Consiglio Nazionale delle Ricerche

B4-32

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

TESTING GLOBAL AUTOMATIC QUADRATURE PROGRAMS

P. Favati, G. Lotti, F. Romani

Nota interna B4-32
Luglio 1990

TESTING GLOBAL AUTOMATIC QUADRATURE PROGRAMS

Paola FAVATI(1), Grazia LOTTI(2) and Francesco ROMANI(3)

(1)*IEI-CNR Via S. Maria 46, 56100 Pisa, Italy*
(2)*Dipartimento di Matematica ed Informatica, University of Udine, Udine, Italy*
(3)*Dipartimento di Informatica, University of Pisa, Corso Italia 40, 56100 Pisa, Italy*

Abstract - A new method for testing global automatic quadrature programs is introduced. This method, with only one run, gives complete information on the behaviour of the integration program for a single function and any user tolerance, and is well suited to statistic and parametric studies with a large number of integrands. The method is applied to study the behaviour of some well known global automatic quadrature routines (QAG, QAGS, QXG, QXGS).

1. Introduction

In a paper of 1977 [13] Lyness and Kaganove introduced a statistical technique for evaluating and comparing automatic quadrature routines. In order to produce their statistics, a quadrature program has to be run on several functions with several different user tolerances. The technique presented in the following allows performing a more accurate analysis for global automatic quadrature programs [2,14] with only one run for each test integrand, thus highly reducing the computational costs.

The basic idea is that, in a global automatic quadrature program, an approximation of the integral, together with an estimate of the error, is available at any time. Hence, by observing the evolution of the approximation to the integral and of the error estimate during a single run, carried out as long as possible, the behaviour of the program on a given integrand for any user tolerance becomes clear.

In section 2 the tests on a single integrand are introduced and some performance measures are discussed. In section 3 the analysis is extended to groups of integrands and the resulting statistical measures are presented for some well known global quadrature programs. Finally, in section 4 some techniques to regularize the performance of programs on a group of integrand are investigated. In this way, the comparison among different programs is made possible.

In the following we often use the negative base-10 logarithms of the quantities denoting errors; this convention will make easier plotting and printing data, but it should be remarked that, in these cases, inequalities are reversed.

2. Testing a quadrature program on a single integrand

A generic global automatic quadrature routine can be outlined as in the following algorithm.

Algorithm 2.1.

```
procedure quadrature(a,b,epquad: real; var:  abserr,result: real; var ier: integer);

{ on input: }
{ a, b  are the extremes of the integration interval }
{ epquad  is the absolute error  tolerance, specified by the user }
{ on output: }
{ abserr is the absolute error  tolerance, estimated by the program }
{ result is the integral value, estimated by the program }
{ ier is the error condition:  ier = 0 means no error detected }

     var  lres,lest,lres1,lest1,lres2,lest2,tol,middle:  real;
begin
     ier:=0;
     local(a,b,result,abserr);                       { integrate on the first interval }
     put_interval(a,b,result,abserr);                { put the interval in the queue }
     while abserr > epquad do begin
          get_interval(a,b,lres,lest);          { get the interval with the largest error }
          middle:=(a+b)/2                       { perform bisection }
          local(a,middle,lres1,lest1);
          local(middle,b,lres2,lest2);
          put_interval(a,middle,lres1,lest1);
          put_interval(middle,b,lres2,lest2);
          result:=result-lres+lres1+lres2;           { update estimates }

 {If any error condition is reported, set the appropriate value for  ier and  exit}

          abserr:=abserr-lest+lest1+lest2;

 {If the program uses extrapolation, the values of result and abserr can be altered here;}
          ...
     end;
end;
```

When the program stops, one has:

$$\text{abserr} \leq \text{epquad}, \quad \text{ier} = 0,$$

or

$$\text{abserr} > \text{epquad}, \quad \text{ier} \neq 0.$$

The latter circumstance is called *quit*. We assume that the program is enough clever to avoid looping forever with too small tolerances; typically the program stops when the data structure which retains the list of active subintervals is full or some roundoff or other numerical troubles are detected. We define epquad_stop the minimal value of epquad for which the program stops without any error condition. Note that, when running the program with epquad $\leq$ epquad_stop, we get abserr = epquad_stop and no further improvement on abserr can be achieved.

The program can be slightly altered in order to communicate to the outer world the pair of values abserr, result each time the termination test is performed. This can be obtained by inserting a call to a test routine.

Algorithm 2.2.

```
procedure quadrature(a,b,epquad  : real; var : abserr,result :real; var ier:integer);
...
     while abserr > epquad do begin
...
          TEST_CALL(abserr,result);
     end;
end;

procedure TEST_CALL(abserr,result : real);
     { es is a global real variable containing the exact integral }
     { old_estimate is a global real variable initialized to the value -maxint }
     { neval is a global integer variable containing the actual number of functional
          evaluations }
     { ord(x) is a real function which computes $-\mathrm{Log}_{10}\, x$ }
     var err,est: real;
begin
     err:=ord(Abs(es-result));               { compute the true error}
     est :=ord(abserr);
     if est > old_estimate then begin
          write(est,err,neval);
          old_estimate:= est
     end
end;
```

The routine TEST_CALL uses the true value of the integral es (clearly this is possible only in experimental environments) to compute a sequence of triples (est_i, err_i, $neval_i$) with $est_i < est_{i+1}$. The main property of this sequence of values is that, if the program is called with an absolute tolerance epquad = 10^{-t}, such that

$$est_{i-1} < t \leq est_i,$$

the program will stop with a true error 10^{-err_i} and $neval_i$ evaluations. This consideration can be expressed as follows:

a) the function which associates the true error to the requested tolerance is the piecewise constant (left continuous) function

$$err(t) = \begin{cases} err_1, & t \leq est_1 \\ err_i, & est_{i-1} < t \leq est_i,\ i>1 \end{cases}$$

defined in the range $(-\infty, t_{stop}]$, $t_{stop} = -\mathrm{Log}_{10}$(epquad_stop);

b) the function which associates the number of functional evaluations to the requested tolerance is the piecewise constant (left continuous) function:

$$\text{neval}(t) = \begin{cases} \text{neval}_1, & t \le \text{est}_1 \\ \text{neval}_i, & \text{est}_{i-1} < t \le \text{est}_i,\ i>1 \\ \text{neval}(t_{\text{stop}}), & t \ge t_{\text{stop}} \end{cases}$$

defined in the range $(-\infty, \infty)$, since the program performs functional evaluations even if the computation is not successful.

When running the program with a user tolerance so small to cause the integration process to quit, we get, with only one run, the value of t_{stop} and the complete plot of neval(t) and err(t).

Example 2.1. Evaluating the integral $\int_0^1 | x - \pi/4 |^{0.2} dx$ with the automatic quadrature routine QAG from QUADPACK [15], with requested absolute tolerance epquad = 10^{-13}, we get the following plots for err(t) and neval(t).

