The comparison of routines for solving Fredholm integral equations of the second kind

I. J. Riddell and L. M. Delves

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

We describe a test package for the comparative testing of automatic and non-automatic integral equation solving routines, and give results obtained for a number of recent routines for solving linear Fredholm equations of the second kind. The methodology used is based on that described by Lyness and Kaganove (1977) for automatic quadrature routines.

(Received September 1977; revised October 1978)

1. Introduction

As the number of routines for solving integral equations increases, it would be useful, for the following reasons, to have a method of comparing routines:

(*a*) to help a user to decide which routine(s) is best for his particular problem (the 'best' routine will not necessarily be the same for different types of problem)

(*b*) to assist in the development of new routines by evaluating any improvements made by the programmer.

In this paper we consider the comparative testing of routines for solving linear Fredholm integral equations of the second kind, although the method is applicable to other types of integral equation solving routines. A linear Fredholm integral equation of the second kind has the form

$$f(x) = g(x) + \int_a^b K(x, y)f(y)dy \qquad 1.1$$

where $g(x)$, the driving term, and $K(x, y)$, the kernel function, are known and we wish to approximate the unknown function $f(x)$. There are two broadly different types of routine for solving the above equation:

1. *Non-automatic routines* where the user specifies a value of N (usually the number of points used in the quadrature rule which replaces the integral in 1.1) and the routine delivers a solution which corresponds to this value of N.

2. *Automatic routines* where the user specifies the accuracy to which he requires his solution and the routine attempts to satisfy this accuracy requirement.

We describe here a test package for comparing both automatic and non-automatic routines, and some preliminary results obtained with it. The test methodology used is a direct extension of that used in the automatic quadrature test package of Lyness and Kaganove (1977). We summarise this methodology, and the reasoning behind it in Section 2 and a description of the problem families used is given in Section 3. In Section 4 we give a brief description of the comparison package for non-automatic routines and show some results. The extension to, and results for, automatic routines are given in Section 5.

2. Commonly used comparison techniques, problem families and performance profiles

Test packages for comparing competing routines have proven useful in a number of fields, e.g. nonlinear optimisation, numerical quadrature, solution of ordinary differential equations. Most of the test packages which have been developed use a common methodology, the 'Battery' method. This approach considers a set of problems which have known solutions and vary in difficulty from easy to difficult. Each problem is solved by the routine under consideration, perhaps several times with different input parameters (in the case of non-automatic routines we use several different values of N and in the case of automatic routines several values of the input tolerance). Certain data, such as the mill-time taken, the actual error and, usually only in the case of automatic routines, the number of function evaluations, are collected at the end of each run. Finally, these values are averaged out so as to give, hopefully, an indication of how the routine would cope with a 'typical' problem and the average values are compared with values obtained using other routines which have undergone exactly the same procedure.

Examples of the Battery method in various fields are described in Hull *et al.* (1972) with ordinary differential equations, Enright *et al.* (1975) with stiff ordinary differential equations, Casselleto, Pickett and Rice (1969) with automatic quadrature routines and Kahaner (1971) also with automatic quadrature routines.

The Battery method, however, rests on two basic assumptions

(*a*) that there is a 'best' routine for solving all problems of the, usually wide, class considered

(*b*) more importantly, that problems of the same difficulty but slightly different parameters will deliver similar results (timings, actual error, etc.).

These assumptions are not necessarily valid. For example, in the field of numerical quadrature, Lyness and Kaganove (1977) show that, of currently available automatic routines, those which do well with 'smooth' integrands are based on high order methods, while those which perform best on 'difficult' integrands are based on low order methods. Hence it would seem to be better to look for a routine which is the best for solving particular types of problem. The objection to the second assumption is best illustrated by an example.

Consider problem (1, 3). The kernel is a ridge whose height is governed by c and whose position and orientation is governed by p_2, p_3 and p_4. For a given value of c we would expect that the difficulty of the problem would be roughly the same whatever the values of p_2, p_3 and p_4. However if we take $p_1 = p_2 = p_4 = 0{\cdot}5$, $c = 0{\cdot}1$ and solve the problem with the NPL non-automatic routine, FRED2B, with $N = 15$ we obtain the following results:

$$p_3 = 0{\cdot}32 \text{ actual error } = 1{\cdot}405 \times 10^{-4}$$

$$p_3 = 0{\cdot}08 \text{ actual error } = 3{\cdot}976 \times 10^{-6}$$

a factor of about 35 difference in the error.

In fact, we can plot the actual error against the value of p_3 to yield the 'performance profile' of the method with this problem (Lyness and Kaganove, 1976). The approximate profile for the integral equation routine FRED2B (see Section 3) with the above problem and with problem (4.1) is given in **Fig. 1.**

Examples of performance profiles for automatic quadrature routines are given in Lyness and Kaganove (1977), where it it shown that the profiles may not even be continuous. The performance profile of different routines with the same function

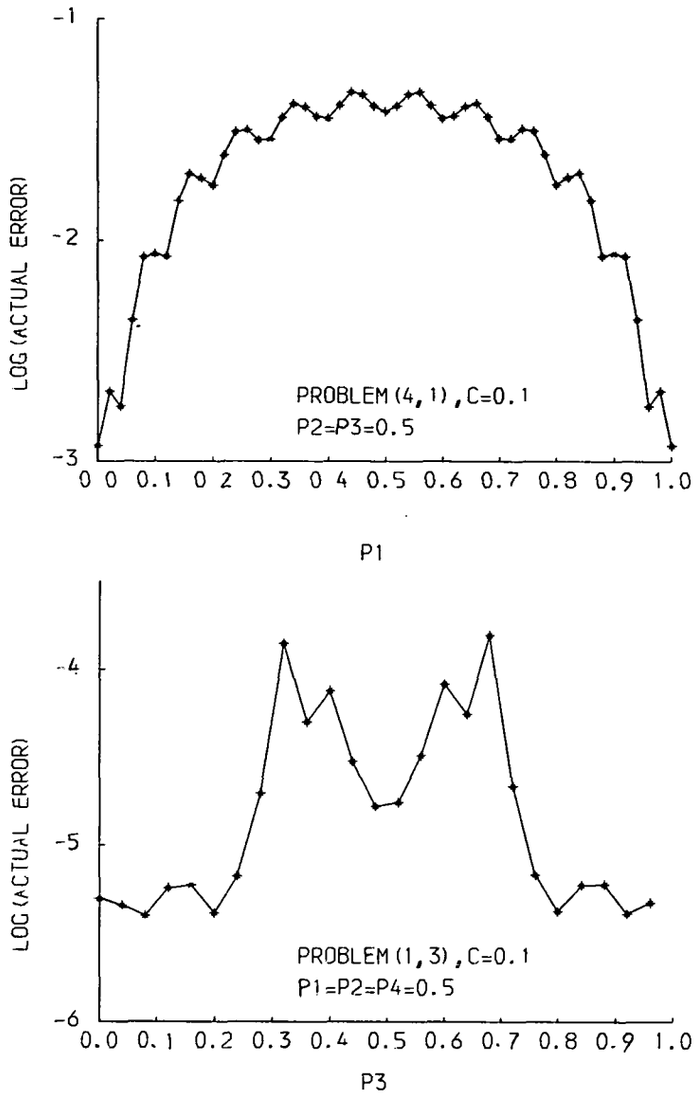


Fig. 1 The accuracy achieved by routine FRED2B on problem families (1,3) and (4,1) with $N = 15$ (logarithms to base 10)
Problem (1,3): $p1 = p2 = p4 = 0{\cdot}5$, $p3$ varied over 0,1
Problem (4,1): $p2 = p3 = 0{\cdot}5$, $p1$ varied over 0,1

are not the same and may cross each other at several places in the range under consideration. Hence with the Battery method one routine would appear better than another for certain choices of parameter but worse for others, giving misleading comparisons if only one value of parameter is used in a test.

Problems (1, 3) and (4, 1) each define a whole class of closely linked problems in which the difficulty (i.e. the ridge in (1, 3) and the spike in (4, 1)) can be moved about the range of interest by varying the values of p_1, p_2, p_3 and p_4. Such problems are called 'problem families'; the basic strategy advocated by Lyness and Kaganove, and adopted here, is to carry out statistical tests on single problem families in order to take account of rapid variations in performance profiles; and to seek a 'best' routine only for a single problem family, or for a set of related problem families.

3. Choice of problem families

The integral equation (1.1) is determined by the three functions $f(x)$, $g(x)$ and $K(x, y)$. Problems may be difficult to solve because of difficult behaviour of either the kernel, $K(x, y)$, or the driving term, $g(x)$. We construct below problem families with both types of difficulty. However, since we seek problems with known solutions, it is easier to treat $K(x, y)$ and $f(x)$ as the two independent functions when describing a given family. We choose then several forms of solution and of kernel. Each form contains either no apparent difficulty ('easy') or some feature likely to cause trouble to a solution method (spike, ridge, discontinuous first derivative). Each form also contains several parameters. Some of these affect the degree of difficulty in an obvious way: for example, the height-to-width ratio of a spike in the solution or the kernel; these parameters are allotted a fixed value within a problem family. Thus, a height to width ratio of 5:1 defines an 'easy' family while a ratio of 100:1 defines a different, and it appears to current routines, difficult, problem family.

Other parameters in the functional forms do *not* affect the apparent difficulty, but determine its position and orientation in the x-y plane (for the kernel) or its position on the x axis (for the solution or driving term). These parameters are varied by the test package, and statistical performance data collected, as described in Sections 4 and 5.

We consider only finite integration ranges in (1.1); the results quoted in Sections 4 and 5 all use the standard range

$$a = 0 \leqslant x, y \leqslant b = 1$$

Solution functions (parameter p_1)

1. $f(x) = p_1$ constant function—easy,
2. $f(x) = x - p_1$ linear function—easy,
3. $f(x) = \begin{cases} \dfrac{c(x-a)}{p_1 - a} & x < p_1 \\ \dfrac{c(b-x)}{b-p_1} & x \geqslant p_1 \end{cases}$ hat function of height c with peak at $x = p_1$
4. $f(x) = \dfrac{c}{(x-p_1)^2 + c^2}$ spike function of height $1/c$, half-width c, with peak at $x = p_1$

Kernel forms (parameters p_2, p_3, p_4)

1. $K(x, y) = (x - p_2)(y - p_3)$ easy
2. $K(x, y) = \begin{cases} (x-p_2)(1-(y-p_2)) & x \leqslant y \\ (y-p_2)(1-(x-p_2)) & x > y \end{cases}$ Green's function
3. $K(x, y) = \dfrac{c}{\{(1-p_4)(x-p_2) - p_4(y-p_3)\}^2 + c^2}$ Ridge of height $1c/$
4. $K(x, y) = \dfrac{c^2}{\{(x-p_2)^2 + c^2\}\{(y-p_3)^2 + c^2\}}$ Hyperbolic spiked kernel height $1/c^2$
5. $K(x, y) = \dfrac{c}{(1-p_4)(x-p_2)^2 + p_4(y-p_3)^2 + c^2}$ Parabolic spiked kernel height $1/c$.

