Mausam, (1985), **36**, 2, 159-166

518.5 : 681.326.06

An international library for numerical analysis : INTERLIB*

O. PIRONNEAU

International Scientific Cooperation, Inria
78153, Le Chesnay, France

(Received 2 February 1981)

सार — आंकिक विश्लेषण के लिये यंत्रेतर सामग्री को निम्नानुसार वर्गीकृत किया जा सकता है।

तुरन्त उपलब्ध यंत्रेतर सामग्री, सुगमतापूर्वक स्वंय लिखी जाने योग्य यंत्रेतर सामग्री तथा दुर्लभ आधुनिक यंत्रेतर सामग्री।

इन्टरलिब, तीसरे प्रकार की यंत्रेतर सामग्री उपयोग करने वालों का एक अन्तर्राष्ट्रीय वर्ग है, ये उपभोक्ता ऐसा समझते हैं कि यदि उनके मित्र की यंत्रेतर सामग्री का ठीक से आलेखन नहीं हुआ है, तो उसका उपयोग करना बहुत मुश्किल है।

एक वर्ष की परीक्षण अवधि के बाद कुछ ऐसे सामान्य सिद्धान्तों का वर्णन किया जा सकता है, जो यह बताएं कि स्वतन्त्र योगदान से निर्मित ऐसे संग्रहालय में क्या किया जा सकता है। संग्रहालय की वर्तमान अवस्था भी प्रस्तुत की जाएगी। इसमें आंशिक अवकलन समीकरणों के हल के लिये अधिकतर फोट्रान रूटीन दिए हैं, किन्तु सैद्धान्तिक रूप से यह किसी प्रकार भी इस क्षेत्र तक ही सीमित नहीं है।

भारतीय तकनीकी संस्थान दिल्ली ने इन्टरलिब में शामिल होने का निश्चय किया है, इनके माध्यम से उपभोक्ता को इन्टरलिब की कोई भी रूटीन उपलब्ध होगा।

ABSTRACT. Softwares for numerical analysis may be classified into groups : the currently available softwares, the easy-to-write-yourself softwares and the hard-to-fiind-modern-software.

INTERLIB is an international pool of users of the third kind of softwares who realize that it is very difficult to use your friend's software if it is not properly documented.

After a trial period of one year some general principles can be stated on what to do in such a library which is built from independent contributions. The present state of the library will also be presented : it contains mostly Fortran routines for the solution of partial differential equations but it is in principle by no means limited to this field.

The Indian Institute of Technology, Delhi, has decided to join INTERLIB, therefore, any routine of INTERLIB is available to the attendant through them.

1. Introduction

With the fast growth of computers more and more problems of the exact sciences (physics, medicine, engineering. . .) are now solved by direct numerical simulation. Thus the range of application of numerical analysis has grown considerably, but simultaneously in many instances the methods have also become very sophisticated. Therefore the computer code developed for the simulation of a physical system currently takes several man-years to be built.

In private industries these codes become such an asset to the firm that they generally refuse to allow their free circulation.

In public research laboratories the situation is different. For a given method to solve a particular problem a scientist is not so much interested by an efficiently implemented code than by a simple version of the algorithm which is available for preliminary tests because either the problem to be solved is a block for the solution of a bigger problem or (more often) the study needs to compare a new algorithm with the given one. Indeed let us take an example that I know well : the case of the Navier-Stokes equation, which describes the flows of incompressible viscous fluids. There are about fifty different methods to solve this equation and none of them work for high Reynolds numbers. There are at least three cases in which I would like to have a copy of my colleagues' code for this problem :

(1) A new method is published and it seems good; I would like to try it on some test problem on which I know how some other methods behave. Even though the author has included some tests they are most of the time difficult to process because he publishes the good results and not the bad ones.

*Paper presented in the symposium "Indo-French school on recent advances in Computer Techniques in Meteorology, Biomechanics and applied systems" held at I.I.T., New Delhi, February 1980.

