Exercise 1. Plotting a function

Goal: We start by getting familiar with the basic plotting features of the programming language Julia.

Task 1: Choose whatever mathematical function you want (e.g. $\sin(x)$, $\log(x)$, $\exp(x)$, etc.) and plot it. Change the linestyle, thickness and color of the curve. Add axis labels and a title to your plot.

Task 2: Choose a second function and plot it in the same figure as the function you chose in task 1. Add labels to both functions and include a legend in your plot.

Exercise 2. Data Fitting

Goal: Fit some data using the Julia package LsqFit.

Assume that we have some data

```
xdata = [ 15.2; 19.9;  2.2; 11.8; 12.1; 18.1; 11.8; 13.4; 11.5;  0.5;
          18.0; 10.2; 10.6; 13.8;  4.6;  3.8; 15.1; 15.1; 11.7;  4.2 ]
ydata = [ 0.73; 0.19; 1.54; 2.08; 0.84; 0.42; 1.77; 0.86; 1.95; 0.27;
          0.39; 1.39; 1.25; 0.76; 1.99; 1.53; 0.86; 0.52; 1.54; 1.05 ]
```

and a function of the form

$$f(x) = \beta_1 \left(\frac{x}{\beta_2}\right) \exp\left\{-\left(\frac{x}{\beta_2}\right)^{\beta_3}\right\}.$$

Task 1: Find the vector $\boldsymbol{\beta}$ that best fits the data using the package `LsqFit` and quantify the fit quality.

Hint: The LsqFit package can be installed in the same way as the IJulia package (see section "Setting up Julia" below).

Task 2: Plot the raw data and the fitting function.

Exercise 3. Determinant Value and Ill-Conditioning

Goal: This exercise should make you familiar with the `LinearAlgebra` package in Julia and at the same time recapitulate the important concepts of determinant value and ill-conditioning.

Take home point 1: *To find out whether or not the problem* $\mathbf{Ax} = \mathbf{b}$ *is ill-conditioned, you should compute the condition number* $\kappa(\mathbf{A}) = \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\|$ *(and forget about the determinant).*

Take home point 2: you implement (use) linear algebra methods in Julia

Determinant value and ill-conditioning

Linear algebra is employed in most of computational physics. The two main problems that linear algebra studies can be trivially stated: (a) $\mathbf{Ax} = \mathbf{b}$ is a concise way of stating a linear system of n equations in n unknowns, and (b) $\mathbf{Av} = \lambda\mathbf{v}$ is the standard form of the matrix eigenvalue problem.

Given the emphasis that numerical analysts have placed on the subject, it is not surprising to hear that state-of-the-art libraries (like BLAS and LAPACK) are robust and dependable. We will study how Julia will perform (maybe use `https://github.com/JuliaLinearAlgebra/IterativeSolvers.jl`).

Here we limit ourselves to the solution of linear systems, $\mathbf{Ax} = \mathbf{b}$ and consider ill-conditioned systems, where a small change in coefficients will produce large changes in the result. In particular, this situation occurs when

$$\det\mathbf{A} \sim 0 \tag{1}$$

is true.

On the face of it, this is plausible: a matrix $\mathbf{A}$ which is singular has zero determinant, $\det(\mathbf{A}) = 0$. It is then tempting (but wrong) to say that a near-singular matrix is one that has a near-zero determinant. The first thing that comes to mind when investigating this tentative criterion is how one could quantify "close to zero." One should be able to relativise such a determination: a number which encapsulates the entire matrix (such as the determinant) may be large or small in comparison to the magnitude of $\mathbf{A}$'s matrix elements. Quantitatively, one introduces the concept of the matrix norm, which is a number summarising how large or small the matrix elements are. For example,

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}|A_{ij}|^2} \tag{2}$$

is the Euclidean or Frobenius norm, and

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{max}(A^\dagger A)} = \sigma_{max}(A) \tag{3}$$

is the Spectral norm, where λ_{max} and σ_{max} are the maximal eigenvalue and singular value respectively, and $A^\dagger$ is the conjugate transpose of A.

