TP bibliothèque pour le calcul numérique

Ce TP propose différents exercices permettant de s'initier dans un premier temps à l'utilisation de la bibliothèque de calcul matricielle Eigen en C++. Puis dans un second temps, au langage de programmation Python.

Contents

1 Eigen

Eigen est une bibliothèque de calcul OpenSource optimisée pour réaliser des opérations matricielles. Elle pourra être utilisée pour le calcul d'inversion de système linéaire, les multiplications de matrices, les inversions de matrices, les factorisations de matrices. La bibliothèque gère le cas des matrices pleines et creuses (Sparse).

1.1 Compilation d'Eigen

La démarche suivante est à suivre si vous ne disposez pas d'une installation par défaut, ou si vous souhaitez vous habituer à la compilation de programmes à partir de leurs sources.

- ▷ Téléchargez l'archive correspondant à ce TP et décompressez celle-ci. Notez qu'il existe un répertoire `eigen/` vide. Nous allons installer `Eigen` dans ce répertoire.
- ▷ Récupérez sur le site la dernière version d'Eigen.
- ▷ Décompressez l'archive et placez une ligne de commande à la racine du répertoire (la où se situe le fichier `CMakeLists.txt`).
- ▷ Créez un nouveau répertoire `build/` et compilez Eigen à partir de ce répertoire en indiquant que l'installation devra ce faire dans le répertoire associé au TP. Pour cela, on pourra suivre les commandes suivantes

```
mkdir build/
cd build/
cmake -DCMAKE_INSTALL_PREFIX=[CHEMIN_VERS_REPERTOIRE_EIGEN] ..
make -j2
make install
```

où vous remplacerez `[CHEMIN_VERS_REPERTOIRE_EIGEN]` par votre répertoire.

Notez qu'optionnellement, il est possible de recourir à l'interface graphique de paramétrage de `CMake` à l'aide de la commande

```
cmake-gui ..
```

Vous devriez obtenir dans le répertoire cible un répertoire `include/` contenant les fichiers d'en-tête, et un répertoire `share` contenant les bibliothèques compilées d'Eigen.

1.2 Fitting d'ensemble de points par un polynome

1.2.1 Théorie

Soit un ensemble de N points du plan (x_i, y_i) pour $i \in [0, N]$. Nous souhaitons approximer ces points par un polynome de la forme

$$P(x) = \sum_{k=0}^{k=d} \alpha_k \, x^k \, ,$$

où d est le degré du polynome. Un exemple de tel approximation est présenté en figure 1.

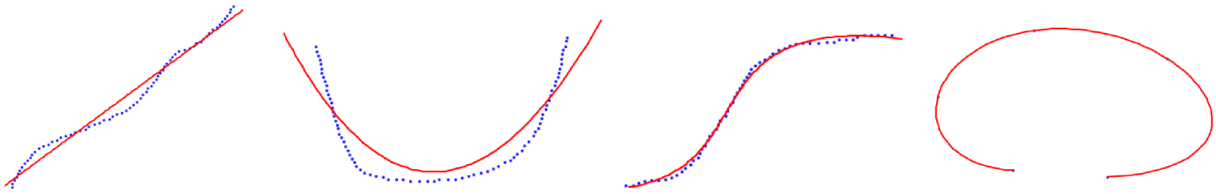

Figure 1: Exemple d'approximation (fitting) de différents ensemble de points par des polynome d'ordre 1 à 4.

Nous cherchons à approximer cet ensemble de point au sens des moindres carrés, c'est à dire que nous cherchons les coefficients α_k du polynome P qui minimise la mesure d'erreur suivante

$$\sum_{i=0}^{i=N-1} \left(P(x_i) - y_i\right)^2 \, .$$

- ▷ Lorsque $N = d + 1$, écrivez le système linéaire que doit satisfaire les coefficients a_k en fonction des x_i et y_i. Ecrivez ce système sous la forme matricielle $\mathrm{M}\boldsymbol{\alpha} = \mathbf{y}$, où $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_d)$, $\mathbf{y} = (y_0, y_1, \ldots, y_{N-1})$, et M est une matrice que l'on explicitera en fonction des valeurs x_i.