Plots of the kernel functions 3, 4 and 5 are given in **Figs. 2-4.** For brevity we label problems by the integer pair (i, j), where i specifies the number of the solution form and j the number of the kernel form. For example, problem (1,3) is the problem with $f(x) = p_1$ and the ridge kernel. Obviously, with each problem we need to construct a driving term and to select a value of the 'difficulty parameter' c. The driving terms for problems currently under consideration (i.e. those for which results are given) are:

Problem (1,3)

$$g(x) = p_1 + \frac{p_1}{p_4}\left\{\arctan\left[\frac{(1-p_4)(x-p_2) - p_4(b-p_3)}{c}\right]\right.$$

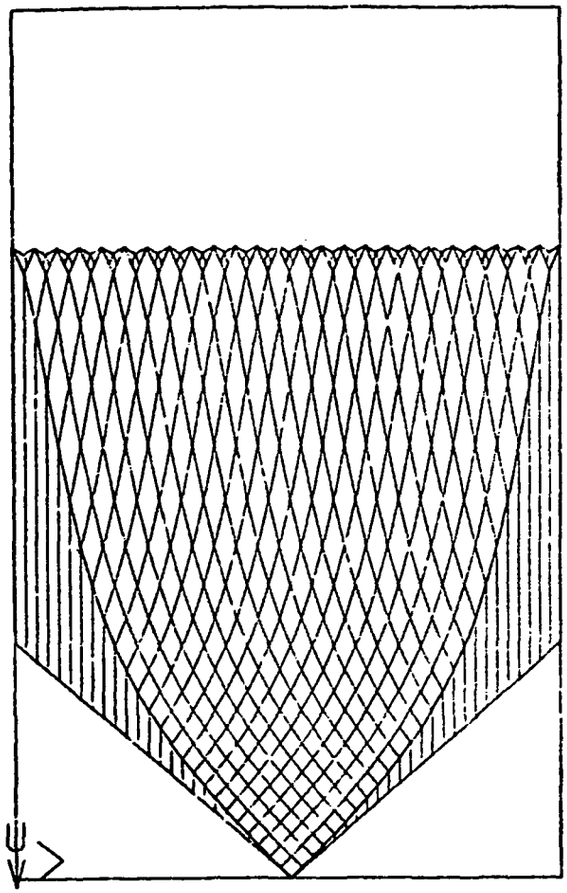


Fig. 2 Ridge kernel: $p2 = 0{\cdot}5, p3 = 0{\cdot}5, p4 = 0{\cdot}5$

Figs. 2-4 Three dimensional plots of the 'difficult kernel' functions

$$- \arctan\left[\frac{(1-p_4)(x-p_2) - p_4(a-p_3)}{c}\right]\Bigg\}$$

The value of c used is $c = 0{\cdot}1$.

Problem (1,4)

$$g(x) = p_1 - cp_1\left\{\arctan\left[\frac{(b-p_3)}{c}\right] - \arctan\left[\frac{a-p_3}{c}\right]\right\}\Big/ \{(x-p_2)^2 + c^2\}$$

$c = 0{\cdot}2$.

Problem (1,5)

$$g(x) = p_1 - \frac{p_1}{d}\frac{c}{\sqrt{p_4}}\left\{\arctan\left[\frac{b-p_3}{d}\right] - \arctan\left[\frac{a-p_3}{d}\right]\right\}$$

where $d = \sqrt{c^2 + (1-p_4)(x-p_2)^2}$

$c = 0{\cdot}1$.

Problem (4,1)

$$g(x) = \frac{c}{(x-p_1)^2 + c^2} - (x-p_2)\,c\left\{\tfrac{1}{2}\,ln\left[\frac{(b-p_1)^2 + c^2}{(a-p_1)^2 + c^2}\right] + \frac{(p_1-p_3)}{c}\left(\arctan\left[\frac{b-p_1}{c}\right] - \arctan\left[\frac{a-p_1}{c}\right]\right)\right\}$$

$c = 0{\cdot}1$.

4. Comparison of non-automatic routines

Non-automatic routines require the user to specify a parameter N: either the number of terms used (for an expansion method) or the number of mesh points (for a Nystrom or quadrature method). Given N, they return an N-point or N-term solution,

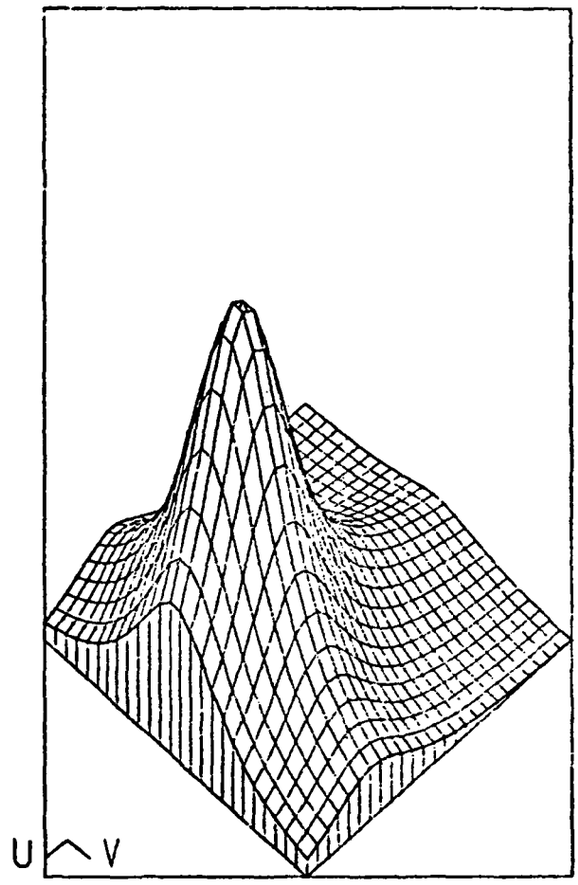


Fig. 3 Hyperbolic spiked kernel: $p2 = 0{\cdot}75, p3 = 0{\cdot}5, p4 = 0{\cdot}2$

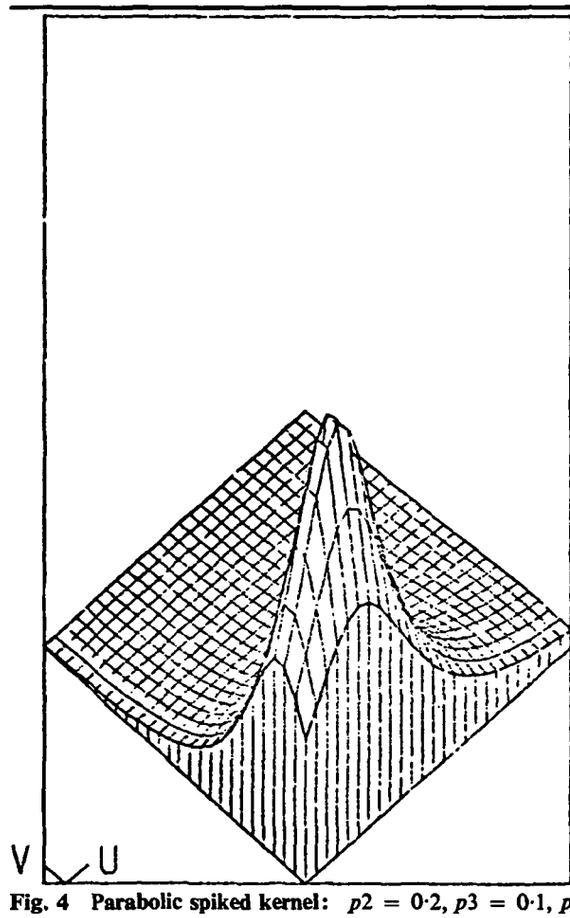


Fig. 4 Parabolic spiked kernel: $p2 = 0{\cdot}2, p3 = 0{\cdot}1, p4 = 0{\cdot}75$

and possibly also an error estimate. To the user the main characteristics of interest in a routine are the time taken and the accuracy achieved as a function of N.

To test the efficiency of a non-automatic routine with a particular problem family we carry out the following procedure with a number of different values of N.

(*a*) run the routine with many different values of problem parameters $\{p_i, i = 1, \ldots, 4\}$

(*b*) after each individual run note the actual error and mill-time taken

(*c*) finally, average out the actual error and mill-time taken. These average values are then plotted against N.

Note that this test procedure gives no credit for extra information (such as an error estimate) returned by the routine.

The values of the problem parameters used are allowed to vary uniformly between the limits $[a, b]$ (see equation (1.1)), and are chosen randomly at each run, the runs being continued until the statistical averages have settled down. This method of varying the problem space involved appears to work well in practice; even with up to four parameters per family relatively few runs are required to obtain reproducible comparisons. The results presented here are based on 100 runs per family for each routine. We note that the time taken for fixed N is independent of the problem parameters for most routines, while differences in achieved accuracy of 20% say, are unlikely to be significant (or even noticed) in practice, so that high statistical accuracy is not required.

A test package to implement this procedure, and to collect and analyse the statistics, has been written in ALGOL 68. Routines to be tested have been translated into ALGOL 68 first if necessary. This is unlikely to affect the accuracy of a routine significantly; for a discussion of the (small) effect on timings, see Section 5.

Results

So far we have tested the following non-automatic routines.

1. *FFTNA—Delves and Abd-Elal* (1977)

A Galerkin procedure using Fast Fourier Transform techniques to evaluate the integrals and an iterative technique to solve the resulting equations, and producing an expansion in Chebyshev polynomials and an error estimate. For an N-term expansion the time taken is $0(N^2 \log N)$ and in practice the routine spends a large proportion of its time in an FFT module; the module used in these tests was the NAG routine C06AAB.

2. *FRED2B—Miller and Symm* (1975)

Available in the NAG ALGOL 60 and FORTRAN routines as routine D05ABA/F.