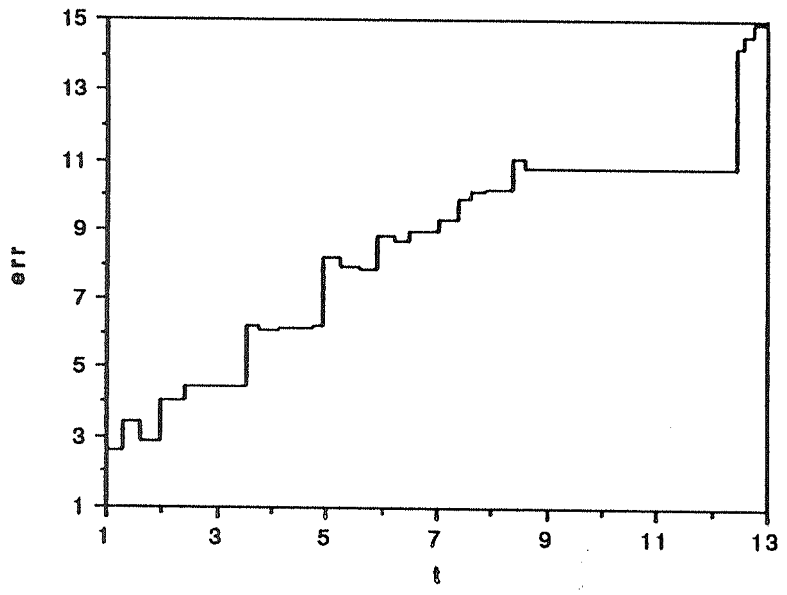

Fig. 1.

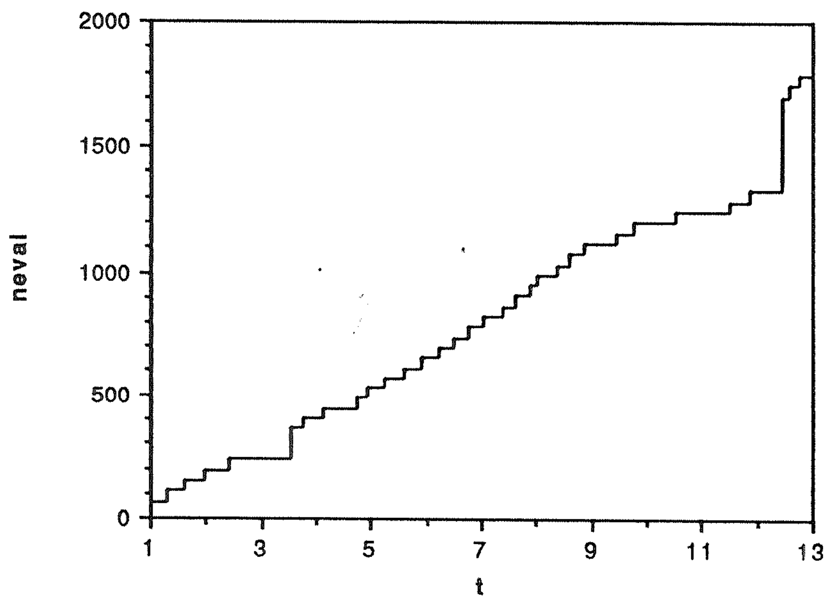

Fig. 2

❑

Clearly the function err(t) can be computed only in test environments where the exact integral is known. In order to decide whether the program works correctly, err(t) may be compared either with the error estimate abserr given by the program or with the requested user tolerance epquad = 10^{-t}.

In the first case a measure lr1(t) can be defined, such that

$$\frac{|es - result|}{abserr} = 10^{-lr1(t)}.$$

It is easy to see that lr1(t) can be expressed as

$$lr1(t) = \begin{cases} err_1 - est_1, & t \leq est_1 \\ err_i - est_i, & est_{i-1} < t \leq est_i,\ i>1 \end{cases}$$

In the second case the corresponding logarithmic measure is

$$lr2(t) = err(t) - t.$$

Example 2.2. For the same test of example 2.1 we get the following plots for lr1(t) and lr2(t).

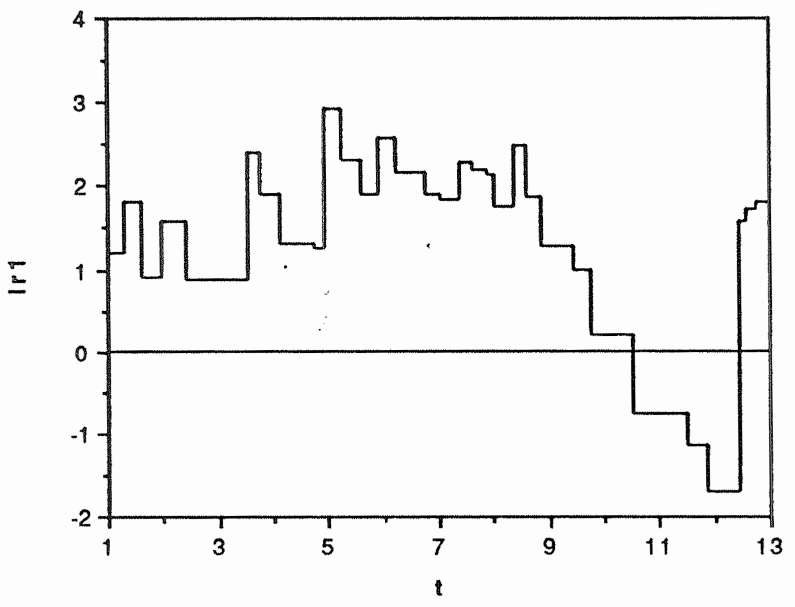

Fig. 3

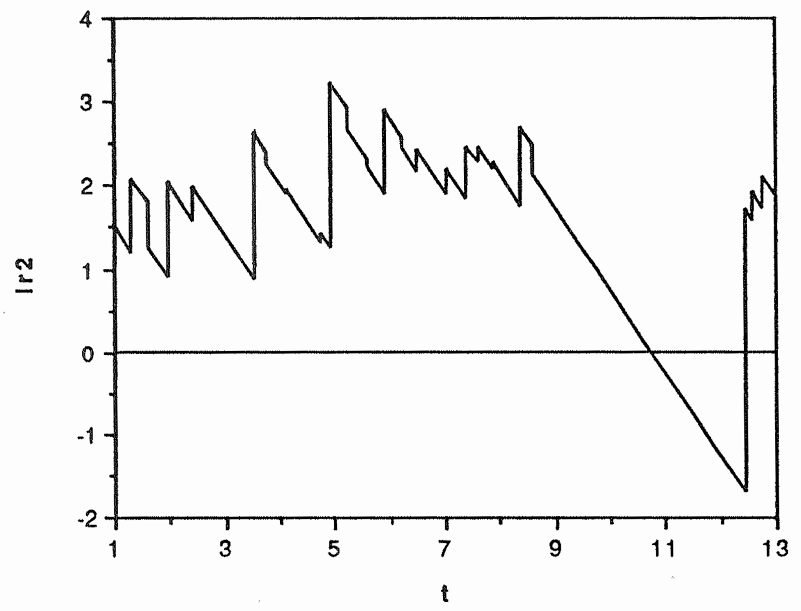

Fig. 4

❑

It is remarkable that lr1(t) < lr2(t) and the three possible pair of signs of the two measures have the following meaning:

lr1(t)>0,	lr2(t)>0	the program integrates correctly at the requested tolerance 10^{-t} and the given estimate is correct.

lr1(t)<0,	lr2(t)>0	the program integrates correctly at the requested tolerance 10^{-t} but the given estimate is not correct.
lr1(t)<0,	lr2(t)<0	the program fails to integrate correctly at the requested tolerance 10^{-t} and, clearly, the given estimate is not correct.

