[ENONCE] 2020-2021 Interpolation

October 14, 2020

Interpolation

```
Evaluation :
<ul>
    <li>Polynôme d'interpolation via la résolution de systèmes.</li>
    <li>Polynôme d'interpolation de Lagrange.</li>
</ul>
```

Recommandations :

- Respectez rigoureusement l'interface des fonctions, et les consignes.
- Copiez-collez le code de votre fonction (ne contenant aucun print ou autre effet de bord)
- Utilisez les alias définis ci-dessous.
- Même si vous travaillez à plusieurs, faites un dépôt par personne.

Objectifs :

- Comprendre des usages de l'interpolation
- Implémenter une méthode d'interpolation : affine par morceaux, splines, polynômes de Lagrange
- Connaitre le phénomène de Runge
- Utiliser des systèmes linéaires pour résoudre des problèmes d'interpolation 1D, 2D
- Utliser un outil d'interpolation pour construire une graphique

[3]:
```
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt

import random

from scipy.interpolate import interp1d
```

Pour aller plus loin :

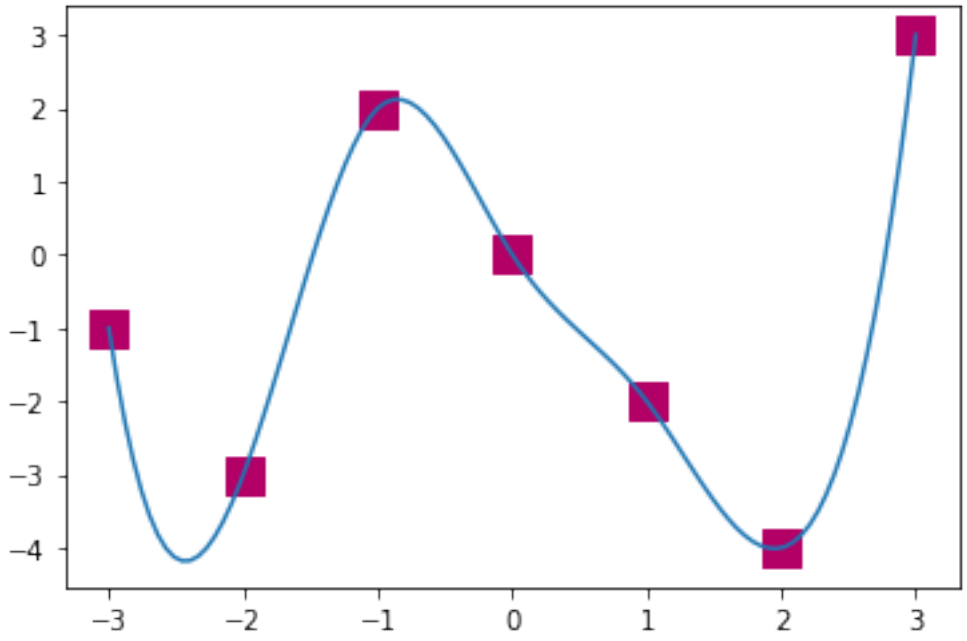
Les paragraphes "pour aller plus loin" sont à traiter en seconde lecture, après avoir fini tous les autres points

Introduction

- Etant donné un nuage de points, la problématique de l'interpolation est de contruire une fonction passant par tous les points du nuage.

- A partir d'un nuage de point, le problème est de modéliser ce nuage par une fonction. La fonction servant alors à déduire des valeurs manquantes.
- Inversement, une méthode d'interpolation accompagnée d'un nuage de point permet de représenter en machine, de manière efficace, une courbe, une image (SVG), un objet (3D), etc.

[4]:
```
"""
ATTENTION : Il y a peut-être un problème de compatibilité avec les couleurs.
Dans ce document, on utilise la syntaxe [[R,G,B]].
Si elle ne fonctionne pas, remplacer par (R,G,B)
"""

# Une nuage de points

X = np.array([-3.0,-2.0,-1.0, 0.0, 1.0, 2.0, 3.0])
Y = np.array([-1.0,-3.0, 2.0, 0.0,-2.0,-4.0, 3.0])

plt.scatter(X,Y,s = 200, marker = 's', color = [[0.7,0.0,0.4]]) # s : size;␣
 ↪marker "square"

x = np.linspace(-3,3,100)
f = interp1d(X,Y,kind='cubic')

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



Exercice de TP : Fonction "sur l'étagère"

Utilisez la fonction `interp1d` : faites 3 appels avec des arguments différents, avec le nuage de points ci-dessous et en vous aidant des documents de référence.

- Référence sur la fonction `interp1d`
- Tutoriel sur l'interpolation

Faites une répresentation graphique des résultats.

Appelez l'enseignant pour qu'il valide votre représentation graphique.

[5]:
```
# Insérez vos tests

