

elrint3d - An Automatic Routine for the Evaluation of Three-Dimensional Integrals

October 5, 2009

1 Brief introduction

elrint3d (EMBEDDED LATTICE RULE INTEGRATOR - 3 DIMENSIONS) is an automatic cubature routine designed to compute a numerical approximation to an integral in one of the following two forms:

$$\int_a^b \int_c^d \int_e^f f(x, y, z) dz dy dx, \quad (1)$$

where a, b, c, d, e and f are all constants (including the possibility of ∞) or

$$\int_a^b \int_{g(x)}^{h(x)} \int_{p(x,y)}^{q(x,y)} f(x, y, z) dz dy dx, \quad (2)$$

where a and b are constants (again, possibly ∞) and $g(x)$, $h(x)$, $p(x, y)$ and $q(x, y)$ are real functions (whose values may be constant, including ∞).

The algorithm has been developed specifically for a 64-bit double precision environment.

To use the routine, the following must be provided:

1. the integrand (either as a class or a function pointer)
2. the integration domain (as constants or function pointers) and
3. the absolute and/or relative accuracy desired in the approximation.

The default maximum number of function evaluations permitted is 421,376, but a smaller maximum may be specified by the user.

The routine will terminate when it believes it has achieved either the requested absolute error or requested relative error (whichever is the easier) or when it determines that neither accuracy can be obtained (due to accumulation of rounding error or the maximum number of function evaluations being reached).

The following are returned or are directly accessible via access functions:

1. the approximation to the given integral
2. the estimated relative error in the approximation
3. an error flag indicating how reliable the returned result is likely to be, and
4. the number of function evaluations actually used in the computation.

Although integrals of type (2) above include those of type (1), it is strongly recommended that the calling sequence designed for type (1) integrals be used whenever the integration domain is cuboid. The routine has in-built efficiencies for this case.

Details of the underlying algorithm can be found in Li T. and Robinson I., “elrint3d: A Three-Dimensional Non-Adaptive Automatic Cubature Routine Using a Sequence of Embedded Lattice Rules”, ACM Transactions on Mathematical Software, to appear.

2 Setting up the package

The algorithm is comprised of 11 source files whose names can be found in Appendix A of this document. Users must ensure that all of these files are in the compile path of their development environment. (If in doubt, check the Help files in the environment.)

The statement `#include <elrint3d.h>` must be present in an application program for the algorithm to become available to that application.

Makefiles have also been provided for compiling and building the whole *elrint3d* package using GCC. See Appendix B for details of the makefile for the sample problems presented in the following sections and a sample makefile template for general usage.

3 Using the package

Integral specification is quite straightforward in *elrint3d*. We provide here five examples of the use of the algorithm to evaluate integrals over a variety of integration domains. The examples also demonstrate the use of both function pointers and a class to define the integrand.

3.1 Example 1

In this example, we integrate a Gaussian function over the unit cube. Specifically, we provide code for evaluating the integral

$$I_1 = \int_0^1 \int_0^1 \int_0^1 e^{-c_1^2(x-w_1)^2 - c_2^2(y-w_2)^2 - c_3^2(z-w_3)^2} dz dy dx.$$

In the sample code, the values of the parameters c_i and w_i , $i = 1, 2, 3$ are provided using assignment statements within the body of the program. In practice,

they would be defined externally and made available to the program by other means.

A function pointer is used to define the integrand.

```
#include    <iostream>
#include    <cmath>
#include    <elrint3d.h>
using namespace std;

/*****
  This example demonstrates the use of elrint3d for integration over
  the unit cube. The integrand is a Gaussian function, defined using
  a function pointer. The requested relative error is 1.0e-12.
*****/

double gaussian(double x, double y, double z)
{
    // Normally, the following parameters would be
    // available as externally-defined variables
    double c1 = 0.2;
    double c2 = 0.5;
    double c3 = 0.8;
    double w1 = 0.4;
    double w2 = 0.7;
    double w3 = 0.25;

    return exp(- c1*c1*(x - w1)*(x - w1)
               - c2*c2*(y - w2)*(y - w2)
               - c3*c3*(z - w3)*(z - w3));
}

int main()
{
    Elrint3d I1(gaussian, 0, 1, 0, 1, 0, 1, 1e-12);
    cout.precision(17);
    cout << "Cubature          = " << I1.evaluate() << endl;
    cout << "Error Flag         = " << I1.errFlag() << endl;
    cout << "Estimated Error = " << I1.estErr() << endl;
    cout << "No of Fun Vals  = " << I1.eval() << endl;
    return 0;
}
```

