Maxima by Example:
Ch.8: Numerical Integration *

Edwin L. Woollett

September 16, 2010

Contents

8.1 Introduction . . . 3
8.2 Numerical Integration Basic Tools: **quad_qags**, **romberg**, **quad_qagi** . . . 3
8.2.1 Syntax for Quadpack Functions . . . 3
8.2.2 Ouput List of Quadpack Functions and Error Code Values . . . 3
8.2.3 Integration Rule Parameters and Optional Arguments . . . 4
8.2.4 **quad_qags** for a Finite Interval . . . 4
8.2.5 **romberg** for a Finite Interval . . . 7
8.2.6 **quad_qags** and **romberg** for Numerical Double Integrals . . . 8
8.2.7 **quad_qagi** for an Infinite or Semi-infinite Interval . . . 10
8.3 Numerical Integration: Sharper Tools . . . 12
8.3.1 **quad_qag** for a General Oscillatory Integrand . . . 12
8.3.2 **quad_qawo** for Fourier Series Coefficients . . . 14
8.3.3 **quad_qaws** for End Point Algebraic and Logarithmic Singularities . . . 15
8.3.4 **quad_qawc** for a Cauchy Principal Value Integral . . . 17
8.3.5 **quad_qawf** for a Semi-Infinite Range Cosine or Sine Fourier Transform . . . 19
8.4 Finite Region of Integration Decision Tree . . . 20
8.5 Semi-infinite or Infinite Region of Integration Decision Tree . . . 21

*This version uses **Maxima 5.18.1**. 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

```
COPYING AND DISTRIBUTION POLICY
This document is part of a series of notes titled
"Maxima by Example" and is made available
via the author's webpage http://www.csulb.edu/~woollett/
to aid new users of the Maxima computer algebra system.

NON-PROFIT PRINTING AND DISTRIBUTION IS PERMITTED.

You may make copies of this document and distribute them
to others as long as you charge no more than the costs of printing.

These notes (with some modifications) will be published in book form
eventually via Lulu.com in an arrangement which will continue
to allow unlimited free download of the pdf files as well as the option
of ordering a low cost paperbound version of these notes.
```

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.

We use qdraw.mac (available on the author's web page with Ch. 5 materials)
for plots.
```

```
  Maxima, a Computer Algebra System.
  Some numerical results depend on the Lisp version used.
  This chapter uses Version 5.18.1 (2009) 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.

8.1 Introduction

This chapter is divided into four sections.

We first discuss the most needed and used tools for routine numerical integration. (We presented many examples of symbolic integration methods in Ch. 7). We then discuss more exotic tools you may be able to use once in a while. In the third short section we present a finite interval integration "decision tree". In the fourth short section we present an "infinite interval integration decision tree". In Chapter 9, we will discuss bigfloats and some examples of using Maxima for high precision quadrature. 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.

8.2 Numerical Integration Basic Tools: quad_qags, romberg, quad_qagi

8.2.1 Syntax for Quadpack Functions

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).

8.2.2 Ouput List of Quadpack Functions and Error Code Values

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

```
[integral.value, est.abs.error, num.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

8.2.3 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.

8.2.4 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)`.

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.

```
(%i1) quad_qags(a*x,x,0,1);
(%o1) 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)

$$w(x) = (x-a)^{\alpha}\,(b-x)^{\beta}\,\ln(x-a)\,\ln(b-x) \tag{8.1}$$

where $\alpha > -1$ and $\beta > -1$ for convergence. See Sec 8.3.3 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.

```
(%i2) g:sqrt(x)*log(1/x)$
(%i3) limit(g,x,0,plus);
(%o3)                                  0
(%i4) (load(draw),load(qdraw))$
(%i5) qdraw( ex(g,x,0,1) )$
```

Here is that plot.