- ▷ Lorsque $N > d + 1$, écrivez les équations normales du système des moindres carrés associé. Montrez que la meilleur solution au sens des moindres carrés satisfait le système linéaire suivant

  $$\mathrm{A}\boldsymbol{\alpha} = \mathbf{b} \, ,$$

  où A est une matrice de taille $(d + 1) \times (d + 1)$ que l'on explicitera, et $\mathbf{b}$ est un vecteur de taille $d + 1$ que l'on explicitera également.

1.2.2 Algorithme

L'algorithme de fitting est donc le suivant:

```
def fitting(x,y):
  Construit matrice A
  Construit vecteur b

  Resout le systeme lineaire A alpha=b

  return polynome(alpha)
```

1.2.3 Implémentation

Le programme 2 propose différentes classes permettant de stocker un ensemble de points (`point_set`), de gérer un polynome (`polynomial`), et fait appel à une fonction `fitting` que vous devez compléter.

- ▷ Observez la fonction `main()`. Quelle erreur devrait-elle afficher dans le cas décrit si le fitting fonctionne correctement ?
- ▷ Complétez la fonction de fitting dans la fonction `fitting` en suivant les notes laissés dans les `to do`.
  Aide: La résolution d'équation linéaire par Eigen est détaillé ici.
  Note: Ne modifiez pas l'interface (le fichier `fitting.hpp` car votre fichier sera utilisé dans d'autres programmes.
- ▷ Une fois complété, vérifiez que votre fonction gère correctement le cas décrit, et testez également le cas sous-determiné nécessitant spécifiquement l'utilsation d'un moindre carrés.

1.3 Utilisation interactive Qt

Le programme 3 propose une interface où l'on peut directement dessiner un ensemble de points sur un `Widget` Qt.

- ▷ Compilez et testez ce programme.
- ▷ Observez la gestion de la souris et de l'affichage dans la classe `render_area`. Notez que vous devriez être en mesure de comprendre l'intégralité du code de ce programme.
- ▷ Copiez vos fichier `fitting.cpp` dans le programme 3 pour remplacer l'actuel. Votre programme devrait compiler et afficher le fitting de votre ensemble de points par une courbe.
- ▷ Modifiez le degré d'interpolation du polynome. Observez les instabilités liés à l'utilisation de polynome de degré élevé.
- ▷ La courbe est modélisée ici comme étant une fonction du type $y = f(x)$, et non une courbe paramétrique du type (y(t),x(t)). Quels limitations sont liées à ce choix ?

Le polynome P est défini sur $\mathbb{R}$, notez que la forme de la courbe peut varier largement à gauche du premier point, et à droite du dernier point de part l'absence de contraintes.

- ▷ Quel est l'intervalle de valeur de x pour lequel le polynome est actuellement affiché ?
- ▷ Faites en sorte de n'afficher le polynome qu'entre le point le plus à gauche et le point le plus à droite. Pour cela, on modifiera la classe `render_area` et on stockera la valeur $x_{\min}$ et $x_{\max}$ qui sera utilisé pour restreindre l'intervalle de l'affichage.

1.4 Supplément: Courbe paramétrique

Le programme 4 propose le fitting d'une courbe paramétrique $(x(t), y(t))$. Le polynome P est alors défini par

$$P(t) = \sum_{k=0}^{k=d} \alpha_k \, t^k \, ,$$

où α_k est désormais un vecteur à deux dimensions. Le polynome est donc également vectoriel est peut être décrit comme étant $P(t) = (P_x(t), P_y(t))$.

L'ensemble de points (`point_set`) contient désormais l'ensemble des positions mais également la valeur du paramètre t correspondant. Chaque point $\mathbf{p}_i(t) = (x_i(t_i), y_i(t_i))$ est donc stocké par $((x_i, y_i), t_i)$ ce qui correspond à la classe `vec2_parameterized`. Le paramètre t est automatiquement calculé en fonction de la forme de la courbe en considérant une paramétrisation curviligne. C'est à dire que pour un ensemble discret de points, on considérera

$$t_i = \|\mathbf{p}_i - \mathbf{p}_{i-1}\| .$$

▷ Compilez et executez le programme actuel.