X = np.array([-3.0,-2.1,-1.9,-1.1,-0.9, 0.9, 1.1, 1.9, 2.1, 3.0])
Y = np.array([-1.0,-1.0, 1.0, 1.0,-2.0,-2.0, 2.0, 2.0, 0.0, 0.0])
```

Fonctions par morceaux

Fonctions affines

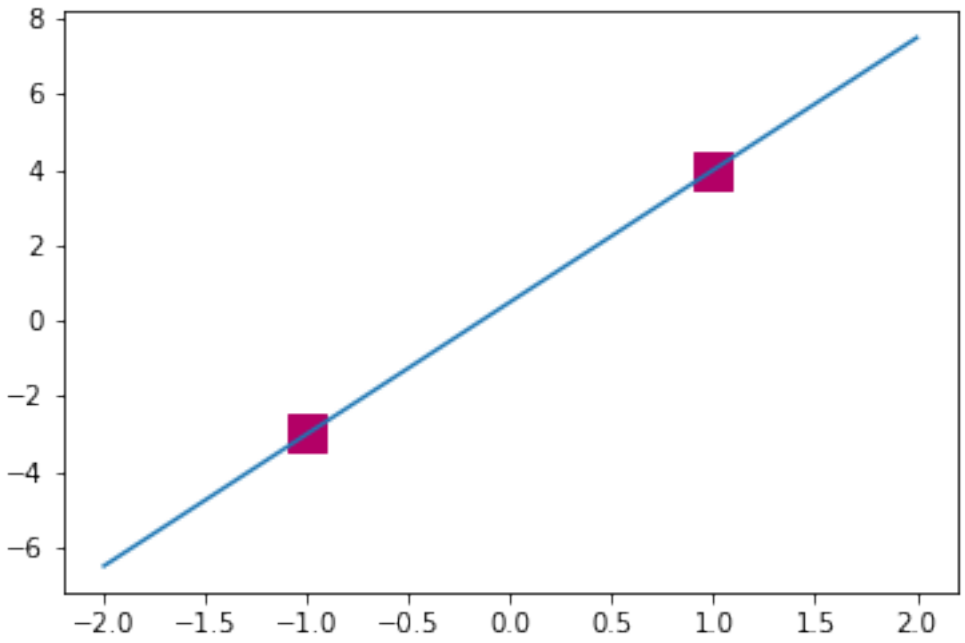
C'est la stratégie la plus simple : deux points consécutifs sont reliés par un segment de droite.

Etant donnés deux points $A(x_A, y_A)$ et $B(x_B, y_B)$ la droite d'interpolation de ces deux points a pour équation :

$$y = \frac{y_B - y_A}{x_B - x_A}x + \frac{y_A x_B - y_B x_A}{x_B - x_A}$$

Remarque : Il y a des hypothèses sur les points A et B.

[6]:
```
# Exemple
A = np.array([-1,-3])
B = np.array([ 1, 4])

y = lambda x : (B[1]-A[1])/(B[0]-A[0])*x + (A[1]*B[0] - B[1]*A[0])/(B[0]-A[0])

x = np.linspace(-2,2,10)

