Algèbre pour le graphique et la CAO
L3-MI

Fascicule des TP

TP1A : Lists - Numpy – Matplotlib

TP2A : Gauss – Conditioning – Gram-Schmidt & orthogonal matrices

TP2B : Numerical integration & Fourier series

TP3A : Interpolation de Lagrange 1D & 2D – base de Lagrange

TP3B : Interpolation de Lagrange – base de Newton

TP3C : Lagrange interpolation in monomial basis with scipy

TP4 : Hermite interpolation

TP5A : Interpolation spline cubique C2 – cas explicite & paramétrique

TP5B : Periodic interpolating cubic splines

TP5C : Splines exponentielles sous tension

TP6A : Approximation aux moindres carrées – Erreur résiduelle

TP7 : Least squares approximation with integral constraint

TP8 : Smoothing splines

TP9A : Subdivision

TP9B : Intersection de courbes de Bézier par Exclusion et Subdivision

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP1-A : Lists - Numpy - Matplotlib

Exercise 1

Sieve of Eratosthene. Write a Python script allowing to determine all the prime numbers lower than $N_{\max}$, with an algorithm using the "sieve of Eratosthene".

```
Enter N_max : 154
[  2   3   5   7  11  13  17  19  23  29  31  37  41
43
 47  53  59  61 67  71  73  79  83  89  97 101 103 107
 109 113 127 131 137 139 149 151]
```

Exercise 2

Write in the simplest way the following vectors and matrices.

$$M1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 8 & 6 & 4 & 2 & 0 \\ 8 & 6 & 4 & 2 & 0 \end{pmatrix} \quad M3 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

$$M4 = \begin{pmatrix} 2. & 1. & 0. & 0. & 0. & 0. & 0. & 0. \\ 1. & 4. & 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 4. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 4. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 4. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 1. & 4. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 4. & 1. \\ 0. & 0. & 0. & 0. & 0. & 0. & 1. & 2. \end{pmatrix}$$

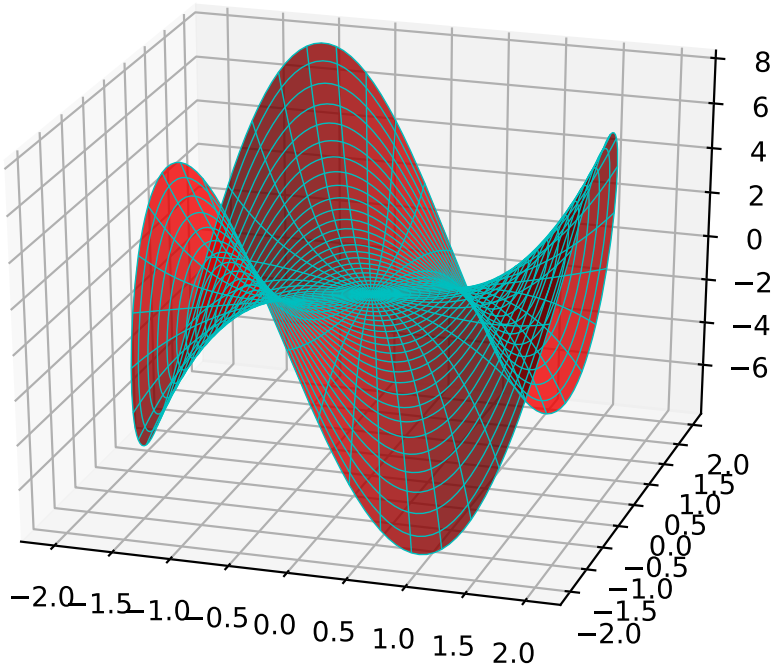
Exercise 3

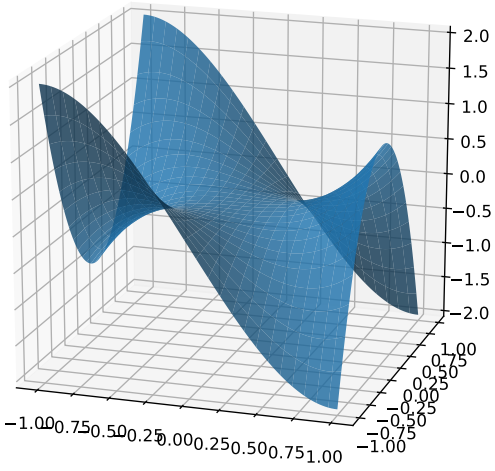
Consider the Monkey saddle defined by the parametric representation

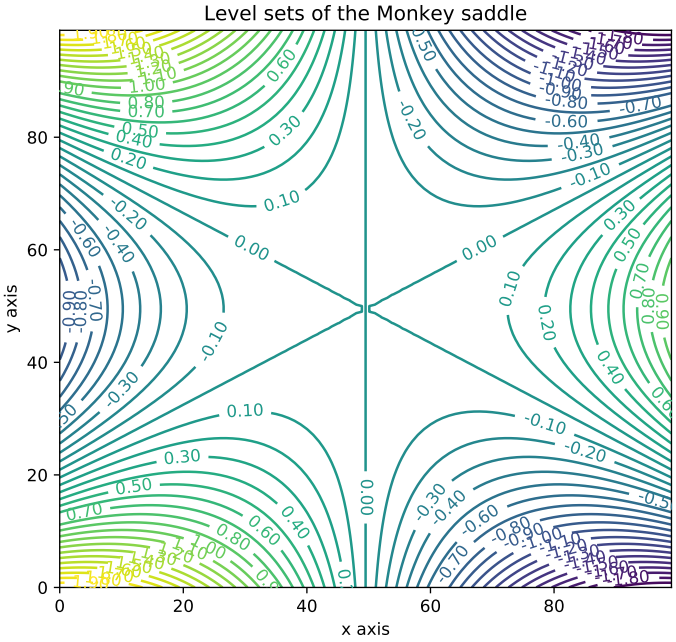
$$(u, v) \in [0, 2] \times [0, 2\,\pi] \quad \mapsto \quad \begin{cases} x &= u\,\cos(v) \\ y &= u\,\sin(v) \\ z &= u^3\,\cos(3v) \end{cases}$$

and admitting also the explicit equation : $z = x^3 - 3\,x\,y^2$

1. *Plot this surface using its parametric representation.*
2. *In the same figure with two subplots,*
   a) plot the surface using its explicit equation,
   b) plot its level sets according to the z-coordinate.







Le compte rendu de ce TP consistera en un unique fichier python dont le nom sera `TP1A_NOM1_NOM2.py`
Ce fichier python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2, puis le script Python demandé pour chacun des exercices.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP2-A : Gauss - Conditioning - Gram-Schmidt & orthogonal matrices

---

This labwork refers to chapter 2 on "*Prerequisites in Maths*" and involves three parts.

1. Experimentation of the conditioning on the well-known example given in the course.
2. Implementation of the full-pivot Gauss reduction, application to the resolution of a linear system $Ax = b$ and comparizon with `numpy.linalg.solve` and the "`inverse` approach". We will assume that the matrix A is non singular.
3. Construction of orthogonal matrices with the Gram-Schmidt process (and evaluation of their condition number)

Part 1 : Conditioning –

- load the file `TP2AConditioningStudents.py`.
- *Experiment this file and answer the 3 questions directly in the python script.*

Part 2 : Gauss reduction & triangular resolution –

- load the file `TP2AGaussStudents.py`
- *Complete this file (Full pivot Gauss reduction and triangular resolution) and then to compare this Gauss resolution with the two other methods proposed in the script.*
  Be aware that this "full strategy" modifies the order of the variables...

Part 3 : Gram-Schmidt & orthogonal matrices –

- load the file `TP2AGramSchmidtStudents.py`
- *Complete this file and experiment the construction of orthogonal matrices with the Gram-Schmidt algorithm.*

Le compte rendu de ce TP consistera en 3 scripts Python dont les noms seront `TP2A1_NOM1_NOM2.py`, `TP2A2_NOM1_NOM2.py`, `TP2A3_NOM1_NOM2.py` selon l'ordre donné ci-dessus. Chaque fichier python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2, puis le script Python complété.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP2-B : Numerical integration & Fourier series

This labwork refers to Sections 11 & 12 of Chapter 2

- load the file `TP2BFourierStudents.py`.
- *Complete this file according to the following information.*

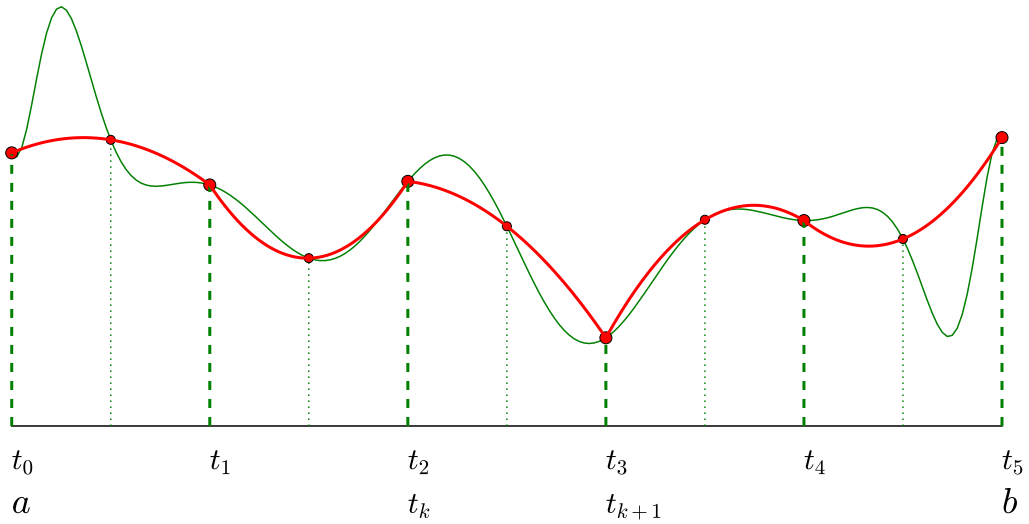
Simpson method

Given a continuous fonction f on $[a,b]$ (the green function below) and a subdivision of the interval $[a,b]$

$$a = t_0 < t_1 < t_1 < \cdots < t_{n-1} < t_n = b$$

we approximate on each interval $[t_k, t_{k+1}]$ the graph of the function f by a parabola, that is by the polynomial $\tilde{f}_k(t)$ of degree 2 interpolating f at points

$$t_k\,, \quad \frac{t_k + t_{k+1}}{2}\,, \quad t_{k+1}.$$



Considering a uniform subdivision $t_{k+1} - t_k = h > 0$ for $k = 0, \ldots, n-1$, we prove that

$$\int_{t_k}^{t_{k+1}} \tilde{f}_k(t)\,dt \;=\; \frac{h}{2}\left[\frac{1}{3}\,f(t_k) + \frac{4}{3}\,f\Big(\frac{t_k + t_{k+1}}{2}\Big) + \frac{1}{3}\,f(t_{k+1})\right]$$

which leads to the *Simpson* approximation

$$\int_a^b f(t)\,dt \;\simeq\; h\sum_{k=0}^{n-1}\left[\frac{1}{6}\,f(t_k) + \frac{2}{3}\,f(\frac{t_k + t_{k+1}}{2}) + \frac{1}{6}\,f(t_{k+1})\right]$$

$$\simeq \underbrace{\frac{h}{6}\left(f(a) + 2\sum_{k=1}^{n-1} f(a + k\,h) + 4\sum_{k=0}^{n-1} f\Big(a + k\,h + \frac{h}{2}\Big) + f(b)\right)}_{I_S(f)} \qquad (1)$$

where $I_S(f)$ denotes the *(composite) Simpson formula.*

Fourier series

Given a T-periodic piecewise C^0 function f, we propose to approximate this function by its Fourier series at order N :

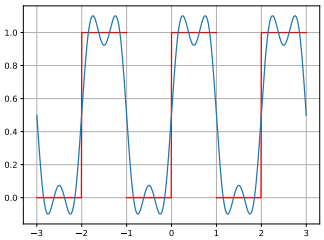
$$S_N(f)(x) = a_0 + \sum_{n=1}^{N} a_n \cos(nwx) + \sum_{n=1}^{N} b_n \sin(nwx) \qquad \text{with } w = \frac{2\pi}{T}$$

$$a_0 = \frac{1}{T}\int_0^T f(t)\,dt\,, \quad a_n = \frac{2}{T}\int_0^T f(t)\cos(nwt)\,dt\,, \quad b_n = \frac{2}{T}\int_0^T f(t)\sin(nwt)\,dt\,, \quad n \geq 1$$

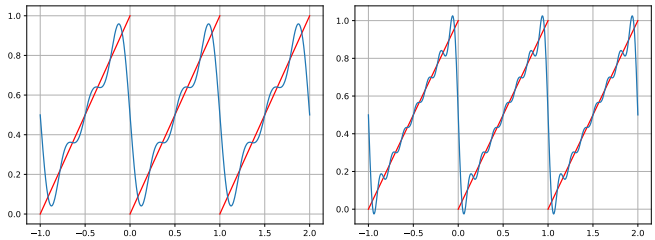
The objective is here to determine the partial sums $S_N(f)(x)$ of the following functions (as shown in the different figures).

$f_0(x) = 0$	if $x \in]-1, 0]$ and $f_0(x) = 1$ if $x \in]0, 1]$,	$T = 2$,
$f_1(x) = x$	on $]0, 1]$,	$T = 1$,
$f_2(x) = x^2$	on $[-1, 1]$,	$T = 2$,
$f_3(x) = \sqrt{x}$	on $]0, 1]$,	$T = 1$.

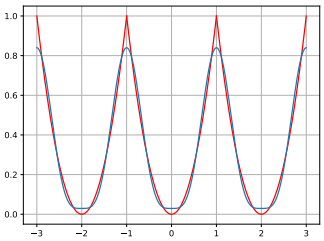
The Fourier coefficients a_n and b_n will be evaluated with the Simpson method (with $n = 8$ leading to a sampling of the function on 2^8 intervals).



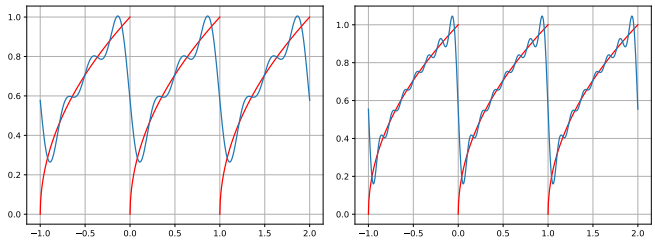
function $S_N(f_0)$ with $N = 3$ and $N = 7$



function $S_N(f_1)$ with $N = 3$ and $N = 7$



function $S_N(f_2)$ with $N = 2$ and $N = 3$



function $S_N(f_3)$ with $N = 3$ and $N = 7$

Le compte rendu de ce TP consistera en le script Python initial complété et commenté dont le nom sera `TP2B_NOM1_NOM2.py`, Ce fichier python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2, puis le script Python complété.
L'éxécution de ce script devra permettre de reproduire directement les figures ci-dessus avec les mêmes données numériques.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP3-A : Interpolation de Lagrange 1D & 2D – base de Lagrange

L'interpolation de Lagrange consiste à faire "*passer*" un polynôme de degré adéquat par des points (x_i, y_i) où les abscisses x_i sont 2 à 2 distinctes. Les méthodes différent selon la base choisie pour déterminer ce polynôme. Les bases de Lagrange et de Newton sont les plus usuelles. On considère ici la base de Lagrange.

Télécharger les fichiers `TP3ALagrange1DStudents.py` et `TP3ALagrange2DStudents.py` que vous complèterez.

