Lab 0: Welcome

In this first lab, you will set up your programming environment and check that you can run iPython notebooks. You may also review course prerequisites and notation conventions.

If you have extra time, you can start working through the weekly labs/courseworks/demos on the course Github at your own pace.

Contents

1 Set up your computing environment

In this course, you will write computer code to evaluate, simulate, explore, and visualise mathematical models.

- **Programming language for instruction:** This course will use Python3 with notebooks (`.ipynb`) and the NumPy, SciPy, and Matplotlib libraries and their associated ecosystems. There may also be some demos in Julia.
- **Programming language for coursework:** You are welcome to use any language you wish for your coursework submission. If most of your experience is with Matlab or R, we encourage you to complete this course in Python or Julia (DifferentialEquations.jl contains some nice numerical integrators you might like to try out). The instructors are familiar with Python and Julia and best placed to support these two languages, should you run into difficulty.
- **Programming Environment:** You will need to set up and support your own programming environment and workspace. The instructors are happy to sit down and troubleshoot difficulties in the first few labs.
- **Hardware:** Feel free to use either your own devices or the lab computers. The lab computers come installed with standard software (although this has not been thoroughly tested). Python notebooks are also supported in *Google Colab*, although you will need a Google account for this. If using the lab machines, be sure to save and export your work in a way that you can access later from other devices.

2 Test your Python setup

Clone the course Github, (at github.com/engmaths/SEMT30003_2025) and browse to the folder Coursework/lab1/Brochier et al data exploration.

Start up your Python environment and confirm that you can execute the code in the brochier_et_al_excerpt.ipynb notebook.

You may be able to run this notebook on Google Colab by replacing `github.com` with `githubtocolab.com` in the Github URL for the `.ipynb` notebook file.

You do not need to understand everything in this notebook now—you will explore data analysis further in the extended coursework (assessed) toward the end of the semester.

3 Prerequisites

The Appendix of Dayan and Abbot (2001) *Theoretical Neuroscience* is a good review of course prerequisites. This course assumes some familiarity with

- Basic ordinary differential equations
- Differential and integral calculus
- Probability and statistics
- Linear algebra
- Programming in Python
- Introductory cell biology and biochemistry

Our use of the above topics will not be especially technical (compared to the typical third-year content in an engineering/CS degree). We will focus on connecting mathematical models, biological phenomena, and computer code.

Calculus Intuition from differential and integral calculus will help when we cover models of neural dynamics and learning. The exam may require differentiating simple functions not necessarily discussed in lecture.

Ordinary Differential Equations (ODEs) Many of the models we explore in this course will involve an ODE in some form. You may be asked to solve simple ODEs whose solutions were covered in lectures, or to analyse and interpret a provided solution to an ODE (e.g. perhaps something more complicated, nonlinear). First-order linear differential equations appear in models of neuronal dynamics in various ways, and we'll spend some time studying them.

Probability, Statistics, Machine Learning You should be familiar with probability, random variables, and common distributions such as Gaussian, Poisson, uniform, and exponential. You should be able to sample from distributions and calculate statistical quantities in your chosen programming language. Familiarity with vocabulary from machine learning may be helpful (loss, objective, cost, gradient, gradient descent), but is not a prerequisite.

4 Notation

We will use notation common to mathematical modelling and machine learning in this course.

Scalars

- Usually lowercase: a, b, c, d.
- Sometimes uppercase for electrical quantities like voltage V.
- *Standard constants:* e will be Euler's constant (unless part of a floating-point literal in computer code); π will be the ratio of a circle's circumference to its diameter; τ denotes a time constant or time, not 2π.

Vectors

- Bold lowercase: $\boldsymbol{x}$ or $\mathbf{x}$ is a *column vector* unless stated otherwise; Handwritten using arrow above $\vec{x}$ or line below $\underline{x}$.
- Given vector $\boldsymbol{x}$, x_i is its i^{th} element; If $\ldots$ is an expression that evaluates to a vector, then $\{\ldots\}_i$ is the i^{th} element of said vector.

Inner products

- I'll use the notation $\mathbf{w}^\top \boldsymbol{x} = \sum_i w_i x_i$ in this course.
- I will avoid $\mathbf{w}\cdot\boldsymbol{x}$, which could sometimes be confused with scalar or element-wise multiplication.

