Fall 2017: Numerical Methods I
Assignment 1 (due Sep. 21, 2017)

Objectives. This class is for you and you should try to get the most out of it for yourself. The methods we study are the basis for more complicated algorithms and will help you to understand functions that are available in numerical software packages. The problems are meant to be challenging and to leave room for discussion. You are not expected to get all of them 100% right. Please make sure you discuss your results whenever that is required; a meaningful discussion that illustrates that you understand the point is usually more important than obtaining the correct result.

Homework submission. Homework assignments must be submitted by 9AM on the morning after the listed due date. Late submissions will only be accepted if you send me an email with a justification *one day before* the due date. Preferably, use LaTeX typesetting (try `LyX` if you are not yet familiar with LaTeX) for your homework. I will make the TEX-files of all assignments available as a starting point if you are still learning LaTeX. Cleanly handwritten homework will be accepted as well. Preferably, hand in a printout of your homework. You can hand in your homework in class, or slide it under my office (HWW #1111) door. If you need to email your homework, please email it as a *single* PDF file. If you are required to hand in code listings, this will explicitly be stated on that homework assignment.

Collaboration, and acknowledging sources. NYU's integrity policies will be enforced. You are encouraged to discuss the problems with other students in person or on Piazza. However, you must write (i.e., type) every line of code yourself and also write up your solutions independently. Copying of any portion of someone else's solution/code or allowing others to copy your solution/code is considered cheating. Please cite every source you use for your homework, even if it is just Wikipedia.

Plotting and formatting. Plot figures carefully and think about what you want to illustrate with a plot. Choose proper ranges and scales (`semilogx, semilogy, loglog`), and always label axes. Sometimes, using a table can be useful, but never submit pages of numbers—numerical mathematics is not about producing numbers, but about analyzing, learning from and gaining insight through computations. Discuss what we can observe in and learn from a plot. If you do print numbers, use `fprintf` to format the output nicely. Use `format compact` and other format commands to control how MATLAB prints things.

Exporting figures. When you create figures using MATLAB (or Python/Octave), please always export them in a vector graphics format (.eps, .pdf, .dxf) rather than raster graphics or bitmaps (.jpg, .png, .gif, .tif). Vector graphics-based plots avoid pixelation and look much cleaner. They can be scaled without losing resolution, and look much more professional. They can directly be embedded in LaTeX documents. Many tutorials on making professional looking figures and visualizations in MATLAB/Python can be found on the web when you search for *MATLAB professional looking plots*. To achieve reproducible prints from within a script/function, I recommend using the `print` command in MATLAB rather than printing from the GUI. For instance, use

```
print -depsc myfigure.eps
```

to print a color figure in .eps format to a file named `myfigure.eps`.

Coding. It is only possible to fully understand and appreciate a numerical method once you have implemented it yourself. We will use MATLAB's build-in methods to compare with, but it is a critical part of this class to implement, debug and to verify your own (small) programs. After you are finished with an implementation, make sure that it is correct and does what you think it does. Use meaningful variable names, try to write clean, concise and easy-to-read code and use comments for explanation. If you feel that you are struggling with MATLAB (or Python/Octave), please spend some time as soon as possible to brush up your programming skills using, for instance, Cleve Moler's book. This will eventually save you time and you will benefit more from the class.

1. **[$1, 2, \infty$-norm, 2+2pt]** Let $\|\cdot\|$ denote a norm in $\mathbb{R}^n$, and $A \in \mathbb{R}^{n\times n}$ be a square matrix. Recall that the induced matrix norm is defined by

$$\|A\| := \sup_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|}.$$

   (a) Show that, for any $\boldsymbol{v} \in \mathbb{R}^n$, we have

$$\|\boldsymbol{v}\|_\infty \leq \|\boldsymbol{v}\|_2 \quad \text{and} \quad \|\boldsymbol{v}\|_2^2 \leq \|\boldsymbol{v}\|_1 \|\boldsymbol{v}\|_\infty.$$

   In each case, give an example of a nonzero $\boldsymbol{v}$ for which equality is obtained.

   (b) Using the result above, show that

$$\|A\|_\infty \leq \sqrt{n}\|A\|_2 \quad \text{and} \quad \|A\|_2 \leq \sqrt{n}\|A\|_\infty$$

   In each case, give an example of a nonzero matrix A for which equality is obtained.