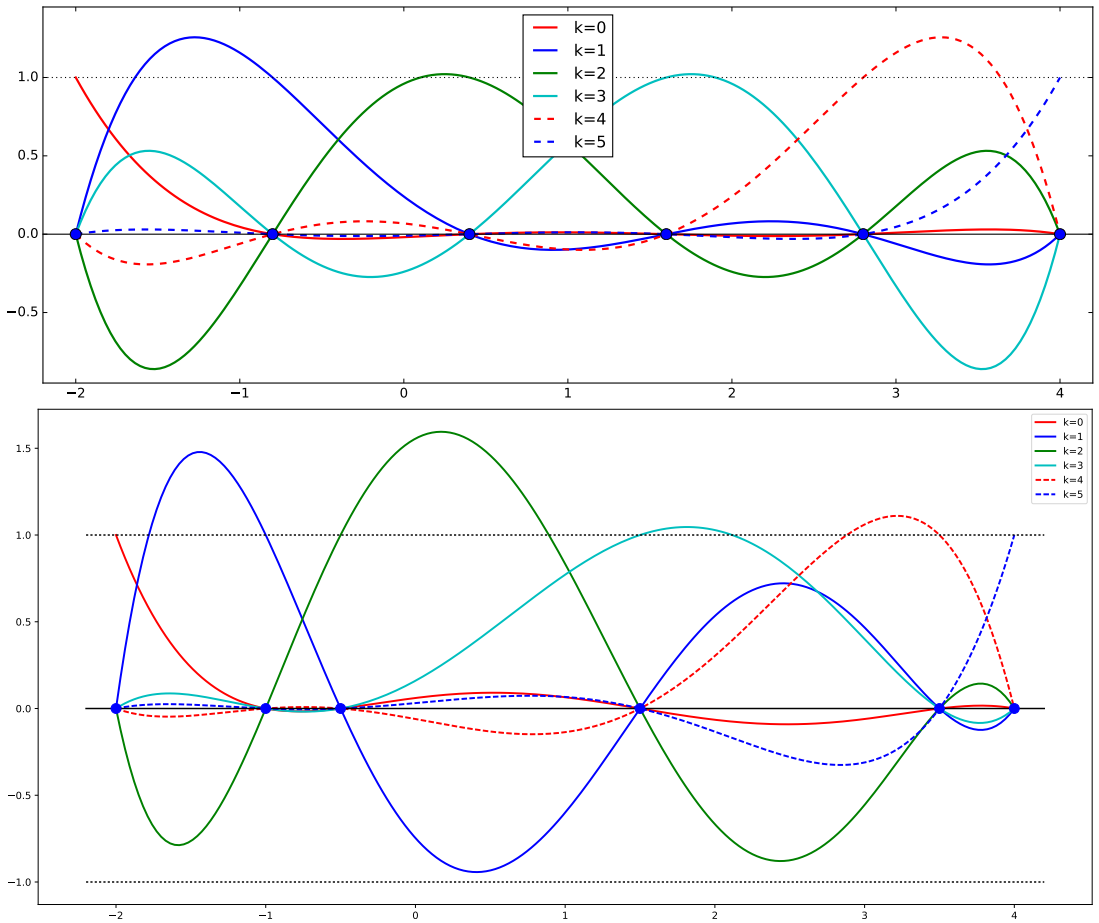
Partie 1 : Base de Lagrange

Soit $n \in \mathbb{N}$. On se donne $n+1$ valeurs réelles deux à deux distinctes dans un intervalle $[a, b]$. Ces valeurs x_i ne seront pas nécessairement équidistantes.

Ecrire un script Python permettant de tracer les $n+1$ polynômes de Lagrange

$$L_i(x) = \prod_{\substack{0 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j}, \quad 0 \le i \le n,$$

sur l'intervalle $[a, b]$, tel que représenté sur les deux figures ci-dessous.



Lagrange polynomials $L_i(x)$ of degree 5, associated with points uniformly and non uniformly distributed in the interval $[-2, 4]$.

Partie 2 : Interpolation de Lagrange en 1D

1. Ecrire un script Python permettant de tracer le graphe de la fonction f (nommée `ftest1`) définie sur $[-2, 4]$ par $f(x) = \cos(1 - x^2)\, e^{-x^2+3\,x-2}$.

2. Ecrire un script Python permettant de tracer le polynôme d'interpolation

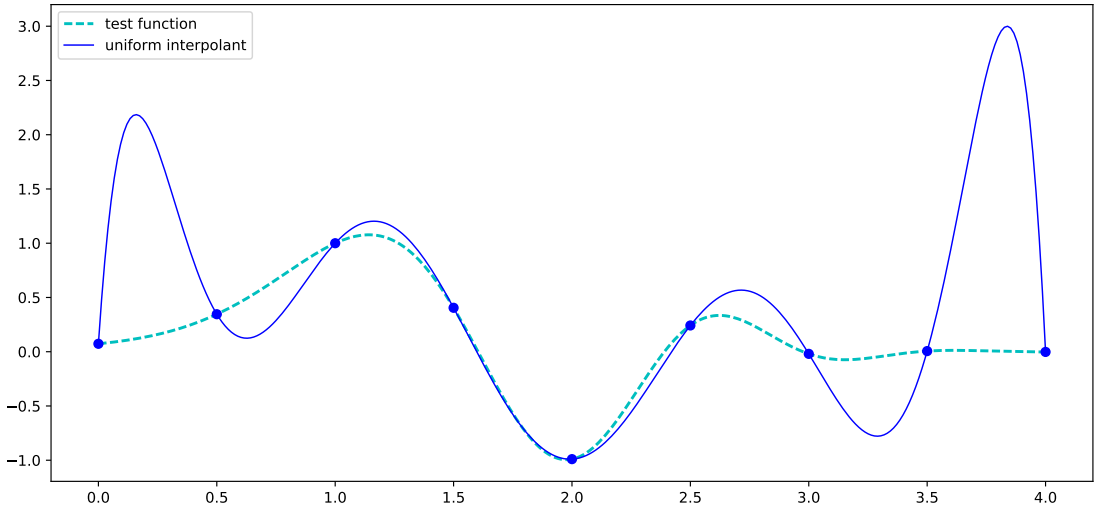
$$P_n(x, f) = \sum_{i=0}^{n} f(x_i)\, L_i(x)$$

de la fonction f aux points d'abscisses équidistantes

$$x_i = a + i\, \frac{b-a}{n}\,, \quad 0 \leq i \leq n\,,$$

avec $a = -2$, $b = 4$, et $n = 8$.

On marquera sur le graphe de la fonction f les point d'interpolations $(x_i, f(x_i))$.



Lagrange interpolant (solid blue line) of degree 8 *of the test function* f*, associated with uniform data in the interval* $[0, 4]$.

3. Etudier le comportement du polynôme d'interpolation $P_n(x, f)$ lorsque le degré n augmente [*réponse directement dans le script Python en commentaire*].

4. Reprendre cette même étude, c'est-à-dire le comportement du polynôme d'interpolation lorsque le degré n augmente avec la nouvelle fonction test g (et nommée `ftest2`) définie sur $[-1, 1]$ par $g(x) = \dfrac{1}{1 + 25x^2}$.

Partie 3 : Interpolation de Lagrange sur une grille 2D

The goal is to write a Python script allowing to interpolate data on a 2D grid. Precisely, given data points x_i and y_j :

$$a \leq x_0 < x_1 < \cdots < x_n \leq b$$

$$c \leq y_0 < y_1 < \cdots < y_m \leq d$$

and a family of $(n+1)(m+1)$ real numbers z_{ij}, $i = 0, \ldots, n,\ \ j = 0, \ldots, m$, we are looking for a function

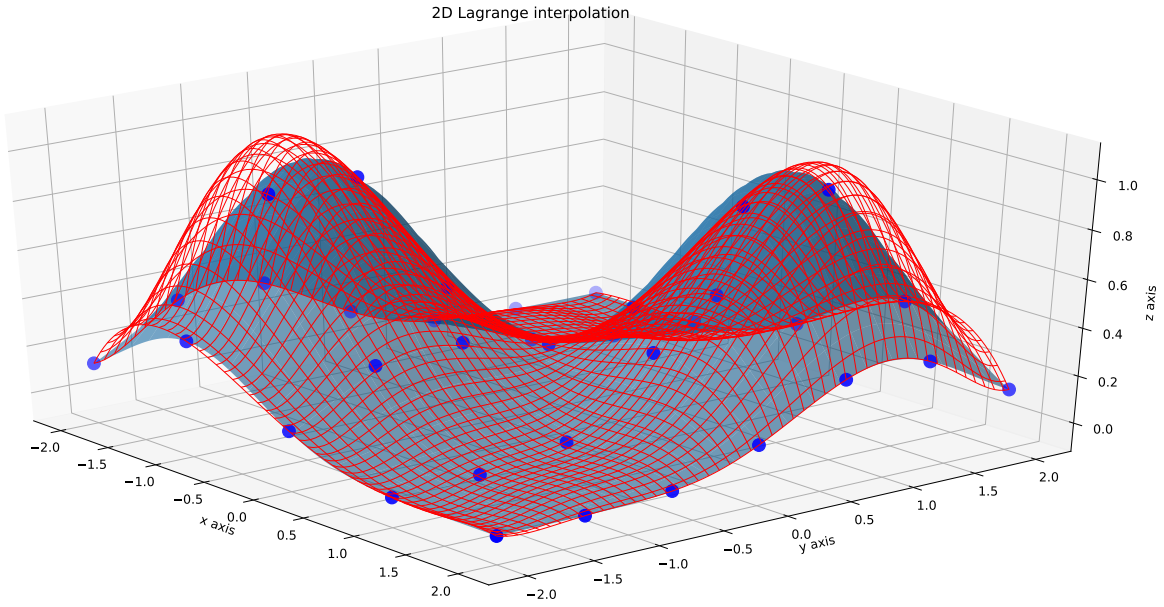
$$f : \quad [a,b] \times [c,d] \longrightarrow \mathbb{R}$$

such that $$f(x_i, y_j) = z_{ij}, \quad i = 0, \ldots, n, \quad j = 0, \ldots, m.$$

For that purpose, we will consider the tensor product Lagrange interpolation :

$$L(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} z_{ij}\, L_{x,i}(x)\, L_{y,j}(y)$$

where polynomials $L_{x,i}(x)$, $i = 0, \ldots, n$ and $L_{y,j}(y)$, $j = 0, \ldots, m$ are the Lagrange polynomials respectively associated with the sequences $x_0, x_1, \ldots, x_n$ and $y_0, y_1, \ldots, y_m$.
This interpolating surface $z = L(x,y)$ is polynomial of bi-degree (n,m).



The blue-grey surface is the graph of the function $\varphi(x,y) = e^{-(x+1)^2-(y+1)^2} + e^{-(x-1)^2-(y-1)^2}$, $(x,y) \in [-2,2]\times[-2,2]$. — *The red wireframe surface is the tensor product Lagrange interpolant of data* $\big(x_i, y_j, z_{ij} = \varphi(x_i, y_j)\big)$, $i = 0, \ldots, 4$, $j = 0, \ldots, 6$, *with* `xi = np.linspace(-2,2,5)` *and* `yj = np.linspace(-2,2,7)`, *which represents a sampling of the surface* φ *at points* (x_i, y_j).

Le compte rendu de ce TP consistera en 2 scripts Python dont les noms seront `TP3A_1D_NOM1_NOM2.py`, `TP2A_2D_NOM1_NOM2.py`.
Chaque fichier python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2, puis les scripts complétés.
En particulier, l'analyse demandée sur le comportement de l'interpolation sera rédigée en commentaire dans le script Python.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP3-B : Interpolation de Lagrange – base de Newton

L'interpolation de Lagrange consiste à faire "*passer*" un polynôme de degré adéquat par des points (x_i, y_i) où les abscisses x_i sont 2 à 2 distinctes. Les méthodes différent selon la base choisie pour déterminer ce polynôme. Les bases de Lagrange et de Newton sont les plus usuelles. On considère ici la base de *Newton*. — `Par abus, on parlera d'interpolation de Newton`.

Dans la première partie, on suppose que l'ensemble des $n+1$ données d'interpolation (x_i, y_i) sont connus initialement. Dans la deuxième partie on considère *l'ajout dynamique* de données d'interpolation. Et enfin dans une dernière partie on considère le cas de l'interpolation paramétrique.

Télécharger les fichiers `TP3B_NewtonStudents.py` et `TP3B_NewtonParamStudents.py` que vous complèterez selon les instructions données ci-dessous.

Partie 1 : interpolation de Newton

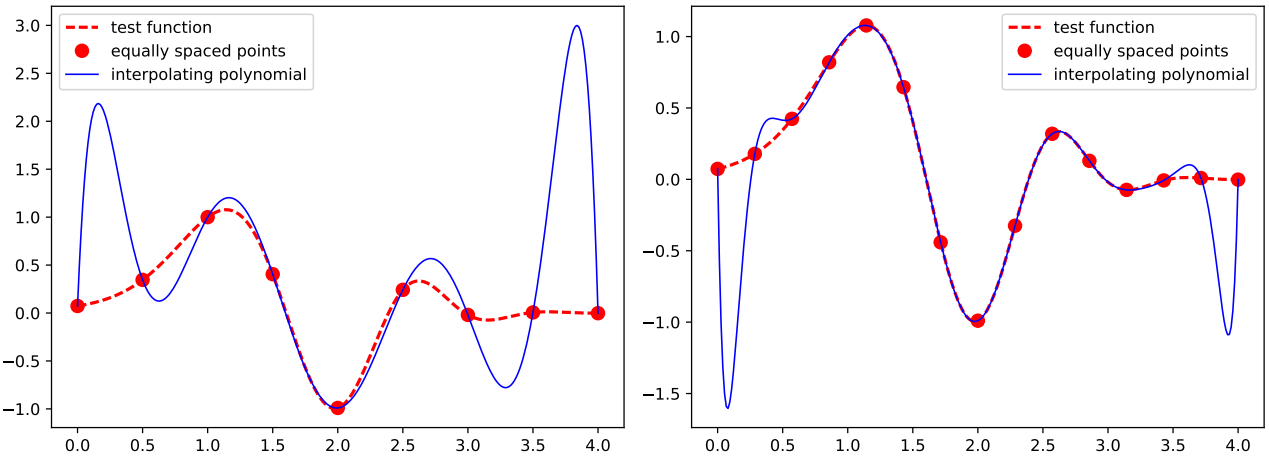
Il s'agit ici de l'interpolation de $n+1$ points dans la base de Newton et donc en particulier du calcul des différences divisées associées.
Dans le script Python fourni on remarquera qu'on a utilisé la formule

$$P_n(x) = \delta[x_0] \; + \sum_{k=1}^{n} \delta[x_0, \ldots, x_k] \;\; (x - x_0) \cdots (x - x_{k-1}) \tag{1}$$

et que ce polynôme d'interpolation $P_n(x)$ est évalué selon le schéma de Horner dans la base de Newton. Par exemple, pour $n = 4$, et en notant $\delta_k = \delta[x_0, \ldots, x_k]$, on écrit

$$P_4(x) = \delta_0 + (x - x_0)\Big[\delta_1 + (x - x_1)\Big[\delta_2 + (x - x_2)\big[\delta_3 + (x - x_3)\big[\,\delta_4\,\big]\big]\Big]\Big]$$

Complétez le script Python et reproduire ensuite la figure ci-dessous (qui est aussi celle du cours). La fonction test est précisée dans le script Python. A noter que l'algorithme de Horner pour les polynômes est introduit au début du chapitre 2 et fait également l'objet d'un exercise du chapitre 3.



Partie 2 : interpolation dynamique de Newton

2A) Complétez le script Python afin de permettre une interpolation dynamique tel que décrite ci-dessous.

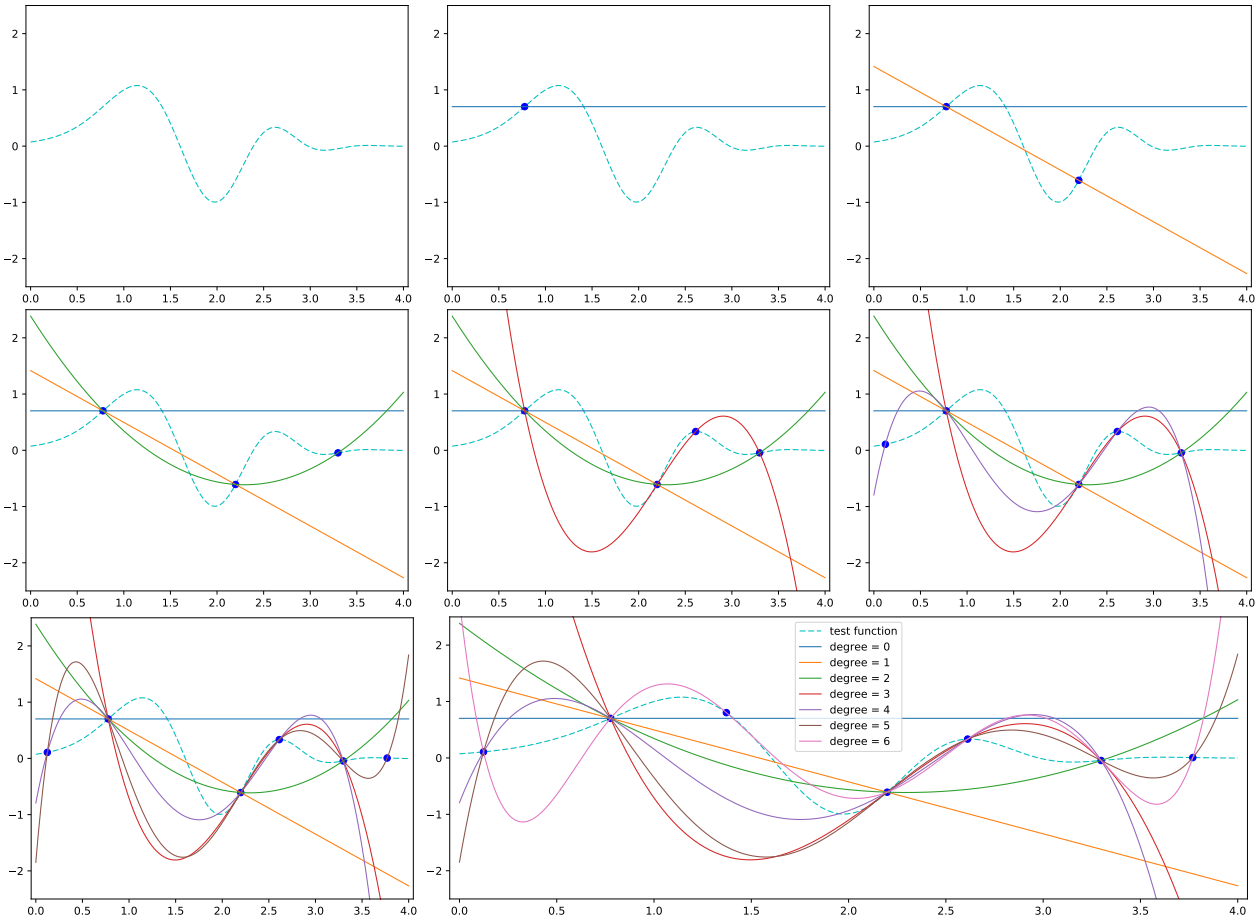
- Les données d'interpolation $(x_i, y_i = f(x_i))$, $i = 0, 1, 2, \ldots$ sont acquises dynamiquement à l'aide de la fonction `AcquisitionOnePoint()`.

