Quadpack メモ

桂田 祐史

2025 年 5 月 7 日, 2025 年 5 月 11 日

目 次

1 はじめに

Quadpack というのは、古典的な数値積分ライブラリィである。ChatGPT は数値積分が必要になると (Python 上の) Quadpack の利用を勧めてくる。これまで (知ってはいたけれど) 利用するのを敬遠してきたけれど、今後は時々使うことになりそうなので、この際少し調べておくことにした。

有名なので、Wikipedia (https://en.wikipedia.org/wiki/QUADPACK) でも紹介されている。

Quadpack の開発者達による文献 Piessens-Kapenga-Überhuber-Kahaner [1] によると、

> QUADPACK is a FORTRAN 77 library for numerical integration of one-dimensional functions.
>
> これは、R. Piessens と E. de Doncker (Appl. Math. and Progr. Div.-K.U.Leuven、ベルギー), C. Ueberhuber (Inst. Fuer Math.-Techn.U.Wien、オーストリア), D. Kahaner (Nation. Bur. of Standards-ワシントン D.C.、アメリカ) の共同プロジェクトから生まれた。

とのこと。

Quadpack はパブリックドメインであり、NETLIB で配布されている。

Netlib にある配布物には、QPDOC[1] という説明文書がある。

> ドキュメンテーションルーチン QPDOC は、1981 年 5 月に A.M.P.D.-Leuven からリリースされた SLATEC ライブラリに準拠したパッケージについて記述している。

とのこと。

というわけで、SLATEC にも含まれている。

GSL (GNU Scientific Library) にも新たに C で実装されたものが含まれている。

絶対誤差または相対誤差の許容誤差を入力すると、ルーチンは要求された誤差よりも大きな誤差がないように積分を実行しようとします。QUADPACK には、多数の非自動ルーチンに加えて、このような自動ルーチンが 9 つあります。自動ルーチンの 1 つを除いて、すべて適応求積法を使用しています [7]。

2 サブルーチンの名前

2.1 自動ルーチンの概要

QNG Is a simple non-adaptive automatic integrator, based on a sequence of rules with increasing degree of algebraic precision (Patterson, 1968).

QAG Is a simple globally adaptive integrator using the strategy of Aind (Piessens, 1973). It is possible to choose between 6 pairs of Gauss-Kronrod quadrature formulae for the rule evaluation component. The pairs of high degree of precision are suitable for handling integration difficulties due to a strongly oscillating integrand.

[1] https://www.netlib.org/quadpack/doc

QAGS Is an integrator based on globally adaptive interval subdivision in connection with extrapolation (de Doncker, 1978) by the Epsilon algorithm (Wynn, 1956).

QAGP Serves the same purposes as QAGS, but also allows for eventual user-supplied information, i.e. the abscissae of internal singularities, discontinuities and other difficulties of the integrand function. The algorithm is a modification of that in QAGS.

QAGI Handles integration over infinite intervals. The infinite range is mapped onto a finite interval and then the same strategy as in QAGS is applied.

QAWO Is a routine for the integration of COS(OMEGA*X)*F(X) or SIN(OMEGA*X)*F(X) over a finite interval (A,B). OMEGA is is specified by the user. The rule evaluation component is based on the modified Clenshaw-Curtis technique. An adaptive subdivision scheme is used connected with an extrapolation procedure, which is a modification of that in QAGS and provides the possibility to deal even with singularities in F.

QAWF Calculates the Fourier cosine or Fourier sine transform of F(X), for user-supplied interval (A, INFINITY), OMEGA, and F. The procedure of QAWO is used on successive finite intervals, and convergence acceleration by means of the Epsilon algorithm (Wynn, 1956) is applied to the series of the integral contributions.

QAWS
```
Integrates W(X)*F(X) over (A,B) with A.LT.B finite,
and   W(X) = ((X-A)**ALFA)*((B-X)**BETA)*V(X)
where V(X) = 1 or LOG(X-A) or LOG(B-X)
               or LOG(X-A)*LOG(B-X)
and   ALFA.GT.(-1), BETA.GT.(-1).
The user specifies A, B, ALFA, BETA and the type of
the function V.
A globally adaptive subdivision strategy is applied,
with modified Clenshaw-Curtis integration on the
subintervals which contain A or B.
```

QAWC Computes the Cauchy Principal Value of F(X)/(X-C) over a finite interval (A,B) and for user-determined C. The strategy is globally adaptive, and modified Clenshaw-Curtis integration is used on the subranges which contain the point X = C.

Each of the routines above also has a "more detailed" version with a name ending in E, as QAGE. These provide more information and control than the easier versions.

命名規則については、Zwillinger [2] によると次のようなものであるとか。

1st letter	2nd letter	3rd letter	4th letter
Q Quadrature	**N** Non-adaptive	**G** General integrand **W** Weight function of specified form	– Simple integrator **S** – Singularities handled **P** – Specified points of local difficulty (singularities, discontinuities ...) **I** – Infinite interval
	A Adaptive	**G** General integrand **W** Weight function of specified form	**O** – Oscillatory weight function (cos or sin) over a finite interval **F** – Fourier transform (cos or sin) **C** – Cauchy principal value

3 サンプル・プログラムを試す

配布物に含まれるドキュメント QPDOC (`doc`) にサンプル・プログラムが含まれている。これを実行してみよう。

もしかすると、どこか (Piessens-Kapenga-Überhuber-Kahaner [1] とか — 未入手) に詳しい解説があるのかもしれないが、少なくとも `doc` には、どういう結果が得られれば、正常に動いているとみなせるかは書いてない。自分で判断しなさい、ということなのか？以下では、Mathematica の力を借りてチェックしてみる。

結果を表示するステートメントを入れるようにコメントがある (自分でやらないといけないの？？)。本来はもっと色々表示してみるべきと考えられるが、今回はとりあえず、単精度版に

```
write(*,*) ' RESULT=', RESULT
```

倍精度版に

```
write(*,'(A,F25.20)') ' RESULT=' ,RESULT
```

を挿入した。

3.1 QNG (成功)

QNG Is a simple non-adaptive automatic integrator, based on a sequence of rules with increasing degree of algebraic precision.

サンプル・プログラムでは次の定積分を計算していると考えられる。

$$\int_0^1 \frac{e^x}{x^2 + 0.1 \times 10^1}\, dx = \int_0^1 \frac{e^x}{x^2+1}\, dx = 1.27072413983362022013\cdots .$$

testqng.f

```
      REAL A,ABSERR,B,F,EPSABS,EPSREL,RESULT
      INTEGER IER,NEVAL
      EXTERNAL F
      A = 0.0E0
      B = 1.0E0
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      CALL QNG(F,A,B,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,IER)
C     C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C     C
      REAL FUNCTION F(X)
      REAL X
      F = EXP(X)/(X*X+0.1E+01)
      RETURN
      END
```

testdqng.f

```
C testdqng.f
      double precision A,ABSERR,B,F,EPSABS,EPSREL,RESULT
      INTEGER IER,NEVAL
      EXTERNAL F
      A = 0.0D0
      B = 1.0D0
      EPSABS = 0.0D0
      EPSREL = 1.0D-3
      CALL DQNG(F,A,B,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,IER)
C     C  INCLUDE WRITE STATEMENTS
      write(*,'(A,F20.15)') ' RESULT=',RESULT
      STOP
      END
C     C
      double precision FUNCTION F(X)
      double precision X
      F = DEXP(X)/(X*X+0.1D+01)
      RETURN
      END
```

testqng の実行結果

```
% gfortran -O -o testqng testqng.f libquadpack.a
% ./testqng
  RESULT=   1.27072430
% gfortran -O -o testdqng testdqng.f libquadpack.a
% ./testdqng
 RESULT=   1.270724139833620
```

($1.270724139833620 22013\cdots$ との差はそれぞれ 1.6×10^{-7}, 2.2×10^{-16} であるから) 単精度、倍精度とも十分な結果であろう。

3.2 QAG (成功)

QAG is a simple globally adaptive integrator using the strategy of Aind (Piessens, 1973). It is possible to choose between 6 pairs of Gauss-Kronrod quadrature formulae for the rule eval-

uation component. The pairs of high degree of precision are suitable for handling integration difficulties due to a strongly oscillating integrand.

