Numerical integration

Files for the present tutorial can be found using following links:

- Header file kwad.h
- Source file kwad.cpp

1. Introduction

Numerical approximation of the definite integral is one of the basic algorithms used in the engineering computations. Note that we can only approximate the value of the definite integral as the result of the definite integration is a scalar value (single number). Indefinite integrals can not be computed (solved) using methods discussed during this course.

2. Newton-Cotes quadratures for integration.

The idea of numerical integration is reduced to the selection of the appropriate quadrature for the integrated function $f(x)$. The function $f(x)$ is called the integrand. Dependent on the required accuracy of approximation of the definite integral, several different types of Newton-Cotes quadratures can be choosen, e.g., mid-point, trapezoidal or the Simpson integration rules.

Exercises

1. Write a program which uses trapezoidal rule and approximates the definite integral:

$$I = \int_a^b f(x)dx$$

Integration algorithm, using complex trapezoidal or Simpson quadrature, is already implemented. It can be found in the function `trapez` see file *kwad.cpp*. Header of the function `trapez` has the following form:

```
double trapez(double a, double b, double (*fun)(double x), int n)
```

Arguments `a`, `b` and `n` denote, respectively, the lower `a` and upper `b` borders of integration interval; `n` is the number of sub-division on which the integration interval $b - a$ is divided. Thus, we use the complex quadrature to approximate the value of the definite integral. Note, the third argument. This argument is a pointer to the function. Thanks to this the function `trapez` can be used without modifications to approximate the value of definite integral of arbitrary user defined function `f(x)`. The name of the function (that is a pointer to this function) can be used as `trapez` argument and hence defines the integrand `f(x)`. Below, three different examples of the calls of the function `trapez` are given: - `trapez(a, b, sin, n);`, - `trapez(1, 5, sqrt, 100);`, - `trapez(a, b, MyFunction, 50);` where `MyFunction` is a user specified function defining `f(x)`. For example `MyFunction` can be defined like this:

```
double MyFunction(double x) // the function must be of type double with one ar
{
    return x*x+sin(x);
}
```

To solve Problem 1., the following steps are required: - write a function computing $f(x)$ and a function computing the value of the definite integral analytically for given (simple enough) $f(x)$ (before writing it solve the definite integral by yourself on the paper sheet). Next, place prototypes of both functions before `main()` and include header file `kwad.h` - read from the keyboard (or initially define in the program) `a`, `b` and `n`- number of sub-divisions of `<a,b>` domain - compute the definite integral numerically `cn` and anlytically `ca` by calling appropriate functions defined by you above - compute the error $E_n = |cn - ca|_n$, note E_n is expected to change dependent on selected `n` - write to the file number of sub-divisions `n`, values of the integral `cn`, `ca` and the error E_n

2. Test your program for two functions:

$$f(x) = \frac{1}{x^2}$$

$$f(x) = \frac{1}{x}$$

choosing $a = 1, b = 5$ or $a = 0.1, b = 5$.

3. Extend your program in a way it writes to the file different rows corresponding to $n = 2, 4, 8, ..., 2^m$, where m is user defined value (e.g. $m = 5$).

4. Extend your program by including the Simpson method (the changes are not large you only need additional varialbes to store `cn`,`ca`, E_n obtained using the Simpson method). The Simpson method is implemented in files `kwad.h`, `kwad.cpp` as well. Its prototype `simpson(double a, double b, double(*fun)(double x), int n)`.

5. Compare on the single diagram errors obtained using `trapez` and `simpson` methods. What can you say about the rate of change (convergence rate) of their errors ?

More about pointers to the function

Analyze the code below. How values of `y1` and `y2` variables are changing during the code execution ?

```
//function: y = 2*x
double fun1(double x)
{
    return 2*x;
}

// function: y = -x
double fun2(double x)
{
    return -x;
}

// function returns y^2
double kwadrat(double xx, double (*pf)(double))
{
    return pf(xx)*pf(xx);
}

void main()
{
    double y1 = kwadrat(2., fun1);
    double y2 = kwadrat(2., fun2);
}
```

3. Spectral integration

The trapezoidal and Simpson rules are both based on approximation of the integrand by the Lagrange interpolation polynomial. More accurate integration methods are available. One of the examples of very accurate numerical integration techniques is the the Gauss-Legendre method (GLM).

GLM in its original form is defined for the definite integral on the interval $[-1, 1]$:

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

This is not a problem as from the lectures you know, each definite integral on the interval $[a, b]$ can be transformed into integral on interval $[-1, 1]$. The value of this definite integral is approximated using formula:

$$I = \int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where w_i are weights and x_i are nodes of the Gauss-Legendre quadrature, n denotes number of the nodes on which the integrand value $f(x_i)$ is evaluated. To approximate the value of the definite integral using GLM the information about the qudarature nodes x_i distribution and weights values w_i must be known beforehand. They can be computed (see Lecture notes) or found in the internet[1]. In our case, they are given in the table below for $n = 5$:

x_i	w_i
-0.9061798459386639927976269	0.2369268850561890875142640
-0.5384693101056830910363144	0.4786286704993664680412915
0.0	0.5688888888888888888888889
0.5384693101056830910363144	0.4786286704993664680412915
0.9061798459386639927976269	0.2369268850561890875142640

Exercise

Implement the Gauss-Legendre method (you can do it in a single loop without function, directly in the main program). Compare the results obtained using GLM

[1] try to google „Legendre Gauss nodes and weights” or check here

with previous results, how many sub-divisions `n` are required in trapezoidal/Simpson methods to reduce the error to the level obtained by the Gauss-Legendre method using only five nodes ?