# DQAINF: an algorithm for automatic integration of infinite oscillating tails

T.O. Espelid* and K.J. Overholt
*Department of Informatics, University of Bergen, Norway*

Received 17 May 1993; revised 14 March 1994
Communicated by M. Redivo Zaglia

We describe an automatic quadrature routine which is specifically designed for real functions having a certain type of infinite oscillating tails. The algorithm is designed to integrate a vector function over an infinite interval. A FORTRAN implementation of the algorithm is included.

The algorithm combines an adaptive subdivision strategy with extrapolation and requires that the decay of all the functions in the vector is the same. The algorithm is based on the assumption that the oscillating behavior is due to a periodic function with the property that it changes sign when evaluated at points of distance half a period. We assume that this period is known. The algorithm offers a choice of three different quasi-linear extrapolation procedures, namely the Euler transformation and two modifications of this transformation.

**Keywords:** Quadrature, automatic, adaptive, numerical integration, extrapolation, oscillatory integrals

## 1. Introduction

We know of only one published routine designed specifically for the integration of infinite oscillating tails, Lyness and Hines [17]. This is slightly surprising since a number of papers have, over the years, addressed such computational problems, e.g. Longman [13–15], Levin [11], Levin and Sidi [12], Sidi [20,21] and Hasegawa and Torii [8]. Piessens et al. [19] suggest using Longman's method combined with some extrapolation method; however, no code, tailored for this problem, is offered.

The available routine is based on a paper by Lyness [16]. The routine applies the Euler transformation on a series generated by partitioning the original integral into an infinite series of integrals all of the same interval length. Assuming that a periodic function is involved, for example a circular function or a Bessel function (asymptotically periodic), these interval lengths equal half the period of the periodic function. This way one achieves an oscillating series. This approach represents a modification of the original Longman method which implies using the zeros or the maximum points of the function involved.

* This work was supported by The Norwegian Research Council for Science and the Humanities.

The routine [17] is automatic in the sense that the user may specify a tolerance and then the routine will try to give an answer where the extrapolation error is within this tolerance. However, the error one introduces by applying the same quadrature rule on each subinterval is not taken into account in the routine. This responsibility rests with the user.

The new algorithm offered here will use the partition idea suggested by Lyness [16] (half periods) of the original interval. However, we will combine an adaptive strategy and extrapolation creating an automatic routine which takes into account the global error in the approximation.

We offer three alternative extrapolation methods: Overholt's *P*-order two algorithm [3,18], the Euler transformation and a suggested modification of the Euler transformation. All three transformations are two-point and quasi-linear. Overholt's algorithm is superior to the other two, which both are of *P*-order one, but requires more information.

This paper is organized as follows: in the next section we present the problem, create the oscillating series and establish some properties of that series. Then we present the different transformations and discuss their properties. Next we outline the basic algorithm and develop the error estimates. Then we give a number of examples and concluding remarks. Finally we present the code and the parameter list in the appendix.

## 2. The basic problem and the series approach

We consider the numerical evaluation of an infinite integral

$$I = \int_a^\infty f(x)\,dx, \tag{1}$$

where the integrand $f$ may be written

$$f(x) = \sum_{i=1}^{k} p_i(x) g_i(x), \quad k \geq 1. \tag{2}$$

All the functions $p_i(x)$ are assumed to be periodic, with the same period $2q$, such that at points of distance a half period $q$,

$$p_i(x+q) = -p_i(x), \quad \text{for } x \geq b \geq a,\ i = 1,2,\ldots,k. \tag{3}$$

Furthermore, the functions $g_i(x)$ all have a similar expansion in $1/x$ as $x \to \infty$,

$$g_i(x) \sim \sum_{j=0}^{\infty} c_j^{(i)} / x^{\gamma+j}, \quad i = 1,2,\ldots,k, \text{ with } \gamma > 0 \text{ and } x \geq b \geq a, \tag{4}$$

with $\sum_{i=1}^{k} |c_0^{(i)}| \neq 0$. We assume that for $x \geq b$ the expansion in (4) is either absolutely convergent, or it is an asymptotic expansion valid with sufficient accuracy for $x \geq b$. It is the responsibility of the user of this software to give an appropriate value of $b$.

For the validity of the operations on series required in the following, we refer to any standard treatise on this subject, e.g. Knopp [10] or Henrici [9].

A number of functions $f(x)$ fit into this framework, e.g. problems involving the circular functions, the Bessel functions, the step functions, $f(x) = \sin(x + 1/x)g(x)$, etc. However, the sawtooth function does not satisfy (3).

Define the Lyness [16] partition

$$x_0 = a, \quad x_l = b + (l-1)q, \quad l = 1, 2, \ldots,$$

and the integrals

$$u_l = \int_{x_l}^{x_{l+1}} f(x)\,dx, \quad l = 0, 1, 2, \ldots.$$

With this partition the problem (1) has been changed to a summation problem

$$S = \sum_{l=0}^{\infty} u_l,$$

and creating partial sums

$$S_n = \sum_{l=0}^{n} u_l \tag{5}$$

we may look for acceleration techniques for such sums. Each $u_l$ has of course to be approximated, but let us postpone the question of how this may be done and concentrate on the series approach first.

We get, by using the assumptions about $p_i(x)$ and $g_i(x)$, that

$$u_n = (-1)^n\{c_0/(c+n)^{\gamma} + c_1/(c+n)^{\gamma+1} + c_2/(c+n)^{\gamma+2} + \cdots\}, \quad n > 0, \tag{6}$$

where (we give only $c$, $c_0$ and $c_1$)

$$\begin{cases} c = b/q, \\ c_0 = -q^{-\gamma} \int_0^q \sum_{i=1}^k c_0^{(i)} p_i(b+t)\,dt, \\ c_1 = -q^{-(\gamma+1)} \int_0^q \sum_{i=1}^k \{c_1^{(i)} - \gamma(t-q)c_0^{(i)}\} p_i(b+t)\,dt. \end{cases}$$

In general, we may have $c_0 = c_1 = \cdots = c_{m-1} = 0$ and $c_m \neq 0$ in (6). Then the leading term in (6) will be $(-1)^n c_m/(c+n)^{\alpha}$, with $\alpha \equiv \gamma + m$. Observe that, at least for $n \gg 1$, we have an alternating series.