plt.scatter([A[0],B[0]],[A[1],B[1]],s = 200, marker = 's', color = [[0.7,0.0,0.
↪4]])
plt.plot(x,y(x))
plt.show()
```



Exercice de CM :

Déterminez une équation de la droite passant par les points $A(-1;-3)$ et $B(1;4)$

Exercice de TP :

- Etant donné un nuage de points dont les abscisses sont stockés dans un `numpy.array` X et les ordonnées dans un `numpy.array` Y, tracez la ligne brisée passant par tous les points du nuage.
- On suppose que les abscisses sont ordonnées de manière croissante
- Utilisez également un `scatter` pour faire apparaitre le nuage sur le même graphique.

Appelez l'enseignant pour qu'il valide votre représentation graphique.

[7]:
```
# Insérez votre code
```

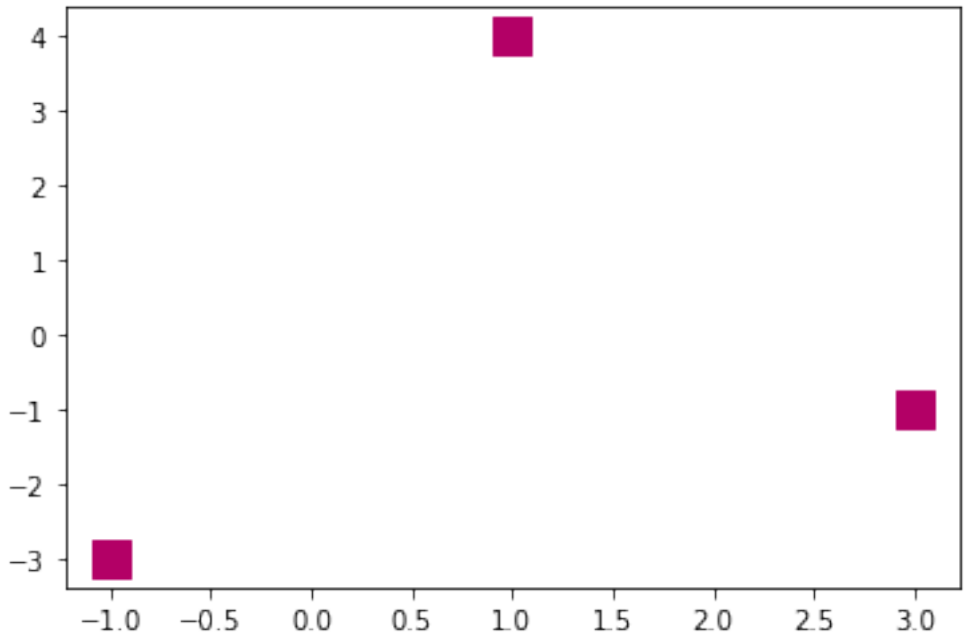
Fonctions quadratiques

Etant donnés deux points, il y passe une unique droite mais une infinité de paraboles.

Il est alors possible de contrôler les tangentes.

[8]:
```
# Exemple
A = np.array([-1,-3])
B = np.array([ 1, 4])
C = np.array([ 3,-1])

"""
On cherche deux paraboles :
```

```
+ La première passe par A et B
+ La seconde par B et C
+ Les tangentes des paraboles en B sont identiques.

!!! A compléter en CM !!!
"""
x = np.linspace(-2,2,10)

plt.scatter([A[0],B[0],C[0]],[A[1],B[1],C[1]],s = 200, marker = 's', color =␣
↪[[0.7,0.0,0.4]])

plt.show()
```



Exercice de CM :

On considère les points $A(-1;-3), B(1;4)$ et $C(3;-1)$

Déterminez une équation des paraboles suivantes :

- Passant par A et B et ayant une tangente horizontale en A
- Passant par A, B et C

Déterminez les équations du couple de parabole défini dans le commetaire de l'exemple ci-dessus.

[9]:
```
# Elements pour le TP

"""
Résolution de systèmes linéaires AX = B
```

```
avec A matrice n x n et B matrice n x 1

ATTENTION : Ne fonctionne qu'avec les systèmes de Cramer ...
"""

A = np.array([[1,1,1,1],[1,-1,1,-1],[1,-1,-1,1],[1,1,-1,-1]])
print(A) # Matrice du système
B = np.array([[1],[1],[0],[0.5]])
print(B) # Second membre
X = np.linalg.solve(A,B) # Résolution du système
print(X)
```

