (Compile-time)	555 lines	85 lines
Code size (Run-time)	4450 bytes	800 bytes
Data size (Run-time)	$340n$ bytes	$n+16h$ bytes

6. Compilation order

The algorithm for determining compilation order can be easily obtained by the appropriate extension of the algorithm for determining initialization order. Since circular dependencies are not allowed in interfaces of modules, our algorithm will always give the same results as dependency analysis. Let us recall that for establishing initialization order only module implementations are taken into account. In order to produce an algorithm for determining compilation order, however,

(1) module interfaces have to be taken into account as well, and
(2) a list of visited module names has to be maintained to report any violation of circular dependency restriction.

Besides the already introduced variables and procedures, the following new variable and procedures are needed for the implementation of a new algorithm:

(1) `visited` - a list of visited interface names which are imported from interfaces,
(2) `InsertFront(M, List)` - inserts module name `M` at the front of a list `List`,
(3) `RemoveFront(List)` - removes a module name from the front of a list `List` (the above two operations are analogous to the `Push` and `Pop` operations on stacks), and
(4) `MakeEmptyList()` - returns an empty list.

If the procedure `Analyze` is to be called to determine *compilation* order, the list `initialization` has to contain the module name to be compiled. If the procedure `Analyze` is to be called to determine *initialization* order, the list `initialization` has to contain `Impl(module)`. The parameter `check` is set to `TRUE` if an interface is to be analyzed.

At the beginning the list `visited` is always set to an empty list and is made local to the module. This is important because of the detection of the circural dependencies. When we analyze an implementation module a new list will be created, and a new compilation order check will start. The algorithm for determining initialization order is not affected.

Because of features of procedures `Intf` and `Impl`, the `Member (Intf (module))` (seventh line of the following procedure) returns `TRUE` if a module is an implementation module (because a null value is a member of every list.) Similarly, `FOR` loop (18th line of the following procedure) will execute only once (for implementation part only).

```
PROCEDURE Analyze(module: ARRAY OF CHAR; check: BOOLEAN;
                  visited: List);
   IF check THEN
      InsertFront(module, Visited)
   END;
   IF module is an implementation module THEN
      IF NOT Member(Intf(module), initialization) THEN
         InsertBefore(Intf(module), module, initialization);
         IF M1 = Intf(M) THEN
            Analyze (M1, TRUE, visited)
         ELSE
            Analyze (M1, FALSE, MakeEmptyList());
         END
      END
   END;
   FOR every import list of module DO
      FOR every module name M in import list DO
         FOR M1 := Intf(M) TO ImplOf(M) DO
            IF NOT Member(M1, initialized) THEN
               InsertBefore(M1, module, initialization);
               Analyze(M1, M1 = Intf(M));
            ELSE
               IF check AND Member(M1, visited) THEN
                  report circular dependency;
               END
            END
         END
      END
   END;
   IF check THEN
      RemoveFront(visited)
   END;
```

7. Conclusion

An algorithm for establishing initialization and compilation order of modular programming languages is proposed. Its main contributions are:

(1) it establishes both compilation and initialization order;
(2) it is smaller and more efficient than "classical" dependency analysis, and thus can be included directly into a compiler;
(3) it improves the definition of modular programming languages by introducing deterministic initialization order in case of circular dependencies.

The third improvement of our algorithm over dependency analysis comes with a cost of producing (in some cases) different initialization order than dependency analysis would. However, according to [8], the emerging ISO Modula-2 standard (for example) gives also a clear advantage to the precise definition of module initialization order than to a classical (and sometimes vague) dependency analysis.

The presented algorithm can be applied without changes to any modular programming language regardless of how many interfaces and implementations of a module M are allowed. That includes languages like: Ada, Modula-2, and Modula-3. In languages like Oberon and Oberon-2, where circular dependencies are forbidden in implementation modules as well, a slight modification is required.

The proposed algorithm is included in a Modula-2 compiler, which is currently under development at the Institute of Mathematics in Novi Sad.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*, Addison Wesley, London, 1985.

[2] Z. Budimac, L. Szarapka, Z. Putnik, and M. Ivanović, *Dependency Analysis in a Compiler of a Functional Language*, Bull. Appl. Math. **1047/94 (LXXIV)** (1994), 43–50.

[3] L. Cardelli, J. Donahue, L. Glassman, M. Jordan, B. Kalsow, and G. Nelson, *Modula-3 Report (revised)*, SRC of Digital Equipment Corp. and Olivetti Research Center, Palo Alto and Menlo Park, 1989.

[4] H. Emmelmann and J. Vollmer, *GMD Modula System MOCKA - User Manual*, University of Karlsruhe, Technical Report, Karlsruhe, Germany, 1994.

[5] L. B. Geissmann, *Separate Compilation in Modula-2 and the Structure of the Modula-2 Compiler on the Personal Computer Lilith*, Ph.D. thesis no. 7286 ETH Zürich, Switzerland, 1983.

[6] S. U. Hänssgen, E. A. Heinz, P. Lukowitz, M. Philippsen, and W. F. Tichy, *The Modula-2* Environment for Parallel Programming*, Proc. of the Working Conf.

on Programming Models for Massively Parallel Computers, 1993, Berlin, Germany, (to appear).

[7] N. Wirth, *Programming in Modula-2*, fourth edition, Springer Verlag, Berlin, 1988.

[8] M. Woodman, *A Taste of Modula-2 Standard*, SIGPLAN Notices **28 (9)** (1993), 15–24.

University of Novi Sad, Faculty of Science, Institute of Mathematics, Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia
E-mail address: `ilehel@unsim.ns.ac.yu`

University of Novi Sad, Faculty of Science, Institute of Mathematics, Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia
E-mail address: `zjb@uns.ns.ac.yu`, `zjb@unsim.ns.ac.yu`

FILOMAT (Niš) 9:2 (1995), 315–324
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

USAGE OF S-EXPRESSIONS AND PREDICATE EXPRESSIONS IN PROCEDURAL PROGRAMMING LANGUAGES

Tatjana Vukelić and Mirjana Ivanović*

ABSTRACT. An extension of a procedural programming language with S-expressions and predicate expressions is described. Several examples in the field of graph theory, logic and set theory, hash tables, and sparse matrices are presented.

1. Introduction

Procedural programming languages are still the most frequently used programming languages. However, during the last decade many other programming paradigms (functional, logical, relational, etc.) came into the wide usage. Various programming languages and programming styles enable more natural and "simpler" solving of various classes of problems.

Great variety of programming styles lead to development of new programming languages and extensions of existing programming languages. Procedural programming languages are a good base that can be easily extended with new concepts and elements.

To enhance expressiveness of programming language Modula-2 [4], S-expressions [1] and predicate expressions (some forms of predicate formulas) [3] are included into the language. Modula-2 is widely used procedural programming language. It has variety of data types and data structures, supports structured and modular programming style and forces a programmer to write clear and readable code. With proposed extensions, Modula-2 programs are even more readable, shorter and simpler than their equivalents written in "real" Modula-2.

1991 *Mathematics Subject Classification*. 68N15.
*Supported by Grant 0401A of RFNS through Math. Inst. SANU

Although extensions described in this paper are part of the extended Modula-2 (and are implemented by a translator of extended Modula-2 programs into the "real" ones,) similar extensions can be achieved by abstract data type mechanism or by building suitable function libraries (in Modula-2 and other procedural languages.)

In the rest of the paper we shall shortly introduce the basic constructs of extended Modula-2 and then proceed with examples of possible applications.

2. S-expressions and predicate expressions

In this section an S-expression as built-in data type of extended Modula-2 and two new language constructs (predicate expressions and FORALL loop) are presented.

2.1. S-expressions.

S-expressions are basic data structures in some functional programming languages. Using Beckus-Naur form, S-expression is defined as follows:

```
S-exp        = atom | "(" S-exp-list ")"
S-exp-list   = S-exp | S-exp "." S-exp | S-exp S-exp-list
atom         = symbolic-atom | numeric-atom
numeric-atom = integer-atom | real-atom
```

The empty S-expression is denoted as `nil`. The following two conventions hold for S-expressions:

(1) `.nil` can be omitted (i.e., need not be written down), and
(2) `.(` and corresponding `)` can be omitted.

S-expression is a built-in data type of extended Modula-2 and is denoted as `SExp` (but it also can be implemented as an abstract data type [1].) `SExp` is supported with the set of primitive functions, predicates, arithmetic operations and input-output operations.

Examples of possible operations over S-expressions are [2] (for every S-expression e, e_1, and e_2):

(1) **Hd**(e) - returns e_1 if e is of the form: (e_1 . e_2),
(2) **Tl**(e) - returns e_2 if e is of the form: (e_1 . e_2),
(3) **Add**(e_1, e_2) - returns the sum of two numerical atoms e_1 and e_2,
(4) **Mul**(e_1, e_2) - returns the product of (numerical atoms) e_1 and e_2,
(5) e_1 :: e_2 - returns a new S-expression of the form (e_1 . e_2),
(6) e_1 ++ e_2 - appends two S-expressions giving a new one.

An empty S-expression is in extended Modula-2 denoted as `NULL` (i.e., `NULL` is a constant value of the type `SExp`). Some of the built-in functions over

S-expressions could be implemented as operators (function **Add**, for example could be implemented by "overloading" operator +). First experiences show however, that chosen set of functions and operators (as presented in this section) enables best readability of resulting programs.

As an example of programming with S-expression, we quote the implementation of procedure `ListOfPair(e1, e2: SExp): SExp` which returns the following S-expression: `( (e1 e2) )`, i.e. `( (e1 .(e2 .nil)) . nil)`.

```
PROCEDURE ListOfPair(e1, e2: SExp): SExp;
BEGIN
  RETURN (e1 :: (e2 :: NULL)) :: NULL
END ListOfPair;
```

2.2. Predicate expressions.

Predicate expressions are special kind of expressions [3] based on formulas of first order predicate calculus. In an extended Modula-2, they have the following form (given in extended Beckus-Naur form:)

```
PredExp     = PredSym Ident {"," PredSym Ident}
              "|" WhereFrom {"," WhereFrom}
              "WHERE" Condition {"," Condition}.
WhereFrom   = Ident "IN" Domain.
PredSym     = "EVERY" | "EXIST".
Domain      = Ident | Set | Interval | S-exp | Array.
Interval    = "[" LowerBound ".." UpperBound "]".
```

where `Condition` is a standard Modula-2 expression, whose value is a logical truth value. The value of predicate expression has a logical truth value as well. Predicate expression can also be implemented as abstract data types and supported with suitable functions, but then the corresponding programs would be less readable.

A following predicate expression:

`EVERY x | x IN X WHERE Condition`

can be read as "is it true that every `x` from `X` fulfills the `Condition`?" This expression returns `TRUE` if for all elements `x` from `X` the value of the (boolean) expression `Condition` is `TRUE`.

A following predicate expression:

`EXIST x | x IN X WHERE Condition`

can be read as "is it true that there exists at least one `x` in `X` such that `Condition` is fulfilled?" This expression returns `TRUE` if for at least one element `x` from `X` the value of the (boolean) expression `Condition` is `TRUE`.

Predicate expressions can be used with S-expressions, sets, arrays, and intervals. Arrays and intervals (i.e., subranges) are the same as in "real" Modula-2. Sets are however, more general. The elements of **Set** in extended Modula-2 [3] can be of arbitrary data types (simple or composite) and cardinality of a **Set** is not limited. All the types of set elements must be the same (like in "real" Modula-2.)

2.3. FORALL loop.

Usage of S-expressions and predicate expressions is immense in various areas and in solving of different problems. However, to make this usage simpler and more powerful, we introduced a new kind of **FOR loop** called **FORALL loop**. A new loop could be defined by the following rule of extended Beckus-Naur form:

```
ForAllLoop = "FORALL" Identifier "IN" Domain "DO"
                      Statements
             "END".
```

`Domain` in **FORALL loop** is the same as domain in predicate expression, and `Statements` are all available statements in extended Modula-2, including **FORALL**. Statement

```
                    FORALL x IN X DO Statements END
```

means that statements inside `FORALL` loop are executed for every element `x` that belongs to S-expression, set, array or interval `X`.

In the next section we proceed with some possible applications of S-expressions and predicate expressions: hash tables, graphs, sets, sparse vectors and matrices. Using S-expressions and predicate expressions, simpler and more readable programs are obtained.

3. Possible applications

3.1. Hash tables.

A hash table is one of the most popular structures for fast data retrieval. It is most often used with dictionaries. A dictionary is presented as a hash table, and consists of n ordered sets. Every set is presented as an S-expression. Hash function is a function that transforms a word into a number between 1 and n. Value of the hash function determines a set that the word belongs to.

Definition of a hash table can be (in extended Modula-2) as follows:

```
CONST  n = 211;
TYPE   HTab = ARRAY [1..n] OF SExp;
```

Procedure `Initialize` initializes elements of a hash table:

```
PROCEDURE Initialize(VAR HT: HTab);
VAR i: [1..n];
BEGIN
   FORALL i IN [1..n] DO HT[i] := NULL END
END Initialize;
```

Procedure `Found` checks if a word belongs to a dictionary:

(1) by the hash function `HashFun` the word is transformed into a hash value (number k)

(2) if the word belongs to the set that contains all words with the same hash value k, function returns `TRUE`.

In the following procedure we shall assume that the data type `String` exists and that it is implemented as a fixed-length array of characters.

```
PROCEDURE Found(Word: String; HT: HTab): BOOLEAN;
VAR x: String;
BEGIN
   RETURN EXIST x | x IN HT[HashFun(Word)] WHERE x=Word
END Found;
```

Procedure `Store` stores a word into a hash table.

```
PROCEDURE Store(Word: String; VAR HT: HTab);
VAR pos: [1..n];
BEGIN
   pos := HashFun(Word);
   HT[pos] := Word :: HT[pos]
END Store;
```

Graphs.

A graph G consists of

(1) set V, whose elements are called nodes and

(2) set of pairs E, whose elements are called edges.

Graph can be defined using adjacency lists. To every node, a list of adjacent nodes is attached. Graph can also be defined as a list of edges. An edge is represented as a pair of nodes, which it connects.

```
TYPE node  = CARDINAL;
     edge  = RECORD c1, c2 : node END;
     Graph = RECORD nodes : SET OF node;
                    edges : SET OF edge
             END;
```

Graph is *connected* if there is a path between every pair of its nodes. We shall assume that function `Path(c1, c2: node): BOOLEAN` returns the value `TRUE` if there is a path between nodes `c1` and `c2`, otherwise returns the value `FALSE`.

Function `Connected` checks if a graph is connected.

```
PROCEDURE Connected(G: Graph): BOOLEAN;
VAR  c1, c2: node;
BEGIN
   RETURN EVERY c1, EVERY c2 | c1 IN G.nodes, c2 IN G.nodes
          WHERE Path(c1,c2)
END Connected;
```

A graph is *complete* if each node is connected to every other node. Procedure `Edge(c1, c2):BOOLEAN` checks if there is an edge incident to nodes `c1` and `c2`. It assumes that if `c1 = c2`, there is an edge between those nodes.

