Computational Physics Lectures: Numerical integration, from Newton-Cotes quadrature to Gaussian quadrature

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

2016

Numerical Integration

Here we will discuss some of the classical methods for integrating a function. The methods we discuss are

1. Equal step methods like the trapezoidal, rectangular and Simpson's rule, parts of what are called Newton-Cotes quadrature methods.

2. Integration approaches based on Gaussian quadrature.

The latter are more suitable for the case where the abscissas are not equally spaced. We emphasize methods for evaluating few-dimensional (typically up to four dimensions) integrals. Multi-dimensional integrals will be discussed in connection with Monte Carlo methods.

Newton-Cotes Quadrature or equal-step methods

The integral

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

has a very simple meaning. The integral is the area enscribed by the function $f(x)$ starting from $x = a$ to $x = b$. It is subdivided in several smaller areas whose evaluation is to be approximated by different techniques. The areas under the curve can for example be approximated by rectangular boxes or trapezoids.

Basic philosophy of equal-step methods

In considering equal step methods, our basic approach is that of approximating a function $f(x)$ with a polynomial of at most degree $N-1$, given N integration points. If our polynomial is of degree 1, the function will be approximated with $f(x) \approx a_0 + a_1 x$.

Simple algorithm for equal step methods

The algorithm for these integration methods is rather simple, and the number of approximations perhaps unlimited!

- Choose a step size $h = (b-a)/N$ where N is the number of steps and a and b the lower and upper limits of integration.
- With a given step length we rewrite the integral as

$$\int_a^b f(x)dx = \int_a^{a+h} f(x)dx + \int_{a+h}^{a+2h} f(x)dx + \cdots \int_{b-h}^{b} f(x)dx.$$

- The strategy then is to find a reliable polynomial approximation for $f(x)$ in the various intervals. Choosing a given approximation for $f(x)$, we obtain a specific approximation to the integral.
- With this approximation to $f(x)$ we perform the integration by computing the integrals over all subintervals.

Simple algorithm for equal step methods

One possible strategy then is to find a reliable polynomial expansion for $f(x)$ in the smaller subintervals. Consider for example evaluating

$$\int_a^{a+2h} f(x)dx,$$

which we rewrite as

$$\int_a^{a+2h} f(x)dx = \int_{x_0-h}^{x_0+h} f(x)dx. \tag{2}$$

We have chosen a midpoint x_0 and have defined $x_0 = a + h$.

Lagrange's interpolation formula

Using Lagrange's interpolation formula

$$P_N(x) = \sum_{i=0}^{N} \prod_{k \neq i} \frac{x - x_k}{x_i - x_k} y_i,$$

we could attempt to approximate the function $f(x)$ with a first-order polynomial in x in the two sub-intervals $x \in [x_0 - h, x_0]$ and $x \in [x_0, x_0 + h]$. A first order polynomial means simply that we have for say the interval $x \in [x_0, x_0 + h]$

$$f(x) \approx P_1(x) = \frac{x - x_0}{(x_0 + h) - x_0} f(x_0 + h) + \frac{x - (x_0 + h)}{x_0 - (x_0 + h)} f(x_0),$$

and for the interval $x \in [x_0 - h, x_0]$

$$f(x) \approx P_1(x) = \frac{x - (x_0 - h)}{x_0 - (x_0 - h)} f(x_0) + \frac{x - x_0}{(x_0 - h) - x_0} f(x_0 - h).$$

Polynomial approximation

Having performed this subdivision and polynomial approximation, one from $x_0 - h$ to x_0 and the other from x_0 to $x_0 + h$,

$$\int_a^{a+2h} f(x)dx = \int_{x_0-h}^{x_0} f(x)dx + \int_{x_0}^{x_0+h} f(x)dx,$$

we can easily calculate for example the second integral as

$$\int_{x_0}^{x_0+h} f(x)dx \approx \int_{x_0}^{x_0+h} \left(\frac{x - x_0}{(x_0 + h) - x_0} f(x_0 + h) + \frac{x - (x_0 + h)}{x_0 - (x_0 + h)} f(x_0) \right) dx.$$

Simplifying the integral

This integral can be simplified to

$$\int_{x_0}^{x_0+h} f(x)dx \approx \int_{x_0}^{x_0+h} \left(\frac{x - x_0}{h} f(x_0 + h) - \frac{x - (x_0 + h)}{h} f(x_0) \right) dx,$$

resulting in

$$\int_{x_0}^{x_0+h} f(x)dx = \frac{h}{2} \left(f(x_0 + h) + f(x_0) \right) + O(h^3).$$

Here we added the error made in approximating our integral with a polynomial of degree 1.

The trapezoidal rule

The other integral gives

$$\int_{x_0-h}^{x_0} f(x)dx = \frac{h}{2} \left(f(x_0) + f(x_0 - h) \right) + O(h^3),$$

and adding up we obtain

$$\int_{x_0-h}^{x_0+h} f(x)dx = \frac{h}{2} \left(f(x_0 + h) + 2f(x_0) + f(x_0 - h) \right) + O(h^3), \tag{3}$$

which is the well-known trapezoidal rule. Concerning the error in the approximation made, $O(h^3) = O((b-a)^3/N^3)$, you should note that this is the local error. Since we are splitting the integral from a to b in N pieces, we will have to perform approximately N such operations.

Global error

This means that the *global error* goes like $\approx O(h^2)$. The trapezoidal reads then

$$I = \int_a^b f(x)dx = h\left(f(a)/2 + f(a+h) + f(a+2h) + \cdots + f(b-h) + f_b/2\right), \tag{4}$$

with a global error which goes like $O(h^2)$.

Hereafter we use the shorthand notations $f_{-h} = f(x_0 - h)$, $f_0 = f(x_0)$ and $f_h = f(x_0 + h)$.

Error in the trapezoidal rule

The correct mathematical expression for the local error for the trapezoidal rule is

$$\int_a^b f(x)dx - \frac{b-a}{2}\left[f(a) + f(b)\right] = -\frac{h^3}{12}f^{(2)}(\xi),$$

and the global error reads

$$\int_a^b f(x)dx - T_h(f) = -\frac{b-a}{12}h^2 f^{(2)}(\xi),$$

where T_h is the trapezoidal result and $\xi \in [a, b]$.

Algorithm for the trapezoidal rule

The trapezoidal rule is easy to implement numerically through the following simple algorithm

- Choose the number of mesh points and fix the step length.
- calculate $f(a)$ and $f(b)$ and multiply with $h/2$.
- Perform a loop over $n = 1$ to $n - 1$ ($f(a)$ and $f(b)$ are known) and sum up the terms $f(a+h) + f(a+2h) + f(a+3h) + \cdots + f(b-h)$. Each step in the loop corresponds to a given value $a + nh$.
- Multiply the final result by h and add $hf(a)/2$ and $hf(b)/2$.

Code example

A simple function which implements this algorithm is as follows

```
double TrapezoidalRule(double a, double b, int n, double (*func)(double))
{
      double TrapezSum;
      double fa, fb, x, step;
      int    j;
      step=(b-a)/((double) n);
      fa=(*func)(a)/2. ;
      fb=(*func)(b)/2. ;
      TrapezSum=0.;
      for (j=1; j <= n-1; j++){
         x=j*step+a;
         TrapezSum+=(*func)(x);
      }
      TrapezSum=(TrapezSum+fb+fa)*step;
      return TrapezSum;
}  // end TrapezoidalRule
```

The function returns a new value for the specific integral through the variable **TrapezSum**.