```
[[ 1  1  1  1]
 [ 1 -1  1 -1]
 [ 1 -1 -1  1]
 [ 1  1 -1 -1]]
[[1. ]
 [1. ]
 [0. ]
 [0.5]]
[[ 0.625]
 [ 0.125]
 [ 0.375]
 [-0.125]]
```

Exercice de TP :

- Etant donné un nuage de points dont les abscisses sont stockés dans un `numpy.array` X et les ordonnées dans un `numpy.array` Y, tracez courbe définies par morceaux par des paraboles joignant deux points consécutifs, telles qu'aux points de raccords les tangentes coïncident.
- Afin d'obtenir un système de Cramer, ajoutez une tangente horizontale au début.
- On suppose que les abscisses sont ordonnées de manière croissante
- Utilisez également un `scatter` pour faire apparaitre le nuage sur le même graphique.

Appelez l'enseignant pour qu'il valide votre représentation graphique.

[10]:
```
# Insérez votre code
```

Pour aller plus loin :

Parmi les méthodes les plus répandues d'interpolation polynômiale par morceaux, on trouve les **splines cubiques**.

Il s'agit d'une généralisation de ce qui précède à des polynômes de degré 3.

L'augmentation d'un degré, permet d'imposer une dérivée seconde nulle aux points de raccord.

A vous de jouer !

[11]:
```
# Insérez votre code
```

Fonctions polynômiales

Interpolation de Lagrange

L'augmentation du degré du polynôme a permis dans le cas précédent de contrôler les tangentes.

Dans le cas de l'interpolation de Lagrange, il permet de passer par plus de points.

Cas général Attention aux notations

Passez du temps sur ces formules, pour bien les comprendre

- On se donne n-points dont les abscisses sont $X = [x_1, \ldots, x_n]$ et les ordonnées $Y = [y_1, \ldots, y_n]$.
- On construit dans un premier temps un polynôme P_i qui s'annulle en tous les $x_j, j \neq i$ et qui vaut 1 en x_i.

$$P_i(x) = \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

$$P_i(x_i) = 1$$

$$P_i(x_j) = 0, j \neq i$$

- On construit l'interpolateur de Lagrange passant par les points donnés :

$$P(x) = \sum_{i=1}^{n} y_i P_i(x)$$

Remarque : Le polynôme P_i ne dépend que des abscisses X

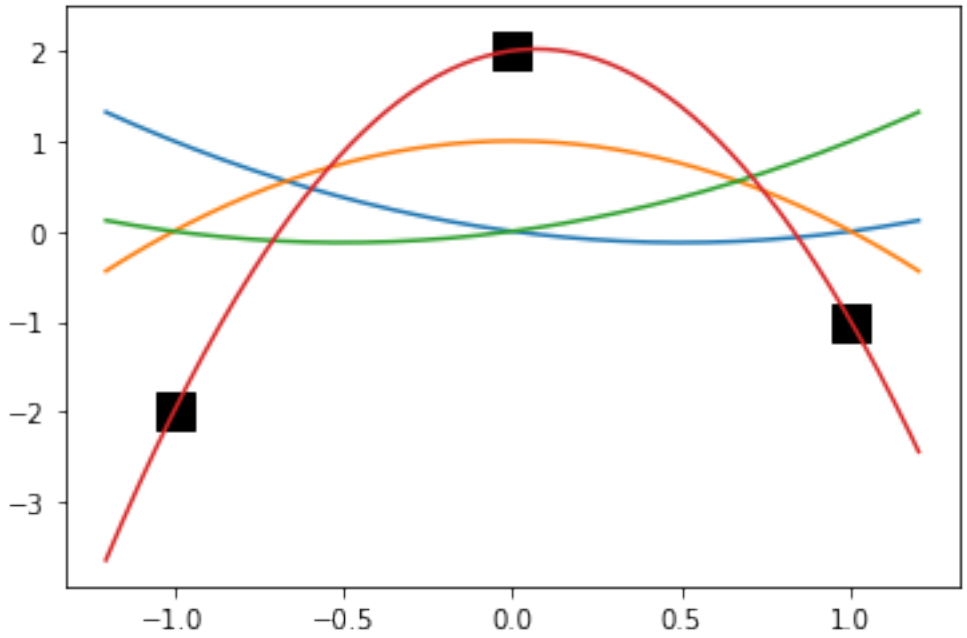
Exercice de CM :

On considère les trois points : $(-1; -2), (0; 2), (1; -1)$.

En utilisant les formules ci-dessus, calculez les polynômes P_i et le polynôme P ci-dessus.

[12]:
```
# Exemple

# Les abscissese : -1, 0, 1

# P_1 : vaut 1 en -1, 0 en 0, et 0 en 1
def P1(x):
    return ((x-0)/(-1-0))*((x-1)/(-1-1))
def P2(x):
    return ((x-(-1))/(0-(-1)))*((x-1)/(0-1))
def P3(x):
    return ((x-(-1))/(1-(-1)))*((x-0)/(1-0))