In [16] Lyness discusses and experiments with different starting values of $b$. This was inspired by an observation by Longman (attributed to J.C.P. Miller) indicating that the function's maximum is better than its zeros as partition points. We see from the expression of $c_0$ that this constant may be zero if a proper value of $b$ is chosen. Such a value of $b$ will therefore give a series converging at least one order faster than for other values of $b$.

If the periodic functions are non-smooth then this gain may be lost due to the numerical difficulties in estimating $u_n$. In such a case some other subdivision points may be far better.

The following definitions appear in [18] and [3, pp. 161–163], presented here in our notation:

**Definition 1**
A sequence $S$ is *P-regular* iff

$$\Delta S_n/\Delta S_{n-1} = z(1 + r_1/(c+n) + r_2/(c+n)^2 + \cdots),$$

where $z$ and $r_1$ are real numbers.

Throughout, the difference operator $\Delta$ works on the index $n$. Using (6) and the definition of $\alpha$ and $m$ we find

$$\Delta S_n/\Delta S_{n-1} = u_{n+1}/u_n = -(1 - \alpha/(c+n) + r_2/(c+n)^2 + \cdots),$$

and therefore the sequence (5) is $P$-regular with $z = -1$ and $r_1 = -\alpha$.

**Definition 2**
A transformation $S \to T$ is *of P-order p* iff

$$\Delta T_n/\Delta S_n = a_p/(c+n)^p + a_{p+1}/(c+n)^{p+1} + \cdots, a_p \neq 0, \quad p \geq 0.$$

Thus, applying a transformation $T$ of $P$-order $p$ to our $P$-regular series $S$ gives

$$\Delta T_n/\Delta T_{n-1} = -(1 - (\alpha+p)/(c+n) + r_2(T)/(c+n)^2 + \cdots).$$

This means that the sequence $T$ is again $P$-regular with $z = -1$ and $r_1(T) = r_1(S) - p$. This implies that $\alpha(T) = \alpha(S) + p$.

The Euler transformation is one well known method which is useful for such logarithmic alternating series (6). One step of this method is given by the simple averaging procedure

$$T_n = S_n - \mu\Delta S_{n-1}, \quad \text{with } \mu = 1/2. \tag{7}$$

Using (6), (7) and the definition of $m$ and $\alpha$ we find

$$\Delta T_n/\Delta S_n = -\alpha/(2c+2n) + O((c+n)^{-2}). \tag{8}$$

Thus the Euler transformation is of $P$-order 1 and we have $\alpha(T) = \alpha(S) + 1$.

Defining $T_{n0} = S_n$ this procedure may be repeated, building a triangular scheme by

$$T_{n,j} = T_{n,j-1} - \mu(T_{n,j-1} - T_{n-1,j-1}), \tag{9}$$

where in the $n$th row $j$ runs from 1 to $n$ and in the $j$th column $n$ runs from $j$ upwards. By repeated use of (8) we find that for the Euler transformation

$$\Delta T_{nj}^{(E)}/\Delta T_{n0}^{(E)} = (-1)^j\alpha(\alpha+1)\cdots(\alpha+j-1)/(2c+2n)^j + O((c+n)^{-(j+1)}). \tag{10}$$

*Remark*

The constant associated with the order $O$ will depend on $j$. Note that with $\alpha = 1$ we have $j!$ as a factor in the leading term in (10). The following minor modification of the Euler transformation is designed to reduce this factor. Replacing $\mu$ in (9) by

$$\mu_{n,j} = \mu\left(1 - \frac{j-1}{2(c+n)}\right) \tag{11}$$

gives the Modified Euler, using (6) as before,

$$\Delta T_{nj}^{(M)}/\Delta T_{n0}^{(M)} = (-1)^j \alpha^j/(2c+2n)^j + O((c+n)^{-(j+1)}). \tag{12}$$

Thus we see that this modification, too, is of $P$-order 1, but the constant in front of the leading term is much smaller, especially for small values of $\alpha$. For example, $\alpha = 1$ gives an improved factor $1/j!$.

Overholt's [3,18] $P$-order 2 modification of the Euler transformation achieves the vanishing of the leading constant in (8) by modifying $\mu$ in (9) as follows

$$\mu_{n,j} = \mu\left(1 - \frac{\alpha + 2(j-1)}{2(c+n)}\right). \tag{13}$$

This gives

$$\Delta T_{nj}^{(O)}/\Delta T_{n0}^{(O)} = O((c+n)^{-2j}).$$

Actually, Overholt shows [3,18] that there exist infinitely many 2-point formulas of $P$-order 2 if we know $r_1$, and if we do not know $r_1$ then there do not exist 2-point methods of $P$-order 2. We may replace the constant $c = b/q$ by a different constant and the method will still be of $P$-order 2. We will take advantage of this freedom in the implemented code.

If $\alpha$ is available then (13) is superior to the other two methods. Now $\alpha = \gamma + m$ and knowing $\gamma$ seems to be reasonable, but even if one does not know $\gamma$ it is possible to estimate $\gamma$ prior to using Overholt's $P$-order 2 method. Finding $m$ seems, on the other hand, much more difficult in general.

Let us simply replace $\alpha$ in (13) by $\gamma$. Then we know (a) that we may have an underestimate of $\alpha$, and (b) that the difference is an integer, possibly zero. If $\alpha > \gamma$ then Overholt's method, with $\gamma$, is of $P$-order 1 only. This will give a practical implementation of Overholt's method

$$\Delta T_{nj}^{(PO)}/\Delta T_{n0}^{(PO)} = (-1)^j m(m-1)\cdots(m-j+1)/(2c+2n)^j + O((c+n)^{-(j+1)}, \quad j = 1, 2, \ldots, m.$$

When $j = m+1$ then the leading term in this expression becomes zero and the method becomes of $P$-order 2. The point being that we have underestimated $\alpha$ (by an integer difference) and the method will recover its $P$-order 2 after a certain number of steps depending on $m$. The conclusion is that the practical version of Overholt's method, replacing $\alpha$ by $\gamma$, will behave very much unaffected by small

changes in $b$. Using the method this way we are not able to take advantage in the case when the original series is converging faster, however we do not lose acceleration speed either.