Transfer of function names

There is one new feature to note here, namely the transfer of a user defined function called **func** in the definition

```
void TrapezoidalRule(double a, double b, int n, double *TrapezSum, double (*func)(double) )
```

What happens here is that we are transferring a pointer to the name of a user defined function, which has as input a double precision variable and returns a double precision number. The function **TrapezoidalRule** is called as

```
TrapezoidalRule(a, b, n, &MyFunction )
```

in the calling function. We note that **a**, **b** and **n** are called by value, while **TrapezSum** and the user defined function **MyFunction** are called by reference.

Going back to Python, why?

Symbolic calculations and numerical calculations in one code! Python offers an extremely versatile programming environment, allowing for the inclusion of analytical studies in a numerical program. Here we show an example code with the **trapezoidal rule** using **SymPy** to evaluate an integral and compute the absolute error with respect to the numerically evaluated one of the integral $4\int_0^1 dx/(1+x^2) = \pi$:

```
from math import *
from sympy import *
def Trapez(a,b,f,n):
   h = (b-a)/float(n)
   s = 0
   x = a
```

```
   for i in range(1,n,1):
       x = x+h
       s = s+ f(x)
   s = 0.5*(f(a)+f(b)) +s
   return h*s

#  function to compute pi
def function(x):
    return 4.0/(1+x*x)

a = 0.0;  b = 1.0; n = 100
result = Trapez(a,b,function,n)
print "Trapezoidal rule=", result
# define x as a symbol to be used by sympy
x = Symbol('x')
exact = integrate(function(x), (x, 0.0, 1.0))
print "Sympy integration=", exact
# Find relative error
print "Relative error", abs((exact-result)/exact)
```

Error analysis

The following extended version of the trapezoidal rule allows you to plot the relative error by comparing with the exact result. By increasing to 10^8 points one arrives at a region where numerical errors start to accumulate.

```
from math import log10
import numpy as np
from sympy import Symbol, integrate
import matplotlib.pyplot as plt
# function for the trapezoidal rule
def Trapez(a,b,f,n):
   h = (b-a)/float(n)
   s = 0
   x = a
   for i in range(1,n,1):
       x = x+h
       s = s+ f(x)
   s = 0.5*(f(a)+f(b)) +s
   return h*s
#  function to compute pi
def function(x):
    return 4.0/(1+x*x)
# define integration limits
a = 0.0;  b = 1.0;
# find result from sympy
# define x as a symbol to be used by sympy
x = Symbol('x')
exact = integrate(function(x), (x, a, b))
# set up the arrays for plotting the relative error
n = np.zeros(9); y = np.zeros(9);
# find the relative error as function of integration points
for i in range(1, 8, 1):
    npts = 10**i
    result = Trapez(a,b,function,npts)
    RelativeError = abs((exact-result)/exact)
    n[i] = log10(npts); y[i] = log10(RelativeError);
plt.plot(n,y, 'ro')
plt.xlabel('n')
```

```
plt.ylabel('Relative error')
plt.show()
```

Integrating numerical mathematics with calculus

The last example shows the potential of combining numerical algorithms with symbolic calculations, allowing us thereby to

- Validate and verify our algorithms.
- Including concepts like unit testing, one has the possibility to test and validate several or all parts of the code.
- Validation and verification are then included *naturally*.
- The above example allows you to test the mathematical error of the algorithm for the trapezoidal rule by changing the number of integration points. You get trained from day one to think error analysis.

The rectangle method

Another very simple approach is the so-called midpoint or rectangle method. In this case the integration area is split in a given number of rectangles with length h and height given by the mid-point value of the function. This gives the following simple rule for approximating an integral

$$I = \int_a^b f(x)dx \approx h\sum_{i=1}^{N} f(x_{i-1/2}), \tag{5}$$

where $f(x_{i-1/2})$ is the midpoint value of f for a given rectangle. We will discuss its truncation error below. It is easy to implement this algorithm, as shown here

```
double RectangleRule(double a, double b, int n, double (*func)(double))
{
      double RectangleSum;
      double fa, fb, x, step;
      int    j;
      step=(b-a)/((double) n);
      RectangleSum=0.;
      for (j = 0; j <= n; j++){
         x = (j+0.5)*step+;   // midpoint of a given rectangle
         RectangleSum+=(*func)(x);   //  add value of function.
      }
      RectangleSum *= step;  //  multiply with step length.
      return RectangleSum;
}  // end RectangleRule
```

Truncation error for the rectangular rule

The correct mathematical expression for the local error for the rectangular rule $R_i(h)$ for element i is

$$\int_{-h}^{h} f(x)dx - R_i(h) = -\frac{h^3}{24}f^{(2)}(\xi),$$

and the global error reads

$$\int_{a}^{b} f(x)dx - R_h(f) = -\frac{b-a}{24}h^2 f^{(2)}(\xi),$$

where R_h is the result obtained with rectangular rule and $\xi \in [a, b]$.

Second-order polynomial

Instead of using the above first-order polynomials approximations for f, we attempt at using a second-order polynomials. In this case we need three points in order to define a second-order polynomial approximation

$$f(x) \approx P_2(x) = a_0 + a_1 x + a_2 x^2.$$

Using again Lagrange's interpolation formula we have

$$P_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0.$$

Inserting this formula in the integral of Eq. (2) we obtain

$$\int_{-h}^{+h} f(x)dx = \frac{h}{3}\left(f_h + 4f_0 + f_{-h}\right) + O(h^5),$$

which is Simpson's rule.

Simpson's rule

Note that the improved accuracy in the evaluation of the derivatives gives a better error approximation, $O(h^5)$ vs. $O(h^3)$. But this is again the *local error approximation*. Using Simpson's rule we can easily compute the integral of Eq. (1) to be

$$I = \int_{a}^{b} f(x)dx = \frac{h}{3}\left(f(a) + 4f(a+h) + 2f(a+2h) + \cdots + 4f(b-h) + f_b\right), \tag{6}$$

with a global error which goes like $O(h^4)$.

Mathematical expressions for the truncation error

More formal expressions for the local and global errors are for the local error

$$\int_a^b f(x)dx - \frac{b-a}{6}\left[f(a) + 4f((a+b)/2) + f(b)\right] = -\frac{h^5}{90}f^{(4)}(\xi),$$

and for the global error

$$\int_a^b f(x)dx - S_h(f) = -\frac{b-a}{180}h^4 f^{(4)}(\xi).$$

with $\xi \in [a,b]$ and S_h the results obtained with Simpson's method.

Algorithm for Simpson's rule

The method can easily be implemented numerically through the following simple algorithm

- Choose the number of mesh points and fix the step.
- calculate $f(a)$ and $f(b)$
- Perform a loop over $n = 1$ to $n-1$ ($f(a)$ and $f(b)$ are known) and sum up the terms $4f(a+h) + 2f(a+2h) + 4f(a+3h) + \cdots + 4f(b-h)$. Each step in the loop corresponds to a given value $a + nh$. Odd values of n give 4 as factor while even values yield 2 as factor.
- Multiply the final result by $\frac{h}{3}$.