- Pour chaque nouveau point acquis (x_{k+1}, y_{k+1}) on détermine et trace sur la même figure le polynôme d'interpolation $P_{k+1}(x)$ des données $(x_0, y_0), \ldots, (x_k, y_k), (x_{k+1}, y_{k+1})$, comme illustré sur la suite des figures ci-dessous.

- Pour le calcul de ce polynôme d'interpolation $P_{k+1}(x)$ sur les $k+2$ données $x_0, \ldots, x_k, x_{k+1}$, on utilisera la relation du cours

$$\begin{aligned} P_{k+1}(x) &= P_k(x) \; + \; \delta_{k+1} \; N_{k+1}(x) \\ &= P_k(x) \; + \; \delta_{k+1} \; N_k(x) \; (x - x_k) \end{aligned} \tag{2}$$

  qui permet d'obtenir ce polynôme $P_{k+1}(x)$ par mise à jour des calculs précédents (différences divisées et base de Newton) ce qui en particulier contraint à ne plus utiliser l'algorithme de Horner.



Newton interpolation of the function $f(x) = \cos(1 - x^2)\, e^{-x^2+3x-2}$ *over 7 points* x_i *dynamically acquired in the interval* $[0, 4]$

2B) Déterminer le coût (nombre de multiplications et/ou divisions) de chacune de ces 2 approches. Serait-il intéressant dans le deuxième cas (données dynamiques) d'utiliser l'algorithme de Horner pour l'évaluation, comme dans le premier cas, sur l'ensemble des données d'interpolation connues (autrement dit en *oubliant* les calculs précédemment effectués) ? *Réponse en commentaire directement dans le script Python*

Partie 3 : interpolation paramétrique

Given a sequence of points $M_i = (x_i, y_i)$, $0 \leq i \leq n$, we want to construct a polynomial parametric curve of degree n (because of the $n+1$ constraints) which interpolates these data points, that is which goes through each point (x_i, y_i) for some parameter t_i. The main question consists in the choice of the interpolation parameters t_i, which are also called the *interpolation nodes* or simply the *nodes*.

Precisely, we look for a n-degree polynomial parametric curve

$$\begin{array}{rccc} & [a,b] \in \mathbb{R} & \longrightarrow & \mathbb{R}^2 \\ m\ : & t & \longmapsto & m(t) = \begin{pmatrix} m_x(t) \\ m_y(t) \end{pmatrix} \end{array}$$

such that

$$m(t_i) = M_i \quad \Leftrightarrow \quad \begin{cases} m_x(t_i) = x_i \\ m_y(t_i) = y_i \end{cases}, \quad 0 \leq i \leq n$$

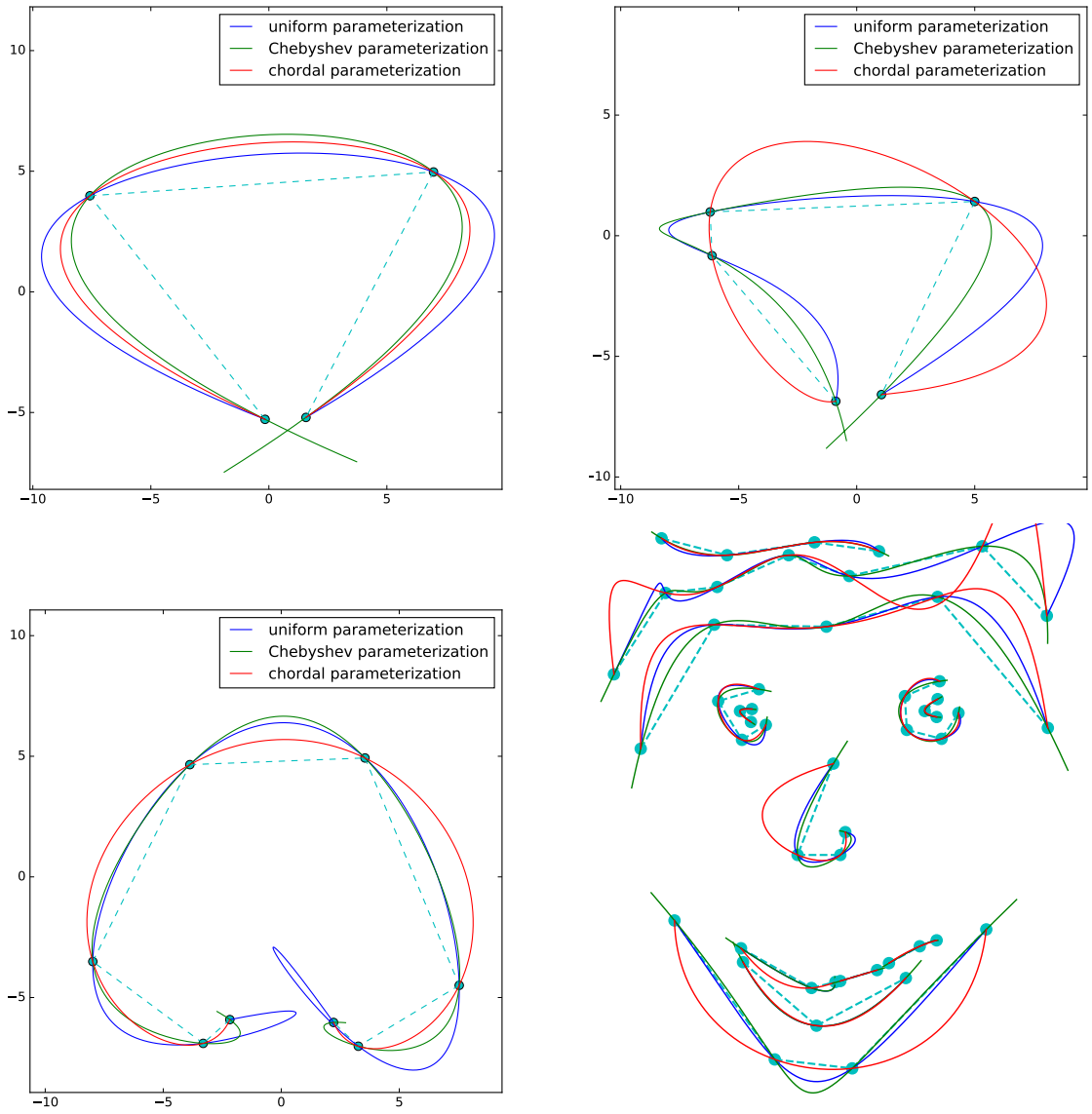
with a sequence of interpolation nodes $a \leq t_0 < t_1 < \cdots < t_n \leq b$, and where $m_x(t)$ and $m_y(t)$ are polynomials of degree n. As a result, we are reduced to solve two separate Lagrange interpolation problems.

We will consider three choices for the interpolation parameters. The first one is the *uniform parametrization* : parameters t_i are evenly spaced in the parameters domain. The second one is the *Chebyshev parametrization* as introduced in the course on interpolation. The last one is the *chordal parametrization* : parameters are chosen in such a way that distances between successive parameters t_i are proportional to the distances between associate successive data points M_i.

- Uniform parameterization : $t_i = a + i\,\dfrac{b-a}{n}$.
- Chebyshev parameterization : $\hat{t}_i = \dfrac{a+b}{2} + \dfrac{b-a}{2}\,\cos\left((2i+1)\,\dfrac{\pi}{2n+2}\right)$.
- Chordal parameterization : $\tilde{t}_{i+1} - \tilde{t}_i = \dfrac{d_i}{\sum_{k=0}^{n-1} d_k}$ with $d_k = \mathrm{dist}(M_k, M_{k+1})$ over $[0,1]$.

Note that the choice of the parameter domain $[a,b]$ does not affect the method, so that we will consider $[a,b] = [0,1]$.

The script `TP3B_NewtonParamStudents.py` allows to acquire a polygon with the mouse (right click to stop the acquisition).
Complete this program so as to interpolate vertices of this polygon according to the three choices of parameters described above (and as shown if the following figures).



Le compte rendu de ce TP consistera en les 2 scripts Python initiaux et complétés dont les noms seront `TP3B_NOM1_NOM2.py` et `TP3Bparam_NOM1_NOM2.py`.
Chaque fichier python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2, puis le script Python complété.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

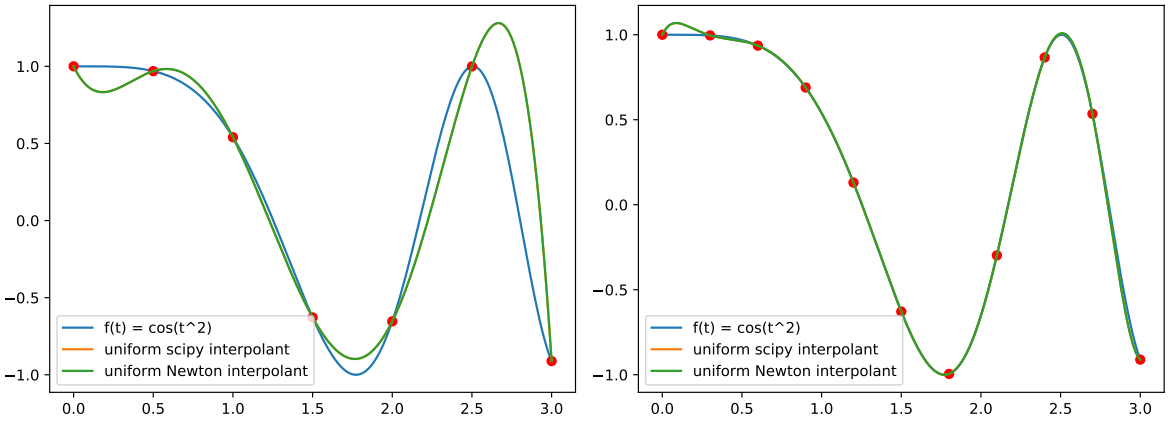
TP3-C : Lagrange interpolation in monomial basis with `scipy`

L'objectif de ce TP est de montrer les limites de l'interpolation de Lagrange dans la base des monômes. Pour cela, on utilisera la fonction `interpolate.lagrange()` de `scipy` qui détermine le polynôme d'interpolation de n données (x_i, y_i) dans la base des monômes et renvoie la suite des coefficients c_k tels que le polynôme $\sum_0^n c_k\, tk$ interpole les données (x_i, y_i).

Récupérer le script Python `TP3CscipyStudents.py` permettant de réaliser l'interpolation de Lagrange dans la base des monômes à l'aide de `scipy` et qui reprend également les TP précédents concernant l'interpolation de Lagrange dans la base de Newton à l'aide des différences divisées.

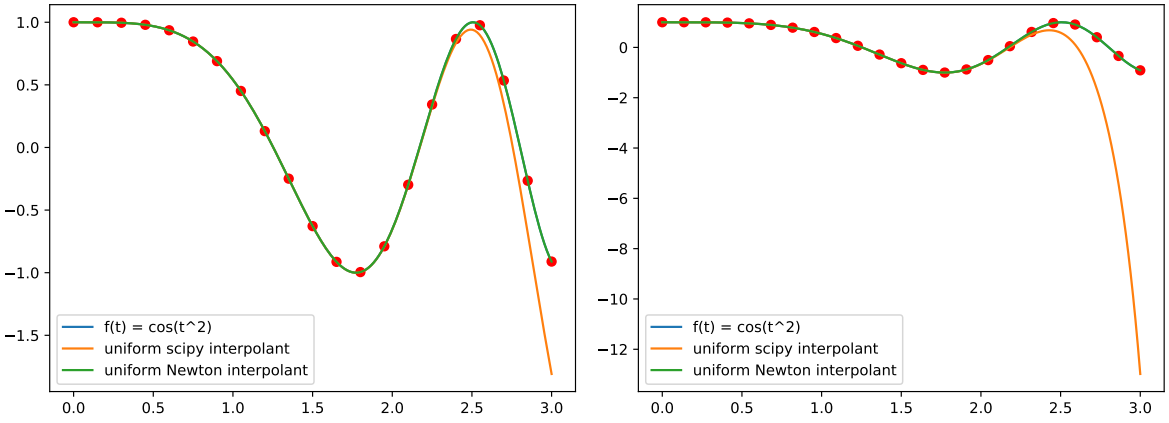
1. Compléter ce script (en particulier à l'aide des précédents TP).

2. En déduire les 2 figures suivantes obtenues respectivement pour $n = 6$ et $n = 10$ où n est le degré du polynôme d'interpolation associé à des données uniformes dans l'intervalle $[0, 3]$. Chaque polynôme d'interpolation est déterminé d'une part dans la base des monômes avec `scipy` et d'autre part dans la base de Newton.



On remarque que pour $n = 6$ et $n = 10$ les deux déterminations du polynôme d'interpolation (dans la base des monômes et dans la base de Newton) coïncident. Par ailleurs, ces polynômes semblent se "rapprocher" de la fonction initiale lorsque n augmente de 6 à 10.

3. Poursuivre cette analyse comparative et numérique pour des degrés plus importants.



Il n'est pas demandé de compte rendu écrit pour ce TP

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP4 : Hermite interpolation

Récupérer le script Python " `TP6HermiteStudents.py` " fournissant un *squelette* à compléter dans les exercices suivants.

• Given two Hermite data **of order one** $(\alpha, y_\alpha, y'_\alpha)$ and $(\beta, y_\beta, y'_\beta)$, where α and β $(\alpha < \beta)$ are two distinct points, there exits a unique (Hermite) cubic polynomial $p(x)$ interpolating these data, i.e., satisfying

$$p(\alpha) = y_\alpha\,, \quad p'(\alpha) = y'_\alpha\,, \quad p'(\beta) = y'_\beta\,, \quad p(\beta) = y_\beta.$$

This cubic Hermite interpolating polynomial $p(x)$ can be written as follows

$$p(x) = y_\alpha\, H_0\Big(\frac{x-\alpha}{\beta-\alpha}\Big) + y'_\alpha\,(\beta-\alpha)\, H_1\Big(\frac{x-\alpha}{\beta-\alpha}\Big) + y'_\beta\,(\beta-\alpha)\, H_2\Big(\frac{x-\alpha}{\beta-\alpha}\Big) + y_\beta\, H_3\Big(\frac{x-\alpha}{\beta-\alpha}\Big) \quad (1)$$

where H_0, H_1, H_2, H_3 are four cubic polynomials forming the *standard cubic Hermite basis on* $[0,1]$ and characterized by the following table.

	H_0	H_1	H_2	H_3
$H_i(0)$	1	0	0	0
$H'_i(0)$	0	1	0	0
$H'_i(1)$	0	0	1	0
$H_i(1)$	0	0	0	1

$$H_0(t) = 1 - 3\,t^2 + 2\,t^3$$
$$H_1(t) = t - 2\,t^2 + t^3$$
$$H_2(t) = -t^2 + t^3$$
$$H_3(t) = 3\,t^2 - 2\,t^3$$

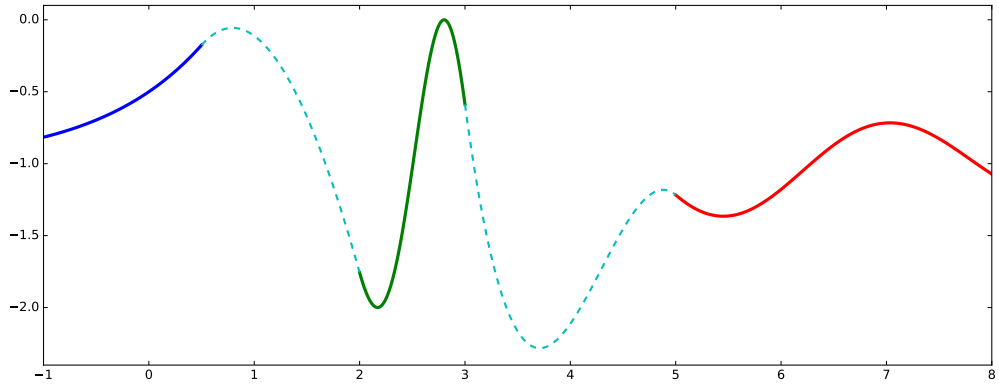
Exercise 1

Consider the three following functions which are plotted below respectively in blue, green and red.

$$f_1(x) = \frac{\exp(x)}{2} - 1, \qquad x \in [-1, 0.5],$$

$$f_2(x) = \sin(x^2) - 1, \qquad x \in [2, 3],$$

$$f_3(x) = -1 + 2\,\frac{\sin(2\,x)}{x}, \quad x \in [5, 8].$$



Determine two functions p_1 and p_2 respectively defined on intervals $[0.5, 2]$ and $[3, 5]$, and plot all these functions, such that the concatenation of the five functions f_1, p_1, f_2, p_2, f_3 provides a C^1 function over the interval $[-1, 8]$.