The choice of lr1(t) or lr2(t) as a measure of error implies a different opinion on the duties of a quadrature program. If we are interested in the error estimate given by the program we can choose lr1(t), but this choice can lead to misleading conclusion. Consider the following isolated set of results:

$$\text{epquad} = 10^{-3}; \qquad |\text{es} - \text{result}| = 10^{-12}; \qquad \text{abserr} = 10^{-13}.$$

The result produced by the program is very good for any user which requires 3 digits of accuracy and it is hard to consider this event a failure. If the goal is to integrate correctly at the given tolerance then the measure lr2(t) has to be used, which is independent of the program error estimate abserr. This latter choice will be adopted in the following.

3. Testing on a group of integrands

In literature both battery experiments and parametric studies of problem families are used to test automatic quadrature routines [1,3,5,9,10,11,12,16]. Typically a battery test consists in performing several integrations on a group of functions with widespread characteristics, (e.g. well behaved, peak, singular, oscillating, step functions). A serious drawback of this technique is that only few elements of a family of functions are tested and, possibly, there exist functions very close to the elements of the sample for which the quadrature routine exhibits a very different behaviour. The other approach uses a problem family, each of whose members can be selected by specifying a parameter λ. The whole family is tested by varying the parameter in a given interval either with a deterministic or a random selection.

Both approaches can be unified by considering a *test set* formed by few families of parametric functions, for which the parameter can assume both deterministic and random values. However it is unlikely that the use of a single test set will give complete information on the behaviour of a quadrature routine. One can build his own test set which presents the characteristics of the functions he has to integrate. On the other hand people writing or exploring numerical quadrature routines will prefer to use several general test sets looking for common characteristics of the program. In the Appendix, two test sets are presented which contain few families of functions with different characteristics, each depending on a real parameter λ. Any test set may be built by using an arbitrarily large number of values of λ, a fixed proportion of which is deterministically chosen and the other ones are random numbers. For each function the analytical exact integral is computed. Moreover we force each exact integral to lie in the interval [1,2] by properly scaling the function, so that we are allowed to deal with homogeneous values of the absolute errors; this approach is somewhat equivalent to use relative errors.

If the experiments of the previous section are carried out on a test set

$$T = \{f_s(x),\ s=1,2,\dots,m\},$$

some statistics are needed to interpret the results. Let

$$\mathrm{neval}^{(s)}(t),\ \ \mathrm{lr2}^{(s)}(t), \qquad s=1,2,\dots,m,\ \ t \in (-\infty, t_{stop}^{(s)}]$$

be the results of the test. We define the following statistical estimators:

$$\mathrm{quit}(t) = \#\{s:\ t_{stop}^{(s)} < t\};$$

$$\mathrm{quit}_{av}(t) = \frac{100}{m}\ \mathrm{quit}(t);$$

$$\mathrm{succ}_{av}(t) = 100\ \#\{s:\ \mathrm{lr2}^{(s)}(t) \geq 0,\ t \leq t_{stop}^{(s)}\}\ /\ \#\{s:\ t \leq t_{stop}^{(s)}\} =$$

$$= \frac{100}{m-\mathrm{quit}(t)}\ \#\{s:\ \mathrm{lr2}^{(s)}(t) \geq 0,\ t \leq t_{stop}^{(s)}\};$$

$$\mathrm{neval}_{av}(t) = \frac{1}{m} \sum_{s=1}^{m} \mathrm{neval}^{(s)}(t).$$

Once given a quadrature program quadrature and a test set T, the plots of the estimators $\mathrm{quit}_{av}(t)$, $\mathrm{succ}_{av}(t)$, $\mathrm{neval}_{av}(t)$ can give complete information on the mean behaviour of the program on T.

Another statistical measure, in some sense equivalent to the percentage of successes but homogeneous with $\mathrm{err}^{(s)}(t)$, is the following:

$$\mathrm{perc}_{\alpha}(t) = \max \left\{ y :\ \frac{100}{m-\mathrm{quit}(t)}\ \#\{s,\ \mathrm{err}^{(s)}(t) \geq y,\ t \leq t_{stop}^{(s)}\} = \alpha \right\}$$

we can observe that

$$\mathrm{perc}_{\alpha}(t) = t \qquad \Rightarrow \qquad \mathrm{succ}_{av}(t)=\alpha.$$

This measure will be used in the next section.

It is easy to see that $\mathrm{quit}_{av}(t)$, $\mathrm{succ}_{av}(t)$, $\mathrm{neval}_{av}(t)$, $\mathrm{perc}_{\alpha}(t)$ are piecewise constant (left continuous) functions, moreover $\mathrm{quit}_{av}(t)$ and $\mathrm{neval}_{av}(t)$ are not decreasing. Using the techniques of section 2 it is possible to keep track of all the points of discontinuity of the statistical measures (i.e. the set of the points $est_i(s)$ for any i and s). However, in such a way, for large test sets the amount of information becomes intractable; a more practical solution is to compute and to plot discrete approximations of the above defined measures. In particular the interval [1,14] is splitted into n intervals of equal length, J_i, i=1,2,...,n, and the values

$$\min \{succ_{av}(t): t \in J_i\}, \quad i=1,2,\ldots,n,$$

are computed, thus guaranteeing that no points with small percentage of successes escape the analysis. In this sense our analysis is more accurate than the classical method of sampling on a discrete set of values of t.

As an example of application of this technique, we have tested four of the most efficient global automatic quadrature programs (QAG, QAGS, [15] and their improvements QXG, QXGS [8]).

QAG is a simple globally adaptive integrator that uses, as local quadrature module a pair of Gauss-Kronrod integration formulas (in our test we use the option key=2, i.e. a 10-21 Gauss-Kronrod pair is selected). The adaptive strategy attempts to reduce the error by subdividing the interval with the largest error estimate; all subdivisions are bisections.

QAGS is an integrator based on globally adaptive interval subdivision in connection with extrapolation. The local quadrature module uses a pair of Gauss-Kronrod integration formulas with 10 and 21 points. The extrapolation is carried out by means of the ε-algorithm [17]. The FORTRAN code of QAG and QAGS has been received via electronic mail [4].

QXG and QXGS are obtained by replacing, in QAG and QAGS, the Gauss-Kronrod formulas with symmetric, closed, interpolatory integration formulas with positive weights and increasing degree of precision called recursive monotone stable (RMS) formulas. These formulas allow applying higher order or compound rules without wasting previously computed functional values and are well suited for automatic adaptive quadrature [7].

The programs were run, in the double precision version, on an Apple Macintosh® SE/30 with MC68882 numeric coprocessor. Two test sets, T1 and T2, consisting of 56000 and 40000 integrands, respectively, were used. More information on T1 and T2 is given in the Appendix.

The plots of $quit_{av}(t)$, $succ_{av}(t)$, $neval_{av}(t)$, are presented in pairs, to allow direct comparison of routines of the same kind (QAG and QXG, QAGS and QXGS).

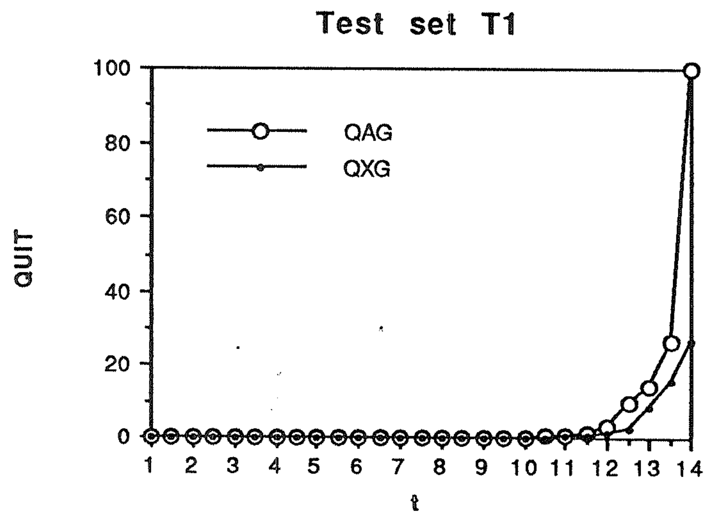

Fig. 5

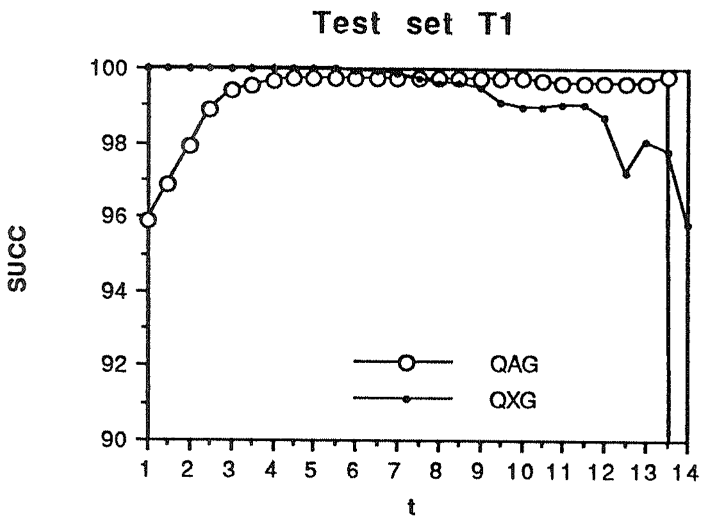

Fig. 6

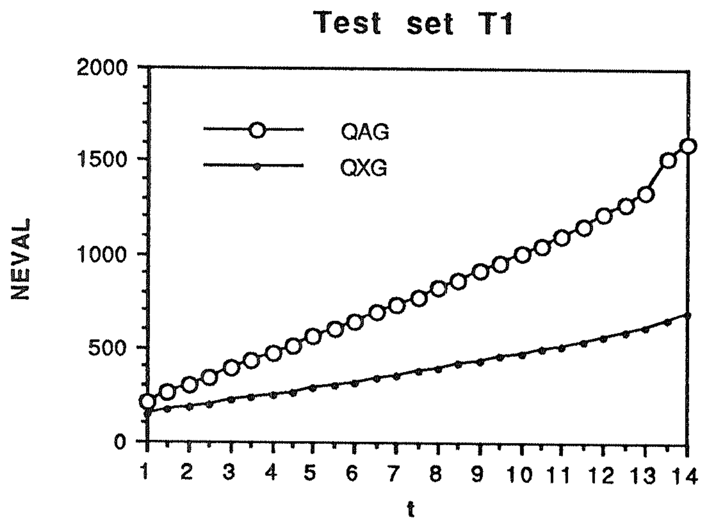

Fig.7

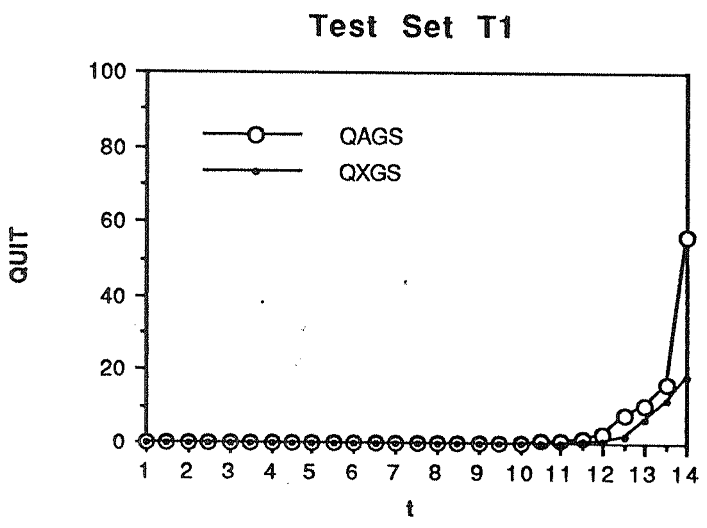

Fig. 8

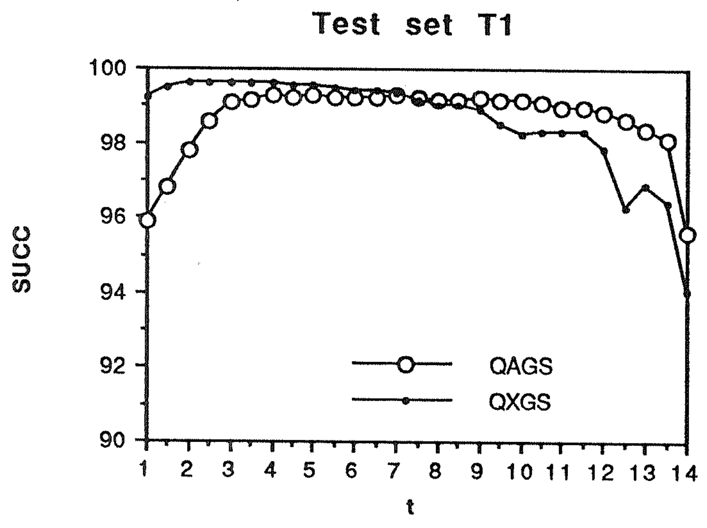

Fig. 9

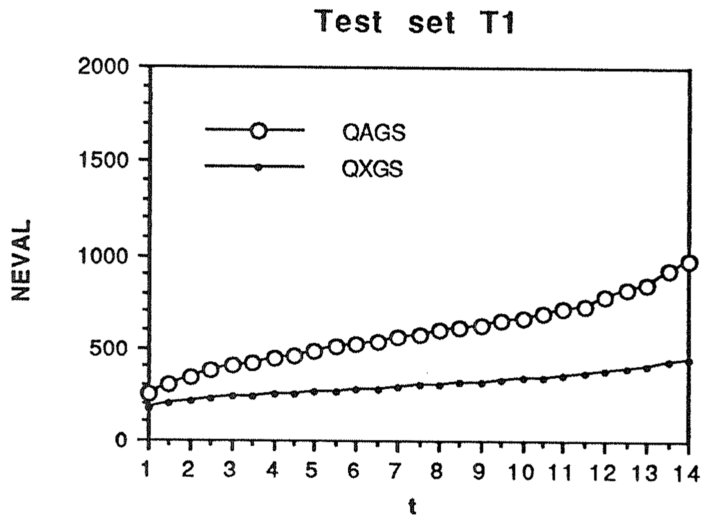

Fig. 10

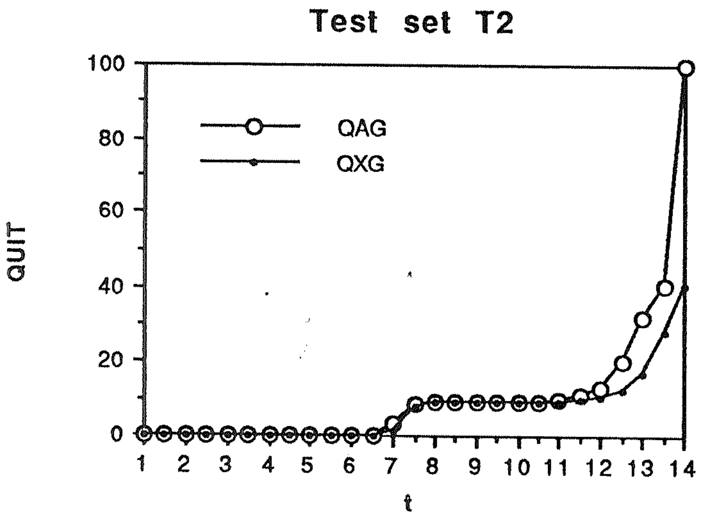

Fig. 11

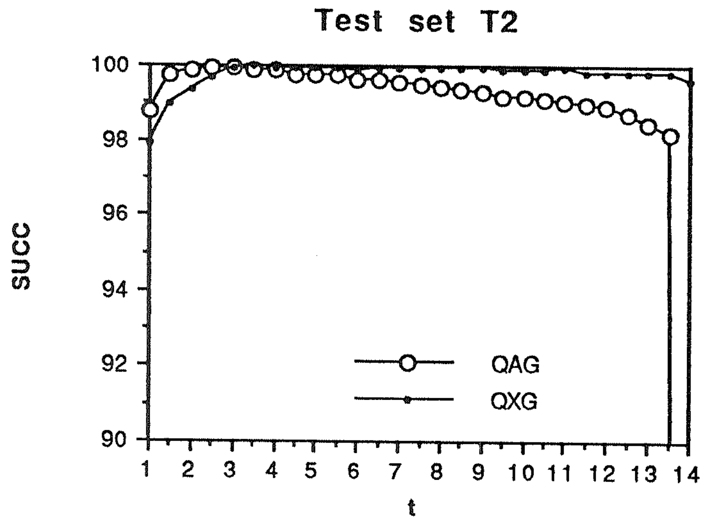

Fig. 12

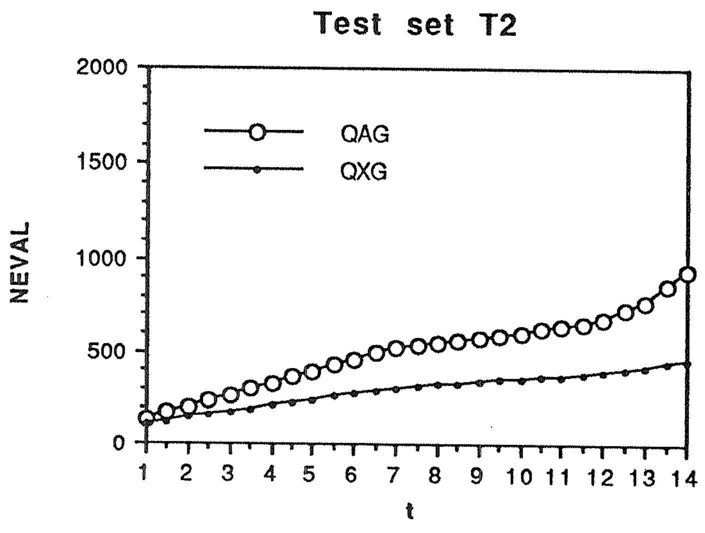

Fig. 13

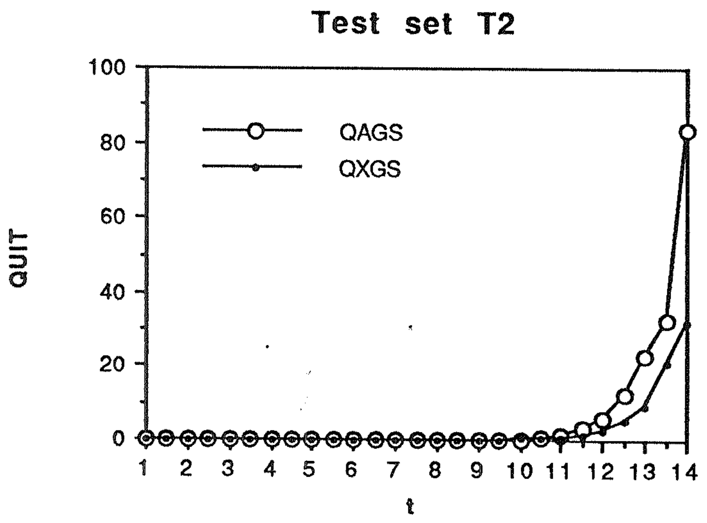

Fig. 14

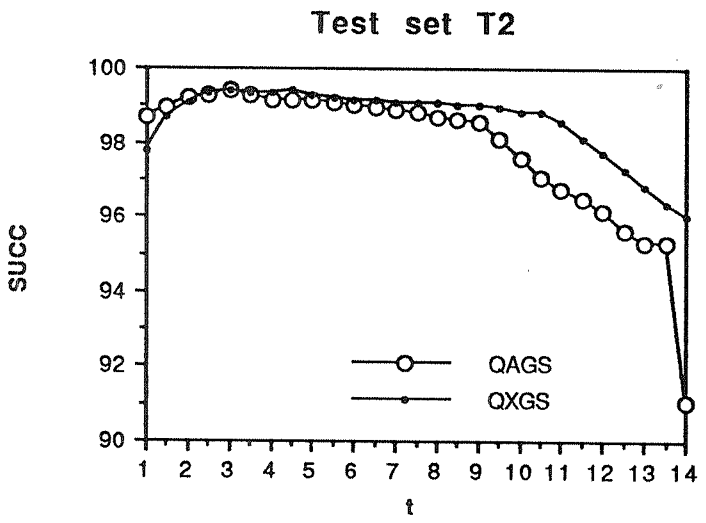

Fig. 15

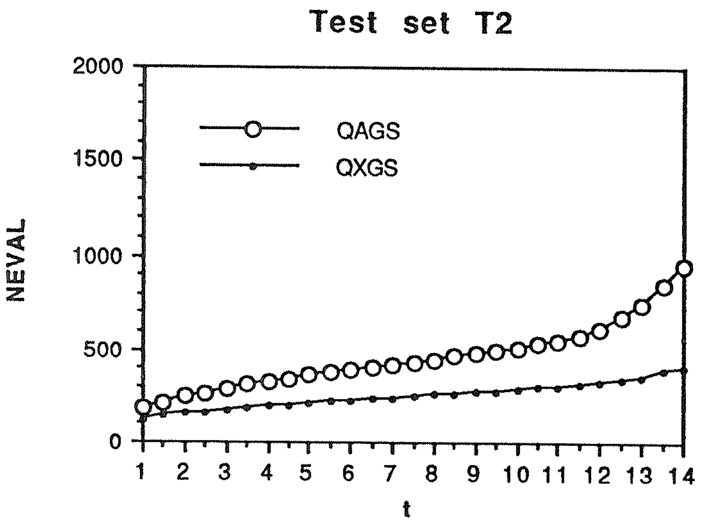

Fig. 16

From the analysis of these plots becomes clear that the behaviour of the error is similar in all routines. For what concerns the number of evaluations and quits, the extrapolated routines are much better than the not extrapolated ones and the improved routines of [8] are better than the older ones.

4. Improving and comparing quadrature programs

A good quadrature program should present, on any test set, the same high reliability for a range of required tolerances as large as possible (e.g. $succ_{av}(t)=99\%$, $1<t<14$). This property, as one can see from the above examples, is typically not satisfied, and techniques to alter the performance of the quadrature program have to be investigated.

A typical trick is to call the program with a internal tolerance (say 10^{-q}) different from the requested user tolerance 10^{-t}. Lyness and Kaganove [13] apply this device by testing with a pair of values $(\varepsilon_{req}, \varepsilon_{quad}) = (10^{-t}, 10^{-q})$ independently varying in a rectangle. They introduce some statistical quantities which can be easily related to other ones used in our work.

Another approach can be to consider q as a monotonic increasing function of t, i.e. the quadrature program is run with an actual tolerance 10^{-q} which is a decreasing function of the user tolerance10^{-t}. In this case we obtain a different quadrature program (called in the following the *modified program*) which can be tested with the techniques explained above; q(t) is called the *associated function* to the modified program. The quantities $\underline{err}(t)$ and $\underline{neval}(t)$, estimating the performance on a single function of the modified program, can be obtained from the corresponding quantities err(t) and neval(t) as follows:

$$\underline{err}(t) = err(q(t)), \qquad q(t) \le t_{stop},$$

and

$$\underline{neval}(t) = neval(q(t)), \quad q(t) \le t_{stop}.$$

The statistical measures on a test set are subject to similar changes, as well:

$$\underline{quit}_{av}(t) = quit_{av}(q(t));$$

$$\underline{neval}_{av}(t) = neval_{av}(q(t));$$

$$\underline{perc}_{\alpha}(t) = perc_{\alpha}(q(t)).$$

Now we try to regularize the plot of $\underline{perc}_{\alpha}(t)$ for a fixed α looking for a suitable function q(t); in order to obtain a ratio of about $\alpha/100$ successes, we need to find a monotonic increasing function q(t) such that:

$$perc_{\alpha}(q(t)) \approx t.$$

If the function $\mathrm{perc}_\alpha(u)$ would be strictly increasing we could solve exactly the previous equation with $q(t) = \mathrm{perc}_\alpha^{-1}(t)$. Since $\mathrm{perc}_\alpha(u)$ is a piecewise constant function, we consider a continuous (possibly not monotonic) function $p(u)$ approximating it in some sense on a given interval $[u_0,u_1]$.

Let us introduce the not decreasing function

$$mp(u) = \max \{p(x): u_0 \le x \le u\}$$

and define the set:

$$I = \{u:\ mp(x) < mp(u),\ u_0 \le x < u\}.$$

Since $p(u)$ is continuous, the set I can be written as a finite union of intervals:

$$I = (x_0, y_0] \cup (x_1, y_1] \cup \ldots \cup (x_k, y_k],\ \ x_0 = u_0,\ y_k \le u_1$$

and the restriction of $p(u)$ on I (say $\eta(u)$) is a monotonic increasing function whose inverse $q(t)$ has the property

$$p(q(t)) = t,\ \ t \in (p(u_0), p(y_k)].$$

Typically, one has a finite set of values of $\mathrm{perc}_\alpha(t)$; the following example shows a possible choice for $p(u)$ in this case.

Example 4.1. Given the sequence $p_i = \mathrm{perc}_\alpha(u_i)$, $i=1,2,\ldots k$, $u_1 < u_2 \ldots < u_k$, let $p(u)$ be the piecewise linear function joining the points (u_i, p_i), $i=1,2,\ldots k$, and $\eta(u)$ be defined as above. It is easy to see that the resulting $q(t)$ is defined in the interval $(p_1, \max \{p_i, i=1,2,\ldots k\}]$. ❑

REMARK. A modified quadrature program with associated function $q(t) = \eta^{-1}(t)$, can be tested with the technique of section 2, by simply applying the transformation $\underline{est_i} = \eta(est_i)$, $est_i \in I$ and neglecting the values $est_i \notin I$; thus the main consequence of using $q(t)$ is to skip the intervals where $\mathrm{perc}_\alpha(t)$ does not increase.

Example 4.2. From the test of QAGS on the set T1 (m=56000), using the technique of Example 4.1, a function $\eta_{QAGS}(u)$ has been derived. The quadrature program modified by the associated function $q_{QAGS} = (\eta_{QAGS})^{-1}$ has been applied on test set T1. The plots of $\underline{succ}_{av}(t)$, $\underline{neval}_{av}(t)$,in the standard and modified cases are presented in the following.

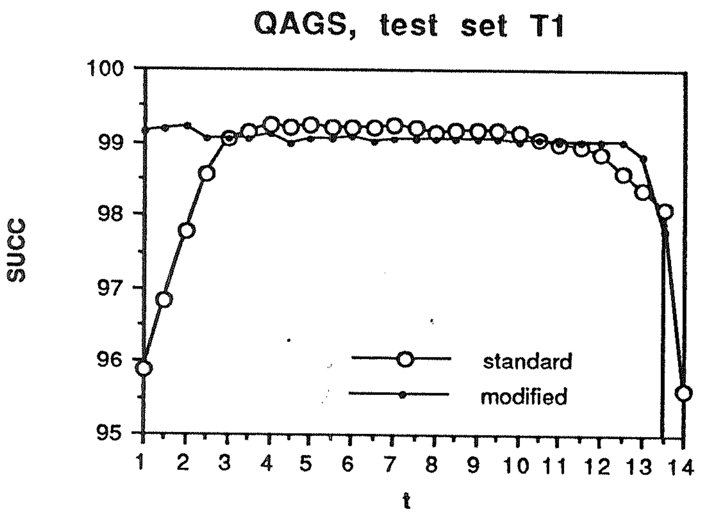

Fig. 17

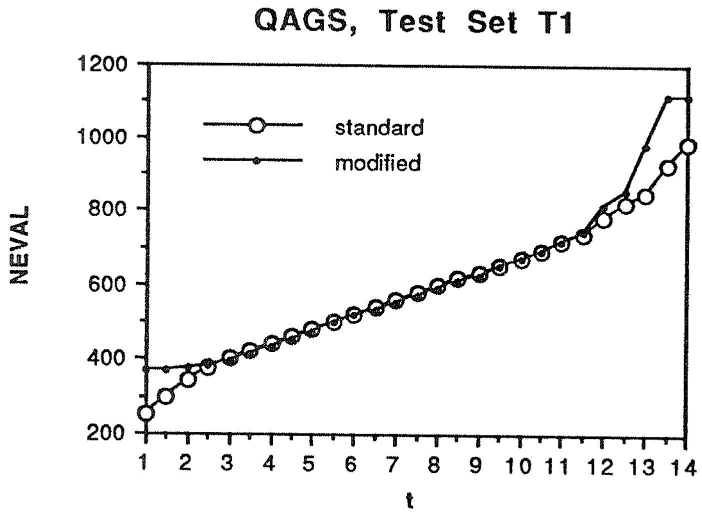

Fig. 18

❑

We note that this technique allows a good regularization only on a given test set. It cannot be viewed as a practical method to improve an integration program unless the user is sure to integrate only functions in that test set. On the other hand, if the plots of $\underline{\text{succ}}_{\text{av}}(t)$ are regularized on a given test set, the corresponding plots of $\underline{\text{neval}}_{\text{av}}(t)$ and $\underline{\text{quit}}_{\text{av}}(t)$ allow a fair comparison of the performances of different routines (this fact was pointed out in [13], as well).

Example 4.3. Using the technique of Example 4.1, we have derived the functions $\eta_{\text{QAGS}}(u)$ and $\eta_{\text{QXGS}}(u)$ which regularize $\underline{\text{succ}}_{\text{av}}(t)$ on the set T1 (m=56000). The plots of $\underline{\text{succ}}_{\text{av}}(t)$, $\underline{\text{quit}}_{\text{av}}(t)$, $\underline{\text{neval}}_{\text{av}}(t)$, after applying the proper modified quadrature programs are presented in the following. It is apparent how the improved routine QXGS outperforms the older one.

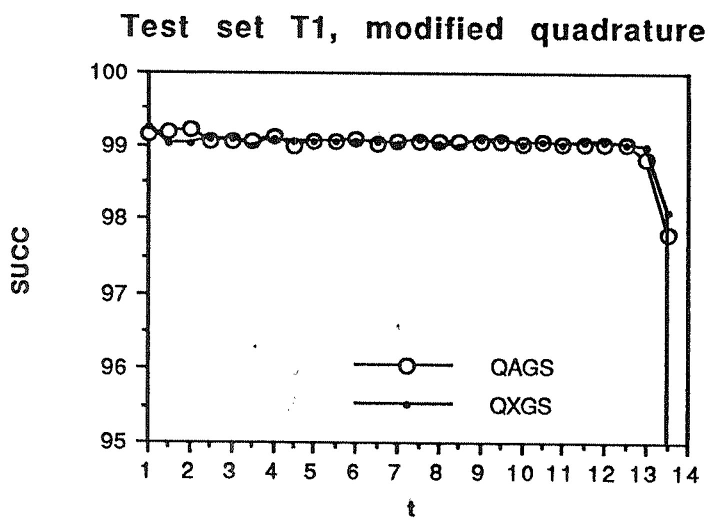

Fig. 19

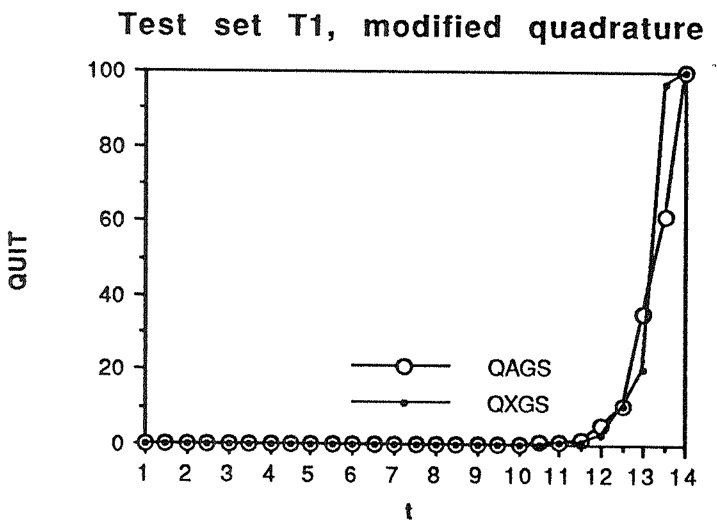

Fig. 20

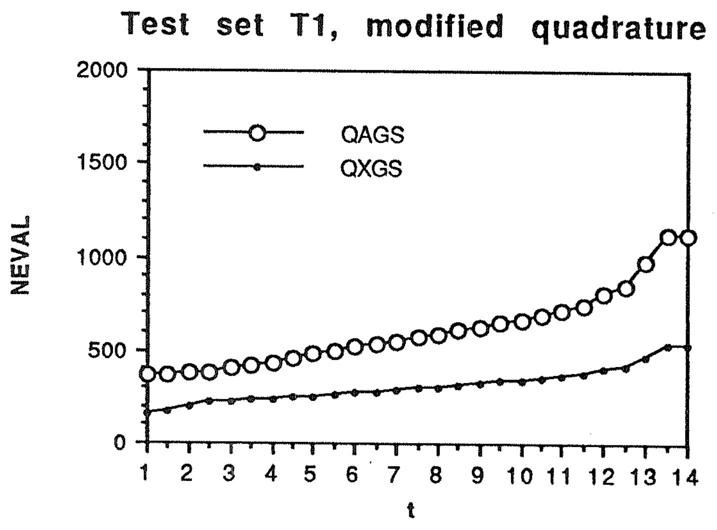

Fig. 21

Appendix. Test sets used in the numerical experiments.

Let $f_i(x,\lambda)$, $i=1,\dots,k$, be k real functions defined for all $x\in[a_i,b_i]$, $\lambda\in[0,1]$. The test set is:

$$T_h = \{f_i(x,\lambda),\ i=1,\dots,k,\ \lambda=j/2^p,\ j=0,1,\dots,2^p\} \cup \{f_i(x,\lambda),\ i=1,\dots,k,\ \lambda=\lambda_q,\dots,\lambda_{h-1}\},$$

where $p=\lfloor\log_2 h\rfloor-1$, $q=2^p+1$ and the λ_i are random values in [0,1]. The set contains m = h k functions; h has to be enough large in order to get significant results. We used the following values of h.

h	# deterministic λ	# random λ	ratio
250	64	186	0.344
1000	256	744	0.344
4000	1024	2976	0.344

Since the seeds of the pseudorandom generator are independent of h it is easy to see

$$T_{250} \subset T_{1000} \subset T_{4000}.$$

The following plot, shows the values of $succ_{av}(t)$ for QAGS on test set T1 with h=250,1000,4000.

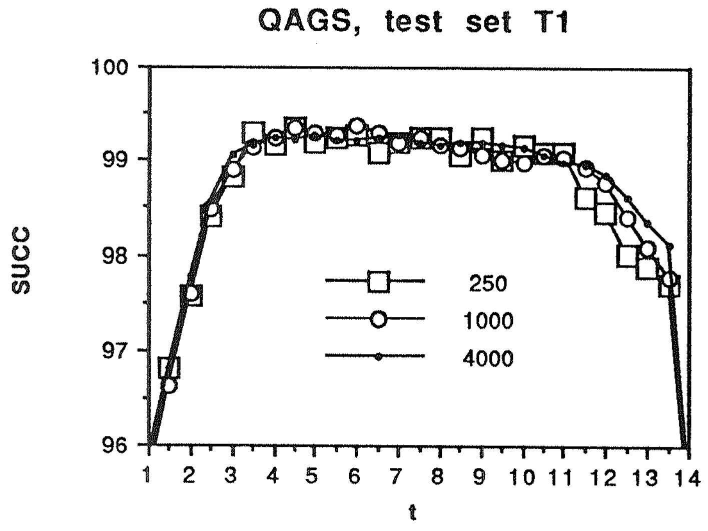

Fig. 22

Now the two test sets used in our computations are described.

Test Set T1. Fourteen families of functions are used, each family depends on a parameter λ and the integration interval depends on a parameter β. For each family if λ has a deterministic value then β assumes the fixed value 1/2, otherwise also β is randomly chosen in [0,1].

The functions and the integration intervals are the following:

1)	$(x-1/2)(x+\lambda^3-1/2)(x+\lambda^5-1/2)(x+\lambda^6/4-1/2)$	$[0,\beta+1/2]$
2)	$e^{-2\pi x}\,\mathrm{Sin}\,(160\lambda+20)\pi x$	$[0,\beta+1/2]$
3)	$x^{1/(8\lambda+2)}$	$[0,\beta+1/2]$
4)	$x^{-1/(8\lambda+2)}$	$[0,\beta+1/2]$
5)	$x^{-(4\lambda+1.5)}$	$[10^{-5}+\beta/2000,\beta+1/2]$
6)	$x^{4\lambda+1.5}$	$[0,\beta+1/2]$
7)	$x^{2\lambda^2}\,\mathrm{Sin}\,x^{2\lambda^2+1}$	$[0,(\beta+2)\pi]$
8)	$(1-Cx^2)^{-1}$, $C=9/10000\lambda+999/1000$	$[\beta/2,1]$
9)	$(C^2+(1-x)^2)^{-1}-(2C^2+(1.1-x)^2)^{-1}$, $C=0.05+\lambda^2/10$	$[0.9-\beta/2,1.2+\beta/2]$
10)	$x^{\lambda}\,\mathrm{Log}\,x$	$[0,\beta+1/2]$
11)	$x^{-0.8\lambda}\,\mathrm{Log}\,x$	$[0,\beta+1/2]$
12)	$-e^x,\ x\le\lambda;\ \ e^x,\ x>\lambda$	$[-\beta-1/2,\beta+1/3]$
13)	$\mathrm{Log}\,\lvert\lambda-x\rvert$	$[0,\beta+9/8]$
14)	$\lvert x-1/3\rvert^{8\lambda}$	$[0,\beta+1/2]$.

❑

Test Set T2. Ten families of functions are used, each family depends on a parameter λ and the integration interval depends on a parameter β. For each family if λ has a deterministic value then β assumes the fixed value 1/2, otherwise also β is randomly chosen in [0,1]. Most functions are taken, with modifications, from the test set of [6]

The functions and the integration intervals are the following:

1)	$\lvert x-\lambda\rvert^{-1/2}$;	$[0,\beta+1/2]$
2)	$0,\ x\le\lambda;\ \ e^{x/2},\ x>\lambda$	$[0,\beta+1/2+\lambda]$
3)	$e^{-2\lvert x-\lambda\rvert}$	$[0,\beta+1/2]$
4)	$10/(1+100(x-\lambda)^2$,	$[0,\beta+1/2]$
5)	$100/\mathrm{Cosh}(C(-1-\lambda+x))$, $C=100\,\mathrm{Log}(2+3^{1/2})$	$[1,2+\beta/2]$
6)	$C_0/(1+C_1\,\mathrm{Sin}\,x)$, $C_0=1/(2+8\lambda)$, $C_1=(1-C_0^2)^{1/2}$	$[0,2\pi+\beta/2-1/4]$
7)	$\mathrm{Cos}\,(1+Cx)$, $C=10^{1+\lambda}$	$[0,\beta+1/2]$
8)	$x^2\,\mathrm{Sin}\,(1+Cx)$, $C=10^{1+\lambda}$	$[0,\beta+1/2]$
9)	$e^{-4x}\,\mathrm{Sin}\,(1+Cx)$, $C=10^{1+\lambda}$	$[0,\beta+1/2]$
10)	$-\lambda^7+\lambda^6x+\lambda^3x^{30}+\lambda^5x^{30}-\lambda^2x^{31}-\lambda^4x^{31}-\lambda x^{60}+x^{61}$,	$[0,\beta+1/2]$.

❑

References

[1] J. CASALETTO, M. PICKET, J.R. RICE, *A comparison of some numerical integration programs*. Signum Newsletter 4 (1969), 30-40.

[2] P.DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration* (1984). Academic Press, New York.

[3] C. DEBOOR, *CADRE: an Algorithm for Numerical Quadrature*. In: Mathematical Software (1971), (J.R. Rice, ed.), Academic Press, New York, 417-449.

[4] J.J. DONGARRA, E. GROSSE, *Distribution of Mathematical Software via Electronic Mail*. Signum Newsletter 20 (1985), 45-47.

[5] H. ENGELS, *Numerical Quadrature and Cubature* (1980). Academic Press, New York.

[6] T.O. ESPELID AND T.SØREVIK, *A Discussion of a New Error Estimate for Adaptive Quadrature*. BIT 29 (1989), 283-294.

[7] P. FAVATI, G. LOTTI, F. ROMANI, *Interpolatory Integration Formulas for Optimal Composition*. To appear in ACM Trans. Math. Software.

[8] P. FAVATI, G. LOTTI, F. ROMANI, *ALGORITHM xxx: Improving QUADPACK Automatic Integration Routines*. To appear in ACM Trans. Math. Software.

[9] G.M. GENTLEMAN, *Implementing the Clenshaw-Curtis Quadrature, I Methodology and Experience*, Comm. ACM 15 (1972), 337-360.

[10] D.K. KAHANER, *Comparison of Numerical Quadrature Formulas*. In: Mathematical Software (1971), (J.R. Rice, ed.), Academic Press, New York, 229-259.

[11] J.N. LYNESS, *When not to Use an Automatic Quadrature Routine*. SIAM Rev. 25 (1983), 63-87.

[12] J.N. LYNESS, J.J. KAGANOVE, *Comments on the Nature of Automatic Quadrature Routines*. ACM Trans. Math. Software 2 (1976), 65-81.

[13] J.N. LYNESS, J.J. KAGANOVE, *A Technique for Comparing Automatic Quadrature Routines*. Comp. J. 20 (1977), 170-177.

[14] M.A. MALCOLM, R.B. SIMPSON, *Local versus Global Strategies for Adaptive Quadrature*. ACM Trans. Math. Software 1 (1975), 129-146.

[15] R. PIESSENS, E. DE DONCKER-KAPENGA, C. ÜBERHUBER AND D.K. KAHANER, *QUADPACK: A Subroutine Package for Automatic Integration* (1983). Springer, Berlin.

[16] I. ROBINSON, *A Comparison of Numerical Integration Programs*. J. Comput. Appl. Math. 5 (1979), 207-223.

[17] P.WYNN, *On a device for computing the $e_m(S_n)$ transformation*. Math. Comp. 10 (1956) 91-96.