Maxima by Example:
Ch.8: Numerical Integration [*]

Edwin L. Woollett

November 16, 2012

Contents

[*]This version uses **Maxima 5.28.0**. This is a live document. Check `http://www.csulb.edu/~woollett/` for the latest version of these notes. Send comments and suggestions to `woollett@charter.net`

Preface

Feedback from readers is the best way for this series of notes to become more helpful to new users of Maxima. *All* comments and suggestions for improvements will be appreciated and carefully considered.

```
The Maxima session transcripts were generated using the Xmaxima
graphics interface on a Windows XP computer, and copied into
a fancy verbatim environment in a latex file which uses the
fancyvrb and color packages.
```

```
  Maxima, a Computer Algebra System.
  Some numerical results depend on the Lisp version used.
  This chapter uses Version 5.28.0 (2012) using Lisp GNU
  Common Lisp (GCL) GCL 2.6.8 (aka GCL).
  http://maxima.sourceforge.net/
```

The author would like to thank the Maxima developers for their friendly help via the Maxima mailing list. Richard Fateman provided early encouragement and advice. Raymond Toy, Stavros Macrakis, and Barton Willis have each contributed importantly to the evolution of the code in the **nint** package.

The present **nint** and **apnint** packages are a work in progress, partially dictated by the progress in Maxima special function float and bfloat values. The new Ch. 8 functions **apnint** and **apquad** are the first step in providing quadrature results which attempt to satisfy user requested precision (for one dimensional quadrature).

8.1 Introduction

Chapter 8 on numerical integration (quadrature) is divided into eight sections.

The syntax of the functions **`nint`** and **`quad`** is exactly the same (**`quad`** is used to force use of Quadpack code) and this syntax is introduced with a few examples. These functions are available once you load our **`nint`** package, and they provide an interface to Maxima's **`integrate`** function and the Quadpack functions for numerical quadrature in one and two dimensions.

When you use **`nint`**, an answer is first sought using **`integrate`** (unless the integrand is a multiple-valued function or otherwise potentially dangerous) and only if **`integrate`** does not succeed is Quadpack resorted to.

The new package functions, **`apnint`** and **`apquad`**, available if you both load the **`nint.mac`** package and then load the **`apnint.mac`** package (see examples below), allow the user to request quadrature results with more digits of precision (for one-dimensional quadrature) than is returned by **`nint`** and **`quad`**. Several examples of the use of these functions is given.

We then discuss the most useful Quadpack functions for direct use, their syntax and examples of direct use.

We then discuss the less often used Quadpack functions which may occasionally be useful.

Finally we present a "decision tree" for one dimensional quadrature using the Quadpack functions, first for a finite integration interval, and then for a non-finite integration interval.

In Chapter 9, we will discuss bigfloats and details of using Maxima for high precision quadrature, parts of which are incorporated into the new Ch. 8 functions **`apnint`** and **`apquad`**. Chapter 10 covers the calculation of Fourier series coefficients and both Fourier transform and Laplace transform type integrals. Chapter 11 presents tools for the use of fast Fourier transforms with examples of use.

If you are new to Maxima, check out Ch. 1 of Maxima by Example, and also take a look at the web page:

```
  http://mathscitech.org/articles/maxima
```

as well as the "10 minute tutorial":

```
http://math-blog.com/2007/06/04/
         a-10-minute-tutorial-for-solving-math-problems-with-maxima/
```

and the Harvard Math web page:

```
http://www.math.harvard.edu/computing/maxima/
```

8.2 Using nint and quad for One or Two Dimensional Quadrature

8.2.1 Loading the nint Package

Maxima will be ready to use the **`nint`** package once you have issued a **`load`** command such as **`load(nint)`** (if you have set up your **`maxima.init`** file as suggested in Ch. 1), or, using a complete path, as in **`load("c:/work2/nint.mac")`**.

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
```

The files that are loaded are **nint.mac**, **nint.lisp**, **quad_util.mac**, **mydefint.mac**, **quad1d.mac**, and **quad2d.mac**. Global parameters which are set during this load are: **display2d:false**, **ratprint:false**, **domain:complex**, and **lognegint:true**.

8.2.2 nint and quad Syntax

Here is a summary of the syntax for either **nint** or **quad**.

We use **var** for the variable of integration, and **a** and **b** for the range limits in the 1d examples here (naturally, you can use any symbol for the variable of integration, as long as the supplied expression **expr** is a function of that variable.

For the 2d syntax examples, we assume **expr** is a function of **x** and **y**, with **x1** and **x2** the range limits for the **x** variable, and with **y1** and **y2** the range limits for the **y** variable.

Again, you can use any symbols for the variables of integration.

The range limit parameters are either real numbers or **inf** or **minf**.

1D INTEGRAL SYNTAX

To first try integrate with a fallback to quadpack:

```
        nint ([expr],var,a, b [options])
```

or, to force use of quadpack:

```
        quad ([expr],var,a, b [options])
```

The options: **strong_osc**, **singular**, **principal_val(v0)**, **points(x1,x2,...)**, **real**, **imaginary**, and **complex** are recognised for one dimensional integrals and are used to decide what quadpack routine to use and how.

Before calling the quadpack routines, **nint** (or **quad**) expends some effort in trying to confirm that the integrand is either real or pure imaginary. If **nint** (or **quad**) cannot assume the integrand is either real or pure imaginary (using a series of tests), **nint** (or **quad**) proceeds to work with the real and imaginary parts separately. Although it is never necessary to use the options **real**, **imaginary**, or **complex**, for a complicated integrand some time may be saved if the relevant option is included (thus avoiding the time for the tests). Of course, if you include the option **real**, you should be sure the integrand evaluates to a real number over the whole domain of integration!

The option **strong_osc** forces use of **quad_qag** if integrate does not succeed. The **strong_osc** keyword should only be used for a finite interval of integration.

The option **singular** forces use of **quad_qags** if integrate does not succeed; should only be used for a finite interval of integration.

The option **principal_val(v0)** assumes that **expr** has the form **g(var)/(var - v0)**, and requests that the quadpack routine **quad_qawc** be used for a numerical principal value evaluation (if integrate is not used or is not successful).

The option **points(x1,x2,..)** forces use of **quad_qagp** if **integrate** is not used or is not successful. **quad_qagp** gives special treatment to the supplied points, where the integrand is assumed to have singular behavior of some type. The supplied points should be entirely inside the range of integration.

```
   1d examples ( in each you can use quad instead of nint,
                    which forces use of quadpack routines ):

        nint (expr,var,a,b)
        nint (expr,var,a,b,real)
        nint (expr,var,a,b,imaginary)
        nint (expr,var,a,b,complex)

        nint (expr,var,a,b,strong_osc)
        nint (expr,var,a,b,strong_osc,real)
        nint (expr,var,a,b,strong_osc,imaginary)
        nint (expr,var,a,b,strong_osc,complex)

        nint (expr,var,a,b,singular)
        nint (expr,var,a,b,singular,real)
        nint (expr,var,a,b,singular,imaginary)
        nint (expr,var,a,b,singular,complex)

        nint (expr,var,a,b,principal_val(v0))
                   where v0 evaluates to a number and
                    a < v0 < b.
        nint (expr,var,a,b,principal_val(v0),real)
        nint (expr,var,a,b,principal_val(v0),imaginary)
        nint (expr,var,a,b,principal_val(v0),complex)

        nint (expr,var,a,b, points(x1,x2,...))
           where x1, x2, etc., evaluate to numbers and
                   a < x1 < b, etc.
        nint (expr,var,a,b, points(x1,x2,...),real)
        nint (expr,var,a,b, points(x1,x2,...),imaginary)
        nint (expr,var,a,b, points(x1,x2,...),complex)
```

2D INTEGRAL SYNTAX

We use a 2d notation similar to Mathematica's. `nint(f,[x,x1,x2],[y,y1,y2])` is the approximate numerical value of the double integral `integrate( integrate (f,y,y1,y2), x,x1,x2)`. The 2d quadrature syntax is then:

```
    nint (expr,[x,x1,x2 [x-option]],[y,y1,y2 [y-option]] [gen-option])
```

which includes a try using integrate, or

```
    quad (expr,[x,x1,x2 [x-option]],[y,y1,y2 [y-option]] [gen-option])
```

which uses only the quadpack routines. For example (in each, you can replace **nint** by **quad**):

```
        nint (expr,[x,x1,x2],[y,y1,y2])

        nint (expr,[x,x1,x2],[y,y1,y2],real)

        nint (expr,[x,x1,x2],[y,y1,y2,strong_osc])

        nint (expr,[x,x1,x2,strong_osc],[y,y1,y2],imaginary)

        nint (expr,[x,x1,x2],[y,y1,y2,principal_val(y0)])

        nint (expr,[x,x1,x2],[y,y1,y2,points(ya,yb,...)])
```

GENERAL REMARKS

nint calls **nint1d** for a 1d integral or **nint2d** for a 2d integral. **nint1d** and **nint2d** are responsible for making sure the work detail lists **nargL** and **noutL** are constructed and available as global lists if **integrate** is used and is successful.

Otherwise these lists are constructed by the code in **quad1d** or **quad2d**, which are called if the word **nint** is replaced by **quad**, or which are called if the call to **integrate** (by **nint**) is not sucessful.

nargL is a global list of method and args input, **noutL** is a global list of method and either **integrate** or quadpack total output.

If **method** is set to **true**, **nint** or **quad** will print out the method used during the work.

If **debug** is set to **true**, many details of progress will be printed to the console screen.

8.2.3 1D Quadrature Using mdefint and ndefint

You might want to try using (i.e., force Maxima to use) **integrate**, even though **nint** avoids using **integrate** for a particular integrand.

For 1d integrals, the package provides the function **mdefint** which uses **integrate** (in a careful way) to try to get a symbolic answer. You can then use **cfloat** to reduce the answer to a floating point value. **cfloat** is a package function which combines **rectform**, **float**, and **expand** in one function, with the definition:

