Homework - Serie 10

Kevin Sturm
Python 3

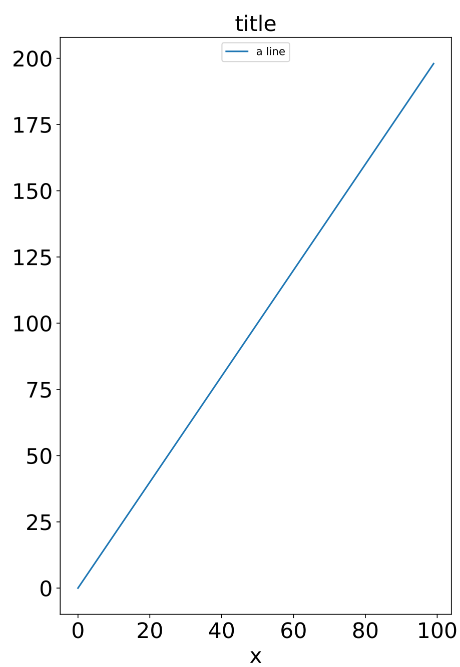
Test your code with examples!

Problem 1.

(a) Create a figure object called fig using `plt.figure()`.

(b) Use `add_axes` to add an axis to the figure canvas at `[0.1, 0.1, 0.8, 0.8]`. Call this new axis ax.

(c) Plot (x, y) on that axes and set the labels and titles to match the plot below:



Problem 2.

(a) Create a figure object and put two axes ax1 and ax2 on it which are located at `[0.1, 0.1 , 0.8, 0.8]` and `[0.2, 0.5, .2, .2]`, respectively.

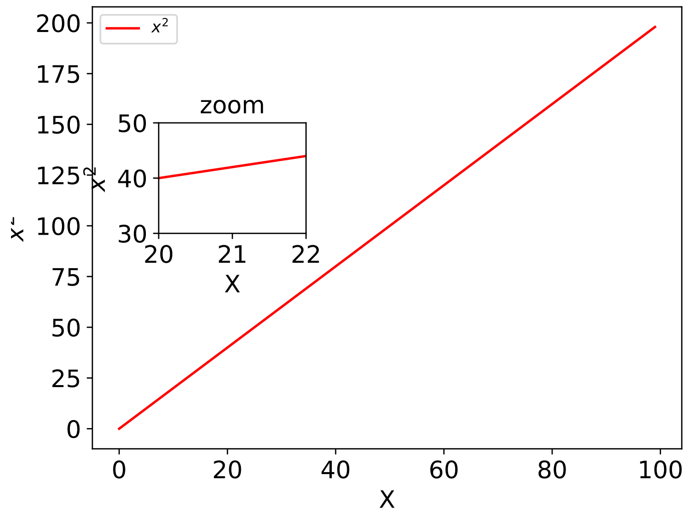


Figure 1: Problem 2

(b) Reproduce Figure 1!

Problem 3.
Use `plt.subplots` to create the following plot. Notice that the columns share the same x range. Also the location of the legends should be identical to the one in Figure **??**.

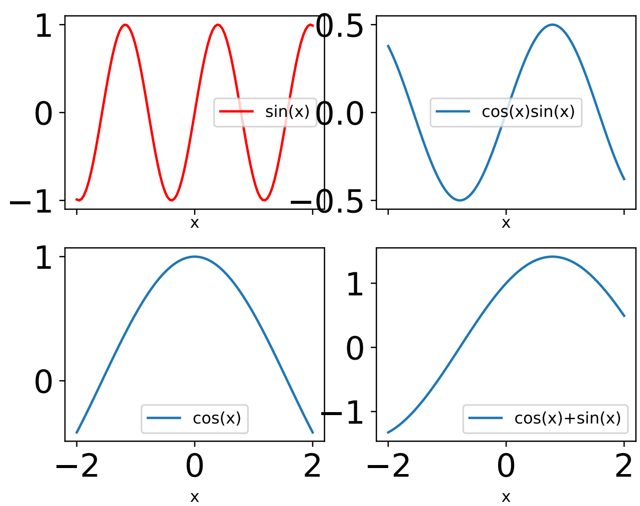


Figure 2: Problem 3

Problem 4. Consider the real nodes $x_1 < \cdots < x_n$ and function values $y_j \in \mathbf{R}$. Then, linear algebra provides a unique polynomial $p(t) = \sum_{j=1}^{n} a_j t^{j-1}$ of degree $n-1$, such that $p(x_j) = y_j$ for all $j = 1, \ldots, n$. Pick a fixed evaluation point $t \in \mathbf{R}$. The *Neville-algorithm* is able to compute the point evaluation $p(t)$ without computing the vector of coefficients $a \in \mathbf{R}^n$. It consists of the following steps: First, define for $j, m \in \mathbb{N}$ with $m \geq 2$ and $j + m \leq n + 1$ the values

$$\begin{aligned} p_{j,1} &:= y_j, \\ p_{j,m} &:= \frac{(t - x_j)p_{j+1,m-1} - (t - x_{j+m-1})p_{j,m-1}}{x_{j+m-1} - x_j}. \end{aligned}$$