• Given two Hermite data **of order two** $(\alpha, y_\alpha, y'_\alpha, y''_\alpha)$ and $(\beta, y_\beta, y'_\beta, y''_\beta)$, where α and β $(\alpha < \beta)$ are two distinct points, there exits a unique (Hermite) quintic polynomial $p(x)$ interpolating these data, i.e., satisfying

$$\begin{array}{lll} p(\alpha) = y_\alpha & p'(\alpha) = y'_\alpha & p''(\alpha) = y''_\alpha \\ p(\beta) = y_\beta & p'(\beta) = y'_\beta & p''(\beta) = y''_\beta \end{array}$$

This quintic Hermite interpolating polynomial $p(x)$ can be written as follows

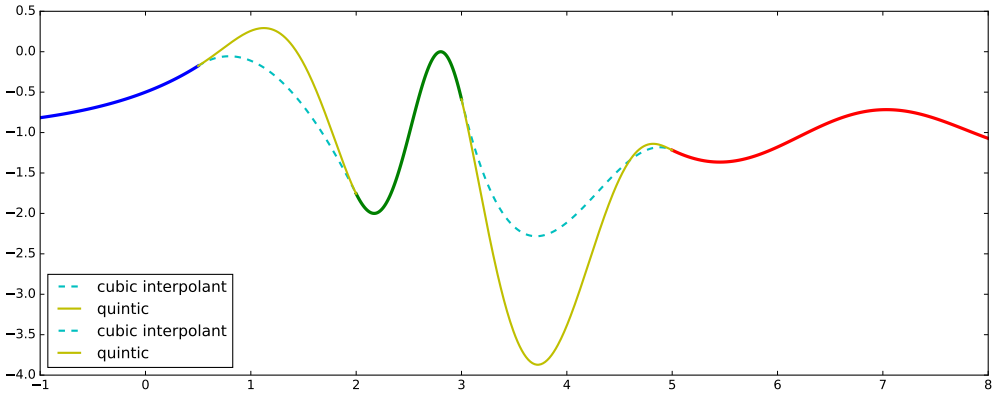
$$p(x) = \Big(Q_0(t), Q_1(t), Q_2(t), Q_3(t), Q_4(t), Q_5(t)\Big) \begin{pmatrix} y_\alpha \\ (\beta-\alpha)\, y'_\alpha \\ (\beta-\alpha)^2\, y''_\alpha \\ (\beta-\alpha)^2\, y''_\beta \\ (\beta-\alpha)\, y'_\beta \\ y_\beta \end{pmatrix} \quad \text{with} \quad t = \frac{x-\alpha}{\beta-\alpha}$$

and with the following *Quintic Hermite basis on* $[0, 1]$

$$\begin{array}{ll} Q_0(x) = -6\,x^5 + 15\,x^4 - 10\,x^3 + 1 & Q_1(x) = -3\,x^5 + 8\,x^4 - 6\,x^3 + x \\ Q_2(x) = \dfrac{1}{2}(-x^5 + 3\,x^4 - 3\,x^3 + x^2) & Q_3(x) = \dfrac{1}{2}(x^5 - 2\,x^4 + x^3) \\ Q_4(x) = -3\,x^5 + 7\,x^4 - 4\,x^3 & Q_5(x) = 6\,x^5 - 15\,x^4 + 10\,x^3 \end{array}$$

Exercise 2 (*Comparison with the cubic Hermite interpolation*)

Consider again the three functions given in exercise 1 and determine the two quintic polynomials $q_1(x)$ and $q_2(x)$ respectively defined on intervals $[0.5, 2]$ and $[3, 5]$, such that the concatenation of the five functions f_1, q_1, f_2, q_2, f_3 provides a C^2 function over the interval $[-1, 8]$.



Comparison between cubic and quintic Hermite interpolation.

Le compte rendu de ce TP consistera en un seul fichier Python dont le nom sera `TP4_NOM1_NOM2.py`
Ce script Python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2. L'exécution de ce script devra afficher la figure donnée dans l'exercice 2 ci-dessus, et donc réaliser l'interpolation `cubique` et `quintique` des données sur un même graphique.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP5A : interpolation spline cubique C2 — cas explicite & paramétrique

---

En 2D nous pouvons donner les définitions suivantes.

Interpolation explicite : il s'agit de l'interpolation de données (x_i, y_i) par une fonction explicite $x \mapsto s(x)$ [ici, une spline]. Si les données sont choisies sur le graphe d'une fonction f, et donc de la forme $(x_i, f(x_i))$, on parle alors d'interpolation de f en les points x_i. Si les points x_i sont équirépartis dans l'intervalle d'étude, on parle d'interpolation *uniforme*.

Interpolation paramétrique : il s'agit de l'interpolation de données (x_i, y_i) par une fonction paramétrique $t \mapsto (s_x(t), s_y(t))$ [ici, chaque composante est une spline]. La question essentielle est celle du choix des paramètres d'interpolation t_i tels que $(s_x(t_i), s_y(t_i)) = (x_i, y_i)$. Si les paramètres (ou noeuds) d'interpolation t_i sont équirépartis dans le domaine de paramétrage, on parle d'interpolation paramétrique *uniforme*. Une solution alternative souvent préférable est celle de la paramétrisation cordale dèjà rencontrée dans le TP3B.

Dans ce document nous considérons uniquement les *splines naturelles* et nous employons le terme *spline* pour *spline cubique C2 naturelle*.

1. Interpolation explicite

Etant donnée une fonction f définie sur un intervalle $[a, b]$, nous considérons l'interpolation de f par une fonction spline en n points distincts de l'intervalle $[a, b]$. Dans une première partie nous considérons le cas *uniforme* de n points équirépartis dans l'intervalle $[a, b]$. Nous considérons ensuite le cas général (*non uniforme*) de n points distincts quelconques dans $[a, b]$.

Télécharger le fichier `TP5AsplinesStudents.py` que vous complèterez.

1.a Cas uniforme

Soit n points x_i équirépartis dans l'intervalle $[a, b]$

$$a = x_1 < x_2 < \cdots < x_{n-1} < x_n = b,$$

$$x_i = a + (i-1)\,h, \quad i = 1, 2, \ldots, n, \qquad h = \frac{b-a}{n-1}.$$

Les données d'interpolation sont les points $(x_i, y_i = f(x_i))$, $i = 1, \ldots, n$. La spline d'interpolation est définie sur chaque intervalle $[x_i, x_{i+1}]$ par un polynôme cubique exprimé dans la base de Hermite associée à cet intervalle. Les dérivées y'_i en chaque point x_i sont déterminées de sorte à assurer la continuité C^2 entre les différents polynômes cubiques aux noeuds intérieurs x_i et la condition des splines naturelles en a et b.
Ces dérivées y'_i sont obtenues par résolution du système linéaire suivant.

$$\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ 1 & 4 & 1 & & & & 0 \\ & & & & & & \vdots \\ 0 & 1 & 4 & 1 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 1 & 4 & 1 & 0 \\ 0 & & & & 1 & 4 & 1 \\ 0 & 0 & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} y'_1 \\ y'_2 \\ y'_3 \\ \vdots \\ y'_{n-2} \\ y'_{n-1} \\ y'_n \end{pmatrix} = \frac{3}{h} \begin{pmatrix} y_2 - y_1 \\ y_3 - y_1 \\ y_4 - y_2 \\ \vdots \\ y_{n-1} - y_{n-3} \\ y_n - y_{n-2} \\ y_n - y_{n-1} \end{pmatrix}.$$

Exercise 1

1. Ecrire un script Python permettant de réaliser l'interpolation spline uniforme.
On rappelle que la construction de la matrice du système linéaire ci-dessus a déjà été réalisée lors du TP d'introduction à Python. On s'appuiera également sur le TP concernant l'interpolation de Hermite.
2. Tester cette interpolation spline pour la fonction $f(x) = \sin(x^2 - 2\,x + 1) + \big[\cos(x^3 + x)\big]^2$ *sur l'intervalle* $[a, b] = [-1, 2]$.

1.b Cas non uniforme

On considère désormais n points distincts quelconques x_i dans l'intervalle $[a, b]$

$$a = x_1 < x_2 < \cdots < x_{n-1} < x_n = b \quad \text{et} \quad h_i = x_{i+1} - x_i,\ i = 1 \ldots, n-1.$$

La méthode est similaire au cas uniforme. La spline d'interpolation des données $(x_i, y_i = f(x_i))$ est définie sur chaque intervalle $[x_i, x_{i+1}]$ par un polynôme cubique exprimé dans la base de Hermite associée à cet intervalle. Les dérivées y'_i réalisant la continuité C^2 et la condition des splines naturelles sont obtenues par résolution du système linéaire suivant.

$$\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ h_2 & 2(h_1+h_2) & h_1 & 0 & & & & 0 \\ 0 & h_3 & 2(h_2+h_3) & h_2 & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & 0 & h_i & 2(h_{i-1}+h_i) & h_{i-1} & 0 & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & 0 \\ 0 & & & & 0 & h_{n-1} & 2(h_{n-2}+h_{n-1}) & h_{n-2} \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} y'_1 \\ y'_2 \\ y'_3 \\ \vdots \\ y'_{n-2} \\ y'_{n-1} \\ y'_n \end{pmatrix}$$

$$= 3 \begin{pmatrix} \frac{1}{h_1}(y_2 - y_1) \\ \frac{h_1}{h_2}(y_3 - y_2) + \frac{h_2}{h_1}(y_2 - y_1) \\ \vdots \\ \frac{h_{i-1}}{h_i}(y_{i+1} - y_i) + \frac{h_i}{h_{i-1}}(y_i - y_{i-1}) \\ \vdots \\ \frac{h_{n-2}}{h_{n-1}}(y_n - y_{n-1}) + \frac{h_{n-1}}{h_{n-2}}(y_{n-1} - y_{n-2}) \\ \frac{1}{h_{n-1}}(y_n - y_{n-1}) \end{pmatrix}.$$

Exercise 2

1. Ecrire un script Python permettant de réaliser l'interpolation spline non uniforme.
2. Tester cette interpolation spline pour la fonction $f(x) = \sin(x^2 - 2\,x + 1) + \big[\cos(x^3 + x)\big]^2$ *sur l'intervalle* $[a, b] = [-1, 2]$. *Les* n *points d'interpolation* x_i *pourront être déterminés selon une loi de distribution uniforme, par exemple :* `xi = a + (b-a) * np.random.rand(n)`.

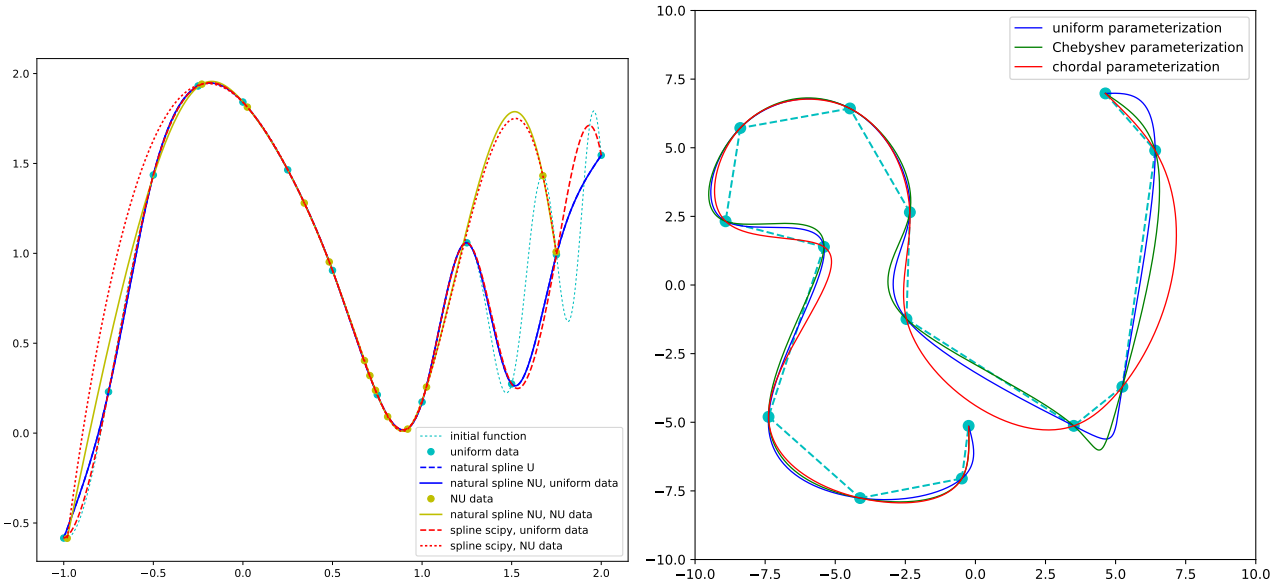
2. Interpolation paramétrique

Le contexte est similaire à celui de l'*interpolation paramétrique polynomiale* du TP3B. Etant donné un polygone de n points $M_i = (x_i, y_i)$, $i = 1 \ldots, n$, on cherche ici une courbe spline paramétrée $t \in [0,1] \mapsto m(t) = (m_x(t), m_y(t))$ interpolant les points M_i, c'est à dire telle que $m(t_i) = M_i$ pour une suite donnée de paramètres t_i dans l'intervalle $[0,1]$.

Télécharger le fichier `TP5AsplinesParamStudents.py` que vous complèterez.

Exercise 3

Ecrire un script Python réalisant cette interpolation spline paramétrique. On considèrera la paramétrisation uniforme et la paramétrisation cordale. On fera attention qu'ici la numérotation des points commence à 1 (et non pas à 0). L'acquisition des points sera à nouveau réalisée à l'aide de la fonction `PolygonAcquisition(color1,color2)`.



3. Comparaison avec l'interpolation paramétrique polynomiale

Il n'y a rien à faire ici.
Uniquement à expérimenter la dernière partie du script `TP5AsplinesParamStudents.py` qui trace les interpolants paramétriques polynomiaux dans les cas uniforme et cordal.
Pour cela, il faut importer la fonction `NewtonInterpol` à partir du script `TP3B_Newton`

Le compte rendu de ce TP consistera en un seul script Python (le cas paramétrique) dont le nom sera `TP5AsplinesParam_NOM1_NOM2.py`
Ce script Python contiendra en entête (et donc en commentaire) les noms NOM1 et NOM2 des éléments du binôme. L'exécution de ce script devra permettre d'éxécuter uniquement l'interpolation spline paramétrique (uniforme et cordale) selon les instructions de l'exercice 3.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP5B : Periodic interpolating cubic splines

Ce TP est une implémentation de l'exercice "Periodic C2 cubic splines" du cours, et que nous reproduisons ci-dessous. Cette implémentation (question 9 ci-dessous) s'appuie bien évidemment sur les questions 1 à 8 dont les réponses sont dans le polycopié du cours.

Téléchargez le fichier `TP5BperiodicSplinesStudents.py` que vous complèterez.

Etant donnée une suite de n points x_i $(n \geq 2)$ telle que

$$a = x_1 < x_2 < \cdots < x_n = b,$$

on considère l'ensemble E des fonctions de $C^2(\mathbb{R})$, périodiques, de période $T = b - a$, dont la restriction à chaque intervalle $[x_i, x_{i+1}]$ est un polynôme de degré 3.

On considère par ailleurs l'espace $\Pi_2^3 = \Pi_2^3(x_1, ..., x_n)$ des splines cubiques C^2 associées à cette famille de noeuds x_i. On rappelle que Π_2^3 est l'ensemble des fonctions de classe C^2 sur $[a, b]$ dont la restriction à chaque intervalle $[x_i, x_{i+1}]$, $i = 1, 2, ..., n-1$ est un polynôme de degré 3.

1. Vérifier que E est un espace vectoriel.

2. On considère le cas $n = 2$ avec $a = 0$ et $b = 1$. Expliciter l'espace E. Donner sa dimension et une base.

3. On se place désormais dans le cas général avec $n > 2$. Pour un élément $f \in E$, on note $\hat{f}$ sa restriction à l'intervalle $[a, b]$ et on considère l'ensemble $\hat{E} = \{\hat{f}\,,\ f \in E\}$. Vérifier que $\hat{E}$ est un sous espace vectoriel de Π_2^3.