```
     cfloat(expr):= (expand(float(rectform(expr))))$
```

However, if the symbolic answer is the difference of two almost equal values, **cfloat** can return an incorrect numerical value. A more careful reduction to a numerical value occurs with the use of the package function **fbfloat (expr, ndigits)** which uses bigfloat methods to reduce **expr** to a floating point value using the requested precision.

The package function **ndefint** combines **mdefint** with **fbfloat** and requests 32 digit accuracy. Some examples of the use of **mdefint** and **ndefint** for 1d quadrature (see below for 2d examples):

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
(%i2) mdefint(log(1/x)/sqrt(x),x,0,1);
(%o2) 4
(%i3) ndefint(log(1/x)/sqrt(x),x,0,1);
(%o3) 4.0
(%i4) mdefint(log(1/x)/sqrt(%i*x),x,0,1);
(%o4) -4*(-1)^(3/4)
(%i5) ndefint(log(1/x)/sqrt(%i*x),x,0,1);
(%o5) 2.82842712474619-2.82842712474619*%i
(%i6) mdefint(log(1/x)/sqrt(%i*x),x,0,1);
(%o6) -4*(-1)^(3/4)
(%i7) cfloat(%);
(%o7) 2.828427124746191-2.828427124746191*%i
(%i8) fbfloat(%o6,32);
(%o8) 2.82842712474619-2.82842712474619*%i
```

8.2.4 1D Quadrature Examples Using nint and quad

We continue with the same integrand used above. Rather than setting **method** to **true**, we can use the global parameter **noutL** to see what actual method(s) were used and perhaps some details. Even if we set **method** to **true**, we can always look at **noutL**; more details are given if quadpack was used.

If no method option is included in the arguments given to **nint** or **quad**, and if quadpack is being used, **quad1d** runs a competition between **quad_qags** and **quad_qag**, and selects the result with the least estimated error magnitude.

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
(%i2) nint(log(1/x)/sqrt(%i*x),x,0,1);
(%o2) 2.82842712474623-2.828427124746158*%i
(%i3) noutL;
(%o3) [[qags,2.82842712474623,3.6770586575585185E-13,315,0],
        [qags,-2.828427124746158,1.1102230246251565E-13,315,0]]
(%i4) quad(log(1/x)/sqrt(%i*x),x,0,1);
(%o4) 2.82842712474623-2.828427124746158*%i
(%i5) noutL;
(%o5) [[qags,2.82842712474623,3.6770586575585185E-13,315,0],
        [qags,-2.828427124746158,1.1102230246251565E-13,315,0]]
(%i6) method:true$
(%i7) nint(log(1/x)/sqrt(%i*x),x,0,1);
   quad_qags
   quad_qags
(%o7) 2.82842712474623-2.828427124746158*%i
```

```
(%i8) quad(log(1/x)/sqrt(%i*x),x,0,1);
   quad_qags
   quad_qags
(%o8) 2.82842712474623-2.828427124746158*%i
(%i9) noutL;
(%o9) [[qags,2.82842712474623,3.6770586575585185E-13,315,0],
        [qags,-2.828427124746158,1.1102230246251565E-13,315,0]]
(%i10) nint(x^2,x,0,2);
 integrate
(%o10) 2.666666666666667
(%i11) mdefint(x^2,x,0,2);
(%o11) 8/3
(%i12) cfloat(%);
(%o12) 2.666666666666667
```

Non-Finite Range Examples

```
(%i13) nint(1/(1+x^2),x,0,inf);
 integrate
(%o13) 1.570796326794897
(%i14) nint(exp (%i*x^2),x,minf,inf);
 integrate
(%o14) 1.2533141373155*%i+1.2533141373155
(%i15) quad(exp (%i*x^2),x,minf,inf);
 quad_qagi
 realpart quad_qagi error =  integration failed to converge
(%o15) false
```

In the last example, **`integrate`** is able to find the result, but quadpack fails. The symbolic answer is:

```
(%i16) mdefint(exp (%i*x^2),x,minf,inf);
(%o16) sqrt(%pi)*(%i/sqrt(2)+1/sqrt(2))
```

Further Examples

Because the integrand contains **`sqrt`**, **`nint`** bypasses **`integrate`** here and calls **`quad1d`**. However, **`integrate`** can actually do this integral and returns the correct numerical value.

```
(%i17) nint(sin(x)/sqrt(x),x,0,5000,real,singular);
 quad_qags
(%o17) 1.251128192999518
(%i18) ndefint(sin(x)/sqrt(x),x,0,5000);
(%o18) 1.251128192999533
```

Here is a bessel function example in which we request the use of **`quad_qag`**:

```
(%i19) nint(bessel_j(0,x)/(1+x),x,0,1,real,strong_osc);
 quad_qag
(%o19) 0.64654342647716
```

Here is a principal value integral done by quadpack:

```
(%i20) quad((x-x^2)^(-1),x,-1/2,1/2,principal_val(0));
 quad_qawc
 quad_qawc
(%o20) 1.09861228866811
```

However, **integrate** can do this principal value integral, and issues the "Principal Value" printout to warn the user:

```
(%i21) ndefint((x-x^2)^(-1),x,-1/2,1/2);
Principal Value
(%o21) 1.09861228866811
```

Here is a quadrature done using the quadpack routine **quad_qagp** by including the **points** option:

```
(%i22) quad(1/sqrt(sin(x)),x,0,10,points(%pi,2*%pi,3*%pi));
 quad_qagp
  quad_qagp
  quad_qagp
(%o22) 10.48823021716681-6.769465521725387*%i
```

integrate cannot find a value for this integral:

```
(%i23) ndefint(1/sqrt(sin(x)),x,0,10);
(%o23) false
(%i24) mdefint(1/sqrt(sin(x)),x,0,10);
(%o24) 'integrate(1/sqrt(sin(x1030868)),x1030868,0,10)
```

8.2.5 2D Quadrature Examples Using nint and quad

The no-option version of 2D syntax also works with **mdefint** and **ndefint**.

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
(%i2) nint(1,[x,0,1],[y,0,1]);
(%o2) 1.0
(%i3) quad(1,[x,0,1],[y,0,1]);
(%o3) 1.0
(%i4) mdefint(1,[x,0,1],[y,0,1]);
(%o4) 1
(%i5) ndefint(1,[x,0,1],[y,0,1]);
(%o5) 1.0
```

and here are uses of the option **real**:

```
(%i6) nint(1/sqrt(x+y),[x,0,1],[y,0,1],real);
(%o6) 1.104569499660576
(%i7) nint(sin(x*y),[x,0,1],[y,0,1],real);
(%o7) 0.23981174200056
```

A 2d example in which the presence of **log** and also fractional powers causes **integrate** to be bypassed in favor of quadpack (set **method** to **true** to see easily what method is being used here):

```
(%i8) method:true$
(%i9) nint(log(y)/x^(4/5),[x,0,1],[y,0,1]);
  qags2
(%o9) -5.000000000000004
(%i10) time(%);
(%o10) [6.23]
(%i11) nint(log(y)/x^(4/5),[x,0,1],[y,0,1],real);
  qags2
(%o11) -5.000000000000004
(%i12) time(%);
(%o12) [3.78]
(%i13) ndefint(log(y)/x^(4/5),[x,0,1],[y,0,1]);
(%o13) -5.0
(%i14) time(%);
(%o14) [0.02]
```

This is a case in which including the option **real** halves the time of the quadpack calculation. This integrand-domain combination is correctly evaluated by **mdefint** in a fraction of the time required by quadpack.

```
(%i15) mdefint(log(y)/x^(4/5),[x,0,1],[y,0,1]);
(%o15) -5
(%i16) time(%);
(%o16) [0.0]
```

2d examples with non-finite limits:

```
(%i17) nint(exp(-abs(x) - abs(y)),[x,minf,inf],[y,minf,inf]);
 integrate
(%o17) 4.0
(%i18) quad(exp(-abs(x) - abs(y)),[x,minf,inf],[y,minf,inf]);
 qagi(qagi(..))
(%o18) 3.999999999996149
(%i19) nint(sin(x*y)*exp(-x)*exp(-y),[x,0,inf],[y,0,inf],real);
 qagi(qagi(..))
(%o19) 0.34337796296064
```

8.2.6 The nint Package Sets domain to complex: Caution!

When you load in the **nint** package, **domain** is set to **complex**, which can result in an incorrect symbolic integral result or an error when you directly use **integrate**.

Here is a 2d example, in which **integrate** gives the correct answer only if **domain** is set to **real** (the normal default in the absence of the package **nint.mac**). (This is a Maxima bug.)

If you are only interested in the numerical value (for a particular value of the parameter **b**), and use our package functions **nint** or **quad**, there is no problem, because the **sqrt** function in the integrand causes **nint** to bypass **integrate** automatically.