Since a matrix norm is non-negative, a plausible way to cast the above tentative criterion is as follows: $|\det(\mathbf{A})| \ll \|\mathbf{A}\|$, i.e., the absolute value of the matrix determinant is much smaller than the matrix norm. This automatically adjusts for the fact that the matrix elements themselves could be small, in order to see if the determinant is even smaller (or not). Let's apply this criterion to an example. For the matrix

$$\mathbf{A} = \begin{pmatrix} 1\times10^{-5} & 2\times10^{-5} \\ 3\times10^{-5} & 4\times10^{-5} \end{pmatrix} \tag{4}$$

we find $|\det(\mathbf{A})| = 2\times10^{-10}$ and $\|\mathbf{A}\|_F \approx 5.5\times10^{-5}$. Thus, since the criterion $|\det(\mathbf{A})| \ll \|\mathbf{A}\|$ is clearly satisfied, you may be thinking that this is an ill-conditioned matrix, i.e., a matrix for which many methods will have a hard time (and unstable methods will simply fail).

But this means that we are free to multiply our $n = 2$ equations by a constant, without changing anything in the statement of the problem. In other words, multiplying the matrix in Eq. (4) with, say, 10^{10} should not lead to a fundamental change. For the matrix

$$\mathbf{B} = \begin{pmatrix} 1 \times 10^5 & 2 \times 10^5 \\ 3 \times 10^5 & 4 \times 10^5 \end{pmatrix} \tag{5}$$

we find $|\det(\mathbf{B})| = 2 \times 10^{10}$ and $\|\mathbf{B}\|_F \approx 5.5 \times 10^5$. In other words, we now find $|\det(\mathbf{B})| \gg \|\mathbf{B}\|$, so this appears to be a (very) well-conditioned matrix. This is the *opposite* conclusion from that drawn above! [1]

The resolution is to employ matrix perturbation theory and carefully study the effect a small change of our matrix has on the solution vector, i.e., what $\mathbf{x} + \Delta\mathbf{x}$ looks like when you start with $\mathbf{A} + \Delta\mathbf{A}$. A quick derivation here shows that[1]

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\| \, \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \, . \tag{6}$$

In words, this relates an error bound on $\|\Delta\mathbf{x}\|/\|\mathbf{x}\|$ with an error bound on $\|\Delta\mathbf{A}\|/\|\mathbf{A}\|$. The coefficient in front of $\|\Delta\mathbf{A}\|/\|\mathbf{A}\|$ tells us whether or not a small perturbation will tend to get magnified. Equation (6) naturally leads us to introduce the *condition number*

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\| \, , \tag{7}$$

where $\|\cdot\|$ could be any matrix norm. If $\|\cdot\| = \|\cdot\|_2$, the condition number becomes

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \, \|\mathbf{A}^{-1}\|_2 \, = \frac{\sigma_{max}}{\sigma_{min}} \, . \tag{8}$$

When $\kappa(\mathbf{A})$ is of order unity we are dealing with a well-conditioned problem, i.e., a small perturbation is not amplified. Conversely, when $\kappa(\mathbf{A})$ is large, a perturbation *is* (typically) amplified, so our problem is ill-conditioned. Crucially, this condition number involves both the matrix $\mathbf{A}$ and its inverse, $\mathbf{A}^{-1}$. (Equally crucially, this condition number has nothing to do with the determinant.) As a result, an overall scaling has no effect: the condition number for both $\mathbf{A}$ and $\mathbf{B}$, from Eq. (4) and Eq. (5) respectively, is 15 when using the Frobenius norm, i.e., not very large. The study of problem conditioning, as well as its relationship to the stability of a given method, is nicely discussed in the book by N. Trefethen and D. Bau.[1] To summarize:

Task 1: Write a function *matgen (n, c)* with n the size of the random quadratic matrix M and c is the condition number of M, computed using the Spectral norm $\|\cdot\|_2$. Hint: Generate two random orthogonal matrices U and V. Create a diagonal matrix σ of singular values in the range of $1 \ldots 1/c$. The result is then given by $M = U\sigma V$.