Note that the two methods of $P$-order 1 do not need $\alpha$ and will be of order one independent of whether $c_0$ is equal to zero or not.

We see that these three schemes are all very simple and easy to implement. Overholt's method will depend on the quality of the given or estimated value of $\gamma$. In experiments performed in [18] the $P$-order 2 methods are at least as good as the Levin $u$-transform [11] on alternating logarithmic series. Levin's $u$-transform was on the other hand reported to be very good in general by Fessler et al. [6]. This fact, combined with the simplicity of these three methods, are the reasons for choosing these methods in this software project.

## 3. Numerical stability

As we have seen, these three methods fit into the same framework. Put $T_{n0} = S_n$ and define

$$T_{n,j} = T_{n,j-1} - \mu_{n,j}(T_{n,j-1} - T_{n-1,j-1}), \quad j = 1, 2, \ldots, n. \tag{14}$$

Choosing $\mu_{n,j}$ appropriately we get the E(uler), the M(odified), the O(verholt) and the P(ractical) O(verholt) schemes. We may put the $T_{nj}$ in a standard tableau

$$\begin{array}{llll} T_{00} & & & \\ T_{10} & T_{11} & & \\ \vdots & \vdots & \ddots & \\ T_{n0} & T_{n1} & \cdots & T_{nn} \end{array}$$

If $0 \le \mu_{nj} \le 1$ in (14) then each entry in the tableau is a convex combination of the two entries to the left. This implies that

$$T_{nn} = \sum_{j=0}^{n} v_{nj} T_{j0},$$

where $0 \le v_{nj}$ and $\sum_{j=0}^{n} v_{nj} = 1$. Now, since $T_{j0} = S_j = \sum_{i=0}^{j} u_i$, we get

$$T_{nn} = \sum_{i=0}^{n} w_{ni} u_i,$$

with $w_{ni} = \sum_{j=i}^{n} v_{nj}$ and therefore

$$\kappa_n = \sum_{j=0}^{n} w_{nj} \le 1 + n.$$

The Euler method has $\kappa_n^{(\mathrm{E})} = 1 + n/2$. Approximating each $u_j$ by say, $\hat{u}_j$, implies that the approximated value $\hat{T}_{nn}$ has the following quality (neglecting

rounding errors)

$$|\hat{T}_{nn} - T_{nn}| \leq \kappa_n \max_{j=0,1,\ldots,n} |\hat{u}_j - u_j|.$$

This implies a very stable scheme where the accelerated approximation $\hat{T}_{nn}$ may be less influenced by errors in $\hat{u}_i$ than the partial sum $\hat{S}_n = \sum_{i=0}^{n} \hat{u}_i$ which it is based on. We consider this stability feature as very important and have chosen to use the freedom in the parameter $c$ such that we achieve $0 < \mu_{nj} < 1$ for all three methods.

## 4. The basic algorithm

We face two different alternatives if we are not satisfied with the global approximation $\hat{T}_{nn}$:

(A) Compute a better value for say, $\hat{u}_j$, and update the $T$-tableau. This can easily be done directly in the last row of this tableau. Estimate the new global error and decide what to do next.

(B) If we are not satisfied with the extrapolation error then: put $x_{n+1} = x_n + q$, compute $\hat{u}_{n+1}$, increase $n$ and create a new row in the $T$-tableau. Estimate the new global error and finally decide what to do next.

An algorithm combining adaptivity and extrapolation for this problem has as input: The integration limit $a$, the first subdivision point $b$, the period $2q$, the strength of the decay of the functions at infinity (optional), $\gamma$, the function(s) $f$ and finally an error tolerance $\epsilon$. The output will normally be an estimate of the integral, $\hat{Q}$, and an error estimate $\hat{E}$. If possible the algorithm computes $\hat{Q}$ with $\hat{E} \leq \epsilon$ (here $\epsilon = \max(\epsilon_a, \epsilon_r|\hat{Q}|)$, where $\epsilon_a$ and $\epsilon_r$ are user specified absolute and relative error tolerances respectively). This does not ensure that $|I - \hat{Q}| \leq \epsilon$, but it does suggest that it is likely. Basic elements in this globally adaptive quadrature algorithm are:

- A collection of intervals organized in a heap. At a certain stage we have $M$ intervals in the heap, say $i_j, j = 1, \ldots, M$.
- A quadrature rule, $Q$, to produce a local estimate, $\hat{Q}_j$, to the integral over each interval $i_j$ in the collection. We offer one basic rule, a Gauss rule using 21 nodes, of polynomial degree 41.
- A procedure to estimate the error in the local approximations over the regions in the heap produced by $Q$. This error estimate, $\hat{E}_j$, is constructed using several null rules, see [1,2,5,4]. The pure extrapolation error enters the global error only and does not affect the heap.
- A strategy for picking the next interval to be processed: In this algorithm this will be the interval with the greatest error estimate. Therefore the heap is organized such that the interval with the greatest error estimate is the top element. The extrapolation error is considered too in order to decide what to

do. If we integrate a vector integrand, then the top element is the interval with the greatest worst case error estimate.

- A strategy for how to divide such a region: We have chosen to follow [7,22] and divide an interval into three equally sized parts.
- An extrapolation procedure: we have chosen to offer the three schemes described: Euler, the modified and the practical Overholt scheme. The user is given the option of having $\gamma$ estimated for him before the extrapolation starts.
- A global error estimate. Note that it is important to know for each region in the heap which $u_l$ it belongs to. We compute the $w_{nl}$-coefficients and thus the global error may be computed by $\sum_{l=0}^{n} w_{nl} E_{\hat{u}_l}$ and adding the extrapolation error.