The code returns the following output:

Cubature = 0.88505988548101766

```
Error Flag      = 0
Estimated Error = 1.6934459552749742e-14
No of Fun Vals  = 26336
```

3.2 Example 2

In the second example, we evaluate the same integral as in Example 1, but this time using *elrint3d*'s class-related features. Note that in this case, the integrand is defined by a function whose name must be 'fun'.

```
#include    <iostream>
#include    <cmath>
#include    <elrint3d.h>
using namespace std;

/*****
  This example demonstrates the use of elrint3d for integration over
  the unit cube. The integrand is a Gaussian function, defined using
  a class. The requested relative error is 1.0e-12.
*****/

class GaussianIntegrand : public Integrand<double>
{
private:
    double c1, c2, c3;
    double w1, w2, w3;

public:
    GaussianIntegrand(double C1, double C2, double C3,
                      double W1, double W2, double W3):
        c1(C1), c2(C2), c3(C3), w1(W1), w2(W2), w3(W3)
    {
    }

    double fun(const double x[]) const
    {
        return exp(-c1*c1*(x[0] - w1)*(x[0] - w1)
                  - c2*c2*(x[1] - w2)*(x[1] - w2)
                  - c3*c3*(x[2] - w3)*(x[2] - w3));
    }
};

int main()
{
    GaussianIntegrand f(0.2, 0.5, 0.8, 0.4, 0.7, 0.25);
```

```

    Elrint3d I1(f, 0, 1, 0, 1, 0, 1, 1e-12);
    cout.precision(17);
    cout << "Cubature          = " << I1.evaluate() << endl;
    cout << "Error Flag       = " << I1.errFlag() << endl;
    cout << "Estimated Error = " << I1.estErr() << endl;
    cout << "No of Fun Vals  = " << I1.evals() << endl;
    return 0;
}

```

The output is identical to that produced by the previous example:

```

Cubature          = 0.88505988548101766
Error Flag       = 0
Estimated Error = 1.6934459552749742e-14
No of Fun Vals  = 26336

```

3.3 Example 3

There are two ways of describing the integration domain in *elrint3d*. If the integration domain is cuboid (finite or infinite), six constant parameters are taken to specify its boundary (as in the first two examples above). In this case, the computation required for the transformation of the cuboid on to the unit cube is carried out only once. If the integration domain is not cuboid, then a combination of constants, one-variable functions and two-variables functions is used to specify the domain. Specifying the domain in this way will result in the domain being mapped on to the unit cube every time the integrand is evaluated.

As an example of the specification of a variable boundary, we use the following code to compute the integral

$$I_2 = \int_0^1 \int_0^{1-x} \int_0^{1-x-y} e^{-c_1^2(x-w_1)^2 - c_2^2(y-w_2)^2 - c_3^2(z-w_3)^2} dz dy dx.$$

```

#include    <iostream>
#include    <cmath>
#include    <elrint3d.h>
using namespace std;

/*****
  This example demonstrates the use of elrint3d for integration over
  a variable domain. The integrand is a Gaussian function, defined
  using a function pointer. The requested relative error is 1.0e-12.
  *****/

double gaussian(double x, double y, double z)

```

```

{
    // Normally, the following parameters would be
    // available as externally-defined variables
    double c1 = 0.2;
    double c2 = 0.5;
    double c3 = 0.8;
    double w1 = 0.4;
    double w2 = 0.7;
    double w3 = 0.25;

    return exp(- c1*c1*(x - w1)*(x - w1)
               - c2*c2*(y - w2)*(y - w2)
               - c3*c3*(z - w3)*(z - w3));
}

//Define the lower boundary in the y direction
double yLower(double x)
{
    return 0.0;
}

//Define the upper boundary in the y direction
double yUpper(double x)
{
    return 1.0 - x;
}

//Define the lower boundary in the z direction
double zLower(double x, double y)
{
    return 0.0;
}

//Define the upper boundary in the z direction
double zUpper(double x, double y)
{
    return 1.0 - x - y;
}

int main()
{
    Elrint3d I2(gaussian, 0, 1, yLower, yUpper, zLower, zUpper, 1e-12);
    cout.precision(17);
    cout << "Cubature          = " << I2.evaluate() << endl;
    cout << "Error Flag          = " << I2.errFlag() << endl;
    cout << "Estimated Error = " << I2.estErr() << endl;
}

