1.12 — metody numeryczne i komputerowe

Jacek Mączyński

# PAPERS ON TECHNIQUES OF PROGRAM DESIGN

5/1983

P. 269

WARSZAWA 1983

ISSN 0208-5658

Praca wpłynęła do Redakcji dnia 25 listopada 1982 r.

57031

N a   p r a w a c h   r ę k o p i s u

Instytut Podstawowych Problemów Techniki PAN

Nakład 150 egz. Ark.wyd. 1,8. Ark.druk. 2,5.
Oddano do drukarni w lutym 1983 r.
Nr zamówienia 188/83   M-24.

Warszawska Drukarnia Naukowa, Warszawa,
ul.Śniadeckich 8

Jacek Mączyński
Pracownia Obliczeń Numerycznych
Computing Laboratory
IPPT PAN - IFTR PASc.

## C o n t e n t s

## Z a w a r t o ś ć

P. 269

## С о д е р ж а н и е

Jacek Mączyński
Computing Laboratory

## Papers on techniques of program design

### F o r e w o r d

The group of papers presented in this volume is an account of such methods as the concatenation algebra, the Task Description Language or a straightforward approach to Virtual Memory. As an additional feature a numerical discussion of an integro-differential equation is given.

The techniques mentioned above have been in use for some time in the Computing Laboratory and proved to be of advantage when streamlining the process of design and implementation of programs involving many thousands of diversified matrix entries resulting from analytical integration of eigenfunctions and their products. Therefore the designing techniques themselves seem to have some intrinsic value.

The language of the presentation is English, because prospective readers /at least some of them/ will prefer poor English to Polish good or bad.

Warszawa, November 1982

Jacek Mączyński
Pracownia Obliczeń Numerycznych

## Ogólne zasady testowania pakietu programowego Tolos

Testowania programów są nierozłącznie związane z ich strukturą. Wynika stąd rola języka opisu zadań programowych /TDL/, a także formalizacji rachunku konkatenacyjnego dla operacji macierzowych. Język TDL jest zbliżony w pewnym sensie do języka LISP, lecz różni się od niego nietylko ortografią, lecz również wymogiem wyraźnego definiowania zbiorów, dla których powtarza się wykonanie segmentów programowych. W zwykłej praktyce programowania stosuje się operację powtórzenia, a zbiory są pośrednim wynikiem tej operacji.

Konkatenacja jednowymiarowa wektorów i dwuwymiarowa macierzy daje się /przy dość ogólnych założeniach/ sformalizować. Dzięki temu tworzenie makromacierzy z macierzy-składników przybiera postać rachunku, nie wymagającego odwoływania się do pomocy rysunków, lub nieraz zawodnej wyobraźni.

Testowanie gotowych programów korzysta z ich struktury i dodatkowo z faktu, że realizują one odwzorowania pomiędzy abstrakcyjnymi przestrzeniami funkcyjnymi.

Яцек Мончинский

Лаборатория Вычислительной Техники

## Общие Принципы Проверки Програмного Пакета ТОЛОС

Проверка програм неразделимо связана с их структурой откуда вытекает роль введенного языка описания програмных задач (ТДЛ - ЯОЗ) и формального подхода к конкатенации (сцеплении) способствующего необходимым матричным вычислениям. Язык ТДЛ близок подмножеству языка ЛИСП, однако отличается от него более чем просто правописанием, именно главным образом, требованием явного определения множеств для которых програмные отрывки повт ряются. В этом отношению обычные програмные языки пользуются инструкцией повторения.

Сцепление матричных векторов и блоков подвергается формальной трактировке приобретая вид исчисления, что надежнее чем рисунки или просто воображение.

Проверка изготовленых програм глубоко использует их структуру и вообще руководствуется тем свойством что програмы - по сути дела - отображения между абстрактными функциональными пространствами.

Jacek Mączyński
Computing Laboratory

# A TEST - BENCH PHILOSOPHY FOR PROGRAM MODULES OF THE THOLOS PACKAGE

## 1. Introduction

The THOLOS package is an implementation of the combined strutt, curved beam and shell theory, for computation of displacements and forces in a prestressed concrete version of a nuclear reactor secondary containment building. The package consists of number of program modules designed to perform mappings from one abstract space to another say, from a load space to an internal force space or to a displacement space. In the process of program design, requirements for the modules had to be specified down to minor details. The formulae had to be put into a form suitable for rapid computation, and this obviously tends to obfuscate the program text, so as to make it, at times, difficult to read and interpret. Again, the shear bulk of the programs necessitated some conceptually guided organizing effort. To cope with this situation which might have easily resulted in undetected, or visually undetectable errors, special methods for program design and testing had to be devised and their philosophy is described below in some detail.

First, a Task Description Language is introduced and explained. Its purpose is mainly to break up a major task into its constituents in successive /or repeated/ steps, so as to produce directly programmable entities. Matrix formulae when dealing with subblocks of various products of matrices or concatenated vectors, are best handled by introducing some formal concatenation operations. Thus, instead of making multiple references to drawings and charts an automatic concatenation is formulated and used. Finally, the written programs have to be tested and debugged. Therefore a two - method testing app ch is used in most cases together with ample use of orthogonal data vectors. Applicability of the methods is wide, particularly with reference to the abstract spaces used. Account must be taken of the properties of the norms which, however, in our case, present no difficulty.

## 2. A Task Description Language /TDL/

The language for task description is a formal metalanguage and bears some structural similarity to the LISP language /cf. WINSTON [1977]; MARTINEK [1980]/.

This similarity stops when intelligibility would be impaired too strongly and therefore texts in LISP and TDL do not look very much alike at the surface. The TDL language has a formal grammar, its definitions /or "productions" in BNF/ are given below. The main idea of TDL consists in maintaining the awareness of a hierarchy of tasks. There is the obvious limitation that names have to denote unique tasks within a single descriptive text.

A task is described in TDL by

/2.1/ ⟨task⟩ ::= ⟨name⟩ (⟨task list⟩); | ⟨predicate (⟨task list⟩); | ⟨name⟩

A name is highly arbitrary, although it is always advisable to use abreviations which bear some relation to the task involved, human first names and other unrelated identificators should better be avoided.°
Next we write

/2.2./ ⟨task list⟩ ::= ⟨task⟩ | '⟨segment⟩' | ⟨task list⟩, ⟨task list⟩

We notice that TDL uses a comma for separating tasks and differs from LISP in this respect.

Further it may at times be useful to define an

/2.3/ ⟨instruction⟩ ::= '⟨ segment ⟩' | ⟨task⟩

A ⟨segment⟩ is usually identifiable with a program segment in some suitable programming language. Since it usually contains names which are not and may not be task identifiers use of string parantheses ' solves the ambiguity. Evidently a ⟨segment⟩ is an "atom" of TDL. No further formal description, other than a possible verbal comment, is needed so it is a terminal symbol of the language. Obviously TDL is wholly unconcerned with semantics of the segmants used.

A ⟨predicate⟩ has 4 concatenated constituents:

/2.4/ ⟨predicate⟩ ::= ⟨quantifier⟩ ⟨argument⟩ ⟨relation operator⟩ ⟨argument⟩ | ⟨compound predicate⟩

where reasonable predicates are formed by careful picking the constituents from the definitiors :

/2.5/ <quantifier> ::= $\wedge$ | $\vee$
/2.6/ <relation> ::= $\in$ | $<$ | $\leq$ | $=$ | $\geq$ | $>$
/2.7/ <argument> ::= <variable> | <set identifier>
/2.8/ <set identifier> ::= <name> | [<element list>]

When TDL is used for program design and description a predicate bears some resemblance to its counterpart from the predicate calculus. The main difference is the same as in any programming language construct - it is more than a statement of fact, it is an action. We have further an obvious definition of a <compound predicate> :

/2.9/ <compound predicate> ::= <predicate> | void |
<predicate> <logical operator> <predicate> | not
<predicate>

where the binary logical operator is:

/2.10/ <logical operator> ::= and | or | ... &c.

There is no necessity to confine the set of logical operators to those which are implemented on some real computer, at least when TDL is used for conveying ideas between human beings.