It can be shown that $p(t) = p_{1,n}$, that is, the function value of p at t can be computed by $p_{1,n}$. Write a function `neville` which computes $p(t)$ for a given evaluation point $t \in \mathbf{R}$ and vectors $x = (x_1, \dots, x_n), y = (y_1, \dots, y_n) \in \mathbf{R}^n$. To do that, you can use the following scheme

$$\begin{array}{ccccccccccccc}
y_1 & = & p_{1,1} & \longrightarrow & p_{1,2} & \longrightarrow & p_{1,3} & \longrightarrow & \dots & \longrightarrow & p_{1,n} & = & p(t) \\
 & & & \nearrow & & \nearrow & & & & \nearrow & & & \\
y_2 & = & p_{2,1} & \longrightarrow & p_{2,2} & & & & & & & & \\
 & & & \nearrow & & & & \nearrow & & & & & \\
y_3 & = & p_{3,1} & \longrightarrow & \vdots & & & & & & & & \\
\vdots & & \vdots & & \vdots & \nearrow & & & & & & & \\
y_{n-1} & = & p_{n-1,1} & \longrightarrow & p_{n-1,2} & & & & & & & & \\
 & & & \nearrow & & & & & & & & & \\
y_n & = & p_{n,1} & & & & & & & & & &
\end{array} \tag{1}$$

One easy way to implement this scheme is by building a matrix with entries $(p_{j,m})_{j,m=1}^n$. For testing, take an arbitrary polynomial resp. nodes, and compute $y_j = p(x_j)$.

Problem 5. Study the documentation of `mlab.quiver3d(ux,uy,uz,vx,vy,vz)` of the mayavi module. In this exercise we want to plot the (outward pointing) unit normal vector field along an ellipsoid

$$E^2 := \{(x, y, z) : \ ax^2 + by^2 + cz^2 = 1\}.$$

In order to plot this vector field consider the parametrisation of the ellipsoid:

$$\varphi : (u, v) \to (a \sin(u) \cos(v), b \sin(u) \sin(v), c \cos(v)) : [0, \pi) \times [0, 2\pi) :\to E^2 \subset \mathbf{R}^3,$$

The functions `(ux,uy,uz)` are the component functions of φ and the functions `(vx,vy,vz)` are the component functions of $\partial_u \varphi \times \partial_v \varphi / \|\partial_u \varphi \times \partial_v \varphi\|_2$. Also put a nice coordinate system into the plot. The output in case of $a = b = c = 1$ should look like Figure 3.

Problem 6. Write a function `saveMatrix` which takes a matrix $A \in \mathbf{R}^{d \times d}$ and writes it into a file `matrix.dat` via `open`. Write another function `loadMatrix`, which takes a string `'matrix.dat'` and reads the file with `open` and stores the data into numpy array. Compare your result with `numpy.savetxt` and `numpy.loadtxt`.

Problem 7. Use the matplotlib function `plt.quiver` to visualise the vector field $F : \mathbf{R}^2 \to \mathbf{R}^2$ given by

$$F(x, y) := \begin{cases} (1, 1) + (-y, x) & \text{if } x > 0 \\ -(1, 1) + (y, -x) & \text{if } x < 0 \end{cases}.$$

Plot the vector field on $[-1, 1] \times [-2, 1]$ and make nice captions and legends. Make sure the font size of your plot is not too small.

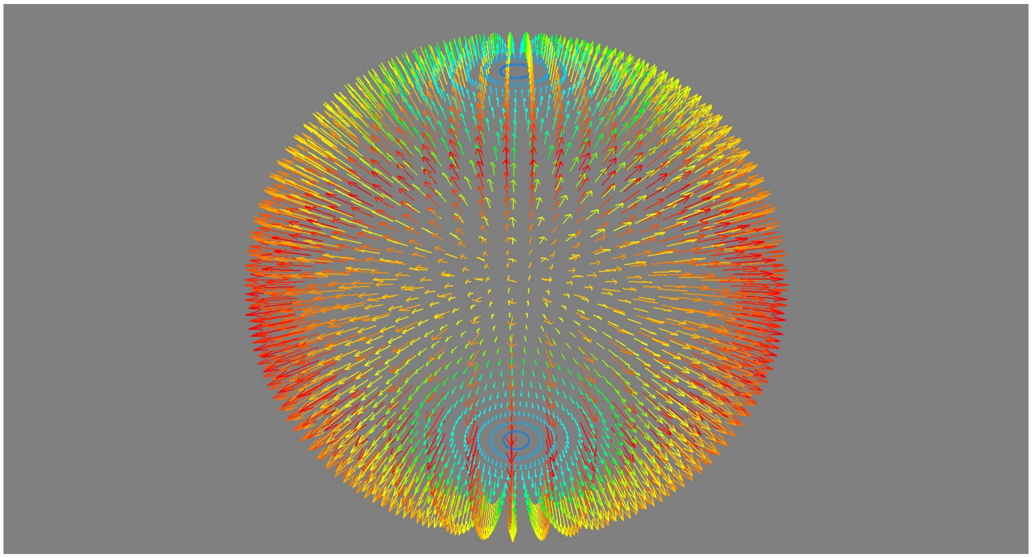


Figure 3: Problem 5

Problem 8. Use the matplotlib function `plt.scatter` to produce the following plots.