```

```

        cout << "No of Fun Vals  = " << I2.ivals() << endl;
        return 0;
    }

```

The following output is returned:

```

Cubature          = 0.15310907308404773
Error Flag        = 0
Estimated Error   = 7.3554261541766131e-13
No of Fun Vals    = 26336

```

3.4 Example 4

We repeat the evaluation of the integral in Example 3, but this time using *elrint3d*'s class-related features.

```

#include    <iostream>
#include    <cmath>
#include    <elrint3d.h>
using namespace std;

/*****
This example demonstrates the use of elrint3d for integration over
a variable domain.  The integrand is a Gaussian function, defined
using a class.  The requested relative error is 1.0e-12.
*****/

class GaussianIntegrand : public Integrand<double>
{
private:
    double c1, c2, c3;
    double w1, w2, w3;

public:
    GaussianIntegrand(double C1, double C2, double C3,
                      double W1, double W2, double W3):
        c1(C1), c2(C2), c3(C3), w1(W1), w2(W2), w3(W3)
    {
    }

    double fun(const double x[]) const
    {
        return exp(- c1*c1*(x[0] - w1)*(x[0] - w1)
                  - c2*c2*(x[1] - w2)*(x[1] - w2)
                  - c3*c3*(x[2] - w3)*(x[2] - w3));
    }

```

```

    }
};

//Define the lower boundary in the y direction
double yLower(double x)
{
    return 0;
}

//Define the upper boundary in the y direction
double yUpper(double x)
{
    return 1 - x;
}

//Define the lower boundary in the z direction
double zLower(double x, double y)
{
    return 0;
}

//Define the upper boundary in the z direction
double zUpper(double x, double y)
{
    return 1 - x - y;
}

int main()
{
    GaussianIntegrand f(0.2, 0.5, 0.8, 0.4, 0.7, 0.25);
    Elrint3d I2(f, 0, 1, yLower, yUpper, zLower, zUpper, 1e-12);
    cout.precision(17);
    cout << "Cubature          = " << I2.evaluate() << endl;
    cout << "Error Flag          = " << I2.errFlag() << endl;
    cout << "Estimated Error = " << I2.estErr() << endl;
    cout << "No of Fun Vals  = " << I2.ivals() << endl;
    return 0;
}

```

The output is identical to that of the previous example:

```

Cubature          = 0.15310907308404773
Error Flag        = 0
Estimated Error   = 7.3554261541766131e-13
No of Fun Vals    = 26336

```


3.5 Example 5

For integrals in which one or more of the limits of integration is infinite, the pre-defined constant INFINITY is provided. In the final example, we evaluate the integral

$$I_3 = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-c_1^2(x-w_1)^2 - c_2^2(y-w_2)^2 - c_3^2(z-w_3)^2} dz dy dx.$$

```
#include    <iostream>
#include    <cmath>
#include    <elrint3d.h>
using namespace std;

/*****
  This example demonstrates the use of elrint3d for integration over
  an infinite domain. The integrand is a Gaussian function, defined
  using a function pointer. The requested relative error is 1.0e-12.
  *****/

double gaussian(double x, double y, double z)
{
    // Normally, the following parameters would be
    // available as externally-defined variables
    double c1 = 0.2;
    double c2 = 0.5;
    double c3 = 0.8;
    double w1 = 0.4;
    double w2 = 0.7;
    double w3 = 0.25;

    return exp(- c1*c1*(x - w1)*(x - w1)
               - c2*c2*(y - w2)*(y - w2)
               - c3*c3*(z - w3)*(z - w3));
}

int main()
{
    Elrint3d I3(gaussian, -INFINITY, INFINITY, -INFINITY,
                INFINITY, -INFINITY, INFINITY, 1e-12);
    cout.precision(17);
    cout << "Cubature          = " << I3.evaluate() << endl;
    cout << "Error Flag        = " << I3.errFlag() << endl;
    cout << "Estimated Error = " << I3.estErr() << endl;
    cout << "No of Fun Vals  = " << I3.eval() << endl;
    return 0;
}
```

