A set of modules for the solution of integral equations

L. M. Delves, L. F. Abd-Elal* and J. A. Hendry†

Department of Computational and Statistical Science, University of Liverpool, Victoria Building, Brownlow Hill, PO Box 147, Liverpool L69 3BX

The provision of software for the solution of integral equations is complicated by the fact that the integral equations arising from practical calculations are very diverse in form, and usually fail to fit into any of the standard categories (Fredholm or Volterra first and second kind, Fredholm third kind) for which routines are normally provided. We describe here a set of ALGOL 68 modules which provide considerable flexibility in the types of equations which can be handled, while retaining reasonable run time efficiency. The modules yield a user interface which is recognisably close to that of the underlying mathematical formalism, and demonstrate the advantages of an extensible language such as ALGOL 68 in providing such a 'natural' interface without the need for an ad hoc preprocessor.

(Received February 1979; revised February 1980)

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024

1. Introduction

Numerical analysts, and writers of numerical software, normally consider integral equations as having a nice tidy classification into:

Volterra equations of the first kind

$$0 = y(s) + \int_0^s K(s, t, x(t))dt \quad (1)$$

Volterra equations of the second kind

$$x(s) = y(s) + \int_0^s K(s, t, x(t))dt \quad (2)$$

Linear Fredholm equations of the first kind

$$0 = y(s) + \int_a^b K(s, t)\, x(t)dt \quad (3)$$

Linear Fredholm equations of the second kind

$$x(s) = y(s) + \int_a^b K(s, t)\, x(t)dt \quad (4)$$

Linear eigenvalue problems (third kind Fredholm equations)

$$x(s) = 0 + \lambda \int_a^b K(s, t)\, x(t)dt \quad (5)$$

In addition, they recognise, and to some extent cater for, different classes of kernels K:

Smooth: $K(s, t)$ well behaved analytically over $[a, b] \times [a, b]$

Split (Green's Function):

$$K(s, t) = K_1(s, t)\ s \geqslant t$$

$$K_1, K_2 \text{ smooth}$$

$$= K_2(s, t)\ s < t$$

Convolution: $K(s, t) = K(s - t)$

Singular: $K(s, t)$ is unbounded on $[a, b] \times [a, b]$, and/or the range $[a, b]$ is not finite.

The commonest classes of singularity appear to be those of type:

Abel: $K(s, t) = K_0(s, t)\,(s - t)^{-\alpha}$

$$K_0 \text{ smooth}$$

Logarithmic: $K(s, t) = K_0(s, t)\, ln\, |\, s - t\, |$

To cater for all possible combinations of kernel type and equation type in this list, is a formidable task, and to date standard software is available, in libraries or as published algorithms, only for the following classes of problems:

Type 4 Smooth kernel (NAG routine DO5ABA/F, DO5CAB)

Type 4 Green's Function kernel (NAG routine DO5AAA/F)

Type 2 Smooth kernel (see, e.g., Phillips, 1978; Pouzet, 1963).

Of course, there may be published algorithms of which we are not aware; but the list is painfully incomplete, and to fill in the gaps would be a very large task.

Worse, it *would not be a particularly worthwhile task*, since in our experience most user problems fail to fit into *any* of the categories listed. This point has been made in Delves and Hendry (1976) and in Delves (1978); in a scan of the applications literature, the only 'standard' integral equation encountered was a first kind Fredholm equation. It was argued in these references that to obtain a reasonable coverage of user problems (4) with manageable cost (and bulk) of software, required a more flexible approach than that provided by self-contained routines such as (for example) DO5CAB.

This routine solves equations of the form of (4) by the Nystrom method. It requires that the user provide function segments for computing $K(s, t)$, $y(s)$, and makes some attempt at flexibility by letting him specify the quadrature rule (so that an infinite range can be handled). The routine then sets up the Nystrom equations, and solves them using Gauss elimination.

DO5CAB perform very badly on either of the following problems

$$x(s) = e^{\alpha s} - \lambda(e^{\alpha+\beta s} - 1)/(\alpha + \beta s) + \lambda \int_0^1 e^{\beta s t}\, x(t)dt \quad (6)$$

with $\alpha = \beta = 1$; $\lambda = 10^6$. Such a problem is very illconditioned, and should be solved by techniques appropriate to the first kind problem (3). The solution is $x(s) = e^{\alpha s}$

$$x(s) = y(s) + \int_0^1 ln\, |\, s - t\, |\, x(t)dt \quad (7)$$

$$y(s) = s - 0{\cdot}5\,[s^2\, ln\, s + (1 - s^2)\, ln\,(1 - s) - (s + 0{\cdot}5)]$$

(Baker, 1978 p. 537).

solution: $x(s) = s$.

This problem has a logarithmic singularity in the kernel, and a sum of singular terms in the driving function. Because of this latter sum, even a routine written for logarithmic kernels will fail to obtain an accurate solution of (7).