# Les ordonnées respectives : -2, 2, -1
def P(x):
    return -2*P1(x)+2*P2(x)-1*P3(x)

x = np.linspace(-1.2,1.2,200)
plt.plot(x,P1(x))
plt.plot(x,P2(x))
```

```
plt.plot(x,P3(x))
plt.plot(x,P(x))
plt.scatter([-1,0,1],[-2,2,-1],200, [[0,0,0]],'s')
plt.show()
```



Eléments de programmation fonctionnelle `Python`, et bien d'autres langages, permettent d'utiliser les fonctions de la même manière que tous les autres objets.

En particulier, il est possible :

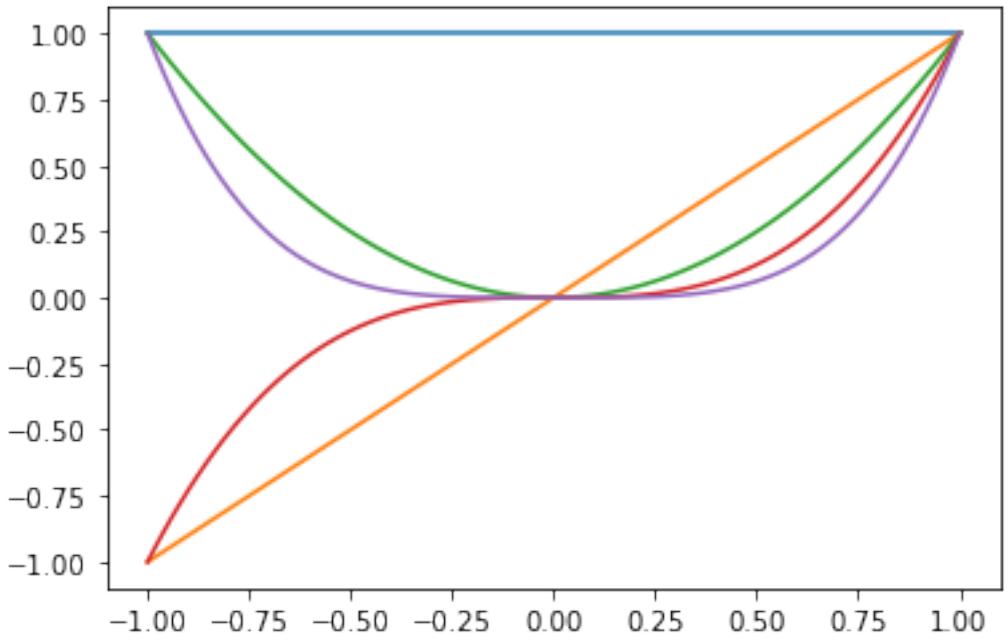
- De faire passer des fonctions en paramètre d'autres fonctions
- De retourner une fonction comme résultat d'une autre fonction.

La fonction `interp1d` utilisée précédemment est une fonction retournant une fonction.

[13]:
```
# Elements pour le TP
def valdeux(f): # Une fonction comme paramètre
    return f(2)

def valix(f,x): # Autre exemple
    return f(x)

def puis(n):
    def mono(x): # définition d'une fonction à l'intérieur d'une autre fonction
        return x**n
    return mono # retourne la fonction créée
```

```python
def expo(f,n): # Une fonction en paramètre et en retour
    def res(x):
        return f(x) ** n
    return res

def expodeux(f,n): # Autre version avec `lambda`
    return lambda x : f(x) ** n

f = puis(2)
print(f)
print(f(3))
print(puis(2)(3))

x = np.linspace(-1,1,100)
for i in range(5):
    plt.plot(x,puis(i)(x))