4. Soit $s \in \Pi_2^3$. Ecrire les conditions nécessaires et suffisantes pour que $s \in \hat{E}$.

5. On rappelle que tout élément s de Π_2^3 s'écrit de manière unique sous la forme

$$s(x) = \sum_{i=0}^{3} \alpha_i\, x^i + \sum_{i=2}^{n-1} \beta_i\, (x - x_i)_+^3\,, \quad x \in [a, b]$$

   ce qui définit l'isomorphisme $F : \big(\alpha_0, \ldots, \alpha_3, \beta_2, \ldots, \beta_{n-1}\big) \in \mathbb{R}^{n+2} \longmapsto s \in \Pi_2^3$.

   a) Montrer que $\hat{E}$ est le noyau d'une application linéaire Φ de Π_2^3 dans $\mathbb{R}^3$.
   b) Ecrire la matrice de l'application linéaire $\varphi = \Phi \circ F : \mathbb{R}^{n+2} \to \mathbb{R}^3$, déterminer son rang et en déduire la dimension de $\hat{E}$.

Interpolation. – On considère maintenant une suite de n réels $y_1, y_2, ..., y_n$ et on souhaite interpoler les données (x_i, y_i), $i = 1, ..., n$ par une fonction de $\hat{E}$. Autrement dit, on cherche $\hat{f} \in \hat{E}$ tel que

$$\hat{f}(x_i) = y_i, \quad i = 1, 2, ..., n.$$

6. Les données y_i peuvent-elles toutes être choisies indépendamment ?

   On suppose désormais que les données y_i sont cohérentes au sens de la question précédente. On notera $x_{i+1} - x_i = h_i$, $i = 1, \ldots, n-1$. On propose de construire l'interpolant $\hat{f}$ selon une méthode semblable à celle des splines naturelles non uniformes développée dans l'annexe du cours.

7. Quelles sont les inconnues à déterminer.

8. Ecrire le système linéaire permettant de déterminer ces inconnues.

Implémentation. – Une courbe plane paramétrique $t \in [a, b] \mapsto m(t) \in \mathbb{R}^2$ est dite fermée périodique de classe C^2 si l'application m est de classe C^2 et si $m(a) = m(b)$, $m'(a) = m'(b)$, $m''(a) = m''(b)$.

9. Ecrire un script Python permettant de

   a) acquérir un polygone à la souris (Figure 1),

   b) fermer ce polygone en joignant le dernier point acquis au premier point (Figure 2),

   c) tracer la courbe spline paramétrée fermée périodique, cubique C^2, interpolant les sommets de ce polygone (Figure 3) pour la paramétrisation cordale.

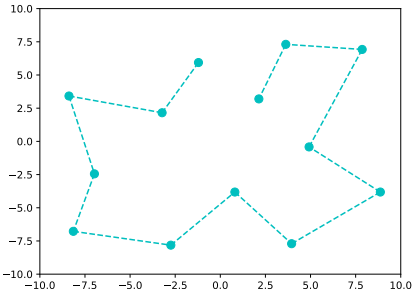


Figure 1

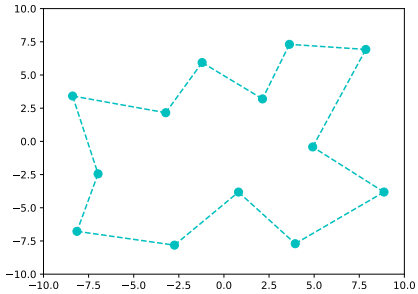


Figure 2

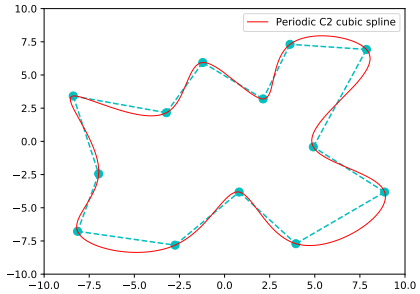
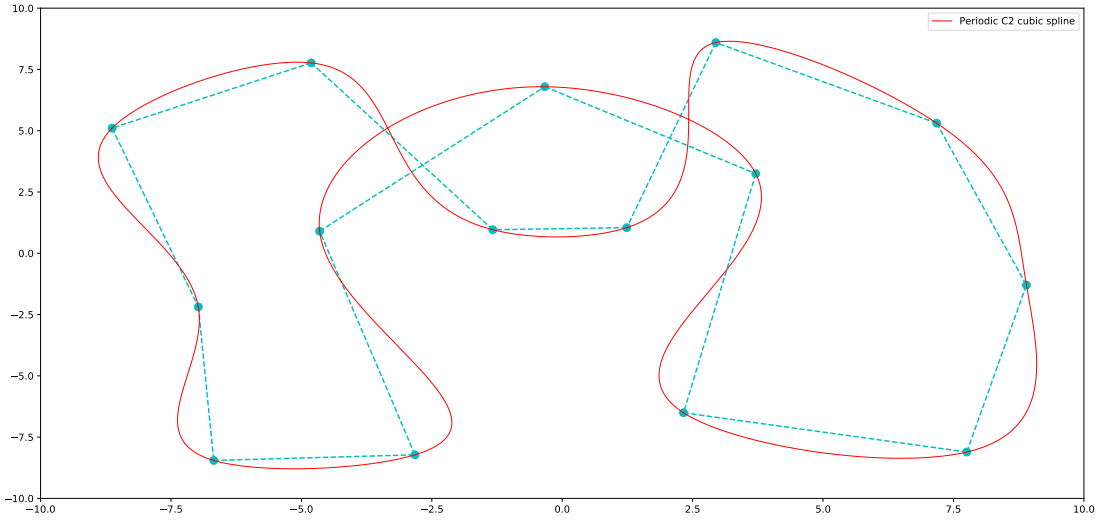


Figure 3



Le compte rendu de ce TP (question 9) consistera en un seul script Python dont le nom sera `TP5B_NOM1_NOM2.py`. L'exécution de ce script devra permettre d'éxécuter l'ensemble des 3 opérations a), b), c) définies dans la question 9 ci-dessus.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP5C : Splines exponentielles sous tension

La première partie de ce document est une étude complète (similaire à celle du cours) des splines exponentielles sous tension.
La seconde partie concerne une implémentation que vous devrez réaliser de ces splines exponentielles sous tension dans le cas le plus général : cas explicite $y = f(x)$ non-uniforme avec paramètres de tension locaux puis application au cas paramérique avec paramétrisation cordale.

Il n'est pas fourni de programme `squelette`... Néanmoins, afin de vérifier votre programme dans l'une des configurations demandées, vous pourrez utiliser les scripts `DataStudentsTS.py` et `DataTS.txt`

Partie I : étude du cas explicite

Considérons une suite de points $(x_i, y_i),\ i = 1, 2, \ldots, N,$ avec

$$a < x_1 < x_2 < \cdots < x_i < x_{i+1} < \cdots < x_{N-1} < x_N < b$$

et soit $\Omega_{2,4}$ l'ensemble des fonctions réelles de classe C^2 sur $[a, b]$ dont la restriction à chaque intervalle $[x_i, x_{i+1}]$ est de classe C^4, dont la restriction aux deux intervalles $[a, x_1]$ et $[x_N, b]$ est linéaire et interpolant les données (x_i, y_i)

$$\Omega_{2,4} = \Big\{ f \in C^2([a,b]),\quad f_i := f_{[x_i,x_{i+1}]} \in C^4([x_i, x_{i+1}]),\ i = 1, \ldots, N-1,\\ f_{[a,x_1]} \in \mathbb{R}_1[x],\ f_{[x_N,b]} \in \mathbb{R}_1[x],\\ f(x_i) = y_i,\ i = 1, \ldots, N \Big\}$$

On associe un paramètre $\sigma_i > 0$ à chaque intervalle $[x_i, x_{i+1}]$ et on considère le problème de minimization suivant.

Déterminer $\hat{f} \in \Omega_{2,4}$ tel que

$$E(\hat{f}) = \min_{f \in \Omega_{2,4}} E(f) \qquad \text{avec} \qquad E(f) = \int_a^b f''(x)^2\,\mathrm{d}x + \sum_{i=1}^{N-1} \sigma_i^2 \int_{x_i}^{x_{i+1}} f'(x)^2\,\mathrm{d}x \qquad (1)$$

Le premier terme de l'énergie $E(f)$ représente une approximation de l'énergie de flexion tandis que le second terme est une énergie de tension conduisant la courbe à être rectiligne et ainsi à minimiser sa longueur sur chaque intervalle $[x_i, x_{i+1}]$. Chaque paramètre $\sigma_i > 0$ agit donc comme un *paramètre de tension* sur l'intervalle $[x_i, x_{i+1}]$.

1. $\Omega_{2,4}$ est-il un espace vectoriel ? Si non, précisez l'espace vectoriel associé $\overrightarrow{\Omega_{2,4}}$
   Pour des valeurs quelconques des y_i, et en particulier si un seul des y_i est non nul, la fonction nulle n'est pas dans $\Omega_{2,4}$ qui n'est donc pas un espace vectoriel.

$$\overrightarrow{\Omega_{2,4}} = \Big\{ f \in C^2([a,b]),\quad f_i := f_{[x_i,x_{i+1}]} \in C^4([x_i, x_{i+1}]),\ i = 1, \ldots, N-1,\\ f_{[a,x_1]} \in \mathbb{R}_1[x],\ f_{[x_N,b]} \in \mathbb{R}_1[x] \Big\}$$

2. Soit $f \in \Omega_{2,4}$. Préciser $f''(x_1)$ et $f''(x_N)$.
$f_{[a,x_1]}$, $f_{[x_N,b]} \in \mathbb{R}_1[x] \Rightarrow f''(x_1) = f''(x_N) = 0$. Réciproquement, notons qu'une fonction $f \in C^2([x_1, x_N])$, avec $f''(x_1) = f''(x_N) = 0$, se prolonge de manière unique en une fonction linéaire sur chacun des intervalles $[a, x_1]$ et $[x_N, b]$, et de classe C^2 sur $[a, b]$.

3. Soit s la spline naturelle (cubique C^2) interpolant les données (x_i, y_i). Justifier que $s \in \Omega_{2,4}$.
Cette spline s vérifie chacune des conditions caractérisant l'ensemble $\Omega_{2,4}$. En particulier, sa restriction à chaque intervalle $[x_i, x_{i+1}]$ est cubique et donc de classe C^4, ses dérivées secondes sont nulles en x_1 et x_N, ce qui permet de la prolonger linéairement comme indiqué ci-dessus.

4. *Espace des solutions sur chaque intervalle* $[x_i, x_{i+1}]$.
Déterminer l'espace vectoriel des solutions E_i sur chaque intervalle $[x_i, x_{i+1}]$, de sorte que si S est la solution du problème (1), alors $S_i := S_{[x_i,x_{i+1}]} \in E_i$ pour $i = 1, \ldots, N-1$
Précisez la dimension de E_i.
Sur chaque intervalle $[x_i, x_{i+1}]$ le problème (1) consiste à déterminer une fonction S_i de classe C^4 qui minimise la fonctionnelle

$$E_i(f) \;=\; \int_{x_i}^{x_i} \Big(f''(x)^2 \,+\, \sigma_i^2\, f'(x)^2 \Big) \mathrm{d}x$$

La solution générale S sur $[a, b]$ sera ensuite construite par "assemblage" de ces solutions locales pour obtenir un élément de $\Omega_{2,4}$, c'est-à-dire de classe C^2 sur $[a, b]$, avec des dérivées secondes nulles en x_1 et x_N, et interpolant les données (x_i, y_i).

La fonction de $\mathbb{R}^4$ dans $\mathbb{R}$ définie par $F(x, f, f', f'') = (f'')^2 + \sigma_i^2\,(f')^2$ est de classe C^3, ce qui permet d'utiliser la caractérisation du cours donnée par les équations d'Euler Lagrange. Précisément, une condition nécessaire pour que S_i réalise le minimum de la fonctionnelle E_i est donnée par *l'équation d'Euler-Lagrange d'ordre 2* :

$$-\,\frac{\mathrm{d}}{\mathrm{d}x}\Big(2\,\sigma_i^2\, f'(x)\Big) \,+\, \frac{\mathrm{d}^2}{\mathrm{d}x^2}\Big(2\, f''(x)\Big) \;=\; -\,2\,\sigma_i^2\, f''(x) \,+\, 2\, f^{(4)}(x) \;=\; 0,$$

ce qui conduit à l'équation différentielle

$$f^{(4)} \;-\; \sigma_i^2\, f'' = 0. \tag{2}$$

Une intégration directe montre que sur chaque intervalle $[x_i, x_{i+1}]$, la solution S_i est de la forme

$$S_i(x) = \lambda_0^i \,+\, \lambda_1^i\, x \,+\, \lambda_2^i\, \exp(\sigma_i\, x) \,+\, \lambda_3^i\, \exp(-\sigma_i\, x). \tag{3}$$

Ainsi

$$E_i = \mathrm{Vect}\big\{1,\, x,\, \exp(\sigma_i\, x),\, \exp(-\sigma_i\, x)\big\}\,, \qquad i = 1, \ldots, N-1.$$

On vérifie ensuite que ces 4 fonctions sont linéairement indépendantes, de sorte que $\dim(E_i) = 4$ pour tout i. — Pour cela, on écrit $f(x) = \lambda_0 + \lambda_1\, x + \lambda_2\, \exp(\sigma_i\, x) + \lambda_3\, \exp(-\sigma_i\, x) = 0$ et $f''(x) = \lambda_2\, \sigma_i^2\, \exp(\sigma_i\, x) + \lambda_3\, \sigma_i^2\, \exp(-\sigma_i\, x) = 0$ pour tout x. Il suffit ensuite de prendre 2 valeurs distinctes de x, par exemple $x = 0$ et $x = 1$ et d'écrire $f(0) = f(1) = f''(0) = f''(1) = 0$ pour déduire que les λ_i sont tous nuls.

5. Montrer que

$$E_i = \mathrm{Vect}\big\{1 - u,\, u,\, \varphi_i(1-u),\, \varphi_i(u)\big\}\,, \qquad i = 1, \ldots, N-1,$$

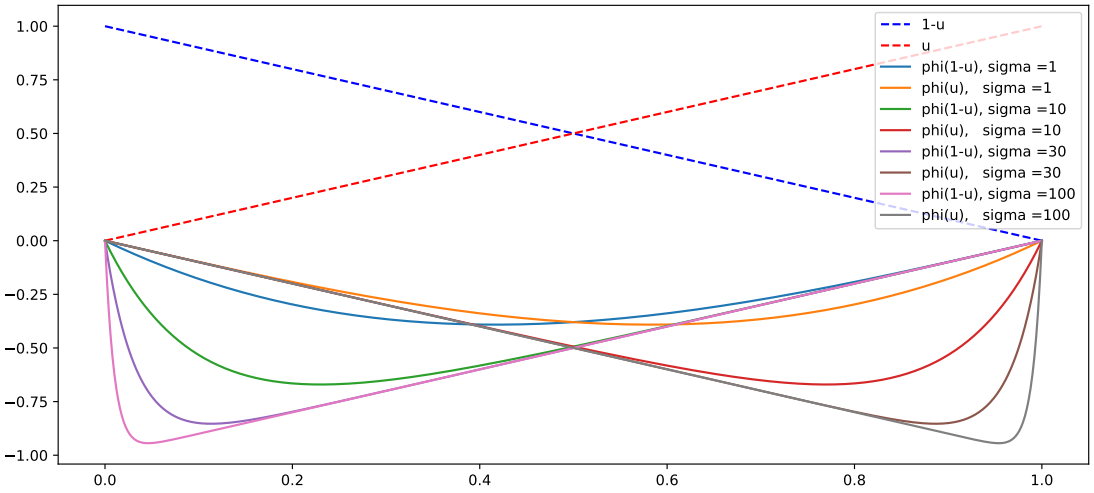
avec

$$\varphi_i(u) = \frac{\sinh(w_i\,u) - u\,\sinh(w_i)}{\sinh(w_i) - w_i},\ u = \frac{x - x_i}{h_i},\ h_i = x_{i+1} - x_i,\ w_i = \sigma_i\,h_i$$

Avec les notations données on montre que

$$\begin{aligned}
&\text{Vect}\{1-u,\ u,\ \varphi_i(1-u),\ \varphi_i(u)\}\\
&= \text{Vect}\Big\{1-u,\ u,\ \frac{\sinh(w_i\,(1-u)) - (1-u)\,\sinh(w_i)}{\sinh(w_i) - w_i},\ \frac{\sinh(w_i\,u) - u\,\sinh(w_i)}{\sinh(w_i) - w_i}\Big\}\\
&= \text{Vect}\{1-u,\ u,\ \sinh(w_i - w_i\,u),\ \sinh(w_i\,u)\}\\
&= \text{Vect}\{1-u,\ u,\ \exp(w_i - w_i\,u) - \exp(-w_i + w_i\,u),\ \exp(w_i\,u) - \exp(-w_i\,u)\}\\
&= \text{Vect}\{1-u,\ u,\ \exp(w_i)\exp(-w_i\,u) - \exp(-w_i)\exp(w_i\,u),\ \exp(w_i\,u) - \exp(-w_i\,u)\}\\
&= \text{Vect}\{1-u,\ u,\ \exp(w_i\,u),\ \exp(-w_i\,u)\}\\
&= \text{Vect}\{1 - \frac{x-x_i}{h_i},\ \frac{x-x_i}{h_i},\ \exp(\sigma_i\,h_i\,\frac{x-x_i}{h_i}),\ \exp(-\sigma_i\,h_i\,\frac{x-x_i}{h_i})\}\\
&= \text{Vect}\{1,\ x,\ \exp(\sigma_i\,x),\ \exp(-\sigma_i\,x)\}\\
&= E_i
\end{aligned}$$

6. Tracer le graphe de chacune des 4 fonctions génératices de E_i en fonction du paramètre local u, donc sur $[0, 1]$, pour différentes valeurs de σ_i.