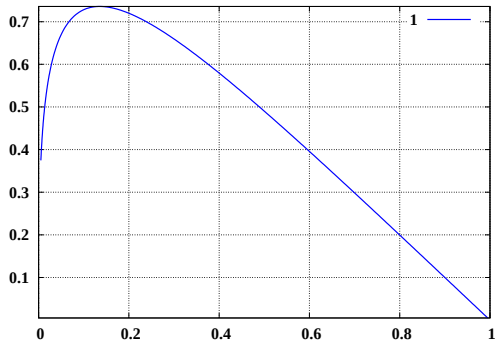

Figure 1: $x^{1/2}\ln(1/x)$

Now we try out **quad_qags** on this integral and compare with the exact answer returned by **integrate**.

```
(%i6) fpprintprec:8$
(%i7) tval:bfloat(integrate(g,x,0,1)),fpprec:20;
(%o7)                              4.4444444b-1
(%i8) qlist:quad_qags(g,x,0,1 );
(%o8)                  [0.444444, 4.93432455E-16, 315, 0]
(%i9) abs(first(%) - tval ),fpprec:20;
(%o9)                              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 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

$$\mathbf{est\_rel\_err} \leq \mathbf{epsrel} \tag{8.2}$$

or

$$\frac{\mathbf{est\_abs\_err}}{\mathbf{est\_answer}} \leq \mathbf{epsrel} \tag{8.3}$$

or

$$\mathbf{est\_abs\_err} \leq \mathbf{epsrel} \times \mathbf{est\_answer} \tag{8.4}$$

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

```
(%i10) est_answer : first(qlist);
(%o10)                              0.444444
(%i11) est_abs_err : second(qlist);
(%o11)                           4.93432455E-16
(%i12) est_rel_err : est_abs_err/est_answer;
(%o12)                           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 $\mathbf{x} = \mathbf{0}$, but the singularity is an "integrable singularity". We want to calculate $\int_0^1 \mathbf{ln}(\mathbf{sin}(\mathbf{x}))\,\mathbf{dx}$. If you answer enough questions correctly, **integrate** returns an "exact" answer involving the limit of expressions involving the dilogarithm function with a complex argument, and the "answer" is rather long.

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

```
(%i1) limit(log(sin(x)),x,0,plus);
(%o1)                                minf
(%i2) ix : taylor(log(sin(x)),x,0,2);
                                        2
                                       x
(%o2)/T/                 log(x) + . . . - -- + . . .
                                       6
(%i3) int_ix:integrate(ix,x);
                                       3
                                      x
(%o3)                        x log(x) - -- - x
                                      18
(%i4) limit(int_ix,x,0,plus);
(%o4)                                0
(%i5) assume(eps>0,eps<1)$
(%i6) integrate(ix,x,0,eps);
                                          3
                      18 eps log(eps) - eps  - 18 eps
(%o6)                 -------------------------------
                                   18
```

Output **%o1** indicates that the integrand $\to -\infty$ as $x \to 0^+$. 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 $x \to 0^+$. 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 $[0, 1]$.

```
(%i7) quad_qags(log(sin(x)),x,0,1);
(%o7)                [- 1.0567202, 1.1731951E-15, 231, 0]
```

Use of **float(integrate)** with this expression returns a very complicated result involving proposed limits which Maxima has not carried out.

8.2.5 romberg for a Finite Interval

If you have loaded in **romberg**, the syntax **romberg(expr,var, a, b)** returns an estimate of an integral using what is known as "Romberg's Method".

The quadpack function **quad_qags** is more accurate and includes error diagnostics. However we include a short discussion of **romberg** since we will try out the big-float version **bromberg** in Ch. 9.

The default behavior of **romberg** is governed by a relative difference test applied to successive iterations and is determined by the default value of **rombergtol** (which default value is **1e-4**) If you want to change that value, you issue the separate assignment statement: **rombergtol : 1e-15** for example.

```
(%i1) fpprintprec:8$
(%i2) load(romberg);
(%o2) C:/PROGRA~1/MAXIMA~3.1/share/maxima/5.18.1/share/numeric/romberg.lisp
(%i3) [rombergtol,rombergabs,rombergit,rombergmin];
(%o3)                        [1.0E-4, 0.0, 11, 0]
```