The output is:

```
Cubature          = 69.604099960396326
Error Flag        = 0
Estimated Error   = 3.5786034818391956e-15
No of Fun Vals    = 526720
```

Note that in this case, the number of function evaluations used is greater than the stated default maximum of 421,736. This is because the routine has tried to evaluate the integral using two different transformations of the infinite domain onto the unit cube. When the first is not successful, the algorithm is automatically restarted using the second transformation, which is successful. The total number of function evaluations reported is the sum of the evaluations in the two attempts.

APPENDIX

A List of source files

1. cubUtil.h – collection of convenient global functions
2. elrint3d.h – interface for creating an *elrint3d* instance with various arguments
3. elrint3dSequence.h – interface for accessing the embedded sequence used by *elrint3d*
4. embeddedCubRule.h – application of a cubature rule to an embedded lattice sequence to produce an integral approximation
5. embeddedSequence.h – generic definition of an embedded lattice sequence
6. infiniteMap.h – mappings from semi-infinite or infinite intervals to $[0,1]$
7. integrand.h – definition of the integrand as a class
8. precomputedCoords.h – interface for accessing the pre-computed coordinates for *elrint3d*
9. elrint3d.cc – implementation of *elrint3d*
10. elrint3dSequence.cc – generation of the embedded sequenced used by *elrint3d*
11. precomputedCoords.cc – the pre-computed coordinates for *elrint3d*

B Makefile

B.1 Makefile for example integrals

For users of GCC, the following makefile is provided for compiling and building the package based on the sample programs given in this document. It compiles successfully with GCC Versions 4.2.4 and 4.1.2.

```
#Generate library filename
LIB=libelrint3d.a
SRC=elrint3d.cc elrint3dSequence.cc precomputedCoords.cc
OBJ=$(SRC:.cc=.o)
AR= ar cq
CXXFLAGS=-Wall -Weffc++ -O3
CPPFLAGS=-I.
LDFLAGS=-L. -lelrint3d
COMPILE.cc=$(CXX) $(CPPFLAGS) $(CXXFLAGS) $(TARGET_ARCH)

#Make binary executable for Examples 1-5
all: $(LIB)
    $(COMPILE.cc) -o test01 test01.cc $(LDFLAGS)
    $(COMPILE.cc) -o test02 test02.cc $(LDFLAGS)
    $(COMPILE.cc) -o test03 test03.cc $(LDFLAGS)
    $(COMPILE.cc) -o test04 test04.cc $(LDFLAGS)
    $(COMPILE.cc) -o test05 test05.cc $(LDFLAGS)

precomputedCoords.o: precomputedCoords.cc precomputedCoords.h
    $(COMPILE.cc) -c $<

elrint3d.o: elrint3d.cc elrint3d.h cubUtil.h infiniteMap.h elrint3dSequence.h\
    embeddedSequence.h embeddedCubRule.h integrand.h
    $(COMPILE.cc) -c $<

elrint3dSequence.o: elrint3dSequence.cc elrint3dSequence.h precomputedCoords.h\
    embeddedSequence.h
    $(COMPILE.cc) -c $<

#Make a library
$(LIB): $(OBJ)
    $(AR) $@ $(OBJ)
    ranlib $@

lib: $(LIB)

#Clean the package
clean:
    rm -rf *.o $(LIB) test01 test02 test03 test04 test05
```

To build all the examples included in the package, type ‘make’ in the command line or ‘make lib’ to build a library named ‘libelrint3d.a’.

B.2 Makefile template

It is straightforward to modify the above makefile to suit individual applications, but the following template is also provided. It is assumed the user will replace the name ‘example’ where it occurs with the name of their own application file.

```
### Example for users
#Assume users want to compile with their own file example.cc
SRC=example.cc
OBJ=$(SRC:.cc=.o)
CXXFLAGS=-Wall -Weffc++ -O3
CPPFLAGS=-I.
LDFLAGS=-L. -lelrint3d

example:    $(OBJ)
            $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) $< -o $@ $(LDFLAGS)

all: example
```