Function `Complete` checks if a graph is complete.

```
PROCEDURE Complete(G: Graph): BOOLEAN;
BEGIN
   RETURN EVERY c1, EVERY c2 | c1 IN G.nodes, c2 IN G.nodes
          WHERE Edge(c1, c2) AND (c1 <> c2)
END Complete;
```

Degree of a node v, is equal to the number of edges that are adjacent to v. Function `Degree` determines the degree of node `v` in the graph `G`.

```
PROCEDURE Degree(v: node; G: Graph): CARDINAL;
VAR  Deg : CARDINAL; E : edge;
BEGIN
   Deg := 0;
   FORALL E IN G.edges DO
      IF (E.c1 = v) OR (E.c2 = v) THEN INC(Deg) END
   END
END Degree;
```

3.2. Sets.

Let us recall that elements of a set in extended Modula-2 can be of arbitrary type and that the number of set types is (conceptually) unlimited. For example, in the following definition:

```
TYPE  SetAnyType = SET OF AnyType
```

where **AnyType** can be of arbitrary type, including arrays, records and other sets. Procedure **SetMember** determines whether an element **x** is a member of a set **S**.

```
PROCEDURE SetMember(x: AnyType; S: SetAnyType);
VAR  e : AnyType;
BEGIN
   RETURN EXIST e | e IN S WHERE e = x
END SetMember;
```

Procedure **SubSet** checks whether set **s1** is a subset of a set **s2**.

```
PROCEDURE SubSet(s1, s2: SetAnyType);
VAR  x1, x2: AnyType;
BEGIN
   RETURN EVERY x1, EXIST x2 |
                   x1 IN s1, x2 IN s2 WHERE x1 = x2
END SubSet;
```

3.3. Sparse vectors and matrices.

A sparse vector is a vector that consists mostly of zero elements. It can be presented by S-expression whose elements are ordered pairs. Every pair presents one non-zero element of a sparse vector. The first element of the ordered pair is an index of the element in a vector, and the second is the value of the element. For example, a vector $V = [1\ 0\ 0\ 0\ 0\ 2\ 0]$ is represented by ((1 1) (6 2)).

Procedure **MulVec** returns a product of a vector **v** and scalar **n**.

```
PROCEDURE MulVec(v: SExp; n: INTEGER): SExp;
VAR  res, el: SExp;
BEGIN
   res := NULL;
   FORALL el IN v DO
      res := res ++ ListOfPair(Hd(el), Mul(Tl(el), n))
   END;
   RETURN res
END MulVec;
```

Procedure SumVec sums two vectors. In this procedure, following procedures will be used:

(1) Find(e,v) which returns a pair in the vector v, whose index is equal to a value e.

(2) Delete(e,v) which deletes a pair e from the vector v.

```
PROCEDURE SumVec(v1, v2: SExp): SExp;
VAR  res, el1, el2 : SExp;
BEGIN
  res := NULL;
  FORALL el1 IN v1 DO
    el2 := Find(Hd(el1), v2);
    IF el2 <> NIL
      res := res ++ ListOfPair(Hd(el1),
                               Add(Tl(el1), Tl(el2)));
      Delete(el2, v2)
    ELSE
      res := res ++ (el1 :: NULL)
    END
  END;
  RETURN res ++ v2;
END SumVec;
```

Procedure VecScPro returns a scalar product of two vectors. In this procedure, the procedure FindVal(n, v) is assumed to return the value of the element with an index n in the vector v.

```
PROCEDURE VecScPro(v1, v2: SExp): SExp;
VAR  res, el1, el2: SExp;
BEGIN
   res := 0;
   FORALL el1 IN v1 DO
      IF EXIST el2 | el2 IN v2 WHERE Hd(el1) = Hd(el2) THEN
         res := Add(res, Mul(Tl(el1), FindVal(Hd(el1), v2)))
```

```
        END
     END;
     RETURN res
  END VecScPro;
```

A sparse matrix is a matrix that contains relatively many zero elements. A sparse matrix can be represented by S-expression that consists of ordered pairs. By every pair a row of matrix that has at least one non-zero element is presented. The first element of the pair is the index of a row of matrix, and the second element of the pair is a sparse vector.
Matrix

$$M = \begin{bmatrix} 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 7 \end{bmatrix}$$

has the representation $M = (\ (1\ ((2\ 4))\ )(\ 3\ ((1\ 6)\ (4\ 7))\ )\ )$. We shall also assume that function procedure `Transpose(M)` returns transposed matrix of matrix `M`.

Procedure `MatVecPro` multiplies a matrix by a vector. In this procedure, procedure `VecScPro` (defined previously) is used.

```
PROCEDURE MatVecPro(V, M: SExp): SExp;
VAR res, s, TM: SExp;
    val : INTEGER;
BEGIN
   res := NULL;
   TM := Transpose(M);
   FORALL s IN TM DO
      val := VecScPro(V, S);
      IF val <> 0 THEN
         res := res ++ ListOfPair(Hd(s), val)
      END
   END;
   RETURN res
END MatVecPro;
```

The result of multiplying a sparse vector and a sparse matrix is a new sparse vector.

Procedure `MatPro`, multiplies two matrices. In this procedure, procedure `MatVecPro` is used.

```
PROCEDURE MatPro(M1, M2: SExp): SExp;
```

```
VAR  res, v, c, TM, tpro : SExp;
BEGIN
   res := NULL;
   TM := Transpose(M2);
   FORALL v IN M1 DO
      tpro := MatVecPro(v, M2);
      res := res ++ ListOfPair(Hd(v), tpro)
   END;
   RETURN res
END MatPro;
```

4. Conclusion

S-expressions and predicate expressions are included into programming language Modula-2. In a similar way they can be included into other procedural programming languages. Inside a procedural programming language, S-expressions bring elements and concepts of functional programming languages. Elements of functional programming style in procedural programming languages bring advantages of both programming styles in different areas. Programs are shorter, simpler and more readable than in pure procedural programs and more efficient than equivalent functional programs.

A usage of predicate expressions in procedural programming languages brings more concise, clearer and more powerful code. Both extensions contribute to better expressiveness of programs.

Usages mentioned in this paper present only a small part of possibilities which extensions of procedural language bring.

References

[1] Z. Budimac and M. Ivanović, *New Data Type in Pascal* (1989), Proc. of DECUS Europe Symposium, The Hague, Holland, 193-199.

[2] M. Ivanović and Z. Budimac, *Usage of S-expression in Pascal* (1989), Proc. of 11th International Symposium "Computer at the University", Cavtat, 3.18.1-3.18.6.

[3] T. Vukelić and M. Ivanović, *Predicate expressions in procedural programming languages* (to appear).

[4] N. Wirth, *Programming in Modula-2*, fourth edition, Springer Verlag, Berlin, 1988.

University of Novi Sad, Faculty of Science, Institute of Mathematics, Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia
E-mail address: {vukelic,mira}@unsim.ns.ac.yu

FILOMAT (Niš) 9:2 (1995), 325–332
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

IMPLEMENTATION OF PREDICATE EXPRESSIONS IN PROCEDURAL PROGRAMMING LANGUAGES

Tatjana Vukelić and Dušan Kamenov

ABSTRACT. *Predicate expressions in a procedural programming language are based on sentences of predicate calculus of first order. The usage of predicate expressions in procedural languages leads to shorter, more effective and more readable programming code, and also decreases number of loops and local variables in procedures and programs. Predicate expressions in programming languages could be used with array, set and interval data types. Elements of array or set could be simple or complex data type. In this paper, definition and implementation of predicate expressions in procedural programming language Modula-2 is presented. Areas of usage are logic, set theory, graph theory, pattern recognition and others.*

1. Introduction

Many statements, particularly in mathematics, are of the form "x satisfies α", where x belongs to set D and α is relation relevant to the elements of set D [1].

Statement "For every $x \in D$, $\alpha(x)$" is an example of a mathematical statement. Symbolically,

$$(\forall x \in D)\alpha(x), \quad \text{or shorter,} \quad (\forall x)\alpha(x)$$

denotes this kind of statement. The part $(\forall x)$ is called universal quantifier.

Statement "Exists $x \in D$, $\alpha(x)$" is also often used in mathematical sentences. Symbolically, this kind of sentence can be presented as

$$(\exists x \in D)\alpha(x), \quad \text{or shorter,} \quad (\exists x)\alpha(x)$$

The part $(\exists x)$ is called existentional quantifier.

1991 *Mathematics Subject Classification.* 68N15.

It is possible to use more than one quantifier in a single sentence. For example,

$$(\forall x)(\exists y)\alpha(x,y)$$

is a valid sentence of predicate calculus.

In the further text, symbols and will be referred to as predicate symbols.

These sentences can be efficiently implemented and used in procedural programming languages. The implementation shown in following chapters is an extension of Modula-2 programming language [2]. In further text, this extension will be called EM2.

Predicate expression in procedural programming languages has the form of list of quantifiers (predicate symbols followed by variables). After that, domains of the variables which follow predicate symbols in quantifiers are stated. At the end stands condition of predicate expression, which is presented by list of boolean expressions. Expressions are separated by commas.

The equivalent symbols for predicate symbols $\forall$ and $\exists$ in EM2 are `EVERY` and `EXIST`, respectively.

Syntax of predicate expressions can be presented by following rules of EBNF:

```
#PredicateExpr =  PredicateSymbol Identifier
     { "," PredicateSymbol Identifier } "|"
     Identifier "IN"  Range { "," Identifier "IN" Range }
     "WHERE" Expression { "," Expression }.
#PredicateSymbol = "EXIST" | "EVERY".
#Range = Array | Set | Interval.
#Array = Identifier.
#Set = Identifier | "{" [ Member {"," Member} ] "}".
#Interval = "[" LowerBound ".." UpperBound "]".
```

Existing set data type in M2 language can also be extended. In Modula-2, elements must be simple data type. In EM2, this restriction is no longer valid; elements can be any data type - single or complex. Elements of set are not ordered, and an element can appear in set several times. For example, $\{1,2,1,3\}$ is a valid set in EM2.

Range of the variables in quantifiers must be finite. Range of those variables is determined by standard Modula-2 data types array or interval, or by new data type set.

Example 1.1. Some simple predicate expressions are:

```
VAR
     a, b    :  BOOLEAN;
     array   :  ARRAY [1..10] OF CARDINAL;
     set     :  SET OF CHAR;
```

```
x       :  CHAR;
y       :  CARDINAL;

        .   .   .

a := EXIST x | x IN set WHERE x < 'f';
b := EVERY y | y IN array WHERE y < 10;
```

The first assignment could be read: "if exist character `x`, which belongs to set `set` and `x` is before character 'f' in the ASCII table, then `a` becomes true, else `a` becomes false". The second assignment could be read: "if for every `y`, where `y` is an element of array `array`, condition `y < 5` is satisfied, then `b` becomes true, else `b` becomes false". Simpler, if every element of array `array` is less than 5 then `b` becomes true, else `b` becomes false.

It is possible to combine more than one quantifier in one predicate expression. Let's see an example. Suppose that `set1` and `set2` are set of cardinals, `a` is type boolean, and `x` and `y` are type cardinal.

Example 1.2.

```
a := EVERY y, EXIST x | y IN set1, x IN set2  WHERE x = y;
```

Simply said, if for every `y` exist `x`, `y` belongs to `set1`, `x` belongs to `set2`, equation `x = y` is satisfied, then `a` becomes true, else `a` becomes false.

2. Implementation of predicate expressions

Predicate expressions can be implemented in procedural programming languages in many different ways. One of them, which is based on translation of predicate expressions to equivalent code in Modula-2 programming language, is presented in further text. To make the translation simpler, some constructions of Modula-2 can be extended. Therefore, following statements are defined:

(1) `FORALL x IN X` (a kind of loop)
(2) `NEXT x`

`FORALL x IN X` means that statements inside loop are executed for all elements of an interval, set or array signed by `X`.

`NEXT x` determines the next element of `X`.

Interval and array are ordered. Theoretically, the order of the elements of a set is irrelevant. But in the computer memory a set is ordered and it is possible to take it's elements one after another.

Example 2.1. Structure

```
    FORALL x IN [1..10] DO
          Write(x);
    NEXT x
```

is equivalent to

```
    x := 1;
    WHILE x <= 10 DO
          Write(x);
          x := x + 1
    END;
```

For simplicity, predicate expressions with one and two quantifiers will be discussed first. After that, we'll make a generalization of translation for any number of quantifiers.

2.1. Predicate expressions with one quantifier.

In general, predicate expression with one quantifier has the form:

PS x | x IN X WHERE *condition*

where predicate symbol is denoted by PS. Following cases are possible:

(a) EVERY x | x IN X WHERE *condition*

First, suppose that condition is satisfied for every x from X, and suppose that predicate expression has the truth value true. If x that does not satisfy the condition *condition* is found in for-all loop, then whole predicate expression gets truth value false.

```
    EV_x := TRUE;
    FORALL x IN X DO
          IF NOT condition THEN
                EV_x := FALSE
          END
    NEXT x;
    Result := EV_x
```

(b) EXIST x | x IN X WHERE *condition*

First, suppose that there is no x from X that satisfies the condition and suppose that whole predicate expression has the truth value false. If x that satisfy the condition condition is found in for-all loop, then the predicate expression gets the truth value true.

```
    EX_x := FALSE;
    FORALL x IN X DO
          IF condition THEN
```

```
                EX_x := TRUE
        END
    NEXT x
    Result := EX_x;
```

2.2. Predicate expressions with two quantifiers.

In general, predicate expression with two quantifiers has the form:

PS x, *PS* y | x IN X, y IN Y WHERE *condition*

where predicate symbol is denoted by *PS*. Following cases are possible:

(*a*) EXIST x, EXIST y | x IN X, y IN Y WHERE *condition*

In this case, construction 2.1.(b) will be used.

```
EX_x := FALSE;
FORALL x IN X DO
        EX_y := FALSE;
        FORALL y IN Y DO
                IF condition THEN
                        EX_y := TRUE
                END
        NEXT y;
        EX_x := EX_x OR EX_y
NEXT x;
Result := EX_x;
```

(*b*) EVERY x, EVERY y | x IN X, y IN Y WHERE *condition*

Construction 3.1.(a) is used in this case.

```
EV_x := TRUE;
FORALL x IN X DO
        EV_y := TRUE;
        FORALL y IN Y DO
                IF NOT condition THEN
                        EV_y := FALSE
                END
        NEXT y;
        EV_x := EV_x AND EV_y
NEXT x;
Result := EV_x;
```

(*c*) EXIST x, EVERY y | x IN X, y IN Y WHERE *condition*

Construction 3.1.(a) and 3.1.(b) are used combined in this case.

```
EX_x := FALSE;
FORALL x IN X DO
      EV_y := TRUE;
      FORALL y IN Y DO
            IF NOT condition THEN
                  EV_y := FALSE;
            END
      NEXT y;
      EX_x := EX_x OR EV_y
NEXT x
Result := EX_x;
```

(*d*) EVERY *x*, EXIST *y* | *x* IN *X*, *y* IN *Y* WHERE *condition*
Similar to case 3.2.(c),

```
EV_x := TRUE;
FORALL x IN X DO
      EX_y := FALSE;
      FORALL y IN Y DO
            IF condition THEN
                  EX_y := TRUE
            END
      NEXT y;
      EV_x := EV_x AND EX_y
NEXT x;
Result := EV_x;
```

2.3. Predicate expressions with any number of quantifiers.

Predicate expressions are analyzed from left to right. For every quantifier there is a for-all loop and one boolean variable that starts with EX_, if quantifier is EXIST, EV_, if quantifier is EVERY. Boolean variable gets its value before entering the for-all loop. Its value is FALSE in case of EXIST quantifier, and TRUE in case of EVERY quantifier.

Inside for-all loop two cases are possible:

(*1*) If quantifier is the last of the quantifiers in predicate expression, then inside for-all loop is an IF statement:

(*a*) If the quantifier is EXIST quantifier then it is following IF statement:

```
IF condition THEN
      EX_ident := TRUE
END;
```

(*b*) If the quantifier is `EVERY` quantifier then it is following `IF` statement:

```
IF NOT condition THEN
        EV_ident := FALSE
END;
```

`NEXT` *ident* follows the `IF` statement.

(*2*) If the quantifier is not the last one in predicate expression then inside the for-all loop are statements that matches quantifiers that come after current quantifier (this part could be implemented by recursion). After that follows:

(*a*) If current quantifier is `EVERY` then

```
EV_ident := EV_ident AND EV_ident2        (1) or
EV_ident := EV_ident AND EX_ident2        (2)
```

Statement (1) if the quantifier after current quantifier is of form `EVERY ident2`, statement (2) if the quantifier after current quantifier is of form `EXIST ident2`.

(*b*) If current quantifier is `EXIST` then

```
EX_ident := EX_ident OR EV_ident2        (3) or
EX_ident := EX_ident OR EX_ident2        (4)
```

Statement (3) if the quantifier after current quantifier is of form `EVERY ident2`, statement (4) if the quantifier after current quantifier is of form `EXIST ident2`.

After this statement follows `NEXT ident` statement.