Summary for equal-step methods

In more general terms, what we have done here is to approximate a given function $f(x)$ with a polynomial of a certain degree. One can show that given $n+1$ distinct points $x_0, \ldots, x_n \in [a,b]$ and $n+1$ values $y_0, \ldots, y_n$ there exists a unique polynomial $P_n(x)$ with the property

$$P_n(x_j) = y_j \qquad j = 0, \ldots, n$$

Lagrange's polynomial

In the Lagrange representation the interpolating polynomial is given by

$$P_n = \sum_{k=0}^{n} l_k y_k,$$

with the Lagrange factors

$$l_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i} \quad k = 0, \ldots, n.$$

Polynomial approximation

If we for example set $n = 1$, we obtain

$$P_1(x) = y_0\frac{x - x_1}{x_0 - x_1} + y_1\frac{x - x_0}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0}x - \frac{y_1x_0 + y_0x_1}{x_1 - x_0},$$

which we recognize as the equation for a straight line.

The polynomial interpolatory quadrature of order n with equidistant quadrature points $x_k = a + kh$ and step $h = (b - a)/n$ is called the Newton-Cotes quadrature formula of order n.

Gaussian Quadrature

The methods we have presented hitherto are tailored to problems where the mesh points x_i are equidistantly spaced, x_i differing from x_{i+1} by the step h.

The basic idea behind all integration methods is to approximate the integral

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^{N} \omega_i f(x_i),$$

where ω and x are the weights and the chosen mesh points, respectively. In our previous discussion, these mesh points were fixed at the beginning, by choosing a given number of points N. The weigths ω resulted then from the integration method we applied. Simpson's rule, see Eq. (6) would give

$$\omega : \{h/3, 4h/3, 2h/3, 4h/3, \ldots, 4h/3, h/3\},$$

for the weights, while the trapezoidal rule resulted in

$$\omega : \{h/2, h, h, \ldots, h, h/2\}.$$

Gaussian Quadrature, main idea

In general, an integration formula which is based on a Taylor series using N points, will integrate exactly a polynomial P of degree $N - 1$. That is, the N weights ω_n can be chosen to satisfy N linear equations, see chapter 3 of Ref. [3]. A greater precision for a given amount of numerical work can be achieved if we are willing to give up the requirement of equally spaced integration points. In Gaussian quadrature (hereafter GQ), both the mesh points and the weights are to be determined. The points will not be equally spaced.

The theory behind GQ is to obtain an arbitrary weight ω through the use of so-called orthogonal polynomials. These polynomials are orthogonal in some interval say e.g., [-1,1]. Our points x_i are chosen in some optimal sense subject only to the constraint that they should lie in this interval. Together with the weights we have then $2N$ (N the number of points) parameters at our disposal.

Gaussian Quadrature

Even though the integrand is not smooth, we could render it smooth by extracting from it the weight function of an orthogonal polynomial, i.e., we are rewriting

$$I = \int_a^b f(x)dx = \int_a^b W(x)g(x)dx \approx \sum_{i=1}^{N} \omega_i g(x_i), \tag{7}$$

where g is smooth and W is the weight function, which is to be associated with a given orthogonal polynomial. Note that with a given weight function we end up evaluating the integrand for the function $g(x_i)$.

Gaussian Quadrature, weight function

The weight function W is non-negative in the integration interval $x \in [a, b]$ such that for any $n \geq 0$, the integral $\int_a^b |x|^n W(x)dx$ is integrable. The naming weight function arises from the fact that it may be used to give more emphasis to one part of the interval than another. A quadrature formula

$$\int_a^b W(x)f(x)dx \approx \sum_{i=1}^{N} \omega_i f(x_i), \tag{8}$$

with N distinct quadrature points (mesh points) is a called a Gaussian quadrature formula if it integrates all polynomials $p \in P_{2N-1}$ exactly, that is

$$\int_a^b W(x)p(x)dx = \sum_{i=1}^{N} \omega_i p(x_i), \tag{9}$$

It is assumed that $W(x)$ is continuous and positive and that the integral

$$\int_a^b W(x)dx$$

exists. Note that the replacement of $f \to Wg$ is normally a better approximation due to the fact that we may isolate possible singularities of W and its derivatives at the endpoints of the interval.

Gaussian Quadrature weights and integration points

The quadrature weights or just weights (not to be confused with the weight function) are positive and the sequence of Gaussian quadrature formulae is convergent if the sequence Q_N of quadrature formulae

$$Q_N(f) \to Q(f) = \int_a^b f(x)dx,$$

in the limit $N \to \infty$.

Gaussian Quadrature

Then we say that the sequence

$$Q_N(f) = \sum_{i=1}^{N} \omega_i^{(N)} f(x_i^{(N)}),$$

is convergent for all polynomials p, that is

$$Q_N(p) = Q(p)$$

if there exits a constant C such that

$$\sum_{i=1}^{N} |\omega_i^{(N)}| \leq C,$$

for all N which are natural numbers.

Error in Gaussian Quadrature

The error for the Gaussian quadrature formulae of order N is given by

$$\int_a^b W(x)f(x)dx - \sum_{k=1}^{N} w_k f(x_k) = \frac{f^{2N}(\xi)}{(2N)!} \int_a^b W(x)[q_N(x)]^2 dx$$

where q_N is the chosen orthogonal polynomial and ξ is a number in the interval $[a, b]$. We have assumed that $f \in C^{2N}[a, b]$, viz. the space of all real or complex $2N$ times continuously differentiable functions.

Important polynomials in Gaussian Quadrature

In science there are several important orthogonal polynomials which arise from the solution of differential equations. Well-known examples are the Legendre, Hermite, Laguerre and Chebyshev polynomials. They have the following weight functions

Weight function	Interval	Polynomial
$W(x) = 1$	$x \in [-1, 1]$	Legendre
$W(x) = e^{-x^2}$	$-\infty \leq x \leq \infty$	Hermite
$W(x) = x^{\alpha} e^{-x}$	$0 \leq x \leq \infty$	Laguerre
$W(x) = 1/(\sqrt{1-x^2})$	$-1 \leq x \leq 1$	Chebyshev

The importance of the use of orthogonal polynomials in the evaluation of integrals can be summarized as follows.

Gaussian Quadrature, win-win situation

Methods based on Taylor series using N points will integrate exactly a polynomial P of degree $N-1$. If a function $f(x)$ can be approximated with a polynomial of degree $N-1$

$$f(x) \approx P_{N-1}(x),$$

with N mesh points we should be able to integrate exactly the polynomial P_{N-1}.

Gaussian quadrature methods promise more than this. We can get a better polynomial approximation with order greater than N to $f(x)$ and still get away with only N mesh points. More precisely, we approximate

$$f(x) \approx P_{2N-1}(x),$$

and with only N mesh points these methods promise that

$$\int f(x)dx \approx \int P_{2N-1}(x)dx = \sum_{i=0}^{N-1} P_{2N-1}(x_i)\omega_i,$$

Gaussian Quadrature, determining mesh points and weights

The reason why we can represent a function $f(x)$ with a polynomial of degree $2N-1$ is due to the fact that we have $2N$ equations, N for the mesh points and N for the weights.