We give here the basic structure of the algorithm

**A globally adaptive quadrature algorithm with extrapolation**

Initialize: Put $x_0 = a$, $x_1 = b$ and $x_2 = b + q$;
Compute $\hat{u}_0$, $\hat{u}_1$, $\hat{E}_{\hat{u}_0}$ and $\hat{E}_{\hat{u}_1}$.
Initialize the heap: $M = 2$;
The tableau: $\hat{T}_{00} = \hat{u}_0$, $\hat{T}_{10} = \hat{u}_0 + \hat{u}_1$ and $n = 1$;
Extrapolate and estimate the pure extrapolation error;
Compute the global error $\hat{E}_{\hat{T}_{11}}$;
Control: **while** $\hat{E}_{\hat{T}_{nn}} > \epsilon$ **do**
**begin**
**if** The pure extrapolation error > The top heap error **then**
Add a new interval to the collection: $x_{n+2} = x_{n+1} + q$;
Process regions: Compute: $\hat{u}_{n+1}$ and $\hat{E}_{\hat{u}_{n+1}}$,
$\hat{T}_{n+1,0} = \hat{T}_{n,0} + \hat{u}_{n+1}$ and put $n = n + 1$;
Extrapolate and estimate the pure extrapolation error;
Compute the global error $\hat{E}_{\hat{T}_{nn}}$;
Update: Update the heap and put $M = M + 1$.
**else**
Pick the top interval from the heap; say $i_t$;
Subdivide this interval into three new subintervals;
Process regions: Compute for the three subintervals: $\hat{Q}_t^{(l)}$ and $\hat{E}_t^{(l)}$, $l = 1, 2, 3$;
Update: Update the heap, $M = M + 2$, $\hat{u}_j$ and $\hat{E}_{\hat{u}_j}$ associated with $i_t$;
Update the tableau and the pure extrapolation error;
Compute the global error $\hat{E}_{T_{nn}}$
**end if**
**end**

*Remark*
In the final implementation of this algorithm the code will always deliver the best possible result, that is: the result with the smallest possible estimated error.

Note that the value of $b$ has to be provided by the user: the theory only implies that $b$ has to be sufficiently great such that the asymptotic properties are satisfied for $x \geq b$. If we choose $b$ much bigger than necessary then the cost of evaluating $u_0$ may be high due to adaptive subdivision of the interval $[a, b]$. The improved asymptotic behavior due to a large value of $b$ may on the other hand reduce the number of extrapolation steps necessary. Increasing $b$ as much as possible without increasing the work spent on $u_0$ is probably a reasonable advice. Some insight may be gained by experimenting with different values of $b$ for a particular class of problems.

The optimal choice of $b$ will, in addition, depend on the requested accuracy. Finally: the three implemented methods will have different optimal values of $b$, the practical Overholt's method being the least sensitive of the three.

## 5. Numerical examples

In this section we will demonstrate how efficient the new code, DQAINF, can be. All computations have been done in double precision arithmetic on a SUN Sparc 10 workstation.

**Example 1**

The first example involves the step function

$$\int_1^\infty p(x)/\sqrt{x}\,dx \approx 0.47958\,07495\,61\ldots, \tag{15}$$

where $p(x) = 1$ for $2k-1 \leq x < 2k$ and $p(x) = -1$ for $2k \leq x < 2k+1$ for $k = 1, 2, \ldots$. Thus we have $q = 1$ and $\gamma = 1/2$ with (3) and (4) satisfied. The periodic function is nonsmooth and some care is necessary when choosing the partition points. Good values for $b$ are $2, 4, 10, \ldots$. When $b = 2$ then we have a partition point at every jump discontinuity, while $b = 4$ gives jump discontinuities inside the first interval at 2 and 3. However, the code will do trisection and immediately divide this region into three parts and the inside jump problem is removed. Thus there is an extra cost using $b = 4$ in this case and we have therefore kept $b = 2$ for all three methods. Noninteger values of $b$ is a bad choice for the same reason: then every interval will have an inside jump discontinuity.

We see that the code performs quite well on this problem for all three methods. The 21 point Gauss rule has the quality that it computes every estimate $\hat{u}_j$ of $u_j$ to machine precision and therefore each new step in figures 1a and 1b is due to a new extrapolation step. We see that the error estimate, designed to be cautious, seems reliable and is overestimating by 1–2 digits in figure 1a. A $P$-order 2 method can be expected to be twice as fast as a $P$-order 1 method in general. We see that there is a factor between 2.5 and 3 between Overholt's method and Euler's method, while the ratio between Overholt's method and the modified Euler