3. An example of usage of predicate expressions

Predicate expressions can be used in solving different classes of problems. One of the areas of usage is mathematical logic.

Definition 3.1. Proposition $P(p, q, \dots)$ that has the truth value true for any truth values of their variables is called **tautology**.

A procedure which determines if a expression is a tautology could be as follows:

```
PROCEDURE Tautology;
VAR
      a, b:  BOOLEAN;
BEGIN
      IF EVERY a, EVERY b | a IN [FALSE..TRUE],
             b IN [FALSE..TRUE] WHERE a OR b OR NOT b THEN
                  WriteStr(" Expression is a tautology ")
      ELSE
            WriteStr(" Expression is NOT a tautology ")
      END
END Tautology.
```

The result of this program will be "Expression is a tautology" because expression is a tautology.

4. Conclusion

Predicate expressions have wide usage in many areas of computer science. Their great power is in area of mathematics. They allow short, readable and concise presentation of different definitions and theorems. They also have wide usage in pattern recognition. Combined with sets, they are powerful tool for fast and natural solving of different problems. Their usage decreases number of loops and local variables to minimum required, which makes the programming code shorter and more readable.

The future of predicate expressions can also be found in functional and logical programming languages. Predicate expressions are a step closer to human-like way of thinking in programming languages.

References

[1] Milić, Svetozar, *Elementi matematičke logike i teorije skupova*, A-Š delo, Beograd, 1991.

[2] Wirth, Niklaus, *Programiranje na jeziku Modula-2*, Dragon, Beograd, 1990.

Seljačkih buna 25, 21000 Novi Sad
E-mail address: {vukelic,ikamenov}@unsim.ns.ac.yu

FILOMAT (Niš) 9:2(1995), 333–343
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

DEPENDENCE TESTING ON LOOPS WITH BOUNDS WHICH ARE FUNCTIONS OF OUTER LOOP INDICES

Suzana Stojković

ABSTRACT. *Parallelizing compilers are compilers which translate sequential programs into parallel ones. Program loops are the most frequent sources of parallelism in sequential programs. Because of that, parallelizing compilers first must detect loops which can be run in parallel. Different iterations of the same loop can execute in parallel if they process different data. Parallel loops can be identified by detecting data dependencies across the loop body. For data dependence testing a few algorithms were developed. In this paper GCD test and Banerjee's test are presented. These algorithms are applicable when bounds of loop indices are constant. This paper shows how Banerjee's test can be exploited when the inner loop bounds are functions of outer loops indices. We, first, must compute minimums of the lower and maximums of the upper loop bounds. We solved this problem when the loop bounds are linear functions. We show that this minimums and maximums are dependent on the data dependence direction vector. We have also modified Banerjee's test, slightly.*

1. Introduction

Developments in semiconductor technology tend to reduce dimension and price of electronic components, but to grow their speed. Hardware performances become better every day. Now, supercomputers are developed.

Fast hardware development lead to a software crisis. A new problem appears: how to exploit all hardware performances. Because of that, parallel algorithms have been developed, in the last few years. Programmer who designs parallel algorithms must be familiar with hardware

architecture for which these algorithms are meant. This leads to the idea that the parallelization can be done by compilers. Now, parallelizing compilers are very popular area of computer science.

The major problem of parallelizing compilers is to detect parallelism during sequential programs. Program loops are the most frequent source of parallelism. Because of that, first problem is to detect loops which can be run in parallel. Different iterations of the same loop can execute in parallel if they process different data. The key to identification of parallel loops is to detect data dependencies across loop body.

There are three types of data dependencies exist:

(1) *Data true dependence* - exists when a variable computed in statement S_1 is used in some next statement S_2. We say that S_2 is data true dependent on S_1, and write this as $S_1\delta^t S_2$.

(2) *Data anti dependence* - exists when a variable is used in statement S_1 and it is defined in some next statement S_2. We say that S_2 is data anti dependent on S_1, and write this as $S_1\delta^a S_2$.

(3) *Data output dependence* - exists if the same variable is defined by statements S_1 and S_2. We say that S_2 is data output dependent on S_1, and write this as $S_1\delta^o S_2$.

Detection of dependence is not difficult if only scalar variables figure in the loop. Difficulties are caused by subscripted variables. For example, we will try to determine all dependencies which exist in the next loop:

$$\begin{array}{ll} L: & \text{DO } 10 \text{ I} = 5, 10 \\ S_1: & \quad \text{A}(I+3) = 2 * A(I-4) \\ S_2: & \quad \text{B}(I) = A(I) + C10 \\ & 10\ CONTINUE \end{array}$$

First dependence which can be identified is the dependence between statements S_1 and S_2. Elements of array A defined in statement S_1 are used in statement S_2. We can say that S_2 is true dependent on S_1 ($S_1\delta^t S_2$).

On the basis of the above, we can say that $S_1\delta^t S_1$. However, if we look a bounds of the loop index I, we will see that the statement S_1 defines elements A(8),...,A(13), but uses elements A(1),...,A(6), and dependence $S_1\delta^t S_1$ does not exist.

2. Data dependence testing algorithms

Below example shows that data dependence testing algorithms must analyze several more different elements, like: dependence among the statements in the loop; dependence in the appropriate region, etc. These algorithms find data dependence direction vectors [2,4], too.

Data dependence direction vector, θ, defines relations between values of loop indices for which dependence exists. The dimension of vector θ (m) is the number of loops which enclose the statements S_1 and S_2. The elements of vector are members of the set $\{<, =, >, *\}$. We will assume that certain loop include two statement S_1 and S_2. Lets us label the I-th iteration of statement S_1 as $S_1(I)$. Let also $S_1(I)\delta_\theta S_2(J)$. The appropriate element of vector θ is:

- $<$, if $I < J$;
- $>$, if $I > J$;
- $=$, if $I = J$;
- *, if relation between I an J is unknown.

For the loop L: m=1 and $S_1\delta_\theta S_2$ where is $\theta = (<)$, because the element of array A which was defined in the first iteration of statement S_1, will be used in forth iteration of statement S_2 $(1 < 4)$.

Let us consider two statements (S_1 and S_2) which are enclosed by m loops:

```
DO 10 $I_1 = T_1, U_1$
  DO 10 $I_2 = T_2, U_2$
      ⋮
    DO 10 $I_m = T_m, U_m$
      ⋮
      ... A($I$) ...
      ⋮
      ... A($J$) ...
      ⋮
10 CONTINUE
```

Indices I and J are functions defined as:

$$I = f_1(I_1, I_2, ..., I_m) = f_1(I) \tag{1}$$

$$J = f_2(I_1, I_2, ..., I_m) = f_2(J) \tag{2}$$

The dependence between S_1 and S_2 exists in those iterations in which I equals J. The goal of these algorithms is to determine whether the equation

$$f_1(I) = f_2(J) \tag{3}$$

has got integer solutions. This equation is a dependence equation.

Functions f_1 and f_2, in most cases, are linear:

$$f_1 = \sum_{k=1}^{m} a_k I_k + a_0 \tag{4}$$

$$f_2 = \sum_{k=1}^{m} b_k J_k + b_0 \tag{5}$$

In this case the dependence equation is a diophantine equation and can be stated as:

$$\sum_{k=1}^{m} a_k I_k - \sum_{k=1}^{m} b_k J_k = b_0 - a_0 \tag{6}$$

Whether the diophantine equation has a solution can be detected by GCD test. The dependence equation can be written as follows:

$$\sum_{k=1,\theta_k='='}^{m} (a_k - b_k) I_k + \sum_{k=1,\theta_k\neq'='}^{m} a_k I_k - \sum_{k=1,\Theta_k\neq'='}^{m} b_k J_k = b_0 - a_0 \tag{7}$$

Let $g = GCD(\{a_k - b_k, \theta_k ='='\}, \{a_k, \theta_k \neq'='\}, \{b_k, \theta_k \neq'='\})$.

GCD test: The dependence equation has solutions **iff** $g \mid (b_0 - a_0)$ or $a_0 = b_0$.

Note that this test does not answer the question whether the solutions exist in the given region. This question can be answered second by a group of data dependence testing algorithms - *Banerjee's test.*

This test needs to introduce a positive part of real number r (r^+), and the negative part of r (r^-), as:

$$r^+ = \begin{cases} 1 & r > 0 \\ 0 & r \leq 0 \end{cases} \tag{8}$$

$$r^- = \begin{cases} -1 & r < 0 \\ 0 & r \geq 0 \end{cases} \tag{9}$$

Banerjee's criteria: The data dependence for a given vector θ does exist if the GCD test is satisfied and the next inequality is satisfied, too:

$$\sum_{k=1}^{m} LC_k \leq b_0 - a_0 \leq \sum_{k=1}^{m} UC_k \tag{10}$$

where:

(11)
$$LC_k = \begin{cases} -(a_k^- + b_k)^+(U_k - T_k - 1) + (a_k - b_k)T_k - b_k & \text{for } \theta_k ='<' \\ -(a_k - b_k)^-(U_k - T_k) + (a_k - b_k)T_k & \text{for } \theta_k ='=' \\ -(b_k^+ - a_k)^+(U_k - T_k - 1) + (a_k - b_k)T_k + a_k & \text{for } \theta_k ='>' \\ -(a_k^- + b_k^+)(U_k - T_k) + (a_k - b_k)T_k & \text{for } \theta_k =' *' \end{cases}$$

$$UC_k = \begin{cases} (a_k^+ - b_k)^+(U_k - T_k - 1) + (a_k - b_k)T_k - b_k & \text{for } \theta_k =' <' \\ (a_k - b_k)^+(U_k - T_k) + (a_k - b_k)T_k & \text{for } \theta_k =' =' \\ (b_k^- - a_k)^+(U_k - T_k - 1) + (a_k - b_k)T_k + a_k & \text{for } \theta_k =' >' \\ (a_k^+ + b_k^-)(U_k - T_k) + (a_k - b_k)T_k & \text{for } \theta_k =' *' \end{cases} \tag{12}$$

Here we present a generalized algorithm for determining dependence relations for the given loop. Data dependence testing is done hierarchically. First, we begin with the assumption that data dependence direction vector is unknown ($\theta_k =' *'$,k[1:m]), and determine if the dependence exists for any vector θ. When the dependence is determined for the unknown vector θ, we have to concretize for which vectors θ it exists. The analysis has to be repeated, but with changed vector θ. In the vector θ, leftmost star will be changed with $' <'$, latter with '=', and at the last one with $' >'$. When we determine independence in some step, this vector θ need not be refined. The tree of analysis for m=2 is shown on the next figure:

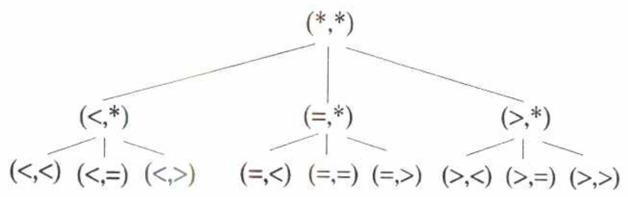


Figure 1.

The order of analysis is determined by PREORDER traversal of tree. If independence is determined for some node, the subtree whose root is that node, need not be analyzed.

3. Dependence testing on loops in which inner loop bounds are functions of outer loop indices

In practice, the loop with constant loop indices bounds are very infrequent. Loops of the form:

```
DO 10 I = 1, $N$
  DO 10 J = $I + 1, N$
```

or

```
DO 10 I = 1, $N$
   DO 10 J = 1, $N - I + 1$
```

are more frequent.

In the general case, we can assume that the inner loop bounds are functions of outer loop indices. We well look at the next, generalized loop:

```
DO 10 I1 = $T1, U1$
   DO 10 I2 = $t2(I1), u2(I1)$
        ⋮
      DO 10 Im = $tm(I1, I2, ..., Im - 1), u2(I1, I2, ..., Im - 1)$
           {loopbody}
10 CONTINUE
```

For the application of Banerjee's test, we must compute loop indices bounds, in first. In the phase of compilation that is impossible because these values are different for all different iterations of outer loops. Because of that, we introduce the worst case assumptions: for the lower bound of index I_i we take $t_i min$, but for the upper bound $u_i max$. Our problem, now, is reduced to the determination of minimums of functions t_i, and maximums of functions u_i. We will assume that the functions t_i and u_i are linear. In that case, the functions t_i and u_i can be described as follows:

$$t_i = T_{i0} + \sum_{j=1}^{i-1} T_{ij} I_j \tag{13}$$

$$u_i = U_{i0} + \sum_{j=1}^{i-1} U_{ij} I_j \tag{14}$$

If we know the minimums and maximums of indices I_j (j[1,i-1]) we can compute the lower and upper bounds of index I_i:

$$T_i = t_{imin} = T_{i0} + \sum_{j=1}^{i-1} (T_{ij}^{+} I_{jmin} - T_{ij}^{-} I_{jmax}) \tag{15}$$

$$U_i = u_{imax} = U_{i0} + \sum_{j=1}^{i-1} (U_{ij}^{+} I_{jmax} - U_{ij}^{-} I_{jmin}) \tag{16}$$

I_jmin and I_jmax are the lower and upper bounds of index I_j (T_j, U_j). From there, the values T_i and U_i can be computed from the next formulas:

$$T_i = T_{i0} + \sum_{j=1}^{i-1}(T_{ij}^{+}T_j - T_{ij}^{-}U_j) \tag{17}$$

$$U_i = U_{i0} + \sum_{j=1}^{i-1}(U_{ij}^{+}U_j - U_{ij}^{-}T_j) \tag{18}$$

These formulas determine the order of computation of bounds, too. Obviouly, bounds T_i and U_i can be computed if the bounds T_j and U_j for $j \in [1, i-1]$ are known.

4. Influence of data dependence direction vector on loops

Banerjee's test checks data dependence for all data dependence direction vector, independently. It imposes a question: can the data dependence direction vector influence the coefficients T_i and U_i?

If $\theta_i =' >'$, the lower bound of index I_i can not be equal to the value computed by formula (17), because there is not a value of index I_i smaller than t_{imin}. Because of that, for $\theta_i =' >'$, the lower bound of index I_i is necessary to grow for 1.

Similarly, it can be shown that for $\theta_i =' <'$, the upper bound of index I_i is necessary to reduce for 1.

Definitive formulas for computation the loop indices bounds are:

$$T_i(\theta) = T_{i0} + \sum_{j=1}^{i-1}(T_{ij}^{+}T_j(\theta) - T_{ij}^{-}U_j(\theta)) + CT_i(\theta) \tag{19}$$

$$U_i(\theta) = U_{i0} + \sum_{j=1}^{i-1}(U_{ij}^{+}U_j(\theta) - U_{ij}^{-}T_j(\theta)) + CU_i(\theta) \tag{20}$$

where:

$$CT_i(\theta) = \begin{cases} 0 & \text{for } \theta_i \in (*, =, >) \\ 1 & \text{for } \theta_i = < \end{cases} \tag{21}$$

$$CU_i(\theta) = \begin{cases} 0 & \text{for } \theta_i \in (*, =, <) \\ -1 & \text{for } \theta_i = > \end{cases} \tag{22}$$

T_i is dependent on these elements of direction vector j for which is $j \leq i$. Thist means that the order of computing of bounds T_i and U_i is identical to the order of dependence testing for corresponding vectors (see figure 1.).

It makes a question: whay do we have to correct the bounds T_i and U_i adding the coefficients CT_i and CU_i, when all this job is done by Banerjee's test, too? In our cases, inner loop bounds are the functions of outer loop bounds. Because of that, the coefficients CT_i and CU_i influence on values of all coefficients T_j and U_j for j¿i. The original Banerjee's test is not taking that influence into consideration.

5. Banerjee's test modification

As we desctibed, we and Banerjee's test, too, correct the lower and upper bounds dependently on corresponding direction vector θ. If we use the our computed loop bounds for dependence testing by Banerjee's test, we take these corrections into consideration two times. Because of that, we have to do a little modification of Banerjee's test. We will take the cases i='¿' and i='¡', because in these cases we must modificate Banerjee's inequalities.

