Norwegian University of Science
and Technology
Department of Mathematical
Sciences

# TMA4215 Numerical mathematics Autumn 2015

# Project 2

Last update: October 27, 2015

## Instructions

This project counts for 20% of the final grade in the course.

**Deadline:** November 17, 23:59.

**Group size:** Up to 4 students.

**To be handed in:**

- A report of maximum 6 pages, using the given LaTeX-template (or as similar as possible if another word-processing system is used). The report should be submitted as a pdf-file.
- One well structured, well documented and self-contained MATLAB file.

Send the files (maximum 2) to `lars.odsater@math.ntnu.no` within the deadline.

Use the **student numbers** (no name, no candidate number) in the report, and make sure that we can identify your student numbers from the MATLAB file.

Failure to meet the instruction above may cause the project to be dismissed!

**Some comments and advices**

- In the evaluation, quite some emphasis will be given to the presentation. If it turns out that you are not able to fulfill the project completely, or your final program do not work, then focus on what you have done. We advice you to stop doing new stuff on Sunday 15th, and spend the last days on fine-tuning the report and the MATLAB file to be handed in.
- Equally important as obtaining the actual results, are to discuss them properly. The lecture notes [2] and text book [1] should be sufficient to do the project, but you are encouraged to use other sources of information as well (e.g. Wikipedia, Google or Google Scholar). Remember to always refer to the source when used.

- We do not want you to present all the tasks in your MATLAB file. What we want is to be able to reproduce the last result(s) you present in your report with that program. This means a program that produce the error plots referred to in Task 11. If you are not able to finish the last task, then your enclosed MATLAB file may solve one of the subtasks.
- With a well-documented and self-contained MATLAB file we mean that the file
  - includes sufficient information in the initial comment lines to make it clear for the user what the program does, and how to use it. This information should be available by writing `help filename`.
  - executes and provides the expected results without any problems. In particular this means that all subfunctions you write must be included in this file.
- The tasks described below are there to help you learn and master the material. They should be done, but the results should in general not be included in the report as **Task 1**, **Task 2**, etc. However, you may want to include some of them as examples, or to justify that the described procedure works.
- An important part of implementing numerical methods is to test the code. This can be done by testing on simple problems where you know the solution. An example is that quadrature rules are exact for some problems, e.g., polynomials of a certain degree. You should perform these kind of tests for your own verification, and may include some of them in your report to demonstrate that your code and algorithms work.

The LaTeX template, including some instruction on how to write the report, will be provided on the project webpage.

## Objective of the project

The topics for this project is numerical quadrature and interpolation. You will implement and compare different methods, and discuss your results. In particular, you will consider the function

$$f(x) = \frac{1}{1+x^2}, \qquad x \in [-5, 5], \tag{1}$$

which should be familiar to you from the lectures and the text book [1]. An important notion for this kind of problem is *Runge's phenomenon*.

## Part I: Numerical integration

In this part you will implement and compare some common quadrature formulas for approximating the integral

$$\int_a^b f(x)\,\mathrm{d}x. \tag{2}$$

In general, quadrature formulas can be written as

$$Q_n(f) = \sum_{i=0}^{n} w_i f(x_i), \tag{3}$$

where $w_i$ and $x_i$ are called the quadrature weights and points, respectively. The number of degrees of freedom (dof) for such a formula is equal to the number of quadrature points, i.e., $n+1$. For a known function $f$, the approximation error, $I_n(f)$, is defined as

$$I_n(f) = \left| \int_a^b f(x)\,\mathrm{d}x - Q_n(f) \right|. \tag{4}$$

1 **Newton-Cotes quadrature**. Implement a MATLAB function that approximates the integral (2) by the Newton-Cotes formula. Your function should take as input a function handle to $f$, the order of the method, $n$, and the end-points $a$ and $b$. It should return the approximation $Q(f)$.

The weights for Newton-Cotes quadrature will be given in a separate file on the web-page.

2 **Gauss quadrature**. Implement a similar function as above, but that instead uses Gauss-Legendre quadrature. The input and output variables should be the same.

The weights and quadrature points for Gauss-Legendre quadrature will be given in a separate file on the web-page.

3 **Comparison**. Apply your implemented quadrature routines on the test problem (1). Plot the error $I_n(f)$ as a function of $n$ on an appropriate scale. If the methods converge, can you determine at which rate? Comment, explain and discuss your results.

A common technique is to use composite methods. This means to divide the interval $[a, b]$ into $m$ subintervals, and then apply a quadrature formula on each subinterval and sum up the results. Denote by $n_c$ the order of the quadrature on each subinterval. For example, composite Newton-Cotes with $n_c = 2$ is known as the composite Simpson rule. For composite Simpson, $n = 2m$ (ct. Equation (3)) and the number of dof is $2m + 1$.

4 **Composite quadrature rules**. Approximate the integral (2) by composite Newton-Cotes and composite Gauss-Legendre. Use equally large subintervals and the same quadrature formula on each subinterval. You should be able to reuse your implementations from Task 1 and 2. Try with different number of subintervals, $m$, and different quadrature orders, $n_c$. Again, plot the error $I_n(f)$ as a function of $n$ on an appropriate scale. If the methods converge, can you determine at which rate? Compare with the results from Task 3. Comment, explain and discuss your results.

*OBS!* When calculating the error, $I_n(f)$, for composite methods it is important that $n$ is equal to the total number of quadrature points minus 1.