The mesh points are the zeros of the chosen orthogonal polynomial of order N, and the weights are determined from the inverse of a matrix. An orthogonal polynomials of degree N defined in an interval $[a,b]$ has precisely N distinct zeros on the open interval (a,b).

Before we detail how to obtain mesh points and weights with orthogonal polynomials, let us revisit some features of orthogonal polynomials by specializing to Legendre polynomials. In the text below, we reserve hereafter the labelling L_N for a Legendre polynomial of order N, while P_N is an arbitrary polynomial of order N. These polynomials form then the basis for the Gauss-Legendre method.

Orthogonal polynomials, Legendre

The Legendre polynomials are the solutions of an important differential equation in Science, namely

$$C(1-x^2)P - m_l^2 P + (1-x^2)\frac{d}{dx}\left((1-x^2)\frac{dP}{dx}\right) = 0.$$

Here C is a constant. For $m_l = 0$ we obtain the Legendre polynomials as solutions, whereas $m_l \neq 0$ yields the so-called associated Legendre polynomials. This differential equation arises in for example the solution of the angular dependence of Schroedinger's equation with spherically symmetric potentials such as the Coulomb potential.

Orthogonal polynomials, Legendre

The corresponding polynomials P are

$$L_k(x) = \frac{1}{2^k k!}\frac{d^k}{dx^k}(x^2-1)^k \qquad k = 0, 1, 2, \ldots,$$

which, up to a factor, are the Legendre polynomials L_k. The latter fulfil the orthogonality relation

$$\int_{-1}^{1} L_i(x)L_j(x)dx = \frac{2}{2i+1}\delta_{ij}, \tag{10}$$

and the recursion relation

$$(j+1)L_{j+1}(x) + jL_{j-1}(x) - (2j+1)xL_j(x) = 0. \tag{11}$$

Orthogonal polynomials, Legendre

It is common to choose the normalization condition

$$L_N(1) = 1.$$

With these equations we can determine a Legendre polynomial of arbitrary order with input polynomials of order $N-1$ and $N-2$.

As an example, consider the determination of L_0, L_1 and L_2. We have that

$$L_0(x) = c,$$

with c a constant. Using the normalization equation $L_0(1) = 1$ we get that

$$L_0(x) = 1.$$

Orthogonal polynomials, Legendre

For $L_1(x)$ we have the general expression

$$L_1(x) = a + bx,$$

and using the orthogonality relation

$$\int_{-1}^{1} L_0(x)L_1(x)dx = 0,$$

we obtain $a = 0$ and with the condition $L_1(1) = 1$, we obtain $b = 1$, yielding

$$L_1(x) = x.$$

Orthogonal polynomials, Legendre

We can proceed in a similar fashion in order to determine the coefficients of L_2

$$L_2(x) = a + bx + cx^2,$$

using the orthogonality relations

$$\int_{-1}^{1} L_0(x)L_2(x)dx = 0,$$

and

$$\int_{-1}^{1} L_1(x)L_2(x)dx = 0,$$

and the condition $L_2(1) = 1$ we would get

$$L_2(x) = \frac{1}{2}\left(3x^2 - 1\right). \tag{12}$$

Orthogonal polynomials, Legendre

We note that we have three equations to determine the three coefficients a, b and c.

Alternatively, we could have employed the recursion relation of Eq. (11), resulting in

$$2L_2(x) = 3xL_1(x) - L_0,$$

which leads to Eq. (12).

Orthogonal polynomials, Legendre

The orthogonality relation above is important in our discussion on how to obtain the weights and mesh points. Suppose we have an arbitrary polynomial Q_{N-1} of order $N-1$ and a Legendre polynomial $L_N(x)$ of order N. We could represent Q_{N-1} by the Legendre polynomials through

$$Q_{N-1}(x) = \sum_{k=0}^{N-1} \alpha_k L_k(x), \tag{13}$$

where α_k's are constants.

Using the orthogonality relation of Eq. (10) we see that

$$\int_{-1}^{1} L_N(x)Q_{N-1}(x)dx = \sum_{k=0}^{N-1} \int_{-1}^{1} L_N(x)\alpha_k L_k(x)dx = 0. \tag{14}$$

We will use this result in our construction of mesh points and weights in the next subsection.

Orthogonal polynomials, Legendre

In summary, the first few Legendre polynomials are

$$L_0(x) = 1,$$

$$L_1(x) = x,$$

$$L_2(x) = (3x^2 - 1)/2,$$

$$L_3(x) = (5x^3 - 3x)/2,$$

and

$$L_4(x) = (35x^4 - 30x^2 + 3)/8.$$

Orthogonal polynomials, simple code for Legendre polynomials

The following simple function implements the above recursion relation of Eq. (11). for computing Legendre polynomials of order N.

```
//  This function computes the Legendre polynomial of degree N

double Legendre( int n, double x)
{
       double r, s, t;
       int m;
       r = 0; s = 1.;
       //  Use recursion relation to generate p1 and p2
       for (m=0; m < n; m++ )
       {
          t = r; r = s;
          s = (2*m+1)*x*r - m*t;
          s /= (m+1);
        } // end of do loop
        return s;
}   // end of function Legendre
```

The variable s represents $L_{j+1}(x)$, while r holds $L_j(x)$ and t the value $L_{j-1}(x)$.

Integration points and weights with orthogonal polynomials

To understand how the weights and the mesh points are generated, we define first a polynomial of degree $2N-1$ (since we have $2N$ variables at hand, the mesh points and weights for N points). This polynomial can be represented through polynomial division by

$$P_{2N-1}(x) = L_N(x)P_{N-1}(x) + Q_{N-1}(x),$$

where $P_{N-1}(x)$ and $Q_{N-1}(x)$ are some polynomials of degree $N-1$ or less. The function $L_N(x)$ is a Legendre polynomial of order N.

Recall that we wanted to approximate an arbitrary function $f(x)$ with a polynomial P_{2N-1} in order to evaluate

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} P_{2N-1}(x)dx.$$

Integration points and weights with orthogonal polynomials

We can use Eq. (14) to rewrite the above integral as

$$\int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} (L_N(x)P_{N-1}(x) + Q_{N-1}(x))dx = \int_{-1}^{1} Q_{N-1}(x)dx,$$

due to the orthogonality properties of the Legendre polynomials. We see that it suffices to evaluate the integral over $\int_{-1}^{1} Q_{N-1}(x)dx$ in order to evaluate $\int_{-1}^{1} P_{2N-1}(x)dx$. In addition, at the points x_k where L_N is zero, we have

$$P_{2N-1}(x_k) = Q_{N-1}(x_k) \qquad k = 0, 1, \ldots, N-1,$$

and we see that through these N points we can fully define $Q_{N-1}(x)$ and thereby the integral. Note that we have chosen to let the numbering of the points run from 0 to $N-1$. The reason for this choice is that we wish to have the same numbering as the order of a polynomial of degree $N-1$. This numbering will be useful below when we introduce the matrix elements which define the integration weights w_i.