7. Déterminer une solution S du problème (1).
On utilisera une méthode similaire à celle développée pour la construction des splines naturelles. La solution locale S_i sur chaque intervalle $[x_i, x_{i+1}]$ s'écrira

$$S_i(x) = s_i(u) = a_i\,(1-u) + b_i\,u + c_i\,\varphi_i(1-u) + d_i\,\varphi_i(u)$$

et on introduira les notations suivantes pour $i = 1, \ldots, N-1$

$$\alpha_i = \varphi_i'(1), \qquad \beta_i = \varphi_i''(1),$$

$$\Delta_i = (1+\alpha_i)\,\frac{y_{i+1} - y_i}{h_i}, \qquad m_i = (1-\alpha_i^2)\,h_i\,\beta_{i-1}, \qquad n_i = (1-\alpha_{i-1}^2)\,h_{i-1}\,\beta_i.$$

Sur chaque intevalle $[x_i, x_{i+1}]$ la solution locale S_i s'écrit

$$S_i(x) = s_i(u) = a_i\,(1-u) + b_i\,u + c_i\,\varphi_i(1-u) + d_i\,\varphi_i(u)\,, \quad a_i, b_i, c_i, d_i \in \mathbb{R}.$$

La solution générale S sur $[a,b]$ est construite par "assemblage" de ces solutions locales pour obtenir un élément de $\Omega_{2,4}$, selon une méthode similaire à celle développée pour la construction des splines naturelles : les dérivées y'_i aux noeuds x_i seront les inconnues et seront choisies de sorte à satisfaire les contraintes de contact C^2 ainsi que les conditions de dérivées secondes nulles en x_1 et x_N.
Pour la suite, remarquons que $\varphi'_i(0) = -1$ et $\varphi''_i(0) = 0$ pour tout i. Les contraintes pour la construction de $S \in \Omega_{2,4}$ sont les suivantes

(a) Interpolation et contact C^0 en chaque noeud x_i :

$$S_i(x_i) = s_i(0) = \mathbf{a_i} := \mathbf{y_i}, \quad S_i(x_{i+1}) = s_i(1) = \mathbf{b_i} := \mathbf{y_{i+1}}, \qquad i = 1, \ldots, N-1 \tag{4}$$

(b) Dérivées en chaque noeud x_i :

$$\begin{array}{rcccl} S'(x_i^+) & = & \frac{1}{h_i}\, s'_i(0) & = & \frac{1}{h_i}\big(- a_i + b_i - c_i\, \varphi'_i(1) + d_i\, \varphi'_i(0)\big) \quad := \mathbf{y'_i} \\ S'(x_{i+1}^-) & = & \frac{1}{h_i}\, s'_i(1) & = & \frac{1}{h_i}\big(- a_i + b_i - c_i\, \varphi'_i(0) + d_i\, \varphi'_i(1)\big) \quad := \mathbf{y'_{i+1}} \end{array}$$

ce qui assure le contact C^1 en en chaque noeud intérieur x_i.
On en déduit les coefficients c_i et d_i en fonction des données d'interpolation y_i et des dérivées (encore à estimer) y'_i

$$\mathbf{c_i} = \frac{-1}{1-\alpha_i^2}\Big((1+\alpha_i)(y_{i+1}-y_i) - h_i\,\big(\alpha_i\, y'_i + y'_{i+1}\big)\Big) \tag{5}$$

$$\mathbf{d_i} = \frac{1}{1-\alpha_i^2}\Big((1+\alpha_i)(y_{i+1}-y_i) - h_i\,\big(y'_i + \alpha_i\, y'_{i+1}\big)\Big) \tag{6}$$

(c) Contact C^2 en chaque noeud intérieur x_i, $i = 2, \ldots, N-1$:

$$\begin{aligned} S''(x_i^-) = S''(x_i^+) \Leftrightarrow\ & \frac{1}{h_{i-1}^2}\, s''_{i-1}(1) = \frac{1}{h_i^2}\, s''_i(0) \\ \Leftrightarrow\ & \frac{1}{h_{i-1}^2}\Big(c_{i-1}\, \varphi''_{i-1}(0) + d_{i-1}\, \varphi''_{i-1}(1)\Big) = \frac{1}{h_i^2}\Big(c_i\, \varphi''_i(1) + d_i\, \varphi''_i(0)\Big) \\ \Leftrightarrow\ & h_i^2\, \beta_{i-1}\, d_{i-1} = h_{i-1}^2\, \beta_i\, c_i \qquad\qquad \text{car}\ \ \varphi''_i(0) = 0\ \ \text{pour tout i} \\ \Leftrightarrow\ & (1-\alpha_i^2)\, h_i^2\, \beta_{i-1}\Big((1+\alpha_{i-1})(y_i - y_{i-1}) - h_{i-1}\,\big(y'_{i-1} + \alpha_{i-1}\, y'_i\big)\Big) \\ & = -(1-\alpha_{i-1}^2)\, h_{i-1}^2\, \beta_i\Big((1+\alpha_i)(y_{i+1}-y_i) - h_i\,\big(\alpha_i\, y'_i + y'_{i+1}\big)\Big) \\ & \text{avec les relations (5) et (6)} \end{aligned}$$

Ce qui fournit la relation linéaire entre les dérivées y'_i et les données d'interpolation y_i :

$$\begin{aligned} & \underbrace{(1-\alpha_i^2)\, h_i\, \beta_{i-1}}_{m_i}\, y'_{i-1} \\ + \ & \Big(\underbrace{(1-\alpha_i^2)\, h_i\, \beta_{i-1}}_{m_i}\, \alpha_{i-1} + \underbrace{(1-\alpha_{i-1}^2)\, h_{i-1}\, \beta_i}_{n_i}\, \alpha_i\Big)\, y'_i \\ + \ & \underbrace{(1-\alpha_{i-1}^2)\, h_{i-1}\, \beta_i}_{n_i}\, y'_{i+1} \\ = \ & \underbrace{(1-\alpha_i^2)\, h_i\, \beta_{i-1}}_{m_i}\, \underbrace{(1+\alpha_{i-1})\, \frac{y_i - y_{i-1}}{h_{i-1}}}_{\Delta_{i-1}} + \underbrace{(1-\alpha_{i-1}^2)\, h_{i-1}\, \beta_i}_{n_i}\, \underbrace{(1+\alpha_i)\, \frac{y_{i+1}-y_i}{h_i}}_{\Delta_i} \end{aligned}$$

Soit finalement

$$m_i\, \mathbf{y'_{i-1}} + \big(\alpha_{i-1}\, m_i + \alpha_i\, n_i\big)\, \mathbf{y'_i} + n_i\, \mathbf{y'_{i+1}} = m_i\, \Delta_{i-1} + n_i\, \Delta_i\,, \quad i = 2, \ldots, N-1 \tag{7}$$

(d) Dérivée seconde nulle aux extrémités :

$$
\begin{aligned}
S''(x_1) = \frac{1}{h_1^2}\, s_1''(0) = 0 \quad &\Leftrightarrow \quad \frac{1}{h_1^2}\Big(c_1\, \varphi_1''(1) + d_1\, \varphi_1''(0)\Big) = \frac{c_1\, \beta_1}{h_1^2} = 0 \\
&\Leftrightarrow \quad c_1 = 0 \\
&\Leftrightarrow \quad \alpha_1\, \mathbf{y_1'} + \mathbf{y_2'} = \underbrace{(1+\alpha_1)\, \frac{y_2 - y_1}{h_1}}_{\Delta_1} \qquad (8)
\end{aligned}
$$

$$
\begin{aligned}
S''(x_{_N}) = \frac{1}{h_{_{N-1}}^2}\, s_{_{N-1}}''(1) = 0 \quad &\Leftrightarrow \quad \frac{d_{_{N-1}}\, \beta_{_{N-1}}}{h_{_{N-1}}^2} = 0 \\
&\Leftrightarrow \quad d_{_{N-1}} = 0 \\
&\Leftrightarrow \quad \mathbf{y'_{N-1}} + \alpha_{_{N-1}}\, \mathbf{y'_N} = \underbrace{(1+\alpha_{_{N-1}})\, \frac{y_{_N} - y_{_{N-1}}}{h_{_{N-1}}}}_{\Delta_{_{N-1}}} \qquad (9)
\end{aligned}
$$

Nous obtenons finalement un système de N équations linéaires (7,8,9) dont les inconnues sont les N dérivées y_i' et dont le second membre s'exprime en fonction des données d'interpolation y_i. Nous montrons dans la question suivante que ce système linéaire admet une unique soution, ce qui prouve qu'il existe une unique solution au problème de minimization (1)

8. Expliciter le système linéaire conduisant à la solution du problème (1) et justifier l'existence et l'unicité d'une solution.
Le système linéaire décrit ci-dessus s'écrit maticiellement

$$
\begin{pmatrix}
\alpha_1 & 1 & 0 & \cdots & \cdots & 0 & 0 \\
m_2 & \alpha_1\, m_2 + \alpha_2\, n_2 & n_2 & 0 & & & 0 \\
0 & \ddots & \ddots & \ddots & & & \vdots \\
\vdots & 0 & m_i & \alpha_{i-1}\, m_i + \alpha_i\, n_i & n_i & 0 & \vdots \\
\vdots & & & \ddots & \ddots & \ddots & 0 \\
0 & & & 0 & m_{_{N-1}} & \alpha_{_{N-2}}\, m_{_{N-1}} + \alpha_{_{N-1}}\, n_{_{N-1}} & n_{_{N-1}} \\
0 & 0 & \cdots & \cdots & 0 & 1 & \alpha_{_{N-1}}
\end{pmatrix}
\begin{pmatrix} y_1' \\ y_2' \\ y_3' \\ \vdots \\ y_{_{N-2}}' \\ y_{_{N-1}}' \\ y_{_N}' \end{pmatrix}
$$

$$
= \begin{pmatrix}
\Delta_1 \\
m_2\, \Delta_1 + n_2\, \Delta_2 \\
\vdots \\
m_i\, \Delta_{i-1} + n_i\, \Delta_i \\
\vdots \\
m_{_{N-1}}\, \Delta_{_{N-2}} + n_{_{N-1}}\, \Delta_{_{N-1}} \\
\Delta_{_{N-1}}
\end{pmatrix}.
$$

Nous montrons maintenant que la matrice de ce système linéaire est à diagonale strictement dominante, ce qui assure de l'existence et de l'unicité d'une solution.
Les coefficients α_i et β_i dépendent des paramètres de tension local $w_i = \sigma_i\, h_i > 0$, précisément :

$$\alpha_i(w_i) = \varphi_i'(1) = \frac{w_i\, \cosh(w_i) - \sinh(w_i)}{\sinh(w_i) - w_i} \qquad \text{et} \qquad \beta_i(w_i) = \varphi_i''(1) = \frac{w_i^2\, \sinh(w_i)}{\sinh(w_i) - w_i} \; > \; 0$$

Nous procédons en 2 étapes.

(a) Supposons que $\alpha_i(w_i) > 1$ pour tout $w_i > 0$ et tout i, de sorte que $\alpha_i^2 > 1$ et donc

$$m_i = (1 - \alpha_i^2)\, \underbrace{h_i\, \beta_{i-1}}_{>0} \; < \; 0 \qquad \text{et} \qquad n_i = (1 - \alpha_{i-1}^2)\, \underbrace{h_{i-1}\, \beta_i}_{>0} \; < \; 0$$

Ainsi,

$$|\alpha_{i-1}\, m_i + \alpha_i\, n_i| = -\alpha_{i-1}\, m_i - \alpha_i\, n_i = \alpha_{i-1}\, |m_i| + \alpha_i\, |n_i| > |m_i| + |n_i|$$

pour tout i, ce qui montre que la matrice du système est à diagonale strictement dominante.

(b) Reste donc à vérifier que

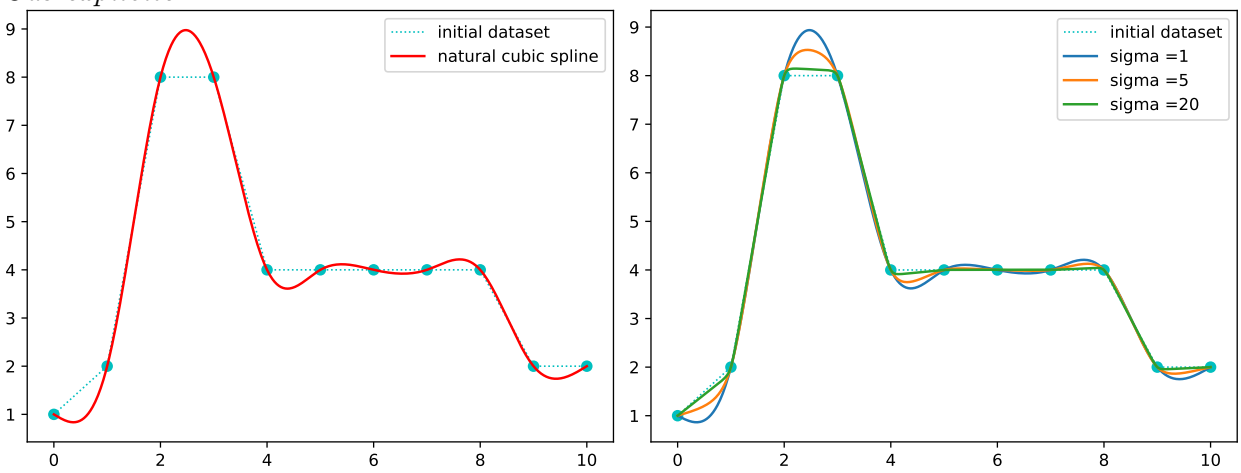
$$\alpha(t) = \frac{t\, \cosh(t) - \sinh(t)}{\sinh(t) - t} \; > \; 1, \qquad \forall\, t > 0$$

On montre en fait que la fonction $\alpha(t) - 2 = \dfrac{t\,(2 + \cosh(t)) - 3\sinh(t)}{\sinh(t) - t} = \dfrac{A(t)}{B(t)}$ est positive sur $]0, +\infty[$.
On a déjà $B(t) > 0$ sur $]0, +\infty[$ et on vérifie que $A(t)$ est positive par une étude de fonction (en allant jusqu'à la dérivéee troisième...)

Partie II : implémentation des cas explicite et paramétrique

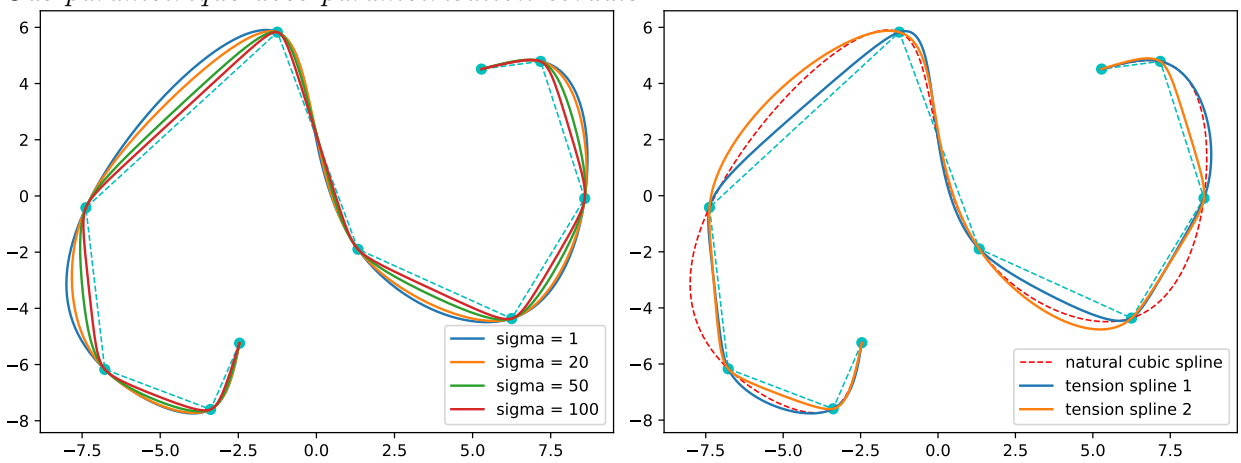
Implémenter la méthode décrite dans la première partie et considérer en particulier les données suivantes.

- *Cas explicite*



Gauche : spline naturelle – Droite : paramètre de tension global : σ_i = sigma pour tout i

- *Cas paramétrique avec paramétrisation cordale*



Gauche : paramètre de tension global σ respectivement égal à 1, 20, 50, 100
Droite : σ_i = 1,1,100,50,1,50,100,1,1 puis σ_i = 50,100,100,1,1,1,100,100,50

Le compte rendu de ce TP consistera en un seul script Python dont le nom sera `TP5C_NOM1_NOM2.py`. L'exécution de ce script devra permettre d'éxécuter l'ensemble des figures ci-dessus (excepté la première figure : spline naturelle explicite) avec les mêmes paramètres (précisés en légende ou commentaire).