Matrices

- Upper-case A or $\mathbf{A}$; Handwritten using double arrows $\vec{\vec{A}}$ or underline $\underline{\underline{A}}$, or simply as A when unambiguous.
- Given matrix $\mathbf{A}$, a_{ij} is its element at row i, column j; If $\ldots$ is an expression that evaluates to a matrix, then $\{\ldots\}_{ij}$ is the i^{th} row and j^{th} column element of of said matrix.
- I may write expressions like $\underline{x}^\top \underline{\underline{A}}\, \underline{x}$ simply as "$x^\top A x$" when things get cluttered, provided it is clear that $\boldsymbol{x}$ is a column vector and $\mathbf{A}$ is a matrix. Let me know if I've left something ambiguous.

Matrices and Vectors

Equations for neural networks are often written in matrix and vector notation. Given column vector $\boldsymbol{x}$ and matrix $\mathbf{W}$, you should be able to quickly recognise that:

$$\begin{aligned} \boldsymbol{x}^\top \boldsymbol{x} &= \textstyle\sum_i x_i^2 && \text{Squared Euclidean norm (also } \|\boldsymbol{x}\|_2^2) \\ \{\boldsymbol{x}\boldsymbol{x}^\top\}_{ij} &= x_i x_j && \text{Outer product} \\ \{\mathbf{W}\boldsymbol{x}\}_i &= \textstyle\sum_j w_{ij} x_j && \text{Matrix–vector multiplication} \\ \boldsymbol{x}^\top \mathbf{W} \boldsymbol{x} &= \textstyle\sum_{ij} w_{ij} x_i x_j && \text{Quadratic form} \end{aligned} \tag{1}$$

In the sums like $\sum_i$ in (1), it is understood that i ranges over all row or column indices, as applicable. You should also be able to readily imagine the summations as `for` loops in computer code, and mentally convert `for` loops into the corresponding mathematical notation as needed.

If you have not already encountered *The Matrix Cookbook* by Petersen and Pedersen, I highly recommend taking a look. The matrix and vector manipulations in this course will be simple, but you may find this resource useful for numerical linear algebra in the future.

Derivative notation:

- We will use Newton's dot notation $\dot{x}(t), \ddot{x}(t), \dddot{x}(t)$ for derivatives of $x(t)$ in time in differential equations. We may omit the argument (t), writing, $x, \dot{x}, \ddot{x}, \dddot{x}$, but you should remember that these quantities change over time.
- You may see Lagrange's notation $f'(x), f''(x), f'''(x), f^{(n)}(x)$ for the first, second, third, and $n > 3$ derivative of a scalar function $f(x)$ in its argument.
- You may also see Leibniz's notation as $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$, $\frac{\mathrm{d}\, f(x)}{\mathrm{d}x}$, or $\frac{\mathrm{d}\, y}{\mathrm{d}x}$ with $y = f(x)$.
- **Partial derivatives:** Consider $f(x, y)$ where $y = g(x)$. The partial derivative $\partial_x f(x, y)$, $\frac{\partial f(x,y)}{\partial x}$, or $\frac{\partial}{\partial x} f(x, y)$, means that we should consider only the derivative of f in terms of x as an argument (ignoring the fact that we know y also depends on x). Contrast this to the *total* derivative $\frac{\mathrm{d}\, f}{\mathrm{d}x} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} = \partial_x f + \partial_y f \cdot \partial_x g$, which accounts for how $f(x, y)$ would actually change if we were to vary x.
- **Gradients of scalar functions:** If $y = f(\boldsymbol{x})$ is a function of a *vector* $\boldsymbol{x}$, we can denote its derivative in $\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}^\top$ using the gradient $\nabla_{\boldsymbol{x}} f = [\partial_{x_1} f, \partial_{x_2} f, \ldots, \partial_{x_N} f]^\top$. In this course we will use the convention that the gradient is a column vector. The symbol ∇ is typeset as `\nabla` in LaTex.

Other

- The notation $a \leftarrow b$ means "assign the value b to variable a", and is written as `a = b` in many computer languages. The symbol $\leftarrow$ can be typeset in LaTex math mode using `\gets`. The $\leftarrow$ notation is common in pseudocode.
- In the scope of this course, if you see the notation $a := f(b)$, it means that the variable a is *defined as* $f(b)$, as opposed to simply happening to have an equal value. Some programming languages also use `:=` as the assignment operator.
- You may see functions with two groups of arguments separated by a semicolon, like $f(\boldsymbol{x}, \boldsymbol{u}; \mathbf{W}, \mathbf{B})$. In machine learning, this usually means that the variables $\mathbf{W}, \mathbf{B}$ right of the semicolon should be interpreted as *parameters*. You may also see a semicolon used to delimit two sets of random variables in the notation for mutual information.

(end)