INTRODUCTION TO COMPUTATIONAL PHYSICS

Second Assessment for Teaching Block 1

U24200 –Academic Session 2019–2020

Instructions

a) The proportion of marks for this coursework

b) **You must undertake this assignment individually.**

c) Submission method: **by Moodle through the available dropbox**.

d) Submission deadline: **August 7, 2020**

1 Reminder about interpolation (already seen in Coursework 1)

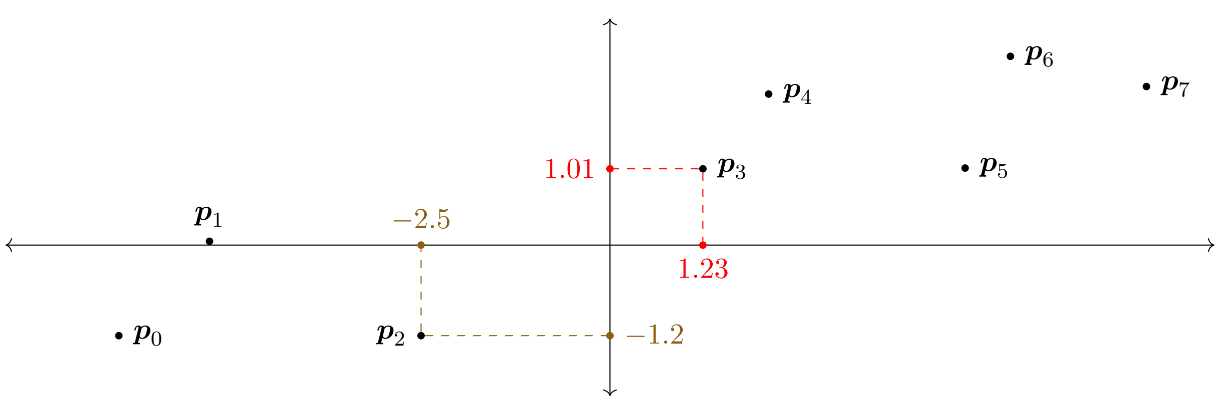
Interpolation consists in building a simple function f (here, a *polynomial*, $f(x) = a_0 + a_1 x + \cdots + a_k x^k$) whose graph curve passes through a given set of points

$$\boldsymbol{p}_0 = (x_0, y_0)\,, \quad \boldsymbol{p}_1 = (x_1, y_1)\,, \quad \ldots, \quad \boldsymbol{p}_m = (x_m, y_m)\,,$$

i.e. such that $f(x_0) = y_0,\ f(x_1) = y_1,\ \ldots,\ f(x_m) = y_m$. This can be seen, for instance in the case in which

$$\begin{array}{llll} \boldsymbol{p}_0 = (-6.5, -1.2)\,, & \boldsymbol{p}_1 = (-5.3, 0.05)\,, & \boldsymbol{p}_2 = (-2.5, -1.2)\,, & \boldsymbol{p}_3 = (1.23, 1.01)\,, \\ \boldsymbol{p}_4 = (2.1, 2)\,, & \boldsymbol{p}_5 = (4.7, 1.02)\,, & \boldsymbol{p}_6 = (5.3, 2.5)\,, & \boldsymbol{p}_7 = (7.1, 2.1)\,. \end{array}$$

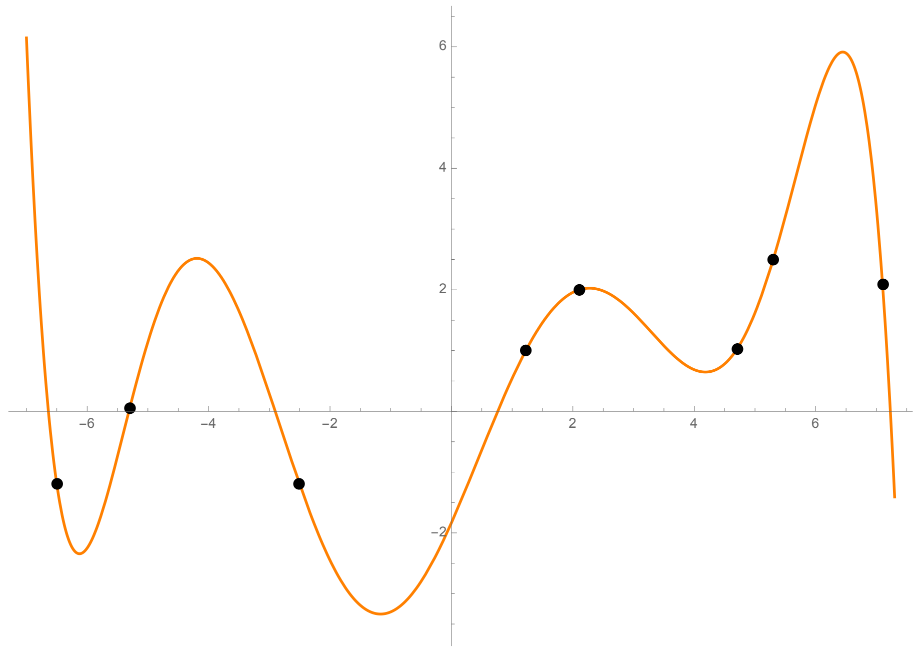
Let us first represent these points graphically:



We wish to find a polynomial whose graph traverses each one of these points. After you have finished your program, you will find that such a function can be approximated as

$$f(x) = -0.000175066x^7 + 0.000289133x^6 + 0.014541x^5 - 0.0211649x^4 - 0.34218x^3 + 0.477095x^2 + 2.24967x - 1.83489,$$

and has the shape shown in the next page. We draw it along with the original points $\boldsymbol{p}_i$.



The following result is fundamental to our purpose:

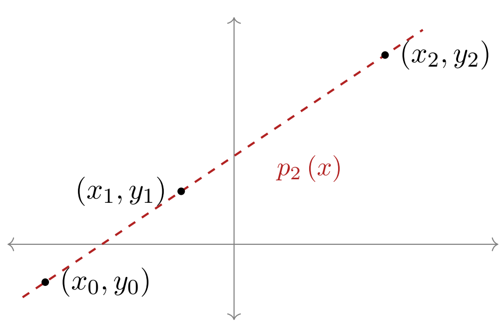
Theorem (**Existence and uniqueness of the interpolating polynomial**). *Let* $(x_0, y_0), \ldots, (x_n, y_n)$ *be* $n+1$ *points in the plane such that* x_i *are pairwise different* ($x_i \neq x_j$ *if* $i \neq j$)*. Then there exists a unique polynomial of degree* at most n, $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$, *interpolating these points, i.e. such that*

$$p(x_0) = y_1, \quad p(x_1) = y_2, \quad \ldots, \quad p(x_n) = y_n. \tag{1}$$

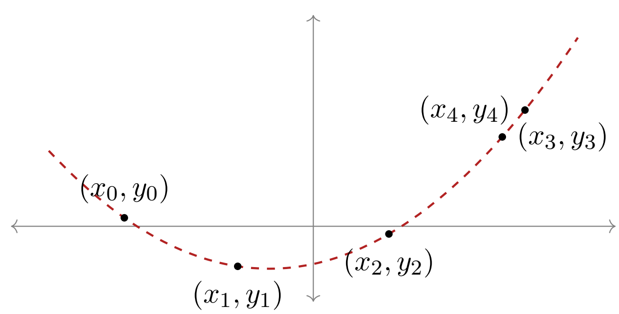
Remarks.

1. A well-known fact in Geometry, namely that any two points are traversed simultaneously by a unique line, is a particular case of the above: a line is the graph of a <u>degree-one</u> polynomial $y = ax + b$, and using the above two notation we would have $\underline{n=1}$ for two points $(x_0, y_0), (x_1, y_1)$.

2. n is the *maximum* value (hence an upper bound) of the degree of the interpolating polynomial but the actual degree could be less than n depending on the disposition of the points. For instance,



Three points hence $n = 2$ but they are *aligned*, thus interpolating polynomial is that line: $p(x) = a + bx + \boxed{0}\, x^2$



Five points ($n = 4$) but they are *all in the same parabola*, thus interpolating polynomial is that parabola: $p_4(x) = A + Bx + Cx^2 + D\boxed{0}\, x^3 + E\boxed{0}\, x^4$

3. Interpolation (and a similar concept called *extrapolation*) will be useful whenever you are given a table of experimental data and need to guess theoretical outputs for values not belonging to that table.

There are many ways of computing the interpolating polynomial p_n of $n+1$ points (but remember: the polynomial, due to its uniqueness, *is still the same* and depends only on the points chosen). Most notably:

- solving the linear system defined by the interpolating conditions
- method of Lagrange polynomials;
- Newton's method of divided differences;
- Aitken's method, Neville's method, etc.

In Coursework 1 we saw the third method. We will now focus on the first method for this Coursework.

2 Solving a linear system

The polynomial $p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ must verify (1), i.e.

$$\begin{aligned} a_n x_0^n + a_{n-1} x_0^{n-1} + \cdots + a_1 x_0 + a_0 &= y_0, && (2) \\ a_n x_1^n + a_{n-1} x_1^{n-1} + \cdots + a_1 x_1 + a_0 &= y_1, && (3) \\ &\vdots && (4) \\ a_n x_n^n + a_{n-1} x_n^{n-1} + \cdots + a_1 x_n + a_0 &= y_n, && (5) \end{aligned}$$

these are $n+1$ linear equations with $n+1$ unknowns $a_0, \ldots, a_n$. It can be proved that the matrix of this linear system, called a **Vandermonde matrix**,

$$V = \begin{pmatrix} x_0^n & x_0^{n-1} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \ldots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \ldots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \end{pmatrix} \tag{6}$$

has a determinant $\neq 0$, thus the linear system defined by (2), (3), ..., (5) has a unique solution $(a_n, a_{n-1}, \ldots, a_0)$. These are the coefficients of the polynomial we are looking for.