QAG は、Aind (Piessens, 1973) の戦略を用いた単純なグローバル適応積分器である。ルール評価コンポーネントは 6 組のガウス・クロンドロッド積分公式から選ぶことができる。高精度の組は、強く振動する被積分関数による積分困難の処理に適している。

サンプル・プログラムでは次の定積分を計算していると考えられる。

$$\int_0^1 \frac{2}{2+\sin(10\pi x)}dx = \frac{2}{\sqrt{3}} = 1.1547005383792515290\cdots$$

testqag.f

```
      REAL A,ABSERR,B,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,IWORK,KEY,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      A = 0.0E0
      B = 1.0E0
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      KEY = 6
      LIMIT = 100
      LENW = LIMIT*4
      CALL QAG(F,A,B,EPSABS,EPSREL,KEY,RESULT,ABSERR,NEVAL,
     *  IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      F = 2.0E0/(2.0E0+SIN(31.41592653589793E0*X))
      RETURN
      END
```

testqag の実行結果

```
% gfortran -O -o testqag testqag.f libquadpack.a
% ./testqag
  RESULT=   1.15470052
% gfortran -O -o testdqag testdqag.f libquadpack.a
% ./testdqag
 RESULT=   1.1547005383925146212
```

($1.1547005383792515290\cdots$ との差はそれぞれ 1.8×10^{-8}, 6.7×10^{-17} であるから)
単精度、倍精度とも十分な結果であろう。

3.3 QAGS (成功)

QAGS is an integrator based on globally adaptive interval subdivision in connection with extrapolation (de Doncker, 1978) by the Epsilon algorithm (Wynn, 1956).

QAGS は，イプシロンアルゴリズム (Wynn, 1956) による外挿 (de Doncker, 1978) と大域適応的区間細分割に基づく積分器である．

サンプル・プログラムでは次の定積分を計算していると考えられる。

$$\int_0^1 \frac{1}{\sqrt{x}}\,dx = 2.$$

(被積分関数の $x = 0$ での値は 0 としてある。)

testqags.f

```
      REAL A,ABSERR,B,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      A = 0.0E0
      B = 1.0E0
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      LIMIT = 100
      LENW = LIMIT*4
      CALL QAGS(F,A,B,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,IER,
     *  LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      F = 0.0E0
      IF(X.GT.0.0E0) F = 1.0E0/SQRT(X)
      RETURN
      END
```

testdqags.f

```
C testdqags.f
      double precision A,ABSERR,B,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      A = 0.0D0
      B = 1.0D0
      EPSABS = 0.0D0
C      EPSREL = 1.0D-3
      EPSREL = 1.0D-12
      LIMIT = 100
      LENW = LIMIT*4
      CALL DQAGS(F,A,B,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,IER,
     *  LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,'(A,F25.20)') ' RESULT=',RESULT
      STOP
      END
C      C
      double precision FUNCTION F(X)
      double precision X
      F = 0.0D0
      IF(X.GT.0.0D0) F = 1.0D0/DSQRT(X)
      RETURN
      END
```

testqags の実行結果

```
% gfortran -O -o testqags testqags.f libquadpack.a
% ./testqags
  RESULT=   2.00000048
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
% gfortran -O -o testdqags testdqags.f libquadpack.a
% ./testdqags
 RESULT=   1.99999999999999844569
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
```

(誤差はそれぞれ 4.8×10^{-7}, 1.6×10^{-15} であるから) 単精度、倍精度とも十分な結果であろう。

3.4 QAGP (?)

QAGP serves the same purposes as QAGS, but also allows for eventual user-supplied information, i.e. the abscissae of internal singularities, discontinuities and other difficulties of the integrand function. The algorithm is a modification of that in QAGS.

QAGP は QAGS と同じ目的を果たすが、ユーザーが提供する情報、すなわち、被積分関数の内部特異点、不連続点、その他の問題のある点の座標 (横座標) を使用することができる。アルゴリズムは QAGS のものを改良したものである。

$$F(x) = |x - 1/7|^{-0.25} \, |x - 2/3|^{-0.55}$$

$x = \dfrac{1}{7}, \dfrac{2}{3}$ では $F(x) = 0$ としてある。

$$\int_0^1 \left| x - \frac{1}{7} \right|^{-1/4} \left| x - \frac{2}{3} \right|^{-11/20} dx = 4.25368768812224946 11 \cdots$$

testqagp.f

```
      REAL A,ABSERR,B,EPSABS,EPSREL,F,POINTS,RESULT,WORK
      INTEGER IER,IWORK,LAST,LENIW,LENW,LIMIT,NEVAL,NPTS2
      DIMENSION IWORK(204),POINTS(4),WORK(404)
      EXTERNAL F
      A = 0.0E0
      B = 1.0E0
      NPTS2 = 4
      POINTS(1) = 1.0E0/7.0E0
      POINTS(2) = 2.0E0/3.0E0
      LIMIT = 100
      LENIW = LIMIT*2+NPTS2
      LENW = LIMIT*4+NPTS2
      CALL QAGP(F,A,B,NPTS2,POINTS,EPSABS,EPSREL,RESULT,ABSERR,
     *  NEVAL,IER,LENIW,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      F = 0.0E+00
      IF(X.NE.1.0E0/7.0E0.AND.X.NE.2.0E0/3.0E0) F =
     *  ABS(X-1.0E0/7.0E0)**(-0.25E0)*
```

```
     *  ABS(X-2.0E0/3.0E0)**(-0.55E0)
      RETURN
      END
```

testdqagp.f

```
C testdqagp.f
      double precision A,ABSERR,B,EPSABS,EPSREL,F,POINTS,RESULT,WORK
      INTEGER IER,IWORK,LAST,LENIW,LENW,LIMIT,NEVAL,NPTS2
      DIMENSION IWORK(204),POINTS(4),WORK(404)
      EXTERNAL F
      A = 0.0D0
      B = 1.0D0
      NPTS2 = 4
      POINTS(1) = 1.0D0/7.0D0
      POINTS(2) = 2.0D0/3.0D0
      LIMIT = 100
      LENIW = LIMIT*2+NPTS2
      LENW = LIMIT*4+NPTS2
      CALL DQAGP(F,A,B,NPTS2,POINTS,EPSABS,EPSREL,RESULT,ABSERR,
     *  NEVAL,IER,LENIW,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,'(A,F20.15)') ' RESULT=',RESULT
      STOP
      END
C      C
      double precision FUNCTION F(X)
      double precision X
      F = 0.0D+00
      IF(X.NE.1.0D0/7.0D0.AND.X.NE.2.0D0/3.0D0) F =
     *  DABS(X-1.0D0/7.0D0)**(-0.25D0)*
     *  DABS(X-2.0D0/3.0D0)**(-0.55D0)
      RETURN
      END
```

testqagp の実行結果

```
% gfortran -O -o testqagp testqagp.f libquadpack.a
% ./testqagp
  RESULT=   4.16159010
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
% gfortran -O -o testdqagp testdqagp.f libquadpack.a
% ./testdqagp
 *** XERROR ***
 Message:abnormal return from dqagp
 Error code:           4
 Severity level:           0
 RESULT=   4.252989528653469
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
```

あれ？？単精度の方は最初の桁しかあっていなくて、倍精度の方は誤差が 7.0×10^{-4} だ。おかしいな。

3.5 QAGI (成功)

QAGI handles integration over infinite intervals. The infinite range is mapped onto a finite interval and then the same strategy as in QAGS is applied.