Integration points and weights with orthogonal polynomials

We develope then $Q_{N-1}(x)$ in terms of Legendre polynomials, as done in Eq. (13),

$$Q_{N-1}(x) = \sum_{i=0}^{N-1} \alpha_i L_i(x). \tag{15}$$

Using the orthogonality property of the Legendre polynomials we have

$$\int_{-1}^{1} Q_{N-1}(x)dx = \sum_{i=0}^{N-1} \alpha_i \int_{-1}^{1} L_0(x)L_i(x)dx = 2\alpha_0,$$

where we have just inserted $L_0(x) = 1$!

Integration points and weights with orthogonal polynomials

Instead of an integration problem we need now to define the coefficient α_0. Since we know the values of Q_{N-1} at the zeros of L_N, we may rewrite Eq. (15) as

$$Q_{N-1}(x_k) = \sum_{i=0}^{N-1} \alpha_i L_i(x_k) = \sum_{i=0}^{N-1} \alpha_i L_{ik} \qquad k = 0, 1, \ldots, N-1. \tag{16}$$

Since the Legendre polynomials are linearly independent of each other, none of the columns in the matrix L_{ik} are linear combinations of the others.

Integration points and weights with orthogonal polynomials

This means that the matrix L_{ik} has an inverse with the properties

$$\hat{L}^{-1}\hat{L} = \hat{I}.$$

Multiplying both sides of Eq. (16) with $\sum_{j=0}^{N-1} L_{ji}^{-1}$ results in

$$\sum_{i=0}^{N-1} (L^{-1})_{ki} Q_{N-1}(x_i) = \alpha_k. \tag{17}$$

Integration points and weights with orthogonal polynomials

We can derive this result in an alternative way by defining the vectors

$$\hat{x}_k = \begin{pmatrix} x_0 \\ x_1 \\ . \\ . \\ x_{N-1} \end{pmatrix} \quad \hat{\alpha} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ . \\ . \\ \alpha_{N-1} \end{pmatrix},$$

and the matrix

$$\hat{L} = \begin{pmatrix} L_0(x_0) & L_1(x_0) & \dots & L_{N-1}(x_0) \\ L_0(x_1) & L_1(x_1) & \dots & L_{N-1}(x_1) \\ \dots & \dots & \dots & \dots \\ L_0(x_{N-1}) & L_1(x_{N-1}) & \dots & L_{N-1}(x_{N-1}) \end{pmatrix}.$$

Integration points and weights with orthogonal polynomials

We have then

$$Q_{N-1}(\hat{x}_k) = \hat{L}\hat{\alpha},$$

yielding (if $\hat{L}$ has an inverse)

$$\hat{L}^{-1} Q_{N-1}(\hat{x}_k) = \hat{\alpha},$$

which is Eq. (17).

Integration points and weights with orthogonal polynomials

Using the above results and the fact that

$$\int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx,$$

we get

$$\int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx = 2\alpha_0 = 2\sum_{i=0}^{N-1} (L^{-1})_{0i} P_{2N-1}(x_i).$$

Integration points and weights with orthogonal polynomials

If we identify the weights with $2(L^{-1})_{0i}$, where the points x_i are the zeros of L_N, we have an integration formula of the type

$$\int_{-1}^{1} P_{2N-1}(x)dx = \sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i)$$

and if our function $f(x)$ can be approximated by a polynomial P of degree $2N-1$, we have finally that

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} P_{2N-1}(x)dx = \sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i).$$

In summary, the mesh points x_i are defined by the zeros of an orthogonal polynomial of degree N, that is L_N, while the weights are given by $2(L^{-1})_{0i}$.

Application to the case $N = 2$

Let us apply the above formal results to the case $N = 2$. This means that we can approximate a function $f(x)$ with a polynomial $P_3(x)$ of order $2N - 1 = 3$.

The mesh points are the zeros of $L_2(x) = 1/2(3x^2 - 1)$. These points are $x_0 = -1/\sqrt{3}$ and $x_1 = 1/\sqrt{3}$.

Specializing Eq. (16)

$$Q_{N-1}(x_k) = \sum_{i=0}^{N-1} \alpha_i L_i(x_k) \qquad k = 0, 1, \ldots, N-1.$$

to $N = 2$ yields

$$Q_1(x_0) = \alpha_0 - \alpha_1 \frac{1}{\sqrt{3}},$$

and

$$Q_1(x_1) = \alpha_0 + \alpha_1 \frac{1}{\sqrt{3}},$$

since $L_0(x = \pm 1/\sqrt{3}) = 1$ and $L_1(x = \pm 1/\sqrt{3}) = \pm 1/\sqrt{3}$.

Application to the case $N = 2$

The matrix L_{ik} defined in Eq. (16) is then

$$\hat{L} = \begin{pmatrix} 1 & -\frac{1}{\sqrt{3}} \\ 1 & \frac{1}{\sqrt{3}} \end{pmatrix},$$

with an inverse given by

$$\hat{L}^{-1} = \frac{\sqrt{3}}{2} \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -1 & 1 \end{pmatrix}.$$

The weights are given by the matrix elements $2(L_{0k})^{-1}$. We have thence $\omega_0 = 1$ and $\omega_1 = 1$.

Application to the case $N = 2$

Obviously, there is no problem in changing the numbering of the matrix elements $i, k = 0, 1, 2, \ldots, N-1$ to $i, k = 1, 2, \ldots, N$. We have chosen to start from zero, since we deal with polynomials of degree $N - 1$.

Summarizing, for Legendre polynomials with $N = 2$ we have weights

$$\omega : \{1, 1\},$$

and mesh points

$$x : \left\{ -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}} \right\}.$$

Application to the case $N = 2$

If we wish to integrate

$$\int_{-1}^{1} f(x)dx,$$

with $f(x) = x^2$, we approximate

$$I = \int_{-1}^{1} x^2 dx \approx \sum_{i=0}^{N-1} \omega_i x_i^2.$$

Application to the case $N = 2$

The exact answer is 2/3. Using $N = 2$ with the above two weights and mesh points we get

$$I = \int_{-1}^{1} x^2 dx = \sum_{i=0}^{1} \omega_i x_i^2 = \frac{1}{3} + \frac{1}{3} = \frac{2}{3},$$

the exact answer!

If we were to emply the trapezoidal rule we would get

$$I = \int_{-1}^{1} x^2 dx = \frac{b-a}{2} \left((a)^2 + (b)^2 \right) /2 = \frac{1-(-1)}{2} \left((-1)^2 + (1)^2 \right) /2 = 1!$$

With just two points we can calculate exactly the integral for a second-order polynomial since our methods approximates the exact function with higher order polynomial. How many points do you need with the trapezoidal rule in order to achieve a similar accuracy?