But we could describe these problems in stages, as follows.

Equation 6

(*a*) The interval is [0, 1]

(*b*) The operator equation has the form $[I - K]x = y$

(*c*) The kernel of K is a smooth function

(*d*) The driving term g is smooth

(*e*) The equation is likely to be illconditioned (for large $|\lambda|$)

Equation 7

(*a*) The interval is [0, 1]

(*b*) The operator equation has the form $[I - K]x = y$

*Department of Mathematics, University of Cairo.
†Computer Laboratory, University of Birmingham.

CCC-0010-4620/81/0024-0184 $03.50

© Heyden & Son Ltd, 1981

Table 1 Solution of the integral equation (7), using FAG 1

User programme [excluding output statements]

```
begin
        interval := finite (0, 1);
        proc g1 = (real x) real: (1·5* x + 0·25);
        proc g2 = (real x) real: (0·5* x * x);
        proc g3 = (real x) real: (0·5* (1 − x * x));
Problem errorestimate e;
description kvalue lhs, rhs;
        lhs := unitop (1·0) − fredholm (logsing (2));
        rhs := noparams g1 − noparams g2* logsing (1) − noparams g3*
               logsing (−1);
        for n from 3 by 2 to 9 do
Problem     [1:n] real x;
solution    linearsolve (lhs, rhs, x, e)
                    od
end
```

Solution obtained with FAG 1

N	3	5	7	9
actual error	$1{\cdot}7 \times 10^{-2}$	$3{\cdot}5 \times 10^{-12}$	1·1, −11	9·6, −12
estimated error	7·5, −2	2·1, −11	3·4, −11	4·8, −11

Solution from Baker (1978) using a modified Nystrom method with N equally spaced points

N	2	4	8	16	32	64
error	1·3, −2	3·6, −3	1·0, −3	2·7, −4	7·1, −5	1·8, −5

(*c*) The kernel of K is $ln \mid x - y \mid$

(*d*) The driving term is $y_1 + y_2\, ln\, s + y_3\, ln\,(1 - s)$ where y_1, y_2, y_3 are smooth.

(*e*) The equation is well conditioned.

We have written a pilot set of modules ('FAG 1') which essentially allows the user to describe his problem in these stages. It contains separate modules for handling the unit operator I and the Fredholm integral operator K, for example, together with a mechanism for defining the sum, difference, and product of operators so that nonstandard forms of equations can be constructed and solved. It also attempts to handle singular problems effectively where possible; the package contains a library of standard singular functions and kernels, such as $ln(1 - s)$, $ln \mid s - t \mid$, and allows the user to add, subtract or multiply both these standard functions and his own user defined functions, to give a complete description of the kernel and driving term in his equations. Finally, it contains a number of solution modules; the user is asked (forced!) to choose the one appropriate to his equation. The code which the user would write to solve equation (7) is shown in **Table 1**, together with the results obtained, and those obtained for the same equations by Baker (1978) using a method designed to cope with the singular kernel. These results demonstrate that flexibility need not imply inefficiency.‡

We give in this paper a brief description of FAG 1 as it currently exists, together with some simple examples of its use. FAG 1 yields an implementation in ALGOL 68 of the Fast Galerkin technique and associated error estimates (Delves, Abd-Elal and Hendry, 1979; Babolian and Delves, 1979); however the choice of underlying algorithm is only important here insofar as it and its error estimates can be extended to cover the class of equations considered. The purpose of the present paper is to demonstrate that the difficulties of handling a wide class of user problems can be at least to some extent overcome by a careful design of the user interface, and to discuss how this interface can be implemented in a relatively straightforward manner using the standard facilities of ALGOL 68.

The description given is pedagogical in style and does not presuppose a knowledge of ALGOL 68; it is intended to demonstrate by example the advantages of an extensible language in developing natural, simple to use and portable software packages without the need for application oriented language preprocessors. The description is also deliberately simplified in various places; 'frills' which exist in the code but are inessential to its overall structure, have been omitted from the description.

2. Class of problems treated

2.1 *Class of equations*

Equation (7) can be written in the operator form

$$[I - K]\, x = g \qquad (7')$$

where I is the unit operator and K a Fredholm integral operator acting in an underlying space R. This space has as elements functions $f(s)$ defined on the interval $a \leqslant s \leqslant b$, and possessing 'suitable' smoothness properties. The required properties depend on the algorithm used to solve (7); for the Fast Galerkin method we require, roughly speaking, that the Chebyshev expansions generated all exist. The package FAG 1 is designed to handle more general equations of the form

$$\text{Alg}\,[I, K_1 - K_n]\, x = \text{Alg}\,(g, -g_m) \qquad (8)$$

where $K_1 - K_n$ are (possibly nonlinear) integral operators on $[a, b]$; $g_1 - g_m$ are functions on $[a, b]$ which may also depend on the solution x; and Alg denotes an arbitrary algebraic expression formed using the elementary operations +, −, * between operators or between functions. The operators K currently available in FAG 1 are listed in **Table 2**. With their use, the form (8) clearly encompasses all of the 'standard' equations (1)-(5), and in addition a wide variety of non-standard relatively common forms; for example the linear

‡To be fair to the technique being demonstrated by Baker, the rapid convergence obtained for equation (7) is *untypical* of what might be expected for a logarithmic kernel, since it depends on strong cancellation of singularities in the kernel and driving term. However, the example does demonstrate that FAG 1 not only handles the singularities effectively and obtains the cancellation, but recognises, via its error estimate, that it has done so.