QAGI は無限区間の積分を扱う。無限範囲は有限区間に写像され、QAGS と同じ戦略が適用される。

ALOG(x) は単精度の $\log x$ を意味する。

サンプル・プログラムでは次の定積分を計算していると考えられる。

$$\int_0^{+\infty} \frac{\sqrt{x}\ \log x}{(x+1)(x+2)} dx = \sqrt{2}\pi \log 2 = 3.0795717821423574190\cdots$$

testqagi.f

```
      REAL ABSERR,BOUN,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,INF,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      BOUN = 0.0E0
      INF = 1
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      LIMIT = 100
      LENW = LIMIT*4
      CALL QAGI(F,BOUN,INF,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,
     *  IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      F = 0.0E0
      IF(X.GT.0.0E0) F = SQRT(X)*ALOG(X)/
     *                   ((X+1.0E0)*(X+2.0E0))
      RETURN
      END
```

testdqagi.f

```
C testdqagi.f
      double precision ABSERR,BOUN,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,INF,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      BOUN = 0.0D0
      INF = 1
      EPSABS = 0.0D0
C      EPSREL = 1.0D-3
      EPSREL = 1.0D-13
      LIMIT = 100
      LENW = LIMIT*4
      CALL DQAGI(F,BOUN,INF,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,
     *  IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,'(A,F25.20)') ' RESULT=',RESULT
      STOP
      END
C      C
      double precision FUNCTION F(X)
      double precision X
      F = 0.0D0
      IF(X.GT.0.0D0) F = DSQRT(X)*DLOG(X)/
     *                   ((X+1.0D0)*(X+2.0D0))
      RETURN
      END
```

testqagi の実行結果

```
% gfortran -O -o testqagi testqagi.f libquadpack.a
% ./testqagi
  RESULT=   3.07956362
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
% gfortran -O -o testdqagi testdqagi.f libquadpack.a
% ./testdqagi
 RESULT=   3.07957178214236915181
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
```

単精度の方は、誤差が 8.2×10^{-6}. こんなものか？ (ほぼ 10^{-5} で少し大きい気がする。)
倍精度の方は誤差が 1.2×10^{-14} で、こんなものか？？

3.6 QAWO (成功)

QAWO is a routine for the integration of COS(OMEGA*X)*F(X) or SIN(OMEGA*X)*F(X) over a finite interval (A,B). OMEGA is is specified by the user. The rule evaluation component is based on the modified Clenshaw-Curtis technique. An adaptive subdivision scheme is used connected with an extrapolation procedure, which is a modification of that in QAGS and provides the possibility to deal even with singularities in F.

QAWO は有限区間 (A,B) 上の COS(OMEGA*X)*F(X) または SIN(OMEGA*X)*F(X) の積分のためのルーチンです。OMEGA はユーザが指定する。ルール評価コンポーネントは、修正 Clenshaw-Curtis 法に基づいている。適応的細分割スキームは、QAGS のそれを修正した外挿手順と関連して使用され、F の特異点を扱う可能性を提供する。

以下のサンプル・プログラムでは、次の定積分を計算していると考えられる。

$$\begin{aligned}
&\int_0^1 \cos(10x)e^{-x}\log x\, dx \\
&\quad = \frac{1}{202}\left(-2\gamma - 20\,\mathrm{Arctan}(10) - (1+10i)\Gamma(1-10i) - (1-10i)\Gamma(1+10i) - \log 10\right) \\
&\quad = -0.17763920651138898590\cdots .
\end{aligned}$$

testqawo.f

```
      REAL A,ABSERR,B,EPSABS,EPSREL,F,RESULT,OMEGA,WORK
      INTEGER IER,INTEGR,IWORK,LAST,LENIW,LENW,LIMIT,MAXP1,NEVAL
      DIMENSION IWORK(200),WORK(925)
      EXTERNAL F
      A = 0.0E0
      B = 1.0E0
      OMEGA = 10.0E0
      INTEGR = 1
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      LIMIT = 100
      LENIW = LIMIT*2
      MAXP1 = 21
      LENW = LIMIT*4+MAXP1*25
      CALL QAWO(F,A,B,OMEGA,INTEGR,EPSABS,EPSREL,RESULT,ABSERR,
     *  NEVAL,IER,LENIW,MAXP1,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
```

```
C      C
       REAL FUNCTION F(X)
       REAL X
       F = 0.0E0
       IF(X.GT.0.0E0) F = EXP(-X)*ALOG(X)
       RETURN
       END
```

testdqawo.f

```
C testdqawo.f
       double precision A,ABSERR,B,EPSABS,EPSREL,F,RESULT,OMEGA,WORK
       INTEGER IER,INTEGR,IWORK,LAST,LENIW,LENW,LIMIT,MAXP1,NEVAL
       DIMENSION IWORK(200),WORK(925)
       EXTERNAL F
       A = 0.0D0
       B = 1.0D0
       OMEGA = 10.0D0
       INTEGR = 1
       EPSABS = 0.0D0
C      EPSREL = 1.0D-3
       EPSREL = 1.0D-12
       LIMIT = 100
       LENIW = LIMIT*2
       MAXP1 = 21
       LENW = LIMIT*4+MAXP1*25
       CALL DQAWO(F,A,B,OMEGA,INTEGR,EPSABS,EPSREL,RESULT,ABSERR,
     *  NEVAL,IER,LENIW,MAXP1,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
       write(*,'(A,F25.20)') ' RESULT=',RESULT
       STOP
       END
C      C
       double precision FUNCTION F(X)
       double precision X
       F = 0.0D0
       IF(X.GT.0.0D0) F = DEXP(-X)*DLOG(X)
       RETURN
       END
```

testqawo の実行結果

```
% gfortran -O -o testqawo testqawo.f libquadpack.a
% ./testqawo
  RESULT= -0.177639186
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
% gfortran -O -o testdqawo testdqawo.f libquadpack.a
% ./testdqawo
 RESULT=  -0.17763920651138892515
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
```

(誤差はそれぞれ 2.1×10^{-8}, 6.1×10^{-17} であるから) 単精度、倍精度とも十分な結果であろう。

3.7 QAWF (成功)

QAWF calculates the Fourier cosine or Fourier sine transform of F(X), for user-supplied interval (A,INFINITY), OMEGA, and F. The procedure of QAWO is used on successive finite

intervals, and convergence acceleration by means of the Epsilon algorithm (Wynn, 1956) is applied to the series of the integral contributions.

QAWF は、ユーザーが指定した区間 (A,INFINITY)、OMEGA、F に対して、F(X) のフーリエ余弦変換

$$\int_a^{+\infty} F(x)\cos(\omega x)\,dx$$

またはフーリエ正弦変換

$$\int_a^{+\infty} F(x)\sin(\omega x)\,dx$$

を計算します。QAWO の手順は、連続する有限区間で使用され、イプシロンアルゴリズム (Wynn, 1956) による収束加速が積分寄与の系列に適用されます。

$F(x) = \dfrac{\sin(50x)}{x\sqrt{x}}$, $\omega = 8$, $a = 0$, INTEGR=2 なので

$$\int_0^{+\infty} \frac{\sin(50x)}{x\sqrt{x}}\sin(8x)dx = \sqrt{29\pi} - \sqrt{21\pi} = 1.4225521625759122427\cdots .$$

testqawf.f

```
      REAL A,ABSERR,EPSABS,F,RESULT,OMEGA,WORK
      INTEGER IER,INTEGR,IWORK,LAST,LENIW,LENW,LIMIT,LIMLST,
     *  LST,MAXP1,NEVAL
      DIMENSION IWORK(250),WORK(1025)
      EXTERNAL F
      A = 0.0E0
      OMEGA = 8.0E0
      INTEGR = 2
      EPSABS = 1.0E-3
      LIMLST = 50
      LIMIT = 100
      LENIW = LIMIT*2+LIMLST
      MAXP1 = 21
      LENW = LENIW*2+MAXP1*25
      CALL QAWF(F,A,OMEGA,INTEGR,EPSABS,RESULT,ABSERR,NEVAL,
     *  IER,LIMLST,LST,LENIW,MAXP1,LENW,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      IF(X.GT.0.0E0) F = SIN(50.0E0*X)/(X*SQRT(X))
      RETURN
      END
```

testdqawf.f

```
C testdqawf.f
      double precision A,ABSERR,EPSABS,F,RESULT,OMEGA,WORK
      INTEGER IER,INTEGR,IWORK,LAST,LENIW,LENW,LIMIT,LIMLST,
     *  LST,MAXP1,NEVAL
      DIMENSION IWORK(250),WORK(1025)
      EXTERNAL F
      A = 0.0D0
      OMEGA = 8.0D0
      INTEGR = 2
```

```
C      EPSABS = 1.0D-3
       EPSABS = 1.0D-9
       LIMLST = 50
       LIMIT = 100
       LENIW = LIMIT*2+LIMLST
       MAXP1 = 21
       LENW = LENIW*2+MAXP1*25
       CALL DQAWF(F,A,OMEGA,INTEGR,EPSABS,RESULT,ABSERR,NEVAL,
      *  IER,LIMLST,LST,LENIW,MAXP1,LENW,IWORK,WORK)
C       C  INCLUDE WRITE STATEMENTS
       write(*,'(A,F25.20)') ' RESULT=',RESULT
       STOP
       END
C       C
       double precision FUNCTION F(X)
       double precision X
       IF(X.GT.0.0D0) F = DSIN(50.0D0*X)/(X*DSQRT(X))
       RETURN
       END
```

testqawf の実行結果

```
% gfortran -O -o testqawf testqawf.f libquadpack.a
% ./testqawf
  RESULT=   1.42255270
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
% gfortran -O -o testdqawf testdqawf.f libquadpack.a
% ./testdqawf
 RESULT=   1.42255216257491468035
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
```

単精度の方は誤差が 5.4×10^{-7} であるから、十分な結果であろう。
倍精度の方は誤差が 1.0×10^{-12} である。(EPSABS = 1.0D-10 とするとエラーになる。)

3.8 QAWS (成功)

QAWS integrates W(X)*F(X) over (A,B) with A.LT.B finite, and W(X) = ((X-A)**ALFA)*((B-X)**BETA)*V(X) where V(X) = 1 or LOG(X-A) or LOG(B-X) or LOG(X-A)*LOG(B-X) and ALFA.GT.(-1), BETA.GT.(-1). The user specifies A, B, ALFA, BETA and the type of the function V. A globally adaptive subdivision strategy is applied, with modified Clenshaw-Curtis integration on the subintervals which contain A or B.

QAWSは, A.LT.Bを有限とする(A,B)上のW(X)*F(X)を積分し, W(X)=((X-A)**ALFA)*((B-X)**BETA)*V(X) ここで, V(X)=1またはLOG(X-A)またはLOG(B-X)またはLOG(X-A)*LOG(B-X)であり，ALFA.GT.(-1)，BETA.GT.(-1)である．ユーザーは、A, B, ALFA, BETA および関数Vのタイプを指定する。AまたはBを含む部分区間では、修正されたクレンショー-カーティス積分を用いて、グローバルに適応する細分化戦略が適用される。

以上をまとめると

$$\int_a^b f(x)w(x)\,dx, \quad w(x) = \begin{cases} (x-a)^\alpha (b-x)^\beta & (\text{integr} = 1) \\ (x-a)^\alpha (b-x)^\beta \log(x-a) & (\text{integr} = 2) \\ (x-a)^\alpha (b-x)^\beta \log(b-x) & (\text{integr} = 3) \\ (x-a)^\alpha (b-x)^\beta \log(x-a)\log(b-x) & (\text{integr} = 4) \end{cases}$$

を計算するということであろう。

以下のサンプル・プログラムでは、$f(x) = \sin 10x$, $a = 0$, $b = 1$, integr=1, $\alpha = \beta = -1/2$ なので、重み関数は $w(x) = (x-a)^{\alpha}(b-x)^{\beta} = x^{-0.5}(1-x)^{-0.5}$ である。

$$\int_0^1 \sin(10x)x^{-0.5}(1-x)^{-0.5}dx = \pi J_0(5)\sin 5 = 0.53501905692236534412\cdots$$

testqaws.f

```
      REAL A,ABSERR,ALFA,B,BETA,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,INTEGR,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      A = 0.0E0
      B = 1.0E0
      ALFA = -0.5E0
      BETA = -0.5E0
      INTEGR = 1
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      LIMIT = 100
      LENW = LIMIT*4
      CALL QAWS(F,A,B,ALFA,BETA,INTEGR,EPSABS,EPSREL,RESULT,
     *  ABSERR,NEVAL,IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      F = SIN(10.0E0*X)
      RETURN
      END
```

testdqaws.f

```
C testdqaws.f
      double precision A,ABSERR,ALFA,B,BETA,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,INTEGR,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      A = 0.0D0
      B = 1.0D0
      ALFA = -0.5D0
      BETA = -0.5D0
      INTEGR = 1
      EPSABS = 0.0D0
C      EPSREL = 1.0D-3
      EPSREL = 1.0D-12
      LIMIT = 100
      LENW = LIMIT*4
      CALL DQAWS(F,A,B,ALFA,BETA,INTEGR,EPSABS,EPSREL,RESULT,
     *  ABSERR,NEVAL,IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,'(A,F25.20)') ' RESULT=',RESULT
      STOP
      END
C      C
      double precision FUNCTION F(X)
```

```
      double precision X
      F = DSIN(10.0D0*X)
      RETURN
      END
```

testqaws の実行結果

```
% gfortran -O -o testqaws testqaws.f libquadpack.a
% ./testqaws
  RESULT=  0.535019159
% gfortran -O -o testdqaws testdqaws.f libquadpack.a
% ./testdqaws
 RESULT=    0.53501905692236562118
```

誤差は単精度、倍精度それぞれ 1.0×10^{-7}, 2.8×10^{-16}、十分な結果であろう。

3.9 QAWC (成功)

QAWC computes the Cauchy Principal Value of F(X)/(X-C) over a finite interval (A,B) and for user-determined C. The strategy is globally adaptive, and modified Clenshaw-Curtis integration is used on the subranges which contain the point X = C.

WC は F(X)/(X-C) のコーシー主値を有限区間 (A,B) 上で、ユーザーが決めた C に対して計算する。この戦略は大域的に適応的であり、点 X = C を含む部分範囲では修正クレンショウ・カーティス積分が使われる。

$$\text{p.v.} \int_a^b \frac{f(x)}{x-c}\,dx$$

以下のプログラムでは、$F(x) = \dfrac{1}{x^2+10^{-4}}$, $a=-1$, $b=1$, $c=1/2$,

$$\text{p.v.} \int_{-1}^{1} \frac{1}{(x^2+1\times 10^{-4})\,(x-1/2)}dx = -\frac{1000\,(100\,\mathrm{Arctan}\,100+\log 3)}{2501}$$
$$= -628.46172850656236622\cdots .$$

```
In[80]:= Integrate[1 / ((x^2 + 10^(-4)) (x - 1 / 2)), {x, -1, 1},
          積分
          PrincipalValue → True]
          主値                真
Out[80]= -(10 000 (100 ArcTan[100] + Log[3]))/2501

In[81]:= N[%, 22]
          数値
Out[81]= -628.4617285065623662291
```