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP6A : Approximation aux moindres carrés – Erreur résiduelle

Ce TP est consacré au problème de l'approximation. Précisément, étant donnée une suite de points $(x_i, y_i)_{i=1,\dots,n}$ du plan, nous nous intéressons ici à la construction d'une courbe polynomiale de degré p qui passe à *proximité* de ces points (le nombre de contraintes est supposé strictement supérieur au nombre de degré de liberté, *typiquement, une droite devant passer par n points non alignés avec $n \geq 3$*), ce qui signifie donc que $n > p + 1$.
La courbe ne pouvant passer par tous les points, l'approximation aux moindres carrés consiste à répartir l'erreur sur l'ensemble des points.

Récupérer le script Python `TP6ALeastSquaresStudents.py` que vous complèterez.

Droite de régression linéaire

Considérons une suite de n points (x_i, y_i) du plan à approcher par une droite $Y = a\,X + b$. L'approche est ici matricielle, de type *moindres carrés*, et permet de considérer des situations plus générales que celle d'une droite (ou même d'un polynôme) passant à proximité de points.

Idéalement, la droite devrait passer par l'ensemble des points (x_i, y_i) qui devraient donc satisfaire le système linéaire suivant :

$$\left\{\begin{array}{rcl} a\,x_1 + b & = & y_1 \\ a\,x_2 + b & = & y_2 \\ & \vdots & \\ a\,x_n + b & = & y_n \end{array}\right. \quad \Leftrightarrow \quad \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{noté} \quad A\,X = B.$$

Ce système linéaire $A\,X = B$ n'admettant généralement pas de solution, on cherche un vecteur $\hat{X}$ qui minimise la *distance* du vecteur $A\,X$ au vecteur B, autrement dit qui minimise la quantité $||A\,X - B||$, ce qui revient à minimiser la quantité $||A\,X - B||^2 = \sum_{i=1}^{n} \big((a\,x_i + b) - y_i\big)^2$.
On rappelle le résultat suivant du cours. *Etant donné un système linéaire $A\,X = B$, où le nombre de lignes de la matrice A est strictement supérieur au nombre de colonnes, le vecteur $\hat{X}$ qui minimise la quantité $||A\,X - B||^2$ est solution du système linéaire carré suivant*

$$\big(A^T A\big)\,X = A^T\,B, \tag{1}$$

où A^T est la matrice transposée de A. Si les colonnes de la matrice A sont linéairement indépendantes, la solution $\hat{X}$ est unique.

Exercise 1

Compléter le script Python `TP6ALeastSquaresStudents.py` *afin de déterminer (par la méthode des moindres carrés) la droite de régression linéaire associée à une suite de points acquise à la souris. Déterminer (à l'aide de la formule donnée dans le cours) le coefficient de corrélation de Pearson associé à ces données. Expérimenter ce programme.*

Approximation polynomiale aux moindres carrés

On souhaite maintenant approcher une suite de n points (x_i, y_i) du plan par une fonction polynomiale $Y = F(X) = a_0 + a_1\,X + a_2\,X^2 + \cdots + a_p\,X^p$ avec $n > p + 1$. L'approche est

identique. Idéalement, l'ensemble des n données (x_i, y_i) devrait satisfaire le système linéaire

$$\left\{\begin{array}{rcl} a_0 + a_1\,x_1 + a_2\,x_1^2 + \cdots + a_p\,x_1^p &=& y_1 \\ a_0 + a_1\,x_2 + a_2\,x_2^2 + \cdots + a_p\,x_2^p &=& y_2 \\ & \vdots & \\ a_0 + a_1\,x_n + a_2\,x_n^2 + \cdots + a_p\,x_n^p &=& y_n \end{array}\right. \Leftrightarrow \left(\begin{array}{ccccc} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{array}\right)\left(\begin{array}{c} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_p \end{array}\right) = \left(\begin{array}{c} y_1 \\ y_2 \\ \vdots \\ y_n \end{array}\right).$$

Ce système linéaire $A\,X = B$ n'admettant généralement pas de solution, on cherchera encore à déterminer une solution aux moindres carrrés.

Exercise 2

Modifier le programme de l'exercice précédent afin de déterminer le polynôme de degré p approchant au mieux, au sens des moindres carrés, une suite de n points (x_i, y_i) acquis à la souris ($n > p+1$). Expérimenter ce programme.

Erreur résiduelle

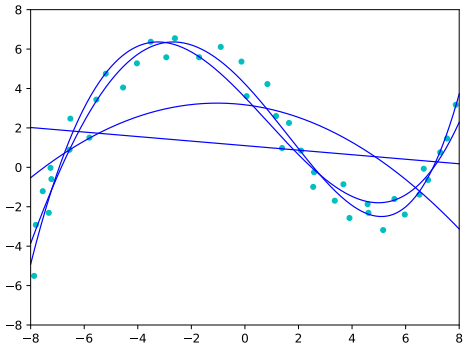
La méthode des moindres carrés consiste à minimiser la somme des erreurs au carrés définie par $||A\,X - B||^2 = \sum_{i=1}^{n}\big(F(x_i) - y_i\big)^2 = \sum_{i=1}^{n}\epsilon_i^2$, où ϵ_i est l'erreur entre la donnée y_i et le modèle $F(x_i)$. Lorsque les paramètres optimaux ont été déterminés, il reste en général une erreur appelée *erreur résiduelle* $E_{res} = \sum_{i=1}^{n}\epsilon_i^2$ qui permet de mesurer l'écart entre les données et le modèle. Afin d'analyser la pertinence d'un modèle, on considère ici *l'erreur résiduelle moyenne* $\left(\sum_{i=1}^{n}\epsilon_i^2\right)/n$ dont la racine carrée est l'*écart type*.

Exercise 3

Modifiez et complétez le script Python de l'exercice précédent afin de faire varier p entre 1 et une valeur p_{max} raisonnable. Vérifier que l'écart type diminue lorsque le degré du modèle polynomial augmente. Cela vous semble-t'il naturel ? Que se passe-t'il si le nombre n de points est égal à $p_{max}+1$? Pouvez-vous proposer une méthode permettant de choisir un modèle pertinent (c-à-d, un degré) associé à une suite de points donnée.

```
degree  1 Residual standard deviation = 3.040348677
degree  2 Residual standard deviation = 2.57235040843
degree  3 Residual standard deviation = 0.934328391479
degree  4 Residual standard deviation = 0.76968149107
```

Dans cet exemple, les points sont entrés à la souris, puis on détermine selon le critère des moindres carrés, le meilleur polynôme d'approximation de degré 1,2,3,4,... (figure à droite), et on affiche l'écart type (ci-dessus).



Le compte rendu de ce TP consistera en le script python initial complété et modifié et dont le nom sera `TP6A_NOM1_NOM2.py`
Ce fichier contiendra en entête les noms NOM1 et NOM2, puis les réponses aux questions de l'exercice 3 ci-dessus en commentaire.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP7 : Least squares approximation with integral constraint

This labwork refers to section 4 of Chapter on "*Approximation under equality constraints*" Considering a strictly increasing sequence of n points

$$\alpha = t_1 < t_2 < \cdots < t_i < \cdots < t_n = \beta,$$

and a given function $f \in C^0[\alpha, \beta]$, as well as a family of p linearly independant functions $v_j \in C^0[\alpha, \beta]$, $j = 1, 2, \ldots, p$, we consider the following problem.

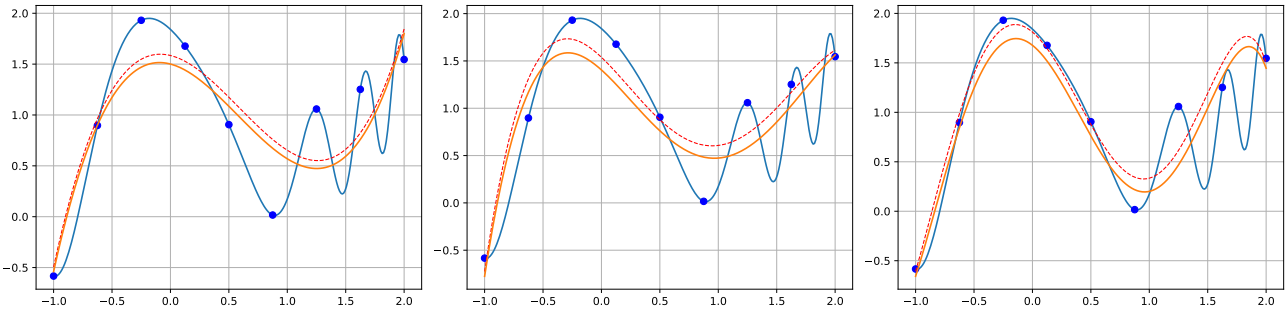
Find a function $x(t) = \sum_{j=1}^{p} x_j\, v_j(t)$ which minimizes $\sum_{i=1}^{n} \Big[x(t_i) - f(t_i)\Big]^2$ subject to the integral constraint $\int_{\alpha}^{\beta} x(t)\, dt = b$, where b is a prescribed value.

Part 1 : Implement the method of the course in the case of a function f sampled at n points `ti = np.linspace(a,b,n)` with the two family of approximating functions $v_j(t)$:

1. $\{v_j\} = \{1, t, t^2, \ldots, t^p\}$ $(p \leq 5)$, which leads to polynomial least squares approximation under constraint,
2. $\{v_j\} = \{1, \cos(2\,t), \sin(2\,t), \cos(3\,t), \sin(3\,t)\}$ which leads to trigonometric least squares approximation under constraint.

in the following 3 situations.

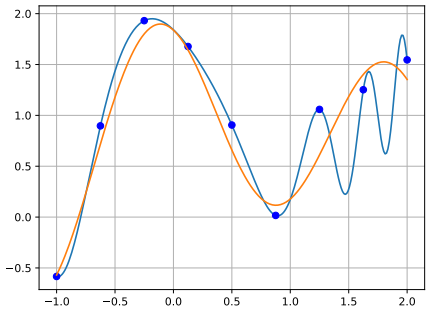
1.a : Polynomial approximation of the function $f(t) = \sin(t^2 - 2\,t + 1) + \cos^2(t + t^3)$, $t \in [-1, 2]$ (the blue curve) at 9 evenly spaced points.



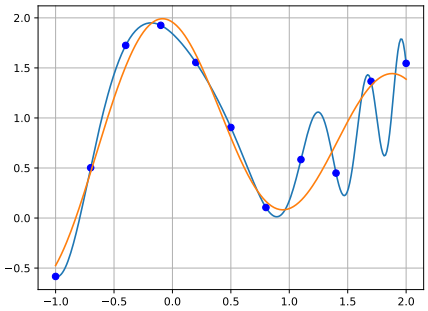
degree 3 *degree 4* *degree 5*

Dotted curves : least squares approximation by polynomials of degree 3, 4, 5 without constraint. Solid curves : least squares approximation by polynomials of degree 3, 4 and 5 subject to satisfy the integral of the initial function : $\int_{-1}^{2} v(t)dt = \int_{-1}^{2} f(t)dt$

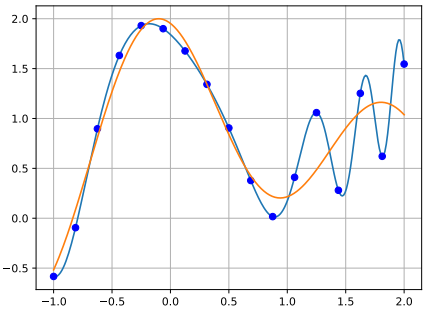
1.b : Trigonometric approximation of the function $f(t) = \sin(t^2 - 2\,t + 1) + \cos^2(t + t^3)$, $t \in [-1, 2]$ (the blue curve) at 9, 11 and 17 evenly spaced points, subject to satisfy the integral of the initial function : $\int_{-1}^{2} v(t)dt = \int_{-1}^{2} f(t)dt$



9 points



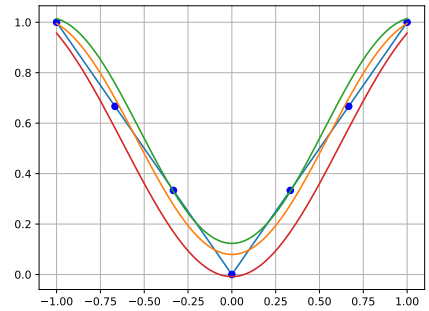
11 points



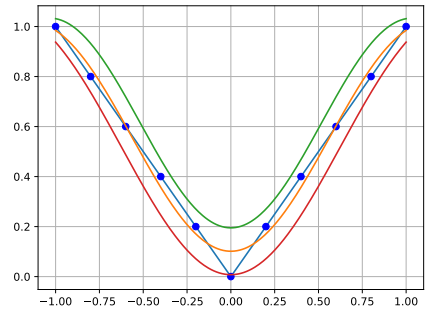
17 points

1.c : Trigonometric approximation of the function $f(t) = |t|$ (the blue curve) at 7, 11 and 17 evenly spaced points such that

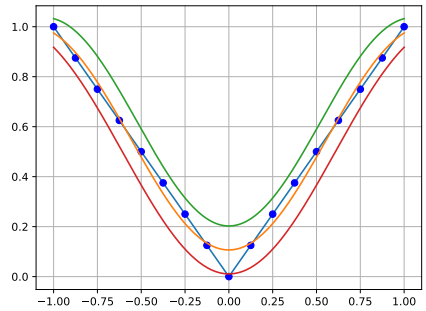
1) $\int_{-1}^{1} v(t)dt = \int_{-1}^{1} f(t)dt = 1$ (orange curves)
2) $\int_{-1}^{1} v(t)dt = 1.2$ (green curves)
3) $\int_{-1}^{1} v(t)dt = 0.8$ (red curves)



7 points



11 points



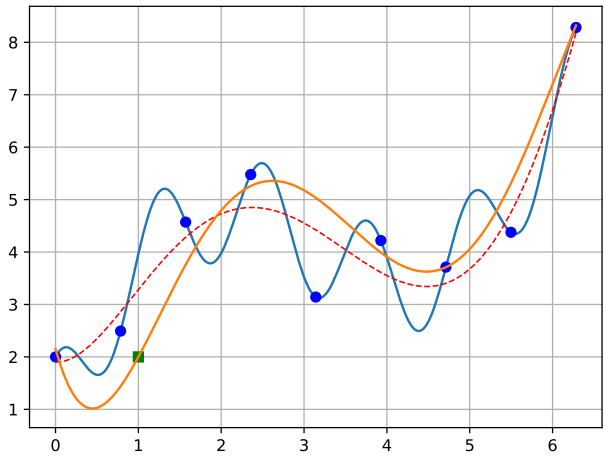
17 points

Part 2 : Update the first part of this labwork so as to consider a second equality constraint : an interpolation constraint.

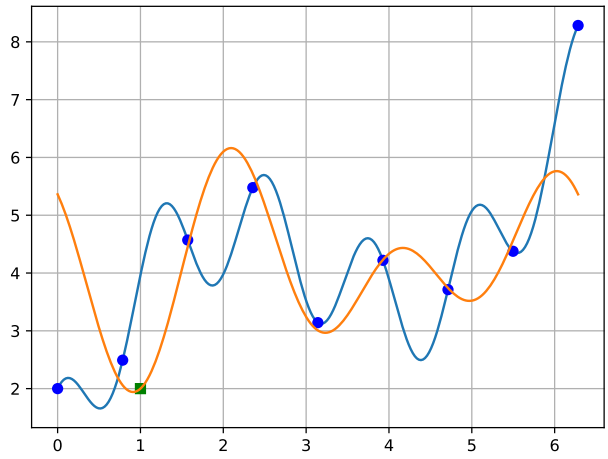
Least squares approximation of the function $f(t) = 1 + t + 2\sin(t) + \cos(5\,t)$, $t \in [0, 2\pi]$ (the blue curve) at 9 evenly spaced points by a polynomial or a trigonometric function $v(t)$ (the orange curve) under two linear constraints :

`C1` : the integral constraint $\int_0^{2\pi} v(t)dt = \int_0^{2\pi} f(t)dt$

`C2` : the interpolation constraint $v(1) = 2$



Polynomial approximation



Trigonometric approximation

— *Left : Dotted curves : least squares approximation by a polynomial of degree 5 without any constraint. Solid curve : least squares approximation at 9 points by a polynomial $v(t)$ of degree 5, subject to satisfy the 2 linear constraints* `C1` *and* `C2`

— *Right : trigonometric least squares approximation at 9 points by a function $v(t)$ of the space $\{1, \cos(2\,t), \sin(2\,t), \cos(3\,t), \sin(3\,t)\}$, subject to satisfy the 2 linear constraints* `C1` *and* `C2`

Load the file `TP7-LSAwithConstraintsPart2Students.py` that you will complete. Of course, this script file (skeleton of the second part) can be useful for the first part of this labwork.