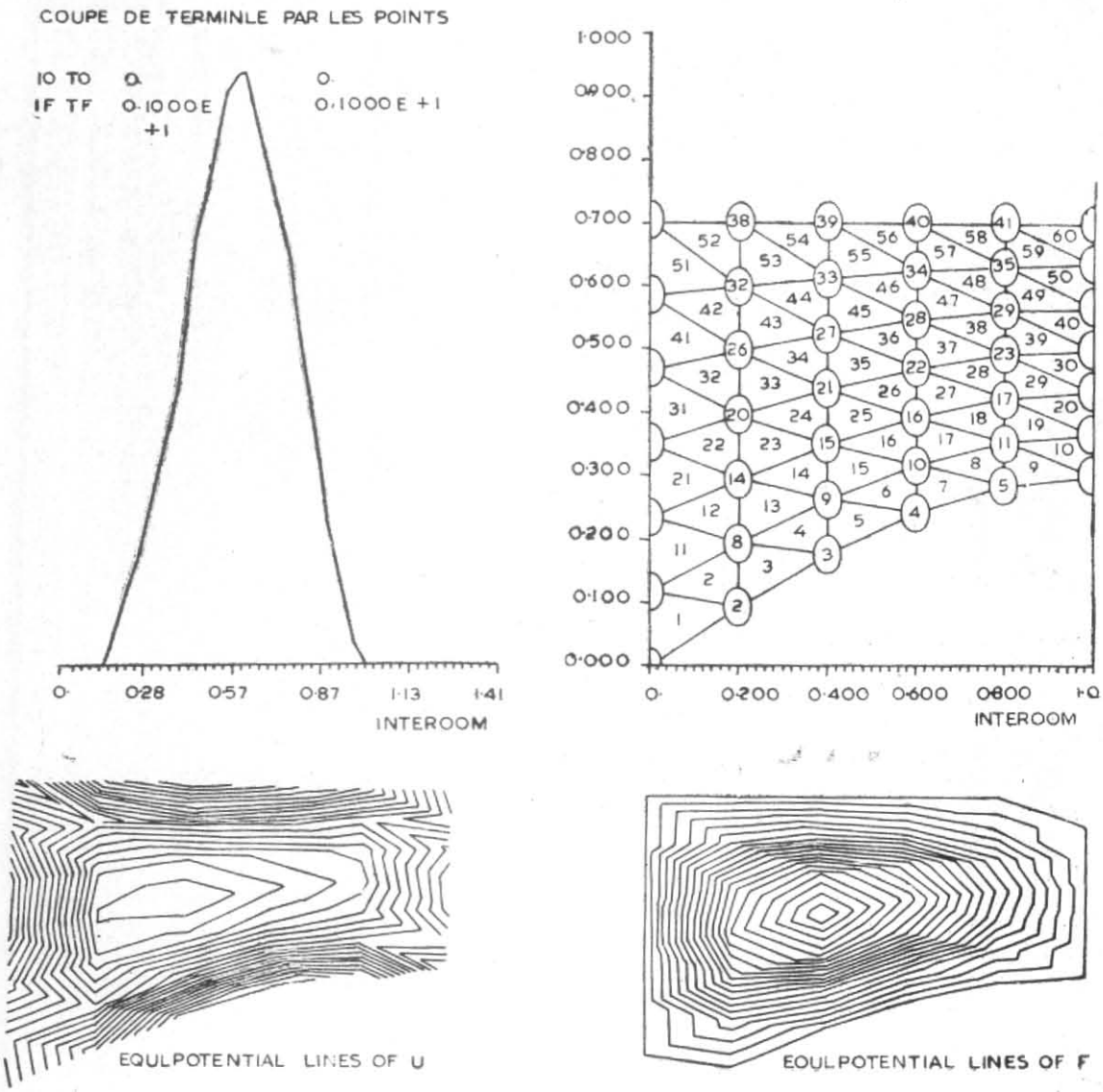


Fig. 1

(2) An engineer from an industry ask me how to solve the Navier-Stokes equation. My experience is that his problem is never standard and there are some non-trivial modifications to be made to the software I own. I know of some other method which would be well suited to his problem unfortunately I can't have him make some preliminary test because it would take me 3 months to build even a simple unsophisticated version of the code.