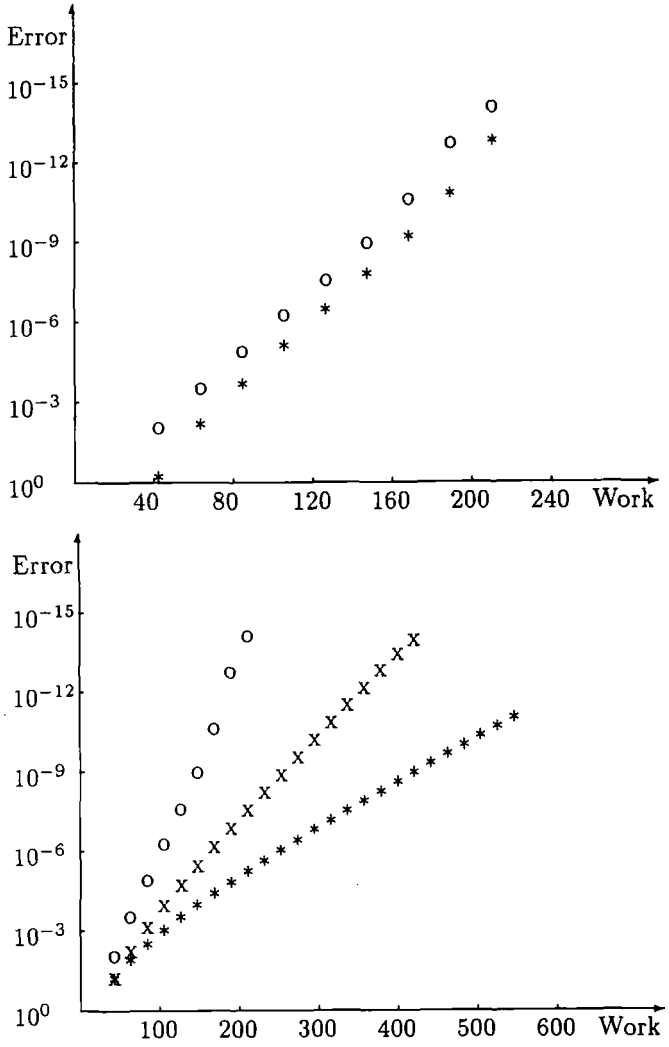


Figure 1. (a) The true relative errors (○) and the estimated relative errors (∗) versus the number of function values used to produce these estimates using DQAINF with KEY = 2 (*P*-order 2) on (15). (b) The true relative errors: using Overholt's *P*-order 2 method (○), the modified Euler (×) and Euler's method (∗) versus the number of function values used to produce these estimates by DQAINF on (15).

method is between 1.5 and 2. In both cases we assume that the work is proportional to the number of function evaluations.

**Example 2**

The next example has a smooth periodic function

$$\int_0^\infty \sin(x)/\sqrt{1+x}\,dx \approx 0.80952\,54817\,47\ldots\,. \tag{16}$$

The choice of different *b*-values for the three methods is done to achieve the

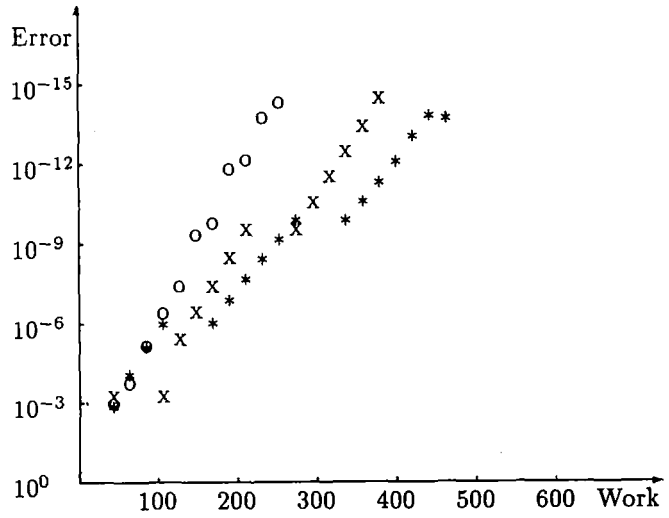


Figure 2. The true relative errors: using Overholt's $P$-order 2 method (○, $b = 3$), the modified Euler ($\times$, $b = 30$) and Euler's method ($*$, $b = 27$) versus the number of function values used to produce these estimates by DQAINF on (16).

precision $10^{-13}$ for all three methods with minimum effort. If the user specifies a tolerance then a different value may be better for each of the three methods.

Due to the large values of $b$ for the two $P$-order 1 methods the adaptive strategy appears in these cases. The modified Euler method needs immediately to subdivide the first interval with the cost of 63 function evaluations, then after 5 extrapolation steps one of these three intervals is subdivided further. The Euler method has a similar behavior. We see that the difference in $b$ effects the performance of the methods.

**Example 3**

In the following example we have $k = 2$ in (2) involving $\sin(x)$ and $\cos(x)$. Furthermore we do not specify $\gamma$ and leave that to the code

$$\int_1^\infty \sin(x + 1/x)/\sqrt{x}\, dx \approx 0.2329481970\,94\ldots . \tag{17}$$

The code estimates $\gamma$ to 10 correct digits using 21 function evaluations. The code uses linear extrapolation on a sequence of ratios $\ln(-f(y_i + q)/f(y_i))/\ln(y_i/(y_i + q))$, where $y_0$ is the starting point chosen after 4 samples in $[b, b + q)$, $y_i = y_0 + 2^i q$, $i = 1, 2, \ldots$. The stopping criterion is designed to avoid that the estimate of $\gamma$ is contaminated by rounding errors. The effect of not knowing $\gamma$ is simply a shift of the plot to the right due to the 21 function values (eight linear extrapolation steps) it took to estimate $\gamma$ in this case.

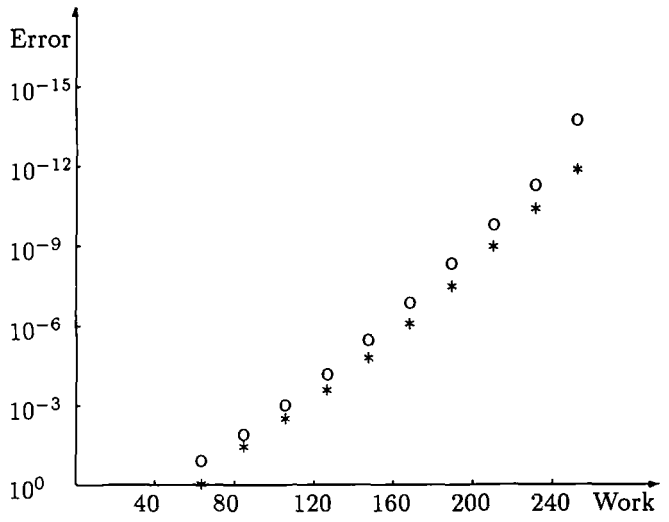


Figure 3. The true relative errors (○) and the estimated relative errors (∗) versus the number of function values used to produce these estimates using DQAINF with KEY = 2 (*P*-order 2), $b = 4$ and unspecified $\gamma$ on (17).