2. **[Properties of induced matrix norms, 1+1+2+2pt]** Let $\|\cdot\|$ denote a norm in $\mathbb{R}^n$, and $A, B \in \mathbb{R}^{n\times n}$ be a square matrices.

   (a) Show that

$$\|A\| = \sup_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\|.$$

   (b) Show that the induced matrix norm satisfies

$$\|AB\| \leq \|A\|\|B\| \text{ for all } A, B \in \mathbb{R}^{n\times n}.$$

   (c) Show that the matrix norm induced by the vector 1-norm $\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|$ is the maximum 1-norm of the matrix columns.
   Hint: Denote the columns of A by $\boldsymbol{a}_j$, i.e., $A = \begin{bmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_n \end{bmatrix}$ and take a vector $\boldsymbol{x}$ with $\|\boldsymbol{x}\|_1 = 1$. Show that

$$A\boldsymbol{x} = \sum_j x_j \boldsymbol{a}_j, \text{ and } \|A\boldsymbol{x}\|_1 \leq \max_j \|\boldsymbol{a}_j\|_1.$$

   Then, by choosing $\boldsymbol{x}$ appropriately, show that one can satisfy $\|A\boldsymbol{x}\|_1 = \max_j \|\boldsymbol{a}_j\|_1$ and use that to argue that the matrix 1-norm is equal to the maximum 1-norm of the matrix columns.

   (d) Now, lets compute the matrix norm induced by the ∞-vector norm defined by $\|\boldsymbol{x}\|_\infty := \max_{1\leq i\leq n} |x_i|$. *Hint:* Denote the rows of B by $\boldsymbol{b}_j$, i.e.,

$$B = \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_m \end{bmatrix},$$

   and consider a vector $\boldsymbol{x}$ with $\|\boldsymbol{x}\|_\infty = 1$. Show first that

$$\|B\boldsymbol{x}\|_\infty \leq \max_i \|\boldsymbol{b}_i\|_1.$$

   Then, show that by choosing $\boldsymbol{x}$ appropriately, one can satisfy $\|B\boldsymbol{x}\|_\infty = \max_i \|\boldsymbol{b}_i\|_1$. Use that to argue that the matrix norm induced by the ∞-norm is the maximum 1-norm of the matrix rows.

3. **[Condition numbers, 1+2+2pt]** Compute absolute and relative condition numbers (for infinitesimal perturbations) for the following problems

   (a) $G(x) = \sqrt{x}$ for $x \in \mathbb{R}, x \geq 0$.

   (b) $G : \mathbb{R}^2 \to \mathbb{R}^2$ defined by

   $$G(\boldsymbol{x}) = \begin{pmatrix} x_2 + 1 \\ x_1^2 x_2 \end{pmatrix},$$

   where $\boldsymbol{x} = (x_1, x_2)^T \in \mathbb{R}^2$. Compute the condition numbers when using the $\|\cdot\|_1$ and the $\|\cdot\|_\infty$ norms in $\mathbb{R}^2$.

   (c) Consider two lines in $\mathbb{R}^2$ given by the equations $y = 0$ and $ax + y = b$, with $a, b \in \mathbb{R}$. Compute the intersection point S of the two lines as a function of a, b. When you change b, for which values of a is S most sensitive? Relate this to the conditioning number of an appropriate 2×2 matrix.

4. **[Computing in finite precision, 2+1+1(+1)pt]**

   (a) Due to the finite precision in floating point number representation, there are gaps between consecutive numbers. The size of these gaps depends on the size of the number and on the precision (e.g., double or single precision). MATLAB provides the function `eps(·)` (the analogue in NumPy is the function `spacing`), which returns, for a number, the distance to the next floating point number in the same precision. Using the form of double and single floating point number representation, explain the values you find for

   `eps(1)`,[1]
   `eps(single(1))`,[2]
   `eps(`2^{40}`)`,
   `eps(single(`2^{40}`))`.

   (b) Try the following experiment and explain the behavior:

   ```
   > a = 0.5;
   > b = 0.7-0.2;
   > a == b
   > sprintf('%20.18f',a)
   > sprintf('%20.18f',b)
   ```

   [3] [4]

   (c) Give examples of nonzero numbers a, b, c whose floating point representations satisfy: `> b + a = a;`
   `> (a + b) + c` $\neq$ `a + (b + c);`

[1] This value, `eps(1)` $=$ `eps` $\approx 2.22 \times 10^{-16}$ is usually referred to as machine epsilon, and it gives an indication of the rounding error when dealing with double precision floating point numbers.