(3) I propose a new method and I would like to compare it with some of my colleagues' methods.

In all cases I can write to the author to get a copy of the code but he almost surely write back a few months later that the code has not been used for a long time that it would take him 40 pages to describe the parameters and that it uses some sub-routine from some library that I might not have. If I am more lucky the code is published and then, almost inevitably again, the author has used some non-standard Fortran IV facilities which do not exists on my site. All these are evidences to someone who has tackled the problems of software portability.

Thus one finds that most laboratories of numerical analysis have in one form or another some project which aims at the building of a scientific library of softwares. INTERLIB is one of these project with some peculiarities which will be described in the present article.

2. Organization of INTERLIB

The purpose of INTERLIB is to build and maintain a library of Fortran routines of numerical analysis with applications in :—

— *Ecology* : diffusion of pollutants, ocean circulation, meteorology . . .

— *Energy productions* : regulation of dams, solar energy, wind mills . . .

— *Biomedical engineering* : electrocardiogram readings, blood flow modelling immunology . . .

Since it aims at the circulation of software it is club of users, each one gives the software that he has or will develop for his own but which are relevant to the above subjects.

Therefore members are research laboratories, public or private, who already work in one of the above 3 fields and feel that they could benefit from such circulations. For practical reasons the club was started last year with 3 members only :

INRIA, Paris — France Director : J. L. Lions

LAN, University of Pavia-Italie Director : E. Magenes

CCAS, University of Novosibirsk-USSR Director : G. I. Marchuk

It should be noted that the club wishes to operate very much like a scientific journal whose material support is a computer tape and a user manual. Anything relevant to the above subject which complies to the norms and is of interest to the other members can be included into the library.

New members will be admitted next year; anyone who wishes to join can do so by submitting a proposal of contribution to the library. The contribution should not be developed just for the sake of joining the club; it should be part of the research of the laboratory.

Membership is renewed every two years; the software distributed by the club to the members can be used without restrictions within the country of each member.

3. Norms

Each member brings along with his program, his notations, his habits and his points of view; the norms are made to ease the use of his program by other users. Therefore they have the following purposes :

(1) Make each programmed code compatible with most computers.

(2) Make each programmed code as easy to use by others as possible. Since the same problems arised with the MODULEF library, we have used their experience and propose the following rules :

(a) — A subset of Fortran IV is used.

(b) — Each program is written in subroutine form and modular, *i.e.*, to the steps common to several problems (ex : triangulation, resolution of linear system) must correspond subroutines compatible if possible with the other subroutines of the library performing the same function.

(c) — Each program or subroutine performing a function which may interest other users must be documented separately.

3 (a). *Fortran compatible with most computers*

List of instructions allowed :

u = number of the logical unit : left as a parameter adjustable by the user.

f = Format label.

k = list of parameters

ASSIGN	READ f, k	EXTERNAL
BACKSPACE	RETURN	FORMAT
CALL	REWIND	FUNCTION
CONTINUE	STOP	INTEGER
DO	WRITE $(u, f)k$	LOGICAL
END FILE	WRITE $(u)k$	REAL
GO TO unconditional	BLOCK DATA	SUBROUTINE
GO TO imposed	COMMON	
GO TO calculated	COMPLEX	
IF arithmetic	DATA	
IF logic	DIMENSION	
PRINT f, k	DOUBLE PRECISION	
READ $(u, f)k$	EQUIVALENCE	

The non-executable instructions should be in the following order :

COMMON . . .

DIMENSION . . .

INTEGER, REAL, LOGICAL, DOUBLE PRECISION, COMPLEX

EQUIVALENCE

DATA

The programming should be independent of the size of the words.

The multi-indices arrays must be declared as follows in the subroutines :

SUBROUTINE SP (A, N1, N2, . . .)

DIMENSION A (N1, N2, 1) . . .

(except if N1 and N2 never-change)

No arrays with variable size are allowed in the list of a COMMON, only the identification of the variables.

If a COMMON is used in one module/subroutine, it must bear a name beginning with the first 4 letters of the name of the module/subroutine.

All initialization of a variable of a list of a labelled COMMON must be done by a DATA contained in a BLOCK DATA (IBM norm).

The use of an index of a Do-loop outside the loop must be done explicitly.

Thus

```
    DO 11=I1, I2
    IF(  )GO TO 2
  1 CONTINUE
  2 K=I+2
```

should be programmed

```
    J=I1
    IF (I1.GT.I2) GO TO 2
    DO 1I=I1, I2
    J=I
    IF (  ) GO TO 2
  1 CONTINUE
  2 K=J+2
```

because they are not interpreted in the same way on all computers.

Integer divisions mixed with arithmetic operations are not allowed :

Ex :

$$x = 2\ (1/2)$$

gives O. on IBM and UNIVAC and 1. on CII.

Print out and writes on tapes/disk should be optimal and controlable by the user. A good suggestion is to use a switch (IPRINT in the example below) on all PRINT statements; example :

```
SUBROUTINE DEMO (..., IPRINT, ...)
      ⋮
IF (IPRINT. GE. 1) PRINT 80, NS, NT, ...
      ⋮
IF (IPRINT, GE. 2) PRINT 81, [W(I), I =
    1, NS] ...
```

Data cards must be avoided as much as possible.

Data are assumed to be in core. The user will write is own routine to put the data in core.

3 (b). *Progress in subroutine and module form*

Each programmed code must be in subroutine form. Namely it is a set of subroutines, each subroutine performs a step in the program, reads as few data cards as possible and prints as little information as possible. However, if a switch is set in the arguments, printing and/or reading is allowed without restriction. This will allow an iterative use of the routine if necessary.

In most cases programs can be broken into independent steps. Each step performs a function. The set of subroutines which corresponds to each step is called a *Module*. Each module will be documented.

3 (c). *Programs in subroutine forms*

In most cases programs can be broken into independent steps. For example, the numerical solution of the Dirichlet problem.

$-\triangle u = f$ in Ω; $u = 0$ on the boundary of Ω by the finite element method can be decomposed into the following steps :

1. Triangulation of Ω,
2. Computation of the stiffness matrix and right hand side,
3. Solution of the linear system,
4. Visualization of the results.

Each step can be used in other contexts. Therefore it would be feasible on this example to write 4 subroutines with enough comments and documents to be usable by others.

All contributions are stored on a single tape, therefore everything should be in subroutine form (the mains are not stored).

Each subroutines which correspond to a step should be in the following form :

```
    SUBROUTINE name (argument 1, arg 2 ...,
                     arg n)
    COMMON
    DIMENSION ...
    INTEGER, REAL ...
    EQUIVALENCE
C   brief description of purpose
    ⋮
C   inputs description
    ⋮
C   outputs description
    ⋮
C   working arrays
    ⋮
C   written by ... on ...
    DATA
    ⋮
    END
```

The other subroutines which are not relevant to the user may be undocumented but they should contains a comment stating where they are called who programmed it and when.

3 (d). *Dynamic allocations*

The restriction that arrays must be declared as arguments of the subroutines makes the list of arguments rather long, and therefore, the subroutines are cumbersome to use. There is a way out of this difficulty.

All working arrays and input or output arrays which are not interesting to the user can be stored into one super array. For example, assuming that B, C, D are working arrays, the statements,

```
C   MAIN
    DIMENSION A(100), B(100), C(10, 100),
              D(200)
    ⋮
    CALL DEMO1 (A, B, C, 10)
    ⋮
    CALL DEMO2 (A, C, D)
    ⋮
    STOP
    END
    SUBROUTINE DEMO1 (A, B)
    DIMENSION A(1), B(1)
    ⋮
    END
    SUBROUTINE DEMO2
       (A, C, N, D)
    DIMENSION A(1), B(1),
       C(N, 1), D(1)
    ⋮
    END
```

to be stored in INTERLIB tape

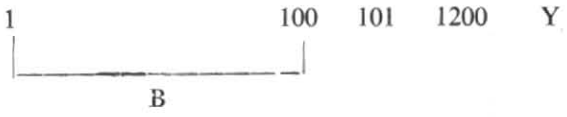
are equivalent to

```
C   MAIN
    DIMENSION A(100), Y(1200)
    ⋮
    CALL DEMOX1 (A, Y, 100)
    ⋮
    CALL DEMOX2 (A, Y, 10, 100)
    ⋮
    STOP
    END
    SUBROUTINE DEMOX1 (A, Y, M1)
    DIMENSION A(1), Y(1)
    CALL DEMO1 [A, Y(1), Y (M1 +1)]
    RETURN
    END
    SUBROUTINE DEMOX2 (A, Y, N, M1)
    DIMENSION A(1), Y(1)
    CALL DEMO2 [A, Y(1), N, Y, (N+M 1+1]
    RETURN
    END
    SUBROUTINE DEMO1 (A,B)
    ⋮
    SUBROUTINE DEMO2 (A,C,N,D)
    ⋮
    END
```

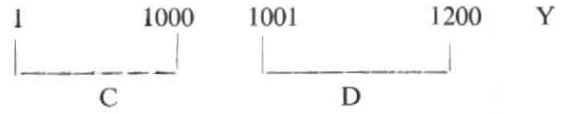
to be stored in INTERLIB tape

because when an argument passed in a subroutine is an array what is really passed to the compiler is the first element of the array.

This procedure has put C and B into Y with overlay. in the first call we have



in the second call we have



This trick also reduces the memory needed (1200 instead of 1300 in this example); it can also be used if B, C, D are double precision or integers and Y single precision provided that the space is computed according to the appropriate word length.

3 (e). *Multiple use of a subroutine*

In many cases a subroutine will be used inside a loop. Therefore the authors should make sure that the same work is not done unnecessarily after. There are at least two ways to solve this problem :

1. Insert a switch parameter KSW, for example, which the user puts to 0 if the routine is called for the first time and 1 otherwise (Hence certain parameters which were outputs become inputs on subsequent calls).
2. Use a subroutine for the initializations. This second method is to be used only if it is also a mean for the initialization of non-critical parameters. This can shorten significantly the list of arguments of a subroutine call and therefore make it easier to use for example, if EPS is a parameter that the author suggests to choose around 10^{-6} in most cases for DEMO one way for him is to include it in INIDEM as follows :

```
C   TEST
    ⋮
    CALL INIDEM
    CALL DEMO (A, B, C)
    ⋮
    STOP
    END
    SUBROUTINE INIDEM
    COMMON/INIXX/EPS
    EPS = 1.E — 6
    RETURN
    END
    SUBROUTINE DEMO (A, B, C)
    COMMON/INIXX/EPS
    ⋮
    END
```

to be stored in INTERLIB tape

This way EPS is completely ignored by most users. It the value is to be changed then it can be done by changing the MAIN to :

```
C   TEST
    COMMON/INIXX/EPS
    ⋮
    CALL INIDEM
    EP =1.E — 8
    CALL DEMO (A, B, C)
    ⋮
```

3 (f). *Documentation*

Each subroutine that corresponds to a step (previously defined) must be documented separately; the documentation is a few pages long only; it should be readable without further references and should contain the following information, in English :

1. Title, name, purpose
2. General description of the algorithm : mathematical background
3. Description of the calling statement : input, output, working arrays
   Restriction of use
   Programming considerations.
4. Name of programmer and date
   List of COMMON names and other SUBROUTINE called References for further details
5. Simple test program and output.

4. Contents of the library

The following modules are expected to be inserted in INTERLIB within a year :

Modules	*Contribution of*
1. Automatic triangulation of a 2-D domain	LAN of Paris VI
2. Visualization of results on a plotter	LABORIA
3. Solution of linear systems 1	LAN of Paris VI
4. Solution of linear systems 2	LAN of Novosibirsk
5. AP[1]-Finite Element Library : Solution of Laplaces' equation, solution of the biharmonic equation, solution of the Navier-Stokes eq, and Euler eq.	LABORIA
6. Optimization of a solar energy equipped house	LABORIA
7. Dispersion of pollutants in lakes	LAN of Novosibirsk
8. Inverse problems in electrocardiography	LAN of Pavia
9. Linear elasticity with linear finite elements	Ecole Polytechnique

Modules 1, 2, 5, 9 are already installed; they form the contribution of France to INTERLIB and we shall only describe these modules.

4.1. *Automatic triangulation of a 2-D domain : the library MEFISTO*

Triangulations of domains are necessary steps in the Finite Element Methods. Their purpose is to divide a domain into non-overlapping elements (triangles or quadrangles) and define on them sets of points called nodes. A triangulation is defined by a certain number of arrays stored in a file.

The MEFISTO Library contains 13 main subroutines making altogether 93 subroutines and 8700 instructions. They are fully described in *"Les Modules de maillage Bidimensionless du Club MODULEF"* (edited by the Laboria) and we shall only describe briefly their general features. In addition 3 subroutines for the triangulation of squares, circles and rings have been included; they may be handy for testing programs or for teaching purposes.

A brief description of MEFISTO

MEFISTO contains the following Modules for triangulations with quadrangles, and/or triangles :

Level 1

QUACOU : constructs a triangulation from the specifications of the nodes on the boundary divided into 4 pieces, by mapping the domain into a square.

TRIAUT : automatic triangulation of a domain from the specification of the nodes on the boundary, by the method of A. George.

COMEFI : reads the triangulation from data cards.

Level 2

AFFLOC : refines the triangulation around given points

RETRIA : divides each elements into N^2 elements

SYMDR : constructs a triangulation symmetric to a given triangulation with respect to a given line.

TRAROT : translates or rotates a given triangulation

RECOL : put together into one triangulation two given triangulations which may have common edges.

Level 3

ADPOIN : add N 1 nodes on each edge, ISET (resp. ISEQ) internal nodes in each triangle (resp. rectangles).

Level 4

RENCMK : renumber the nodes to minimize the bandwidth

RENEL : renumber the elements to minimize the bandwidth (for the frontal method)

Level 5

TRABE 1 : visualize a triangulation on a BENSON plotter

VISU 1 : visualize a triangulation on a 2250 IBM or TEKTRONIX screen.

All modules (of level 1) automatically store the outputs on an auxiliary memory (tape or disk on file number NFICi). The outputs are a set of arrays which are called the "Data Structure MEFI". Since it was

decided that the norms for the storage of a triangulation are those of MODULEF there is a last module, GENOPO, which reads MEFI and stores it in the "Data Structure NOPO" on the same auxiliary memory. The data structure NOPO is one big arry M divided into segments, each segment containing one of the arrays necessary to represent the triangulation. The data structure NOPO is described in the previous reference.

The modules of Level 2-5 read the data from MEFI.

An involved domain can be easily triangulated by the above modules if it is divided into simple subdomains.

The MEFISTO library was programmed by O. Koutchmy, P. Joly, A. Perronnet.

4.2. *A P^1 Finite Element Library*

These modules were developed for research and teaching purposes at Laboria. It uses very simple finite elements for the solution of the problems :

(1) $-\triangle u=f$ in Ω, $u \mid_{\Gamma_1}=u_0$,

$$\frac{\partial u}{\partial n} \mid_{\Gamma_2}=g \quad (\Gamma_1 \cup \Gamma_2=\Gamma=\partial\Omega,\ \Gamma_1 \cap \Gamma_2=\phi)$$

(2) $-\triangle u=f$ in Ω, $u \mid_{\Gamma}=u_0$, $\frac{\partial u}{\partial n} \mid_{\Gamma_2}=g$

(3) $\frac{\partial \triangle \psi}{\partial t}-\nu \triangle^2 \psi=\nabla \psi \times \nabla \triangle \psi$ in Ω,

$$\psi \mid_{\Gamma}=\psi_0,\ \frac{\partial \psi}{\partial n} \mid_{\Gamma}=g$$

(4) $\operatorname{div}\left[\alpha\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)+\beta \frac{\partial u_i}{\partial x_i}\right]=g+$ boundary conditions

The first problem is solved by

FEMICF or FEMICH

the second problem is solved in a mixed formulation by

DELT2 or *DELT2M*

the third problem is solved by an H^{-1} least square/optimal control method and a mixed p^1 approximation of ψ and of $-\triangle\psi=\omega$. The module is called :

NAVSTI

An other module called

EULER

solves (3) when $\nu=0$ by the method of characteristics on the vorticity equation.