plt.show()
```

```
<function puis.<locals>.mono at 0x7f52e2ff2a60>
9
9
```



Exercice de TP :

Attention aux notations

- Ecrivez une fonction `LagrangeI(X,i)` qui retourne la fonction $x \mapsto P_i(x)$ où P_i est le $i^{\text{ième}}$ polynôme interpolateur de Lagrange. Le paramètre X contient les abscisses de tous les points. Le polynôme P_i s'annule en tous les points donnés sauf celui d'indice i.
- Ecrivez une fonction `Lagrange(X,Y)` qui retourne la fonction $x \mapsto P(x)$ où P est le polynôme interpolateur de Lagrange. Le paramètre X contient les abscisses de tous les points, Y les ordonnées. Le polynôme P passe par tous les points donnés.
- Testez vos fonctions avec les points $X = [-1, 0, 1]$ et $Y = [1, -1, 4]$. Tracez les courbes représentatives des P_i et de P.
- Testez à nouveau vos fonctions avec le nuage de points inital et observez un beau phénomène de Runge (oscillation incontrôlées).

Déposez vos fonctions `LagrangeI(X,i)` et `Lagrange(X,Y)` sur Moodle avant la fin de la semaine

[14]:
```
# Insérez votre code
```

[15]:
```
# Tests
```

Dessin comme un SVG

Observez le code source des fichiers :

- arbre
- feuille

Il s'agit de "dessin vectoriel" : l'information stockée correspond à une description des courbes, pas de leur tracé.

Exercice de TP :

Dans cette partie, vous aurez besoin de trois outils :

- Tracer le segment de droite $[A, B]$ avec les coordonnées de A et B sous la forme `A = np.array([2,9])` (ce problème a été résolu précédemment)
- Tracer la parabole joignant A, B et C avec les coordonnées de A, B et C sous la forme `A = np.array([2,9])` (nouveau problème à résoudre)
- Tracer la cubique joignant A et B dont les tangentes en A et B sont imposées; les points seront alors donnés sous la forme `A = np.array([abscisseA,ordonnéeA,pentetangenteA])`. (nouveau problème à résoudre)

Quelques indications :

- Une cubique a pour équation $ax^3 + bx^2 + cx + d$ (4 paramètres inconnus).
- Le problème possède 4 contraintes : passer par A, passer par B, avoir la bonne pente en A, avoir la bonne pente en B.
- Pour résoudre le problème, il suffit de résoudre un système linéaire 4 x 4 dont les inconnues sont a, b, c, d.

Les points à interpoler :

- Droites :
  - $(-9, -1), (-6, 8)$,
  - $(2, 9), (7, 8)$,

  - $(7, 8), (10, -1)$.
- Paraboles :
  - $(-13; -3), (-11; 0), (-8; 2)$
  - $(-13; -3), (-5; -8), (0; -9)$
  - $(0; -9), (7; -8), (15; -3)$
  - $(9; 2), (13; 0), (15; -3)$
- Cubiques :
  - $(-6, 8, 0), (-2, 7, -1)$,
  - $(-2, 7, 0), (2, 9, 2)$,
  - $(-8, 2, -0.5), (0, 0, 0)$,
  - $(0, 0, 0), (9, 2, 0.5)$
  - $(-9, -1, -0.5)$, $(0, -3, 0)$
  - $(0, -3, 0)$, $(10, -1, 0.5)$

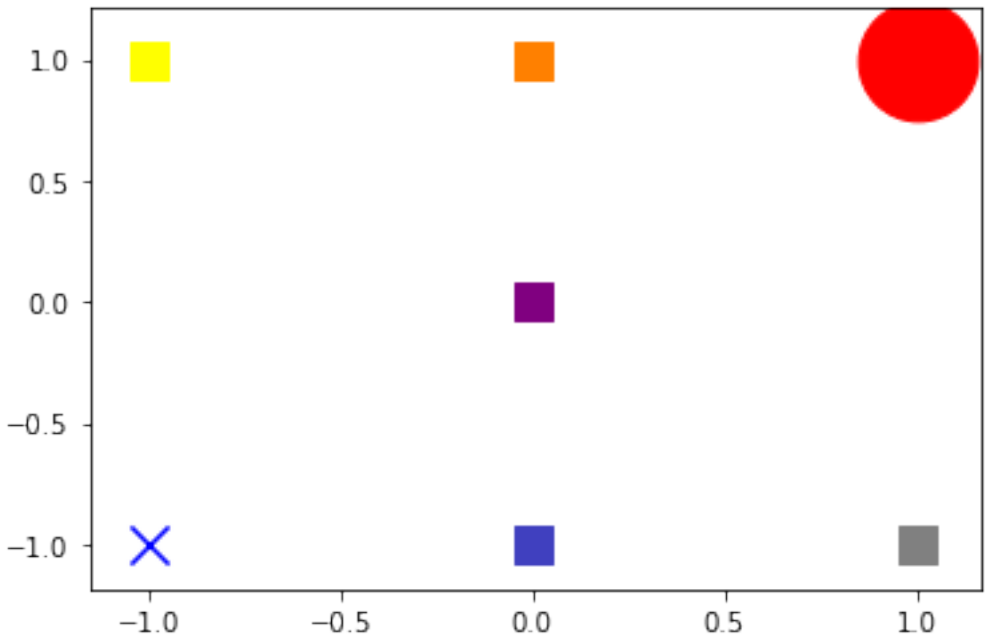
[16]:
```
# Insérez votre code
```

Appelez l'enseignant pour qu'il valide votre représentation graphique.

Interpolation 2D

Un beau dessin vaut mieux qu'un long discours ...

[17]:
```
# En TP, il faudra faire de belles interpolations !