Sometimes you will also see the matrix written the other way,

$$\begin{pmatrix} 1 & x_0 & \ldots & x_0^{n-1} & x_0^n \\ 1 & x_1 & \ldots & x_1^{n-1} & x_1^n \\ 1 & x_2 & \ldots & x_2^{n-1} & x_2^n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_n & \ldots & x_n^{n-1} & x_n^n \end{pmatrix} \tag{7}$$

in which case the solution will be written in the opposite order: $(a_0, a_1, \ldots, a_{n-1}, a_n)$. This is just as correct as if you used (6).

3 Coursework exercises

1. These will be the functions used to compute p_n:

(i) Implement a function `Cramer` to solve linear systems with an invertible matrix. If you check through the Moodle material for TB1 you will find this, but make sure you write it in your own style (don't just copy it) and test it on separate systems before adapting it to your coursework.

(ii) Write up a function `VandermondeSolution` with the following arguments:

- an array of abscissae $\boldsymbol{x} = (x_0, \ldots, x_n)$ that have been previously checked to be pairwise different;
- an array of ordinates $\boldsymbol{y} = (y_0, \ldots, y_n)$ comprising the other half of the table we wish to interpolate.

and returning matrix V and the solution $\boldsymbol{a}$ of the system $V\boldsymbol{a} = \boldsymbol{y}$, where

- V is the Vandermonde matrix for $x_0, \ldots, x_n$. Feel free to choose either version: (6) or (7).
- $\boldsymbol{a} = (a_n, \ldots, a_0)$ or $\boldsymbol{a} = (a_0, \ldots, a_n)$ is the array of coefficients of the desired $a_n x^n + \cdots + a_0$.

(iii) Write up a function `Horner` with the following arguments:

- an array of terms $a_0, \ldots, a_n$,
- and a variable floating-point number x,

and returning the output $a_0 + x\,(a_1 + x\,(a_2 + \ldots))$ where you minimise the number of operations used. You can adapt the generalised Horner's method seen in Coursework 1, only that in this simpler case you will multiply each new block by x instead of by the successive $x - x_{n-1}$, $x - x_{n-2}$, etc.

(i), (ii), (iii) above are enough to interpolate a given set of points and compute $p_n(x)$ for any given x.

2. And these constitute an example of how to apply Exercise 1:

(i) Create a text file `table.txt` and fill it with pairs of numbers distributed in two columns:

$$\left.\begin{array}{ll} \texttt{x0} & \texttt{y0} \\ \texttt{x1} & \texttt{y1} \\ \vdots & \vdots \end{array}\right\} \qquad \text{make sure } \texttt{x0, x1,...} \text{ are all different} \tag{8}$$

(ii) Write a function `read_from_file` with two arguments: a file `name` (in string form) and a tolerance `tol`. The function will read two-column data such as (8) externally from file `name` and will check that there are no abscissae `x0`, `x1`, ... (in any order) differing from each other by less than `tol` in absolute value.

(iii) Write a function `write_to_files` one of whose arguments must be the number of points of the interpolating polynomial you wish to plot. It will write these points, again in the same disposition as (8), in a new file called `function_to_plot.txt`. It will also store the Vandermonde matrix, ordinates $y_0, \ldots, y_n$ and coefficients of your polynomial in another file named `matrix_and_vectors.txt`.

All that is needed for you to upload to Moodle is:

- One Python file `UPXXXXXX.py` containing your student number.
- One Jupyter library showcasing applications of (and any necessary comments about) your Python file.
- One file `table.txt` containing a sample table $\boxed{x_i \mid y_i}$ to interpolate.
- One file `matrix_and_vectors.txt` containing the Vandermonde matrix V, the ordinates $\boldsymbol{y}$ and the solution $\boldsymbol{a} = (a_n, \ldots, a_0)$ to the system $V\boldsymbol{a} = \boldsymbol{y}$, i.e. the coefficients of p_n.
- One file `function_to_plot.txt` with a larger (> 1000) set of points interpolating those in `table.txt`.
- A plot of the data in `function_to_plot.txt` (you can skip this if your Jupyter file contains plots).

Mark scheme: if your program

- does none of the following: run without errors, produce a plot or a numerical output: **0-50 marks**
- fails to do *at least* one of the things described above: **30-60 marks**
- runs without errors and produces *one* plot, but lacks what is said in the items below: **50-70 marks**
- same as item above and the interpolating process is correct for arbitrary degrees: **60-90 marks**
- same as item above and a good Jupyter notebook is present: **80-100 marks**

Mark ranges overlap because the global layout of your program and your ability to detect glitches will also count.