General integration intervals for Gauss-Legendre

Note that the Gauss-Legendre method is not limited to an interval [-1,1], since we can always through a change of variable

$$t = \frac{b-a}{2}x + \frac{b+a}{2},$$

rewrite the integral for an interval [a,b]

$$\int_a^b f(t)dt = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{(b-a)x}{2} + \frac{b+a}{2}\right)dx.$$

Mapping integration points and weights

If we have an integral on the form

$$\int_0^\infty f(t)dt,$$

we can choose new mesh points and weights by using the mapping

$$\tilde{x}_i = tan\left\{\frac{\pi}{4}(1+x_i)\right\},$$

and

$$\tilde{\omega}_i = \frac{\pi}{4}\frac{\omega_i}{cos^2\left(\frac{\pi}{4}(1+x_i)\right)},$$

where x_i and ω_i are the original mesh points and weights in the interval $[-1,1]$, while $\tilde{x}_i$ and $\tilde{\omega}_i$ are the new mesh points and weights for the interval $[0,\infty)$.

Mapping integration points and weights

To see that this is correct by inserting the the value of $x_i = -1$ (the lower end of the interval $[-1,1]$) into the expression for $\tilde{x}_i$. That gives $\tilde{x}_i = 0$, the lower end of the interval $[0,\infty)$. For $x_i = 1$, we obtain $\tilde{x}_i = \infty$. To check that the new weights are correct, recall that the weights should correspond to the derivative of the mesh points. Try to convince yourself that the above expression fulfills this condition.

Other orthogonal polynomials, Laguerre polynomials

If we are able to rewrite our integral of Eq. (7) with a weight function $W(x) = x^\alpha e^{-x}$ with integration limits $[0,\infty)$, we could then use the Laguerre polynomials. The polynomials form then the basis for the Gauss-Laguerre method which can be applied to integrals of the form

$$I = \int_0^\infty f(x)dx = \int_0^\infty x^\alpha e^{-x} g(x)dx.$$

Other orthogonal polynomials, Laguerre polynomials

These polynomials arise from the solution of the differential equation

$$\left(\frac{d^2}{dx^2}-\frac{d}{dx}+\frac{\lambda}{x}-\frac{l(l+1)}{x^2}\right)\mathcal{L}(x)=0,$$

where l is an integer $l\geq 0$ and λ a constant. This equation arises for example from the solution of the radial Schrödinger equation with a centrally symmetric potential such as the Coulomb potential.

Other orthogonal polynomials, Laguerre polynomials

The first few polynomials are

$$\mathcal{L}_0(x)=1,$$

$$\mathcal{L}_1(x)=1-x,$$

$$\mathcal{L}_2(x)=2-4x+x^2,$$

$$\mathcal{L}_3(x)=6-18x+9x^2-x^3,$$

and

$$\mathcal{L}_4(x)=x^4-16x^3+72x^2-96x+24.$$

Other orthogonal polynomials, Laguerre polynomials

They fulfil the orthogonality relation

$$\int_0^\infty e^{-x}\mathcal{L}_n(x)^2dx=1,$$

and the recursion relation

$$(n+1)\mathcal{L}_{n+1}(x)=(2n+1-x)\mathcal{L}_n(x)-n\mathcal{L}_{n-1}(x).$$

Other orthogonal polynomials, Hermite polynomials

In a similar way, for an integral which goes like

$$I=\int_{-\infty}^{\infty}f(x)dx=\int_{-\infty}^{\infty}e^{-x^2}g(x)dx.$$

we could use the Hermite polynomials in order to extract weights and mesh points. The Hermite polynomials are the solutions of the following differential equation

$$\frac{d^2H(x)}{dx^2}-2x\frac{dH(x)}{dx}+(\lambda-1)H(x)=0. \tag{18}$$

Other orthogonal polynomials, Hermite polynomials

A typical example is again the solution of Schrodinger's equation, but this time with a harmonic oscillator potential. The first few polynomials are

$$H_0(x) = 1,$$

$$H_1(x) = 2x,$$

$$H_2(x) = 4x^2 - 2,$$

$$H_3(x) = 8x^3 - 12,$$

and

$$H_4(x) = 16x^4 - 48x^2 + 12.$$

They fulfil the orthogonality relation

$$\int_{-\infty}^{\infty} e^{-x^2} H_n(x)^2 dx = 2^n n! \sqrt{\pi},$$

and the recursion relation

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x).$$

Demonstration of Gaussian Quadrature

Let us here compare three methods for integrating, namely the trapezoidal rule, Simpson's method and the Gauss-Legendre approach. We choose two functions to integrate:

$$\int_1^{100} \frac{\exp(-x)}{x} dx,$$

and

$$\int_0^3 \frac{1}{2+x^2} dx.$$

Demonstration of Gaussian Quadrature, simple program

A program example which uses the trapezoidal rule, Simpson's rule and the Gauss-Legendre method is included here.

```
#include <iostream>
#include "lib.h"
using namespace std;
//     Here we define various functions called by the main program
//     this function defines the function to integrate
double int_function(double x);
//   Main function begins here
int main()
{
     int n;
     double a, b;
```

```
     cout << "Read in the number of integration points" << endl;
     cin >> n;
     cout << "Read in integration limits" << endl;
     cin >> a >> b;
//   reserve space in memory for vectors containing the mesh points
//   weights and function values for the use of the gauss-legendre
//   method
     double *x = new double [n];
     double *w = new double [n];
//   set up the mesh points and weights
     gauss_legendre(a, b,x,w, n);
//   evaluate the integral with the Gauss-Legendre method
//   Note that we initialize the sum
     double int_gauss = 0.;
     for ( int i = 0;  i < n; i++){
        int_gauss+=w[i]*int_function(x[i]);
     }
//    final output
      cout << "Trapez-rule = " << trapezoidal_rule(a, b,n, int_function)
           << endl;
      cout << "Simpson's rule = " << simpson(a, b,n, int_function)
           << endl;
      cout << "Gaussian quad = " << int_gauss << endl;
      delete [] x;
      delete [] w;
      return 0;
}  // end of main program
//  this function defines the function to integrate
double int_function(double x)
{
  double value = 4./(1.+x*x);
  return value;
} // end of function to evaluate
```

Demonstration of Gaussian Quadrature

To be noted in this program is that we can transfer the name of a given function to integrate. In the table here we show the results for the first integral using various mesh points,.

N	Trapez	Simpson	Gauss-Legendre
10	1.821020	1.214025	0.1460448
20	0.912678	0.609897	0.2178091
40	0.478456	0.333714	0.2193834
100	0.273724	0.231290	0.2193839
1000	0.219984	0.219387	0.2193839

We note here that, since the area over where we integrate is rather large and the integrand goes slowly to zero for large values of x, both the trapezoidal rule and Simpson's method need quite many points in order to approach the Gauss-Legendre method. This integrand demonstrates clearly the strength of the Gauss-Legendre method (and other GQ methods as well), viz., few points are needed in order to achieve a very high precision.

Demonstration of Gaussian Quadrature

The second table however shows that for smaller integration intervals, both the trapezoidal rule and Simpson's method compare well with the results obtained with the Gauss-Legendre approach.

N	Trapez	Simpson	Gauss-Legendre
10	0.798861	0.799231	0.799233
20	0.799140	0.799233	0.799233
40	0.799209	0.799233	0.799233
100	0.799229	0.799233	0.799233
1000	0.799233	0.799233	0.799233

Comparing methods and using symbolic Python

The following python code allows you to run interactively either in a browser or using ipython notebook. It compares the trapezoidal rule and Gaussian quadrature with the exact result from symbolic python **SYMPY** up to 1000 integration points for the integral

$$I = 2 = \int_0^\infty x^2 \exp -x dx.$$

For the trapezoidal rule the results will vary strongly depending on how the infinity limit is approximated. Try to run the code below for different finite approximations to ∞.

```
from math import exp
import numpy as np
from sympy import Symbol, integrate, exp, oo

# function for the trapezoidal rule
def TrapezoidalRule(a,b,f,n):
   h = (b-a)/float(n)
   s = 0
   x = a
   for i in range(1,n,1):
       x = x+h
       s = s+ f(x)
   s = 0.5*(f(a)+f(b)) +s
   return h*s

# function for the Gaussian quadrature with Laguerre polynomials
def GaussLaguerreRule(n):
   s = 0
   xgauleg, wgauleg = np.polynomial.laguerre.laggauss(n)
   for i in range(1,n,1):
       s = s+ xgauleg[i]*xgauleg[i]*wgauleg[i]
   return s

#  function to compute
def function(x):
    return x*x*exp(-x)
```

```
# Integration limits for the Trapezoidal rule
a = 0.0; b = 10000.0
# define x as a symbol to be used by sympy
x = Symbol('x')
# find result from sympy
exact = integrate(function(x), (x, a, oo))
# set up the arrays for plotting the relative error
n = np.zeros(40); Trapez = np.zeros(4); LagGauss = np.zeros(4);
# find the relative error as function of integration points
for i in range(1, 3, 1):
    npts = 10**i
    n[i] = npts
    Trapez[i] = abs((TrapezoidalRule(a,b,function,npts)-exact)/exact)
    LagGauss[i] = abs((GaussLaguerreRule(npts)-exact)/exact)
print "Integration points=", n[1], n[2]
print "Trapezoidal relative error=", Trapez[1], Trapez[2]
print "LagGuass relative error=", LagGauss[1], LagGauss[2]
```

Treatment of Singular Integrals

So-called principal value (PV) integrals are often employed in physics, from Green's functions for scattering to dispersion relations. Dispersion relations are often related to measurable quantities and provide important consistency checks in atomic, nuclear and particle physics. A PV integral is defined as

$$I(x) = \mathcal{P}\int_a^b dt \frac{f(t)}{t-x} = \lim_{\epsilon\to 0^+}\left[\int_a^{x-\epsilon} dt\frac{f(t)}{t-x} + \int_{x+\epsilon}^b dt\frac{f(t)}{t-x}\right],$$

and arises in applications of Cauchy's residue theorem when the pole x lies on the real axis within the interval of integration $[a,b]$. Here $\mathcal{P}$ stands for the principal value. *An important assumption is that the function $f(t)$ is continuous on the interval of integration.*

Treatment of Singular Integrals

In case $f(t)$ is a closed form expression or it has an analytic continuation in the complex plane, it may be possible to obtain an expression on closed form for the above integral.

However, the situation which we are often confronted with is that $f(t)$ is only known at some points t_i with corresponding values $f(t_i)$. In order to obtain $I(x)$ we need to resort to a numerical evaluation.

To evaluate such an integral, let us first rewrite it as

$$\mathcal{P}\int_a^b dt \frac{f(t)}{t-x} = \int_a^{x-\Delta} dt\frac{f(t)}{t-x} + \int_{x+\Delta}^b dt\frac{f(t)}{t-x} + \mathcal{P}\int_{x-\Delta}^{x+\Delta} dt\frac{f(t)}{t-x},$$

where we have isolated the principal value part in the last integral.

Treatment of Singular Integrals, change of variables

Defining a new variable $u = t - x$, we can rewrite the principal value integral as

$$I_\Delta(x) = \mathcal{P}\int_{-\Delta}^{+\Delta} du \frac{f(u+x)}{u}. \tag{19}$$

One possibility is to Taylor expand $f(u+x)$ around $u = 0$, and compute derivatives to a certain order as we did for the Trapezoidal rule or Simpson's rule. Since all terms with even powers of u in the Taylor expansion dissapear, we have that

$$I_\Delta(x) \approx \sum_{n=0}^{N_{max}} f^{(2n+1)}(x)\frac{\Delta^{2n+1}}{(2n+1)(2n+1)!}.$$

Treatment of Singular Integrals, higher-order derivatives

To evaluate higher-order derivatives may be both time consuming and delicate from a numerical point of view, since there is always the risk of loosing precision when calculating derivatives numerically. Unless we have an analytic expression for $f(u+x)$ and can evaluate the derivatives in a closed form, the above approach is not the preferred one.

Rather, we show here how to use the Gauss-Legendre method to compute Eq. (19). Let us first introduce a new variable $s = u/\Delta$ and rewrite Eq. (19) as

$$I_\Delta(x) = \mathcal{P}\int_{-1}^{+1} ds \frac{f(\Delta s + x)}{s}. \tag{20}$$

Treatment of Singular Integrals

The integration limits are now from -1 to 1, as for the Legendre polynomials. The principal value in Eq. (20) is however rather tricky to evaluate numerically, mainly since computers have limited precision. We will here use a subtraction trick often used when dealing with singular integrals in numerical calculations. We introduce first the calculus relation

$$\int_{-1}^{+1} \frac{ds}{s} = 0.$$

It means that the curve $1/(s)$ has equal and opposite areas on both sides of the singular point $s = 0$.

Treatment of Singular Integrals

If we then note that $f(x)$ is just a constant, we have also

$$f(x)\int_{-1}^{+1} \frac{ds}{s} = \int_{-1}^{+1} f(x)\frac{ds}{s} = 0.$$

Subtracting this equation from Eq. (20) yields

$$I_\Delta(x) = \mathcal{P}\int_{-1}^{+1} ds \frac{f(\Delta s + x)}{s} = \int_{-1}^{+1} ds \frac{f(\Delta s + x) - f(x)}{s}, \tag{21}$$

and the integrand is no longer singular since we have that $\lim_{s\to 0}(f(s+x) - f(x)) = 0$ and for the particular case $s = 0$ the integrand is now finite.

Treatment of Singular Integrals

Eq. (21) is now rewritten using the Gauss-Legendre method resulting in

$$\int_{-1}^{+1} ds \frac{f(\Delta s + x) - f(x)}{s} = \sum_{i=1}^{N} \omega_i \frac{f(\Delta s_i + x) - f(x)}{s_i}, \tag{22}$$

where s_i are the mesh points (N in total) and ω_i are the weights.

In the selection of mesh points for a PV integral, it is important to use an even number of points, since an odd number of mesh points always picks $s_i = 0$ as one of the mesh points. The sum in Eq. (22) will then diverge.

Treatment of Singular Integrals

Let us apply this method to the integral

$$I(x) = \mathcal{P}\int_{-1}^{+1} dt \frac{e^t}{t}. \tag{23}$$

The integrand diverges at $x = t = 0$. We rewrite it using Eq. (21) as

$$\mathcal{P}\int_{-1}^{+1} dt \frac{e^t}{t} = \int_{-1}^{+1} \frac{e^t - 1}{t}, \tag{24}$$

since $e^x = e^0 = 1$. With Eq. (22) we have then

$$\int_{-1}^{+1} \frac{e^t - 1}{t} \approx \sum_{i=1}^{N} \omega_i \frac{e^{t_i} - 1}{t_i}. \tag{25}$$

Treatment of Singular Integrals

The exact results is 2.11450175075..... With just two mesh points we recall from the previous subsection that $\omega_1 = \omega_2 = 1$ and that the mesh points are the zeros of $L_2(x)$, namely $x_1 = -1/\sqrt{3}$ and $x_2 = 1/\sqrt{3}$. Setting $N = 2$ and inserting these values in the last equation gives

$$I_2(x = 0) = \sqrt{3}\left(e^{1/\sqrt{3}} - e^{-1/\sqrt{3}}\right) = 2.1129772845.$$

With six mesh points we get even the exact result to the tenth digit

$$I_6(x = 0) = 2.11450175075!$$

Treatment of Singular Integrals

We can repeat the above subtraction trick for more complicated integrands. First we modify the integration limits to $\pm\infty$ and use the fact that

$$\int_{-\infty}^{\infty}\frac{dk}{k-k_0}=\int_{-\infty}^{0}\frac{dk}{k-k_0}+\int_{0}^{\infty}\frac{dk}{k-k_0}=0.$$

A change of variable $u=-k$ in the integral with limits from $-\infty$ to 0 gives

$$\int_{-\infty}^{\infty}\frac{dk}{k-k_0}=\int_{\infty}^{0}\frac{-du}{-u-k_0}+\int_{0}^{\infty}\frac{dk}{k-k_0}=\int_{0}^{\infty}\frac{dk}{-k-k_0}+\int_{0}^{\infty}\frac{dk}{k-k_0}=0.$$

Treatment of Singular Integrals

It means that the curve $1/(k-k_0)$ has equal and opposite areas on both sides of the singular point k_0. If we break the integral into one over positive k and one over negative k, a change of variable $k\to -k$ allows us to rewrite the last equation as

$$\int_{0}^{\infty}\frac{dk}{k^2-k_0^2}=0.$$

Treatment of Singular Integrals

We can use this to express a principal values integral as

$$\mathcal{P}\int_{0}^{\infty}\frac{f(k)dk}{k^2-k_0^2}=\int_{0}^{\infty}\frac{(f(k)-f(k_0))dk}{k^2-k_0^2},\tag{26}$$

where the right-hand side is no longer singular at $k=k_0$, it is proportional to the derivative df/dk, and can be evaluated numerically as any other integral.

Such a trick is often used when evaluating integral equations.

Example of a multidimensional integral

Here we show an example of a multidimensional integral which appears in quantum mechanical calculations.

The ansatz for the wave function for two electrons is given by the product of two $1s$ wave functions as

$$\Psi(\mathbf{r}_1,\mathbf{r}_2)=e^{-\alpha(r_1+r_2)}.$$

The integral we need to solve is the quantum mechanical expectation value of the correlation energy between two electrons, namely

$$I=\int_{-\infty}^{\infty}d\mathbf{r}_1d\mathbf{r}_2e^{-2\alpha(r_1+r_2)}\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|}.$$

The integral has an exact solution $5\pi^2/16=0.19277$.

Parts of code and brute force Gauss-Legendre quadrature

If we use Gaussian quadrature with Legendre polynomials (without rewriting the integral), we have

```
      double *x = new double [N];
      double *w = new double [N];
//    set up the mesh points and weights
      GaussLegendrePoints(a,b,x,w, N);

//    evaluate the integral with the Gauss-Legendre method
//    Note that we initialize the sum
      double int_gauss = 0.;
//    six-double loops
      for (int i=0;i<N;i++){
              for (int j = 0;j<N;j++){
              for (int k = 0;k<N;k++){
              for (int l = 0;l<N;l++){
              for (int m = 0;m<N;m++){
              for (int n = 0;n<N;n++){
         int_gauss+=w[i]*w[j]*w[k]*w[l]*w[m]*w[n]
        *int_function(x[i],x[j],x[k],x[l],x[m],x[n]);
                      }}}}}
         }
```

The function to integrate, code example

```
//  this function defines the function to integrate
double int_function(double x1, double y1, double z1, double x2, double y2, double z2)
{
   double alpha = 2.;
// evaluate the different terms of the exponential
   double exp1=-2*alpha*sqrt(x1*x1+y1*y1+z1*z1);
   double exp2=-2*alpha*sqrt(x2*x2+y2*y2+z2*z2);
   double deno=sqrt(pow((x1-x2),2)+pow((y1-y2),2)+pow((z1-z2),2));
   return exp(exp1+exp2)/deno;
} // end of function to evaluate
```

Laguerre polynomials

Using Legendre polynomials for the Gaussian quadrature is not very efficient. There are several reasons for this:

- You can easily end up in situations where the integrand diverges
- The limits $\pm\infty$ have to be approximated with a finite number

It is very useful here to change to spherical coordinates

$$d\mathbf{r}_1 d\mathbf{r}_2 = r_1^2 dr_1 r_2^2 dr_2 dcos(\theta_1) dcos(\theta_2) d\phi_1 d\phi_2,$$

and

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

with

$$\cos(\beta) = \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2))$$

Laguerre polynomials, the new integrand

This means that our integral becomes

$$I = \int_0^\infty r_1^2 dr_1 \int_0^\infty r_2^2 dr_2 \int_0^\pi dcos(\theta_1) \int_0^\pi dcos(\theta_2) \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-2\alpha(r_1+r_2)}}{r_{12}}$$

where we have defined

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}}$$

with

$$\cos(\beta) = \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2))$$

Laguerre polynomials, new integration rule: Gauss-Laguerre

Our integral is now given by

$$I = \int_0^\infty r_1^2 dr_1 \int_0^\infty r_2^2 dr_2 \int_0^\pi dcos(\theta_1) \int_0^\pi dcos(\theta_2) \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{\exp -2\alpha(r_1+r_2)}{r_{12}}$$

For the angles we need to perform the integrations over $\theta_i \in [0, \pi]$ and $\phi_i \in [0, 2\pi]$. However, for the radial part we can now either use

- Gauss-Legendre wth an appropriate mapping or
- Gauss-Laguerre taking properly care of the integrands involving the $r_i^2 \exp -(2\alpha r_i)$ terms.

Fixing $N = 20$ with Gauss-Legendre

$r_{\max}$	Integral Error	
1.00	0.161419805	0.0313459063
1.50	0.180468967	0.012296744
2.00	0.177065182	0.0157005292
2.50	0.167970694	0.0247950165
3.00	0.156139391	0.0366263199

Fixing $r_{\max} = 2$ with Gauss-Legendre

N	Integral	Error
10	0.129834248	0.0629314631
16	0.167860437	0.0249052742
20	0.177065182	0.0157005292
26	0.183543237	0.00922247353
30	0.185795624	0.00697008738

Results with Gauss-Laguerre

N	Integral	Error
10	0.186457345	0.00630836601
16	0.190113364	0.00265234708
20	0.19108178	0.00168393093
26	0.191831828	0.000933882594
30	0.192113712	0.000651999339

The code that was used to generate these results can be found under the programs link.