Banerjee's test begins from assumptions:

$$T_i \leq I_i \leq J_i - 1 \leq U_i - 1 \qquad \text{for } \theta_i =< \tag{23}$$

$$T_i + 1 \leq J_i + 1 \leq I_i \leq U_i \qquad \text{for } \theta_i => \tag{24}$$

Instead of these, we take in next assumptions:

$$T_i(\theta) \leq I_i \leq J_i - 1 \leq U_i(\theta) \qquad \text{for } \theta_i =< \tag{25}$$

$$T_i(\theta) \leq J_i + 1 \leq I_i \leq U_i(\theta) \qquad \text{for } \theta_i => \tag{26}$$

In that case modificated Banerjee's coefficients have a next form:

(27)

$$LC_i = \begin{cases} -(a_i^- + b_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) - b_i & \text{for } \theta_k =' <' \\ -(a_i - b_i)^-(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) & \text{for} \theta_k =' =' \\ -(b_i^+ - a_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) + b_i & \text{for } \theta_k =' >' \\ -(a_i^- + b_k^+)(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) & \text{for} \theta_k =' *' \end{cases}$$

(28)

$$UC_k = \begin{cases} (a_i^+ - b_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) - b_i & \text{for } \theta_k =' <' \\ (a_i - b_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i & \text{for} \theta_k =' =' \\ (b_i^- - a_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) + b_i & \text{for } \theta_k =' >' \\ (a_i^+ + b_i^-)(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) & \text{for } \theta_k =' *' \end{cases}$$

Proof:

We need a next lemma [1] for proving of our assertion:

Lemma 1. Let f(x,y)=ax+by denote a linear function, and $U \geq q_0$ a number.

1) $min(ax + by : 0 \leq y \leq x \leq U) = -(a - b^-)^- U$

2) $max(ax + by : 0 \leq y \leq x \leq U) = (a + b^+)^+ U$

Let f1 and f2 be the index functions defined by equations (1) and (2), respectively; and let is:

$$h(I, J) = f_1(I) - f_2(J). \tag{29}$$

In given region, dependence exists for given vector θ **iff** the function h(I,J) has a null into that region.

Thus,

$$min(h(I, J)) \leq 0 \leq max(h(I, J)) \tag{30}$$

By combining (4),(5) and (29) we obtain:

$$h(I, J) = \sum_{k=1, \theta_k='='}^{m} (a_k - b_k) I_k + \sum_{k=1, \theta_k \neq '='}^{m} a_k I_k - \sum_{k=1, \Theta_k \neq '='}^{m} b_k J_k + a_0 - b_0 \tag{31}$$

Let we take a case $\theta_i =<$:

We need minimum and maximum of function:

$$f = a_i I_i - b_i J_i \tag{32}$$

Next inequality is derived from our assumption (25):

$$0 \leq I_i - T_i(\theta) \leq J_i - T_i(\theta) - 1 \leq U_i(\theta) - T_i(\theta) \tag{33}$$

Because of that we will transform the function f (32) on follows:

$$a_i I_i - b_i J_i = -b_i (J_i - T_i(\theta) - 1) + a_i (I_i - T_i(\theta)) + (a_i - b_i) T_i(\theta) - b_i \tag{34}$$

Now, by using of Lemma 1. we obtain:

for $\theta_i =<$:

$$LC_i = -(a_i^- + b_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) - b_i \tag{35}$$

$$UC_i = (a_i^+ - b_i)(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) - b_i \tag{36}$$

Second correction of Banerjee's test was done for $\theta_i =>$. In that case, we, first, must transform our assumption (26) as follows:

$$0 \leq J_i - T_i(\theta) + 1 \leq I_i - T_i(\theta) \leq U_i(\theta) - T_i(\theta) \tag{37}$$

The function f (32) can be written as follows, too:

$$a_iI_i - b_iJ_i = a_i(I_i - T_i(\theta)) - b_i(J_i - T_i(\theta) + 1) + (a_i - b_i)T_i(\theta) + b_i \tag{38}$$

Now, by using Lemma 1. we obtain:

for $\theta_i =>$:

$$LC_i = -(b_i^+ + a_i)^+(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) + b_i \tag{39}$$

$$UC_i = (a_i^+ + b_i^)(U_i(\theta) - T_i(\theta)) + (a_i - b_i)T_i(\theta) + b_i \tag{40}$$

This completes proof of our modification of Banerjee's test.

6. Conclusion

In designe process of a FORTRAN parallelizing compiler appeared a problem: how do we test the dependence on loops which bounds are not constant. In our testing examples, the most frequent cases were the loops in which inner loop bounds are linear function of outer loop indices. For dependence testing on loops with constant loop bounds we used Banerjee's test. It makes a question: is it possible to replace these functions with constants? We solwe this problem at next way: we change the lower bounds functions with their minimums, and we also change the upper bounds functions with their maximums. The expressions for computing these bounds values, when corresponding function are linear, are given in chapter 3 of this paper. We show that the loop bound values, in our case, are dependent on data dependence direction vector, too. We had to do some modification of Banerjee's test, because we take in the direction vector influence on loop bounds. Modificated Banerjee's test was presented in chapter 5.

References

[1] Z.Li, P.C.Yew, *An Eficient Interprocedural Analysis for Program Parallelization and Restructuring* (1988), ACM Press, New York.

[2] M. Walf, *Optimizing Supercompiler for Supercomputers*, Pitman, London, 1989.

[3] M. Walf, *The power test for data dependence*, IEEE Transactions on Parallel and Distributed Systems **3, No. 5** (September 1992), 591–601.

[4] H. Zima, B. Chapman, *Supercompilers for Parallel and Vector Computers*, ACM Press, New York, 1988.

[5] T. M. O'Keefe, H. G. Deitz, *Loop Coalescing and Scheduling for Barier MIMD Architectures*, IEEE Transactions on Parallel and Distributed Systems **4, No. 9** (September 1993), 1060–1064.

[6] T. H. Tzen, L. M. Ni, *Dependence Uniformization: A Loop Parallelization Technique*, IEEE Transactions on Parallel and Distributed Systems **4, No. 5** (May 1993.), 547–558.

Suzana Stojković, Faculty of Electronic Engeneering, Computeer Science departmetnt, Beogradska 14, 18000 Niš

FILOMAT (Niš) 9:2 (1995), 345–355
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

THE GENERATION OF PERMUTATIONS THROUGH GDD

Dragan Janković and Milena Stanković

ABSTRACT. *In this paper we consider the generation of permutations, i.e. all ordered n-tuples of different elements from the set $A_n = \{a_0, a_1, ..., a_{n-1}\}$ which is a combinatorial problem often occurring in practice. We give a method for the generation of all permutations of n given items through generalized decision diagrams. Each of n! paths in the appropriate decision diagram maps into one of n! permutations. The proposed method is suitable for generating all permutations for direct generation of only one permutation without generating and saving preceding permutations. Our method provides efficient hardware realization.*

1. Introduction

The generation of permutations on a given set $A_n = \{a_0, a_1, ..., a_{n-1}\}$ with n elements is in fact the generation of all ordered n-tuples of elements from A_n. This problem occurs frequently in practice as a part of many complex combinatorial problems. For example, many problems in logic design: minimization, simetry examination, NPN classification or function decomposition are combinatorial problems in solving of which different permutations of variables or function values of examined functions are often required [3,6,7]. Important field for application of permutations are permutation interconnect networks which are consistent parts of many multiprocessor systems for discrete transform calculation (DFT, WHT, ...) [4]. In this paper we use generalized decision diagram to generate the permutations.

The basic idea of the presented method was found in the representation of switching functions by binary decision diagram (BDD) [1,2]. BDD for an n-variable switching function is a binary tree with n-levels and 2^n terminal nodes. The terminal node values are the function values of the represented switching function.

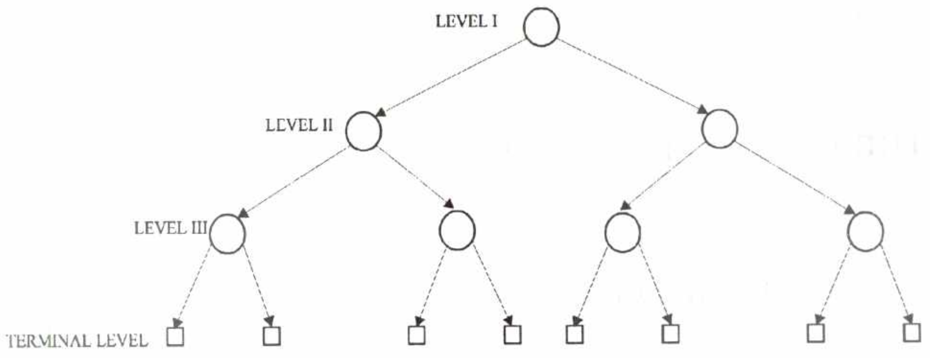


Figure 1. BDD

Example 1. BDD for three variable function is shown in Fig. 1.

If it is allowed that nodes at different levels have different number of edges (assuming that the number of edges of all nodes at one level is equal) we obtain generalized decision diagrams (GDDs) suitable for the representation of the multiple valued functions [5].

Example 2. The typical GDD is shown in Fig 2.

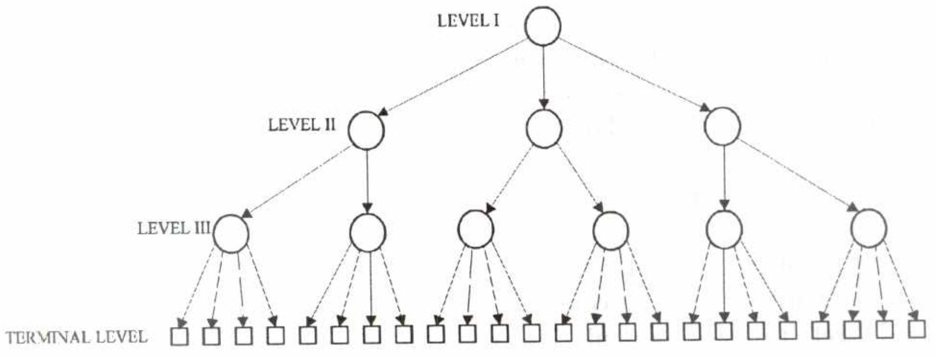


Figure 2. GDD

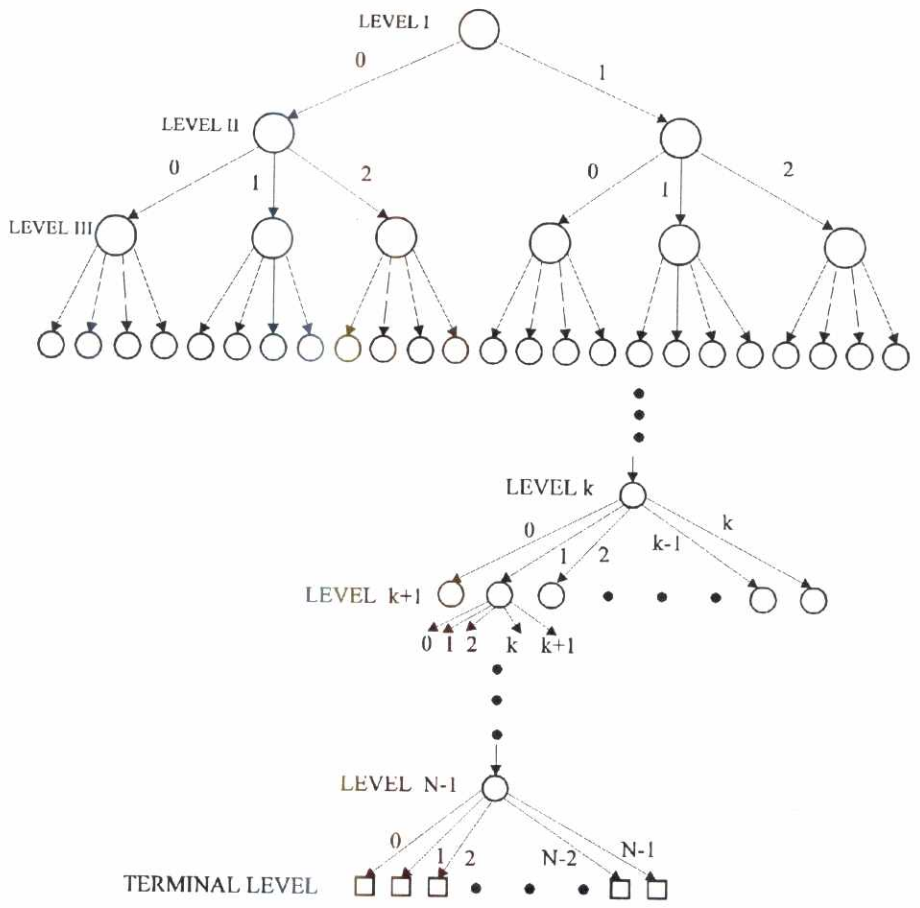


Figure 3. GDD for generation of the permutations

2. The representation of permutations with decision diagram

It is possible to represent all permutations of items from set A_n through particular GDD consisting of $n-1$ levels. The first level consists of one node (root node) with two edges, the second of two nodes with three edges, the third of six nodes with four edges, etc. There are $k!$ nodes with $k+1$ edges at k-th level (Fig. 3).

Thus defined GDD with $n-1$ level has $n!$ terminal nodes. Therefore, we can assign one of $n!$ permutations to each node. The GDD nodes are denoted as shown in Fig. 4., where we have a node at k-th level denoted by q connected to the root node by the path p. The node q is connected to $k+1$ nodes $(q_0, q_1, \ldots, q_k)$ at (k+1)-st level by the output edges denoted by $0, 1, \ldots, k$, respectively. If the string $x_0 x_1 \ldots x_{k-1}$, $(x_i \in A_n$, for $i = 0, 1, \ldots, k-1)$ is assigned to the node q, to the each node q_i may be assigned the string derived by the following rule:

$$q_i = x_0 x_1 x_2 \ldots x_{i-1} a_k x_i \ldots x_{k-1} \quad for \quad i \neq k$$

$$q_k = x_0 x_1 x_2 \ldots x_{k-1} a_k$$

Each node q_i is connected to the root node by the path $p_i = pi$.
For k=0 (root node) $q = a_0$ and $p = 0$.

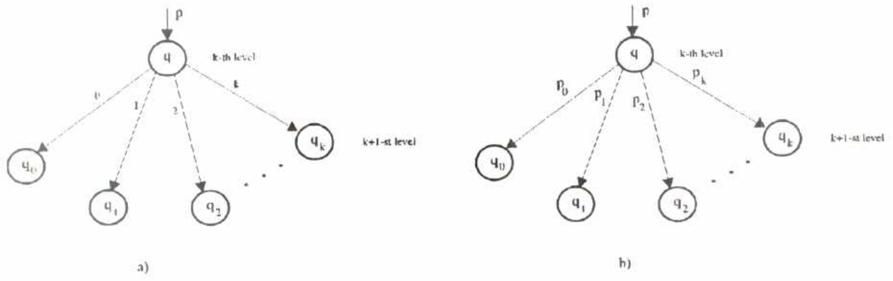


Figure 4. a) The node notation b) The path notation

With the introduced notation, each of the n! terminal nodes corresponds to one permutation, as shown in Fig. 5. for $A_4 = \{0, 1, 2, 3\}$.

3. The procedure for generation of permutations

For the generation of a particular permutation the corresponding path from the root node must be found. Moving from one to another level along this path we generate the required permutation. When we move from the level k to the level $k+1$ trough the edge i we insert the value a_k at i-th position in the generated string. Repeating this procedure for all terminal nodes (moving along all pats in the GDD) we obtain all permutations of n items (Fig. 5).