# Points d'entrée
plt.scatter(1,1,2000,[[1.0,0.0,0.0]],'o') # Rouge
plt.scatter(-1,1,200,[[1.0,1.0,0.0]],'s') # Jaune
plt.scatter(-1,-1,200,[[0.0,0.0,1.0]],'x') # Bleu
plt.scatter(1,-1,200,[[0.5,0.5,0.5]],'s') # Gris

# Points calculés

# Au milieu d'un coté (interpolation linéaire entre deux points)
plt.scatter(0,1,200,[[1.0,0.5,0.0]],'s') # Orange
plt.scatter(0,-1,200,[[0.25,0.25,0.75]],'s') # Gris-bleu

# Au milieu du carré (interpolation linéaire entre quatre points)

# Centre : moyenne des quatre coins :
# matplotlib.pyplot.scatter(0,0,200,[[0.625,0.375,0.375]],'s')

# Centre : moyenne d'une diagonale
plt.scatter(0,0,200,[[0.5,0.0,0.5]],'s')
plt.show()
```



Calculs :

- Dans la situation ci-dessus, l'interpolation se fait dans l'espace RGB des couleurs, en fonction de la position des points dans le plan.
- Chacune des composantes `RGB` de la couleur d'un point est calculée en fonction de sa position (x, y) :

$$r = r(x, y), g = g(x, y), b = b(x, y)$$

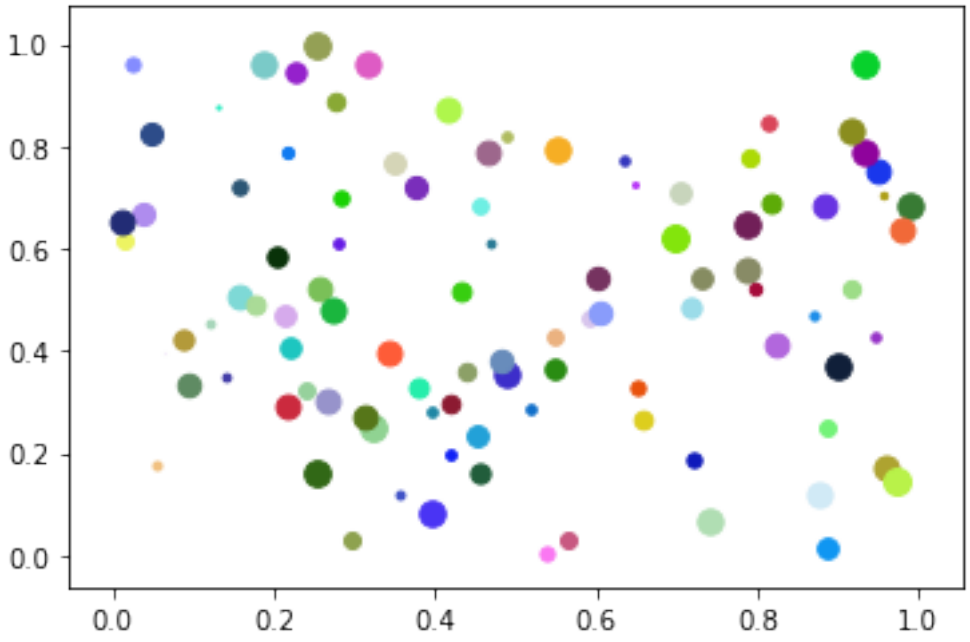
- Détaillons la composante r, les autres calculs sont identiques. Le modèle (plus ou moins arbitraire : 4 inconnues pour 4 sommets) choisi est :

$$r(x, y) = a + bx + cy + dxy$$

- Il reste à trouver les coefficients a, b, c, d en fonction des données (les points connus). C'est un système linéaire à résoudre ... donc c'est facile :-)

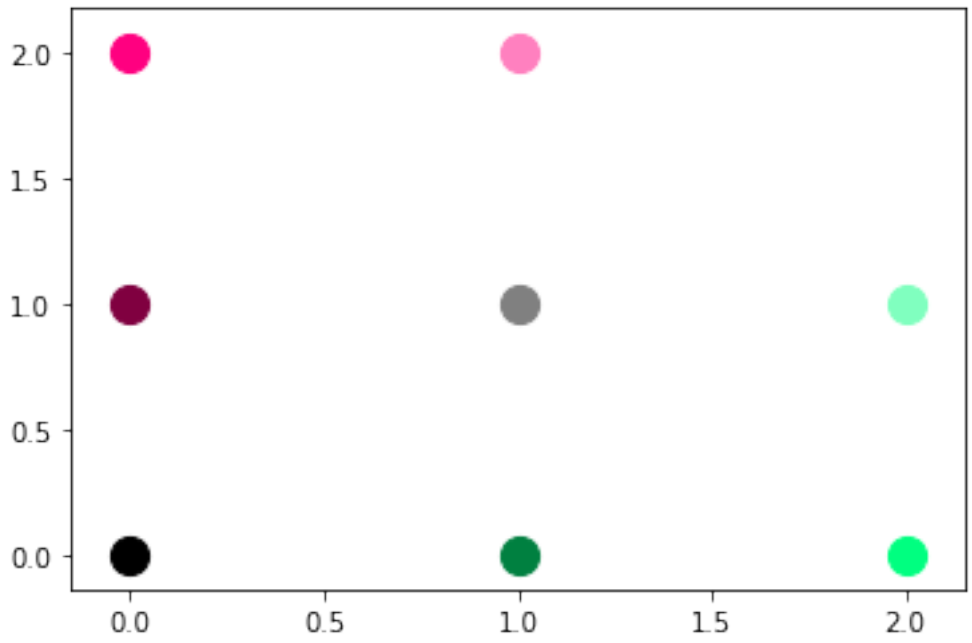
[18]:
```
# Eléments pour le TP