© Heyden & Son Ltd, 1981

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024

Table 2 Operators currently available in FAG 1

Operator φ	$(\varphi x)(s)$	*Comments*
Unit (λ)	$\lambda x(s)$	Only given a parameter to avoid a compiler bug
Fredholm (K)	$\int_a^b K(x; s, t)\, x(t)dt$	
Volterra (K)	$\int_a^s K(x; s, t)\, x(t)dt$	
Inverse Volterra (K)	$\int_s^b K(x; s, t)\, x(t)dt$	
Green's Function (K_1, K_2)	$\int_a^b K(x; s, t)\, x(t)dt$ $K = K_1(s, t)$, $t < s$ $= K_2(s, t)$, $t > s$	Equivalent to Volterra (K_1) + Inverse Volterra (K_2)

Table 3

FAG 1 function	*Parameter values*	*represents the function*	*comments*
logsing (n)	-1	$ln[b - s]$	
	0	$ln\left\lvert s - \left(\frac{a+b}{2}\right)\right\rvert$	
	1	$ln(s - a)$	
	2	$ln \lvert s - t \rvert$	
algsing (n, α)	-1, α	$(b - s)^\alpha$	$\alpha > -0{\cdot}5$
	0, α	$\left\lvert s - \frac{a+b}{2}\right\rvert^\alpha$	$\alpha > -0{\cdot}5$
	1, α	$(s - a)^\alpha$	$\alpha > -0{\cdot}5$
	2, α	$\lvert s - t \rvert^\alpha$	$\alpha > -0{\cdot}5$

Table 4 The types of user defined function acceptable to FAG 1

Here, **mode vector** = **ref** [ ] **real**; the vector **a** represents the discretised solution, α a row of input parameters

Mode	*Representing*	*Usage*
Proc (real) real	$g(s)$	**noparams** g
Proc (real, vector) real	$g(s, \alpha)$	g **withparams** α
	$g(s, \mathbf{a})$	**noparams** g
Proc (real, vector, vector) real	$g(s, \alpha, \mathbf{a})$	g **withparams** α
Proc (real, real) real	$K(s, t)$	**noparams** K
Proc (real, real, vector) real	$K(s, t, \alpha)$	K **withparams** α
	$K(s, t, \mathbf{a})$	**noparams** K
Proc (real, real, vector, vector) real	$K(s, t, \alpha, \mathbf{a})$	K **withparams** α

equation

$$x(s) = g(s) + \lambda \int_a^b K(s, z) \int_a^b K(z, t)\, x\,(t)dt\, dz \qquad (9)$$

corresponding to the operator equation

$$[I - \lambda K * K]\, x = g\,, \qquad (9')$$

2.2 *Class of kernels K and driving terms g*

The kernel function (s) $K(x; s, t)$ arising in integral equations are often singular, or have singular derivatives of low order, at one or more points in the domain $a \leqslant s, t \leqslant b$; the driving function may also be singular in nature. Examples are given by the logarithmic kernel of (7), which is singular along the line $s = t$; and by a typical Green's function kernel (see Table 2) which is normally continuous but with discontinuous partial derivatives, along this line. Such singularities lead to slow convergence unless the numerical method used deals with them explicitly. In order for this to be possible, FAG 1 allows the user to define both kernel functions and driving functions as algebraic combinations of simpler terms:

$$K(x; s, t) = \text{Alg}\,(K_1 - K_p\,, g_1 - g_q) \qquad (10)$$
$$g(x; s) \quad = \text{Alg}\,(g_{q+1} - g_m)$$

where the terms $K_i\,(x; s, t)$ and $g_i(s)$ are of two types: (*a*) user defined functions, which may be nonlinear (that is, depend upon the (unknown) solution x), and which FAG 1 hopes are *smooth* (*b*) library functions containing singular factors which FAG 1 is able to treat efficiently. **Table 3** contains a list of the one and two dimensional singular functions kown to FAG 1, together with an indication of how they are accessed; their use is considered in more detail in **Table 4**.

Equation (7) yields a simple example of equation (10); the kernel has only a single, standard singular term, while the driving term is an algebraic expression composed of the functions:

smooth: s, $-0{\cdot}5\, s^2$, $-0{\cdot}5(1 - s^2)$, $0{\cdot}5(s + 0{\cdot}5)$

singular: $ln\, s$, $ln\,(1 - s)$

3. Kernel and driving term description

We now give a bottom-up description of the way in which this class of problems can be conveniently described, using the standard facilities of ALGOL 68.