The decimal index of permutation path, $Dec(p)$, is defined as:

$$Dec = \sum_{i=1}^{N-1} g_i N!/(i+1)!$$

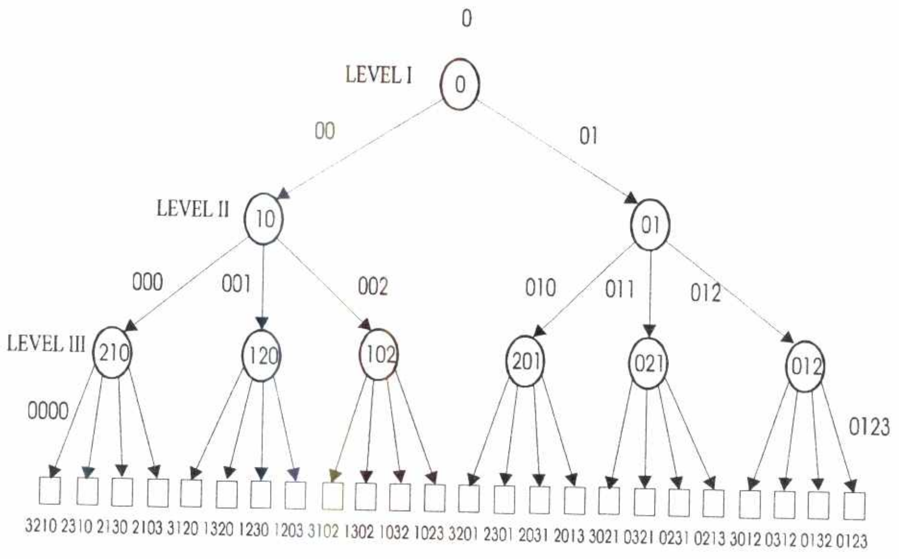


Figure 5. GDD for n=4

where $p[i]$ is i-th element in permutation path p.

All permutations are ordered on the basis of $Dec(p)$. The ordering $\diamond$ can be defined as follows:

Let P and Q be two distinct permutations with paths p and q, respectively.

$$P \diamond Q \;\; if \;\; Dec(p) < Dec(q).$$

For example, for $n = 4$, the permutations of $A_4 = \{0,1,2,3\}$ ordered according to $\diamond$ are given in Table 1.

This ordering is very useful for generation one permutation or the permutations from interval. We generate the permutation from decimal index. The decimal index to be mapped to the permutation path after which the described procedure is applied. For this method generating and saving all previous permutations are not necesserly. This method is not recursive, which is very important for execution time and permutation length. The permutation length is practically unlimited in this method. No permutation is generated again. Therefore, there is no need to check whether the permutation has been generated earlier.

Table 1: Decimal indices of the permutations

Dec.ind.	path	perm.
0	0000	3210
1	0001	2310
2	0002	2130
3	0003	2103
4	0010	3120
5	0011	1320
6	0012	1230
7	0013	1203
8	0020	3102
9	0021	1302
10	0022	1032
11	0023	1023
12	0100	3201
13	0101	2301
14	0102	2031
15	0103	2013
16	0110	3021
17	0111	0321
18	0112	0231
19	0113	0213
20	0120	3012
21	0121	0312
22	0122	0132
23	0123	0123

4. Implementation

The implementation of the described procedure for the generation of all permutations is given as follows:

```
1. initialization (the length of the permutations and the beginning path)
2. for i=2,n do
     begin
       2.1 shift all the permutation elements from p-th element for one
           position to the right (the element p is the weight of the i-th
           element in the path)
       2.2 set the i-th element at the p-position
```

```
    end
3. print the generated permutation
4. generate the new path
5. if the generated path is different from the beginning path go to step 2
6. stop
```

Some advantages of GDD can be used in implementation. There is no need to move along the complete path for each permutation. It may be continued from the position where the new path is different from the old one. In this way the execution time may be decreased considerably with the increase of N, as shown in Table 2 where the execution time is given (in millisecond) using the complete path and a part of the path too.

Table 2: The execution times when the complete path and a part of path are used

len.perm	complete path (ms)	a part of path (ms)
2	0.017	0.023
3	0.05	0.06
4	0.3	0.28
5	1.9	1.4
6	15	9
7	134	70
8	1318	606
9	14240	5830
10	171600	62200
11	2158000	738000

5. The modification of basic procedure

Described procedure can be modified according to some specific requirements of the application of permutations. If the generation of a permutation or permutations from interval are needed, then it is enough to run the corresponding initialization (set the value for array $NIVO(i)$, $i = 0, n-1$). The generation of a permutation from another one is the problem that often appears in practice. In this case, our method is very successful. The move from one permutation to another one is executed by the following procedure:

1. starting from the terminal node corresponding to the beginning permutation and then moving up to the crossing of the beginning and the desired permutations.

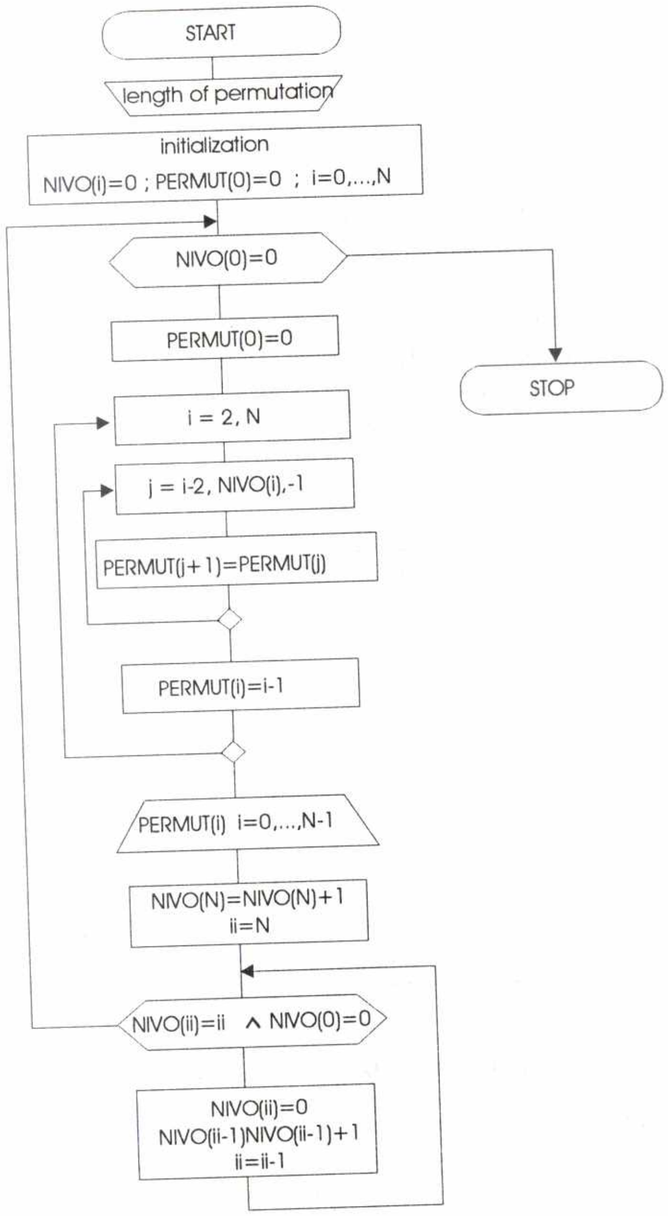


Figure 6. Algorithm for the generation of all permutations with length n

2. moving down to the terminal node corresponding to the desired permutation. Moving up, the elements are ejected from the sequence (i.e. permutation), and moving down, the elements are inserted into the sequence.

The example for the generation of permutation 1032 from 2310 permutation is shown in figure 7. The moving through graph is depicted by a dotted line.

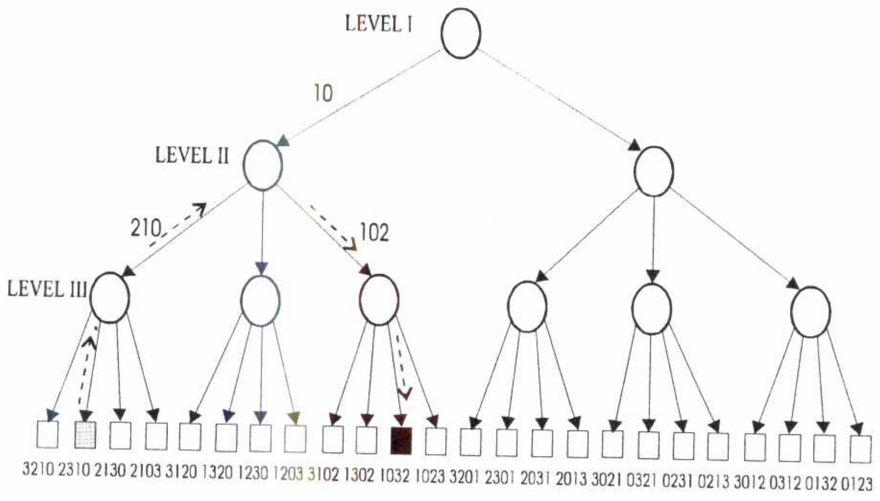


Figure 7. The generation 1032 permutation from 2310 permutation

6. The hardware implementation

Our method provides efficient hardware realization. The types of hardware realization depend on the actual application. As an example, the descriptions of pipeline realization, shown in figure 8, follow. The generation of permutation of n items requires n processing elements (PEs). Every PE has two inputs and one output. PE passes one of the two inputs depending on the state of the counter which runs as the adder modulo k if PE is at k level (i.e. k-th in pipe). If the immediate state of the counter is p, the PE passes p inputs X, (and) afterwards input Y and finally $n-p-1$ inputs X. The counter of PE at level k changes its state when the state of counter of PE on level $k+1$ becomes $k+1$. In other words, every PE activates the counter of the previous PE (Figure 9).

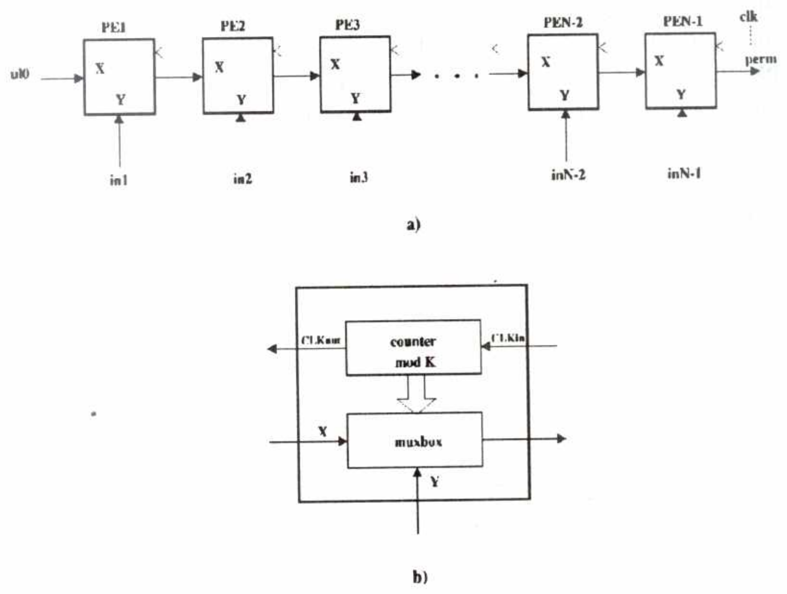


Figure 8. a) Pipeline system b) PE

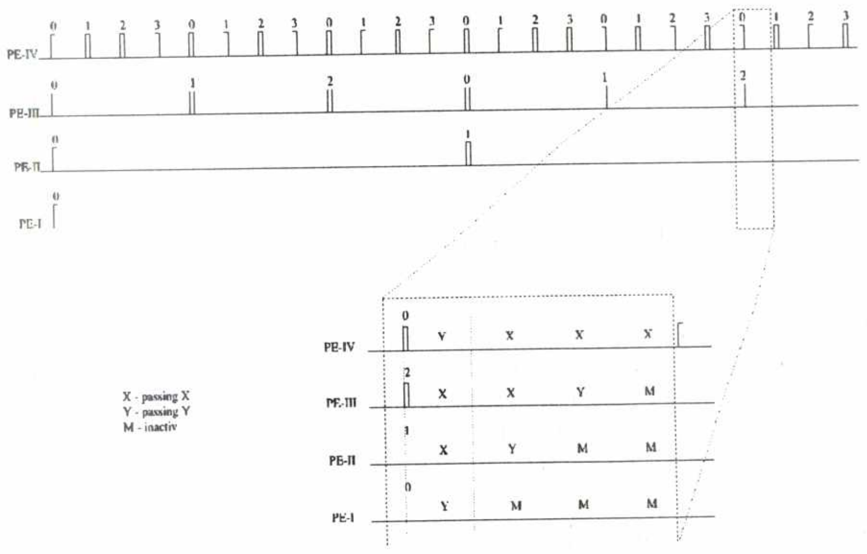


Figure 9. The state diagram of the counter of PEs for n=4

7. Conclusions

In this paper, we propose the method for the generation of the permutations with unlimited length, through GDD. We define GDD appropriated for the generation of all the $n!$ permutations of n given items. Based on GDD, the efficient procedure for mapping the paths in GDD into permutations is also presented. The proposed method is suitable for both software and different hardware realizations. As an example, pipeline realization is described.

References

[1] S. B. Akers, *Binary decision diagram*, IEEE Transaction on Computers **C-27, No. 6** (June 1978), 509–516.

[2] R. Bryant, *Graph-based algorithms for Boolean function*, IEEE Transaction on Computers **C-35, No. 8** (August 1986), 677–691.

[3] D. Cvetković, *Diskretne matematičke strukture*, Naučna Knjiga, Beograd, 1987.

[4] P. Fragopoulou, S. G. Akl, *A parallel algorithm for computing Fourier transforms on the Star graph*, IEEE Transaction on Parallel and Distributed Systems **5, No. 5** (May 1993), 525–531.

[5] D. Janković, R. Stanković, M. Nikić, *Calculation of the Fourier transform on finite Abelian groups through GDD* (1994), Proc. Yugoslav Conference for ETRAN, Niš, Yugoslavia.

[6] C. J. Lin, *Parallel generation of permutations on systolic arrays*, Parallel Computing **15** (1990), North-Holland, 267–276.

[7] R. Sedwick, *Permutation generation method*, Computing Surveys, **9, No. 2** (1977), 137–164.

Faculty of Electronic Engineering Computer Science Department, Beogradska 14, 18000 Niš

FILOMAT (Niš) 9:2 (1995), 357–366
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

A SYSTEM FOR STORAGE, MANIPULATION AND CONTROL OF DIFFERENT GRAPHICS FORMATS

Zoran Putnik

ABSTRACT. *In this paper, a detailed outline of a system for memorizing, manipulation and control of pictures given in different graphic formats is given. System consists of several modules, already known and available, but the value of the system is mainly in combination of several useful functions, enabling complete and efficient management of miscellaneous kinds of pictures and cutting on expenses and possible errors in manipulation with various graphics formats.*

1. Introduction

Manipulation of drawings and other graphics elements is mush more then just a simple storing/retrieving of data and drawings. It is rather a complicated process of drawings' creation - starting by a designer, external and internal skilled consultants, through artists who actually make drawing, up to users of the finished drawings, or some of its parts. During drawing creation, standard parts from shared or private libraries are incorporated, or referred to, and usual necessary data - names, dates, references, are given. Dates of drawing creation are still not the final dates of need for a drawing. Often changes, especially for technical drawings, demand easy access to a drawing for a long period. This demand, naturally includes a need for some tools for transferring drawings from one graphics format to another. As for any other data stored in a computer, manipulation with drawings requires handling of standard problems: efficient storing system, fast and simple data retrieval, enabling changes in existing drawings or using existing drawing in creation of a new ones, managing an efficient data base about drawings and related data, transferring drawings from paper to a computer and similar.

1991 *Mathematics Subject Classification.* 68U05, 68P15.
Work supported in part by Ministry for Science of Republic Serbia

Toward overcoming of mentioned problems, in this paper, a system for storage, manipulation and control of drawings in different graphics formats is given. Separate modules, this system consists of, are not new nor original, instead, most of them are available for a commercial use in some form. Value of this system is mainly in unifying and combining all necessary functions, enabling simple and efficient control of data and drawings flow even for a long period and reducing expenses and chance for errors in drawings' manipulation.

2. State of the art

We can notice several different logical modules in modern systems for information and documents management (from now on CDMS - Corporate Document Management Systems):

- module for storing information - data base
- module for data search - key-words data base
- module for documentation viewing and control
- communication module - fax, e-mail, modems
- module for controlling computer network
- module for changing documents (in original programs)
- module for automatic text recognition
- module for handling pictures

Separate programs for each of the mentioned modules are developing for years (more or less successfully), everything toward creation of "paperless office." Information stored in digital - electronic form, does not need a paper as a storage media. But, to be easily available to the user, it demands another elements of a system for data storing and retrieval. As main elements, we can mention:

- computer for data storage - "main computer"
- (computer for communication with a "main computer")
- software for "reading" and "presenting" given information

As much as textual data are concerned, several mostly used text-processors can be identified, that each CDMS have to support, with always present, final solution, of recognizing text in its simplest form - ASCII standard. For graphics data, such standards do not exist. We can talk of "most frequent" graphics forms, i.e. *.PCX, *.TIF, *.GIF, *.IMG - as bit-mapped, or *.DWG, *.DXF or *.CDR - as vectorized, but, basic standard does not exist jet.

Computer system for storage, manipulation and control of graphics formats is very important subsystem of a system for creation, management and archiving documents - CDMS. It must successfully and efficiently integrate some commonly accepted programs for graphical documentation management, from scanning and character recognition programs, through programs for editing bit-mapped, vectorized or ASCII graphics files and transferring drawings from one form to another, up to programs for presentation and printout graphical documentation on various kinds of output devices. This subsystem is also an useful step towards creation of a multimedial data base, which will enable fast and simple finding, retrieval and exporting any document stored in any existing form. Modular and flexible, this subsystem has to be (theoretically) usable equally in small and in big business systems, despite working area. In its nature, such subsystem assumes (and gives best results) computers connected in a network, which again permit successful control of document flow, transforming several "personal" computers in an efficient information system.

A system for storage, manipulation and control of graphics formats (somewhere called EDM, standing for "Engineering Drawing Management" or "Electronic Data Mng" or "Engineering Data Mng") has to emulate, for a successful work all standard activities in a process of creation, storing and "maintenance" of drawings. Main of these activities are:

- Control function - **DIRECTOR** - a module handling work and communication of other modules, controlling them and controlling users' behavior according to his priority level;
- Data storage function - **LIBRARIAN** - a relational data base, enabling a search for a specified drawing based on a key-word and creation of a report on a drawing including place and time of creation, author, dates and types of changes, current status, list of key-words and list of access rights;
- Digitalization function - **SCANNER** - a module for connection between "old" and "new" technology of drawing creation and for connection with third parties, which produce their drawings in a paper form. It should also contain some standard way of data compression (for example - scanned drawing of A0 format, with resolution of only 400 dpi, as a result requires 40 MB of storage space if stored in a bit-mapped form);
- Editing function - **EDITOR** - a module that enables that drawing we want to change (coming through module LIBRARIAN or module SCANNER) can be edited either with standard geometric functions (scaling, rotation, translation ...) or manually (adding or deleting picture pieces, coloring, text editing...)
- Vectorization function - **VECTOR** - a module that (if needed) enables transformation of bit-mapped drawing into a vectorized drawing. Experience shows that this function is not always necessary,

since very good abilities for changing raster images are developed, and on the other hand, vectorization process takes a lot of time not always bringing significant improvement of quality;

- External communication function - **TRANSFER** - a module that overcomes a problem of using different software for drawing creation and enables combining of drawings created on different places in a different ways;
- Text recognition function - **READER** - a module enabling usage of text documents created in most standard text processing programs, or, if nothing else is possible, enables text recognition using usual optical character recognition techniques;
- Viewing and printing function - **OUTPUT** - a module enabling that library drawings, can be viewed on (any kind of) a screen, and/or plotted/printed on (any kind of) a printing device.

3. System modules

3.1. Drawings storage module - data base.

Creation of a complex drawing, consisted of several drawings, sometimes already created in different graphics formats, using different software tools, is usual very difficult. Reason for this is existence of three principally different formats - bit-mapped drawings, vectorized drawings and drawings created of ASCII characters - with a huge number of subtypes for the first two. Emerging of a new version of existing graphics software, usually brings lots of problems to the end-users. Besides that, for each graphics document, some extra information is needed, for example: date and time of creation and editings, names of authors, coauthors, consultants and "maintenance" employees, references to parts taken from standard libraries or to bigger drawings of which the given one is a part of, and so on. The most convenient method for storage of this kind of data is some standard, relational data base, which will enable easy sorting, searching and editing of existing data.

This module has to provide a simple and obvious searching method through the graphical data base on any criteria, without previous knowledge of programming languages or data bases. This can be achieved through a simple and readable graphics interface, enabling easy entering of wanted search criteria. Multiple criteria search, easy access to the results of a previous search and other similar, practical options are usual in any serious data base, so there is no need to explain them separately.

As a first result, a search gives simplified, smaller picture of all drawings satisfying given criteria. Later, those pictures, depending on users access level, could be viewed, edited, printed, commented and so on. Naturally, for advanced users, it would be very useful to have programming language, which can define either aestethic (i.e., shape of a search screen) or essential search details (definition of new fields of a data base, with their attributes, creation and organization of an archive of technical and business documentation, catalogues of products, data bases of persons involved in drawings creation and so on).

3.2. Module for transferring paper documentation into electronic form.

Because of paper documentation inherited from previous work and because of need for cooperation with other parties producing paper drawings, this module is necessary in any system for storage, manipulation and control of graphics formats. It should cover following functions:

- picture scanning
- editing of errors of scanning
- optical character recognition
- editing of bit-mapped pictures
- picture vectorization
- editing of vectorized pictures
- editing of ASCII pictures

3.2.1. Subsystem for scanning.

Any "real-life" business system, besides documentation created on a computer, is doomed to have contact with paper documentation. That documentation is, seldom or rarely, used, saving of some documentation is usually legal obligation. Transformation of that documentation into an electronic form by repeated drawing is usually too complicated and too expensive. Instead, it is more natural to keep it in a computer archive in a form of scanned pictures. After scanning, these pictures can be edited more or less, vectorized, if necessary, or transformed into text, which all are parts of subsystems that will be mentioned later.

Process of scanning and editing of scanned pictures, should be, according to latest trends in this field [5], equipped with tools for performing following functions:

- scanning errors' correction
- straightening of aslanted pictures
- removal of "snow" emerging because of a dirt on a paper
- thickening or thinning of lines
- definitions of separate, different filters, for specific parts of a picture
- linking of disconnected contours, or separating of badly connected contours
- standard functions for adding, editing and deleting parts of drawing

3.2.2. OCR subsystem.

This module confirms to all standard demands for this class of programs, which will not be especially discussed in this context. It should only be emphasized that this subsystem has to supply a connection between a graphics document in an unknown format and ASCII file obtained by process of scanning and optical character recognition. This is, naturally, performed only as a final measure, if information about the contents of a picture cannot be obtained by any other means.

3.2.3. Subsystem for picture editing.

This is again a standard subsystem, that should not be explained in much details. It should be only mentioned, that this subsystem in fact is consisted of three separate parts, for editing different types of drawings - bit-mapped, vectorized and ASCII character drawings. Since one drawing can be created as a combination of all these types, all editing tools have to be available at any moment.

3.2.4. Vectorization subsystem.

There is often a need for large amount of changes that should be performed on an existing drawing. This is usually much easier (end with higher quality) performed on a vectorized picture. Besides, vectorized picture, compared to a bit-mapped picture, usually take much less space, which is a very important demand in this field,. Considering all mentioned, subsystem for vectorization is an obligatory part of a system for storage, manipulation and control of graphics formats.

There is a set of standard tools for this process and usual procedures for manual and automatic vectorization. Here, some more advanced actions about vectorization will be underscored:

- definition of vectorization "filter" (for example artistic or technical, or even more specific - electronic, architect, engineering ...), which as a consequence, brings different definitions of some standard vectorization parameters:

(1) characteristics for approximation of curves,
(2) definition of smallest object that is vectorized,
(3) minimal offset of horizontal/vertical line that is not neglected,
(4) method of text recognition,
(5) minimal distance that separates two lines and so on.;

- enabling manual or automatic vectorization and vectorization of a whole picture, of a part of a picture or definition of a part of a picture that should not be vectorized;

- enabling recognition of at least some basic contours - circle, ellipse, square, for example - as contour, and not as a combination of simple lines and curves. The same should stand for a combination of those basic shapes. For example, a square written *IN* a circle, should be vectorized as those two contours, and not as a combination of four lines and four curves;
- text, as a part of a picture, can be recognized either as graphics (transformed to curves), as letters (i.e., optical character recognition of ASCII characters) or completely removed from a drawing;
- after finished automatic vectorization, there should be a possibility for comparison of bit-mapped original and achieved result. Naturally, there should be an ability for additional, manual changing of vectorized picture;

3.3. Output module.

This module has to enable rough and/or detailed view of "all" important graphics formats - bit-mapped or vector, including documents created by important text-processing programs, spreadsheets or data-base programs. For this module, only a quick and simple access to document is important, including output abilities on all output devices, screens, printers and plotters. Eventual changes of documents should not be incorporated into this module, since these abilities are a part of another modules.

3.4. Module for manipulation of technical drawings.

Special problem in this field is production, maintenance and editing of technical drawings. During creation, technical drawings go through many phases of treatment, addition and editing, so that, as a result there is too many paper versions of a drawing, usually right one at the wrong place. Chief problem with technical drawings (for example drawings of bridges, buildings and similar) is that they have to be saved and maintained for several tenths of years.

Unfortunately, introducing computer aided design (CAD) into this area, can put us in an even worse situation. Part of documentation is saved on a computer, part on a paper, some initial versions of a drawing are declared final, while some final versions are rejected as unnecessary. In order to overcome these problems it is urgent to, right after introduction of computer aided design, transfer all documentation into electronic form, no matter of what kind, origin or shape they are and organize a data base to accompanies that documentation. Later phase will usually demand several computers connected into network.

3.5. Communication module.

This module has to enable safe, fast and easy communication between different modules in a system for storage, manipulation and control of graphics formats. As much as an user is concerned, it should supply simple usage, different methods to perform functions (keyboard, mouse, arrows ...), readability of a screen, easy-to-use help system and all other standard requirements for a proper user-friendly graphic interface [7].

4. Future development

It seems, considering fast development of science, especially computer science, that it will be possible in near future to spread system like this one in several different areas. Even though commercial versions are still unavailable, some fields are developing very fast and we can expect soon expansion of system for storage, manipulation and control of graphics formats, for example with:

(1) Optical recognition of text - not characters

Latest research shows [1] [4], that optical character recognition systems are very close to their upper limits. Although those limits are rather high (over 95%), for large texts, and, more important, for texts that allow no errors, this is insufficient. Consequently, organizations that want to work with "electronic documents" cannot rely on them. These facts, initiated research in a field of optical recognition of texts, based on analysis of a document structure and its contents. A system like that, must contain several text characteristics: big dictionaries, text styles, font types, document styles and structure, word meanings, relationships between words and assumptions about text contents - expected contents, expected contents of certain parts or knowledge on relationship between text and field of its application.

(2) Intelligent interpretation of a drawing

Drawings, especially technical, could be scanned and recognized, much better and more precise, by using certain algorithms for determining location of textual parts of a drawing and its separation, or methods for analysis of scanned drawings in order of acquiring regular shapes, instead of set of lines, irregular in their shape, size and thickness, algorithms for recognizing fill patterns and similar [2] [3].

5. Comment instead of conclusion

By some available statistics (from year 1992) [5], it is estimated that there is over 15 billion of paper drawings used in different companies, which have to be used and controlled, and that only 13% of them are in electronic form. It is also estimated, that over 10% of those drawings are lost or misplaced, because of inefficient organization, and that, only in USA, about 43 million man/hours are spent on storage, search, copying and other manipulation of paper graphics documentation and space of over 1.5 million of square meters is used for drawings storage and saving.

There is a lot of legal and practical reasons to store drawings for several years, including potential need for drawing editing. Drawings used in machine construction, had to be treasured as long as machines are produced, and even later, because of maintenance. The same, but for much longer period, stands for architecture drawings or civil engineering, for example.

Introduction of CAD systems, aimed for improvements in a field of productivity in drawings creation, easier usage of graphics libraries, easier storage, editing and communication with drawings. But, need for communication with companies not using electronic systems for picture manipulation, forced a situation in which every company had to keep people, offices and working methods, for handling both paper and electronic drawings. Consequently, instead of increase in productivity, that usually lead to duplicated capacities and decreasing of efficiency, because of a need for cooperation between two very incompatible, parallel systems.

Everything mentioned, clearly shows urgent need for creation and usage of efficient system for storage, manipulation and control of different graphics formats, toward which this paper hopefully leads.

References

[1] A. Dengel, *Stepping from Automatic Spelling towards Automatic Reading*, International Summer School "Information Technologies and Programming" Sofia (1992).

[2] S.H. Joseph, T.P. Pridmore, *Knowledge-Directed Interpretation of Mechanical Engineering Drawings*, IEEE Trans. on Pattern Analysis and Machine Intelligence **14, No. 9** (1992).

[3] R. Kasturi, S. Bow, W. El-Masri, J. Shah, J. Gattiker, U. Mokate, *A System for Interpretation of Line Drawings*, IEEE Trans. on Pattern Analysis and Machine Intelligence **12, No. 10** (1990).

[4] K. Kukich, *Techniques for Automatic Correcting Words in Text*, ACM Computing Surveys **24, No. 2** (1992).

[5] T. Maxwell, *Engineering Drawing Management*, DECSYM'92 - Latest Trends in Computing.

[6] J. McKendree, J.M. Carroll, *Proceedings of CHI'86 Human Factors in Computing System*, ACM, 1986.

[7] Z. Putnik, *Intelligent HELP System as a Help in Educational Process*, Proceedings of IV Symposium "Informatics in Education and new Educational Technologies", Novi Sad, (in Serbian) (1994), 86-91.

[8] R. Radiev, V. Dimitrov, N. Marinov, *A System for Development of Intelligent Interfaces*, Proceedings of 17 School with Conference Information Technologies and Programming, Sofie, Bulgaria (1992).

Zoran Putnik, University of Novi Sad, Faculty of Science, Institute for Mathematics, Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia

FILOMAT (Niš) 9:2 (1995), 367–376
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ONE METHOD OF IMPLEMENTATION OF LISP INTERPRETER TO TRANSPUTERS

Jozef Kratica

ABSTRACT. *The paper describes one method of implementation of LISP interpreter to transputers. Developed interpreter contains standard functions common for almost all LISP versions. Architecture is binary tree message passing. Implementation was developed on transputer parallel C language (ANSI C with procedures for interprocessor communications and synchronization). Part intended for evaluation of functions (expressions) was parallelized, but I/O operation and parsing were sequential. This is caused by the technical limitations of transputer systems, because I/O operations can executed only by first transputer, and interprocessor communication is slow. Maxima increase in speed equals 6.5 times, on transputer system with 17 transputers T800, by as compared to single transputer T800. That increase in speed is obtained for recursive problems demanding much computing. Small increase in speed is obtained for problems with more I/O operations.*

1. Implementation method

In LISP implementation on uniprocessor machines ([2], [3]), the basic part for parallelizing is part for evaluating expressions (functions). Provided that only first transputer can perform I/O operations, these operations (I/O) must be executed sequentially. Parsing functions are also executed on the first transputer, because interprocessor communication is slow. First transputer sends function definitions to other transputers when they need them (when other transputers evaluate functions).

Technical limitations of transputer systems are ([7]):

a) Every transputer have 4 links to other transputers;

b) Every transputer must be reset (one of its 4 links) by other transputer. Only first transputer is reset by the host.

c) Every transputer can reset maximally another 2 transputers, one by system, and the other by subsystem reset link.

Graph theory defines precisely technical limitations by term RS complete graph maximal degree 4. [3]

Binary tree architecture satisfies technical constrains of transputers (RS complete graph maximal degree 4) [3]. Binary tree architecture is applied in this paper.

Transputers can be grouped in 3 categories:

a) The first transputer;

b) Transputers that have successors (transputers with numbers 2-8.);

c) Transputers which have no successor (other 9 transputers).

File with NIF extension describes architecture (configuration) of the transputer system. Example of NIF file for our implementation, which contains 17 transputers T800 is shown below:

```
1,    lisptr1,  R0,   0,    2,    3,    ;
2,    lisptr2,  R1,   4,    1,    5,    ;
3,    lisptr2,  S1,   6,    7,    1,    ;
4,    lisptr2,  R2,   2,    8,    9,    ;
5,    lisptr2,  S2,   10,   11,   2,    ;
6,    lisptr2,  R3,   3,    12,   13,   ;
7,    lisptr2,  S3,   14,   3,    17,   ;
8,    lisptr2,  R4,   18,   4,    19,   ;
9,    lisptr2,  S4,   ,     ,     4,    ;
10,   lisptr2,  R5,   5,    ,     ,     ;
11,   lisptr2,  S5,   ,     5,    ,     ;
12,   lisptr2,  R6,   ,     6,    ,     ;
13,   lisptr2,  S6,   ,     ,     6,    ;
14,   lisptr2,  R7,   7,    ,     ,     ;
17,   lisptr2,  S7,   ,     ,     7,    ;
18,   lisptr2,  R8,   8,    ,     ,     ;
19,   lisptr2,  S8,   ,     ,     8,    ;
```

Every line contains:

a) Number of the transputer (the first transputer must be connected to the host by link 0);

b) Name of a program that will be executed on that transputer;

c) R or S (system or subsystem reset), and number of the transputer which will reset him;

d) Number of the transputer which is connected by link 0;

e) link 1;

f) link 2;

g) link 3;

Free connection by that link marking empty place.

Example: 5. Transputer execute LISPTR2, reset by subsystem link of 2. Transputer (S2). Link 0 connects to transputer number 10, link 1 to transputer 11, link 2 to transputer number 2. Link 3 is free.

Configuration of the transputer system given in previous NIF file is binary tree (Figure 1). More about a configuration of a transputer in a network is presented [3] and [7].

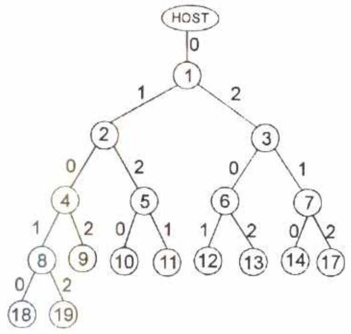


FIGURE 1. Architecture of the transputer system

1.1 Work done by the first transputer.

First transputer performs following operations:

a) Loading input data;

b) Parsing input data for definitions of user-defined functions;

c) Saving that definitions;

d) Saving names of variables and functions;

e) Parsing function calls from input data;

f) Printing output results;

g) Deciding about the execution of the functions (whether to execute function itself, or to send it to "successors").

1.1.1 Calling of user-defined function.

If the first transputer evaluates user defined function, two cases can arise:

a) If the function contains only calls of built-in functions, the first transputer itself evaluates all parts of the function, because in many cases this evaluation is short.

b) In case that the user-defined function also contains calls of other user-defined function (functions), much computing can be expected. In that case, if some of "successors" are free, this transputer sends parts of those user-defined function to free "successors" for evaluation. If all "successors" are busy, then this transputer itself evaluates all function calls.

1.1.2 Calling of built-in function.

In this case, the first transputer performs all computing alone, because the evaluation of calls of built-in functions is short.

1.2 Work performed by the transputers that have "successors".

Every transputer that has "successors" (in this configuration of 17 transputers, these are transputers Nos. 2-8), waits for the message "COMPUTE" from "parent" transputer.

After receiving the message "COMPUTE", it receives the following data:

a) Expression (function) that it will evaluate;

b) Names and values of variables in that expression;

c) Definitions of functions, that the expression (function) needs for the evaluation;

d) Contents of argument stack in that moment.

After that, the transputer evaluates function calls, in the same way that the first transputer does. After the end of the evaluation of that function call, the transputer sends the "FREE" command to the "parent", and saves a result to its communication stack.

In the moment in which the "parent" needs this result, the transputer loads this value from his communication stack and sends it to the "parent".

1.3 Work performed by transputers that have no "successors".

Every transputer that has no "successors" (in this configuration of 17 transputers, these are transputers Nos. 9-19), waits the message "COMPUTE" from the "parent" transputer.

After the receival of the message "COMPUTE", ite receives the following data:

a) Expression (function) which it will evaluate;

b) Names and values of variables in that expression;

c) Definitions of functions, which the expression (function) needs for evaluation;

d) Contents of argument stack in that moment. After that, it itself evaluates the function call (because it has no "successors").

After the end of the evaluation of the function call, a transputer sends the "FREE" command to a "parent", and saves the result to its communication stack. In the moment in which the "parent" needs this result, the transputer loads this value from its communication stack and sends it to the "parent".

2. Realization

The implementation of LISP interpreter for transputers (multiprocessors), was based upon the corresponding implementation for uniprocessor machines

[3]. Changes in parts of implementation for uniprocessor machines are minor. The implementation for multiprocessors (transputers) contains two new parts:

1. Argument passing and
2. Control part

In this implementation there are two segments of the program:

a) the segment which will be executed on the first transputer;

b) the segment that will be executed on the other transputers. This segment does not contain the procedures which other transputers cannot execute (I/O operations, parsing, ...).

2.1 The segment for the first transputer.

2.1.1 Argument passing.

The parallel C contains only procedures intended for passing of integers or characters to (from) communication channels. In the program the complex and powerful data structures (pointers, linked lists, ...) and procedures necessary for passing those data structures to (from) communication channels were used. This part of the program contains procedures that enable those possibilities.

2.1.2 The control part.

This is the most important part of the program.

It performs following operations:

a) receives messages from input channels, and performs their commands;

b) takes note of transputers which ended theis previous evaluation, and now are free;

c) when it evaluates function calls, it analyses following cases: if the transputer has "successors", if its "successors" are free, and if expression is user-defined function, then it sends a function to be evaluatet to the first free "successor". In other case it itself evaluates a function call;

d) it sends the message "GIVE ME" to a "successor", demanding the value it computed. Then it waits until it receives the value.

2.1.3 The segment intended for other transputers.

On the other transputers some procedures are disposed as unnecessary. Some procedures are new.

In the part Argument passing new procedures are procedures intended for the communication stack (not necessary for the first transputer).

In the part Control parts there are several operations to be performed:

a) receival of function intended for evaluation (and all necessary data) from the "parent".

b) receival of the message "GIVE ME", from the "parent";

c) receival of the message "END" from the "parent". This message means the end of the interpreter work. In that moment, execution of program ends, and the user exits from the interpreter to the operating system.

3. The efficiency of the implementation

This implementation is efficient, in case of the great number of operations, and recursive oriented solutions. However, increase in speed depends upon the nature of a problem.

The testing was performed using few test examples. The increase in speedup was notable only for problems with a small number of I/O operations, and a great number of computing operations. In the alternate case (a great number of I/O operations) the increase in speed is small, because the communication time for one datum is 4 times greater than the time needed for the arithmetic operation on that datum.

In Tables 1-3 all times are given in ms. The maximal error of measurement equals 5ms.

The results for different arguments are given in different rows of each table.

In each row are given:

a) arguments of functions;

b) the execution time on 1 transputer;

c) the execution time on configuration with 3 transputers, and increase in speedup in comparison to time on 1 transputer;

d) the execution time on configuration with 7 transputers, and increase in speed compared to the time on 1 transputer.

e) the execution time on configuration with 15 transputers, and increase in speed compared to the time on 1 transputer;

f) the execution time on configuration with 17 transputers, and increase in speed compared to the time on 1 transputer;

Example 1: The function with 2 recursive calls:

```
(defun t2 ( x )
   (if (= x 0)
      1
      ( + (t2 (- x 1)) (t2 (- x 1)))))
```

The method of evaluation of (t2 17) is given in Fig. 2.

Example 2: The recursive search of Fibonacci numbers:

```
(defun fib ( x )
   (if (¡ x 2)
      x
      (+ (fib (- x 1)) (fib (- x 2)))))
```

The method of evaluation of (fib 24) is presented in Fig. 3.

TABLE 1. Times for the Example 1

X	1 tr.	3 tr.	spe.	7 tr.	spe.	15	spe.	17	spe.
10	741	428	1.73	232	3.19	125	5.92	125	5.92
11	1481	850	1.74	457	3.24	237	6.24	237	6.24
12	2961	1695	1.74	906	3.26	461	6.42	461	6.42
13	5921	3384	1.75	1804	3.28	911	6.49	910	6.50
14	11841	6764	1.75	3541	3.34	1809	6.54	1808	6.54
15	23681	13522	1.75	7073	3.34	3605	6.56	3604	6.5
16	47362	27039	1.75	14138	3.35	7197	6.58	7197	6.58
17	94722	54073	1.75	28267	3.35	14382	6.58	14382	6.58

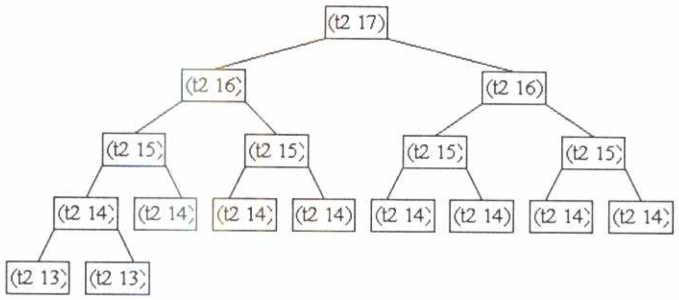


FIGURE 2. Scheme of evaluation for example 1

TABLE 2. Times for the Example 2

X	1 tr.	3 tr.	spe.	7 tr.	spe.	15	spe.	17	spe.
10	65	49	1.32	36	1.80	26	2.5	22	2.95
15	710	502	1.41	337	2.10	201	3.53	138	5.14
16	1148	809	1.41	543	2.11	317	3.62	216	5.31
17	1857	1306	1.42	874	2.12	506	3.67	343	5.41
18	3004	2111	1.42	1411	2.12	826	3.63	548	5.48
19	4860	3412	1.42	2279	2.13	1310	3.71	879	5.52
20	7862	5519	1.42	3684	2.13	2115	3.71	1416	5.55
21	12721	8927	1.42	5957	2.13	3418	3.72	2284	5.57
22	20582	14433	1.42	9476	2.17	5523	3.72	3688	5.58
23	33301	23353	1.42	15311	2.17	8932	3.72	5961	5.58
24	53881	37758	1.42	24761	2.17	14453	3.72	9639	5.59

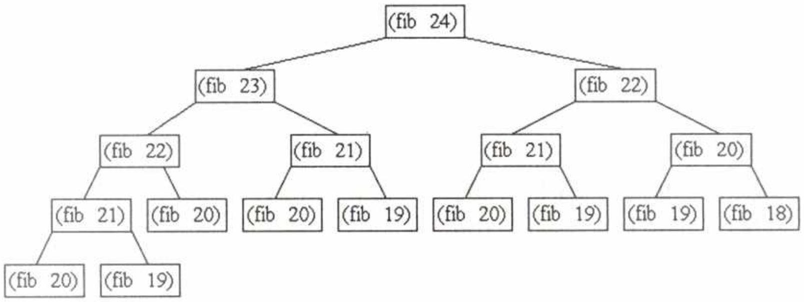


FIGURE 3. Scheme of evaluation for example 2

Example 3: Some problems have a great number of I/O operations, a lot of communication, or their execution is slow due to certain technical limitations of transputers. In the evaluation of those problems a small increase in speed is obtained, or, conversely, more time is needed than on one transputer. The example of such a problem is the program which forms a "big" list.

```
(define form (n)
( if (= n 0) (set list (cons '1 list))
   (begin
   (form (- n 1))
   (form (- n 1))
   (form (- n 1))
   (form (- n 1))
   (form (- n 1)))))
(set list '())
```

TABLE 3. Times for the Example 3

X	1	3	spe.	7	spe.	15	spe.	17	spe.
2	53	63	0.84	63	0.84	63	0.84	63	0.84
3	293	344	0.85	344	0.85	344	0.85	344	0.85
4	1461	1721	0.84	1721	0.84	1721	0.84	1721	0.84
5	7338	8633	0.85	8633	0.85	8633	0.85	8633	0.85

Remaining methods of implementation are extensively described in [1]. Some of them are:

1. Translation of a program code into metalanguage, that is more suitable for evaluation [6];

2. Division of the problem into subproblems (divide and conquer approach) [5];

3. Translation of the program into a code that does not the requirements of speed, and after that, during run-time, automatic improvement of its performances [4].

4. Conclusions

Most implicit parallel languages implement functional programming languages. Reasons for using functional paradigm (insted of the procedural one) are:

1. Smaller kernel of language;
2. A precise grammar, and, consequently, uniform constructions;
3. No side effects;
4. Easy writing of recursive functions;
5. No explicit sequence of execution.

Because of that, parallel implicit programming languages are most popular.

In this paper the interpreter for LISP that implicitly solves problems of communication and synchronization between processors was developed. This method is the most general one, but it does not, in the same time, produce the fastest code. The code is equal to the code used for uniprocessor machines, and all programs written in sequential LISP can operate on those machines as well. But a programmer can manually write the fastest code (in explicit parallel programming languages, like Parallel C or Occam).

Architecture is binary tree. This means easier control of processors (communication and synchronization), but it also means the unnecessary waiting of some processors (transputers). A more complex architecture (than the tree) can reduce waiting of processors, but it will also increase a communication.

The methods of improving this implementation are:

1. the implementation of new built-in functions in accordance with the Common LISP standard. It should be noted that there are few thousand of built-in functions in Common LISP;

2. using more complex architecture of the transputer system. New generation of transputers has more interprocessor channels (16) than this generation (4). This means that architecture can be more complex, and the increase in speed can be greater.

References

[1] ASHCROFT E.A., FAUSTINI A.A., JAGANNATHAN R., *An Intensional Language for Parallel Aplication Programming*, Parallel Functional Languages and Compilers, ACM Press, 1991, pp. 11 – 50.

[2] KAMIN N. S., *Programming languages - An interpreter based approach*, Addison-Wesley, 1990.

[3] KRATICA J., *Paralelization of functional programming languages and implementation to transputer systems*, Mag. thesis, University of Belgrade, Faculty of Mathematics, 1994.

[4] LEUNG S., ZAHORJAN J., *Improving the Performance of Runtime Paralelization*, Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (1993), ACM Press, 208–217.

[5] MOU Z.G., *A Formal Model for Divide-and-Conquer and its Parallel Realization*, Ph.D. thesis, Yale University, Department of Computer Science, 1990.

[6] SKEDZIELEVSKI S.K., GLAUERT J., *IF1 - An intermediate form for aplicative languages*, Manual M-170, Lawrence Livermore National Labaratory, 1985.

[7] *Transputer Toolset*, Inmos corp., 1989.

27. MARTA 80, 11000 BELGRADE

FILOMAT (Niš) 9:2 (1995), 377–384

Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

CODING FOR (5,13) CHANNEL CONSTRAINTS

Milan Simić and Rade Petrović

ABSTRACT. *Data Translation codes for the particular channel constraints are designed and presented in this paper. The encoding schemes belongs to the $RLL(5,k)$ codes family and can be used in digital recording and telecommunication practise.*

1. Introduction

Runlength limited codes, RLL, are used for digital storage, or as Translation codes for digital data transmission /1/. System for digital transmission can be defined as a system designed to best use a given channel, while the analog communication system is defined as the one designed to best fit a given signal source. Unconstrained data stream must be converted to constrained stream of symbols, (d,k), in order to solve the problems of spectral shaping, self-timing, and intersymbol interference (ISI). Lower bound of zero runs defined by parameter d is used to control ISI, while upper bound defined by parameter k is used to insure data stream selfclocking. Generally, parameters of any translation encoding scheme belong to the range of $0 \leq d < k \leq \infty$. Through the numberof already published papers, we have shown that channels with constraints in the range of $d = 5$ and $12 \leq k \leq 16$, are interesting for the future use in both areas of application. The channel $(5,13)$ have not been especially treated yet, and it is the purpose of this paper. The presented encoding schemes offer a great opportunities in choosing encoding rules, so that RLL codes can be combined with permutation codes, and the signal spectral density can be adjusted. Permutation codes are a class of error correction codes which have been suggested for use on the Gaussian channel.