This routine from NPL (in ALGOL 60) produces a Chebyshev expansion using the method of El-Gendi (1969); that is, a Nystrom procedure with Clenshaw-Curtis quadrature followed by conversion of the approximate solution to Chebyshev series form. This routine spends most of its time in the direct solution of an $N \times N$ set of equations using NAG routine F04AJA. We have converted the routine to ALGOL 68 before testing, using the equivalent NAG ALGOL 68 routine F04AJB.

3. *D05ACB—NAG ALGOL 68 library*

This uses a Nystrom method based on the NAG ALGOL 68 Gauss quadrature package; the quadrature rule used is specified by the user. For test purposes we have called on Gauss-Legendre quadrature. This routine also spends most of its time in NAG routine F04AJB.

4. *FE2SR—K. S. Thomas* (1976)

This uses a Nystrom method with Simpson's rule, and produces also an error estimate and a corrected solution. Again this routine spends most of its time in the NAG routine F04AJB which is also called in producing the correction and the error estimate.

To be fair to the routines it is important to use the same criterion in evaluating the error. We use the following:

$$\text{Error} = \frac{1}{m}\left\{\sum_{i=1}^{m} |f_t(x_i) - f_N(x_i)|^2\right\}^{\frac{1}{2}}$$

where $f_t(x_i)$ is the true, and $f_N(x_i)$ the approximate, solution at x_i; and x_i, $i = 1(1)m$ are equally spaced points over the range $[a, b]$ (see equation (1.1)). In FFTNA and FRED2B the solution is given as a set of coefficients of a Chebyshev series;

$$f_N(x_i) = \sum_{j=1}^{N}{}' a_j T_j(x_i) \quad i = 1(1)m$$

however, for D05ACB and FE2SR the solution is given at the quadrature points and we use the 'natural' interpolation formula

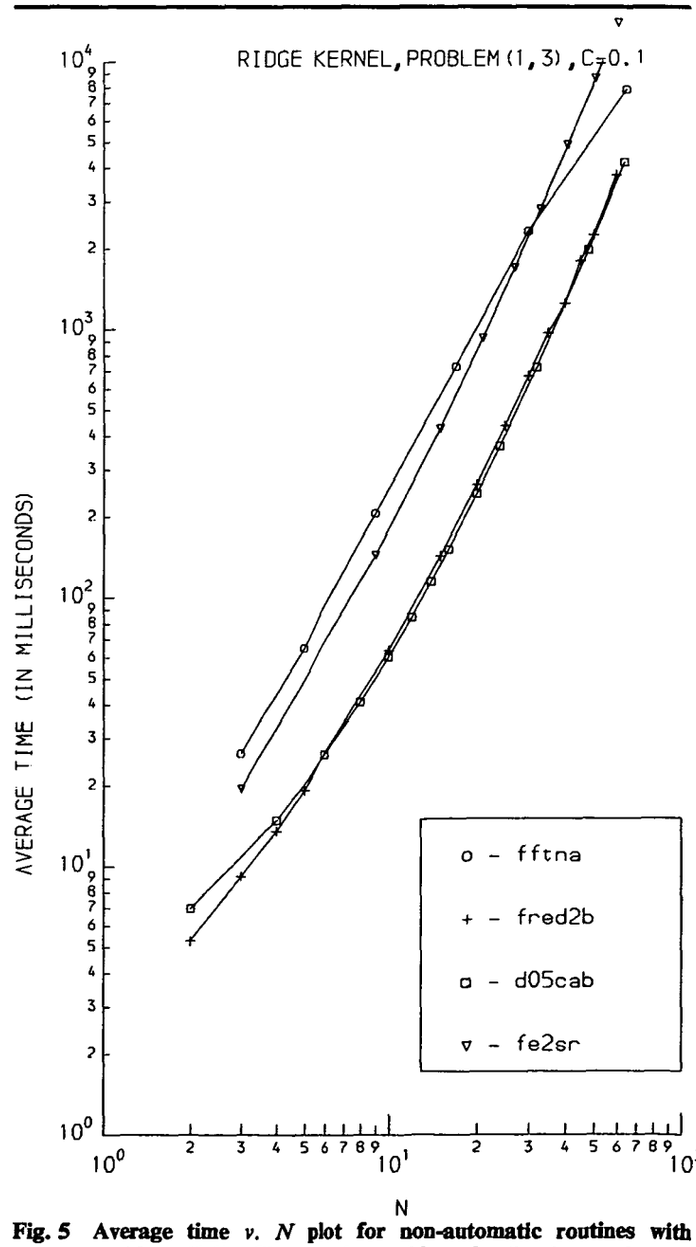


Fig. 5 Average time *v.* N plot for non-automatic routines with problem (1,3). Changes in problem family do not have a significant effect on the plots of time *v.* N.

$$f_N(x_i) = g(x_i) + \sum_{j=1}^{N} K(x_i, \bar{x}_j)\bar{f}_N(x_j)\bar{w}_j \quad i = 1(1)m$$

where $\bar{x}_j$ are the quadrature points, $\bar{w}_j$ the quadrature weights and $\bar{f}_N(x_j)$ the solutions delivered by the routine.

Discussion of results

Fig. 5 gives a plot of mill time (in 1906S milliseconds) against N for the routines tested. The times taken are essentially independent of family, except for routine FFTNA which uses an iterative solution of the equations. **Figs. 6-9** show the achieved average accuracy against N, on a logarithmic scale, for the problem families tested to date.

Timings

The results for routine FFTNA are consistent with the theoretical estimate $T = 0(N^2 1n\ N)$. The routine is initially slower than other routines tested but there is a crossover point at about $N = 27$ where FE2SR becomes slower and by extrapolation, the curves of both FRED2B and DO5ACB cross the FFTNA curve at about $N = 80$. In practice the routine spends a large proportion of its time in the FFT module, and hence using a faster FFT module would have a marked effect on speeding up this routine.

DO5ACB, FRED2B and FE2SR are all based on the Nystrom method, but use different quadrature rules. As N increases the time taken in solving the Nystrom equations becomes increasingly dominant, so, since we use the same routine to solve the equations in each case (FO4AJB), we would expect little difference in the timings. In fact the curves for DO5ACB and FRED2B are almost coincident, and the curve for FE2SR is roughly parallel to them, FE2SR being slower because the error estimate and correction generated by FE2SR each require the solution of a set of N equations. We can also see from the results that as the solution of the equations begins to dominate the gradient of each curve is approaching 3 (i.e. $0(N^3)$ for Gauss elimination).

Accuracy

Routines DO5ACB and FRED2B, based respectively on Gauss-Legendre and Gauss-Chebyshev quadratures, have similar error curves for each of the problems containing difficult kernels (i.e. problems (1,3); (1,4); (1,5)) whilst, of those routines tested, FE2SR is the least successful and even the correction mechanism (denoted FE2SR(c) in the graphs) fails to make it competitive here. For these problem families FFTNA compares favourably with DO5ACB and FRED2B only while N is less than about 25. For larger values of N, its performance appears to be affected by premature termination of the iterative solution of the equations. The termination criterion

Figs. 6-9 Plots of average error, on a logarithmic axis, *v.* N for the non-automatic routines

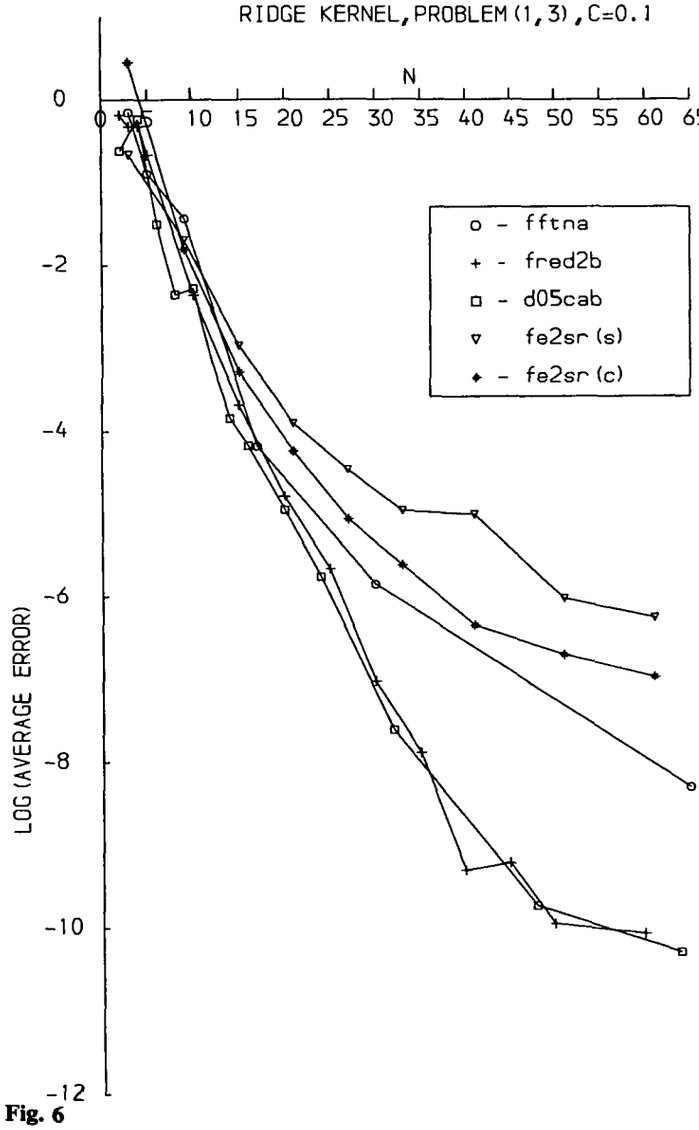


Fig. 6

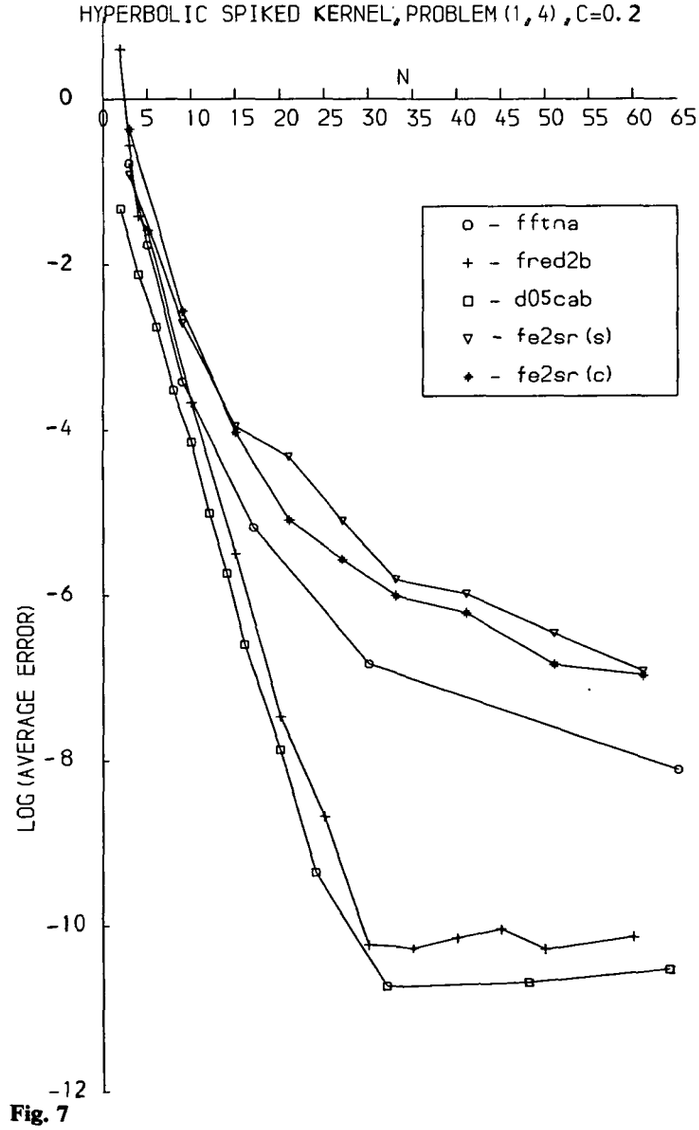


Fig. 7

used by FFTNA is based on an *a priori* error estimate derived from inspection of the Galerkin matrix and righthand side vectors. The problem families considered here all have 'difficult' kernels and 'easy' solutions, and hence show considerable cancellation effects between the kernel and driving terms which are not detected by this *a priori* estimate.

For problem (4,1), which contains a spiked solution, an entirely different situation exists. Again DO5ACB does well, and in fact, is considerably better than any other routine tested so far. Routines FRED2B and FFTNA perform equally well and surprisingly badly, in this instance. The behaviour of FE2SR is similar to that already described although the effect of the correction is less marked. On the basis of these results we would declare DO5ACB the overall winner for the classes of problems considered.

5. Comparing automatic routines

An automatic routine is one which, given an input tolerance ε_{inp}, attempts to adjust the discretisation order N until some measure of the error is less than ε_{inp}; it then returns f_N, and possibly other information such as its own estimate of the achieved accuracy, and whether the result appears reliable. Again we give no credit for this extra information (or penalty for its absence) but analyse only the way in which the routine satisfies its basic task. The user has two main questions to ask of such a routine:

1. Speed: For a given accuracy, how fast is it?
2. Reliability: How often will it achieve the requested input accuracy?

It is traditional to measure speed in terms of the number of function evaluations used; here, the number of times the kernel $K(x, y)$ is evaluated provides a measure. However, unlike the situation in numerical quadrature, optimisation and (usually) ordinary differential equations, 'red tape' operations can completely dominate the time taken; on the examples run, routines typically spend less than 10% of their time in kernel evaluations, given the rather simple kernels used here. It would take a complicated kernel to change this situation significantly, and we therefore quote also the mill-time taken, in milliseconds, on an ICL 1906S computer.

Requirements 1 and 2 are interdependent: a routine writer can trade speed for reliability at the stroke of a keypunch (by setting $\varepsilon_{test} := 0{\cdot}1 \times \varepsilon_{inp}$, for example). The Lyness and Kaganove (1977) analysis for automatic quadrature routines, which we follow here, attempts to disentangle the intrinsic speed of a routine from its intrinsic reliability by factoring out the effect of such possible internal scalings; and they achieve this in the following manner.

Suppose we wish to evaluate a routine with a particular

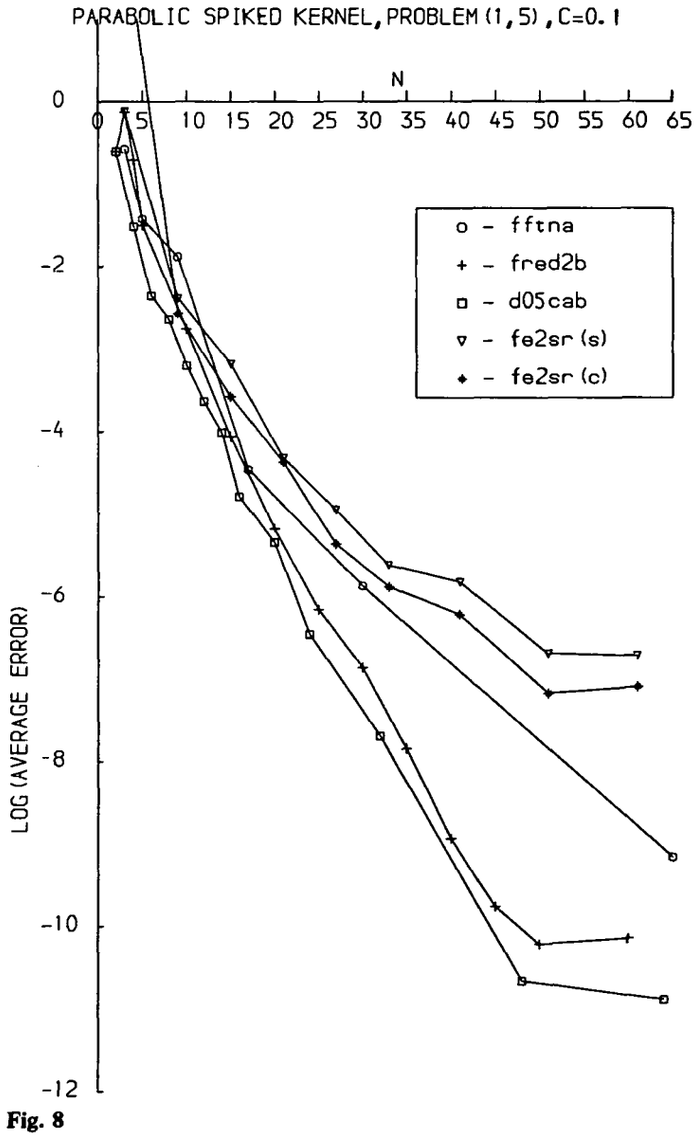


Fig. 8

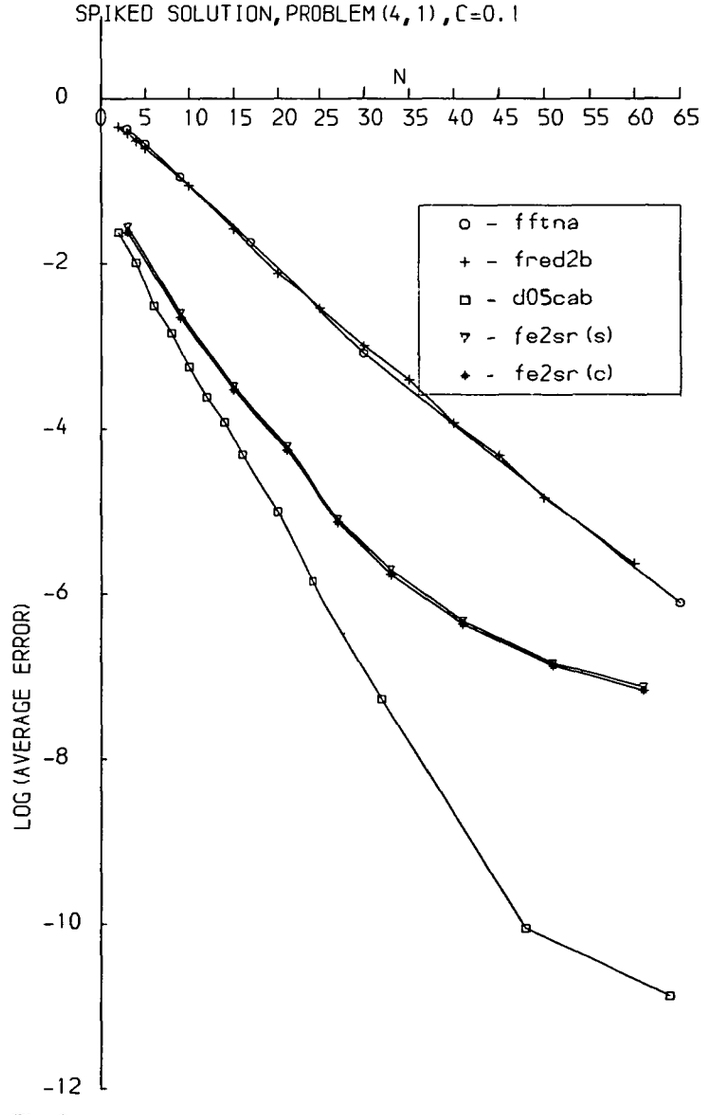
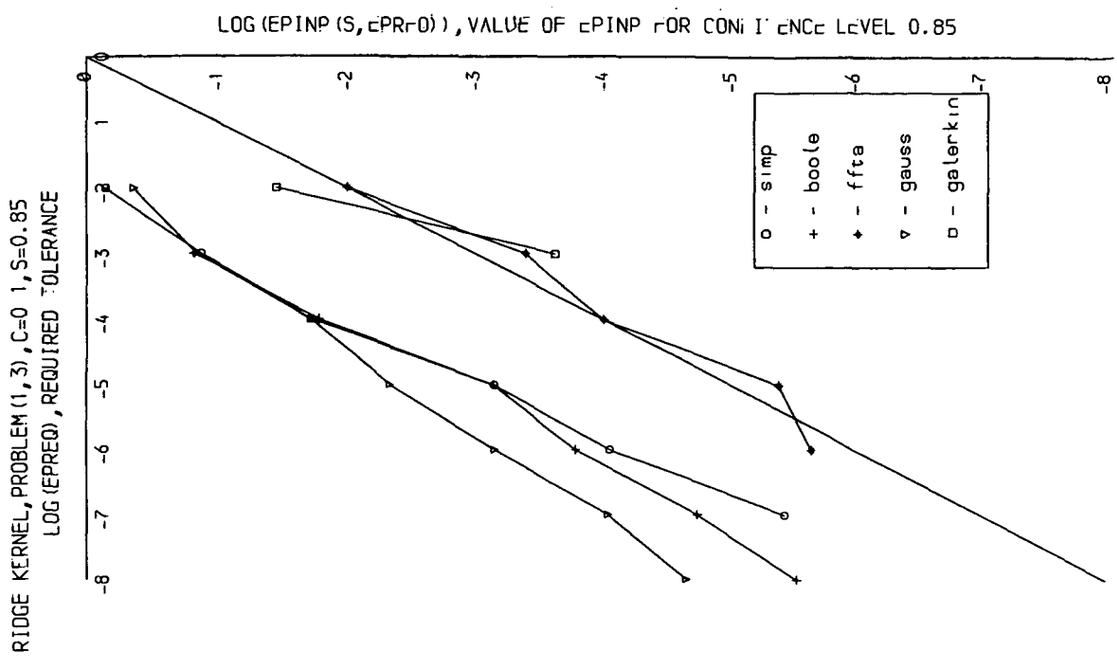


Fig. 9



Figs. 10-12 Plots of automatic results using problem (1,3), $c = 0{\cdot}1$, $s = 0{\cdot}85$

Fig. 10 Plot of log(Epinp (s,epreq)) *v.* log(epreq)

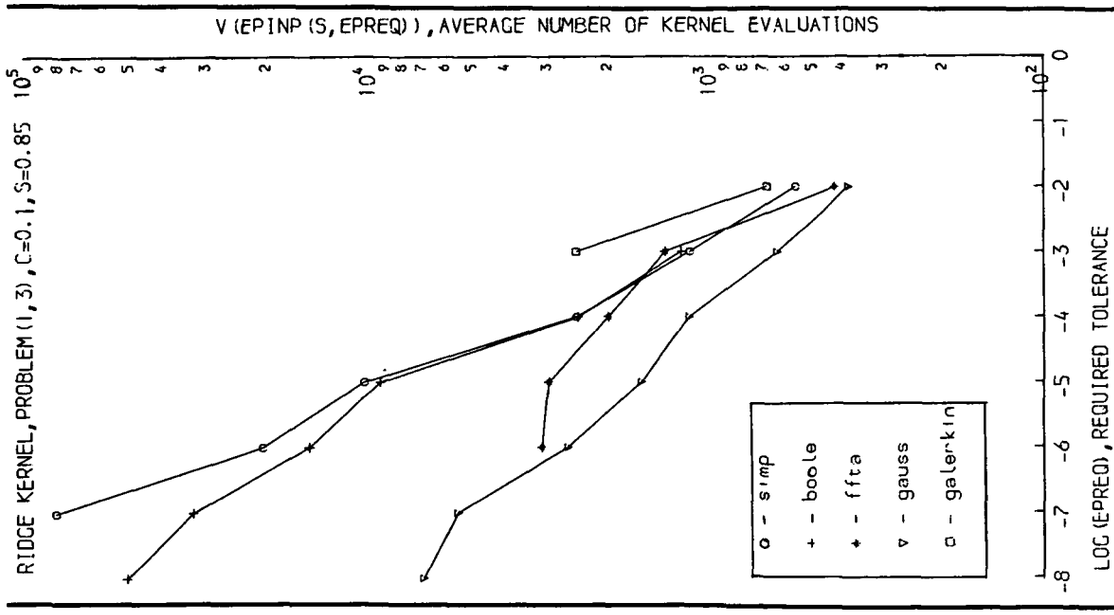


Fig. 11 Plot of V(Epinp (s,epreq)) *v.* log(epreq)

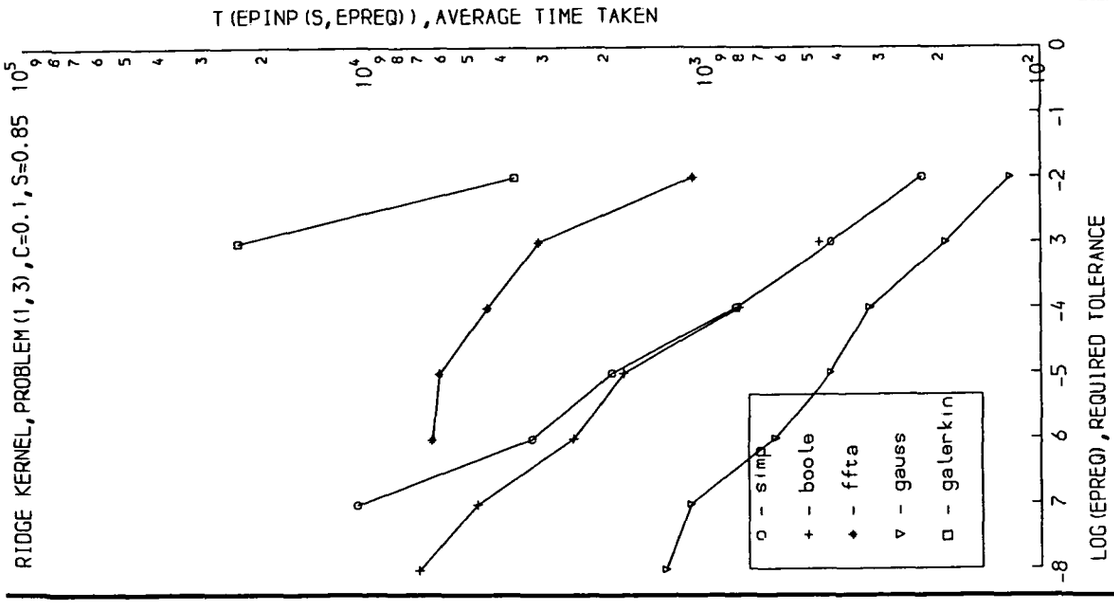


Fig. 12 Plot of t(Epinp (s,epreq)) *v.* log(epreq)

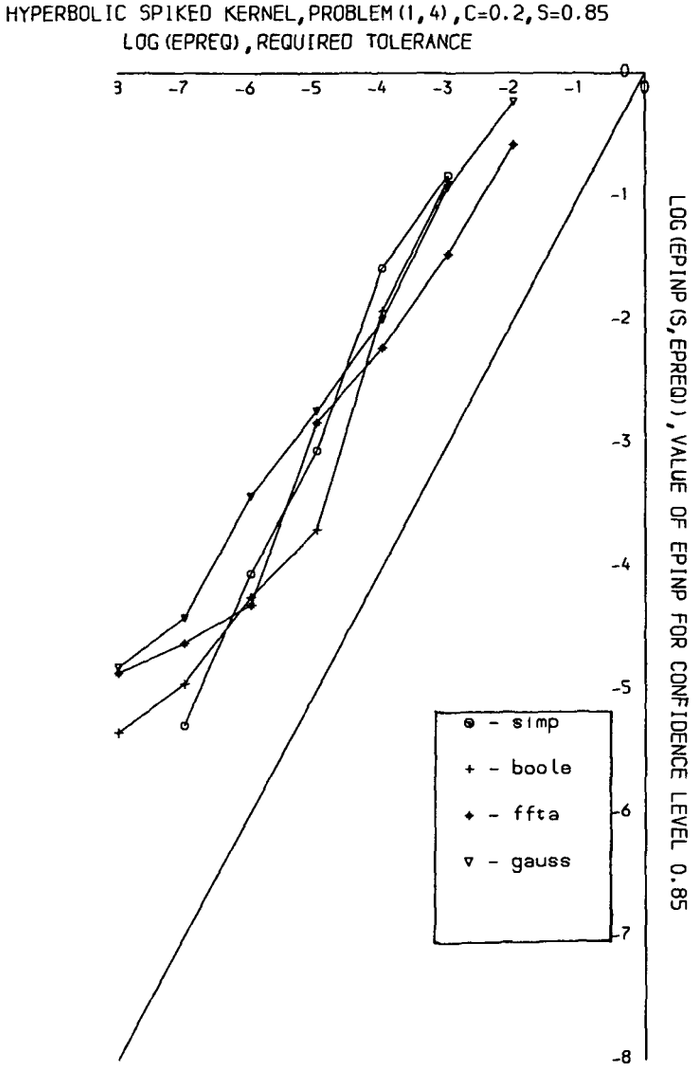


Fig. 13

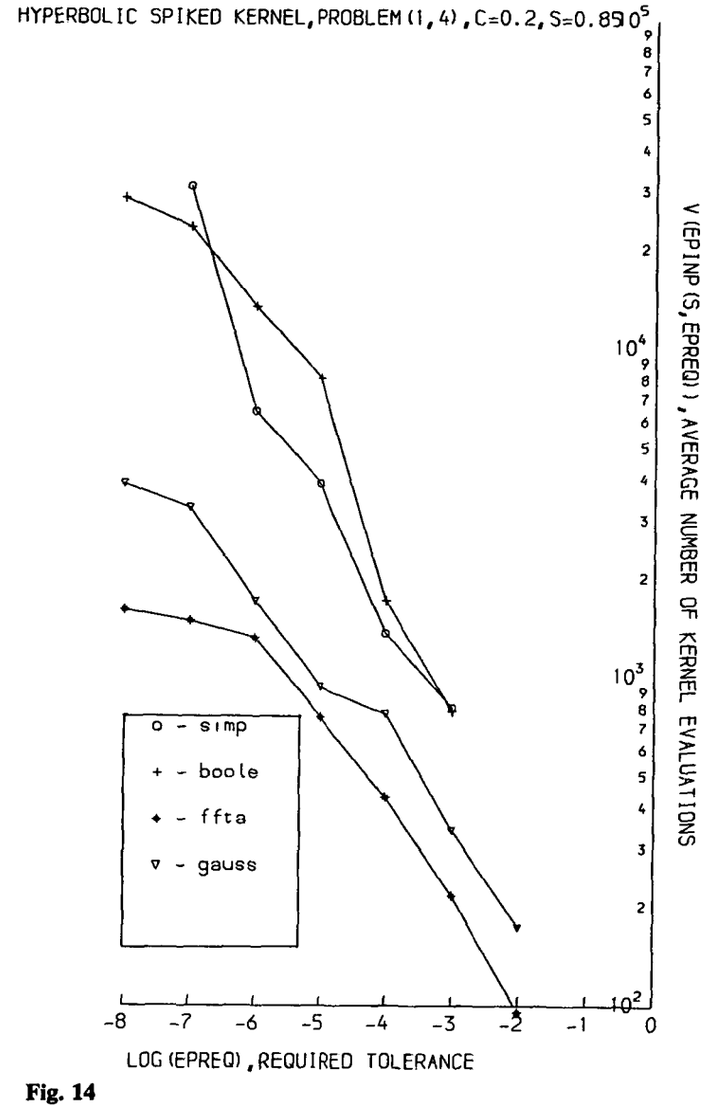


Fig. 14

Figs. 13-15 As Figs. 10-12 using problem (1,4), $c = 0{\cdot}2$, $s = 0{\cdot}85$

problem family; then for each value of input tolerance (ε_{inp}) we run the routine a number of times with random values of problem parameters. At the end of each run we note the following:

ε_{act} (ε_{inp}, $\mathbf{p}_i$)—the actual error. This is related to the input tolerance ε_{inp} and the values of the problem parameters $\mathbf{p} = (p_1, p_2, p_3, p_4)$

$v(\varepsilon_{inp}, \mathbf{p}_i)$— the number of kernel evaluations. Again this is related to ε_{inp} and $\mathbf{p}$

$t(\varepsilon_{inp}, \mathbf{p}_i)$— the mill-time taken

where i is the number of the run. This information is used to construct the following statistics:

$$v(\varepsilon_{inp}) = \frac{1}{M}\sum_{i=1}^{M} v(\varepsilon_{inp}, \mathbf{p}_i)$$

$$t(\varepsilon_{inp}) = \frac{1}{M}\sum_{i=1}^{M} t(\varepsilon_{inp}, \mathbf{p}_i)$$

$$\phi(x; \varepsilon_{inp}) = \frac{1}{M}\{\text{number of values of } i \text{ for which } |\varepsilon_{act}(\varepsilon_{inp}, \mathbf{p}_i)| \leqslant x\}$$

Here M is the total number of runs for each value of ε_{inp}; $\phi(x; \varepsilon_{inp})$ is a statistical distribution function and gives the proportion of the time an actual error x is achieved for a given input tolerance.

We now put ourselves mentally in the place of a knowledgeable (or cynical) user who will reason as follows:

1. Let ε_{req} be the accuracy I require. No automatic routine can achieve the accuracy I want every time. I will nominate a percentage, s, and ask that it achieve the desired accuracy exactly s % of the time. To achieve more will slow the routine down, to achieve less will be to fail.
2. I will make sure the routine does this by taking as *input* tolerance the value $E_{inp}(s, \varepsilon_{req})$, which is the value of ε_{inp} for which the tolerance ε_{req} is achieved exactly s percent of the time. We find $E_{inp}(s, \varepsilon_{req})$ as the largest value of ε_{inp} for which $s = \phi(\varepsilon_{req}; \varepsilon_{inp})$.

Plots of $v(E_{inp}(s, \varepsilon_{req}))$ and $t(E_{inp}(s, \varepsilon_{req}))$ against ε_{req} then indicate the intrinsic efficiency of the routine, while a plot of $E_{inp}(s, \varepsilon_{req})$ against ε_{req} indicates the quality of the error control in the routine. The plots depend upon the percentage reliability s chosen, but in practice the relative orderings of different routines are not sensitive to s (if they were, it would be difficult to interpret the results) and here we take the standard value $s = 0{\cdot}85$ (i.e. 85%).

Results

We give results for the following routines

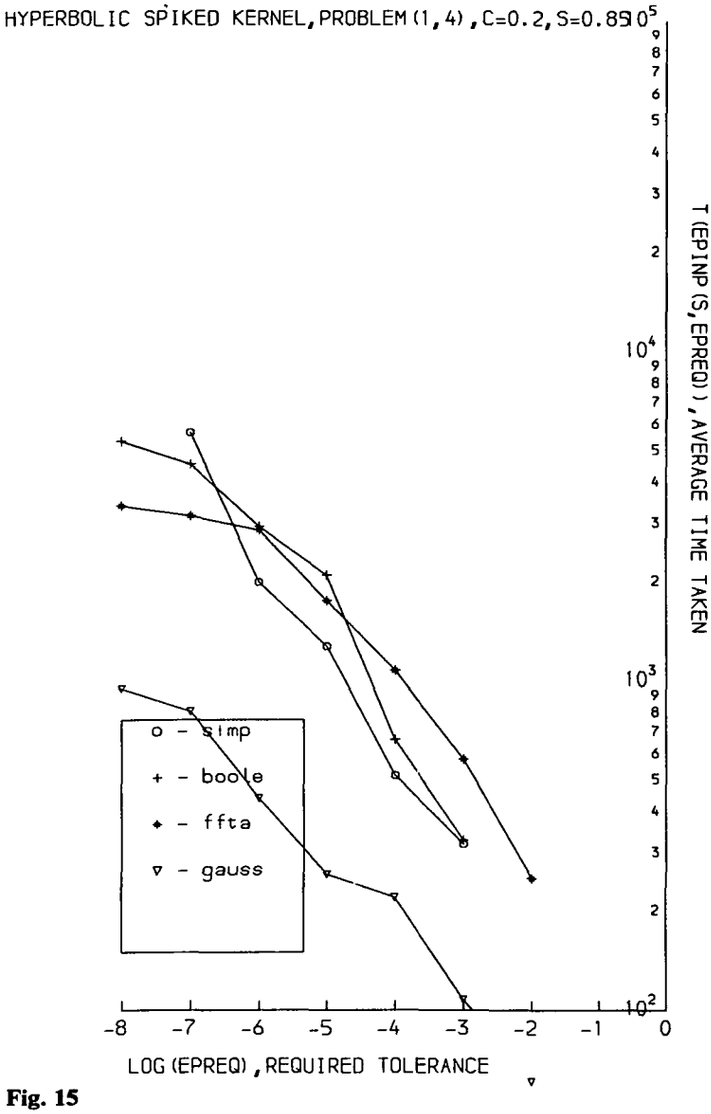


Fig. 15

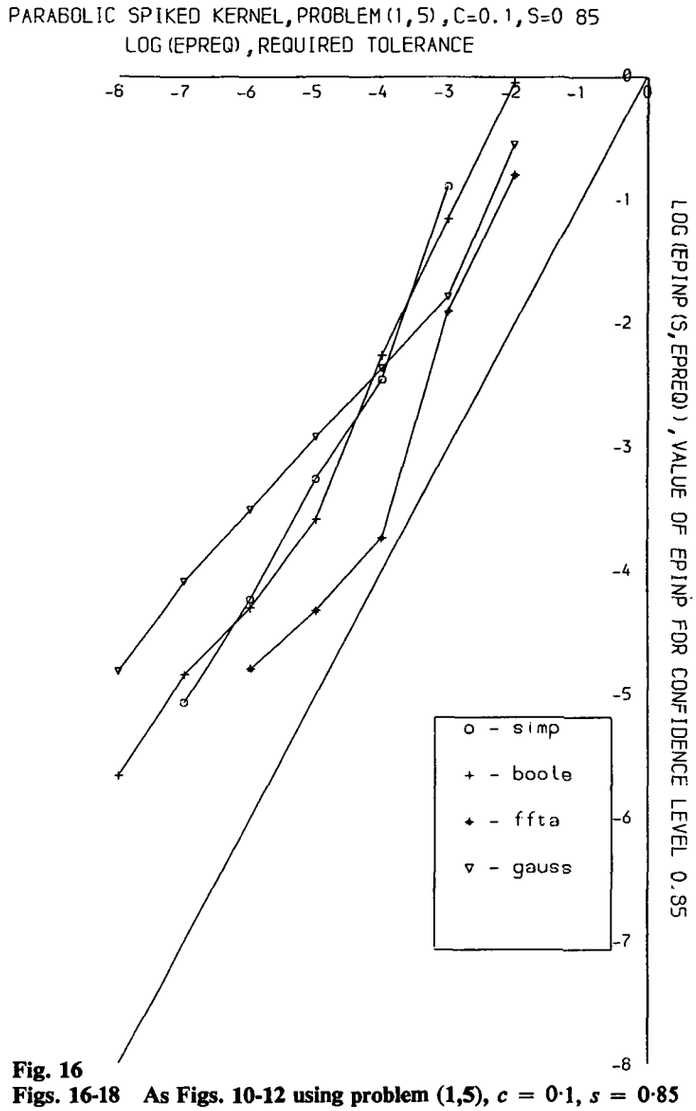


Fig. 16

Figs. 16-18 As Figs. 10-12 using problem (1,5), $c = 0{\cdot}1$, $s = 0{\cdot}85$

1. *FFTA—Delves* (unpublished)
A simple extension of the non-automatic routine FFTNA considered previously. A value of the error is estimated for a starting value of N and if this is not sufficiently small then the value of N is stepped up and the procedure is repeated.

2. *SIMP—K. E. Atkinson* (1976)
An ALGOL 68 translation of the routine IESIMP based on the Nystrom method using Simpson's rule and interative solution of the Nystrom equations.

3. *BOOLE—K. E. Atkinson* (1976)
As SIMP with higher order quadrature rule.

4. *Galerkin*—a version of FFTA based on the standard Galerkin rather than Fast Galerkin (1977) technique.

5. *Gauss*—a routine also by K. E. Atkinson similar in approach to SIMP and BOOLE, but using a sequence of Gauss-Legendre rules to provide basic approximations.
Our initial intention was to convert Gauss also to ALGOL 68. However, it makes heavy use of COMMON and EQUIVALENCE, and the FORTRAN facility to break up workspace declared to be of one mode and to use it as another mode. It proved easier to rewrite the test package in FORTRAN, and to run the FORTRAN Gauss routine. To estimate the relative speeds of the ALGOL 68 and FORTRAN compilers, we have run routine SIMP in both FORTRAN and ALGOL 68 versions. The timings obtained are shown in **Table 1**; they differ by an order of under 10%, which is not significant in the current context.

Performance results for the above routines, with problem families (1,3), (1,4), (1,5) are shown in **Figs. 10-21.**

Discussion of these results

1. *Problems with smooth solutions*: (1,3), (1,4), (1,5)
Several features of these results stand out clearly

1. The Fast Galerkin procedure used in FFTA is much more economic in practice than the standard Galerkin procedure; indeed, we discontinued tests with GALERKIN because of the costs involved.
2. Routines based on high order rules: FFTA, GALERKIN, GAUSS, are much more economical of kernel evaluation for these families than those based on low order rules (SIMP, BOOLE).
3. Nonetheless, the low overheads of SIMP and BOOLE make their run times competitive with FFTA (but not with GAUSS) for accuracies up to 10^{-6}.
4. Routine GAUSS emerges as a clear overall winner for these families of problems. For some reason not clear to us, BOOLE fails to improve on SIMP.

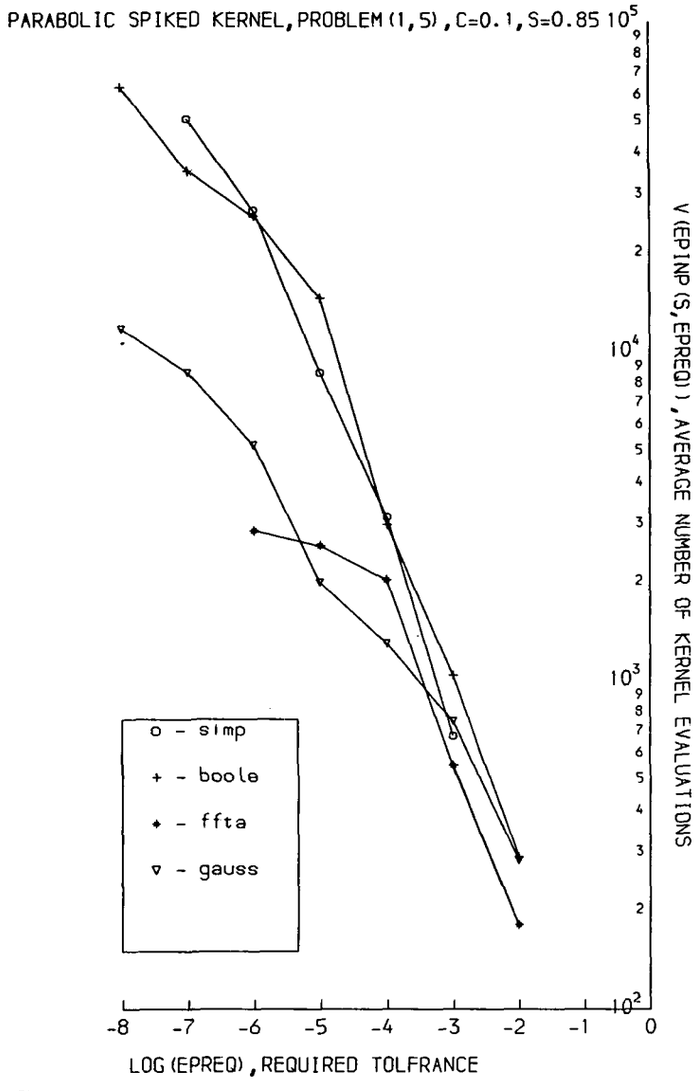


Fig. 17

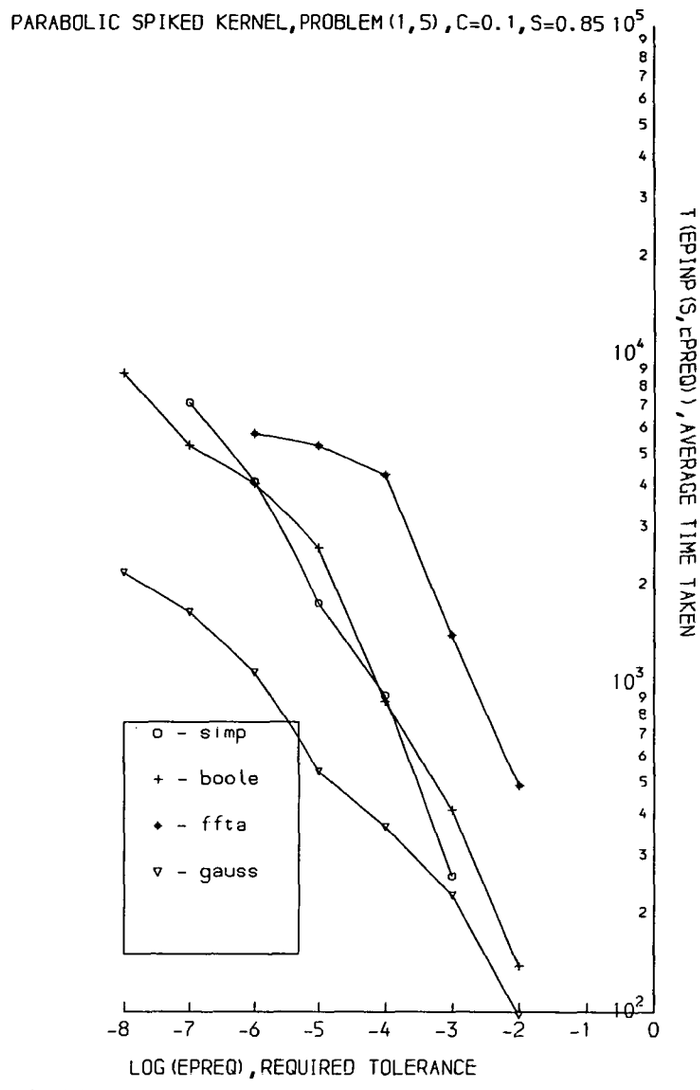


Fig. 18

2. *Problems with 'difficult' solutions*: (4,1)
Only results for BOOLE and SIMP are given here. Routine FFTA returned results which it predicted correctly were of low accuracy while routine GAUSS failed to compute any solution at all. Routine SIMP is therefore declared a winner in this class.

Finally we note that, again following Lyness and Kaganove, we find it hard to say much about the error control figures.

Acknowledgements
We are grateful to Kendall Atkinson for discussions of his routines, and to the Science Research Council for the award of a Research Studentship during the tenure of which this work was carried out.

Table 1 Values of $\log_{10}t(Epinp(s, epreq))$, the average time taken to achieve the requested accuracy EPREQ in exactly 100s% of the runs made, using the ALGOL68 and FORTRAN versions of the routine SIMP with problem (1,3), $c = 0{\cdot}1$, $s = 0{\cdot}85$.

log(EPREQ)	$\log_{10}t$ (ALGOL68)	$\log_{10}t$ (FORTRAN)	t(ALGOL68)/ t(FORTRAN)
0	2·270	2·161	1·285
−1	2·666	2·601	1·161
−2	2·986	2·891	1·245
−3	3·235	3·212	1·054
−4	3·497	3·458	1·094
−5	3·801	3·836	0·923

References

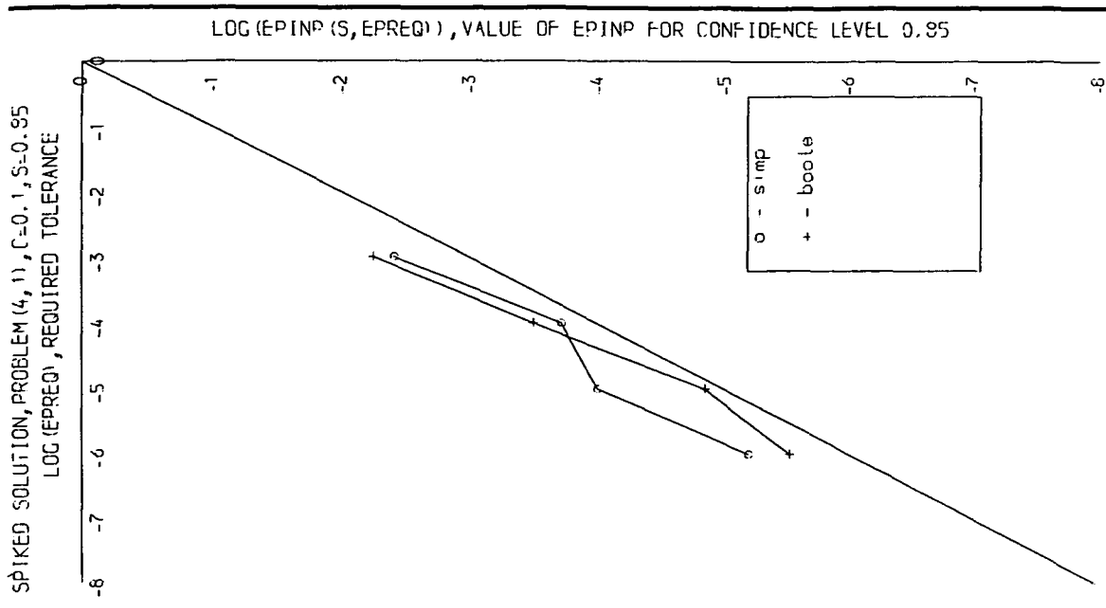
ATKINSON, K. E. (1976). An automatic program for linear Fredholm integral equations of the second kind, *Trans. on Math. Software*, Vol. 2, pp. 154-171.

CASSELLETO, J., PICKETT, M. and RICE, J. R. (1969). A comparison of some Numerical Integration Programs, *SIGNUM*, Vol. 4, pp. 30-40.

DELVES, L. M. and ABD-ELAL, L. F. (1977). The Fast Galerkin algorithm for the solution of linear Fredholm equations, Algorithm 97, *The Computer Journal*, Vol. 20, pp. 374-376.

EL-GENDI, S. E. (1969). Chebyshev solution of differential, integral and integro-differential equations, *The Computer Journal*, Vol. 12, pp. 282-287.

ENRIGHT, W. H., HULL, T. E. and LINDBERG, B. (1975). Comparing numerical methods for stiff systems of ODE's, *BIT*, Vol. 15, pp. 10-48.



Figs. 19-21 As Figs. 10-12 using problem (4,1), $c = 0{\cdot}1$, $s = 0{\cdot}85$

Fig. 19

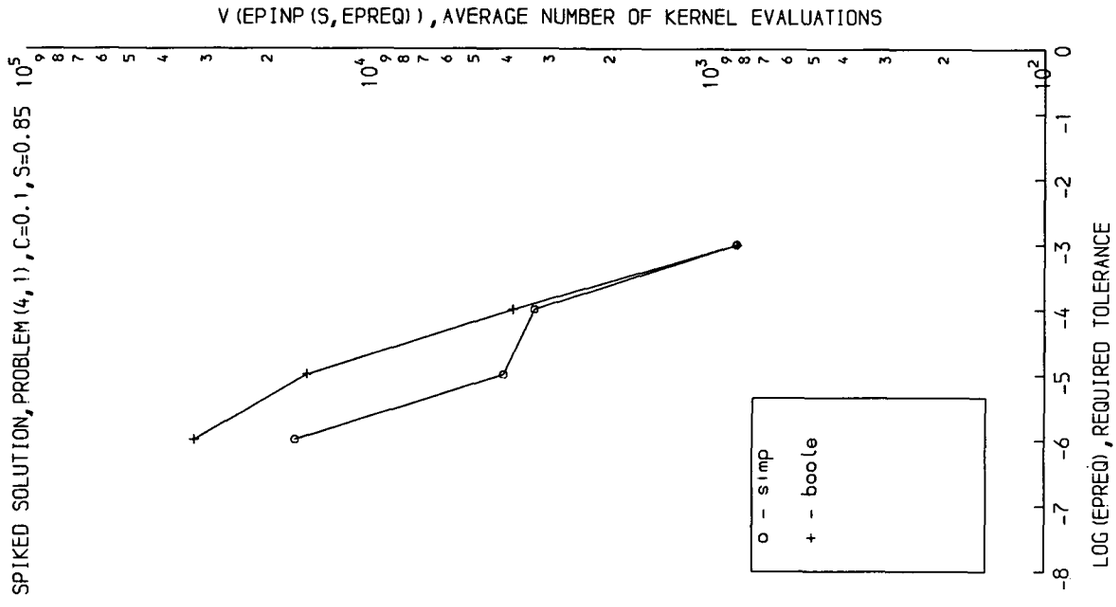


Fig. 20

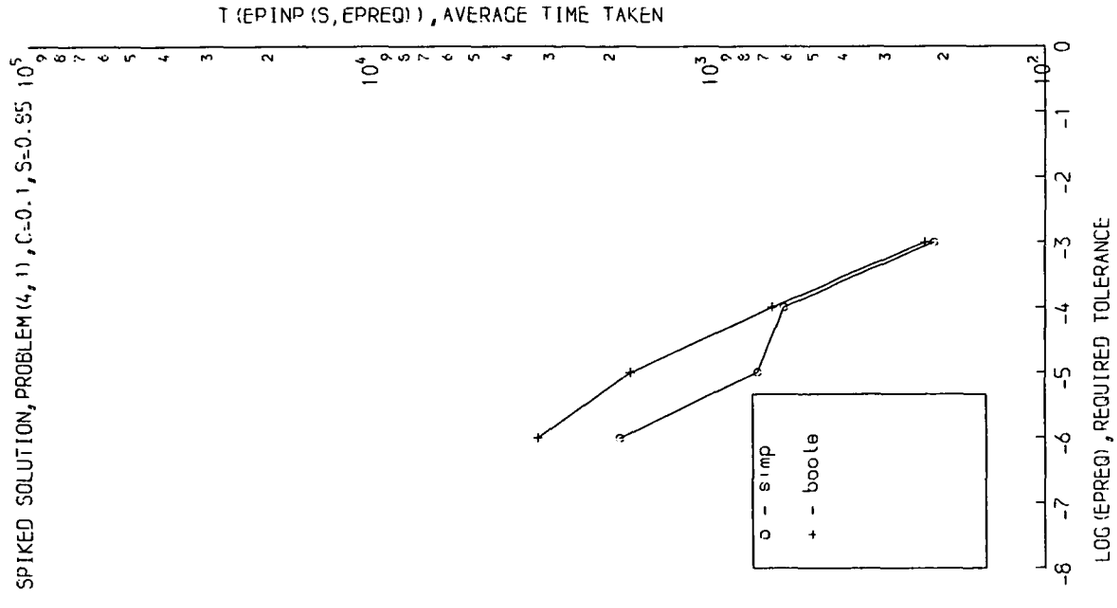


Fig. 21

HULL, T. E., ENRIGHT, W. H., FELLON, B. M. and SEDGWICK, A. E. (1972). Comparing numerical methods for ordinary differential equations, *S.I.A.M. J. Numer. Anal.*, Vol. 9, pp. 603-637.
KAHANER, D. K. (1971). Comparison of Numerical Quadrature Formulas in *Mathematical Software*, ed. J. R. Rice, Academic Press, N.Y., pp. 229-259.
LYNESS, J. N. and KAGANOVE, J. J. (1976). Comments on the nature of automatic quadrature routines, *ACM Trans. Math. Soft*, Vol. 2, pp. 65-81.
LYNESS, J. N. and KAGANOVE, J. J. (1977). A technique for comparing quadrature routines, *The Computer Journal*, Vol. 20, pp. 170-177.
MILLER, G. F. and SYMM, G. T. (1975). Procedure FRED2B, NPL routine, ref. no. DS/06/1/Algol 60/7/75.
THOMAS, K. S. (1976). On the approximate solution of operator equations, Part II, Preprint, University of Oxford.

Book reviews

Information Management Systems/Virtual Storage, by Myles E. Walsh, 1980; 308 pages. (*Prentice-Hall*, £11·00)

How to put a record in a file quickly, allow for a possible future change in its size and make it appropriately cross referenced to permit rapid retrieval, is the problem. This book deals with the specific solution for batch and real time using IMS/VS (which is the most comprehensive of Data Base Management System packages). More restrictively this book is for DP managers of departments with IBM mainframes who need to bridge the gap between the glowing descriptions of the salesmen's glossy flysheets and the volumes of technical manuals. Within this brief the author does very well. The book is readable without being juvenile or condescending, basic terms are defined and there is a good index.

The first chapters build up a clear idea, with the help of analogues, of a data base and its management. Mr Walsh then proceeds to deal with the technical aspects of the system, its batch and telecommunications features, its utilities and other features. A chapter deals with actual managerial problems, like staffing and educating. Another chapter intercompares major DBMS packages: The author's honesty is to be commended when he says some think IMS/VS is the cat's whiskers (Cadillac) and others believe it to be a lemon (Edsel). Likewise he admits IBM was a world follower in so-called Virtual Storage.

The book is understandably peppered with IBM buzz words but unfortunately there is no glossary of mnemonics. IMS/VS may be ideal for many DPDs (Data Processing Departments) but is nvg (not very good) for smaller mainframes DBMSs (Data Base Management Systems) or distributed micro network TP (Teleprocessing) systems, which are now part of the EDP/MIS (Electronic Data Processing/Management Information System) landscape. However having said that the book is well worth selectively reading in order to understand the architecture and concepts of one manufacturer's answer to IMS, especially as other solutions cannot be radically different.

I. R. WILLIAMS (London)

Learning to Program in Structured Cobol, by Edward Yourdon, 1979; 465 pages. (*Prentice-Hall*, £10·35)

Structured Analysis and System Specification, by Tom De Marco, 1979; 348 pages. (*Prentice-Hall*, £16·25)

A fashionable algorithm for the construction of book titles is: (1) devise a title which describes the content in a form suitable for the market; (2) select a noun from the title; (3) insert the word 'structured' before the selected noun. That this algorithm fails on occasion is illustrated by the first of the books reviewed here: it is really about 'Learning to write well-structured programs in ANSI 1974 Cobol'. There are two parts: 'Part 1 [co-authors Gane, Sarson] can be used as a stand-alone introduction to structured programming or it can be used in conjunction with the more advanced concepts and features presented in Part 2 [co-author Lister]'. These 'advanced features' include *inter alia* arithmetic expressions, arrays, and the *call* statement. A punched card environment is assumed throughout. Yet for the intended readership—'people with no previous knowledge of computers'—this book is much better than many which are currently available. It always explains why particular coding techniques are chosen and it makes the reader aware of the virtues and dangers of each language feature.

The best post-ANSI 1974 tyro's book I have seen is McCracken's latest offering (Wiley, 1976) despite its use of flowcharts and its failure to treat indexed and relative files adequately. Yourdon's book, for all its virtues, cannot match McCracken's for price, elegance or, most important, accuracy—the naive reader, of all people, should be spared the confusion and insecurity which result from even a few errors of detail.

De Marco's is another 'structured' book from the Yourdon organisation. Structured analysis turns out to consist of disciplined use of a number of familiar tools: data flow diagrams, a data dictionary, 'Structured English' (resembling pseudo-code) and decision tables. The process of 'deriving a logical file structure' is akin to Codd's normalisation of relations, but there is no indication that the author is aware of the relational model—indeed, he declares himself indebted to two other Yourdon denizens who have remarked that his logical file structure is a 'third normal form set' from 'set theory'. De Marco argues strongly for the use of data flow as a means of partitioning the analysis, and for the use of all the above tools as a means of communicating with users ('something you can't show to a user is totally worthless as an analysis tool'), with much advice on how to make them acceptable to the users. Given this view, it is not surprising that the end-product of the analysis phase, the functional specification (called here, of course, 'structured specification') is itself expressed in terms of the above tools. Data flow diagrams are supported by the data dictionary, the processes being elaborated by Structured English or decision tables as appropriate. The information which the author recommends be included in the specification could well be supplemented (see the paper by S. J. Waters in this journal, vol. 22, no. 3) without affecting the approach. The author is at pains to specify methods which eliminate redundancy from the specification; he dismisses the unrealistic and undesirable notion of a 'frozen' specification and regards maintainability as being of prime importance.

The practising systems analyst will find this book painless to read; the exposition is clear, the pace slow, and the arguments strongly presented.

JIM INGLIS (London)

Digital Networks and Computer Systems 2nd Edition by Taylor L. Booth, 1978; 592 pages. (*John Wiley*, £14·75)

The first edition of this book, published in 1971, pioneered the unified hardware/software approach to the introduction of digital and computer systems which is gaining popularity in universities on both sides of the Atlantic. Indeed, the increasing realisation that this grey area between Mathematics and Engineering is fast becoming a third discipline indicates the need for such a book. The second edition has the same form as the successful 1971 text, but the content has been consolidated and enhanced. It is aimed at the first year undergraduate who wishes to invest his meagre funds in a single textbook covering introductory courses on logic design, digital systems, computer organisation and computer programming.

Taylor Booth adopts a rather formal, but not pedantic, approach to the specification and design of digital information processing systems. Hardware and software implementations of algorithmic processes are given equal weighting. However the means of implementation show the age of the original text. Hardware realisations are mainly at the *SSI* gate level; *MSI* and *LSI* devices are hardly mentioned. A simulated educational computer, called SEDCOM, is used to illustrate the principles of computer organisation and assembly language programming. SEDCOM is a rather thinly disguised version of the PDP 8 which is rather too long-in-the-tooth for many readers.

I recommend this book to the serious student as a good foundation for more advanced study.

R. M. LEA (Uxbridge)