We can compare **romberg** with **quad_qags**. Of course the default relative error is different, but by setting **rombergtol** to the value **1e-15** we can then compare their output. We will use the simple integral $\int_0^1 e^x \, dx$:

```
(%i4) tval : bfloat(integrate(exp(x),x,0,1)),fpprec:20;
(%o4)                             1.7182818b0
(%i5) rombergtol:1e-15$
(%i6) romberg(exp(x),x,0,1);
(%o6)                              1.7182818
(%i7) abs(% - tval),fpprec:20;
(%o7)                            3.6660941b-16
(%i8) %/tval,fpprec:20;
(%o8)                            2.1335813b-16
(%i9) quad_qags(exp(x),x,0,1 );
(%o9)                 [1.7182818, 1.90767605E-14, 21, 0]
(%i10) abs(%[1] - tval),fpprec:20;
(%o10)                           7.7479797b-17
(%i11) %/tval,fpprec:20;
(%o11)                           4.5091437b-17
```

So we see that **quad_qags** yields an answer with a somewhat smaller relative error than **romberg**.

8.2.6 quad_qags and romberg for Numerical 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

$$\int_{u1}^{u2} du \int_{v1(u)}^{v2(u)} dv\, f(u,v) \equiv \int_{u1}^{u2} \left(\int_{v1(u)}^{v2(u)} f(u,v)\, dv \right) du \tag{8.5}$$

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 **romberg** to find the same double integral (numerically), we use the syntax:

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

and 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 compare **quad_qags** and **romberg** on the double integral:

$$\int_1^3 \left(\int_0^{x/2} \frac{x\,y}{x+y}\, dy \right) dx \tag{8.6}$$

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

```
(%i1) fpprintprec:8$
(%i2) g : x*y/(x+y)$
(%i3) tval : bfloat (integrate (integrate (g,y,0,x/2),x,1,3) ),fpprec:20;
Is  x  positive, negative, or zero?

p;
(%o3)                              8.1930239b-1
(%i4) load(romberg)$
(%i5) [rombergtol,rombergabs,rombergit,rombergmin];
(%o5)                         [1.0E-4, 0.0, 11, 0]
(%i6) rombergtol:1e-15$
(%i7) romberg( romberg(g,y,0,x/2),x,1,3);
(%o7)                                0.819302
(%i8) abs(% - tval),fpprec:20;
(%o8)                              1.7298445b-16
(%i9) %/tval,fpprec:20;
(%o9)                              2.1113627b-16
```

```
(%i10) quad_qags ( quad_qags (g,y,0,x/2)[1],x,1,3);
(%o10)                  [0.819302, 9.09608385E-15, 21, 0]
(%i11) abs (%[1] - tval),fpprec:20;
(%o11)                             6.1962154b-17
(%i12) %/tval,fpprec:20;
(%o12)                             7.5627942b-17
```

We see that **quad_qags** is slightly more accurate.

Example 2

For our second example, we consider the double integral

$$\int_0^1 \left(\int_1^{2+x} e^{x-y^2} \, dy \right) dx \tag{8.7}$$

(Note we still have **rombergtol** set to **1e-15**).

```
(%i13) g : exp(x-y^2)$
(%i14) tval : bfloat(integrate( integrate(g,y,1,2+x),x,0,1)),fpprec:20;
(%o14)                             2.3846836b-1
(%i15) romberg( romberg(g,y,1,2+x),x,0,1);
'romberg' failed to converge
                                  2
                               x - y
(%o15)     romberg(romberg(%e      , y, 1.0, x + 2.0), x, 0.0, 1.0)
(%i16) quad_qags( quad_qags(g,y,1,2+x)[1],x,0,1);
(%o16)                  [0.238468, 2.64753066E-15, 21, 0]
(%i17) abs (%[1] - tval),fpprec:20;
(%o17)                             4.229659b-18
(%i18) %/tval,fpprec:20;
(%o18)                             1.7736772b-17
```

We see that **romberg** failed to converge to an answer. If you decrease the value of **rombergtol**, you can eventually get an answer from **romberg** for this double integral.

Maxima coordinator Robert Dodier has (on the Mailing List) commented on the general preference for using **quad_qags** instead of **romberg**:

> ... the quadpack functions are stronger than romberg: they succeed on problems for which romberg fails, they product diagnostic codes when they do fail, they produce the same accuracy with fewer function calls when they succeed, and there are specialized methods for various special cases (infinite domain, etc).
>
> I can't think of a problem for which I'd recommend romberg over quadpack.