Le compte rendu de ce TP consistera en un script Python commenté dont le nom sera `TP7_NOM1_NOM2.py` — L'éxécution de ce script devra permettre de reproduire directement les figures ci-dessus avec les mêmes données numériques.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP8 : Smoothing splines

A smoothing spline allows to satisfy a compromise between the fidelity to noisy observations and the smoothness of a fitting spline. Precisely, given a set of data points (u_k, z_k), $k = 1, \ldots, N$, with $u_1 < u_2 < \cdots < u_N$, where the observations z_k are assumed to be noisy, we consider a sequence of spline knots

$$x_1 < x_2 < \cdots < x_n$$

such that $\{u_k\}_{1 \leq k \leq N} \subset [x_1, x_n]$, and the space $S[x_1, x_n]$ of the natural splines associated with these knots. We then consider the optimization problem

$$\min_{s \in S[x_1, x_n]} E_{0,2}(s) \tag{1}$$

with

$$E_{0,2}(s) = \sum_{k=1}^{N} \Big(z_k - s(u_k)\Big)^2 + \rho \int_{x_1}^{x_n} \Big[s''(t)\Big]^2 dt \tag{2}$$

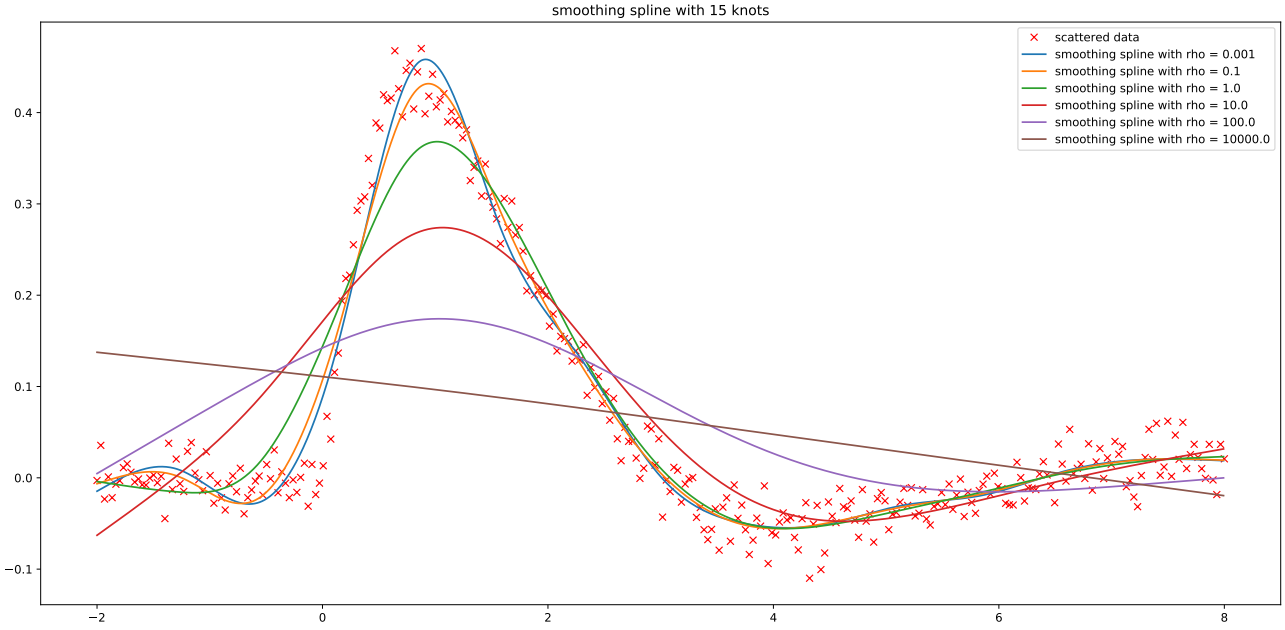
where $\rho \in \mathbb{R}^+$ is a smoothing parameter that controls the tradeoff between data fidelity and smoothness of the function. With reasonable assumptions, this minimization problem admits a unique solution $\hat{s} \in S[x_1, x_n]$ which is called a *smoothing spline*.

Exercise 1

Récupérer le fichier `data.txt` *contenant les données* (u_k, z_k) *de la figure ci-dessous. Ces données seront ensuite récupérées avec la commande*

```
(uk, zk) = np.loadtxt('data.txt')
```

Implémenter la méthode du cours (cas général) permettant de déterminer la spline de lissage pour différentes valeurs du paramètre ρ*. On considérera le cas des splines uniformes.*



Le compte rendu de ce TP consistera en un script Python dont le nom sera `TP8_NOM1_NOM2.py` dont l'éxécution permettra de reproduire la figure ci-dessus avec les mêmes données numériques.

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP9A : Subdivision

This labwork refers to Chapter on "*Bézier Curves*" and more specifically to the De-Casteljau algorithm and the *subdivision process*.
Recover the Python script `TP9ASubdivisionStudents.py` that you will complete so as to get the following figures.

1. Mouse acquisition of a control polygon (Figure 1) and display of the associated Bézier curve thanks to the Bernstein parameterization (Figure 2) : these 2 steps are given in the script.

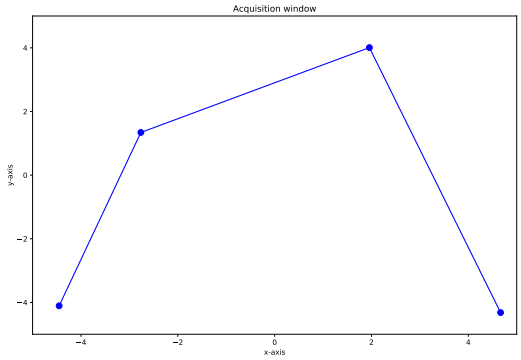


Figure 1

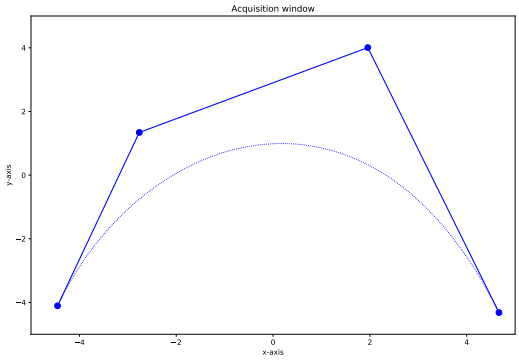
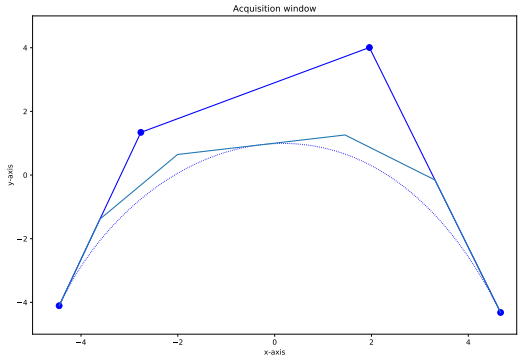
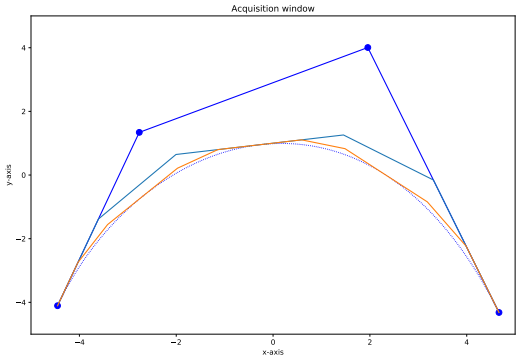


Figure 2

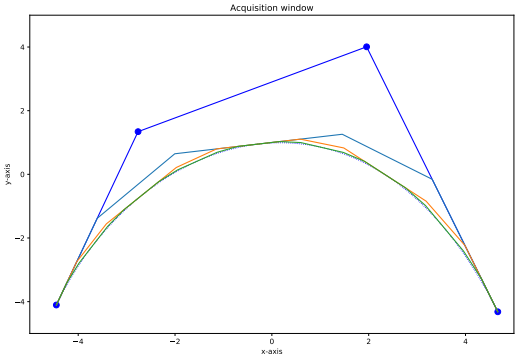
2. N subdivision steps : you need to complete the script `TP9ASubdivisionStudents.py`



subdivision : step 1



subdivision : step 2



subdivision : step 3

Le compte rendu de ce TP consistera en un script Python dont le nom sera `TP9A_NOM1_NOM2.py`
L'éxécution de ce script devra permettre de reproduire directement les figures ci-dessus .

Université Grenoble Alpes
Algèbre pour le graphique et la CAO – L3 MI

TP9B : Intersection de courbes de Bézier par subdivision et exclusion

---

1. Exclusion principle

The exclusion principle is similar to the bisection method and is simply expressed in our context as follows.

Let A, $\bar{A}$, B, $\bar{B}$ four subsets of the affine plane $\mathbb{R}^2$ such that $A \subset \bar{A}$ and $B \subset \bar{B}$. Then :

$$\bar{A} \cap \bar{B} = \emptyset \quad \Rightarrow \quad A \cap B = \emptyset.$$

2. Polynomial 2D curves intersection

Consider now the problem of the intersection of two Bézier 2D-curves respectively of degree n and m.

1. Eeach curve is included in the convex hull of its control polygon. Thus, if these two convex hulls do not intersect, we conclude by the exclusion principle that the two curves do not intersect.

2. This approach is specifically interesting when associated with subdivision principle of Bézier curves. Precisely, in case the two convex hull intersect, the subdivision of each Bézier curve (for instance, according to parameter $\alpha = 1/2$) leads to test the intersection of convex hulls smaller and smaller (which are also closer and closer to the curve).

3. The intersection of two convex polygons (the convex hulls) is complex. It is much simpler to determine the smallest rectangle parallel to the axes that contains each convex hull, and then to apply again the exclusion principle and to intersect these rectangles.

3. Implementation

We consider the intersection of the two plane Bézier curves

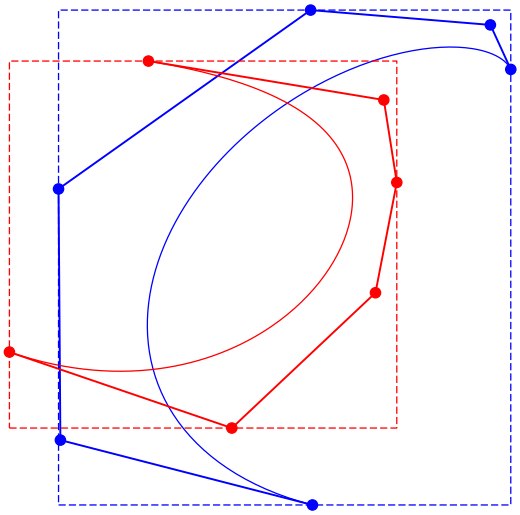
$$P(u) = \sum_{i=0}^{n} P_i \, B_i^n(u) \,, \quad u \in [0,1] \qquad \text{and} \qquad Q(v) = \sum_{i=0}^{m} Q_i \, B_i^m(v) \,, \quad v \in [0,1]$$

1. The first step consists in the mouse acquisition of two Bézier curves of different degrees.

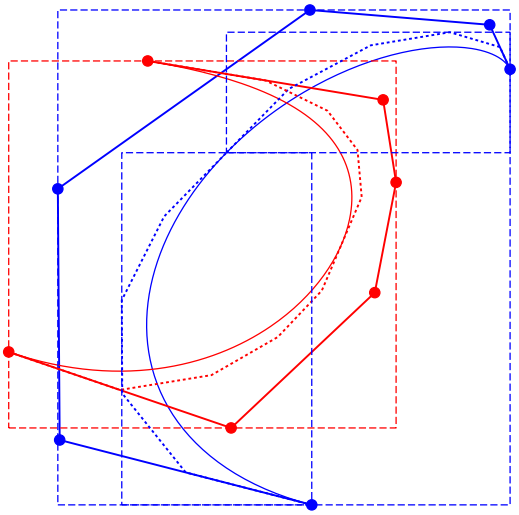
2. It is recommended to implement this algorithm in recursive form.

3. Your algorithm must :
   a) either conclude that the two curves do not intersect,
   b) or in case of intersection, for each intersection point :

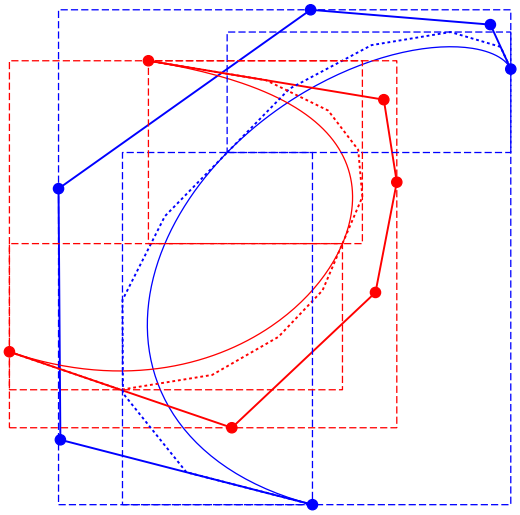
   - display an approximation of the intersection point in the 2D plane as a green point,
   - determine an approximation of the intersection parameters u_i and v_i such that $P(u_i) \approx Q(v_i)$ (plot these 2 points as blue and red points),
   - improve the accuracy of this intersection thanks to the Newton-Raphson method (plot these 2 new points as blue and red crosses).



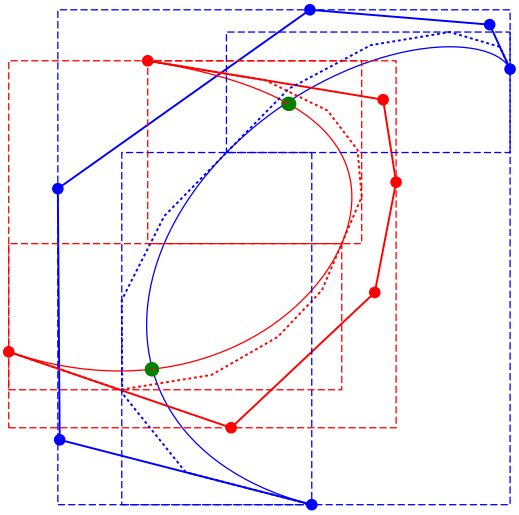
Step 1 : rectangular bounding boxes associated with each Bézier curve.



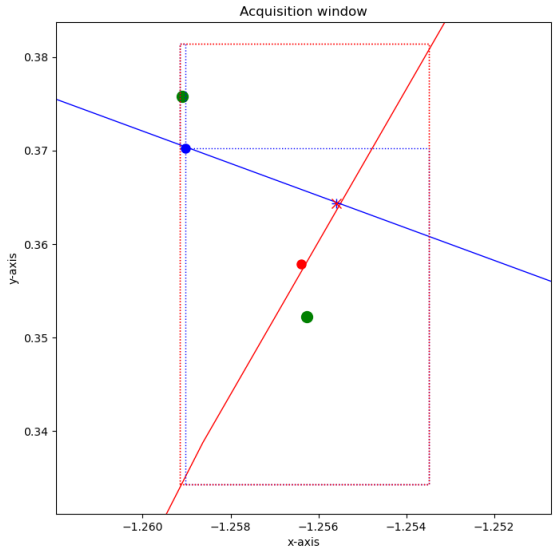
Step 2a : subdivision of the first (blue) curve and the two associated rectangular bounding boxes.



Step 2b : subdivision of the second (red) curve and the two associated rectangular bounding boxes.



Final step with precision $eps = 0.1$
Two intersections are found (and plotted with a green point).



Zoom on an intersection point :
- green points are the center of the boxes at the end of a branch of the recursive function
- blue and red points are the images by functions $P(u)$ and $Q(v)$ of the estimated intersection parameters u_i and v_i
- blue and red cross correspond to the intersection points evaluated by the Newton-Raphson method

4. Newton-Raphson method

We recall the Newton-Raphson method for solving an equation $F(X) = 0$ with

$$F : \mathbb{R}^2 \to \mathbb{R}^2 \qquad X = \begin{pmatrix} u \\ v \end{pmatrix} \mapsto F(X) = \begin{pmatrix} F_1(u,v) \\ F_2(u,v) \end{pmatrix}$$

The method is based on the following iteration

$$X_{n+1} = X_n - \Big[\mathrm{Jac}\big(F\big)(X_n)\Big]^{-1} F(X_n)$$

with an initial value X_0 close to the desired solution $\hat{X}$, and the sequence $\big(X_n\big)$ is expected to converge to $\hat{X}$.
We will apply this method to the intersection problem

$$P(u) = Q(v) \quad \Leftrightarrow \quad \left\{ \begin{array}{rcl} P_1(u) - Q_1(v) & = & 0 \\ P_2(u) - Q_2(v) & = & 0 \end{array} \right.$$

Le compte rendu de ce TP consistera en un script Python dont le nom sera `TP9B_NOM1_NOM2.py`
L'éxécution de ce script devra permettre de déterminer l'intersection de 2 courbes de Bézier entrées à la souris selon l'algorithme décrit ci-dessus.
Ce script sera commenté.