# Part II: Interpolation

In this part you will implement different methods to interpolate a given function $f(x)$ over an interval $[a, b]$. In particular, these methods will be considered:

- Lagrange interpolation
- Linear splines interpolation
- The natural qubic splines interpolation

Denote by $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ the vector of interpolation points and assume that

$$a \leq x_0 < x_1 < \ldots < x_n \leq b.$$

Furthermore, $\mathbf{y}$ will be the vector of function values at the interpolation points, i.e., $y_i = f(x_i)$. Detailed descriptions of the interpolation methods can be found e.g. in [1] or [2]. Let $p_n(x)$ be the interpolation function that interpolates the $n+1$ points $(x_i, y_i)$, i.e.,

$$p_n(x_i) = y_i, \quad \text{for } i = 0, 1, \ldots, n.$$

We measure the interpolation error by the two norms,

$$E_2 = \sqrt{\int_a^b \left(f(x) - p_n(x)\right)^2 \, \mathrm{d}x},$$
$$E_\infty = \max_{x \in [a,b]} |f(x) - p_n(x)|.$$

### Degrees of freedom

The number of degrees of freedom (dof), denoted $N$ hereafter, is the number of free parameters required to determine the interpolation function. For Lagrange interpolation, it is simply equal to the number of interpolation points, i.e., $N = n + 1$. For composite methods it is not that simple. A part of the exercises below is to determine $N$ for the different methods.

When comparing the error for different methods, it is important to use the same number of dof in each method.

5 **Lagrange interpolation**. First, implement a routine for calculating the divided difference table from $\mathbf{x}$ and $\mathbf{y}$. Then, implement a routine that takes as input this table and a vector of evaluation points, and returns the value of the Lagrange interpolation polynomial at these evaluation points. Finally, write a function that takes as input $\mathbf{x}$ and $\mathbf{y}$, and returns a function handle to evaluate the Lagrange polynomial.

Test your code on the problem (1) with different numbers of equidistant interpolation points. Calculate the inf-norm, $E_\infty$, and see if you can reproduce the results in Table 6.1 in [1]. This should serve as a verification of your code.

*Hint:* Exercise set 6 may be of good help here. Furthermore, if `eval_lagrange` is the function taking a divided difference table (denote `tab` below) and a vector of evaluation points `t`, then the command

```
fun = @(t) eval_lagrange(tab, t);
```

defines a function handle `fun`. Calling `fun(a)` should then give you the Lagrange polynomial evaluated at `a` (`a` can be a vector).

6 **Interpolation error**. Implement a function that takes as input function handles to the analytic function $f$ and the interpolation polynomial $p_n$, and the endpoints of the interval. The function should calculate and return the interpolation error $E_2$ using numerical quadrature. Chose one of your quadrature routines from Part I. In you final report, you should give some motivation for your choice.

*Hint:* Let `f` and `p` be the two function handles. Then a functional handle to the integrand can be defined by the following command:

```
integrand = @(t) (f(t)-p(t)).^2;
```

7 **Chebyshev nodes**. Using equidistant interpolation points may not be the optimal choice for some problems. An alternative is the Chebyshev nodes.

Write a function that takes as input the endpoints of the interval, and the number of nodes, and then returns a vector with the Chebyshev nodes.

8 **Composite Lagrange interpolation**. As was done for numerical integration, you should test composite Lagrange interpolation. Hence, divide the interval $[a, b]$ into $m$ equally large subintervals and use Lagrange interpolation of order $n_c$ on each subinterval.

Show that the number of degrees of freedom, $N$, for composite Lagrange is equal to $m(n_c + 1) - (m - 1) = mn_c + 1$.

9 **Linear splines**. Implement a function that takes as input $\mathbf{x}$, $\mathbf{y}$ and a vector of evaluation points, and returns the linear splines approximation at these evaluation points. What are the number of degrees of freedom for linear splines?

Test your implementations on the function (1). Use equidistant points, such that $x_i - x_{i-1} = h$ for all $i$, and plot the error $E_\infty$ against $h$ in a log-log plot. Use this plot to estimate the convergence order. Do your results agree with theory?

10 **Cubic natural splines**. The cubic natural splines are described in Section 11.4 in [1] or Section 5.2 in [2]. Let $\sigma = (\sigma_0, \sigma_1 \ldots, \sigma_n)$ be the vector of second derivatives of the cubic spline $s_2$ at the interpolation points $\mathbf{x}$. The values of $\sigma$ can be determined from the linear system of equations (11.7) in [1] (or see page 33 in [2]). Write this system as a matrix-vector equation, $A\sigma = b$. Implement routines to calculate the cubic splines that interpolates $f$.

What are the number of degrees of freedom for the natural cubic splines, that is, how many coefficients for determining the piecewise polynomial are there in total?

*Hint:* To avoid solving the linear system $A\sigma = b$ every time you need to evaluate the cubic spline, you should use a similar approach as was suggested for Lagrange interpolation.

11 **Comparison**. Consider the test problem (1). For each of the interpolation methods below, plot the error against the number of degrees of freedom, $N$. Use both the 2-norm and the $\infty$-norm. Plot the interpolation polynomials together with the original function $f$ for $N = 11$.

**a)** Lagrange interpolation with equidistant nodes

**b)** Lagrange interpolation with Chebyshev nodes

**c)** Composite Lagrange interpolation

**d)** Linear splines with equidistant nodes

**e)** Natural cubic splines with equidistant nodes

Compare the results and discuss them. It is important that you base your discussion on the lecture material or other sources. Remember to always refer to the source.

## References


[1] Süli, E. & Mayers, D. F. *An introduction to numerical analysis* (Cambridge university press, 2003).

[2] TMA4215 Numerical Mathematics: Collection of lecture notes (2013). URL `http://www.math.ntnu.no/emner/TMA4215/2014h/Exercises/AK-LectureNotes.pdf`.