The use of predicates is perhaps closer to the usual mathematical habit than LISP constructions. By way of an example, a text in LISP:

```
    LOOP (COND ((ZEROP N) (RETURN ACTION)
    SETQ N (SUB1 N))
   (GO LOOP)
```

becomes somewhat more natural in TDL:

$\wedge$ N $\in$ {N1, (N>0)} ('N:=N-1;' ACTION) :

A similar construct:

$\wedge$ N $\in$ SN (ACTION)

is self - explaining with N being a name of a <variable>, SN a <set identifier>, ACTION a <name> which is a task name.

A characteristic feature of TDL is a closer conceptual approach to conditional and cyclic instructions. A strong quantifier $\wedge$ is used for cyclic instructions which may be read: "perform the stated action for all values of the control variable from the given set".
Cautious consideration of the sets involved is essential to an early elimination of possible errors, and perhaps a better habit then just a REPEAT concept. Conversely, a weak quantifier is used for conditional statements since what is meant is : "perform once if at least one value of

the control variable is in the given set". Ordered sets are denoted in mathematical fashion, say $[1, \ldots, n]$ and $N > n$ stops execution.

A text in TDL defines the task when elimination of names produces a sequence of segments interpersed with predicates. Implementation of actual programs is not excessively restricted by a task description. GO TO statements may possibly be used, where applicable, for skipping certain program actions but wherever possible constructs: $\bigvee p$ /ACTION/ which do not impair program structure and are semantically equivalent are preferable. It is widely accepted that program efficient operation precludes recursion. When designing a program there is no need early to decide whether subroutines have to be used or a homogeneous statement sequence is semantically right. A repetition of the same task name is conducive to subroutine implementation in many cases. A program description in TDL is shown in Appendix A.
A loop with an exit, a very frequent and fundamental programming construct, has a slightly artificial TDL representation. Namely, what is programmed with two well located jumps and two labels :

```
 label1 :
<segment 1;>
 if condition then go to label2;
<segment 2;>
 go to label1;
 label2:
 ...,
```

becomes in TDL :

$\bigwedge k \in [1, 2, \ldots, \text{upper}]$ ( segment 1, $\bigvee$condition ( 'upper : = 0'),
$\bigvee$ not condition (segment 2))
It is subsumed that a break in an ordered set stops execution /this is true for algol <u>for</u>, <u>step</u>, <u>until</u> constructs/. Avoidance of a special <u>while</u> construct is deliberate since TDL is not just another form of structured programming /cf. DIJKSTRA/. It mainly differs by its emphasis on formal language properties. Again, TDL is conceived as a means of communication between those who state problems and those who actually implement solutions.

The particular field of application is in the interface between theoretical mechanics /and other branches of mathematical physics/ and computer programs, i.e. in these domains where algorithms arise naturally from formula manipulation, however extensive, rather than in those where numerical processes have to be conceived by skillful handling of operations on data and states, as in the design of operating systems /executors/. At least, the impulse comes from the former computer applications and an extension of the latter problems of computer science is conceivable although further investigations are needed if applicability is to be decided upon. Nevertheless, the very possibility of a computer implementation of the TDL language appears to be promising.

## 3. Concatenation algebra

Consider two concatenation operators $\ominus$, $\phi$ . The first "$\ominus$" serves to create horizontal concatenations or rows of elements which may be single entries, vectors /in matrix sense/ or compatible blocks. If concatenated parts are compatible we say that the concatenate exists and we write:

/3.1/ $\quad a \ominus b \qquad \overset{n}{\underset{i=1}{\ominus}} a_i$

The other "$\phi$" produces columns of elements, again

/3.2/ $\quad a \phi b \qquad \overset{n}{\underset{i=1}{\phi}} a_i$ are columns

of compatible blocks or single entries. The requirement of compatibility is to be understood in the sense of matrix definition i.e. the resulting concatenate is a matrix containing rows of equal length.

A matrix A of entries $\{a_{ij}\}$, i=1,2,...,n, j = 1,2, ... , m may be simply defined by

/3.3/...

$$A = \overset{n}{\underset{i=1}{\phi}} \overset{m}{\underset{j=1}{\ominus}} a_{ij}, \qquad A^T = \overset{n}{\underset{i=1}{\ominus}} \overset{m}{\underset{j=1}{\phi}} a_{ij}$$

and

/3.4/ $\quad \overset{n}{\underset{i=1}{\ominus}} \overset{m}{\underset{j=1}{\ominus}} a_{ij}$

is a linear vector of lexicographically arranged entries of A.

The concatenation operators satisfy the following axioms :

1. they commute with multiplication by a real or complex number $\lambda$ :

$$\lambda \overset{n}{\underset{i=1}{\ominus}} a_i = \overset{n}{\underset{i=1}{\ominus}} \lambda a_i, \qquad \lambda \overset{n}{\underset{i=1}{\phi}} a_i = \overset{n}{\underset{i=1}{\phi}} \lambda a_i.$$

2. They are asymetric:

$$a \phi b \neq b \phi a, \qquad a \ominus b \neq b \ominus a.$$

3. They are separately associative:

$$a \ominus b \ominus c = (a \ominus b) \ominus c = a \ominus (b \ominus c)$$

$$a \phi b \ \phi c = (a \phi b) \ \phi c = a \phi (b \phi c)$$

but

$$(a \ominus b) \ \phi \ = \ a \ominus (b \phi c)$$

4. No zero element and no inverse operation is defined for $\ominus$ or $\phi$ . /Although there are theoretical means to do so cf. MAGGIOLO-SCHETTINI/

5. If /for the sake of clarity/ the sign X is used to denote matrix multiplication, then

$$\underset{i=1}{\overset{n}{\ominus}} a_i \times \underset{j=1}{\overset{n}{\phi}} b_j = \sum_{i=1}^{n} (a_i \times b_j)$$

where all the product terms $a_i \times b_j$ must exist and be matrix summable.

6. Conversely

$$\underset{i=1}{\overset{n}{\phi}} a_i \times \underset{j=1}{\overset{m}{\ominus}} b_j = \underset{i=1}{\overset{n}{\phi}} (\underset{j=1}{\overset{m}{\ominus}} a_i \times b_j)$$

where existence and compatibility of the products with regards to $a_i \times b_j$ is subsumed.

7. Left and right matrix multiplication is associative with horizontal and respectively vertical concatenation :

$$(\underset{i=1}{\overset{n}{\phi}} a_i) \times b = \underset{i=1}{\overset{n}{\phi}} (a_i \times b),$$

$$b \times \underset{i=1}{\overset{n}{\ominus}} a_i = \underset{i=1}{\overset{n}{\ominus}} (b \times a_i)$$

where $a_i$'s are compatible with respect to the respective concatenations.

8. Horizontal and vertical concatenation is commutative since we may write

$$\underset{i}{\phi} \ \underset{j}{\ominus} \ a_{ij} = \underset{j}{\ominus} \ \underset{i}{\phi} \ a_{ij}$$

Thus a matrix is a column of rows or a row of columns.

