Numerical Mathematics I, 2017/2018, Lab session 2

Deadline for discussion: Wednesday 02/05/18 at 5 pm

Keywords: interpolation, numerical differentiation & integration

Remarks

- Make a new folder called `NM1_LAB_2` for this lab session, save all your functions in this folder.
- Whenever a new `MATLAB` function is introduced, try figuring out yourself what this function does by typing `help <function>` in the command window.
- Make sure that you have done the preparation before starting the lab session. The answers should be worked out either by pen and paper (readable) or with any text processing software (LaTeX, Word, etc.).

1 Preparation

1.1 Lagrangian interpolation

1. Study (Textbook, Section 3.3 u/i Subsection 3.3.3).
2. Look up *estimates* of the Lebesgue constants (condition numbers) for equidistant and Chebyshev-Gauss-Lobatto Lagrangian interpolation.

   What is the meaning of these constants?

1.2 Numerical differentiation

1. Study (Textbook, Section 4.2).
2. Consider approximating a first order derivative. What is the order of accuracy when using the centered finite difference formula (Textbook, Equation 4.9)?

   What if the forward finite difference formula (Textbook, Equation 4.4) is used?

1.3 Numerical integration

1. Study (Textbook, Section 4.3) and (Textbook, Section 4.5).
2. Study the given function `trapComp.m`, which implements the composite trapezoidal rule. Which parts of the code must be changed to turn this into an implementation of the composite Simpson rule?
3. Study the adaptive Simpson method (Textbook, Section 4.5) and study the given function `trapAdpt.m`, which implements the adaptive trapezoidal method. Which parts of the code must be changed to turn this into a recursive implementation of the adaptive Simpson method?

   How can you estimate the error using mesh halving when using the Simpson formula?

2 Lab experiments

2.1 Lagrangian interpolation

Introduction

In this exercise you will investigate the *stability* of Lagrangian polynomial interpolation. Suppose you are given $n+1$ data points (x_i, y_i) from some functional relationship $y_i = f(x_i)$. Let $\Pi_n f(x)$ be the n-th order interpolating polynomial of the values y_i at the nodes x_i (Textbook, Equation 3.2). Also consider the perturbed data, given by

$$\tilde{y}_i = y_i + \epsilon_i,$$

with its corresponding n-th order interpolating polynomial $\Pi_n \tilde{f}(x)$. The stability is determined by the ratio of the maximum error of the interpolating polynomial (on the interval $I = [x_0, x_n]$)

$$\max_{x \in I} |\Pi_n f(x) - \Pi_n \tilde{f}(x)|,$$

to the maximum error in the data

$$\max_{0 \le i \le n} |\epsilon_i|.$$

This ratio is bounded by the *Lebesgue* constant Λ_n (Textbook, Equation 3.9).

Experiment

Investigate the stability of the Lagrangian polynomial interpolation applied to the cosine function on the interval $[-\pi/2, \pi/2]$. For this you should use the given `MATLAB` function `lagrangeStability.m` (study this function and read the comments).

Compare two different distributions of the nodes. First consider equidistant nodes $x_i = -\frac{\pi}{2} + \frac{i\pi}{n}$ for $i = 0, \ldots, n$. Secondly, use Chebyshev-Gauss-Lobatto nodes $x_i = -\frac{\pi}{2} \cos\left(\frac{\pi i}{n}\right)$. For both nodal distributions, compare the ratio mentioned above with the corresponding theoretical estimates. Test the stability for $n = 8, 16, 32$ subintervals with the maximum perturbation given by `epsilon` $= 10^{-2}$. Note: `MATLAB`'s built-in function `polyfit` might give some warning messages due to the bad conditioning of the interpolating polynomial.