Dodier has also 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.

8.2.7 quad_qagi for an Infinite or Semi-infinite Interval

See Sec.(8.5) for a decision tree for quadrature over an infinite or semi-infinite region.

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 $[0, \infty]$:

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

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

```
(%i5) quad_qagi(exp(-x^2),x,minf,0);
(%o5)                 [0.886227, 7.10131839E-9, 135, 0]
(%i6) abs(first(%) - tval),fpprec:20;
(%o6)                           7.2688979b-17
```

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

```
(%i7) tval : bfloat(2*tval),fpprec:20;
(%o7)                            1.7724538b0
(%i8) quad_qagi(exp(-x^2),x,minf,inf);
(%o8)               [1.7724539, 1.42026368E-8, 270, 0]
(%i9) abs(first(%) - tval),fpprec:20;
(%o9)                           1.4537795b-16
```

Example 1

Here is another simple example:

```
(%i10) g : exp(-x)*x^(5/100)$
(%i11) tval : bfloat( integrate(g,x,0,inf) ),fpprec:20;
(%o11)                          9.7350426b-1
(%i12) quad_qagi(g,x,0,inf);
(%o12)                [0.973504, 1.2270015E-9, 315, 0]
```

```
(%i13) abs(first(%) - tval),fpprec:20;
(%o13)                              7.9340695b-15
(%i14) %/tval,fpprec:20;
(%o14)                               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 $\int_0^\infty \mathrm{e}^{-x}\ln(x)\,\mathrm{d}x = -\gamma$, 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. The Maxima function **integrate** cannot make any progress with this integral.

```
(%i15) g : exp(-x)*log(x)$
(%i16) tval : bfloat( integrate(g,x,0,inf) ),fpprec:20;
The number 0 isn't in the domain of gamma_incomplete
 -- an error.  To debug this try debugmode(true);
(%i17) tval : bfloat(-%gamma),fpprec:20;
(%o17)                            - 5.7721566b-1
(%i18) quad_qagi(g,x,0,inf);
(%o18)                 [- 0.577216, 5.11052578E-9, 345, 0]
(%i19) abs(first(%) - tval),fpprec:20;
(%o19)                             2.6595919b-15
(%i20) %/tval,fpprec:20;
(%o20)                            - 4.6076226b-15
```

Example 3

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

$$\mathrm{g}(\mathrm{x}) = \frac{1}{\sqrt{2\,\pi}} \int_{-\infty}^{\infty} \mathrm{G}(\mathrm{k})\,\mathrm{e}^{\mathrm{i}\,\mathrm{k}\,\mathrm{x}}\,\mathrm{dk} \tag{8.8}$$

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

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

```
(%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);
                                          2
                                         k
                                       - ---
                                         4 b
                                    a %e
(%o23)                             ---------------
                                   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);
                                          2
                                        - x
(%o24)                                 %e
(%i25) fft : subst([a=1,b=1,k=1],gft);
                                          - 1/4
                                        %e
(%o25)                                  -------
                                        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 **quad_qags**).

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

8.3 Numerical Integration: Sharper Tools

There are specialised Quadpack routines for particular kinds of one dimensional integrals. See Sec.(8.4) for a "decision tree" for a finite region numerical integral using the Quadpack functions.

8.3.1 quad_qag for a General Oscillatory Integrand

The function **quad_qag** is sometimes of use 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$
(%i2) f : cos(50*x)*sin(3*x)*exp(-x)$
(%i3) tval : bfloat( integrate (f,x,0,1) ),fpprec:20;
(%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.