[2] Note that the MATLAB command `single` switches to single precision.

[3] The double equal sign checks if the numbers to its left and right are the same (i.e., every bit coincides!). An answer of 0 means *false* (not the same), and 1 means *true*. Note that you should *never* do this for floating point numbers! You should always test if the difference between two numbers is small (e.g., `10*eps(1)`) rather than if they are equal.

[4] This outputs a large number of digits and helps to be show rounding errors due to the finite precision of the computer representation of numbers. Use `help sprintf` to read up on this MATLAB syntax (which is the same as in other languages, such as C). The analogue output in Python is generated by `print "%20.18f'' % a`.

(d) **[1pt extra credit]** Let us experiment with how fast operations with floating point numbers[5] can be performed. Consider the following small program, which performs 10^9 additions of (random) floating point numbers, and measures the time it takes for doing that.

```
N = 1e5; M = 1e4;
a = rand(N,1);
b = rand(N,1);
tic, for i = 1:M, a.*b; end, toc
```

Report the run time of this program. Now modify N, M such that their product remains the same, and again report the run times. Finally, repeat the exercise but use single precision random numbers (the default is double precision) by replacing the definitions of the vectors a, b as follows:

```
a=single(rand(N,1));
b=single(rand(N,1));
```

The differences you will observe are due to the need to read numbers from memory (which takes time) before they can be added by the processing unit (the CPU), and due to the fact that numbers can be re-read faster if they have been read recently.[6] Taking this into account, try to explain the different timings you observe.

5. **[Finite differences in finite precision, 2+3pt]** Often, derivatives of functions can only be computed numerically. For a three times differentiable function $f : \mathbb{R} \to \mathbb{R}$ one can use Taylor expansions around the point x_0 to show the following approximations:

$$f'(x_0) = D_1 f(x_0) + O(|h|) \quad \text{with } D_1 f(x_0) := \frac{f(x_0+h) - f(x_0)}{h} \qquad \text{(one-sided finite differences)},$$

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + O(|h|^2) \qquad \text{(centered finite differences)}.$$

In these approximations, one subtracts very similar numbers from each other, which is badly conditioned and can lead to significant cancellation errors. Here, we discuss an alternative.

(a) Assuming the function f can also be defined for complex arguments, use a Taylor expansion of $f(x_0 + ih)$ to show that

$$f'(x_0) = \frac{\mathrm{Im}(f(x_0+ih))}{h} + O(|h|^2),$$

where "Im" denotes the imaginary part of a complex number. This is called *complex step differentiation*, and it obviously avoids numerical cancellation.

(b) Compare these three approximations to compute the derivative of the function

$$f(x) = \frac{\exp(x)}{\cos(x)^3 + \sin(x)^3}$$

at $x_0 = \pi/4$. In order to do so, use progressively smaller perturbations $h = 10^{-k}$ for $k = 1, \dots 16$. Present the errors in the resulting approximations in a log-log plot, i.e.,

[5] Usually one refers to additions and multiplications as floating point operations, and the speed of computing processors has traditionally been given by the number of these operations they are—in theory—able to perform. See https://en.wikipedia.org/wiki/FLOPS for more details.

[6] This has to do with the memory hierarchy in modern computers, see for instance https://en.wikipedia.org/wiki/Memory_hierarchy. What exactly "recently" means depends on the size of the CPU cache, see https://en.wikipedia.org/wiki/CPU_cache if you are interested to learn more details.

use a logarithmic scale to plot the values of h on the x-axis, and a logarithmic scale to plot the errors between the finite difference approximations and the exact value, which is $f'(x_0) = 3.101766393836051$, on the y-axis. Discuss your result.

6. **[Rounding and Taylor series, 1+2+2pt]** Write a program that approximates $\exp(x)$ with its Taylor series:

$$\exp(x) \approx \sum_{i=0}^{n} \frac{x^i}{i!} \tag{1}$$

and plot, for $x = -5.5$ the difference to the exact value obtained with $n = 1, \ldots, 30$, where you use the (truncated) Taylor expansion in three different ways:

- Use the formula (1) directly to compute $\exp(-5.5)$.
- First use that $\exp(-5.5) = 1/\exp(5.5)$, and then use (1).
- Use that $\exp(-5.5) = (1/\exp(0.5))^{11}$ and then use (1).

Discuss the results, which show that a different mathematical formulation can significantly influence the accuracy of a computation.