2.2 Numerical differentiation

Introduction

Recall from calculus that the derivative of a function $f : \mathbb{R} \mapsto \mathbb{R}$ at a point x, is defined as the following limit

$$f'(x) := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

If this limit exists, it follows that for sufficiently small $h > 0$ we have

$$\frac{f(x+h) - f(x)}{h} \approx f'(x). \tag{1}$$

Experiment

To investigate this, write a `MATLAB` function named `diffConsistency.m` that tests the validity of (1). That is, for $h_i = h_0 10^{-i}$, calculate the absolute value of the error (the error here is the difference between the left-hand side and the right-hand side of (1)) made in approximating the derivative, where i should run from 0 up to some predefined maximum `iMax`. You should use random values for x using the function `rand` (use `rng(10)` before calling `rand`; this ensures that everyone gets the same "random" results). The header of this function should be of the following form:

```matlab
% INPUT
% f          function handle
% df         function handle of the derivative of f
% x          point at which to evaluate f and df
% iMax       maximum number of refinements
% h0         initial h
% OUTPUT
% errList    array containing the errors in approximating df
% hList      array containing values of h
function [errList, hList] = diffConsistency(f, df, x, iMax, h0)
```

Note that the function should stop refining h whenever $h <$ `eps`[*]. Test your implementation on the function $f(x) = \sin(x) + e^x$. Plot the logarithm of the absolute value of the error versus the logarithm of h (use `loglog`). Make sure `iMax` is sufficiently large.

2.3 Numerical integration

Write a `MATLAB` implementation of the *composite Simpson formula*. Base your implementation on the composite trapezoidal formula as given in `trapComp.m`. Name your function `simpComp.m`, the input and output should be the same as that of the composite trapezoidal formula.

Next, do the same for the *adaptive Simpson method*. Base your *recursive*[†] implementation on the adaptive trapezoidal method as given in `trapAdpt.m`. Name your function `simpAdpt.m`, the input and output should be the same as that of the adaptive trapezoidal method.

Compute the number of subintervals needed using the standard composite trapezoidal formula (`trapComp.m`), the adaptive trapezoidal method (`trapAdpt.m`), the composite Simpson formula and the adaptive Simpson method for `tol` $= 10^{-i}$, where $i = 1, \ldots, 10$. Also measure the computation time using the functions `tic` and `toc`. Perform the experiment on the following integral

$$\int_0^2 \arctan(\sqrt{x})dx = 3\arctan(\sqrt{2}) - \sqrt{2}.$$

For `tol` $= 10^{-10}$, plot the number of subintervals against the error estimate using the `stats` output for all four methods in a log-log plot.

3 Discussion

3.1 Interpolation

1. Why is it important for an interpolation method to be stable?

[*] `eps` equals the distance from 1.0 to the next largest double-precision number and is equal to 2^{-52}. See also (Textbook, Equation 1.2).

[†] The adaptive Simpson implementation that can be found in the book is not recursive.

2. Which node distribution leads to a more stable interpolation? Does this agree with the theory?

3. How does (piecewise) interpolation relate to integration?

3.2 Numerical differentiation

1. How does the the error depend on h? Does this agree with the theory?

2. What happens if h becomes too small? Given an analytical expression for the optimal value of h.

3.3 Numerical integration

1. What order of accuracy do you observe using the composite trapezoidal formula? Do you expect this from the theory? Hint: The second order derivative is given by

$$f''(x) = \frac{-1 - 3x}{4x\sqrt{x}(1+x)^2}.$$

2. What is the advantage of using adaptive mesh refinement over uniform mesh refinement? Relate this to the expression of the error (Textbook, Equation 4.20).

3. Explain the changes you have made to the composite trapezoidal formula to obtain the composite Simpson formula. Next explain the changes you have made to the adaptive trapezoidal method to obtain the adaptive Simpson method.

4. How many subintervals do you need using the standard composite Simpson formula with `tol = 1E-10`? And how many subintervals do you need using the adaptive Simpson method? Which method is more efficient?

5. Use the `stats` output from your adaptive formula function. Where are most of the adaptively chosen nodes located? Why are they located there?