The fourth problem, *i. e.*, the two-dimensional elasticity problem, is solved by

ELASI.

5. Visualization of results : The Library AM-BIBENS

This module is borrowed from the Library AM-BIBENS written by A. Marrocco at Laboria. It contains 8 main routines calling altogether 51 subroutines of a total of 4400 instructions. Two additional modules of visualization come from MEFISTO.

The routines are :

TRIANG : plotting of a triangulation (triangles only)

TRINUM : plotting of a triangulation (triangles only) with numbering

EQUI : plotting of the equipotential lines of a function given at the nodes of a triangulation (triangles only).

KOUPE : plotting of the trace of a function given at the node of a triangulation. The trace is on a straight segment.

LIGNES : plotting of the equipotential lines of a function, or of the lines orthogonal to the equipotential, which runs through a set of given points.

We can find also the routines TRACϕU (plotting of a curve), CϕUPE (an extension of KϕUPE), and TRF (plotting of boundaries).

TRABE1 : plotting of a triangulation stored in NOPO

VISU1 : visualization of a triangulation, stored in NOPO, on a Tektronix 4014 or IBM 2250.

If TRIANG, TRINUM, EQUI, KOUPE, LIGNES are used with a triangulation stored in NOPO, the routine NOPOPI must be used for the transfer of the information from NOPO into the central memory. The routines TRABE1 and VISU1 are described in : *"Les modules de Maillage Bi-dimensionnels du Club MODULEF"*, Laboria pub.

The routines of AM-BIBENS are described in "A two-dimensional visualisation software : AM-BIBENS" Interlib report, Oct. 79. As they must be used with a BENSON plotter but an other version with the same calling procedure exists for visualizations on the TEKTRONIX 4014.

6. Difficulties encountered and future development

The need for the free circulation of programs cannot be denied but the difficulties attached cannot be overviewed either.

The problem can be phrased in the usual words : "publish or perished". It is clear that if the author does not comply to the minimal norms explained in § 3 no one else will use his program. I have many instances of "these de 3eme cycle" which have perished because the work was to produce a program and the author did not documented it.

The basic difficulty that we found was to notice that even inside INRIA the programs did not circulate. Thus the first work to do and perhaps also the

most fruitful because it forced some self discipline of INRIA to include their work in INTERLIB. Now on an international level this is not so obvious. I beleive that two ways can be tried :

1. A journal publishes the programs of INTERLIB.

   Then the other members would be motivated because the extra work involved would count like a publication.

   There are already some journal which publishes programs but to my knowledge they do not impose any norm and seldom distribute a tape; as a result it takes quite a lot of work to try the programs.

2. INTERLIB becomes so popular that every one will want to join.

   While one can hope for the second solution, it may not be so wise to count on it. This year after the inclusion of the Russian contribution the situation will be summarized and it will be decided whether solution 1 is worth implementing and whether new members can be accepted.

Sample Test : Clamped plate; results on Fig. 2.

Find ϕ solution of

$\triangle^2 \phi = 1$ in $\Omega =$

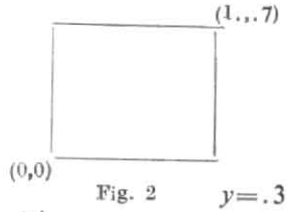


Fig. 2

$y = .3 \sin\left(\frac{\pi x}{2}\right)$

(2) $\phi \mid_{\Gamma} = \frac{\partial \phi}{\partial n}\Big|_{\Gamma} = 0$

The triangulation is done by QUACOU then (2) is solved by DELT2 (mixed FEM of order 1) and ϕ and $-\triangle\phi$ are plotted by EQUI. Then a cut off ϕ along the line passing through (0, 0) and (1, 1) is plotted by KOUPE. The triangulation is plotted by TRINUM.