▷ Observez la construction des coordonnées curvilignes dans la classe `point_set`. Observez également que les points qui seraient trop proches ne sont pas nécessairement ajoutés (ce qui évite certaines instabilités lors du fitting).

Le polynome P doit désormais minimiser la fonction suivante

$$\sum_{i=0}^{i=N-1} \|P(t_i) - \mathbf{p}_i\|^2 .$$

Les coordonnées x et y étant indépendantes, il est possible de trouver indépendamment $P_x(t)$ et $P_y(t)$.

▷ Quelles fonctions doivent être minimisées par $P_x(t)$ et $P_y(t)$?

▷ Ecrivez le système matricielle à résoudre, similairement au cas précédent.

▷ Implémentez votre solution dans le fichier `fitting.cpp`.

▷ Observez qu'il est désormais possible de tracer des courbes paramétriques dans le plan.

2 Python

2.1 Premier programme

▷ Créez un fichier nommé `[NOM].py`, où `[NOM]` est le nom de votre choix.

▷ Copiez le code suivant dans ce fichier à l'aide d'un éditeur de texte de votre choix (Kate, Emacs, SublimeText, etc.)

```
T=[1,4,7,6,1]
print(T)
T.append(-4)

for valeur in T:
  print(valeur)
```

▷ Lancez l'interpréteur Python sur ce fichier depuis la ligne de commande:

```
python [NOM].py
```

2.2 Affichage

Python permet l'affichage de courbes. Pour cela, il est possible d'utiliser la bibliothèque `matplotlib` ainsi que la librairie de calcul numérique `numpy`.

▷ Recopiez le code suivant et observez le résultat à l'écran.

```
import numpy as np
import matplotlib.pyplot as plt

N=100
t=np.linspace(0,2*np.pi,N)

plt.plot(np.cos(t),np.sin(t))
plt.plot(np.cos(t),np.sin(t),'r.')
plt.axis('equal')
plt.show()
```

Remarques

- `import numpy` permet d'importer un package d'une bibliothèque. Les fonctions du package de `numpy` sont disponibles à partir de l'appel `numpy.[nom_fonction]`.
- `import numpy as np` permet de donner un alias plus court pour faire appel aux fonction de numpy sous la forme `np.[nom_fonction]`.
- `plt.plot` fait appel à la fonction `plot` de la bibliothèque matplotlib. La syntaxe est proche de celle proposée par `Matlab`.
- `np.cos(t)` est un *cos* définit dans la bibliothèque matplotlib car il s'applique de manière vectorielle, terme à terme sur le vecteur t.

2.3 Lecture de fichier

▷ À l'aide d'un tableur (par exemple Calc de Libre Office, ou Excel), entrez une suite de valeurs sur deux colonnes (l'une pour les valeurs x, l'autre étant pour les valeurs de y).

▷ Sauvegardez vos données dans un fichier au format csv. Sous LibreOffice, choisissez un encodage UTF-8, un délimiteur par point espace, et un délimiteur de texte par guillemets.

▷ Observez votre fichier à l'aide d'un éditeur de texte. Notez qu'il s'agit d'un fichier ascii.

▷ Utilisez le code Python suivant sur votre fichier et assurez vous qu'il affiche bien votre graphe.

```
import os
import numpy as np
import matplotlib.pyplot as plt

x=[]
y=[]

fid=open("fichier.csv")
for line in fid:
  p=line.split()
```

```
  if len(p)==2:
    x.append(p[0])
    y.append(p[1])
fid.close()

plt.plot(x,y)
plt.show()
```

- La lecture de fichier se fait lignes par lignes par défaut sous Python.
- La méthode `split` sur une string (`str`) scinde la chaine de charactère en sous chaine. Le séparateur peut être spécifié en argument, dle cas par défaut considère un séparateur sous forme d'espace.

2.4 Parcours de répertoires

Python permet de s'interfacer aisément avec le système d'exploitation et d'accéder aux commandes du bash.

▷ Testez cette fois le code suivant

```
import os

#read environment variable
home_directory=os.environ['HOME']
print("You home directory is",home_directory)

#list files (equivalent of os.system("ls "+home_directory))
directory_in_home=os.listdir(home_directory)
print(len(directory_in_home),"files or dir in your home path")

#Use the bash operation
os.system("mkdir -p new_directory")
print("new_directory/ is created")

#read the output of the OS call
process_txt=os.popen("ps").read()
process=process_txt.split("\n") #split every line
nbr_process=len(process)-1 #first line do not count
print("There is currently",nbr_process,"in this window")
print("The first process is \"",process[1],"\"")
```

2.5 Polynome

Soit un polynôme quelconque dont les coefficients sont quelconques (potentiellement aléatoires). On pourra considérer le polynome de degré d sous la forme

$$P(x) = \sum_{k=0}^{k=d} \alpha_k \, x^k \, ,$$

avec $\alpha_d = 1$.

On rappelle que l'on appelle matrice compagnon du polynome P la matrice C telle que

$$\mathrm{C} = \begin{pmatrix} 0 & 0 & \dots & 0 & -\alpha_0 \\ 1 & 0 & \dots & 0 & -\alpha_1 \\ 0 & 1 & \dots & 0 & -\alpha_2 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{d-1} \end{pmatrix}.$$

Les valeurs propres de la matrice C sont les racines du polynôme P.

▷ Construisez une fonction Python qui, étant donné un polynôme donné par ses coefficients sous forme de liste de nombre renvoie sa matrice compagnon (matrice numpy).

▷ Affichez la position des racines du polynome dans le plan complexe. Quelles propriétées doivent vérifier les racines complexes si les coefficients du polynome sont réels?

Aide

- Un nombre complex appartient à la classe `complex`.
- Le nombre complex unitaire s'écrit `1j`.
- Étant donné un nombre complex z, il est possible d'accéder à sa partie réelle et imaginaire par l'appel à `z.real` et `z.imag`.
- Les valeurs/vecteurs propres d'une matrices peuvent être calculés par la fonction `eigenval,eigenvec=np.linalg.eig([MATRICE])`.
- Les axes x et y peuvent avoir les même échelles de longueurs suite à l'appel à `plt.axis("equal")`.
- Il est possible de limiter la taille des axes de l'affichage par l'appel à `plt.axis([x_min,x_max,y_min,y_max])`.

2.6 Analyse de système

▷ Afficher le graphique de l'histrogramme du nombre de lignes de l'ensemble de vos fichiers sources et affichez tous les fichiers qui sont trop longs.

Pour cela, vous parcourerez l'ensemble de vos fichiers de manière récursive à partir d'un répertoire de départ. Si un fichier termine par .c ou .cpp, il est considéré comme un fichier source. Comptabilisez alors son nombre de ligne et stockez ce nombre dans un tableau. Si le nombre de ligne est supérieur à 500, affichez également ce fichier avec un warning. Enfin, affichez le graphique de l'histogramme du nombre de fichier ayant un certain nombre de lignes.

Aide

- Il est possible de parcourir récursivement vos répertoires à partir d'un répetoire source à l'aide du code suivant (`[ROOT]` désigne le répertoire de départ).

```
for root,dirs,files in os.walk([ROOT]):
  for name in files: #name designe le fichier courant
    path=os.path.join(root,name) #chemin complet du fichier
```

- La commande bash `wc -l [FICHIER]` permet de connaitre le nombre de lignes d'un fichier.
- Il est possible de vérifier si un fichier débute ou termine par une chaine de caractère particulière à l'aide des méthodes `beginswidth(chaine)` et `endswidth(chaine)`.
- Il est possible d'afficher l'histogramme d'un ensemble de valeur à l'aide du code suivant

```
hist,bins=np.histogram(values)
width=0.7*(bins[1]-bins[0])
center=(bins[:-1]+bins[1:])/2
plt.bar(center,hist,align='center',width=width)
plt.show()
```