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
(%i2) domain;
(%o2) complex
```

```
(%i3) foo :((-%pi)/2+acos(y/b)+acos(x/b))/(2*%pi)$
(%i4) assume(b>0,x>0, x<b);
(%o4) [b > 0,x > 0,b > x]
(%i5) integrate(integrate(foo,y,0,sqrt(b^2 - x^2)),x,0,b);
expt: undefined: 0 to a negative exponent.
 -- an error. To debug this try: debugmode(true);
(%i6) domain:real;
(%o6) real
(%i7) integrate(integrate(foo,y,0,sqrt(b^2 - x^2)),x,0,b);
(%o7) b^2/(4*%pi)
```

8.2.7 Case: Symbolic Definite Integral the Difference of Two Almost Equal Numbers

Here we compare the direct use of **integrate**, **nint**, and **quad** with an integrand whose definite integral is the difference of two almost equal numbers.

Use of **`cfloat`** (an **`nint`** package function which combines **`rectform`**, **`float`**, and **`expand`**) produces a wildly wrong numerical answer. Use of the package function **`fbfloat`**, and choice of 32 digit precision, uses bigfloat methods to get an accurate answer, which is then converted back to 16 digit display inside the function **`fbfloat`**.

The package function **`fchop`** is used routinely in the package to ignore very small numbers.

The package function **`nint`** tries **`integrate`** first, and if successful then uses **`fbfloat`** with 32 digit precision requested to obtain the numerical value from the symbolic definite integral, and then uses **`fchop`**.

The package function **`quad`** always uses **`fchop`** on the result returned by quadpack to round off the numerical answer. (But see below about turning off the chopping of very small numbers.)

It is clear, from the nature of the integrand, that the answer must be a real number.

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
(%i2) ee : integrate(exp(x^5),x,1,2);
(%o2) ((-1)^(4/5)*gamma_incomplete(1/5,-32)-(-1)^(4/5)
                                          *gamma_incomplete(1/5,-1))/5
(%i3) cfloat(ee);
(%o3) 1.0132394896940175E+12-1.9531250000000001E-4*%i
(%i4) fbfloat(ee,32);
(%o4) 1.0512336862202133E-20*%i+1.0132394896940183E+12
(%i5) fchop(%);
(%o5) 1.0132394896940183E+12
(%i6) nint(exp(x^5),x,1,2);
(%o6) 1.0132394896940183E+12
(%i7) noutL;
(%o7) [integrate,1.0132394896940183E+12]
(%i8) quad(exp(x^5),x,1,2);
(%o8) 1.0132394896940144E+12
(%i9) noutL;
(%o9) [qag,1.0132394896940144E+12,0.011249218166387,155,0]
```

In the above output, the global parameter **`noutL`** includes the method and the result(s) returned by the method.

The global parameter **nargL** is a list which includes both the method and the arguments used by the method. The names of the integration variables actually used are automatically generated by the code to allow global assumptions to remain in force after the computation, hence the weird names.

```
(%i10) nint(exp(x^5),x,1,2);
(%o10) 1.013239489694018E+12
(%i11) nargL;
(%o11) [integrate,%e^wx62375^5,wx62375,1,2]
(%i12) quad(exp(x^5),x,1,2);
(%o12) 1.013239489694014E+12
(%i13) nargL;
(%o13) [qag,%e^x76014^5,x76014,1.0,2.0,3,limit = 800]
```

8.2.8 Turning Off Automatic Chop of Answer

The global parameter **dochop** is set to **true** by the package and causes automatic chopping of the answer. Set **dochop** to false to prevent automatic chopping of the answer.

```
(%i1) load(nint);
(%o1) "c:/work2/nint.mac"
(%i2) dochop;
(%o2) true
(%i3) nint(exp(x^5),x,1,2);
(%o3) 1.013239489694018E+12
(%i4) dochop:false$
(%i5) nint(exp(x^5),x,1,2);
(%o5) 1.051233686220213E-20*%i+1.013239489694018E+12
(%i6) dochop:true$
(%i7) nint(exp(x^5),x,1,2);
(%o7) 1.013239489694018E+12
```

8.2.9 Changing the Chop Value Used by nint

The global parameter **_small%** holds the small chop value; numbers whose absolute values are smaller than **_small%** are automatically "chopped" by **fchop**. Set **_small%** to smaller values if you want less chopping.

```
(%i8) method : false$
(%i9) nint(exp(x^5),x,1,2);
(%o9) 1.013239489694018E+12
(%i10) _small%;
(%o10) 1.0E-14
(%i11) _small% : 1.0e-30$
(%i12) nint(exp(x^5),x,1,2);
(%o12) 1.051233686220213E-20*%i+1.013239489694018E+12
```

8.3 Arbitrary Precision One Dimensional Quadrature with apnint and apquad

Syntax

The new Ch. 8 functions **apnint** (arbitrary precision numerical integration which first tries integrate) and **apquad** (arbitrary precision quadrature which avoids integrate) are defined in the file **apnint.mac** (which loads **tsquad.mac** and **dequad.mac**). To use **apnint** and/or **apquad**, you should first load the **nint.mac** package, and then the **apnint.mac** package.

Both functions have the same syntax:

```
    apnint (expr, x, x1,x2,rp,wp)
    apquad (expr, x, x1,x2,rp,wp)
```

in which **expr** depends on the variable **x** (of course you can use any symbol), **x1** is less than **x2**, **x1** must evaluate to a finite real number, but **x2** can either be finite (evaluating to a real number) or the symbol **inf**.

rp is the "requested precision" ("precision goal", the number of trustworthy digits wanted in the result), and **wp** is the "working precision" (**fpprec** is set to **wp** during the course of the calculation).

These two functions do not set values of **noutL** and **nargL**, but you can set **method** to **true** to have a short printout of the method used. The finite numerical case uses a new function **tsquad** (which accepts a complex integrand), defined in the file **tsquad.mac**.

The non-finite numerical case uses a new function **dequad** (which also accepts a complex integrand), defined in the file **dequad.mac**.

Both of these new functions return the result as a bigfloat number. In general, **bfloat(expr)** returns a bigfloat number whose precision depends on the current value of **fpprec**. Both package functions **apnint** and **apquad** worry about setting the value of **fpprec** locally to the values needed corresponding to the **wp** arg. You do not have to set the global value of **fpprec** yourself, although that global value will not affect the precision of the calculation: only the supplied value **wp** affects the precision of the actual calculation.

Here are some examples of bigfloat number returns (the default value of **fpprec** is **16**). When comparing two bigfloat numbers, you should set the value of **fpprec** to a large enough value. Using **float** to convert a bigfloat number to an ordinary floating point number leads to a loss of significant digits. (See Ch. 9 for more discussion of bigfloats.)

```
(%i1) fpprec;
(%o1) 16
(%i2) pi16 : bfloat(%pi);
(%o2) 3.141592653589793b0
(%i3) pi30 : bfloat(%pi),fpprec:30;
(%o3) 3.14159265358979323846264338328b0
(%i4) fpprec;
(%o4) 16
(%i5) abs(pi16 - pi30);
(%o5) 0.0b0
(%i6) abs(pi16 - pi30),fpprec:40;
(%o6) 1.144237745221949411169021904042519539695b-17
(%i7) pi30f : float(pi30);
(%o7) 3.141592653589793
```

```
(%i8) abs(pi16 - pi30f),fpprec:40;
(%o8) 1.1102230246251565404236316680908203125b-16
```

An Accuracy Test

A test used in Ch.9 is the known integral

$$\int_0^1 \sqrt{t}\,\ln(t)\,dt = -4/9 \qquad (8.1)$$

You need to load both the **nint.mac** package, and then the **apnint** package (the latter loads **tsquad.mac** and **dequad.mac**).

```
(%i1) (load(nint),load(apnint));
   _kmax% =  8   _epsfac% =  2
(%o1) "c:/work2/apnint.mac"
(%i2) integrate(sqrt(x)*log(x),x,0,1);
(%o2) -4/9
(%i3) tval : bfloat(%),fpprec:45;
(%o3) -4.44444444444444444444444444444444444444444445b-1
(%i4) apval : apquad(sqrt(x)*log(x),x,0,1,30,40);
 construct _yw%[kk,fpprec] array for kk =
  8  and fpprec =  40   ...working...
(%o4) -4.444444444444444444444444444444444444445b-1
(%i5) abs(apval - tval),fpprec:45;
(%o5) 4.46422161518439410412204461320543256312937649b-41
```

Requesting thirty digit accuracy with forty digit arithmetic returns a value for this integral which has about forty digit precision. See Ch.9 for other tests using known integrals.

The loading of the **apnint.mac** file causes the loading of other files of the package. The loading of the package file **tsquad.mac** produces the printout: **_kmax% = 8   _epsfac% = 2** seen in the above example, which warns that two global parameters are set by the package. Ch.9 discusses the meaning of those parameters, and the possibility of changing them to fit your special problem.

An array of transformation coefficients is constructed, using 40 digit arithmetic, in the above example. Once that array has been constructed, other numerical integrals can be evaluated using the same set of coefficients (as long as the same working precision **wp** is requested).

If you change the value of **wp**, a new array of coefficients is constructed, which is available for use with other integrals (the original 40 digit array is still available also).

```
(%i6) apquad(sqrt(x)*log(x),x,0,1,20,30);
 construct _yw%[kk,fpprec] array for kk =
  8  and fpprec =  30   ...working...
(%o6) -4.44444444444444444444444444445b-1
(%i7) apquad(sqrt(x)*log(x),x,0,1,30,40);
(%o7) -4.444444444444444444444444444444444444445b-1
```

Some Examples

Using **apquad** forces use of a numerical method discussed in Ch.9, instead of first trying **integrate**. The **expr** to be integrated can be complex.

```
(%i8) apquad(sin(x)*exp(%i*x),x,0,2,20,30);
(%o8) 1.189200623826982062843159773363b0*%i+4.134109052159029786597920457744b-1
```

However this integral can be done using **integrate**, and Maxima is successful in obtaining a **bfloat** value of the symbolic result:

```
(%i9) method:true$
(%i10) apnint(sin(x)*exp(%i*x),x,0,2,20,30);
 integrate
(%o10) 1.189200623826982062843159773363b0*%i+4.134109052159029786597920457744b-1
```

Here is an example of a non-finite integral with a complex integrand. We force use of a numerical method by using **apquad**:

```
(%i11) apquad(exp(-x +%i*x),x,0,inf,20,30);
dequad
(%o11) 5.0b-1*%i+5.0b-1
```

The package function **dequad** is defined in the file **dequad.mac**, loaded by **apnint.mac**, and can handle a complex integrand, passing the real and imaginary parts separately to the Ch.9 function **quad_de** (the latter assumes a real integrand).

As in the previous example, using **apnint** produces an answer from **integrate**:

```
(%i12) apnint(exp(-x +%i*x),x,0,inf,20,30);
 integrate
(%o12) 5.0b-1*%i+5.0b-1
```

Here is an example of an integrand containing a Bessel function, and **integrate** cannot find the definite integral over the non-finite domain, nor can **integrate** find the indefinite integral:

```
(%i13) integrate(bessel_j(1,x)*exp(-x),x,0,inf);
(%o13) 'integrate(bessel_j(1,x)*%e^-x,x,0,inf)
(%i14) integrate(bessel_j(1,x)*exp(-x),x);
(%o14) 'integrate(bessel_j(1,x)*%e^-x,x)
```

If we force the use of the numerical bigfloat methods, the integrand is first tested to see if evaluation at a bigfloat specified point returns a bigfloat number, and for this example the Bessel function factor is not reduced:

```
(%i15) bfloat(bessel_j(1,0.5b0)*exp(-0.5b0));
(%o15) 6.065306597126334b-1*bessel_j(1.0b0,5.0b-1)
```

and **apquad** returns **false**.

```
(%i16) apquad(bessel_j(1,x)*exp(-x),x,0,inf,20,30);
apquad: cannot obtain bfloat value from integrand
(%o16) false
```

The presence of **sqrt** and/or **log** in an integrand causes **apnint** to bypass **integrate**, (just as happens with **nint**).

```
(%i17) apnint(log(1/x)/sqrt(%i*x),x,0,1,20,30);
tsquad
(%o17) 2.828427124746190097603377448420b0-2.828427124746190097603377448420b0*%i
```

Here is an example of a complex integrand in which **quad** can get a numerical answer (about 15 digit precision), but **apquad** can only get an answer for the **realpart** of the integrand:

```
(%i18) quad(log(-3+%i*x),x,-2,3);
   quad_qag
   quad_qags
(%o18) 2.449536971144524*%i+6.02070929514083
(%i19) quad(realpart(log(-3+%i*x)),x,-2,3);
   quad_qag
(%o19) 6.02070929514083
(%i20) quad(imagpart(log(-3+%i*x)),x,-2,3);
   quad_qags
(%o20) 2.449536971144524
(%i21) apnint(realpart(log(-3+%i*x)),x,-2,3,20,30);
tsquad
(%o21) 6.020709295140831356948887113887b0
(%i22) apnint(imagpart(log(-3+%i*x)),x,-2,3,20,30);
quad_ts: vdiffnew > vdiffold before vdiff < eps0 reached
quad_ts: abort calc.
(%o22) false
(%i23) imagpart(log(-3+%i*x));
(%o23) atan2(x,-3)
```

The bigfloat calculation for the imaginary part of the integrand was aborted when the differences between the current estimate and the previous estimate of the integral started growing (instead of decreasing steadily).

8.4 Using the Wolfram Alpha Web Site for Integrals

The Wolfram Alpha web page, **http://www.wolframalpha.com**, allows free one-line commands (integrals, derivatives, plots, etc.,) which may require translating Maxima syntax into Mathematica syntax.

A web page with some translation from Maxima to Mathematica is

```
  http://www.math.harvard.edu/computing/maxima/
```

A larger comparison of Maxima, Maple, and Mathematica syntax is at

```
http://beige.ucs.indiana.edu/P573/node35.html
```

Maxima syntax for 1d symbolic integration is

```
(%i13) integrate(cos(x),x,0,1);
(%o13) sin(1)
(%i14) float(%);
(%o14) 0.8414709848079
```

The corresponding Mathematica request would be (note the curly brackets):

```
Integrate[Cos[x], {x, 0, 1}]
```

for a symbolic answer, and

```
 NIntegrate[Cos[x], {x, 0, 1}]
```

produces a numerical result corresponding to Maxima's

```
float(integrate(cos(x),x,0,1))
```

Mathematica also has the function **N** which computes the numerical value of **expr** with the syntax:

```
N[ expr ]
```

or

```
 expr //N
```

Two examples of numerical two dimensional integration in Mathematica syntax are:

```
 NIntegrate[1/Sqrt[x + y], {x,0,1},{y,0,1} ]
 NIntegrate[Sin[ x*y ],{x,0,1}, {y,0,1} ]
```

8.5 Direct Quadpack Use of quad_qags, quad_qag, and quad_qagi

The quadpack subroutine package originally was a set of Fortran subroutines, described in the book:

```
Quadpack: A Subroutine Package for Automatic Integration,
by R. Piessens, E. de Doncker-Kapenga, C.W. Uberhuber,
     and D.K. Kahaner,
Springer-Verlag, 1983
```

and designed for the computation of 1d integrals. The Maxima developer Raymond Toy has adapted the package to Maxima via a translation from Fortran to Common Lisp (using f2cl) combined with a special interface for the Maxima user.

See the Maxima Help Manual for an extensive discussion in Sec. 19.3 (Sec. 19 is Integration).

8.5.1 Syntax for Quadpack Functions

The integrand **expr** supplied as the first argument to a quadpack function must evaluate to a real number throughout the requested range of the independent variable. All the quadpack functions ("q" is for "quadrature"), except two, are called using the syntax:

```
   quad_qaxx ( expr, var, a, b, (other-required-args), optional-args)
```

The exceptions are the Cauchy principle value integration routine **quad_qawc** which does the integral $\int_a^b f(x)/(x-c)\,dx$, using the syntax `quad_qawc(expr,var,c,a,b,optional-args)`, and the cosine or sine Fourier transform integration routine **quad_qawf** which does integrals $\int_a^\infty f(x)\cos(\omega\,x)\,dx$ or $\int_a^\infty f(x)\sin(\omega\,x)\,dx$ using the syntax `quad_qawf(expr,var,a,omega,trig,optional-args)`.

8.5.2 Ouput List of Quadpack Functions and Error Code Values

All Quadpack functions return a **list** of four elements:

```
[num-val, est-abs-error, number-integrand-evaluations, error-code].
```

The **error-code** (the fourth element of the returned list) can have the values:

0 - no problems were encountered

1 - too many sub-intervals were done

2 - excessive roundoff error is detected

3 - extremely bad integrand behavior occurs

4 - failed to converge

5 - integral is probably divergent or slowly convergent

6 - the input is invalid

8.5.3 Integration Rule Parameters and Optional Arguments

All the Quadpack functions (except one) include **epsrel**, **epsabs**, and **limit** as possible optional types of allowed arguments to override the default values of these parameters: **epsrel = 1e-8, epsabs = 0.0, limit = 200**.
To override the **epsrel** default value, for example, you would add the optional argument **epsrel = 1e-6** or **epsrel=1d-6** (both have the same effect).

These Quadpack functions apply an integration rule adaptively until an estimate of the integral of **expr** over the interval **(a, b)** is achieved within the desired absolute and relative error limits, **epsabs** and **epsrel**. With the default values **epsrel = 1e-8, epsabs = 0.0**, only **epsrel** plays a role in determining the convergence of the integration rules used, and this corresponds to getting the estimated relative error of the returned answer smaller than **epsrel**.

If you override the defaults with the two optional arguments (in any order) **epsrel = 0.0, epsabs = 1e-8**, for example, the value of **epsrel** will be ignored and convergence will have been achieved if the estimated absolute error is less than **epsabs**.

The integration region is divided into subintervals, and on each iteration the subinterval with the largest estimated error is bisected. This "adaptive method" reduces the overall error rapidly, as the subintervals become concentrated around local difficulties in the integrand.

The Quadpack parameter **limit**, whose default value is **200**, is the maximum number of subintervals to be used in seeking a convergent answer. To increase that limit, you would insert **limit=300**, for example.

8.5.4 quad_qags for a Finite Interval

The "s" on the end of **qags** is a signal that **quad_qags** has extra abilities in dealing with functions which have integrable **singularities**, but even if your function has no singular behavior, **quad_qags** is still the best choice due to the sophistication of the quadrature algorithm used. Use the syntax

```
quad_qags ( expr, var , a, b, [ epsrel, epsabs, limit ] )
```

where the keywords inside the brackets indicate possible optional arguments (entered in any order), such as **epsrel = 1e-6**.

Thus the approximate numerical value of $\int_0^1 e^{x^2}\,dx$ would be the first element of the list returned by **quad_qags (exp (x^2), x, 0, 1)**.

```
(%i1) display2d:false;
(%o1) false
(%i2) domain;
(%o2) real
(%i3) quad_qags(exp (x^2),x,0,1);
(%o3) [1.462651745907182,1.6238696453143376E-14,21,0]
```

If you call **quad_qags** with an unbound parameter in the integrand, a noun form will be returned which will tell you all the defaults being used.

```
(%i4) quad_qags(a*x,x,0,1);
(%o4) quad_qags(a*x,x,0,1,epsrel = 1.0E-8,epsabs = 0.0,limit = 200)
```

There is a more efficient Quadpack function available, **quad_qaws**, for integrals which have end point algebraic and/or logarithmic singularites associated with of the form $\int_a^b f(x)\,w(x)\,dx$, if the "weight function" $w(x)$ has the form (note: $f(x)$ is assumed to be well behaved)

$$w(x) = (x-a)^\alpha\,(b-x)^\beta \ln(x-a)\ln(b-x) \tag{8.2}$$

where $\alpha > -1$ and $\beta > -1$ for convergence. See Sec 8.6.3 for more information on **quad_qaws** and its use.

Example 1

Here is an example of the use of **quad_qags** to compute the numerical value of the integral $\int_0^1 \sqrt{x}\ln(1/x)\,dx$. The integrand $g = x^{1/2}\ln(1/x) = -x^{1/2}\ln(x)$ has the limiting value 0 as $x \to 0$ from the positive side. Thus the integrand is not singular at $x = 0$, but it is rapidly changing near $x = 0$ so an efficient algorithm is needed. (Our Example 2 will work with an integrand which is singular at $x = 0$.)

Let's check the limit at $x = 0$ and plot the function.

```
(%i5) g:sqrt(x)*log(1/x)$
(%i6) limit(g,x,0,plus);
(%o6) 0
(%i7) (load(draw),load(qdraw))$
(%i8) qdraw( ex(g,x,0,1) )$
```

Here is that plot.

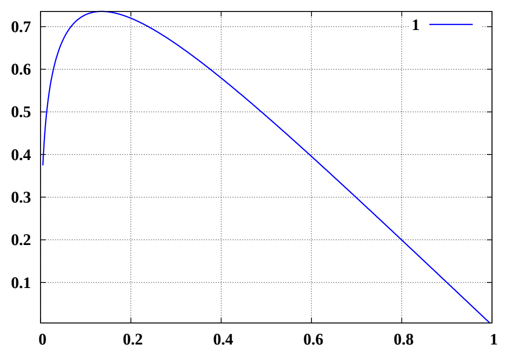

Figure 1: $\mathbf{x^{1/2}\ln(1/x)}$

Now we try out **quad_qags** on this integral and compare with the exact answer returned by **integrate**, converted to 20 digit accuracy using **bfloat**.

```
(%i9) fpprintprec:8$
(%i10) tval : block([fpprec:20], bfloat(integrate(g,x,0,1)) );
(%o10) 4.4444444b-1
(%i11) qlist:quad_qags(g,x,0,1 );
(%o11) [0.444444,4.93432455E-16,315,0]
(%i12) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o12) 8.0182679b-17
```

The first element of the returned list **qlist** is the approximate numerical value of the integral. We have used **integrate** together with **bfloat** and **fpprec** to generate the "true value" good to about **20** digits (see Chapter 9), and we see that the absolute error of the answer returned by **quad_qags** is about $\mathbf{10^{-16}}$. The second element of **qlist** is the **estimated** absolute error of the returned answer. The third element shows that **315** integrand evaluations were needed to attain the requested (default) relative error **epsrel = 1e-8**. The fourth element is the error code value **0** which indicates no problems were found.

The algorithm used to find an approximation for this integral does not know the "exact" (or true) answer, but does have (at each stage) an estimate of the answer and an estimate of the likely error of this estimated answer (gotten by comparing the new answer with the old answer, for example), and so can compute an estimated relative error (est_abs_err/est_answer) which the code checks against the requested relative error goal supplied by the parameter **epsrel**.

We are assuming that the defaults are being used, and hence **epsabs** has the value **0.0**, so that the convergence criterion used by the code is

$$\mathbf{est\_rel\_err \le epsrel} \tag{8.3}$$

or

$$\mathbf{\frac{est\_abs\_err}{est\_answer} \le epsrel} \tag{8.4}$$

or

$$\mathbf{est\_abs\_err \le epsrel \times est\_answer} \tag{8.5}$$

We can check that the values returned by **`quad_qags`** are at least consistent with the advertised convergence criterion.

```
(%i13) est_answer : first(qlist);
(%o13)                          0.444444
(%i14) est_abs_err : second(qlist);
(%o14)                       4.93432455E-16
(%i15) est_rel_err : est_abs_err/est_answer;
(%o15)                       1.11022302E-15
```

We see that the **`est_rel_err`** is much less than **epsrel**.

Example 2

Here we use **`quad_qags`** with an integrand which is singular at $x = 0$, but the singularity is an "integrable singularity". We want to calculate $\int_0^1 \ln(\sin(x))\, dx$. **integrate** returns an "exact" answer involving the dilogarithm function with a complex argument (see below).

We first show that the integrand is singular at $x = 0$, first using Maxima's **limit** function and then using **taylor**. We then use the approximate integrand for small x to show that the indefinite integral is a function of x which has a finite limit at $x = 0$.

```
(%i1) display2d:false$
(%i2) limit(log(sin(x)),x,0,plus);
(%o2) minf
(%i3) ix : taylor(log(sin(x)),x,0,2);
(%o3) +log(x)-x^2/6
(%i4) int_ix:integrate(ix,x);
(%o4) x*log(x)-x^3/18-x
(%i5) limit(int_ix,x,0,plus);
(%o5) 0
(%i6) assume(eps>0,eps<1)$
(%i7) integrate(ix,x,0,eps);
(%o7) (18*eps*log(eps)-eps^3-18*eps)/18
(%i8) expand(%);
(%o8) eps*log(eps)-eps^3/18-eps
(%i9) limit(eps*log(eps),eps,0,plus);
(%o9) 0
```

Output **%o2** indicates that the integrand $\rightarrow -\infty$ as $x \rightarrow 0^+$. To see if this is an "integrable singularity", we examine the integrand for x positive but close to 0, using a Taylor series expansion. We let **eps** represent ϵ, a small positive number to be used as the upper limit of a small integration interval.

The approximate integrand, **ix**, shows a logarithmic singularity at $x = 0$. The indefinite integral of the approximate integrand, **int_ix**, is finite as $x \rightarrow 0^+$. And finally, the integral of the approximate integrand over $(0, \epsilon)$ is finite. Hence we are dealing with an integrable singularity, and we use **`quad_qags`** to evaluate the numerical value of the exact integrand over $[0, 1]$.

```
(%i10) quad_qags(log(sin(x)),x,0,1);
(%o10) [-1.056720205991585,1.173195103278496E-15,231,0]
```

Use of **integrate** with this integrand-domain returns an expression involving the dilogarithm function **li[2](z)** in which **z** is a complex number. At present, Maxima cannot find numerical values of the dilogarithm function for non-real arguments.

```
(%i11) float(li[2](1));
(%o11) 1.644934066848226
(%i12) float(li[2](1+%i));
(%o12) li[2](%i+1.0)
(%i13) integrate(log(sin(x)),x,0,1);
(%o13) -%i*atan(sin(1)/(cos(1)+1))-%i*atan(sin(1)/(cos(1)-1))+log(sin(1))
                                  -log(2*cos(1)+2)/2-log(2-2*cos(1))/2
                                  +%i*li[2](%e^%i)+%i*li[2](-%e^%i)
                                  -%i*%pi^2/12+%i/2
```

8.5.5 quad_qags for Double Integrals

In Sec.7.6 of Ch.7 we presented the notation (for an exact symbolic double integral):

To evaluate the exact symbolic double integral

$$\int_{u1}^{u2} du \int_{v1(u)}^{v2(u)} dv\, f(u,v) \equiv \int_{u1}^{u2} \left(\int_{v1(u)}^{v2(u)} f(u,v)\, dv \right) du \tag{8.6}$$

we use **integrate** with the syntax:

```
integrate( integrate( f(u,v), v, v1(u), v2(u) ), u, u1, u2 )
```

in which **f(u,v)** can either be an expression depending on the variables **u** and **v**, or a Maxima function, and likewise **v1(u)** and **v2(u)** can either be expressions depending on **u** or Maxima functions.

Both **u** and **v** are "dummy variables", since the value of the resulting double integral does not depend on our choice of symbols for the integration variables; we could just as well use **x** and **y**.

To use **quad_qags** we use

```
quad_qags ( quad_qags ( f(u,v), v, v1(u), v2(u) )[1], u, u1, u2 )
```

Example 1

For our first example, we use **quad_qags** on the double integral:

$$\int_1^3 \left(\int_0^{x/2} \frac{x\,y}{x+y}\, dy \right) dx \tag{8.7}$$

comparing with the **integrate** result, first using the absolute error, then the relative error.

```
(%i1) (fpprintprec:8,display2d:false)$
(%i2) g : x*y/(x+y)$
(%i3) tval : (block [fpprec:20],bfloat (integrate (integrate (g,y,0,x/2),x,1,3) ));
Is  x  positive, negative, or zero?
p;
(%o3) 8.1930239b-1
```

```
(%i4) quad_qags ( quad_qags (g,y,0,x/2)[1],x,1,3);
(%o4) [0.819302,9.09608385E-15,21,0]
(%i5) block([fpprec:20], bfloat( abs (%[1] - tval)));
(%o5) 6.1962154b-17
(%i6) %/tval;
(%o6) 7.5627942b-17
```

Bear in mind that the quadpack functions (at present) have, at most, 16 digit accuracy. For a difficult numerical integral, the accuracy can be much less than that.

Example 2

For our second example, we consider the double integral

$$\int_0^1 \left(\int_1^{2+x} e^{x-y^2}\, dy \right) dx \tag{8.8}$$

```
(%i7) g : exp(x-y^2)$
(%i8) tval : block([fpprec:20],bfloat(integrate( integrate(g,y,1,2+x),x,0,1)));
(%o8) 2.3846836b-1
(%i9) quad_qags( quad_qags(g,y,1,2+x)[1],x,0,1);
(%o9) [0.238468,2.64753066E-15,21,0]
(%i10) block([fpprec:20],bfloat(abs (%[1] - tval)));
(%o10) 4.229659b-18
(%i11) %/tval;
(%o11) 1.7736772b-17
```

The Maxima coordinator, Robert Dodier has warned about the lack of accuracy diagnostics with the inner integral done by **quad_qags**:

> A nested numerical integral like this has a couple of drawbacks, whatever the method. (1) The estimated error in the inner integral isn't taken into account in the error estimate for the outer. (2) Methods specifically devised for multi-dimensional integrals are typically more efficient than repeated 1-d integrals.

8.5.6 quad_qag for a General Oscillatory Integrand

The function **`quad_qag`** is useful primarily due to its ability to deal with functions with some general oscillatory behavior.

The "key" feature to watch out for is the required fifth slot argument which the manual calls "key". This slot can have any integral value between 1 and 6 inclusive, with the higher values corresponding to "higer order Gauss-Kronrod" integration rules for more complicated oscillatory behavior.

Use the syntax:

```
quad_qag( expr, var, a, b, key, [epsrel, epsabs, limit] )
```

Thus the approximate numerical value of $\int_0^1 e^{x^2}\, dx$ would be the first element of the list returned by **`quad_qag ( exp ( x^2 ), x, 0, 1, 3)`** using the "key" value **3**.

Example 1

Since the main feature of **quad_qag** is the ability to select a high order quadrature method for a generally oscillating integrand, we begin by comparing **quad_qag** with **quad_qags** for such a case. We consider the integral $\int_0^1 \cos(50\,x)\,\sin(3\,x)\,e^{-x}\,dx$.

```
(%i1) (fpprintprec:8,display2d:false)$
(%i2) f : cos(50*x)*sin(3*x)*exp(-x)$
(%i3) tval : block ([fpprec:20],bfloat( integrate (f,x,0,1) ));
(%o3) -1.9145466b-3
(%i4) (load(draw),load(qdraw))$
(%i5) qdraw( ex(f,x,0,1) )$
```

Here is that plot: and here we make the comparison.

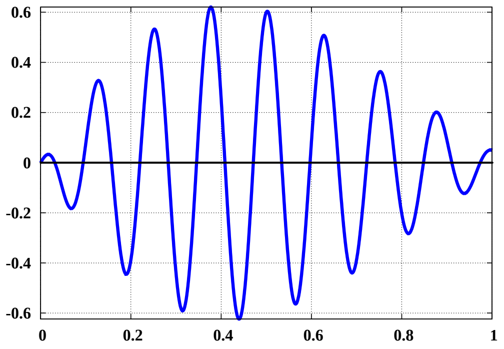

Figure 2: $\cos(50\,x)\,\sin(3\,x)\,e^{-x}$

```
(%i6) quad_qag(f,x,0,1,6);
(%o6) [-0.00191455,2.86107558E-15,61,0]
(%i7) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o7) 9.8573614b-17
(%i8) %/tval;
(%o8) -5.148666b-14
(%i9) quad_qags(f,x,0,1 );
(%o9) [-0.00191455,2.8536E-15,315,0]
(%i10) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o10) 1.1804997b-16
(%i11) %/tval;
(%o11) -6.1659493b-14
```

We see that both methods returned results with about the same relative error, but that **quad_qag** needed only about one fifth the number of integrand evaluations as compared with **quad_qags**. This extra efficiency of **quad_qag** for this type of integrand may be of interest in certain kinds of intensive numerical work. Naturally, the number of integrand evaluations needed does not necessarily translate simply into time saved, so timing trials would be appropriate when considering this option.

Example 2

We compare **quad_qag** and **quad_qags** with the numerical evaluation of the integral $\int_0^1 e^x\, dx$, which has neither special oscillatory behavior (which **quad_qag** might help with) nor singular behavior (which **quad_qags** might help with).

```
(%i12) tval : block ([fpprec:20], bfloat( integrate (exp(x),x,0,1) ));
(%o12) 1.7182818b0
(%i13) quad_qag(exp(x),x,0,1,6);
(%o13) [1.7182818,1.90767605E-14,61,0]
(%i14) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o14) 1.445648b-16
(%i15) %/tval;
(%o15) 8.413335b-17
(%i16) quad_qags(exp(x),x,0,1);
(%o16) [1.7182818,1.90767605E-14,21,0]
(%i17) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o17) 7.7479797b-17
(%i18) %/tval;
(%o18) 4.5091437b-17
```

We see that using **quad_qags** for this function results in slightly smaller relative error while using only two thirds the number of integrand evaluations compared with **quad_qag**.

Example 3

We compare **quad_qag** and **quad_qags** with the numerical evaluation of the integral $\int_0^1 \sqrt{x}\, \ln(x)\, dx$, which has quasi-singular behavior at $x = 0$. See Sec.(8.5.4) for a plot of this integrand.

```
(%i19) f : sqrt(x)*log(1/x)$
(%i20) tval : block ([fpprec:20], bfloat( integrate (f,x,0,1) ));
(%o20) 4.4444444b-1
(%i21) quad_qag(f,x,0,1,3);
(%o21) [0.444444,3.17009685E-9,961,0]
(%i22) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o22) 4.7663293b-12
(%i23) %/tval;
(%o23) 1.072424b-11
(%i24) quad_qags(f,x,0,1);
(%o24) [0.444444,4.93432455E-16,315,0]
(%i25) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o25) 8.0182679b-17
(%i26) %/tval;
(%o26) 1.8041102b-16
```

We see that using **quad_qags** for this function (which has a quasi-singular behavior near $x = 0$) returns a much smaller relative error while using only one third the number of integrand evaluations compared with **quad_qag**.

You can use **quad_qag** for numerical double integrals, following the pattern we used with **qaud_qags**, and you can use different methods for each of the two axes.

8.5.7 quad_qagi for a Non-finite Interval

See Sec.(8.8) for a decision tree for quadrature over a non-finite interval.

The syntax is

```
   quad_qagi ( expr, var, a, b, [epsrel, epsabs, limit] ), where
           (a,b) are the limits of integration.
 Thus  you will have (omitting the optional args):
 quad_qagi (expr, var, minf, b ) with b finite,
 quad_qagi ( expr, var, a, inf ), with a finite, or
 quad_qagi (expr, var, minf, inf ).
```

If at least one of **(a,b)** are not equal to **(minf,inf)**, a noun form will be returned.

The Maxima function **quad_qagi** returns the same type of information (approx-integral, est-abs-error, nfe, error-code) in a list that **quad_qags** returns, and the possible error codes returned have the same meaning.

Here we test the syntax and behavior with a simple integrand, first over $[\mathbf{0}, \infty]$:

```
(%i1) (fpprintprec:8,display2d:false)$
(%i2) tval : block ([fpprec:20],bfloat( integrate (exp(-x^2),x,0,inf) ));
(%o2) 8.8622692b-1
(%i3) quad_qagi(exp(-x^2),x,0,inf);
(%o3) [0.886227,7.10131839E-9,135,0]
(%i4) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o4) 7.2688979b-17
(%i5) %/tval;
(%o5) 8.202073b-17
```

and next the same simple integrand over $[-\infty, \mathbf{0}]$:

```
(%i6) quad_qagi(exp(-x^2),x,minf,0);
(%o6) [0.886227,7.10131839E-9,135,0]
(%i7) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o7) 7.2688979b-17
```

and, finally over $[-\infty, \infty]$:

```
(%i8) tval : block ([fpprec:20],bfloat(2*tval));
(%o8) 1.7724538b0
(%i9) quad_qagi(exp(-x^2),x,minf,inf);
(%o9) [1.7724539,1.42026368E-8,270,0]
(%i10) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o10) 1.4537795b-16
```

Example 1

Here is another simple example:

```
(%i11) g : exp(-x)*x^(5/100)$
(%i12) tval : block ([fpprec:20],bfloat( integrate(g,x,0,inf) ));
(%o12) 9.7350426b-1
(%i13) quad_qagi(g,x,0,inf);
(%o13) [0.973504,1.2270015E-9,315,0]
(%i14) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o14) 7.9340695b-15
(%i15) %/tval;
(%o15) 8.15001b-15
```

We see that the use of the default mode (accepting the default **epsrel:1d-8** and **epsabs:0**) has resulted in an absolute error which is close to the floating point limit and a relative error of the same order of magnitude (because the value of the integral is of order 1).

Example 2

We evaluate the integral $\int_0^\infty e^{-x} \ln(x)\,dx = -\gamma$, where γ is the Euler-Mascheroni constant, **0.5772156649015329**. Maxima has this constant available as **%gamma**, although you need to use either **float** or **bfloat** to get the numerical value.

```
(%i16) g : exp(-x)*log(x)$
(%i17) tval : block ([fpprec:20],bfloat( integrate(g,x,0,inf) ));
(%o17) -5.7721566b-1
(%i18) quad_qagi(g,x,0,inf);
(%o18) [-0.577216,5.11052578E-9,345,0]
(%i19) block ([fpprec:20], bfloat(abs(first(%) - tval)));
(%o19) 2.6595919b-15
(%i20) %/tval;
(%o20) -4.6076226b-15
```

Example 3

A symmetrical version of a Fourier transform pair is defined by the equations

$$g(x) = \frac{1}{\sqrt{2\,\pi}} \int_{-\infty}^{\infty} G(k)\, e^{i\,k\,x}\, dk \tag{8.9}$$

$$G(k) = \frac{1}{\sqrt{2\,\pi}} \int_{-\infty}^{\infty} g(x)\, e^{-i\,k\,x}\, dx \tag{8.10}$$

An example of such a Fourier transform pair which respects this symmetrical definition is: if $g(x) = a\,e^{-b\,x^2}$, then $G(k) = (a/\sqrt{2\,b})\,e^{-k^2/(4\,b)}$.

```
(%i21) assume(a>0,b>0,k>0)$
(%i22) g :a*exp(-b*x^2)$
(%i23) gft:integrate(exp(-%i*k*x)*g,x,minf,inf)/sqrt(2*%pi);
(%o23) a*%e^-(k^2/(4*b))/(sqrt(2)*sqrt(b))
```

To check this relation with **quag_qagi**, let's define **f** to be **g** for the case **a, b** and **k** are all set equal to **1**.

```
(%i24) f : subst([a=1,b=1,k=1],g);
(%o24) %e^-x^2
(%i25) fft : subst([a=1,b=1,k=1],gft);
(%o25) %e^-(1/4)/sqrt(2)
(%i26) float(fft);
(%o26) 0.550695
```

If we try to submit an explicitly complex integrand to **quad_qagi** we get a noun form back, indicating failure. (A similar result occurs with any quadpack function.).

```
(%i27) quad_qagi(f*exp(-%i*x),x,minf,inf);
(%o27) quad_qagi(%e^(-x^2-%i*x),x,minf,inf,epsrel = 1.0E-8,epsabs = 0.0,limit
                                                                        = 200)
```

You can use **quad_qagi** for numerical double integrals, following the pattern we used for **quad_qags**, and you can choose different methods for each of the axes.

8.6 Numerical Integration: Sharper Tools

There are specialised Quadpack routines for particular kinds of one dimensional integrals. See Sec.(8.7) for a "decision tree" for a finite region numerical integral using the Quadpack functions.

8.6.1 quad_qagp for Internal Integrand Singularities

The quadpack function **quad_qagp** is both more efficient and more accurate than **quad_qags** for the case of integrand singularities interior to the integration interval. The syntax is the same as **quad_qags**, except that the fifth argument should be a list of either one or more points where the interior singularities occur. The elements of the points list should evaluate to numbers.

We compare the use of **quad_qagp** and **quad_qags** for the numerical value of the integral

$$\int_0^3 x^3 \ln\left(\left|\left(x^2-1\right)\,\left(x^2-2\right)\right|\right)\,dx \tag{8.11}$$

which has internal singularities of the integrand at $x = 1$ and $x = \sqrt{2}$.

```
(%i28) quad_qagp(x^3*log(abs((x^2-1)*(x^2-2))),x,0,3,[1,sqrt(2)]);
(%o28) [52.740748,2.62476263E-7,1029,0]
(%i29) quad_qags(x^3*log(abs((x^2-1)*(x^2-2))),x,0,3);
(%o29) [52.740748,4.08844336E-7,1869,0]
```

8.6.2 quad_qawo for Fourier Series Coefficients

This function is the most efficient way to find numerical values of Fourier series coefficients, which require finding integrals $\int_a^b f(x)\cos(\omega\, x)\,dx$ or $\int_a^b f(x)\sin(\omega\, x)\,dx$.

This function has the syntax

```
quad_qawo(expr, var , a, b, omega, trig,[epsrel, epsabs, limit, maxp1])
```

For the integral $\int_{-2}^{2}(x + x^4)\cos(3\,x)\,dx$ (default options), use `quad_qawo(x + x^4, x, -2, 2, 3, cos)`.

For the integral $\int_{-2}^{2}(x + x^4)\sin(5\,x)\,dx$ (default options), use `quad_qawo(x + x^4, x, -2, 2, 5, sin)`.
Here we compare **quad_qawo** with **quad_qags** for the integral $\int_{-2}^{2}(x + x^4)\cos(3\,x)\,dx$.

```
(%i1) fpprintprec:8$
(%i2) g : (x+x^4)*cos(3*x)$
(%i3) tval : bfloat( integrate(g,x,-2,2) ),fpprec:20;
(%o3)                           3.6477501b0
(%i4) quad_qawo(x+x^4,x,-2,2,3,cos);
(%o4)                     [3.6477502, 0.0, 25, 0]
(%i5) abs(first(%) - tval),fpprec:20;
(%o5)                          4.0522056b-18
(%i6) %/tval,fpprec:20;
(%o6)                          1.110878b-18
(%i7) quad_qags(g,x,-2,2);
(%o7)               [3.6477502, 8.89609255E-14, 63, 0]
(%i8) abs(first(%) - tval),fpprec:20;
(%o8)                          1.7723046b-15
(%i9) %/tval,fpprec:20;
(%o9)                          4.8586239b-16
```

We see that **quad_qawo** finds the numerical value with “zero” relative error as compared with a “non-zero” relative error using **quad_qags**, and with many less integrand evaluations.

8.6.3 quad_qaws for End Point Algebraic and Logarithmic Singularities

The syntax is:

```
quad_qaws (f(x), x, a, b, alpha, beta, wfun,[epsrel, epsabs, limit])
```

This Maxima function is designed for the efficient evaluation of integrals of the form $\int_a^b f(x)\,w(x)\,dx$ in which the appropriate “singular end point weight function” is chosen from among different versions via the three parameters **wfun**, α (represented by **alpha**), and β (represented by **beta**).

The most general case in which one has both algebraic and logarithmic singularities of the integrand at both end points corresponds to $\alpha \neq 0$ and $\beta \neq 0$ and

$$w(x) = (x - a)^{\alpha}\,(b - x)^{\beta}\,\ln(x - a)\,\ln(b - x) \tag{8.12}$$

The parameters α and β govern the “degree” of algebraic singularity at the end points. One needs both $\alpha > -1$ and $\beta > -1$ for convergence of the integrals.

In particular, one can choose $\alpha = 0$ and/or $\beta = 0$ to handle an algebraic singularity at only one end of the interval or no algebraic singularities at all.

The parameter **wfun** determines the existence and location of possible end point logarithmic singularities of the integrand.

wfun	w(x)
1	$(x-a)^{\alpha}\,(b-x)^{\beta}$
2	$(x-a)^{\alpha}\,(b-x)^{\beta}\,\ln(x-a)$
3	$(x-a)^{\alpha}\,(b-x)^{\beta}\,\ln(b-x)$
4	$(x-a)^{\alpha}\,(b-x)^{\beta}\,\ln(x-a)\,\ln(b-x)$

Example 1: Pure Logarithmic Singularities

For the case that $\alpha = 0$ and $\beta = 0$, there are no end point algebraic singularities, only logarithmic singularities. A simple example is $\int_0^1 \ln(x)\, dx$, which corresponds to $\mathbf{wfun} = 2$:

```
(%i1) fpprintprec:8$
(%i2) tval : bfloat( integrate(log(x),x,0,1) ),fpprec:20;
(%o2)                               - 1.0b0
(%i3) quad_qaws(1,x,0,1,0,0,2);
(%o3)                  [- 1.0, 9.68809031E-15, 40, 0]
(%i4) abs(first(%) - tval),fpprec:20;
(%o4)                                0.0b0
(%i5) quad_qags(log(x),x,0,1);
(%o5)                  [- 1.0, 1.11022302E-15, 231, 0]
(%i6) abs(first(%) - tval),fpprec:20;
(%o6)                                0.0b0
```

which illustrates the efficiency of **quad_qaws** compared to **quad_qags** for this type of integrand.

Example 2: Pure Algebraic Singularity

The case $\mathbf{wfun} = 1$ corresponds to purely algebraic end point singularities.

Here we compare **quad_qaws** with **quad_qags** for the evaluation of the integral $\int_0^1 \frac{\sin(x)}{\sqrt{x}}\, dx$. You will get an exact symbolic answer in terms of **erf(z)** for this integral from **integrate**.

```
(%i7) expand(bfloat(integrate(sin(x)/sqrt(x),x,0,1))),fpprec:20;
(%o7)       1.717976b0 cos(0.25 %pi) - 8.404048b-1 sin(0.25 %pi)
(%i8) tval : bfloat(%),fpprec:20;
(%o8)                           6.205366b-1
(%i9) quad_qaws(sin(x),x,0,1,-1/2,0,1);
(%o9)                 [0.620537, 4.31887834E-15, 40, 0]
(%i10) abs(first(%) - tval),fpprec:20;
(%o10)                          8.8091426b-19
(%i11) %/tval,fpprec:20;
(%o11)                          1.4196008b-18
(%i12) quad_qags(sin(x)/sqrt(x),x,0,1);
(%o12)                [0.620537, 3.48387985E-13, 231, 0]
(%i13) abs(first(%) - tval),fpprec:20;
(%o13)                          1.1014138b-16
(%i14) %/tval,fpprec:20;
(%o14)                          1.7749378b-16
```

We see that **quad_qaws** uses about one sixth the number of function evaluations (as compared with **quad_qags**) and returns a much more accurate answer.

Example 3: Both Algebraic and Logarithmic Singularity at an End Point

A simple example is $\int_0^1 \frac{\ln(x)}{\sqrt{x}}\, dx$. The integrand is singular at $x = 0$ but this is an "integrable singularity" since $\sqrt{x}\,\ln(x) \to 0$ as $x \to 0^+$.

```
(%i1) fpprintprec:8$
(%i2) limit(log(x)/sqrt(x),x,0,plus);
(%o2)                                minf
```

```
(%i3) integrate(log(x)/sqrt(x),x);
(%o3)                  2 (sqrt(x) log(x) - 2 sqrt(x))
(%i4) limit(%,x,0,plus);
(%o4)                               0
(%i5) tval : bfloat( integrate(log(x)/sqrt(x),x,0,1)),fpprec:20;
(%o5)                            - 4.0b0
(%i6) quad_qaws(1,x,0,1,-1/2,0,2);
(%o6)                  [- 4.0, 3.59396672E-13, 40, 0]
(%i7) abs(first(%) - tval),fpprec:20;
(%o7)                             0.0b0
(%i8) quad_qags(log(x)/sqrt(x),x,0,1);
(%o8)                  [- 4.0, 1.94066985E-13, 315, 0]
(%i9) abs(first(%) - tval),fpprec:20;
(%o9)                          2.6645352b-15
(%i10) %/tval,fpprec:20;
(%o10)                        - 6.6613381b-16
```

Again we see the relative efficiency and accuracy of **quad_qaws** for this type of integral.

8.6.4 quad_qawc for a Cauchy Principal Value Integral

This function has the syntax:

```
          quad_qawc (f(x), x, c, a, b,[epsrel, epsabs, limit]).
```

The actual integrand is $g(x) = f(x)/(x - c)$, with dependent variable x, to be integrated over the interval $[a, b]$ and you need to pick out $f(x)$ by hand here. In using **quad_qawc**, the argument c is placed between the name of the variable of integration (here x) and the lower limit of integration.

An integral with a "pole" on the contour does not exist in the strict sense, but if $g(x)$ has a simple pole on the real axis at $x = c$, one defines the Cauchy principal value as the symmetrical limit (with $a < c < b$)

$$P \int_a^b g(x)\,dx = \lim_{\epsilon \to 0^+} \left[\int_a^{c-\epsilon} g(x)\,dx + \int_{c+\epsilon}^b g(x)\,dx \right] \tag{8.13}$$

provided this limit exists. In terms of $f(x)$ this definition becomes

$$P \int_a^b \frac{f(x)}{x-c}\,dx = \lim_{\epsilon \to 0^+} \left[\int_a^{c-\epsilon} \frac{f(x)}{x-c}\,dx + \int_{c+\epsilon}^b \frac{f(x)}{x-c}\,dx \right] \tag{8.14}$$

We can find the default values of the optional method parameters of **quad_qawc** by including an undefined symbol in our call:

```
(%i11) quad_qawc(1/(x^2-1),x,1,0,b);
                   1
(%o11) quad_qawc(------, x, 1, 0, b, epsrel = 1.0E-8, epsabs = 0.0, limit = 200)
                  2
                 x  - 1
```

We see that the default settings cause the algorithm to look at the relative error of succeeding approximations to the numerical answer.

As a simple example of the syntax we consider the principal value integral

$$P\int_0^2 \frac{1}{x^2-1}\,dx = P\int_0^2 \frac{1}{(x-1)\,(x+1)}\,dx = -\ln(3)/2 \tag{8.15}$$

We use **assume** to prep **integrate** and then implement the basic definition provided by Eq. (8.13)

```
(%i1) fpprintprec:8$
(%i2) assume(eps>0, eps<1)$
(%i3) integrate(1/(x^2-1),x,0,1-eps) +
               integrate(1/(x^2-1),x,1+eps,2);
                     log(eps + 2)   log(2 - eps)   log(3)
(%o3)                ------------ - ------------ - ------
                          2              2           2
(%i4) limit(%,eps,0,plus);
                                    log(3)
(%o4)                             - ------
                                      2
(%i5) tval : bfloat(%),fpprec:20;
(%o5)                           - 5.4930614b-1
```

We now compare the result returned by **integrate** with the numerical value returned by **quad_qawc**, noting that $f(x) = 1/(1+x)$.

```
(%i6) quad_qawc(1/(1+x),x,1,0,2);
(%o6)                [- 0.549306, 1.51336373E-11, 105, 0]
(%i7) abs(first(%) - tval),fpprec:20;
(%o7)                            6.5665382b-17
(%i8) %/tval,fpprec:20;
(%o8)                          - 1.1954241b-16
```

We see that the relative error of the returned answer is much less than the (default) requested minimum relative error.

If we run **quad_qawc** requesting that convergence be based on absolute error instead of relative error,

```
(%i9) quad_qawc(1/(1+x),x,1,0,2,epsabs=1.0e-10,epsrel=0.0);
(%o9)                [- 0.549306, 1.51336373E-11, 105, 0]
(%i10) abs(first(%) - tval),fpprec:20;
(%o10)                           6.5665382b-17
(%i11) %/tval,fpprec:20;
(%o11)                         - 1.1954241b-16
```

we see no significant difference in the returned accuracy, and again we see that the absolute error of the returned answer is much less than the requested minimum absolute error.

8.6.5 quad_qawf for a Semi-Infinite Range Cosine or Sine Fourier Transform

The function **quad_qawf** calculates a Fourier cosine *or* Fourier sine transform (up to an overall normalization factor) on the semi-infinite interval $[a, \infty]$. If we let **w** stand for the angular frequency in radians, the integrand is **f(x)*cos(w*x)** if the **trig** parameter is **cos**, and the integrand is **f(x)*sin(w*x)** if the **trig** parameter is **sin**.

The calling syntax is

```
quad_qawf (f(x), x, a, w, trig, [epsabs, limit, maxp1, limlst])
```

Thus `quad_qawf (f(x), x, 0, w, 'cos)` will find a numerical approximation to the integral

$$\int_0^\infty f(x)\cos(w\,x)\,dx \tag{8.16}$$

If we call **quad_qawf** with undefined parameter(s), we get a look at the default values of the optional method parameters:

```
(%i12) quad_qawf(exp(-a*x),x,0,w,'cos);
                  - a x
(%o12) quad_qawf(%e     , x, 0, w, cos, epsabs = 1.0E-10, limit = 200,
                                                maxp1 = 100, limlst = 10)
```

The manual has the optional parameter information:

```
The keyword arguments are optional and may be specified in any order.
They all take the form keyword = val. The keyword arguments are:
1. epsabs, the desired absolute error of approximation. Default is 1d-10.
2. limit, the size of the internal work array.
   (limit - limlst)/2 is the maximum number of
   subintervals to use. The default value of limit is 200.
3. maxp1, the maximum number of Chebyshev moments.
      Must be greater than 0. Default is 100.
4. limlst, upper bound on the number of cycles.
    Must be greater than or equal to 3. Default is 10.
```

The manual does not define the meaning of “cycles”. There is no “epsrel” parameter used for this function.

Here is the manual example, organised in our way. In this example $w = 1$ and $a = 0$.

```
(%i1) fpprintprec:8$
(%i2) integrate (exp(-x^2)*cos(x), x, 0, inf);
                                 - 1/4
                               %e      sqrt(%pi)
(%o2)                          -----------------
                                       2
(%i3) tval : bfloat(%),fpprec:20;
(%o3)                            6.9019422b-1
(%i4) quad_qawf (exp(-x^2), x, 0, 1, 'cos );
(%o4)                 [0.690194, 2.84846299E-11, 215, 0]
(%i5) abs(first(%) - tval),fpprec:20;
(%o5)                            3.909904b-17
(%i6) %/tval,fpprec:20;
(%o6)                            5.664933b-17
```

We see that the absolute error of the returned answer is much less than the (default) requested minimum absolute error.

8.7 Finite Region of Integration Decision Tree

If you are in a hurry, use **quad_qags**. [*]

If your definite integral is over a finite region of integration **[a, b]**, then

1. If the integrand has the form **w(x)*f(x)**, where **f(x)** is a smooth function over the region of integration, then
   - If **w(x)** has the form of either **cos(c*x)** or **sin(c*x)**, where **c** is a constant, use **quad_qawo**.
   - Else if the factor **w(x)** has the form (note the limits of integration are **[a, b]**

     ```
     (x - a)^ae * (b - x)^be * (log(x - a))^na * (log(b - x))^nb
     ```

     where **na**, **nb** have the values **0** or **1**, and both **ae** and **be** are greater than -1, then use **quad_qaws**. (This is a case where we need a routine which is designed to handle end point singularities.)
   - Else if the factor **w(x)** is **1/(x - c)** for some constant **c** with $a < c < b$, then use **quad_qawc**, the Cauchy principle value routine.
2. Otherwise, if you do not care too much about possible inefficient use of computer time, and do not want to further analyze the problem, use **quad_qags**.
3. Otherwise, if the integrand is **smooth**, use **quad_qag**.
4. Otherwise, if there are **discontinuities or singularities** of the integrand or of the derivative of the integrand, and you know where they are, split the integration range at these points and separately integrate over each subinterval.
5. Otherwise, if the integrand has **end point singularities**, use **quad_qags**.
6. Otherwise, if the integrand has an oscillatory behavior of nonspecific type, and no singularities, use **quad_qag** with the fifth **key** slot containing the value **6**.
7. Otherwise, use **quad_qags**.

[*] These "decision trees" are adapted from the web page
`http://people.scs.fsu.edu/~burkardt/f77_src/quadpack/quadpack.html`.

8.8 Non-Finite Region of Integration Decision Tree

Here is the tree for the non-finite domain case:

```
A.  If the integration domain is non-finite consider first making a
       suitable change of variables to obtain a finite domain and
           using the finite interval decision tree described above.

B.  Otherwise, if the integrand decays rapidly to zero, truncate the
      integration interval and use the finite interval decision tree.

C.  Otherwise, if the integrand oscillates over the entire non-finite range,

       1. If integral is a Fourier transform, use quad_qawf.

       2. else if the integral is not a Fourier transform, then sum the
           successive positive and negative contributions by integrating
           between the zeroes of the integrand, using the finite
           interval criteria.

D.  Otherwise, if you are not constrained by computer time, and do not
     wish to analyze the problem further, use quad_qagi.

E.  Otherwise, if the integrand has a non-smooth behavior in the range of
     integration, and you know where it occurs, split off these regions
     and use the appropriate finite range routines to integrate over them.
     Then begin this non-finite case tree again to handle the remainder
         of the region.

F.  Otherwise, truncation of the interval, or application of a suitable
     transformation for reducing the problem to a finite range may be
     possible.

G.  Otherwise use quad_qagi.
```