3.1 *Description of the solution*

Since the user-defined kernel and driving terms may depend on the solution x, it is necessary to consider its representation

© Heyden & Son Ltd, 1981

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024

first; this depends on the underlying algorithm used. In FAG 1, this is an expansion technique (Delves, Abd-Elal and Hendry, 1979) in which the interval $[a, b]$ is first mapped to $[-1, 1]$ and then $x(s)$ expanded as a Chebyshev series:

$$z = z(s) \qquad -1 \leqslant z \leqslant 1 \tag{11}$$
$$x(s) = X(z)$$
$$X(z) \simeq X_N(z) = \sum_{j=0}^{N} a_j T_j(z)$$

The solution $x(s)$ is therefore represented by the vector $\mathbf{a} = \{a_i, i = 0 - N\}$, together with the mapping $Z(s)$. This latter is specified (see Table 1) by the line

interval: = finite (0, 1);

Here, interval is a globally declared variable structure which is assumed by FAG 1 to contain procedures to evaluate $z(s)$, $z^{-1}(s)$, $z'(s)$; it has the ALGOL 68 declaration

mode vector = **ref** [ ] **real**;
mode map = **struct** (**proc** (**real**, **vector**) **real** *map*, *inverse*, *derivative*,
vector *data*);
map *interval*;

The value of interval is set by a call of the procedure *finite*, which takes as parameter the data of the map and delivers as result a structure of mode **map**. *finite* is a FAG 1 procedure implementing a simple linear map; others (for infinite intervals, for example) can be predefined or user provided.

Note that this construct makes use of the ALGOL 68 treatment of procedures as 'just another mode'; procedures can be stored away in structures or delivered as results by other procedures, for example. We have leant heavily on this convenient facility.

3.2 *Treatment of individual terms*

A kernel or driving function is in general (see equation (10)) an algebraic expression formed from a number of terms. The terms may be:

a user defined function	in one or two variables
a singular library function	with or without parameters
a constant	dependent or not dependent on the solution

The ALGOL 68 modes of the various possible defined functions are listed in Table 4: FAG 1 requires that objects of any of these modes be accepted and stored, together with data parameters where relevant, for later use. ALGOL 68 provides the convenient concept of **unions** to deal systematically with objects which come in different variants.

We therefore declare a generalised procedure mode;

mode kproc = **union** (all of the modes listed in Table 4, plus others for constant terms and for singular library functions)

We can now store individual terms in a structure containing the procedure, its parameters (if any) and any other information that might be useful later:

mode kstruct = **struct** (**kproc** *proc*, **vector** *params*, **etc** etc);

It then only remains to provide a painless way for the user to put his functions into such structures. In FAG 1, this is carried out by the FAG 1 operators **withparams** and **noparams**, declared as

op noparams = (**kproc** *p*) **kstruct**
op withparams = (**kproc** *p*, [ ] **real** *params*) **kstruct**

and used as indicated in Tables 1 and 4. The library routines logsing and algsing (see Table 3) also return a result of mode **kstruct** when called. Constant terms are treated separately for efficiency; and hence an individual term k_i or g_i in equation (10) may be either a (real) constant or a **kstruct**:

mode kterm = **union** (**real**, **kstruct**);

3.3 *Forming algebraic expressions (kernels and driving functions)*

A complete kernel or driving term may be a single term (e.g. the kernel in Table 1) or an algebraic expression; we now have to consider the latter. Fortunately, this is relatively straightforward in ALGOL 68 (even for a numerical analyst) since meaning can be given to the usual operator symbols +, −, * between operands of any user defined types and new operator symbols declared if required, as for **withparams** and **noparams** above. We require that the operators produce a suitable *linked list* which can be traversed later to recover the expression. This may be done in a number of ways; that implemented in FAG 1 makes use of the standard ALGOL 68 operator precedence for +, −, * to do most of the sorting out. If we consider the expression for the driving term in Table 1:

$$g = g_1 - g_2 * h_1 - g_3 * h_2 \tag{12}$$

and split this at any operator into a left part and a right part, we see that either of these may be a single term, or a (simpler) algebraic expression. We characterise each such node by the operator at the node, and by its left and right parts:

mode kvalue = **union** (**ref klink**, **kterm**)
(an expression may be a single term, or a compound expression of mode **klink**)
mode klink = **struct** (**kvalue** *left*, *right*, **int** *op*)
(where op = 1, 2, 3 stands for +, −, *)

We can now define the operators +, −, *; for example

op + = (**kvalue** *left*, *right*) **ref klink**
(**heap klink** *result* := (*left*, *right*, 1));

The code defining equation (12) is shown in Table 1; it yields a **kvalue** structure of the form

$$g = ((g_1, (g_2, h_1, 3), 2), (g_3, h_2, 3), 2)$$

corresponding to the parse tree shown in **Fig. 1.**

Note that the nesting is automatically correct, with multiplications coming first, because the operator * has higher priority than +, − and hence is obeyed first when the program of Table 1 is run.

Because the same mode objects are used to hold one variable and two variable functions, kernel expressions can be set up in the same manner. Moreoever, since one variable functions can occur in kernel expressions, FAG 1 allows the intermixing of one and two variable functions. Thus it is possible for the user to write code fragments such as (with notation as in Table 4):

noparams $g \otimes$ **noparams** K or **noparams** $K \otimes$ **noparams** g

where $\otimes$ = +, − or *.

The first of these is taken to mean $g(s) \otimes K(s, t)$; the second,

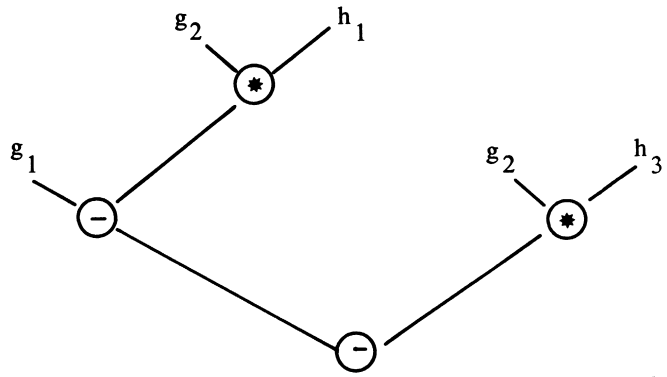

Fig. 1 Structure generated by FAG 1 for the driving function of Table 1

© Heyden & Son Ltd, 1981

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024

$K(s, t)\otimes g(t)$. It is the user's responsibility, however, to ensure that the driving term which he sets up is a function only of one variable (FAG 1 will fail him at solution time if he makes a mistake).

3.4 *Defining integral operators*

Integral operators (and the unit operator) are defined by calling library procedures which take as arguments a kernel description (of **mode kvalue**) and return a result of this mode; for example, a Fredholm operator with kernel $K(s, t)$ defined by the call

Fredholm (K)

To see what the routine Fredholm accomplishes, it is necessary to describe the underlying solution method. We consider for simplicity a linear Fredholm equation of the second kind in the mapped space $[-1, 1]$:

$$X(z) - \int_{-1}^{1} K(z, t)\, X(t)\, dt = g(z)$$

Introducing the expansion (11), multiplying by $(T_i(z)(1 - z^2)^{-\frac{1}{2}}$ and integrating we obtain the standard Galerkin equations for the coefficients **a**:

$$[D - K]\,\mathbf{a} = \mathbf{g}$$

where D is a diagonal matrix with elements

$$D_{ii} = \int_{-1}^{1} \frac{T_i^2(z)dz}{(1 - z^2)^{\frac{1}{2}}} = \begin{matrix} 2 \\ 1 \end{matrix}\Bigg\} \begin{matrix} i = 0 \\ i > 0 \end{matrix} \qquad (13)$$

and

$$K_{ij} = \int_{-1}^{1} dz \int_{-1}^{1} dt\, T_i(z)T_j(t)\, \frac{K(z, t)}{(1 - z^2)^{\frac{1}{2}}} \quad 0 \leqslant i, j \leqslant N \qquad (14)$$

$$g_i = \int_{-1}^{1} \frac{g(z)\, T_i(z)\, dz}{(1 - z^2)^{\frac{1}{2}}} \qquad 0 \leqslant i \leqslant N$$

Table 5 Solution of a problem containing the product of two singularities

Equation (Baker, 1978)

$$x(s) = y(s) + 2/3 \int_{-1}^{1} |s - t|^{-\frac{1}{2}}\, x(t)dt$$

$$y(s) = (1 - s^2)^{\frac{1}{2}} - \pi\sqrt{2}\,(2 - s^2)/4$$

Solution: $x(s) = (1 - s^2)^{\frac{1}{2}}$

The singular function $(1 - s^2)^{\alpha}$ is not available in FAG 1, but $(1 - s)^{\alpha}$ and $(1 + s)^{\alpha}$ are. FAG 1 treats the product of two singular functions as a special case, in an attempt to retain accuracy. The following program demonstrates this.

```
begin
 errorestimate e;
 interval := finite (−1, 1);
 proc g1 = (real s) real: (pi * sqrt(2) * (2 − s * s)/4);
 kvalue lhs, rhs;
 lhs := unitop(1) − (2/3) * fredholm (algsing (2, −0·5));
 rhs := algsing (−1, 0·75) * algsing (1, 0·75) − noparams
  g1;
 for n from 3 by 2 to 15 do
  [1:n] real x;
  linearsolve (lhs, rhs, x, e)
  od
end
```

Results for FAG 1

N	3	5	7	9	11	13	15
error	4·4, −1	3·6, −3	1·5, −3	8·7, −4	3·6, −4	4·1, −4	1·6, −4
estimated	2·1, −1	1·4, −3	1·8, −3	1·3, −3	9·5, −4	7·1, −4	5·5, −4

Results from Baker (1978) using a modified Nystrom method

N	10	20	30	40	50	60	70
error	6·2, −2	2·3, −2	1·2, −2	7·7, −3	5·5, −3	4·3, −3	3·4, −3

Apart from a constant factor, the elements g_i are identical to the expansion coefficients in a Chebyshev expansion of the driving function g:

$$g(z) = \frac{2}{\pi} \sum_{i=0}^{\infty}{}' g_i T_i(z) \qquad (15)$$

FAG 1 computes the coefficients g_i at solution time, from Chebyshev expansions of the individual terms in the algebraic expression given for $g(s)$. Similarly, the coefficients K_{ij} are related to the Chebyshev expansion of the modified kernel $K(1 - t^2)^{\frac{1}{2}}$

$$K(z, t)(1 - t^2)^{\frac{1}{2}} = \frac{4}{\pi^2} \sum_{i,j=0}^{\infty}{}' K_{ij} T_i(z) T_j(t) \qquad (16)$$

These coefficients will be evaluated at solution time from the algebraic expansion given by the user for K. All that the procedure Fredholm need do, therefore, is to (formally) multiply the algebraic expression by $(1 - t^2)^{\frac{1}{2}}$ to ensure that the correct matrix is generated. A similar situation holds with Volterra or inverse Volterra generators, save that the singular function needed is modified. Thus, for a Volterra equation, the factor $(1 - t^2)^{\frac{1}{2}}$ is replaced by $(1 - t^2) \times h_r(z, t)$, where

$$h_r(z, t) = 1, \quad z > t$$
$$= 0, \quad z \leqslant t \qquad (17)$$

Finally, algebraic operator expressions are handled using the same mechanism as kernel and driving term expressions, save that a separate flag (op = 4) and corresponding operator (**op***) is required for operator products.

4. Setting up and solving the equations

There are solution modules in FAG 1 for solving linear or non-linear, illposed or wellposed, inhomogeneous equations; there is currently no eigenvalue solver, though it would be simple to add one. Since nonlinear problems are solved as an iterative sequence of linear problems, each of these routines has as its major task the production of the Galerkin matrices (13) and (14). The numerical techniques used are described in Delves, Abd-Elal and Hendry (1979). The matrices and vectors required are formed from the Chebyshev expansions of the constituent functions, user defined functions being expanded numerically using the Fast Fourier Transform and singular (library) functions being expanded analytically using a set of recurrence relations. The final equations are pieced together during a recursive scan of the integral operator and driving term structures; however, a preliminary scan is made first to effect one important optimisation. FAG 1 is able to multiply a rapidly convergent expansion (one resulting from a smooth kernel or function) by a slowly convergent expansion (resulting from a singular kernel, or from an 'induced' singularity such as is represented by the factor $(1 - t^2)^{\frac{1}{2}}$) without loss of accuracy. There is always one such singular function, induced by the treatment of the integral operators; if there is a second singular factor, FAG 1 will absorb the two so that they are treated as a single compound factor. Accuracy is therefore lost only when two library singularities are explicitly multiplied; even then, the result is better than would be obtained by coding the singular factor as a user function. For an example in which two

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024

© Heyden & Son Ltd, 1981

singular factors are multiplied, see **Table 5**. Once the matrix equations are formed, their solution is straightforward. FAG 1 distinguishes between wellposed problems (as exemplified by equations (2) and (4)) and illposed problems (exemplified by (1) and (3)). The routines *linearsolve*, *illposed solve*, implement respectively the iterative scheme of Delves (1977), and the augmented Galerkin scheme of Babolian and Delves (1979).

5. Error estimates

Errors in a Galerkin calculation contain two main components:

1. Truncation errors stemming from the use of a truncated basis.
2. Quadrature errors: the matrices are computed numerically only to finite accuracy.

The Fast Galerkin scheme provides estimates for both sources of error. FAG 1 keeps these estimates for each operator, and estimates from them the total error resulting from individual operations $(+, -, \times)$ on two operator matrices. It finishes up with an error estimate for the user defined equation, which is again in two parts: the truncation error associated with the final $N \times N$ matrix, and the error in each of its elements. From these, FAG 1 produces an overall estimate of the error in the solution, which it returns to the user (it also returns a flag if it has any reason to suspect that the error estimate may be unreliable). The results shown in Tables 1 and 5 indicate that these error estimates, which are very cheap to produce, are reasonably successful, being usually slightly pessimistic but within a factor 10 of the true error.

The module *illposed solve* also uses the error information associated with the matrices to produce an error estimate, although for illposed problems this is very difficult to do and the estimate is largely heuristic. More importantly, it is possible for problems to be posed which have no solution. A feature of the augmented Galerkin method is that it generates a numerical criterion for the existence of an L^2 solution of the problem being solved. *illposed solve* implements this criterion, and although it will always return a computed solution $x(s)$ to the user, it returns also an opinion as to whether a genuine solution to the problem exists or not. This criterion has proven rather successful in practice (so far). An example of the use of *illposed solve* to solve a first kind Fredholm equation is given in **Table 6**.

6. Discussion

We may discuss the capabilities of FAG 1 on two levels: as a package for solving integral equations, or as a demonstrator of the virtues of a modular approach to numerical software, and of an extensible language such as ALGOL 68 for its implementation.

Viewed as an integral equation solver, FAG 1 is reasonably successful. Although its capabilities are still rather limited, it shows that a modular approach of this sort to the solution of integral equations is both feasible and useful; it is easy to use and accepts a wide variety of problems. Its achieved accuracy is startlingly high on some problems, and as good (so far as we have comparisons) as that of special purpose routines on others, for a fixed discretisation length N. Although its operation count is $\phi(N^2 \, l_n N)$, the Galerkin scheme is still rather slower (by a factor almost 2), over the range $N < 50$, than the nominally $\phi(N^3)$ Nystrom scheme with direct solution of the equations for wellposed problems (see Riddell and Delves (1979) for comparative timings). However, it repays this additional cost (which would be reduced with a faster FFT module) by providing an effective and usually quite realistic error estimate. For illposed problems, comparative timings are more difficult to obtain; again, the augmented Galerkin scheme used here is slower than would be a regularisation scheme with a Nystrom discretisation of the regularised equations; but it is certainly quicker than the method recommended as best in a review by Lewis (1975). More important than speed however is stability for such problems; and the method performs well in this respect.

Some of the limitations of the current versions of FAG 1 can be easily removed. For example, we plan to add in due course a linear eigenvalue solver, and to add to the library of singularities.

We would also like to add differential operators and their associated boundary conditions, giving an ability to solve integrodifferential equations; the algorithm described in Babolian and Delves (1979) represents one way of doing this. Other limitations are less easy to remove: there are no plans to treat multidimensional problems. Finally, some are straightforward but would cost considerable effort to remove. The chief extension of this type, which would add considerably to the usefulness of the package, is the ability to handle coupled equations. Another desirable extension would be the ability to split the range $[a, b]$, computing independent expansions on several subranges. In both these respects, FAG 1 is much more primitive than current counterparts in initial value ordinary differential equations or numerical quadrature, although probably no more so than available boundary value o.d.e. routines.

Viewed as a demonstrator of the usefulness of an extensible language for the production of numerical software, we believe FAG 1 to be quite persuasive. Its user interface is pleasant and natural, and is achieved entirely within the standard language; further, the package was not especially difficult to write, although some of the programming techniques involved are perhaps mildly unfamiliar to a traditional numerical analyst.

Table 6 A first kind Fredholm equation

Problem

$$\int_0^1 K(s, t)x(t)dt = e^s + (1 - e)s - 1$$

$$K(s, t) = t(s - 1) \quad t < s$$

$$= s(t - 1) \quad t \geqslant s$$

Solution: $x(s) = e^s$

Program

```
begin
  errorestimate e; [ ] int nuse = (3, 5, 7, 10, 15, 20);
  interval := (0, 1);
  proc k1 = (real s, t) real: (t * (s − 1));
  proc k2 = (real s, t) real: (k1 (t, s));
  proc y = (real s) real: (exp(s) + (1 − exp(1)) * s − 1);
  for i to 6 do
   [1 : nuse [i]] real x;
   illposedsolve (Greens function (noparams k1, noparams k2),
                                   noparams y, x, e)
  od
end
```

Comments

The solution quickly reaches a minimum error of 10^{-7}, and then remains stable: a feature of the augmented Galerkin method.

Solution

N	3	5	7	10	15	20
error	1·9, −1	4·9, −2	1·6, −4	2·2, −7	8·7, −7	9·4, −7
solution exists?	yes	yes	yes	yes	yes	yes

© Heyden & Son Ltd, 1981

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024

Acknowledgements
We would like to thank E. Babolian and I. Barrodale for many useful discussions on the solution of illposed problems.

References

BABOLIAN, E. and DELVES, L. M. (1979). An Augmented Galerkin Method for First Kind Fredholm Equations, *JIMA*, Vol. 24, pp. 157-174.

BAKER, C. T. H. (1978). *The Numerical Treatment of Integral Equations*, Oxford University Press.

DELVES, L. M. (1977). A Linear Equation Solver for Galerkin and Least Squares Methods, Algorithm 96, *The Computer Journal*, Vol. 20, pp. 371-374.

DELVES, L. M. (1978). Numerical Software for Integral Equations, Contribution to *Numerical Software—Needs and Availability*, ed. D. Jacobs, Academic Press.

DELVES, L. M., ABD-ELAL, L. F. and HENDRY, J. A. (1979). A Fast Galerkin Algorithm for Singular Integral Equations, *JIMA*, Vol. 23, pp. 139-166.

DELVES, L. M. and HENDRY, J. A. (1976). A Package for Solving Integral Equations, Internal Report, Department of Computational and Statistical Science, University of Liverpool (unpublished).

LEWIS, B. A. (1975). On the Numerical Solution of Fredholm Integral Equations of the First Kind, *JIMA*, Vol. 16, pp. 207-220.

PHILLIPS, C. (1978). An Improved Algorithm for the Solution of Volterra Integral Equations of the Second Kind by Runge-Kutta Processes, Research Report, Dept of C.S.S., University of Liverpool.

POUZET, D. (1963). Methode D'Integration numerique des equations Integrales et alegro differentielles du type de Volterra de seconde espece. Formules de Runge-Kutta, in *Symposium on the Numerical Treatment of Ordinary Differential Equations, Integral and Integro-differential equations*, Birkauser Verlag, Basel, pp. 362-368.

RIDDELL, I. and DELVES, L. M. The comparison of routines for solving integral equations, *The Computer Journal*, Vol. 23 No. 3, pp. 274-285.

Book reviews

Principles of Database Systems by J. D. Ullman, 1980; 379 pages. (*Pitman*, £12·50)

The author's approach is similar to that of, for instance, C. J. Date or A. F. Cardenas; his book differing from theirs mainly in including more up to date material. It begins with an introductory survey of the objectives and underlying ideas of data base systems, followed by a useful summary of relevant file storage and access strategies (including a lucid account of *B*-trees). Then comes an extensive treatment of the relational, network, and hierarchical data models and their various implementations, and finally two chapters on security, integrity, and the handling of concurrent accesses to a data base.

Of the data models, the relational is given the lion's share of attention; there are chapters on: the algebra and calculus and their manifestations in query languages such as ISBL, SEQUEL, and Query-by-Example; relational normalisation; and the optimisation of query language expressions to minimise the inefficiency often associated with the standard relational operators. In this area the author draws upon a number of recently published research papers and puts the proposed algorithms into context admirably. His treatment of DBTG systems and IMS (the sole representative, as usual, of the hierarchical data model) is much more cursory.

I found a great deal of interest in the book but I should hesitate to recommend it to a complete beginner in the subject since, in spite of being based on an introductory course offered at Princeton, it is hardly a teaching text. The author comments that in the past it has been difficult, when teaching about data base systems, to achieve "the mix of principles and practice" well established in other branches of computer science. An understanding of the principles is obviously assisted by practice, and now that students are being given the opportunity to use real data base systems they will initially need simple, specific and practical guides rather than the fairly hefty volumes produced by Date, Cardenas, James Martin and the current author, all of whom attempt to cover the whole of a very wide field.

I wonder, incidentally, whether one in a hundred readers of books of this kind actually does the exercises so copiously supplied at the end of each chapter? The most industrious learner must baulk at constructing complex query language expressions, or even programs, which he will never have the chance to test!

S. JONES (London)

Mathematical Methods in Computer Graphics and Design edited by K. W. Brodlie, 1980; 147 pages. (*Academic Press*, $23.00, £9·80)

This book contains an account of the proceedings of a conference organised by the Institute of Mathematics and its Applications (IMA) at the University of Leicester in September 1978. Its intention was to bring together those developing graphical algorithms and those likely to use them.

Six papers are included together with brief summaries of the discussions, two each on curve drawing, contouring and geometric problems in design. On curve drawing, Brodlie himself reviews algorithms where the curves are required to pass through the data points, and McLain treats least-squares fitting of curves to data which are subject to error—rather confusingly he refers to this as interpolation.

The contributions on contouring each assume precomputed data points—rectangular and skewed rectangular grids by Sutcliffe, and scattered points by Sabin. If a reviewer may presume to state an interest, there may sometimes be an advantage in choosing the data points dynamically while contour following so as to economise in the number of height determinations (Reeves, 1980).

The paper by Forrest on the three problems of nearest neighbour searching, intersections of bodies and hidden surface computation, is of particular interest in relating the complexity of algorithms to the choice of suitable structures in which to embed the data. Finally, Braid, Hillyard and Stroud in a paper rather too dense for an innocent reader discuss the stepwise construction of polyhedra in geometric modelling.

The book is carefully produced and contains numerous references to the work reviewed.

C. M. REEVES (Keele)

Reference

REEVES, C. M. (1980). A procedure for economical curve tracing, *IUCC Bulletin*, Vol. 2 No. 1, pp. 37–42.

© Heyden & Son Ltd, 1981

Downloaded from https://academic.oup.com/comjnl/article/24/2/184/338251 by guest on 03 April 2024