**Example 4**

Next we integrate a function with period $2q = 22\pi$.

$$\int_0^{\infty} \frac{\cos(x) - \cos(7x/11)}{x}\, dx = \ln 7/11. \tag{18}$$

This example illustrates the interplay between the adaptive strategy and the extrapolation. The extrapolation sequence is in this example: 1, 1, 1, 1*, 2, 2*, 3, 3*, 4, 4*, 5. A * indicates an attempt to take another extrapolation step, but since the error estimate was not improved the code decided not to update the result. The half

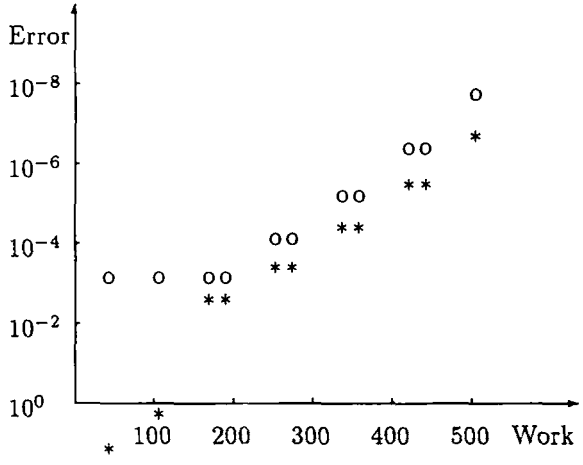


Figure 4. The true relative errors (○) and estimated errors (∗) for Overholt's *P*-order 2 method versus the number of function values used to produce these estimates by DQAINF on (18).

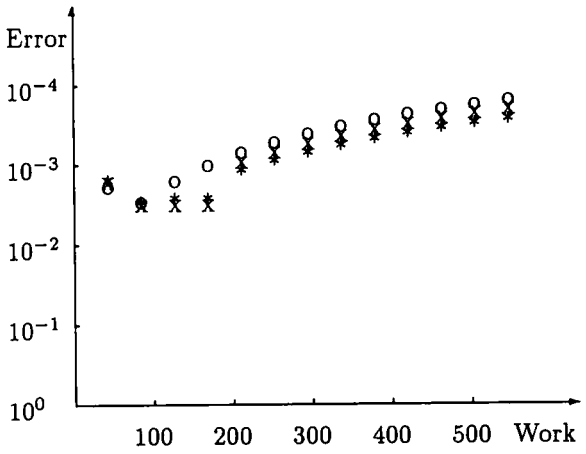


Figure 5. The true relative errors using: Overholt's method (○), the modified Euler method (×) and the Euler method (*) versus the number of function values used to produce these estimates by DQAINF on (19). $b = 6.28$ for all three methods.

period $q = 11\pi$ is so big that $\cos(x)$ changes sign repeatedly, and each new interval has to be subdivided to give an estimate of good enough quality.

**Example 5**

The final two examples are included to illustrate what may happen if one of the two assumptions (3,4) is violated. The first example violates the antisymmetry property (3).

$$\int_0^\infty \frac{\cos(x) - \cos(2x)}{x}\, dx = \ln 2. \tag{19}$$

The periodic function is indeed oscillating since integrating over a full period then

$$\int_b^{b+2\pi} (\cos(x) - \cos(2x))\, dx = 0, \quad \text{for all } b.$$

Figure 5 demonstrates that the extrapolation has hardly any effect. The reason is that even though the initial sequence is oscillating this fundamental property is lost after one extrapolation step with all three methods.

The problem may of course be transformed by a simple splitting into two finite integrals

$$\int_0^b \frac{\cos(x) - \cos(2x)}{x}\, dx + \int_b^{2b} \frac{\cos(x)}{x}, \quad b > 0.$$

**Example 6**

In the final example (4) is no longer valid

$$\int_1^\infty \sin(x + 1/\sqrt{x})/\sqrt{x}\, dx \approx 0.04163\,28516\,893\ldots\,. \tag{20}$$

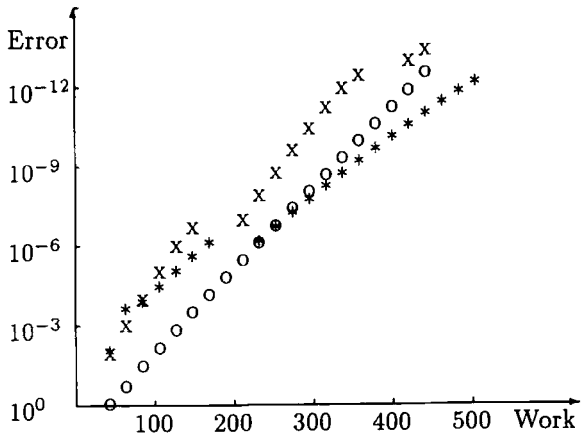


Figure 6. The true relative errors: using Overholt's $P$-order 2 method ($\bigcirc$, $b = 3$), the modified Euler ($\times$, $b = 20$) and Euler's method ($*$, $b = 20$) versus the number of function values used to produce these estimates by DQAINF on (20).

Instead of the exponent sequence $\gamma + j$ in (4) we get for this problem $\gamma + j/2$, for $j = 0, 1, 2, \ldots$ with $\gamma = 1/2$.

As we see from figure 6 all three methods have a similar behavior and a closer analysis shows that all three methods will have $P$-order 1 and the oscillating property is maintained through the different transformations. Thus we lose the $P$-order 2 effect when (4) is not satisfied.

## 6. Concluding remarks

The main goal in the development of the code, DQAINF, was to develop a routine that could handle infinite oscillating tails effectively and at the same time have good reliability properties. We feel that the power of this new code is clearly demonstrated through these examples and thus we claim that the main goal has been achieved.

The new algorithm may be applied to a vector of similar integrands over a common integration interval. All of the integrands in the vector have to have oscillating tails with the same period $2q$. The $P$-order 2 method is useful on a vector integrand only when all functions have the same decay parameter $\gamma$ (or at least an integer difference between the possibly different $\gamma$ values). In general integrating a vector simultaneously should be used with caution, because the algorithm chooses a subdivision of the integration region that may be refined according to the behavior of particular integrands which have certain local intervals of difficulty. For integrands that have enough similarity, the new algorithm may save time and space because of the common subdivisions.

## 7. DQAINF: A FORTRAN implementation

The algorithm described in section 4, including the error-estimating procedure from [2,4], has been implemented in a FORTRAN subroutine DQAINF, a double precision adaptive quadrature routine with extrapolation. The single precision version has the name SQAINF and similarly the first letter in all subroutine-names is changed to S. DQAINF has the calling sequence

```
CALL DQAINF (NUMFUN,FUNSUB,PERIOD,GAMMA,A,B,WRKSUB,
             EMAX,MINPTS,MAXPTS,EPSABS,EPSREL,NW,KEY,
             RESTAR,RESULT,ABSERR,NEVAL,IFAIL,DONE,
             WORK,IWORK)
```

DQAINF computes approximations RESULT to the vector integral

$$\mathbf{I} = \int_A^\infty \mathbf{f}(x)\,dx,$$

where $\mathbf{f}(x)$ is a vector of integrands of length NUMFUN all given by one call to the user specified subroutine FUNSUB. The lower integration limit is given by A. PERIOD gives the period of all components in the vector and GAMMA (optional) gives the decay of all functions towards infinity. B is the point from where the asymptotic property is assumed to hold. DQAINF attempts to compute each component of RESULT such that

$$|\mathbf{I}[K]\text{-RESULT}[K]| \leq \max(\text{EPSABS}, \text{EPSREL}^*|\mathbf{I}[K]|), \quad K = 1, \ldots, \text{NUMFUN},$$

where EPSABS and EPSREL are absolute and relative tolerances. The program has been successfully run on a Sun SPARC 10 workstation.

Having a vector integrand implies a change in the algorithmic structure since we have NUMFUN error estimates over each subinterval. Picking the next interval to subdivide and process is based on the worst case error estimate. Next we either update the extrapolation tableau or extend the extrapolation tableau depending on whether an old interval was subdivided or a new interval added. We continue this process with all integrands until the global error test is satisfied for all integrands or the number of evaluation points exceeds MAXPTS.

The integration routine and the subprograms are organized according to the structure given in figure 7.

In the following list we briefly describe the purpose of DQAINF and supporting subroutines.

- DQAINF is basically a driver for the algorithm controller DADINF. Inside DCHINF the input to DQAINF is checked and the workspace for DADINF is separated into mnemonic variables.
- DCHINF checks the validity of the input to DQAINF. If GAMMA $\leq 0$ and KEY $= 2$ ($P$-order 2) then the routine DGAINF is called in order to estimate GAMMA.
- DGAINF is a routine designed to estimate GAMMA.

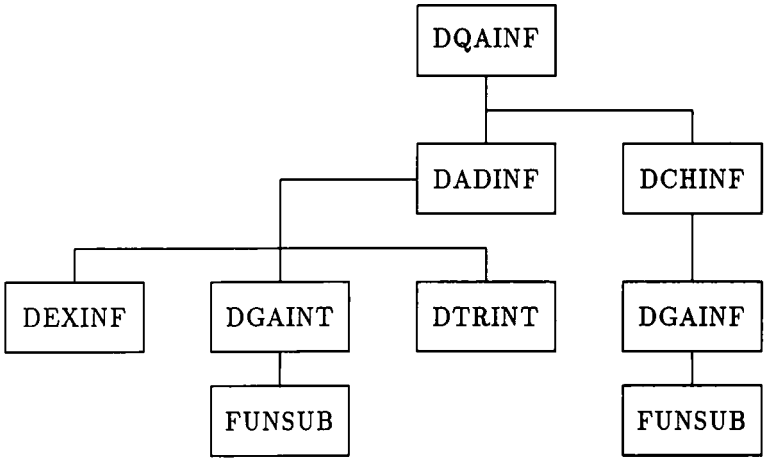


Figure 7.

- DADINF is the algorithm controller which at each subdivision step decides whether to stop or to continue. This routine also decides whether a new extrapolation step will be taken or just an update of the extrapolation tableau will be done.
- DEXINF does either the extrapolation or the update of the extrapolation tableau. The pure extrapolation error is estimated and the global error is then computed. The type of transformation is decided through the parameter KEY: 0 (Euler), 1 (modified Euler) and 2 (*P*-order 2).
- DTRINT maintains the heap of subregions.
- DGAINT computes approximations to the integral and the error over each finite interval. In this code a 21 point Gaussian rule is offered, but the user may replace this code with his favorite rule and associated error estimation.

## Appendix: The parameters for DQAINF

*Input parameters*

NUMFUN The number of components in the vector integrand function.

FUNSUB An externally declared subroutine for computing all components in the vector function for the given evaluation points. It must have parameters (X, NUMFUN, NP, FUNVLS). The input parameters: X is an array of dimension NP giving the $x$-coordinates of the NP evaluation points. NUMFUN is the number of components of the vector integrand. The output parameter FUNVLS is an array of dimension (NP, NUMFUN) that stores NUMFUN values of the vector integrand for each of the NP evaluation points.

PERIOD All functions are assumed to have the same asymptotic period.

| | |
|---|---|
| GAMMA | All functions are assumed to decay as c/x**GAMMA, when x ≫ B and we go in steps of length PERIOD. GAMMA > 0. |
| A | The lower limit of integration is A(1). |
| B | B(1) is the right endpoint of the first interval. B(1) is chosen by the user and it is assumed that the integrand oscillates for x > B(1). The oscillations are expected to be observed for points of distance PERIOD/2. On exit A and B contain the endpoints of the intervals used to produce the approximations to the integrals. The original A(1) and B(1) may have changed. |
| WRKSUB | Defines the length of the arrays A and B. (And thus the maximum number of subregions allowed.) |
| EMAX | The maximum number of extrapolation steps that will be allowed. |
| MINPTS | The minimum number of FUNSUB calls the code has to use. |
| MAXPTS | The maximum number of FUNSUB calls the code is allowed to use. See the code for details. |
| EPSABS | Requested absolute error. |
| EPSREL | Requested relative error. |
| NW | Defines the length of the working array WORK. See the code for details. |
| KEY | The choice of extrapolation method:<br>KEY = 0: The Euler method.<br>KEY = 1: The modified Euler method.<br>KEY = 2: Overholt's *P*-order 2 method.<br>This last method is the only one that needs the value of GAMMA. |
| RESTAR | If RESTAR = 0, this is the first attempt to compute the integral.<br>If RESTAR = 1, then we continue a previous attempt. In this case the only parameters for DQAINF that may be changed (with respect to the previous call of DQAINF) are MINPTS, MAXPTS, EPSABS, EPSREL and RESTAR. |