Task 2: With this matrix at hand use the standard library Linear Algebra of Julia to explore the observations made above.

[1] There is obviously something wrong with the criterion: multiplying all of our equations with a constant changes the norm, but it has a more dramatic impact on the determinant. However, such a simple rescaling has nothing to do with how well-conditioned or ill-conditioned our problem is.

References

[1] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, (Society for Industrial and Applied Mathematics, 1997).

Additional instructions

We expect you to have already some, at least basic, knowledge about computer programming and experience in working with a programming language like `python`, `C++`, etc. as well as with tools like `Mathematica`, `Matlab`, etc.

While any familiarity with such languages and tools is highly favourable, being an absolute beginner will not disqualify you from taking the course. We will do our best to help you establish a solid and comprehensive background in this field which you will be able to further develop in the future and from which you will have benefits for the rest of your career.

Linux as operating system

We generally regard Linux as the most convenient operating system and Julia as the most suitable programming language under which we are going to develop our programs and simulations. The reason for this is to have a uniform working environment which facilitates the interaction between the students. However, everyone is free to choose their operating system and programming language as long as the goal of the lectures and the exercises is reached.

In the following sections we will help you to get acquainted with Linux.

Logging in and getting things to work

If you do not have your own computer, ETH provides you with an access to the ETH computers. The ETH computers are dual boot machines, i.e. you may have to restart in order to use Linux.

1. Log in with your account information.
2. Open a browser. Under the following address you can find all the information about this course: `https://moodle-app2.let.ethz.ch/course/view.php?id=13049`.
3. Open a terminal (shell). A shell is a program that provides the traditional, text-only user interface for Linux and other Unix-like operating systems. Its primary function is to read commands that are typed into a console (i.e. an all-text display mode) or terminal window (an all-text window) and then execute (i.e. run) them. (Definition from `http://www.linfo.org`.)
4. Create a new directory named e.g. `comphys`: `mkdir comphys`
5. Enter the created directory: `cd comphys`
6. Open Jupyter.

Some useful commands

COMMAND	USAGE
pwd	shows the path of the current directory
mkdir *name*	creates the directory 'name'
cd *name*	changes to the directory 'name'
gedit	starts the GNU-Editor
(x)emacs	starts the (X)EMACS editor
ls	lists the files and directories in the current directory
cp, mv	copies, moves files and directories
man	displays the documentation of a command
julia *name*.jl	runs the Julia program 'name'
python3 *name*.py	runs the python program 'name'
g++ *name*.cpp -o *name*.out	compiles the C++ program 'name'
./name	runs the executable file 'name'
command `&`	moves the command 'command' to the background
exit	exits shell/window

Setting up Julia

The preferred programming language in this course is Julia, a fairly new option that has been gaining popularity in the past few years. It is a high-level, dynamically typed language, similar to python in syntax, but much more efficient thanks to its wide use of multiple dispatch, giving it computational speeds comparable to C.

You can download the latest stable version of Julia (1.9.3 as of September 2023) on their website `https://julialang.org/downloads/`, or directly from the GitHub repository `https://github.com/JuliaLang/julia`. Julia can be used interactively in the terminal, by simply typing `julia`, or can be run as scripts with `julia script.jl`. Julia can also be used in Jupyter notebooks, just like Python. To install the Julia Jupyter package run the following commands in the terminal:

1. `julia`
2. `using Pkg`
3. `Pkg.add("IJulia")`

Assuming that you already had Jupyter installed, you can now create notebooks with Julia. Otherwise, head to `https://jupyter.org/install` to download Jupyter.

We have prepared an introductory Julia notebook `julia_introduction.ipynb` to get you acquainted with the language.