9. Transposition rules hold:

$$(\underset{i}{\phi} a_i)^T = \underset{i}{\ominus} a_i^T ,$$

$$(\underset{i}{\ominus} a_i)^T = \underset{i}{\phi} a_i^T$$

It is apparent that compatibility of contiguous arguments which produces a non-overlapping full coverage of the resulting rectangular matrix is a semantic property which transgresses the bounds set by the formal axioms of formula manipulation and bears a similarity in this respect to the exclusion of zero in the definition of arithmetic division. By way of an example axioms 7,8, again 7 and 5, may be applied to obtain the wellknown matrix multiplication rule, without recourse to drawings or explicit mentioning of rows and columns or "lively gesticulation". Take two matrices $\mathop{\Phi}\limits_{i=1}^{n} \mathop{\ominus}\limits_{j=1}^{m} a_{ij}$ and $\mathop{\Phi}\limits_{j=1}^{m} \mathop{\ominus}\limits_{k=1}^{p} b_{jk}$, find their product $\mathop{\Phi}\limits_{i=1}^{n} \mathop{\ominus}\limits_{k=1}^{p} c_{ik}$:

$$\mathop{\Phi}\limits_{i=1}^{n} \mathop{\ominus}\limits_{j=1}^{m} a_{ij} \times \mathop{\Phi}\limits_{j=1}^{m} \mathop{\ominus}\limits_{k=1}^{p} b_{jk} = \mathop{\Phi}\limits_{i=1}^{n} (\mathop{\ominus}\limits_{j=1}^{m} a_{ij} \times \mathop{\Phi}\limits_{j=1}^{m} (\mathop{\ominus}\limits_{k=1}^{p} b_{jk})) =$$

$$= \mathop{\Phi}\limits_{i=1}^{n} (\mathop{\ominus}\limits_{j=1}^{m} a_{ij} \times \mathop{\ominus}\limits_{k=1}^{p} \mathop{\Phi}\limits_{j=1}^{m} b_{jk}) = \mathop{\Phi}\limits_{i=1}^{n} \mathop{\ominus}\limits_{k=1}^{p} (\mathop{\ominus}\limits_{j=1}^{m} a_{ij} \times \mathop{\Phi}\limits_{j=1}^{m} b_{jk}) =$$

$$\mathop{\Phi}\limits_{i=1}^{n} \mathop{\ominus}\limits_{k=1}^{p} (\sum_{j=1}^{m} a_{ij} \times b_{jk}) = \mathop{\Phi}\limits_{i=1}^{n} \mathop{\ominus}\limits_{k=1}^{p} (\sum_{j=1}^{m} a_{ij} \times b_{jk}).$$

Hence:

$$c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk} \qquad \text{q.e.d.}$$

## 4. Testing by the two paths method

As it was adumbrated in the introduction, reasonable assurance as to program correctness may not be achieved by simple, however careful, reading of the program text. A more potent tool is needed to gain confidence and exclude effects of misprints or confusion which have always to be reckoned with.

In the first place the methods of program description and design, the TDL language and the concatenation operator method tend to reduce the effort of imagination and simplify the implementation. Now, a formally correct program must undergo testing which would yield results and these would be compared with some other results obtaiment by a different method.

Thefollowing examples of alternative methods may be quoted:

1° formula integration and numerical integration,

2° equilibrium computation,

3° Betti - type calculations.

The latter two methods are usually combined with orthogonal testing. This operation consists of mapping the successive basis elements of the data into an integral measure of the results by means of a program module. Since such integral measures may be obtained by an alternative formula manipulation and computation a two - path method results.
Take linear spaces of structural mechanics. The theory of linear elastic structures, in its present form, involves a wide spectrum of particular conceptual elements which evolved during various stages of the development of mathematical analysis and higher algebra.

Some unification of the conceptual apparatus must be performed when numerical calculations are considered, in order to isolate the individual operations of computation design. Also, particular physical assumptions have to be separated from an overall organisation of the problem stated and of the resulting program package.

Such an overall organisation follows from the application of the notion of a linear space. Structural mechanics makes also use of real and complex numbers, say, of number fields of higher algebra. It should be borne in mind that, due to the truncation error, an approximation to a number field appears on a computer not the number field itself.

As aforesaid, another important notion in present - day structural mechanics is the linear space concept. The elements of a linear space are not just numbers any more and they are endowed with some simple properties called axioms.

As a fundamental linear space of structural mechanics let us consider the set of all possible self - equilibrated force systems /SEFS for short/ which may act on a selected structural element and deform it. The linear space of all SEFS satisfies the following axioms:

A. A sum of two /or any number/ of SEFS's is again a SEFS.

B. There is a unique null SEFS, which may be added without altering the deformation of the body.

D. multiplication of a SEFS by a number from a field of real or complex numbers results in a SEFS. Multiplication by 1 and zero results in an unaltered SEFS and a null SEFS respectively.
C. Summation of SEFS's is distributive and commutative. A SEFS may be annihilated by adding a negated SEFS to it.
E. Multiplication of a SEFS by a number is distributive with respect to SEFS summation, with respect to sums of numbers and their products.

The above axioms of SEFS are the same as these of a vector space. Therefore the SEFS space is a linear space.

Note that a non - null SEFS is necessary to produce static deformation of a body. Therefore it follows that SEFS is a fundamental concept of structural mechanics. A deformed state of a body determines the generalized displacements of its points with an additional freedom given by the group of stiff translation and rotation. There is thus a set of generalized displacement spaces associated with a SEFS space for a particular body. By fixing the body in a position a single GD space may be made to correspond to a SEFS space. Thus a mapping of the SEFS space onto a GD space characterizes the body within the linear theory, together with a support of the basis functions called the region occupied by the body.

The above argument applies /with some limitations/ to infinitely dimensional SEFS and GD spaces. Spaces of infinite dimension naturally result from say, Fourier series expansions. For numerical purposes they have to be approximated by finite dimensional ones. Such an approximation or truncation suffers from a truncation error which may be visualized as a distance of the true solution from the approximating space. Since a priori estimation of the neceassary number of terms of the expansion may not be practicable, computation may be repeated for a few truncations and thus accuracy may be estimated.
Starting from the well known notion of a modulus /i.e. norm/ of a vector, the notion of a distance together with a norm engendered by it may be introduced for the SEFS and GD spaces. Thus in the first place:
1. A basis of a finite dimensional SEFS /or GD/ space is assumed:

2. Any SEFS /resp. GD/ is expressed as a linear combination of basis elements:

$$c^T f$$

in matrix notation.

A distance $\varrho$ /defining the metrics of the space/ is a non-negative function of 2 arguments $z_1$, $z_2$ which are the elements of the space studied. Distance satisfies the conditions (axioms):

1. $\varrho(z_1, z_2) = 0$ only when $z_1$, $z_2$ coincide.
2. $\varrho(z_1, z_2) = \varrho(z_2, z_1)$ /symmetry/
3. $\varrho(z_1, z_3) \leqslant \varrho(z_1, z_2) + \varrho(z_2, z_3)$ /triangle inequality/

It is easy to verify that the Euclidean metrics:

$$\varrho(z_1, z_2) = \sqrt{(c_1^T - c_2^T)(c_1 - c_2)}$$

satisfies the axioms of distance /1-3/, so it is a definition of distance acceptable for use with a SEFS or GD space. Euclidean metrics engenders a norm

$$\|z\| = (\varrho(z, 0))^{1/2}.$$

The spaces SEFS and GD /for fixed body position/ are isomorphic. Once a problem has been solved for $c$ components of forces and displacements /or stresses and strains/ in the body follow directly as functionals of the elements of the SEFS space. At this stage elements of the basis are used to express dependence on spatial coordinates. This is obviously a generalisation of the influence line concept, although the notion of a linear space requires an overall approach which tends to organize the arguments and counteract ommissions.

Finite dimensional spaces are complete in the sense that every Cauchy sequence of elements of the space has a limit which belongs to the space. A complete linear space endowed with a norm is a Banach space. Important theorems on iterative processes of computation are formulated for Banach spaces and known as theorems on norm - reducing operators. /cf. Banach - - Cacciopoli theorem &c./

Checking program modules makes use of the property that the norms used are homogeneous:

$$\|\lambda z\| = |\lambda| \, \|z\|.$$

Therefore "orthogonal testing" is used in which any mapping program is tested for each basis element or each class of basis elements separately. In order to visualize the method let us consider a curved rod whose axis follows a segment of a circular arc. The rod is clamped at $\theta = 0$ and is subject to load distributed along the median line according to the complex exponential $\exp(\pm a \pm ib)\theta$ /$\theta$ - angular coordinate/ and there exists a program module which is capable of calculating the rod median line displacements for this kind of load, which may be called load of class I. On the other hand the static response of a rod clamped at one end to a concentrated load /class II load/ at its free tip is also known. The Betti rule says that work performed by load I on displacements of class II is the same as the work of load II on displacements of class I. But the latter work is obtained by simple algebra, though however work "I on II" necessitates computing a few integrals over the basis functions. These integrals are verified by cross-checking formulae with numerical calculation. When no collision is found in the whole environment the last step of calculation checks the effects of geometrical and physical data by finding the response of the rod tip to changes in size or in stiffness. Extensive testing results usually in pinning down of some malicious misprints.

## References

[1] Dahl O.J., Dijkstra E.W., Hoare C.A.R., Structured Programming, New York, Academic Press 1972.

[2] Maggiolo - Schettini A. Una teoria algebrica delle strutture e delle loro derivazioni, in: Conferenze del Seminario di Matematica dell' Università di Bari nr. 174. Laterza, Bari 1980.

[3] Martinek J., LISP Opis, realizacja, zastosowania /LISP Description, implementation and applications/ in Polish, English summary, Warszawa WNT 1980.

[4] Winston P.H., Artifical Intelligence, Addison - Wesley 1977.

Appendix A

A example of the two - path method of program module testing

Consider a segment of a cylindrical shell cut - off by 2 axial planes /$\theta$=const/ and 2 {z=const}-planes, thus a median surface segment:

/A1/ $$\bar{\Omega} = [0,\theta_L] \times [0,\zeta_k]$$

$$0 \leq \theta \leq \theta, \quad 0 \leq \zeta \leq \zeta_k .$$

Introduce the notations:

$\mathcal{C}$ - a vector of degrees of freedom composed of subvectors $\mathcal{C}^{(s)}$ related to various modes of deforming the segment. (s=1, ... , 5),

$\overset{g}{A}{}^{(s)}_J$ - matrix operators for g-th component of generalized displacement, s-th deformation mode, J=1,2,

$\psi^{(s)}$ - the s-th functional basis.

$Q^{(s)}$ - the s-th matrix of coefficients of a Fourier series expansion by the DOFO method /interspersed $\{0.5a_0, a_1, b_1, \ldots , a_{kN}, b_{kN}\}$ for consecutive $\psi$'s of a $\psi^{(s)}$ basis/,

$p_g$ - weights,

K - edge index /K=1,2 left, right or top, bottom/,

J - edge pair index /J=1,2 upright and horizontal/.

$$\overset{g}{A}{}^{(0)}_J = \sum_{s=1}^{5} \overset{(s)}{M}{}^{*T} \overset{g}{A}{}^{(s)} \overset{(s)}{M},$$

$$\psi^{(0)} = \bigcup_{s=1}^{5} \psi^{(s)} .$$

/where $M^*$ and M are given /0-1/ matrices/

One path of the algorithm calculates the expansion coefficients of the 4 generalized displacements over the 4 edges of the shell segment, starting from a given vector $\mathcal{C}$ . This vector may be in "ortogonal" form i.e. the 1-out-of-n item is non - zero and equal to one.

Any generalized displacement of a point of the segment's median surface $\bar{\Omega}$ is given by the representation in terms of $\theta$ and $\zeta$ :

/A2/ $$(GD)^{g} = \sum_{s=1}^{6} c^{(s)} \overset{g}{A}{}_{J}^{(s)} \psi^{(s)}$$

Due to the sheer number of $\overset{g}{A}{}_{J}^{(s)}$'s a whole program module gen GD implements the formula. This represents a forward path of the calculation. The GD's obtained are next expanded in a Fourier series to yield $\overset{g}{U}_{KJ}$ and treated as starting data for calculating the free term:

/A3/ $$Q_{III} = \sum_{g=1}^{4} p_g \sum_{J} \sum_{K} \left[ \overset{g}{A}{}_{J}^{(o)} \int_{0}^{\hat{\delta}_{KJ}} \psi^{(o)} \Big|_{KJ} \overset{g}{\Lambda}{}^{T}_{KJ} d\delta_{KJ} \overset{g}{U}_{KJ} \right]$$

where a vertical line denotes a trace, $\psi^{(o)}$ is the set - theoretical sum of $\psi^{(s)}$'s, $\overset{g}{\Lambda}{}^{T}_{KJ}$ is an appropriate trigonometric function, $\hat{\delta}_{KJ}$ is associated with the /K, /-th edge. $\overset{g}{A}{}^{(o)}_{J}$ is a palimpsest of the $\overset{g}{A}{}^{(s)}_{J}$'s, $p_g$ is a weight vector. A similar formula holds for the matrix

/A4/ $$A_{III} = \sum_{g} p_g \sum_{K} \sum_{J} \overset{g}{A}{}^{(o)}_{J} \int_{0}^{\hat{\delta}_{KJ}} \psi^{(o)} \psi^{(o)T} \Big|_{KJ} d\delta_{J} \overset{g}{A}{}^{(o)T}_{J}.$$

After solving the equation:

$$A_{III} c = Q_{III}$$

for $c$ the results are next compared with the starting $c$ values and symptoms of possible misprints may be found or aggreement certified. Details of the calculation comprise two-path testing of the integration in formulae /A3/ and /A4/, and numerous logical decisions concerning the choice of appropriate $A$'s, $\Lambda$'s, U's, $\delta$'s etc.

Another point is the Fourier expansion performed by the method exposed in [1]. The extent of the truncation error is also appreciated by comparing the starting $c$ values with the values resulting from the expanded GD traces on the edges.

In TDL the whole procéss assumes the form:

```
   ⋀ i ∈ [1,...,22](setc,. gen GD, gen QIII,
     take A III, solve to get c, compare and interpret );
```

The TDL descriptions of the successive parts are given below.

Describe "gen GD" in TDL in a somewhat self - explaining way:

```
gen GD( recall param, init c (zerc, one in c),

  g∈[1,2,3,4] ( gen GDg, print GD, store GD);

gen GDg( zer ref GD,  J∈[1,2]( gen GDgJ));
gen GDgJ ( zerQs,  K∈[1,2] ( gen GDgJK,
      print accu, accu to GD,
      incr storef GD );
gen GDgJK ( zer accu, zer par Qs, gen Qs part I, ab to Qs, set param,
   ⋀ s ∈ [1, ... , 5]( part c to cs, zer AgsJ, gen AgsJ, print A cs Qs,
gen Qs part II, gen W, W to accu);
genQs part I(  J=1(expand 1z, pack in Qs),
      ⋁J=2 (expand 1 th, pack in Qs));
genQs part II(⋁J=1(expand rest psiz, pack in Qs,
    mult constantsI), ⋁J=2 (expand rest psi th,
    pack in Qs, mult constants II ));
```

Another module, gen QIII may also be given a TDL description:

```
genQIII (recall param, find U, zer QIII,
    ⋀ g ∈[1,2,3,4] ( set storef U,
    ⋀ J ∈ [1,2]  (gen A g OJ,
    ⋀ K ∈ [1,2]  ( recall U, zer 1, print U,
      zer b,⋀ k2 ∈ [0,...,2 kN]( print2 blank lines,
    ⋀ k1 ∈ [1,...,13] (integralS,  wrong ( printS) , accb),
    incr l by 4), print b, AgOJS to AUX, acc QIII))))); 
    print QIII, zer l, set q n1,
    $\bigwedge$ i ∈ [1,....,n1] (cut QIII), print QIII, outtape QIII);
```

Formula /A3/ has been given a form

$$/A3a/ \quad Q_{III} = \sum_{g=1}^{4} p_g \sum_{J=1}^{2} {}^{g}A_J^{(o)} \sum_{K=1}^{2} \sum_{k_2=1}^{2 \cdot kN} \left[ \left( \int_0^{\hat{\sigma}_{KJ}} \psi^{(o)} \, {}^{g}\Lambda_{KJ}^{(k2)} \, d\sigma_{KJ} \right) {}^{g}U_{KJ}^{(k2)} \right].$$

to explicitate the order of operations designed to keep the size of arrays at a possibly low value.

An inquisitive reader might require some detailed information on the TDL segments written above. He will probably notice that "recall" and "store" have their usual storage retrieval and initialization significance, "print" or "mult" need no explanation, "set" is shorter than "intialize", "zer" means zeroe, "gen" is short for generate, "find" means supplying external storage reference address, "incr" stands for increment a counter, "pack" denotes putting in an appropriate location within a set, "expand" and "cut" denote changes in set size to get a non - singular matrix. "to" is used for transfers between sets, large or small, "sto ref" is storage reference. Other names are directly linked to the significance of the implemented formulae.

## Reference

[1] Mączyński J. An algorithm for Fourier series expansion in a subinterval of the circumference. Algorithm 79, Applicationes Mathematicae 17, 1, /1980/.

Appendix B

## MATRIX MULTIPLICATION REVISITED

By applying the concatenation operators $\ominus$ and $\Phi$ matrix multiplication may be brought to the form:

$$\text{(B1)} \quad \underset{i}{\Phi}\,\underset{j}{\ominus}\,\sum_{k} a_{ik}\, b_{kj} = \sum_{k}\,\underset{i}{\Phi}\,\underset{j}{\ominus}\, a_{ik}\, b_{kj} = \sum_{k}\,\underset{j}{\ominus}\,\underset{i}{\Phi}\, a_{ik}\, b_{kj} = \underset{j}{\ominus}\,\sum_{k} (\Phi\, a_{ik})\times b_{k} \ .$$

Proof:
Use Example in § 3, rule 8 therein and interchange elements in the sums.

The above formula may immediately serve to organize multiplication of large matrices for efficient use of external mass storage (EMS).

Rapid access storage (core) contains in this case a few single columns of the matrices involved. Therefore the algorithm demands a sequential arrangement of matrix columns and no transposition or picking out of single elements is necessary any more.

A program for this purpose may be outlined in TDL as follows:

```
$\bigwedge j \in [1,\ldots,n]($
take $\underline{b}$ - $\underline{u}$ ($j$-th column of $[b_{kj}]$),
$\underline{w} \leftarrow 0$,
$\bigwedge k \in [1,\ldots,p]($
take $\underline{a}_k \leftarrow \underline{v}$,
$\bigwedge i \in [1,\ldots,n]($
$w_i$ - $w_i + u_k * v_i$ )),
put $\underline{w}$ - $\underline{c}_j$
);
```

The operators take and put may be devised to suit some buffering technique. Thus concatenation algebra may serve to obtain algorithms for efficient manipulation of matrices.

## Appendix C

### The simple sum and the simple product of matrices

The concatenation operators $\phi$ and $\bar{o}$ may be used to define further operations on matrices such as the simple sum and the simple product of matrices. The simple product is ascribed to Kronecker and sometimes called the Kronecker product.

Write $A_{U*V}$ to denote a matrix with U rows and V columns, then a simple sum of a pair of matrices $A^{(1)}_{P*Q}$ and $A^{(2)}_{R*S}$ is:

$$A_{(P+R)*(Q+S)} = (A^{(1)}_{P*Q} \;\bar{o}\; O_{P*S})\,\phi\,(O_{R*Q} \;\bar{o}\; A^{(2)}_{R*S})$$

where $O_{U*V}$ is a zero matrix of U rows and V columns.

A simple sum of n matrices is associative but not commutative, therefore the indices (1,2) in the definition are a pair and may not be interchanged. For any n we write:

$$A_{\Sigma P_i * \Sigma Q_i} = \bigoplus_{i=1}^{n} A^{(1)}_{P_i * Q_i}$$

where no pair of the row of indices [1,2,...,n] may be interchanged, although formation of partial sums is not prohibited.

Forming a concatenation of elements:

$$A^{(1)} = \phi_i \, \bar{o}_j \, a^{(1)}_{ij}$$

a simple product of two matrices $A^{(1)}_{P*Q}$ and $A^{(2)}_{R*S}$ is written:

$$A^{(1)}_{P*Q} \otimes A^{(2)}_{R*S} = A_{PR*QS} = \phi_i^P \, \bar{o}_j^Q \, a^{(1)}_{ij} A^{(2)}_{R*S} .$$

A simple product is neither associative nor commutative. In general

$$A_{\Pi P_i * \Pi Q_i} = \bigotimes_{i=1}^{N} A^{(1)}_{P_i * Q_i} .$$

By a special aggreement the products are formed by taking successive pairs in a decreasing order of index pairs (rom right to left). Here $\Pi$ denotes an ordinary multiplication of integers.

The following relations hold:

(1) $(A^{(1)} \oplus A^{(2)})(B^{(1)} \oplus B^{(2)}) = A^{(1)}B^{(1)} \oplus A^{(2)}B^{(2)}$,

(2) $(A^{(1)} \otimes A^{(2)})(B^{(1)} \otimes B^{(2)}) = A^{(1)}B^{(1)} \otimes A^{(2)}B^{(2)}$,

(3) $A^{(1)}_{P_1 * Q_1} B^{(1)}_{Q_1 * S} \;\phi\; A^{(2)}_{P_2 * Q_2} B^{(2)}_{Q_2 * S} =$
$(A^{(1)}_{P_1 * Q_1} \oplus A^{(2)}_{P_2 * Q_2})(B^{(1)}_{Q_1 * S} \;\phi\; B^{(2)}_{Q_2 * S})$,

(4) $A^{(1)}_{P_1*Q} B_{Q*S} \phi A^{(2)}_{P_2*Q} B_{Q*S} = (A^{(1)}_{P_1*Q} \phi A^{(2)}_{P_2*Q}) B_{Q*S} =$
$= (A^{(1)}_{P_1*Q} \otimes A^{(2)}_{P_2*Q})([1]_{2*1} \otimes B_{Q*S})$

where $[1]_{2*1}$ (in general $[1]_{N*1}$) is a column of unit elements.

The relation

$$A_{P*Q} \otimes B_{R*S} = (A_{P*Q} \otimes I_R)(I_Q \otimes B_{R*S})$$

follows from (2).

Such and similar relations hold for simple products and simple sums of matrices. Their application to the special matrices of digital holography may be found in [1] where further references are given.

## R E F E R E N C E S

[1] Yaroslavskii, L.P., Merzlyakov N.S., Methods of digital holography, N.Y. Plenum Press 1980 (cf. new Russian edition: Цифровая Голография, Москва, Наука I982).

Jacek Mączyński

Pracownia Obliczen Numerycznych IPPT PAN

## Prosta Technika Pamięci wirtualnej

Nieskomplikowane procedury WPISZ i POLE obsługują dostęp do zewnetrznej pamieci masowej /EMS/, co sprzyja uproszczeniu algorytmow pracujących z dużymi tablicami. Technika pamięci wirtualnej może znaleźć zastosowanie w takich zagadnieniach jak rozwiązywanie równań różniczkowych o pochodnych cząstkowych metodą różnic skonczonych itp.

Яцек МончиНский

Лаборатория Вычислительной Техники

## Об простой технике виртуальной памяти

Несложнче процедуры WPISZ и POLE ("впиши" и "поле") обслуживают доступ к внешной массовой памяти (EMS ) что способствует упрощению алгоритмов работающих с массивами больших размеров. Техника виртуальной памяти может наити примaнение в таких задачах как решение дифференциальных уравнении с частными производными методом конечных разниц и проч.

Jacek Mączyński
Computing Laboratory

## A straightforward virtual memory technique

There are many situations when the size of arrays needed in a program transgresses the available core size. The need arises to organize communication with the external mass storage (EMS). This implies some skill, patience and attention which may regrettably be diverted from the main purpose of the computing job undertaken which is e.g. supplying application results in a particular field of science. There is thus a trade-off between somewhat lower computation efficiency when a semi-universal technique (like the one described below) is selected and the cost of either hiring a specialist or that of do-it-yourself study and trials.

Again a program text is an excellent medium for defining an algorithm and therefore a first working version should be kept as clear as is physically possible and nevertheless operate with unimpaired array sizes. This is another advantageous feature of the technique suggested here.

The simple virtual memory technique described below is universal when arrays are conceived from the very start to be one-dimnensional vectors. Some very simple calculation such as:

$$l = k + n*(j - 1 + p*(i-1))$$

(when a lexicographically stored array, say A[1:n,1:p,1:q] with an element A[i,j,k] is to be simulated) supplies the necessary current field index l as a function of the given indices i,j,k and their bound pair values (1:n), (1:p), (1:q).

The technique is implemented at some small expense of attention. Two procedures conveniently named WPISZ (write in) and POLE (field) together with an array OKNO (window), a real variable ENTRY and an integer variable dok have to be declared in the program, preferably in the most external block of an ALGOL program.

Next, dok and drumplace (this is a non-local standard integer variable to be supplied when missing in some algol version) are put equal to 1 and 0 respectively. Thus own variables (or COMMON variables should FORTRAN be considered) are simulated without having to explicitate them as procedure parameters.

Writing a real number into an EMS field involves two operations:
- initializing the variable ENTRY,
- calling the WPISZ procedure with a required index value.

In the version implemented index expressions may be used as actual parameters of WPISZ (cf. the text program below).

Once a transmission stage has been terminated and a retrieval stage is to begin, a mopping-up operation is needed. This consists in transmitting the contents of the buffer OKNO to the EMS in order to update the field and save the contents of the buffer .

Conversely, an item (a real number) is retrieved from EMS by putting some real variable equal to POLE(i) where the parameter i is the required current index in the EMS field.

An experienced reader may have noticed by now that the variable dok seems to have been treated as a program constant. The procedures WPISZ and POLE leave it unchanged. The role of this variable is to provide an encoded identifier for the set in EMS with which contact is established. This is a number which is set to be the first half-item of the field in EMS. If more than one storage field is used by the program, additional care must be taken properly to classify the contacts with the respective fields and to initialize dok. It must be borne in mind that when there are N items in a field (say, n*p*q items of an array A[1:n,1:p,1:q]) there are 2N half-items and dok has to be increased in steps which are double values of the field sizes in items (on the particular computer considered). Such doubling should present no difficulty and a rough sketch of the memory fields (which may be called sets) required by the program should be highly appropriate.

The above recipe is wholly sufficient for successful use of the technique. No resposibility may be assumed for failures due to overreaching of bounds or some inadvertent overlaying of data or to contacting an uninitialized field.

The technique having been tuned, prospective users are invited to try their ability to avail themselves of it. Therefore the test program below and the description of the procedures should best be read carefully. TDL [1] was used for design and description of the algorithms because of its capability to convey program structure.

The procedure POLE

Consider the TDL description:

POLE(set m, set ref, ∨ OKNO on portion and i ∊ range(take from OKNO), ∨ not OKNO on portion or not i ∊ range (take, backdrumplace));

where:

| | |
|---|---|
| i | - index of an item in an EMS field, |
| m | - OKNO buffer size in items (set to 512) |
| setm | - initialisation of m, |
| set ref | - finding of the reference zero value of the buffer position over the EMS field expressed in half-items (j) and whole items (k). |

Further:

OKNO on portion is true when the value of the position (in the EMS field) of the first half-item to enter the buffer (i.e. dok+j) is equal to the actual drumplace value. It is assumed that this should not happen by chance and therefore drumplace is set to zero at the start of the program (since drumplace is used to signal a correct position of the buffer, so it has to be desensitized at first).

**i ∈ range  is true when i is within the buffer,**
**take from OKNO is a take from buffer program segment,**
**dok - an EMS field zero reference in half-items,**
**take - here an item is taken from a previously filled buffer,**

We have:

    take(set drumplace, supply portion, take from buffer),

where:

| | |
|---|---|
| set drumplace - | buffer relative position j and the EMS field position dok yield the drumplace value, |
| supply portion - | the whole buffer is refilled from the EMS, |
| back drumplace - | the drumplace value is put to the value returned by set drumplace. |

As a result either values in the buffer have been previously up-dated so they show the right portion of the EMS field or conversely an up-date is spurred by demand and a correct buffer content is established. Note that the buffer is visualized as a mobile window hovering over the EMS field.

The procedure POLE is a real function and may appear within expressions. Economy of the number of parameters and relatively few instructionw needed to read a buffer value when no up-date is needed, should make the use of POLE reasonably efficient.

## The procedure WPISZ

WPISZ(set m, set ref, ∨ OKNO on portion and i ∈ range(put in buf), ∨ not OKNO on portion or not i ∈ range(transmit, take, put in buf, backdrumplace));

The variables i, m, k, j, dok, the array OKNO, the tasks set m, setref, take, back drumplace and the predicates OKNO on portion, i ∈ range are the same as in POLE.

| | |
|---|---|
| put in buf - | carries ENTRY into OKNO, |
| transmit - | uses the previous drumplace value to store the actual contents of OKNO. |

To avoid an excessive number of transmissions the buffer (OKNO) holds its contents as long as the buffer hovers in place over a part of the EMS field. Hence a terminating construct is needed before the variable dok is altered and this is provided by:

    to drum(512, OKNO[1])

in the test program below.

Note that the constructs to drum and from drum increase the value of the drumplace by 2N when N items are transmitted to or from the drum. This side effect has to be simulated on other computers.

The technique has been implemented and tested by the author on an ODRA-1204 computer. The portability of the technique is seemingly high and no major difficulty in applying it with say, disc storage and with some other algorithmic language is apparent. Obviously the procedures POLE and WPISZ have to be re-written in this case with some caution and tested before incorporation into a larger program.

## R e f e r e n c e s

[1] J. Mączyński, A test-bench philosophy for the program modules of the Tholos package (this issue).

```
comment
TEST WPISZ, POLE;
begin
integer dok; real ENTRY; array OKNO[1:512];
∫
 procedure WPISZ(i);
value i; integer i;
comment non-local integer dok, real ENTRY,
array OKNO[1:512];
begin integer k, j, m;
m:=512;
k:=(i-1)div m*m; j:=k+k;
if drumplace=dok+j and i ge k+1 and i le k+m then
OKNO[i-k]:=ENTRY else
begin
to drum(m, OKNO[1]);
drumplace:=dok+j;
from drum(m, OKNO[1]);
OKNO[i-k]:=ENTRY;
drumplace:=drumplace-m-m;
end;
end;

. . .   . . . . . .
real procedure POLE(i);
value i; integer i;
begin integer k, j, m;
comment non-local integer dok, array OKNO[1:512];
m:=512;
k:=(i-1)div m*m ; j:=k+k;
if drumplace = dok+j and i ge k+1 and i le k+m then
POLE:=OKNO[i-k] else
begin
drumplace:=dok+j;
from drum(m, OKNO[1]);
POLE:=OKNO[i-k];
drumplace:=drumplace-m-m;
end;
end;
∫
dok:=1; drumplace:=0;
begin
integer i; format('1234,'); print('∫test wpisz pole∫');
for i:=1 step 1 until 4096 do
begin
ENTRY:=1.0*i; WPISZ(i);
end; to drum(512, OKNO[1]);
for i:=1 step 1 until 4096 do
if POLE(i) ne i then print(i);
end interior;
print('∫koniec testu wpisz, pole∫');
end∫
```

Jacek Mączyński
Pracownia Obliczeń Numerycznych

## O pewnym równaniu różniczkowo-całkowym

Równanie różniczkowo-całkowe /1/ wynikające z rozwiązań elastostatyki trójwymiarowej zostało zastosowane do badania rozkładu przemieszczen w prostoliniowym zbrojeniu cięgnem. Przedstawiono wyprowadzenie algorytmu obliczeniowego i przykład liczbowy.

Яцек Мончинский
Лаборатория
Вычислительной Техники

## Об одном интегродифференциальном уравнении

Интегрально-дифференциальное уравнение (I) следующе из фундаментальных решении эластостатики применяется к исследованию распределения перемещении в прямолинейной волокнистой армировке. Решение получается методом кубических сплайнов. Представлено выведение вычислительного алгоритма и числовой пример.

J. Mączyński
Computation Laboratory

## On an integro-differential equation

A preliminary solution of the homogeneous integro differential equation:

$$(1) \qquad u(z) = \int_{z_0}^{L + z_0} (K(z-s) - K(z+s))\, \ddot{u}(s)\, ds = \mathcal{L}(\ddot{u})$$

with the non-homogeneous boundary conditions:

$$(2) \qquad \dot{u}(z_0) = P_0/c, \quad \dot{u}(z_0 + L) = 0$$

is presented.

This equation approximately describes the displacement field along a straight extensible fibre of length L which is pulled by a force $P_0$ at $z=z_0$ and submerged in a linearly elastic matrix filling a half space which almost everywhere is rigidly supported over the $z=0$ plane with the exception of the point $z=0$, $r=0$ where the thin fibre intersects the plane.

The kernel K is obtained by considering the classical fundamental solutions of Boussinesq-Kelvin for a small fibre diameter $r_0$. A simple approximation to the kernel (replacing it by a concentrated "Dirac mass") produces a rapidly decaying rough solution which may be used to compare the result with the behaviour of the solutions obtained for a more refined kernel shape.

The method of solution presented below viz. spline interpolation, was selected from a number of other approaches, such as e.g.

- the Bellman-Kalaba embedding technique,
- Fourier transformation.

The spline interpolation proved to be by far the most straightforward, numerically effective, and therefore most suitable for a computer implementation. Other, more traditional methods e.g. some iterative approach or series expansion proved either divergent on inspection or cumbersome (hence unreliable).

Integral and also integro-differential equations require for their solution an inversion of a discretised form of the operator such as $\mathcal{L}$ above. Thus usually a system of equations has to be solved. Therefore, keeping down the number of unknown s is a main requi rement. Spline interpolation provides a sufficiently high ($O(h^4)$) order of accuracy of integration and therefore combines economy and reliability. The main problem concerns the accuracy obtained. Therefore calculations were performed for a few different mesh sizes as presented below.

The kernel of Eq.(1) is composed of two terms, each depending on the same function K given by the formula:

$$K(\sigma) = c_a\frac{1}{r} + c_b\frac{\sigma^2}{r^3}, \tag{3}$$

where

$$r = (r_0^2 + \sigma^2)^{1/2}, \tag{4}$$

$c_a$, $c_b$ are constants.

Putting (3) in dimensionless form we write:

$$\tilde{K}(\tilde{\sigma}) = \tilde{c}_a\frac{1}{\tilde{r}} + \tilde{c}_b\frac{\tilde{\sigma}^2}{\tilde{r}^3} \qquad \text{(cf. Fig. 1)} \tag{5}$$

where

$$\tilde{\sigma} = \sigma/r_0, \quad \tilde{r} = r/r_0, \quad \tilde{c}_a = c_a/r_0^2, \quad \tilde{c}_b = c_b/r_0^2. \tag{6}$$

Values of $\sigma$ are taken from the interval:

$$\min_{i,j\in I}\left((z_j-z_i),(z_j+z_i)\right) \le \sigma \le \max_{i,j\in I}\left((z_j-z_i),(z_j+z_i)\right) \tag{7}$$

where

$$I = [0,1,\dots,n], \quad z_0 \le z_i \le L+z_0. \tag{8}$$

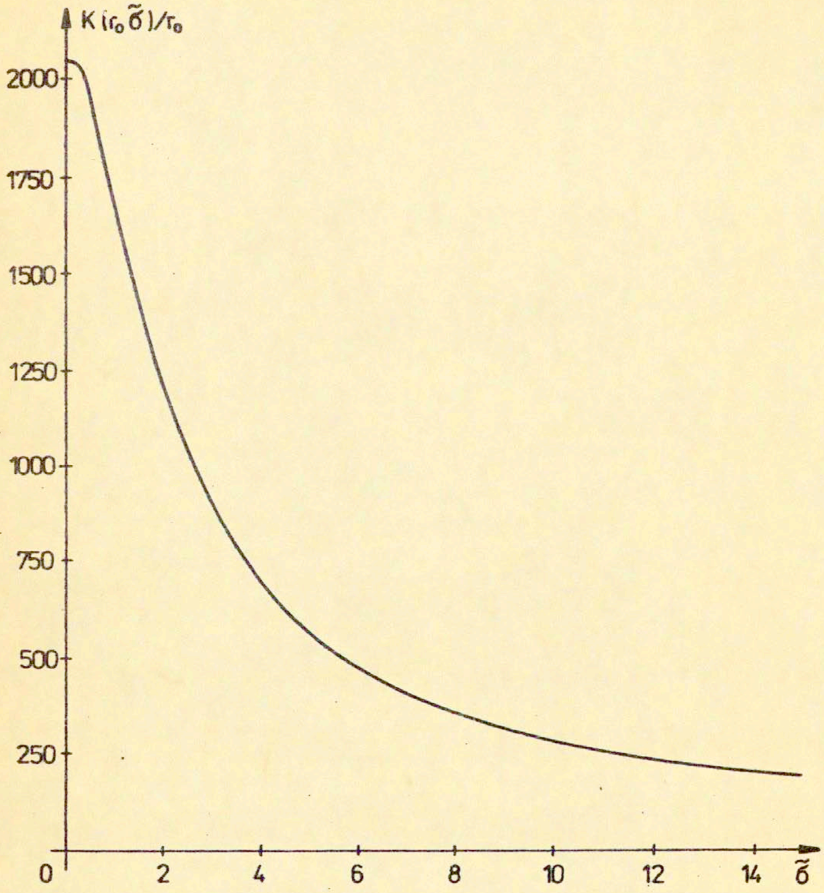


Fig. 1

In order to apply the spline technique we may use a direct approach without trying to make the program highly effective at this stage.

Consider cubic splines over $[z_o, L+z_o]$ divided into $n$ equal subintervals. Second derivatives $\ddot{u}(z)$, $z \in [z_o, L+z_o]$ are represented by a continuous broken line made of straight line segments. Such a line is uniquely described by $n+1$ values: $\ddot{u}_o, \ddot{u}_1, \dots, \ddot{u}_{n-1}, \ddot{u}_n$ at nodal points. Thus for $z \in (z_{k-1}, z_k)$, $h=L/n$:

(10) $$\ddot{u} = \ddot{u}_{k-1}\frac{z_k - z}{h} + \ddot{u}_k\frac{z - z_{k-1}}{h}, \qquad (k = 1,\dots,n)$$

On integration:

(11) $$\dot{u} = -\ddot{u}_{k-1}\frac{(z_k-z)^2}{2h} + \ddot{u}_k\frac{(z - z_{k-1})^2}{2h} + C_k - C_{k-1},$$

$$u = \ddot{u}_{k-1}\frac{(z_k-z)^3}{6h} + \ddot{u}_k\frac{(z-z_{k-1})^3}{6h} + C_k(z-z_{k-1}) + C_{k-1}(z_k-z).$$

The constants $C_k$, $C_{k-1}$ are from continuity:

$$C_k = \frac{1}{h}(u_k - \frac{1}{6}\ddot{u}_k h^2), \quad C_{k-1} = \frac{1}{h}(u_{k-1} - \frac{1}{6}\ddot{u}_{k-1}h^2).$$

On substitution and for $z_k - z = z - z_{k-1} = h/2$:

$$u_{k-1/2} = \frac{1}{2}(u_k + u_{k-1}) - \frac{1}{6}h^2(\ddot{u}_k + \ddot{u}_{k-1}).$$

The boundary conditions for spline uniqueness are assumed to be:

(13) $$\ddot{u}_o = \ddot{u}_1, \quad \ddot{u}_{n-1} = \ddot{u}_n.$$

Therefore $n-1$ internal nodes only have to be considered and $(n-1)$-dimensional matrix vectors are used in the computation.

Write from (2, 11, 12):

(14) $$P_o/c = -\ddot{u}_1\frac{h}{2} + \frac{u_1 - u_o}{h}$$

$$0 = \ddot{u}_{n-1}\frac{h}{2} + \frac{u_n - u_{n-1}}{h}$$

(15) $$\ddot{u}_{k-1} + 4\ddot{u}_k + \ddot{u}_{k+1} = \frac{6}{h^2}(u_{k-1} - 2u_k + u_{k+1}) \qquad (k = 2,\dots,n-2)$$

whereas for $k=1$ from (14):

(16) $$\underline{8}\ddot{u}_1 + \ddot{u}_2 = \frac{6}{h^2}(u_2 - u_1) - \frac{6}{hc}P_o,$$

and for $k=n-1$:

(17) $$\ddot{u}_{n-2} + 8\ddot{u}_{n-1} = \frac{6}{h^2}(u_{n-2} - u_{n-1}).$$

Therefore (15) together with (16) and (17) are put into a matrix form:

(18) $$T_4\ddot{U} = T_2U + \underline{k}$$

Eq.(1) on substitution of (10) and after performing integration assumes the discretized form $(i = 1,\dots,n-1)$:

(19) $$u_i = \ddot{u}_1 \overset{i}{K}_{c1} + \ddot{u}_{n-1}\overset{i}{K}_{cn} +$$

$$+ \frac{1}{h}\sum_{k=2}^{n-1}\left[\ddot{u}_{k-1}(z_k\overset{i}{K}_{ck} - \overset{i}{K}_{csk}) + \ddot{u}_k(\overset{i}{K}_{csk} - z_{k-1}\overset{i}{K}_{ck})\right]$$

where the coefficients are written:

(20) $$\overset{i}{K}_{ck} = \sum_{\alpha=-1,1}\alpha\sum_{p=0,1}(-1)^pK_1(z_{k-p}-\alpha z_i)$$

$$\overset{i}{K}_{csk} = \sum_{\alpha=-1,1}\alpha\sum_{p=0,1}(-1)^p\left[K_2(z_{k-p}-\alpha z_i) + \right.$$

$$\left. + z_iK_1(z_{k-p}-\alpha z_i)\right]$$

and by integrating the kernel (3):

$$K_1(\sigma) = c_a\ln(\sigma+r) + c_b(-\frac{\sigma}{r} + \ln(\sigma+r)),$$

$$K_2(\sigma) = c_ar + c_b(r + r_o^2/r), \quad r = (r_o^2 + \sigma^2)^{1/2}.$$

Eq.(19) may be written:

(22) $$u_i = \sum_{j=1}^{n-1} M_{ij}\ddot{u}_j \quad \text{or} \quad U = M\ddot{U}$$

where

$$M_{i1} = \overset{i}{K}_{c1} + (z_2\overset{i}{K}_{c2} - \overset{i}{K}_{cs2})/h,$$

(23) $$M_{ik} = \frac{1}{h}(\overset{i}{K}_{csk} - z_{k-1}\overset{i}{K}_{ck} + z_{k+1}\overset{i}{K}_{ck+1} - \overset{i}{K}_{csk+1}),$$

$$M_{i\ n-1} = -\frac{1}{h}(z_{n-2}\overset{i}{K}_{c\ n-1} - \overset{i}{K}_{cs\ n-1}) + \overset{i}{K}_{cn}.$$

Equations (18) and (22) are combined into the matrix solution:

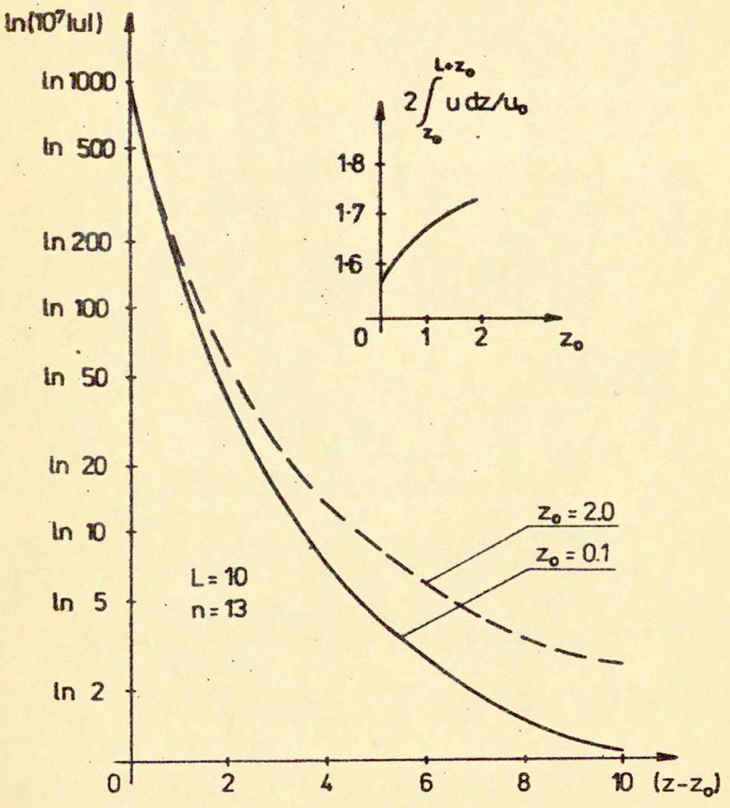
(24) $$U = (I - M\,T_4^{-1}T_2)^{-1}MT_4^{-1}\underline{k}.$$

The method described above bears a similarity to the solution methods of integral equations as described in in, say [1] or [5].

The numerical solution of Eq.(24) for $E_w$=628318, $r_o$= 0.05, L=10, E=20000, $\nu$ =0.16, P= 0.628 (in consistent units) and n = 13 is shown on a semilogarithmic scale in Fig. 2. When varying n from 10 to 13 the usual uniform convergence pattern appears. Curves become more and more closely packed with increasing n. The second derivative $\ddot{u}$ exhibits a somewhat oscillatory behaviour (with a correct trend, though) for lower n values and becomes regular for n=13 with some remaining oscillation close to the end-points. This may be attributed to the oversimplified spline boundary condition (Eq.(13)). The ratio $\ddot{u}/u$ which Greszczuk [3] assumes to be constant shows a regular decrease by an order of magnitude over the main portion of the fibre length L. This is a novel result and is attributable to the shape of the kernel which definitely is not a Dirac mass and follows from exact fundamental solutions of the theory of elasticity.

Another approach to the pull-out problem is given in [2] where a matrix bar is assumed to be uniformly stressed over a finite cross-section by the pull on the fibre and the front part of the bar is free, the far away rear end of the bar being clamped. Therefore direct comparison of the results is not feasible, although an alternative experimental arrangement is suggested by the method presented here.

The insert in Fig. 2 shows an integral measure of the displacement distribution (obtained by Simpson integration) as plotted against $z_o$. This measure (or characteristic length) accentuates the behaviour of the initial portion of the displacement curve i.e. that portion where the displacements are largest. Naturally, the characteristic length is larger when the matrix offers a lesser stiffness due to the increased thickness of the padding layer.



ln(10⁷|u|)
ln 1000
ln 500
ln 200
ln 100
ln 50
ln 20
ln 10
ln 5
ln 2
0
2
4
6
8
10
(z-z₀)
L = 10
n = 13
z₀ = 2.0
z₀ = 0.1
2∫ u dz/u₀ (z₀ to L+z₀)
1·8
1·7
1·6
0
1
2
z₀

Fig. 2

Since Eq.(1) may be brought to a dimensionless form (the result being formally the same as that of inserting a tilda ($\sim$) over the kernel, the variables and the constants) and writing $\tilde{u} = c\, r_0\, u/P$, $\tilde{z} = z/r_0$, $\tilde{s} = s/r_0$, $\tilde{L} = L/r_0$, $\tilde{z}_0 = z_0/r_0$ together with $\tilde{K}$ given by Eq.(5) above, the numerical values obtained hold for parameter values which leave the 5 dimensionless similarity criteria:

$L/r_0$, $z_0/r_0$, $P/c = P/\pi r_0^2 E_w$, $E_w/E$, $\nu$ unchanged.
The boundary condition becomes $d\tilde{u}/d\tilde{z} = 1$ in this case.

## R e f e r e n c e s

[1] Ahlberg J.H., Nilson E.N., Walsh J.L., The Theory of Splines and Their Applications, Chapt. II, § 2.8, Academic Press, New York & London 1967.

[2] Bond Action and Bond Behaviour of Reinforcement (State of the Art Report, Com. Euro-Internat. du Béton, CEB Planary Session, Munich 1982) Apr. 1982

[3] Greszczuk L.B., Theoretical studies of the Mechanics of the Fiber-Matrix Interface in Composites, Interfaces in Composites, ASTM STP 452, Amer. Soc. Test. Mat., pp. 42-58, 1969.

[4] Sokołowski M., On a One-dimensional Model of the Fracture Process, Rozprawy Inżyn. (Warszawa), 25, 2, 369-393 1977.

[5] Stechkin S.B., Subbotin Yu.N., Splines in Computational Mathematics (in Russian), Ch. VI, §3, Nauka, Moskva 1976.