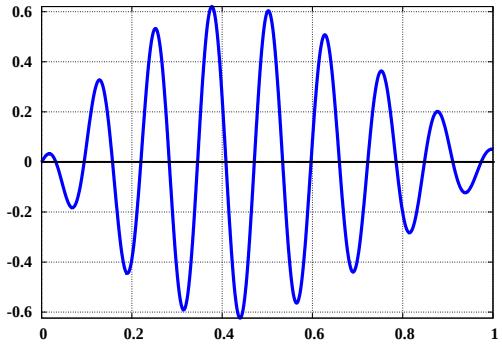

Figure 2: $\cos(50\,x)\sin(3\,x)\,e^{-x}$

```
(%i6) quad_qag(f,x,0,1,6);
(%o6)                 [- 0.00191455, 2.86107558E-15, 61, 0]
(%i7) abs(first(%) - tval),fpprec:20;
(%o7)                               9.8573614b-17
(%i8) %/tval,fpprec:20;
(%o8)                             - 5.148666b-14
(%i9) quad_qags(f,x,0,1 );
(%o9)                  [- 0.00191455, 2.8536E-15, 315, 0]
(%i10) abs(first(%) - tval),fpprec:20;
(%o10)                              1.1804997b-16
(%i11) %/tval,fpprec:20;
(%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 : bfloat( integrate (exp(x),x,0,1) ),fpprec:20;
(%o12)                          1.7182818b0
(%i13) quad_qag(exp(x),x,0,1,6);
(%o13)                [1.7182818, 1.90767605E-14, 61, 0]
(%i14) abs(first(%) - tval),fpprec:20;
(%o14)                          1.445648b-16
(%i15) %/tval,fpprec:20;
(%o15)                          8.413335b-17
(%i16) quad_qags(exp(x),x,0,1 );
(%o16)                [1.7182818, 1.90767605E-14, 21, 0]
(%i17) abs(first(%) - tval),fpprec:20;
(%o17)                          7.7479797b-17
(%i18) %/tval,fpprec:20;
(%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.(8.2.4) for a plot of this integrand.

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

8.3.2 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 $\int_{-2}^{2}(x+x^4)\cos(3x)\,dx$ (default options), use **quad_qawo(x + x^4, x, -2, 2, 3, cos)**.

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

```
(%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.

8.3.3 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

$$w(x) = (x-a)^{\alpha}\,(b-x)^{\beta}\,\ln(x-a)\,\ln(b-x) \tag{8.10}$$

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**.

```
(%i15) expand(bfloat(integrate(sin(x)/sqrt(x),x,0,1))),fpprec:20;
(%o15)       1.717976b0 cos(0.25 %pi) - 8.404048b-1 sin(0.25 %pi)
(%i16) tval : bfloat(%),fpprec:20;
(%o16)                              6.205366b-1
(%i17) quad_qaws(sin(x),x,0,1,-1/2,0,1);
(%o17)                  [0.620537, 4.31887834E-15, 40, 0]
(%i18) abs(first(%) - tval),fpprec:20;
(%o18)                              8.8091426b-19
(%i19) %/tval,fpprec:20;
(%o19)                              1.4196008b-18
(%i20) quad_qags(sin(x)/sqrt(x),x,0,1);
(%o20)                 [0.620537, 3.48387985E-13, 231, 0]
(%i21) abs(first(%) - tval),fpprec:20;
(%o21)                              1.1014138b-16
(%i22) %/tval,fpprec:20;
(%o22)                              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.

8.3.4 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 $a < c < b$)

$$P\int_a^b g(x)\,dx = \lim_{\epsilon\to 0^+}\left[\int_a^{c-\epsilon} g(x)\,dx + \int_{c+\epsilon}^b g(x)\,dx\right] \tag{8.11}$$

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

$$P\int_a^b \frac{f(x)}{x-c}\,dx = \lim_{\epsilon\to 0^+}\left[\int_a^{c-\epsilon} \frac{f(x)}{x-c}\,dx + \int_{c+\epsilon}^b \frac{f(x)}{x-c}\,dx\right] \tag{8.12}$$

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

$$\mathbf{P}\int_0^2 \frac{1}{x^2-1}\,dx = \mathbf{P}\int_0^2 \frac{1}{(x-1)(x+1)}\,dx = -\ln(3)/2 \tag{8.13}$$

We use **assume** to prep **integrate** and then implement the basic definition provided by Eq. (8.11)

```
(%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.

8.3.5 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

$$\int_0^\infty f(x)\cos(w\,x)\,dx \tag{8.14}$$

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.

8.4 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.

8.5 Semi-infinite or Infinite Region of Integration Decision Tree

Here is the tree for the semi-infinite or infinite domain case:

```
A.  If the integration domain is semi-infinite or from minus to plus infinity,
       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 infinite 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 semi-infinite or infinite 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.
```