2. Capacity and Coding Rate Consideration

Based on Schannon's FSM channel model (Finite State Machine), general algorithms are already developed for practical encoding schemes design.

Code designers try to enlarge the parameter d, and to shorten the parameter k, for the same coding rate $R = m/n$, defined as ratio of unconstrained input data symbols number, m, to the number of constrained signal symbols, n, at the coder output. The operation of string translators, named encoders and decoders, is to map input string of symbols from one alphabet into the output string of symbols from the other alphabet. Input string may possibly be of infinite length, but for the practical reasons it is devised into finite strings of fixed or variable lengths, so that we have FL or VL encoding schemes.

Codeword assignment is obviously the function of the incoming dataword, but also it can be dependent on the channel state, presented by the FSM, when we have state dependent coding. Finally, it can be the function of the future dataword, and in that case we have Future dependent coding - FD. Shannon proved that, as codeword length n grows, the number $N(n)$ of (d,k) sequences approaches the value 2^{Cn}, where C is called information capacity. Capacity C can be treated as the theoretically maximum achievable coding rate R for infinite value of codeword length n, according to the equation:

$$C = \lim_{n\to\infty} \frac{1}{n}(log_2 N(n)) \tag{1}$$

It is clear that coding rate R always satisfies inequality $R < C$. The code is called the efficient one if the coding rate R is close to the capacity C. Different (d,k) sequences information capacities are already given in the references, but we did some more calculations based on the solution of the characteristic equation given by $det(A - \lambda I) = O$, where A is FSM state-transition matrix. The calculated capacity C is the capacity of a discrete noiseless channel expressed in units of **bits per channel symbol**, although it can also be calculated in units of **bits per second, bps**. Capacity of the channel in bits per channel symbol differ only by a factor equal to the number of channel symbols per second. Considering channel characteristics there are four concepts related to one another:

Data rate in bps, at which data can be communicated,

Bandwidth of the transmitted signal and the nature of the transmission medium in hertz,

Noise or average level of noise over the communications path,

Error rate, the rate at which errors occur.

For the coding purposes, or alphabet conversion, it is convenient to consider capacity in bits per symbol. Our conclusion was that the codes with coding rate $R = 1/3$ and parameters $(d,k) = (5,k)$ could be of interest in recording, as well as telecommunication practice. Density ratio of these codes, given by $DR = R(d+1) = 2$, is valuable improvement over existing codes.

After the considerations described above, we have found out that it is possible to design new coding schemes defined by parameters $(d, k) = (5, 13)$ and $R = 1/3$, as the $(5, 13)$ constrained sequences information capacity is $C = 0.343 < 1/3 = R$. Clock rate, defined as $CLR = 1/RT$, is increased, $CLR = 3$, and thus compensates information rate loss caused by translation of unconstrained input data sequences to the constrained sequences. The original information bit time interval, which corresponds to NRZ clock signal, is called bit window, and is labelled with T.

3. Encoding Schemes for (5,13) Constraints

- *State and Future Dependent Coding*

Figure.1 illustrates State transition diagram or FSM, for general constraints (d, k). In our particular case when $d = 5$ and $k = 13$, it is a graph with 14 nodes, or channel states, where arrows directed edges represents state transitions, and are labelled with channel bits. In the terminology of synchronous Bounded Delay (BD), or FD coding /2-5/, set S_c is a set of coding, or terminal states, which are the states entered at the end of codewords. Codewords are the paths through the FSM graph.

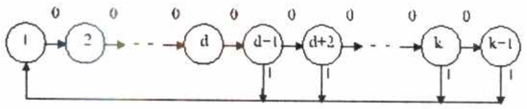


Fig. 1 State transition diagram for the (d, k) sequence

The existence of set $S_c = (S_c)$, as a subset of all FSM states set, $S = (S_i), i = 1, ..., 14$, is a necessary and sufficient condition for the existence of a code. FD RLL(5,13) code can be defined with 26 codewords, the lengths of which vary from 3 to 21 signal symbols, representing 1 to 7 data bits. Coding states set is $Sc = (S_c), c = 1, 2, 4, 5, 6, 7, 8$. The codeword choice is a function of the current state (State Dependent-SD), the information to be represented, and the future information. Code conversion rules are given in Table 1. Data bits in brackets indicate future bits in certain states, and are related to the states S_1, S_2, S_4, and S_5, where we have future dependency.

TABLE 1. Future Dependent RLL(5,13) Code

	Initial State	Input Data	Output Sequences	Final State
1	S_6, S_7, S_8	00	100000	S_6
2		010	010000000	S_8
3		011	001000000	S_7
4		100	000100000	S_6
5		1010	000010000000	S_8
6		1011	000001000000	S_7
7		1100	000000100000	S_6
8		11010	010000010000000	S_8
9		11011	010000001000000	S_7
10		11100	001000001000000	S_7
11		111010	000010000010000000	S_8
12		111011	000010000001000000	S_7
13		111100	000001000001000000	S_7
14		1111010	010000010000010000000	S_8
15		1111011	010000010000001000000	S_7
16		1111100	010000001000001000000	S_7
17		1111101	001000001000001000000	S_7
18		1111110(0)	000010000010000010000	S_5
19		1111110(1)	000010000010000001000	S_4
20		1111111(1)	000001000001000001000	S_4
21		1111111(00)	010000010000010000010	S_2
22		1111111(01)	010000010000010000001	S_1
23	S_1	0	000	S_4
24	S_2	0	000	S_5
25	S_4	1	000	S_7
26	S_5	0	000	S_8

- *State Independent Coding*

Using the same codepaths in the state transition diagram from Fig.1, state independent SI RLL(5,13) code can be defined with no look-ahead. In this case, coding states set is $S_c = (S_c), c = 6, 7, 8$. Code translation table can consists of 22 or 21 codewords, the lengths of which vary from 6 to 24 signal symbols, representing 2 to 8 data bits. Generally, future dependency can shorten codeword length, but in this case it does not affect the error propagation limiting (EPL), or codec complexity reduction. The next RLL(5,13) code is designed with given coding states set S_c, where the translation rules are defined for 21 VL codewords as presented in the Table2.

TABLE 2. State Independent RLL(5,13) Code

	Input Data	Output Sequences
1	00	100000
2	010	010000000
3	111	001000000
4	100	000100000
5	1010	000010000000
6	0111	000001000000
7	1011	000000100000
8	11001	010000010000000
9	11011	010000001000000
10	11000	001000001000000
11	011001	000010000010000000
12	011011	000010000001000000
13	011000	000001000001000000
14	1101001	010000010000010000000
15	1101011	010000010000001000000
16	1101000	010000001000001000000
17	1101010	001000001000001000000
18	01101001	000010000010000010000000
19	01101011	000010000010000001000000
20	01101000	000010000001000001000000
21	01101010	000001000001000001000000

The problem of EPL is directly related to the appropriate codeword to the data word assignment and the decoder design /6/. Based of that, similar data sequences are coded with similar symbol sequences. The encoder for the New code can be designed as any PAL SM (State Machine) encoder for RLL codes, as for example for RLL(2,7), or RLL(5,16) codes, but we propose sliding window decoder.

The decoder for New RLL(5,13) code has 26 bits shift register, as sliding window, with serial input-parallel output and PLA array architecture for combinatorial logic design. Programmable AND-OR array generates canonical form sum-of-products of the variables involved in a function. Variables are taken from the shift register positions. Sequential decoder generates one output data bit for each incoming 3 symbol bit pattern, after the time delay for 9 symbols, or 3 data bits. There is no internal feedback in the decoder as the output depends only on the input string in length of 26 bits, so that the error propagation is limited to only 9 data bits since $24 < 26 < 27$. If we denote the contents of the shift register positions by $x_i, i = 1, 2, ..., 26$, with shifts from x_1 to x_{26}, the decoded group occupies positions x_{10}, x_{11} and x_{12}, while the past $(x_i, i = 13-26)$, as well as, the future $(x_i, i = 1-9)$ bit groups, can affect the decoder decision. After a great deal of calculation, since the truth table has 116 rows and 26 columns, it can be shown that the decoder output, $d = f(x_1, ..., x_{26})$, is defined by Boolean expression:

$$d = \sum_{i=1}^{18} p_i$$

(2)
$$\begin{gathered}
p_1 = x_7x_{13};\ p_2 = x_7x_{14};\ p_3 = x_8x_{14};\ p_4 = x_9x_{18};\ p_5 = x_9x_{19}; \\
p_6 = x_9x_{20};\ p_7 = x_{13}\bar{x}_{19};\ p_8 = x_{13}x_{25};\ p_9 = x_{14}\bar{x}_{20};\ p_{10} = x_4x_{11}\bar{x}_{17}; \\
p_{11} = x_5x_{11}\bar{x}_{17};\ p_{12} = x_{10}\bar{x}_{16}\bar{x}_{17};\ p_{13} = \bar{x}_{10}x_{16}\bar{x}_{22};\ p_{14} = \bar{x}_{10}\bar{x}_{11}x_{17}x_{23}; \\
p_{15} = \bar{x}_1\bar{x}_2x_8\bar{x}_{15}\bar{x}_{16}\bar{x}_{17};\ p_{16} = x_6\bar{x}_{12}\bar{x}_{13}\bar{x}_{14}\bar{x}_{15}\bar{x}_{16}\bar{x}_{17}; \\
p_{17} = x_{12}\bar{x}_{18}\bar{x}_{19}\bar{x}_{20}\bar{x}_{21}\bar{x}_{22}\bar{x}_{23};\ p_{18} = x_{15}\bar{x}_{21}\bar{x}_{22}\bar{x}_{23}\bar{x}_{24}\bar{x}_{25}\bar{x}_{26};
\end{gathered}$$

The truth table used in the evaluation of decoder function is only a part of the whole table with $2^{26} = 67108864$ rows.

- *Improved ACH coding*

Referring to the FSM model, as in any other method, with ACH approach it is possible to derive encoder state transition table in systematic manner, for any channel constraints, if coding is realisable depending on R and C. Recently /7/ the novel method, or improved ACH, was presented. The same approach was used for the following scheme design. Constrained channel is described by 14-by-14 state transition matrix D:

$$\begin{aligned}
&D = (d_{ij}); \quad i, j = 1, ..., 14 \\
&d_{i1} = 1 \quad \text{for} \quad i \geq (d+1) = 6 \\
&d_{ij} = 1 \quad \text{for} \quad j = i+1 \\
&d_{ij} = 0 \quad \text{for the other cases}
\end{aligned}$$

Next step was to derive $B = D^3$ from D, and it should be for our channel the following matrix :

$$\begin{aligned}
&B = D^3 = (b_{ij}); \quad i, j = 1, ..., 14 \\
&b_{i1} = 1 \quad \text{for} \quad 4 \leq i \leq 12 \\
&b_{i2} = 1 \quad \text{for} \quad 5 \leq i \leq 13 \\
&b_{i3} = 1 \quad \text{for} \quad 6 \leq i \leq 14 \\
&b_{ij} = 1 \quad \text{for} \quad j = i+3 \\
&b_{ij} = 0 \quad \text{for the other cases}
\end{aligned}$$

After that we have found vector V with positive integer components such that:

$$D^3V \geq 2V$$

Two optimal, from many possible solutions, are the following:

V1 transposed = (4 5 6 8 10 12 12 11 9 9 7 3 3 3)

$m = v(i)$ for $i = 1$ to $14; m = 102$

V2 transposed = (4 5 6 8 10 12 12 11 9 9 7 3 3 1)

$m = v(i)$ for $i = 1$ to $14; m = 100$

where $v(i)$ are components of the vector V.

The number of encoder states, in the state splitting process, is given by the corresponding component of vector V so that the total number of encoder and decoder states is m. The Encoder matrix E is squared, 100x100 matrix in the other case which is the best one, and we can use Milan approach, with H matrix, to define the encoder /7/.

Since the number of states is $100 \leq 128 = 2^7$ states, the error propagation is limited to only 7 data bits, for this class of codes /7/.

4. Conclusion

Encoding schemes presented in this article are coding problem solutions for (5,13) channel constraints. Error propagation is limited in each case, as more precisely presented in the previous papers for similar codes, and the further analyse can be done in order to adjust signal spectres. In addition to that, RLL codes can be combined with permutation codes to improve the reliability, or Data Rate in the communication channel.

The last one scheme from this paper, can give us more freedom to make the appropriate codewords to datawords choice. Since the presented channel codes are selfclocking, and according to the existing standards they can be used for voice and all other source data transfer, so they are suitable to be data encoding schemes for ISDN, or BISDN via fibre optic media. Finally, the FDDI code is only RLL(0,3) encoding scheme.

References

[1] Richard, E.B., *Digital transmission of information*, Addison Wesley Publishing Company, 1990.

[2] Adler, R.L., Coppersmith, D., Hassner, M., *Algorithms for Sliding Block Codes*, IEEE Trans.Inf. Theory **IT-29, No. 1, January** (1983), 5–22.

[3] Simić, M., Petrović, R., *New RLL code for digital data storage*, Electron. Lett. **25, (15)** (1989), 951–954.

[4] Franaszek, P.A., *Synchronous bounded delay coding for input restricted channels*, IBM J. Res. and Develop. **24, No.1** (January 1980,), 43–48.

[5] Franaszek, P.A., *A general method for channel coding*, IBM J. Res. Develop. **24, No.5** (September 1980), 638–641.
[6] Simić, M., Petrović, R., *New EPL RLL(5,16) code*, Electron.Lett. **27, (23)** (1991), 2100–2102.
[7] Simić, M., *RLL(5,12) Coding*, ETRAN, Niš 1994.

Milan Simić, Faculty of Philosophy, Department of Mathematics, University of Niš, Yugoslavia

Rade Petrović, University of Mississippi, Centre for Telecommunication, PO Box 9031, University, MS, 38677

FILOMAT

FILOMAT is a continuation of *Zbornik radova Filozofskog fakulteta u Nišu, Serija Matematika (vol. 1(1987)- vol. 6(1992)* and is published yearly. It publishes original papers in all fields of pure and applied mathematics.

Instructions for authors

All manuscripts (the original and a copy) must be written in English. The extent of the papers is limited to ten pages in length; manuscripts over 10 pages are accepted only exceptionally. Manuscripts should not require many language corrections.

FILOMAT is produced using $\mathcal{AMS}$-TEX. Authors are encouraged to prepare their manuscripts using $\mathcal{AMS}$-TEX. Only a hard copy should be submitted for assessment, but if the paper is accepted the author will be asked to send the text on an IBM PC compatible diskette.

The author(s) should write their names, addresses and the title of the paper on a separate sheet. All manuscripts should start with a short Abstract and include the footnote 1991 Mathematics Subject Classification on the first page. Definitions, theorems, lemmas, remarks, proofs etc. should be written using only one of two alternative styles: **Definition** 2.1. or 3.2. **Theorem** consistently throughout the paper.

Figures are included in the text and must be numbered (by arabic numbers) and mentioned in the text. Equations (or formulas) must be numbered (for future references) in parentheses ( ) at the left margin.

References should be listed alphabetically in the following form:

[3] E.Hewitt and K.A.Ross, *Abstract Harmonic Analysis*, Vol. I, Springer-Verlag, Berlin, 1963.

[11] Đ.Kurepa, *On regressive functions*, Z. Math. Logik 4 (1958), 148-156.

[15] P.Petrović, *Neka svojstva ...*, Doctoral dissertation, University of Belgrade, 1980.

A total of 30 reprints of each paper will be available free of charge; additional reprints can be ordered.

All correspondence concerning both manuscripts and exchange should be addressed to:

Filozofski fakultet (FILOMAT), ul. Ćirila i Metodija 2, 18000 Niš, Yugoslavia.

The subscription price is 20 dollars USA per volume, post free.

For subscriptions write to the same address.

CONTENTS

ISSN 0354–5180