図 1: Mathematica での検算

testqawc.f

```
      REAL A,ABSERR,B,C,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
```

```
      A = -1.0E0
      B = 1.0E0
      C = 0.5E0
      EPSABS = 0.0E0
      EPSREL = 1.0E-3
      LIMIT = 100
      LENW = LIMIT*4
      CALL QAWC(F,A,B,C,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,
     *  IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,*) ' RESULT=',RESULT
      STOP
      END
C      C
      REAL FUNCTION F(X)
      REAL X
      F = 1.0E0/(X*X+1.0E-4)
      RETURN
      END
```

testdqawc.f

```
C testdqawc.f
      double precision A,ABSERR,B,C,EPSABS,EPSREL,F,RESULT,WORK
      INTEGER IER,IWORK,LAST,LENW,LIMIT,NEVAL
      DIMENSION IWORK(100),WORK(400)
      EXTERNAL F
      A = -1.0D0
      B = 1.0D0
      C = 0.5D0
      EPSABS = 0.0D0
C      EPSREL = 1.0D-3
      EPSREL = 1.0D-13
      LIMIT = 100
      LENW = LIMIT*4
      CALL DQAWC(F,A,B,C,EPSABS,EPSREL,RESULT,ABSERR,NEVAL,
     *  IER,LIMIT,LENW,LAST,IWORK,WORK)
C      C  INCLUDE WRITE STATEMENTS
      write(*,'(A,F25.20)') ' RESULT=',RESULT
      STOP
      END
C      C
      double precision FUNCTION F(X)
      double precision X
      F = 1.0D0/(X*X+1.0D-4)
      RETURN
      END
```

testqawc の実行結果

```
% gfortran -O -o testqawc testqawc.f libquadpack.a
% ./testqawc
  RESULT=  -628.461731
% gfortran -O -o testdqawc testdqawc.f libquadpack.a
% ./testdqawc
 RESULT=-628.46172850656239461387
```

単精度の方は (絶対) 誤差 2.5×10^{-6} (相対誤差は 2.4×10^{-9})、十分な結果であろう。
倍精度の方も絶対誤差 2.8×10^{-14} (相対誤差は 4.5×10^{-17})、十分な結果であろう。

3.10　結果のまとめ

前項までの結果を表にまとめた。

すべて絶対誤差である。ほとんどの問題で真の値が 1 のオーダーであるが、最後の QAWC だけは真の値が $-628.4\cdots$ と絶対値が大きいので、相対誤差は 2 桁小さい。すると十分な高精度ということがわかる。

QAGP のみ非常に精度が低い。何かバグでもあるのだろうか。

QAGP 以外は十分な高精度と思われる。QAWF の倍精度は少し精度が低い (理由は不明である)。QAGI の精度も低いが、満足出来るだろうか。

	QNG	QAG	QAGS	QAGP	QAGI	QAWO	QAWF	QAWS	QAWC
単精度	1.6×10^{-7}	1.8×10^{-8}	4.8×10^{-7}	1 桁	8.2×10^{-6}	2.1×10^{-8}	5.4×10^{-7}	1.0×10^{-7}	2.5×10^{-6}
倍精度	2.2×10^{-16}	6.7×10^{-17}	1.6×10^{-15}	7.0×10^{-4}	1.2×10^{-14}	6.1×10^{-17}	1.0×10^{-12}	2.8×10^{-16}	2.8×10^{-14}

4　中身を理解する

(しばらく工事中)

4.1　何を理解すべきか

Quadpack では、Gauss-Kronrod 公式、Clenshaw-Curtis 公式が使われている。これらの方法の有効性は認められているようで、Quadpack 以外にも多くのソフトウェアで採用されている (Mathematica, MATLAB, Python, Boost, ...)。ところがこの 2 つの方法については、率直に言って日本語の解説が少ない。資料を探索して、読むべきものを見つけ、エッセンスを学ぶ必要がある。

4.2　Gauss-Kronrod 公式

Gauss-Kronrod 法については、Gauss–Kronrod 求積法[2] が最初に読むのに良いかも。

オリジナルは Kronrod [3] (ロシア語から英語への翻訳) なのか？

膨大な数の論文が出版されている (一体どれが重要なのか分からない)。Monegato [4], [5], Calio-Gautschi-Marchetti [6], Notaris [7] を見よ。

Kronrod 点の計算については、Laurie[8] (MATLAB で計算する話、多倍長ツールボックスを使うと好きな桁数の値が得られるとか), Piessens-Branders [9] (Quadpack 開発者の一人), Monegato Laurie[10] がある。

杉原-室田では言及されていない。

Stieltjes 多項式というのは古くから知られているようであるが…

4.3　Clenshaw-Curtis 公式

杉原-室田には、「Clenshaw-Curtis 公式 分点を

$$x_k = \frac{(b-a)\cos(\pi k/n) + (b+a)}{2} \quad (k = 0, 1, \cdots, n)$$

とする補間型公式である. 性質の良い関数に対しては，Gauss-Legendre 公式と同程度に有効である.」と書かれている (This is the all)。

[2] https://qiita.com/Baruim/items/1e8237833f77b2e0d870

Clenshaw-Curtis 法については、Chawla [11], Riess-Johnson [12], Trefethen [13] がある。特に Gauss 型公式との比較について述べた [13] は興味深い。

私なりの [13] の解読 (桂田 [14])

Clenshaw-Curtis 積分公式は、実質上 Gauss 型積分公式に迫る性能である。一部では知られていたその事実を SIAM Review という論文誌で宣言して、それを簡潔な MATLAB コードによる数値実験で裏付け、複素積分による誤差解析で説明した。

https://en.wikipedia.org/wiki/Clenshaw%E2%80%93Curtis_quadrature

大浦氏によるプログラム intcc.tar.gz[3] が https://www.kurims.kyoto-u.ac.jp/~ooura/intcc-j.html で公開されている。

A Quadpack のコンパイル

A.1 Netlib からとりあえずソースプログラムを入手

https://www.netlib.org/quadpack/

```
wget --mirror --no-parent -P quadpack -nH https://www.netlib.org/quadpack/
```

A.2 Makefile version 0

まず、Netlib から入手した Fortran プログラムをコンパイルして、1 つのライブラリィ・アーカイブ・ファイルにまとめてみる。

Makefile version 0

```
#
# Makefile for quadpack (version 0)
#
FC      =       gfortran
FFLAGS  =       -O
SRCS    =       dqag.f dqage.f dqagi.f dqagie.f dqagp.f dqagpe.f \
                dqags.f dqagse.f dqawc.f dqawce.f dqawf.f dqawfe.f \
                dqawo.f dqawoe.f dqaws.f dqawse.f dqc25c.f dqc25f.f \
                dqc25s.f dqcheb.f dqelg.f dqk15.f dqk15i.f dqk15w.f \
                dqk21.f dqk31.f dqk41.f dqk51.f dqk61.f dqmomo.f \
                dqng.f dqpsrt.f dqwgtc.f dqwgtf.f dqwgts.f qag.f \
                qage.f qagi.f qagie.f qagp.f qagpe.f qags.f qagse.f \
                qawc.f qawce.f qawf.f qawfe.f qawo.f qawoe.f qaws.f \
                qawse.f qc25c.f qc25f.f qc25s.f qcheb.f qelg.f qk15.f \
                qk15i.f qk15w.f qk21.f qk31.f qk41.f qk51.f qk61.f \
                qmomo.f qng.f qpsrt.f qwgtc.f qwgtf.f qwgts.f
OBJS    =       dqag.o dqage.o dqagi.o dqagie.o dqagp.o dqagpe.o \
                dqags.o dqagse.o dqawc.o dqawce.o dqawf.o dqawfe.o \
                dqawo.o dqawoe.o dqaws.o dqawse.o dqc25c.o dqc25f.o \
                dqc25s.o dqcheb.o dqelg.o dqk15.o dqk15i.o dqk15w.o \
                dqk21.o dqk31.o dqk41.o dqk51.o dqk61.o dqmomo.o \
                dqng.o dqpsrt.o dqwgtc.o dqwgtf.o dqwgts.o qag.o \
                qage.o qagi.o qagie.o qagp.o qagpe.o qags.o qagse.o \
                qawc.o qawce.o qawf.o qawfe.o qawo.o qawoe.o qaws.o \
                qawse.o qc25c.o qc25f.o qc25s.o qcheb.o qelg.o qk15.o \
```

[3] https://www.kurims.kyoto-u.ac.jp/~ooura/intcc.tar.gz

```
                qk15i.o qk15w.o qk21.o qk31.o qk41.o qk51.o qk61.o \
                qmomo.o qng.o qpsrt.o qwgtc.o qwgtf.o qwgts.o
LIB     =       libquadpack.a

libquadpack0.a: $(OBJS)
        ar cr $@ $(OBJS) && ranlib $@

clean:
        rm -f $(OBJS)

vc:
        rm -f $(OBJS) $(LIB)
```

A.3 何が足りないのか

前項までで一応コンパイルしてライブラリィ・アーカイブ・ファイルが出来るが、テスト・プログラムのコンパイル＆リンクには失敗する。

リンクでエラーになる

```
% gfortran testqag.f libquadpack0.a
Undefined symbols for architecture arm64:
  "_r1mach_", referenced from:
      _qage_ in libquadpack0.a[38](qage.o)
      _qage_ in libquadpack0.a[38](qage.o)
      _qk15_ in libquadpack0.a[58](qk15.o)
      _qk15_ in libquadpack0.a[58](qk15.o)
      _qk21_ in libquadpack0.a[61](qk21.o)
      _qk21_ in libquadpack0.a[61](qk21.o)
      _qk31_ in libquadpack0.a[62](qk31.o)
      _qk31_ in libquadpack0.a[62](qk31.o)
      ...
  "_xerror_", referenced from:
      _qag_ in libquadpack0.a[37](qag.o)
ld: symbol(s) not found for architecture arm64
collect2: error: ld returned 1 exit status
mk@katsuradayuujinoMacBook-Pro quadpack %
```

r1mach と xerror がない。

A.3.1 r1mach, d1mach, i1mach

https://netlib.org/slatec/src/r1mach.f で良いか？

```
C   R1MACH can be used to obtain machine-dependent parameters for the
C   local machine environment.  It is a function subprogram with one
C   (input) argument, and can be referenced as follows:
C
C        A = R1MACH(I)
C
C   where I=1,...,5.  The (output) value of A above is determined by
C   the (input) value of I.  The results for various values of I are
C   discussed below.
C
C   R1MACH(1) = B**(EMIN-1), the smallest positive magnitude.
C   R1MACH(2) = B**EMAX*(1 - B**(-T)), the largest magnitude.
C   R1MACH(3) = B**(-T), the smallest relative spacing.
C   R1MACH(4) = B**(1-T), the largest relative spacing.
C   R1MACH(5) = LOG10(B)
```

r1mach は、単精度の浮動小数点数の “パラメーター” を尋ねるサブルーチンということだ。中をのぞくと、色々なシステムのパラメーターが注釈として書いてあって、コメントを外して使う、そういうプログラムになっている。システムとしては、とても古いものが並んでいる。中で一番新しいものが Sun Workstation (古い！) で、それは IEEE 754 なので、値としては今使おうとしているシステム (Mac) と実質同じなのかもしれないが、値の設定の仕方が Sun の Fortran コンパイラー向きで、gfortran には合わない。この際、新しくコードを作ることにした。こういうのは ChatGPT にやらせよう。

もう一つ、r1mac の中でエラー出力をするために (引数が想定している 1, 2, 3, 4, 5 のいずれでもなかった場合— これは明らかにコーディングのミスでしか起こらない) `XERMSG` というサブルーチンを call しているが、`xermsg.f` を持ってくると、芋づる式にたくさんの Fortran プログラムが必要になり、それらをコンパイル出来るように修正するのが結構面倒なことが分かった。これは簡単なメッセージの出力に置き換えれば十分であろう。

ChatGPT 製 `r1mach.f` — IEE754 単精度向き

```
      FUNCTION R1MACH(I)
      REAL R1MACH
      INTEGER I
      REAL B, T, EMIN, EMAX
      REAL SMALL, LARGE, EPS, RSP, LOGB

      PARAMETER (B = 2.0, T = 24.0)
      PARAMETER (EMIN = -125.0, EMAX = 127.0)

      SMALL = B**(EMIN - 1.0)
      LARGE = B**EMAX * (1.0 - B**(-T))
      EPS   = B**(-T)
      RSP   = B**(1.0 - T)
      LOGB  = LOG10(B)

      SELECT CASE (I)
      CASE (1)
          R1MACH = SMALL
      CASE (2)
          R1MACH = LARGE
      CASE (3)
          R1MACH = EPS
      CASE (4)
          R1MACH = RSP
      CASE (5)
          R1MACH = LOGB
      CASE DEFAULT
```

```
          WRITE(0,*) 'R1MACH: Invalid index I =', I
          STOP 1
      END SELECT

      RETURN
      END
```

後でこの倍精度版 d1mach.f が必要になる。

ChatGPT 製 d1mach.f — IEE754 倍精度向き

```
      FUNCTION D1MACH(I)
      DOUBLE PRECISION D1MACH
      INTEGER I
      DOUBLE PRECISION B, T, EMIN, EMAX
      DOUBLE PRECISION SMALL, LARGE, EPS, RSP, LOGB

      DATA B /2.0D0/
      DATA T /53.0D0/
      DATA EMIN /-1021.0D0/
      DATA EMAX /1024.0D0/

      SMALL = B**(EMIN - 1.0D0)
      EPS   = B**(-T)
      RSP   = B**(1.0D0 - T)
      LOGB  = LOG10(B)

      LARGE = EXP(LOG(B) * EMAX) * (1.0D0 - EPS)

      SELECT CASE (I)
      CASE (1)
          D1MACH = SMALL
      CASE (2)
          D1MACH = LARGE
      CASE (3)
          D1MACH = EPS
      CASE (4)
          D1MACH = RSP
      CASE (5)
          D1MACH = LOGB
      CASE DEFAULT
          WRITE(0,*) 'D1MACH: Invalid index I =', I
          STOP 1
      END SELECT

      RETURN
      END
```

浮動小数点数とは関係ないが、システムの設定を記述する i1mach.f というものがある。https://www.netlib.org/slatec/src/i1mach.f が参考になる。

https://www.netlib.org/slatec/src/i1mach.f から抜粋

```
C   I/O unit numbers:
C     I1MACH( 1) = the standard input unit.
C     I1MACH( 2) = the standard output unit.
C     I1MACH( 3) = the standard punch unit.
C     I1MACH( 4) = the standard error message unit.
C
C   Words:
C     I1MACH( 5) = the number of bits per integer storage unit.
C     I1MACH( 6) = the number of characters per integer storage unit.
```

```
C
C   Integers:
C     assume integers are represented in the S-digit, base-A form
C
C                 sign ( X(S-1)*A**(S-1) + ... + X(1)*A + X(0) )
C
C                 where 0 .LE. X(I) .LT. A for I=0,...,S-1.
C     I1MACH( 7) = A, the base.
C     I1MACH( 8) = S, the number of base-A digits.
C     I1MACH( 9) = A**S - 1, the largest magnitude.
```

Quadpack で必要になる部分だけに対応した自作 i1mach.f を参考までに掲げておく。

ChatGPT 製 i1mach.f

```
      FUNCTION I1MACH(I)
      INTEGER I1MACH, I
      INTEGER NBITS, INTMIN, INTMAX

      NBITS = BIT_SIZE(0)
      INTMIN = -2**(NBITS - 1)
      INTMAX =  2**(NBITS - 1) - 1

      SELECT CASE (I)
      CASE (1)
          I1MACH = 5
      CASE (2)
          I1MACH = 6
      CASE (3)
          I1MACH = 7
      CASE (4)
          I1MACH = 0
      CASE (5)
          I1MACH = 5
      CASE (6)
          I1MACH = 6
      CASE (7)
          I1MACH = 0
      CASE (8)
          I1MACH = NBITS
      CASE (9)
          I1MACH = INTMIN
      CASE (10)
          I1MACH = INTMAX
      CASE DEFAULT
          WRITE(0,*) 'I1MACH: Invalid index I =', I
          STOP 1
      END SELECT

      RETURN
      END
```

しかし、自作 xerror.f を使う限り、これを call することはない。

A.3.2 xerror

xerror も元々は SLATEC に含まれるものだそうだ。

ウィキペディアの SLATEC 解説

SLATEC Common Mathematical Library is a FORTRAN 77 library of over 1,400 general purpose mathematical and statistical routines. The code was developed at US government research laboratories and is therefore public domain software.

SLATEC 共通数学ライブラリは、1,400 以上の汎用数学・統計ルーチンを含む FORTRAN77 ライブラリである。このコードは米国政府の研究所で開発されたものであるので、パブリックドメインソフトウェアである。

SLATEC のソースプログラムは、現在も Netlib で配布されている。しかし、現在は `xerror.f` は SLATEC には含まれていない。一番近そうなものに alliant の `xerror.f`[4] というのがある。

alliant/quad

A subset of the quadpack routines optimized for the alliant. To get the index use "send index from alliant/quad". The complete non-Alliant versions are available via "send index from quadpack"

by Dick Hessel, Oct 16, 1987

そこにある `xerror.f` の中をのぞくと、`xerrwv` というサブルーチンを call している。これは Quadpack にはなくて、やはり SLATEC に入っているものか？少し探したところ、`xerrwv.f`[5] が一番近いものだろうか？

その `xerrwv.f` の中でまた別のサブルーチンを call している…(xerprt, xersav, fdump, xerabt, xerctl, xgetua — 当然 `xerprt.f`, `xersav.f`, `fdump.f`, `xerabt.f`, `xerctl.f`, `xgetua.f` を入手することになる？これらは一応入手可能ではある。

alliant のプログラムを入手 (して修正しようと考えたがやめた)

```
mkdir alliant
cd alliant
curl -O https://www.netlib.org/alliant/quad/xerror.f
curl -O https://www.netlib.org/alliant/quad/xerrwv.f
curl -O https://www.netlib.org/alliant/quad/xerprt.f
curl -O https://www.netlib.org/alliant/quad/xersav.f
curl -O https://www.netlib.org/alliant/quad/fdump.f
curl -O https://www.netlib.org/alliant/quad/xerabt.f
curl -O https://www.netlib.org/alliant/quad/xerctl.f
curl -O https://www.netlib.org/alliant/quad/xgetua.f
```

しかし…これらは現在すんなりとはコンパイル出来ない。文字列を処理する機能に乏しかった古い FORTRAN で無理やり書いた、という感じのコーディングである。それを gfortran でコンパイルできるように修正するのはちょっと面倒くさい。

xerror の目的は単にエラー処理であるから、簡易バージョンを自作してすませることにする。

ChatGPT 製 `xerror.f` — いいかげんバージョン

```
C made by ChatGPT (2025/5/7)
      SUBROUTINE XERROR(MESSG, NMESSG, NERR, LEVEL)
      CHARACTER*(*) MESSG
      INTEGER NMESSG, NERR, LEVEL

C     簡素なエラーメッセージ表示
      WRITE(*,*) '*** XERROR ***'
```

[4] https://www.netlib.org/alliant/quad/xerror.f
[5] https://www.netlib.org/alliant/quad/xerrwv.f

```
      WRITE(*,*) 'Message:', MESSG(1:NMESSG)
      WRITE(*,*) 'Error code:', NERR
      WRITE(*,*) 'Severity level:', LEVEL

C     LEVEL >= 2 は重大なエラーと見なして終了
      IF (LEVEL .GE. 2) THEN
         CALL EXIT(-1)
      ENDIF

      RETURN
      END
```

A.4 sgtsl.f と dgtsl.f

前項までの修正を施すと、サンプル・プログラムのうちにはコンパイルできるものもあるが、sgtsl がないというエラーが生じることもある。それを call しているのは qc25f.f で

```
c           solve the tridiagonal system by means of gaussian
c           elimination with partial pivoting.
c
      call sgtsl(noequ,d1,d,d2,v(4),iers)
```

と書いてある。ネットで sgtsl.f を検査すると、LINPACK の https://www.netlib.org/linpack/sgtsl.f がひっかかる。三重対角 (行列) の方程式、つまり三項方程式ということで符合する感じがある。引数の個数も一致する。LINPACK も SLATEC のうちであるし。

あ、qc25f.f の倍精度版である dqc25f.f には

```
c           solve the tridiagonal system by means of gaussian
c           elimination with partial pivoting.
c
c***        call to dgtsl must be replaced by call to
c***        double precision version of linpack routine sgtsl
c
      call dgtsl(noequ,d1,d,d2,v(4),iers)
```

とある。これで sgtsl が LINPACK のサブルーチンであることはほぼ確実であろう。

ところで sgtsl の倍精度版の dgtsl であるが、dgtsl.f というのは見つからなかった。今回は sgtsl.f を元にして自作した。修正のポイントは以下の 3 つ。

- real を double precision にする。
- 0.0e0 を 0.0d0 にする (サボっても動くだろう)。
- abs() を dabs() にする。

A.5 Makefile version 1

前項までの修正でとりあえず完成する。Makefile を以下に示す。

Makefile version 1

```
#
```

```
# Makefile for quadpack (version 1)
#
FC      =       gfortran
FFLAGS  =       -O
SRCS1   =       dqag.f dqage.f dqagi.f dqagie.f dqagp.f dqagpe.f \
                dqags.f dqagse.f dqawc.f dqawce.f dqawf.f dqawfe.f \
                dqawo.f dqawoe.f dqaws.f dqawse.f dqc25c.f dqc25f.f \
                dqc25s.f dqcheb.f dqelg.f dqk15.f dqk15i.f dqk15w.f \
                dqk21.f dqk31.f dqk41.f dqk51.f dqk61.f dqmomo.f \
                dqng.f dqpsrt.f dqwgtc.f dqwgtf.f dqwgts.f qag.f \
                qage.f qagi.f qagie.f qagp.f qagpe.f qags.f qagse.f \
                qawc.f qawce.f qawf.f qawfe.f qawo.f qawoe.f qaws.f \
                qawse.f qc25c.f qc25f.f qc25s.f qcheb.f qelg.f qk15.f \
                qk15i.f qk15w.f qk21.f qk31.f qk41.f qk51.f qk61.f \
                qmomo.f qng.f qpsrt.f qwgtc.f qwgtf.f qwgts.f
SRCS2   =       r1mach.f d1mach.f
SRCS3   =       xerror.f
SRCS4   =       sgtsl.f dgtsl.f
OBJS1   =       dqag.o dqage.o dqagi.o dqagie.o dqagp.o dqagpe.o \
                dqags.o dqagse.o dqawc.o dqawce.o dqawf.o dqawfe.o \
                dqawo.o dqawoe.o dqaws.o dqawse.o dqc25c.o dqc25f.o \
                dqc25s.o dqcheb.o dqelg.o dqk15.o dqk15i.o dqk15w.o \
                dqk21.o dqk31.o dqk41.o dqk51.o dqk61.o dqmomo.o \
                dqng.o dqpsrt.o dqwgtc.o dqwgtf.o dqwgts.o qag.o \
                qage.o qagi.o qagie.o qagp.o qagpe.o qags.o qagse.o \
                qawc.o qawce.o qawf.o qawfe.o qawo.o qawoe.o qaws.o \
                qawse.o qc25c.o qc25f.o qc25s.o qcheb.o qelg.o qk15.o \
                qk15i.o qk15w.o qk21.o qk31.o qk41.o qk51.o qk61.o \
                qmomo.o qng.o qpsrt.o qwgtc.o qwgtf.o qwgts.o
OBJS2   =       r1mach.o d1mach.o
OBJS3   =       xerror.o
OBJS4   =       sgtsl.o dgtsl.o # testqawo.f
OBJS    =       $(OBJS1) $(OBJS2) $(OBJS3) $(OBJS4)
LIB     =       libquadpack0.a
TESTPROGS       =       testqag testqagp testqawc testqawo \
                        testqng testqagi testqags testqawf testqaws \
                        testdqag testdqagp testdqawc testdqawo \
                        testdqng testdqagi testdqags testdqawf testdqaws
####

libquadpack0.a: $(OBJS)
        ar cr $@ $(OBJS) && ranlib $@

####
test: $(TESTPROGS)
        ./testqag
        ./testdqag
        ./testqagp
        ./testdqagp
        ./testqawc
        ./testdqawc
        ./testqawo
        ./testdqawo
        ./testqng
        ./testdqng
        ./testqagi
        ./testdqagi
        ./testqags
        ./testdqags
        ./testqawf
        ./testdqawf
        ./testqaws
        ./testdqaws
```

```
testqag: testqag.f
	$(FC) $(FFLAGS) -o $@ testqag.f $(LIB)

testqagp: testqagp.f
	$(FC) $(FFLAGS) -o $@ testqagp.f $(LIB)

testqawc: testqawc.f
	$(FC) $(FFLAGS) -o $@ testqawc.f $(LIB)

testqawo: testqawo.f
	$(FC) $(FFLAGS) -o $@ testqawo.f $(LIB)

testqng: testqng.f
	$(FC) $(FFLAGS) -o $@ testqng.f $(LIB)

testqagi: testqagi.f
	$(FC) $(FFLAGS) -o $@ testqagi.f $(LIB)

testqags: testqags.f
	$(FC) $(FFLAGS) -o $@ testqags.f $(LIB)

testqawf: testqawf.f
	$(FC) $(FFLAGS) -o $@ testqawf.f $(LIB)

testqaws: testqaws.f
	$(FC) $(FFLAGS) -o $@ testqaws.f $(LIB)

testdqag: testdqag.f
	$(FC) $(FFLAGS) -o $@ testdqag.f $(LIB)

testdqagp: testdqagp.f
	$(FC) $(FFLAGS) -o $@ testdqagp.f $(LIB)

testdqawc: testdqawc.f
	$(FC) $(FFLAGS) -o $@ testdqawc.f $(LIB)

testdqawo: testdqawo.f
	$(FC) $(FFLAGS) -o $@ testdqawo.f $(LIB)

testdqng: testdqng.f
	$(FC) $(FFLAGS) -o $@ testdqng.f $(LIB)

testdqagi: testdqagi.f
	$(FC) $(FFLAGS) -o $@ testdqagi.f $(LIB)

testdqags: testdqags.f
	$(FC) $(FFLAGS) -o $@ testdqags.f $(LIB)

testdqawf: testdqawf.f
	$(FC) $(FFLAGS) -o $@ testdqawf.f $(LIB)

testdqaws: testdqaws.f
	$(FC) $(FFLAGS) -o $@ testdqaws.f $(LIB)

######
clean:
	rm -f $(OBJS)

vc:
	rm -f $(OBJS) $(LIB) $(PROGS) $(TESTPROGS)
```

A.6　公開しても問題ないかな

Quadpack はパブリック・ドメインに置かれていて、何 (再配布とか改変とか) をしてもしかられないようなので (もし私の勘違いであれば教えていただければ幸いです)、私が試したものを入手できるようにしておきます。何の保証もしません (自分の Mac で、MacPorts でインストールした gfortran 14.2 でコンパイル&実行してみただけです)。私自身は何の権利も主張しません。

- Quadpack 本体の Fotran プログラムは、何の改変もしていません。Netlib から取得したままの内容です。
- `xerror.f`, `r1mach.f`, `d1mach.f` は、Quadpack をとりあえずコンパイルできるように作ったものです。もともとの仕様 (そもそもそれが分かるようになっていない) を守るとかは、まったく考えていません。
- `sgtsl.f` は LINPACK に入っていたものです。これも Netlib から入手したままの状態です。`dgtsl.f` は `sgtsl.f` を元に作成したものです。

curl と gfortran がある状態で、ターミナルで次のようにすれば動く (かもしれません)。

入手、コンパイル、テスト実行

```
curl -O https://m-katsurada.sakura.ne.jp/misc/20250509/quadpack-mk.tar.gz
tar xzf quadpack-mk.tar.gz
cd quadpack-mk
make
make test
```

参考文献

[1] Piessens, R., Doncker-Kapenga, E., Überhuber, C. W. and Kahaner, D.: *QUADPACK: A subroutine package for automatic integration*, Springer-Verlag (1983), PDF でなら購入可能. 東大数理に所蔵 (買うか見に行くか…).

[2] Zwillinger, D.: *The Handbook of Integration*, A K Peters/CRC Press (1992/11/2).

[3] Kronrod, A. S.: *Nodes and Weights of Quadrature Formulas, English transl. from Russian*, Consultants Bureau (New York): 1965.

[4] Monegato, G.: A Note on Extended Gaussian Quadrature Rules, *MATHEMATICS OF COMPUTATION*, Vol. 30, No. 136, pp. 812–817 (1976).

[5] Monegato, G.: Stieltjes Polynomials and Related Quadrature Rules, *SIAM Repview*, Vol. 24, No. 2, pp. 137–158 (1982).

[6] Franca Calio, W. G. and Marchetti, E.: On Computing Gauss-Kronrod Quadrature Formulae, *Mathematics of Computation*, Vol. 47, No. 176, pp. 639–650 (1986).

[7] Notaris, S. E.: Gauss-Kronrod Quadrature Formulae — A Survey of Fifty Years of Research, *Electronic Transactions on Numerical Analysis*, Vol. 45, pp. 371–404 (2016).

[8] Laurie, D. P.: Calculation of Gauss-Kronrod Quadrature Rules, *Mathematics of Computation*, Vol. 66, No. 219, pp. 1133–1145 (1997).

[9] Piessens, R. and Branders, M.: A Note on the Optimal Addition of Abscissas to Quadrature Formulas of Gauss and Lobatto, *Mathematics of Computation*, Vol. 28, No. 125, pp. 135–139 (1974).

[10] Monegato, G.: Some remarks on the construction of extended Gaussian quadrature rules, *Mathematics of Computation*, Vol. 32, No. 141, pp. 247–252 (1978), Kronrod 点の計算法など.

[11] Chawla, M. M.: Error Estimates for the Clenshaw-Curtis Quadrature, *Mathematics of Computation*, Vol. 22, No. 103, pp. 651–656 (1968), https://www.jstor.org/stable/2004542.

[12] Riess, R. D. and Johnson, L. W.: Error estimates for Clenshaw-Curtis quadrature, *Numerische Mathematik*, Vol. 18, pp. 345–353 (1971).

[13] Trefethen, L. N.: Is Gauss Quadrature Better than Clenshaw-Curtis?, *SIAM Review*, Vol. 50, No. 1, pp. 67–87 (2008).

[14] 桂田祐史：Trefethen, Is Gauss Quadrature Better than Clenshaw – Curtis? を読む, https://m-katsurada.sakura.ne.jp/labo/library/numerical-integration/trefethen2008.pdf これは読書ノートなので内輪の資料室に置く。 (2025/5/10).