*Output parameters*

| | |
|---|---|
| RESULT | An array of dimension NUMFUN containing approximations to all components of the vector integral. |
| ABSERR | An array of dimension NUMFUN of estimates for absolute errors for all components of the vector integral. |
| NEVAL | The number of function evaluations used by DQAINF. |
| IFAIL | IFAIL = 0 for normal exit, with<br>ABSERR(K) ≤ max (EPSABS, ABS(RESULT(K))*EPSREL)<br>for 1 ≤ K ≤ NUMFUN, using MAXPTS or less FUNSUB calls.<br>IFAIL = 1 if MAXPTS is too small for DQAINF to obtain the requested error. In this case DQAINF returns values of RESULT with estimated absolute errors ABSERR.<br>IFAIL = 2 if NUMFUN < 1. |

IFAIL = 3 if A(1) > B(1).
IFAIL = 4 if unlegal PERIOD.
IFAIL = 5 if MAXPTS < MINPTS or MINPTS < 21.
IFAIL = 6 if both EPSABS $\leq$ 0 and EPSREL $\leq$ 0.
IFAIL = 7 if WRKSUB is too small.
IFAIL = 8 if unlegal value of EMAX.
IFAIL = 9 if unlegal RESTAR.
IFAIL = 10 if KEY < 0 or KEY > 2.
IFAIL = 11 if NW is too small. (See the code DCHINF).
IFAIL = 12 if either PERIOD is wrong or B(1) is too small.
IFAIL = 13 if unable to estimate GAMMA (In case KEY = 2 only).
IFAIL = 14 if EMAX turns out to be too small.

DONE — DONE(I) = .TRUE. if function number I has been integrated to the specified precision, else DONE(I) = .FALSE. (Note that IFAIL = 1 just tells you that something is wrong, however most of the integrals in the vector may have been computed to the specified precision).

WORK — A work array of dimension NW. See the code for details.

IWORK — A work array of dimension 3*WRKSUB. See the code for details.

**Software available via netlib, library NUMERALGO**

## References

[1] J. Berntsen, Practical error estimation in adaptive multidimensional quadrature routines, J. Comp. Appl. Math. 25 (1989) 327–340.
[2] J. Berntsen and T.O. Espelid, Error estimation in automatic quadrature routines, ACM Trans. Math. Softw. 17 (1991) 233–252.
[3] C. Brezinski and M. Redivo Zaglia, *Extrapolation Methods, Theory and Practice*, vol. 2 of *Studies in Computational Mathematics* (North-Holland, 1991).
[4] T.O. Espelid, DQAINT: An algorithm for adaptive quadrature over a collection of finite intervals, in: *Numerical Integration, Recent Developments, Software and Applications*, NATO ASI Series C: Mathematical and Physical Sciences, Vol. 357 (Kluwer Academic, Dordrecht, The Netherlands, 1992) pp. 341–342.
[5] T.O. Espelid, Integration rules, null rules and error estimation, Reports in Informatics 33, Dept. of Informatics, Univ. of Bergen (1988).
[6] T. Fessler, W.F. Ford and D.A. Smith, Hurry: An acceleration algorithm for scalar sequences and series, ACM Trans. Math. Softw. 9 (1983) 346–354.
[7] W. Hanke, Die optimaler Sektion bei adaptiven Integrationsverfahren mit globaler Strategie, ZAMM 62 (1982) 327–329.
[8] T. Hasegawa and T. Torii, Indefinite integration of oscillatory functions by the Chebyshev series expansion, J. Comp. Appl. Math. 1 & 2 (1987) 21–29.
[9] P. Henrici, *Applied and Computational Complex Analysis* (Wiley, 1974).
[10] K. Knopp, *Theorie und Anwendungen der unendlichen Reien* (Springer, 1964).
[11] D. Levin, Development of non-linear transformations for improving convergence of sequences, J. Inst. Math. Appl. B3 (1973) 371–388.
[12] D. Levin and A. Sidi, Two classes of non-linear transformations for accelerating the convergence of infinite integrals and series, Appl. Math. Comp. 9 (1981) 175–215.

[13] I.M. Longman, Note on a method for computing infinite integrals of oscillatory functions, Proc. Cambridge Phil. Soc. 52 (1956) 764–768.
[14] I.M. Longman, Tables for the rapid and accurate numerical evaluation of certain infinite integrals involving Bessel functions, MTAC 11 (1957) 166–180.
[15] I.M. Longman, A method for the numerical evaluation of finite integrals of oscillatory functions, Math. Comp. 14 (1960) 53–59.
[16] J.N. Lyness, Integrating some infinite oscillating tails, J. Comp. Appl. Math. 12 & 13 (1985) 109–117.
[17] J.N. Lyness and G. Hines, To integrate some infinite oscillating tails, ACM Trans. Math. Softw. 12 (1986) 24–25.
[18] K.J. Overholt, The P-algorithms for extrapolation, Report 36, Dept. of Informatics, University of Bergen (1989).
[19] R. Piessens, E. de Doncker-Kapenga, C.W. Überhuber and D.K. Kahaner, *QUADPACK, A Subroutine Package for Automatic Integration*, Series in Computational Math. vol. 1 (Springer, 1983).
[20] A. Sidi, Extrapolation methods for oscillatory infinite integrals, *J. Inst. Math. Appl.* 26 (1980) 1–20.
[21] A. Sidi, The numerical evaluation of very oscillatory infinite integrals by extrapolation, Math. Comp. 38 (1982) 517–529.
[22] T. Sørevik, Reliable and efficient algorithms for adaptive quadrature, Technical report, Thesis for the degree Doctor Scientiarum, Department of Informatics, University of Bergen (1988).