"""
Dessiner des points de couleur et taille variable
"""
def point(x,y,size,color):
    plt.scatter(x,y,size,color)

def children_draw(n):
    for i in range(100):
        x = random.random()
        y = random.random()
```

```
        s = random.random()*100
        color = [[random.random(),random.random(),random.random()]]
        point(x,y,s,color)
    plt.show()

children_draw(200)
```



Exercice de TP

[19]:
```
# On considère les 9 points suivant :

X = np.array([0,1,2,0,1,2,0,1,2])
Y = np.array([0,0,0,1,1,1,2,2,2])
R = np.array([0,0,0,0.5,0.5,0.5,1,1,1])
G = np.array([0,0.5,1,0,0.5,1,0,0.5,1])
B = np.array([0,0.25,0.5,0.25,0.5,0.75,0.5,0.75,1])

for i in range(9):
    plt.scatter(X[i],Y[i],200,[[R[i],G[i],B[i]]])

plt.show()
```



- La figure ci-dessus est composée de 9 définissant 4 “petits carrés”. Le point gris au centre est un sommet commun aux 4 carrés.
- Ecrivez une fonction `addPoint(X,Y)` qui ajoute un point à la figure. Les coordoonées (x, y) de ce point seront choisies aléatoirement (entre 0 et 2). Les trois canaux de couleur seront calculés à partir de la méthode d’interpolation vue en cours.
- La fonction `addPoint(X,Y)` fera de l’interpolation à chaque fois entre quatre points, pour les quatre carrés de la figure. Une fois tirées aléatoirement, les coordonnées (x, y) placent le point considéré dans un des 4 carrés de la figure. Les couleurs des sommets de ce carré, dans lequel se situe le point, seront utilisées pour interpoler la couleur du point tiré.

Appelez l’enseignant pour qu’il valide votre représentation graphique.

[20]:
```
# Insérez votre code
```

Pour aller plus loin :

Généraliser votre fonction `addPoint` pour qu’elle travaille avec un quadrillage quelconque en entrée.

[21]:
```
# Insérez votre code
```

Attention :

- Pour le TP02, pendant la séance, vous avez 4 exercices à montrer : graphique de `interp1d`, graphique de ligne brisée, graphique “comme un SVG”, graphique d’interpolation 2d.
- Pour le TP02, avant la fin de semaine, vous avez 2 fonctions à déposer : `LagrangeI(X,i)` et `Lagrange(X,Y)`.
- Lors de l’évaluation sur feuille, il vous sera demandé de calculer “à la main”, des polynômes d’interpolation. Il faudra connaitre les formules pour les polynômes de Lagrange et savoir

résoudre des systèmes linéaires.

[ ]: