BUT1 – OUTILS MATHÉMATIQUES FONDAMENTAUX
TRAVAUX PRATIQUES

IUT DE NANTES – DÉPARTEMENT INFORMATIQUE – 2021/2022

Table des matières

Le but de ces TP est notamment de vous faire découvrir un certain nombre de librairies Python, servant à l'étude expérimentale et la résolutions de problèmes mathématiques ou plus généralement scientifiques.

Il est important de synthétiser et consigner vos expérimentations et résultats dans un document bien rédigé, de manière soignée. Cela est nécessaire pour vos révisions qui seront utiles avant toute évaluation.

L'auteur remercie P. Rannou et D. Gobin d'avoir partagé leurs ressources.

TP 1 – Graphes de fonctions

Exercice 1.1 – Numpy et Matplotlib – Dans cet exercice, on apprend à utiliser deux librairies de base en Python. La librairie `numpy` gère un certain nombre d'objets classiques en maths (comme les matrices). La librairie `matplolib.pyplot` gère quant à elle l'affichage de graphiques (on utilise ici qu'une sous-librairie de `matplotlib`). Ces deux librairies sont très complètes. Ainsi, vous pouvez consulter les sites suivants afin de vous documenter plus en détail :

- Un premier site (un peu vilain) qui contient une doc assez simple sur différentes librairies Python : http://www.python-simple.com/, avec notamment :
  - http://www.python-simple.com/python-numpy/creation-numpy.php pour la librairie `numpy`
  - http://www.python-simple.com/python-matplotlib/pyplot.php pour la librairie `matplotlib.pyplot`
- Le site officiel de la librairie `numpy` : https://numpy.org/
- Il existe énormément de possibilités avec la librairie `matplotlib` : on pourra consulter les différents exemples présents sur le site officiel https://matplotlib.org/ qui, dans des cas complexes, peuvent permettre de partir d'un code existant pour le modifier et arriver au résultat souhaité.

Classiquement, on importe ces deux librairies de la manière suivante :

```
1 import numpy as np
2 import matpotlib.pyplot as plt
```

1. On commence par présenter quelques objets de la librairie `numpy`. Déterminer ce qu'effectuent les commandes suivantes :

```
3  np.array([3,7,-1,2])
4  np.ones(4)
5  np.array([3,7,-1,2])+np.ones(4)
6  np.array([3,7,-1,2])+1
7  np.array([[3,7],[-1,2]])
8  np.array([1,2,3,4])**2
9  np.arange(1,5,1)
10 np.arange(10,30,5)
11 np.linspace(0,2,9)
12 np.sin(np.linspace(0,2*np.pi,20))
```

   (Noter les différents résultats que vous obtenez dans votre document de synthèse).

   La librairie `numpy` contient un certain nombre d'autres fonctions mathématiques usuelles.

2. On va maintenant utiliser la librairie `matplotlib.pyplot` pour tracer des graphes de fonctions.

   a) On commence par faire des tracés simples. On définit deux tableaux

```
13 x=np.array([1,3,5,7,10,13])
14 y=np.array([3,2,0,1,-4,6])
```

puis on effectue les deux commandes suivantes :

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

À quoi correspondent-elles ?

b) Plusieurs options sont disponibles :

— `color = 'r'` : change la couleur de la ligne en rouge

— `linestyle = 'dashed'` : affiche la ligne en pointillés

— `marker = 'o'` : place un cercle sur chaque point dessiné

— ...

On peut également afficher des légendes sur le graphique :

— `plt.legend(["fonction 1","fonction 2"])` : pour préciser les fonctions tracées

— `plt.ylabel("unité ord.")` : pour un label sur l'axe des ordonnées

— `plt.xlabel("unité abs.")` : pour un label sur l'axe des abscisses

— `plt.title("titre graphique")` : pour un titre au graphique

Déterminer une suite de commandes permettant d'afficher les fonctions f et g définies par :

$$f(x) = x^2 \sin(x) + 4 \qquad \text{et} \qquad g(x) = \frac{30}{x^2+1}$$

sur l'intervalle $[0, 3\pi]$, l'une en rouge avec ligne en pointillés et l'autre en vert avec ligne continue. On pourra commencer par discrétiser l'intervalle $[0, 3\pi]$ (la constante π est donnée par `np.pi`) avec 100 valeurs en se servant de la commande `np.linspace`.

c) On peut aussi utiliser la fonction `plt.clf()` pour effacer la fenêtre graphique courante.

```
17 plt.clf()
18 plt.scatter(x,y)
19 plt.show()
```

Constatez ce que fait la commande `plt.scatter(x,y)`.

d) Pour gérer finement nos graphiques, on peut avoir une utilisation plus avancée de `matplotlib`. Un affichage graphique est constitué de trois "couches". Tout d'abord d'une fenêtre (qu'on appelle `figure`), puis d'un ou plusieurs systèmes d'axes de coordonnées (ou calques qu'on appelle `axes`), et enfin d'objets graphiques tracés dans les systèmes d'axes.

On commence par définir les fonctions que l'on veut tracer.

```
1 import numpy as np
2 import matplotlib.pyplot
3 x = np.linspace(0,10,200)
4 y = np.sin(x)
5 z = np.cos(x)
```

On ouvre ensuite une fenêtre graphique

```
6 fig = plt.figure(1)
```

On ajoute ensuite un système d'axes par la commande `gca` ("get current axes")

```
7 ax = fig.gca()
```

On peut ensuite créer les objets graphiques désirés dans le système d'axes

```
8 line1 = ax.plot(x,y,label="sinus")
9 line2 = ax.plot(x,z,label="cosinus")
```

Mettons un titre

```
10 ax.set_title("Fonctions sinus et cosinus")
```

puis affichons les légendes

```
11 ax.legend()
```

On étend ensuite les limites du graphique :

```
12 ax.set_xlim(-1,13)
13 ax.set_ylim(-1.2,1.2)
```

On modifie les points de repère de l'axe des abscisses

```
14 plt.xticks([0,np.pi,2*np.pi,3*np.pi],[r'$0$',r'$\pi$',r'$2\pi$',r'$3\pi$'])
```

(Dans cette dernière commande, les chaînes de caractères servant à indicer l'axe des abscisses utilise le langage LaTeX qui est utilisé pour rédiger des documents scientifiques).

On peut également ajouter un point sur le graphique, ou annoter le graphique avec du texte

```
15 point = ax.scatter(np.pi,np.cos(np.pi),label="")
16 ax.annotate("texte",xy=(np.pi,np.cos(np.pi)),xytext=(np.pi,np.cos(np.pi)+0.1))
```

Si on veut enregistrer l'image générée dans un fichier, on peut utiliser la commande suivante (où l'on pourra remplacer `.pdf` par `.png` si l'on veut sauvegarder l'image sous un autre format)

```
17 plt.savefig("fichier.pdf")
```

Effectuer les précédentes commandes pour constater le résultat.

e) On peut aussi avoir envie de placer les axes comme un mathématicien. Effectuer les commandes suivantes une à une

```
import numpy as np
import matplotlib.pyplot
fig, ax = plt.subplots()
ax.spines["right"].set_color("none") # pas de trait a droite
ax.spines["top"].set_color("none") # pas de trait en haut
ax.spines["bottom"].set_position(("data",0)) # position de l'axe en x=0
ax.spines["left"].set_position(("data",0)) # position de l'axe en y=0
xx = np.linspace(-0.75, 1., 100)
ax.plot(xx, xx**3);
```

Exercice 1.2 – Résolution d'équation du second degré –

1. Écrire une fonction `resoudre(a,b,c)` qui renvoie la liste des racines réelles du polynôme du second degré $aX^2 + bX + c$. On fera attention au cas où $a = 0$.
2. Tester cette dernière fonction pour `(a,b,c)=(3,7/3,-5.2)` en évaluant la fonction $x \mapsto ax^2 + bx + c$ sur les valeurs obtenues.
3. Faire de même pour `(a,b,c)=(1,-12365478,2)`. Que constatez-vous ?
4. Écrire une fonction `parabole(a,b,c)` qui trace le graphe de la fonction polynomiale $x \mapsto ax^2 + bx + c$. On optimisera l'affichage grâce à l'exercice précédent. On marquera notamment le sommet avec un point de couleur sur la courbe et on indiquera les valeurs des racines (si elles existent) sur le graphique.

Exercice 1.3 – Approximation locale : les développements limités – Cet exercice a pour but d'illustrer le fait que toute fonction peut localement être approximé par un polynôme (sous condition d'être assez dérivable).

Pour cela, on va utiliser une librairie Python appelée SymPy utile pour le calcul formel. Une telle librairie de calcul formel nous permet de travailler avec des expressions mathématiques comme nous le ferions à la main.

1. *Initiation à* SymPy. Pour commencer, on illustre quelques fonctionnements de base de la librairie SymPy. Pour cela, on exécutera les lignes de code suivantes.

   a) On commence par importer la librairie SymPy, puis on définit un symbole formel x par les lignes de code suivantes.

   ```
   import sympy as sp
   x = sp.symbols("x")
   ```

   Dans SymPy, on a un certain nombre de fonctions qui sont déjà implémentées. Attention : elles ne doivent pas être traitées de la même manière que les fonctions numpy.

   b) On peut ensuite définir des expressions mathématiques comme on écrirait des fonctions en maths, en utilisant notre symbole x comme une variable.

   ```
   expr_f = sp.cos(x)
   expr_g = sp.exp(x) +x**2 +x**0
   ```

On peut faire des opérations algébriques simples avec ces expressions :

```
print(expr_f)
print(expr_g)
print(expr_f + expr_g)
```

c) Si de telles expressions représentent des fonctions, on veut pouvoir les évaluer en un nombre. On pourra tester la commande suivante

```
expr_f(2)
```

qui renvoie un message d'erreur. Pour évaluer une expression en une valeur, on utilisera la méthode `.subs` de la manière suivante :

```
print(expr_f.subs(x,np.pi))
print(expr_g.subs(x,0))
```

d) Pour l'instant, on ne voit pas l'intêret d'une librairie de calcul formel. Cela apparait par exemple avec la fonction suivante.

```
expr_df = sp.diff(expr_f, x)
print(expr_df)
expr_dg = sp.diff(expr_g, x)
print(expr_dg)
expr_ddg = sp.diff(expr_g, x,2)
print(expr_dg)
expr_dddg = sp.diff(expr_g, x,3)
print(expr_dg)
```

Que font ces différentes lignes de codes ?

Pour plus d'informations sur la librairie `SymPy`, vous pourrez consulter l'adresse suivante : https://www.sympy.org/en/index.html

2. *Développpement limités :* À présent, on va utiliser ce qu'on vient de voir pour illustrer un nouvel objet mathématique : *les développements limités.*

   Considérons une fonction f définie sur une intervalle I, dérivable n-fois (possiblement infiniment dérivable). Soit $x_0 \in I$. Alors, au voisinage de x_0, on peut approximer l'expression de f par le polynôme suivant : pour x assez proche de x_0

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

   Ce polynôme est appelé *développement limité à l'ordre n de f en x_0*. (Remarque : on a ici un symbole approximation et non pas un symbole égalité. La théorie nous donne un contrôle de l'erreur que l'on commet ici, mais nous ne rentrerons pas dans ces détails).

   a) À quoi correspond le développement limité à l'ordre 1 de f en x_0 ?

   b) Calculer à la main le développement limité de la fonction $\cos$ en 0 à l'ordre 2, 3 et 4.

c) Coder une fonction `dev_lim(expr,x0,n)` qui prend en entrée l'expression symbolique `expr_f` d'une fonction f, un nombre réel `x0` et un entier `n` et qui renvoie l'expression du développement limité de f en x_0 à l'ordre n. Vérifier vos calculs précedents avec cette fonction.

d) Coder une fonction `trace_DL(expr_f,x0,n)` qui trace le graphe de la fonction f ainsi que les graphes de ces développements limités d'ordre compris entre 1 et n. (Attention : la méthode `.subs` ne peut pas prendre un vecteur (`np.array`) en argument. Il faudra palier à ce petit défaut.) Constater que pour des fonctions usuelles comme $\cos$, un développement limité à l'ordre 4 fournit une bonne approximation.

TP 2 – Polynômes – Géométrie plane

Exercice 2.1 – Polynôme – Un polynôme P de degré n à coefficient réel est la donnée de $n+1$ nombres $p_0, \ldots, p_n \in \mathbb{R}$ avec $p_n \neq 0$ que l'on notera de la manière suivante

$$P = p_0 + p_1 X + \ldots + p_{n-1} X^{n-1} + p_n X^n \ .$$

Numériquement, on codera un tel polynôme par une liste `P = [p_0,p_1, ..., p_n]`. Par exemple, le polynôme $P = 3X^3 + 5X^2 + 4$ sera représenté par la liste $[4, 0, 5, 3]$.

1. Écrire une fonction `degre(P)` qui renvoie le degré de P.

2. Écrire une fonction `somme(P,Q)` qui renvoie le polynôme $P + Q$

3. Écrire une fonction `produit(P,Q)` qui renvoie le polynôme $P \times Q$.

4. Écrire une fonction `affiche(P)` qui renvoie une chaine de caractères contenant le polynôme P comme un mathématicien l'aurait écrit.

5. Écrire une fonction `eval(P,x)` qui renvoie la valeur du polynôme P en x.

Exercice 2.2 – Division euclidienne de polynômes – On a vu (sur des exemples) que lorsque l'on a une solution a d'une équation polynomiale $P(x) = 0$, alors on peut factoriser le polynôme P par $(x - a)$, autrement dit, il existe un polynôme Q tel que, pour tout $x \in \mathbb{R}$

$$P(x) = (X - a)Q(x) \ .$$

On dit également que le polynôme P est divisible par le polynôme $(x - a)$.

De manière générale, il existe une division euclidienne des polynômes : c'est le sujet du théorème suivant

Théorème – Division euclidienne de polynômes

Soient A et B deux polynômes à coefficients réels, avec B non nul, il existe un unique couple (Q, R) de polynômes tels que $A = BQ + R$ et où R est de degré strictement plus petit que celui de B. On appelle alors Q le quotient et R le reste de la division euclidienne de A par B.

Étant donné deux polynômes A et B avec B non nul, on a un algorithme afin de déterminer les polynômes Q et R tels que $A = BQ + R$ avec $\deg R < \deg B$. On présente cet algorithme sur l'exemple suivant.

On considère les polynômes $A = X^5 - X^4 - X^3 + 3X^2 - 2X$ et $B = X^2 - X + 1$.

— On calcule un couple de polynôme (P_1, R_1) tel que R_1 soit de degré strictement inférieur à A : on prendra $P_1 = X^3$ et $R_1 = A - P_1 \times B$, ce que l'on résume de la manière suivante

$$\begin{array}{rcccccc|ccccc} A & = & X^5 & -X^4 & -X^3 & +3X^2 & -2X & X^2 & -X & +1 & = & B \\ -P_1 \times B & = & -X^5 & +X^4 & -X^3 & & & X^3 & & & = & P_1 \\ R_1 & = & & & -2X^3 & +3X^2 & -2X & & & & & \end{array}$$

— On réitère le procédé précédent sur le couple (R_1, B) : on prend $P_2 = -2X$ et $R_2 = R_1 - P_2 \times B$, ce qui nous donne :

$$\begin{array}{rcccccc|cccccc} A & = & X^5 & -X^4 & -X^3 & +3X^2 & -2X & X^2 & -X & +1 & = & B \\ -P_1 \times B & = & -X^5 & +X^4 & -X^3 & & & X^3 & -2X & & = & P_1 + P_2 \\ R_1 & = & & & -2X^3 & +3X^2 & -2X & & & & & \\ -P_2 \times B & = & & & +2X^3 & -2X^2 & +2X & & & & & \\ R_2 & = & & & & X^2 & & & & & & \end{array}$$

— On réitère le procédé précédent sur le couple (R_1, B) : on prend $P_2 = -2X$ et $R_2 = R_1 - P_2 \times B$, ce qui nous donne :

$$\begin{array}{rcccccc|cccccc} A & = & X^5 & -X^4 & -X^3 & +3X^2 & -2X & X^2 & -X & +1 & = & B \\ -P_1 \times B & = & -X^5 & +X^4 & -X^3 & & & X^3 & -2X & & = & P_1 + P_2 \\ R_1 & = & & & -2X^3 & +3X^2 & -2X & & & & & \\ -P_2 \times B & = & & & +2X^3 & -2X^2 & +2X & & & & & \\ R_2 & = & & & & X^2 & & & & & & \end{array}$$

— On réitère une dernière fois le procédé précédent sur le couple (R_2, B) : on prend $P_3 = 1$ et $R_3 = R_2 - P_3 \times B$, ce qui nous donne :

$$\begin{array}{rcccccc|cccccc} A & = & X^5 & -X^4 & -X^3 & +3X^2 & -2X & X^2 & -X & +1 & = & B \\ -P_1 \times B & = & -X^5 & +X^4 & -X^3 & & & X^3 & -2X & 1 & = & P_1 + P_2 + P_3 \\ R_1 & = & & & -2X^3 & +3X^2 & -2X & & & & & \\ -P_2 \times B & = & & & +2X^3 & -2X^2 & +2X & & & & & \\ R_2 & = & & & & X^2 & & & & & & \\ -P_3 \times B & = & & & & -X^2 & +X & -1 & & & & \\ R_3 & = & & & & & X & -1 & & & & \end{array}$$

— On conclut alors en posant $Q = P_1 + P_2 + P_3$ et $R = R_3$. On a bien

$$\underbrace{X^5 - X^4 - X^3 + 3X^2 - 2X}_{A} = \underbrace{(X^3 - 2X + 1)}_{Q}\underbrace{(X^2 - X + 1)}_{B} + \underbrace{(X - 1)}_{R} .$$

1. Calculer les divisions euclidiennes de A par B avec
   a) $A = 3X^3 + 2X^2 + X$ et $B = X - 3$;
   b) $A = X^4 - 2X^3 + 3X^2 - 1$ et $B = X^2 + 3X + 2$;
   c) $A = 2X^3 - \frac{2}{3}X^2 - \frac{11}{3}X + 1$ et $B = X - 1$;
2. Écrire une fonction `division_euclidienne(A,B)` qui renvoie le couple Q, R, le quotient et le reste de la division euclidienne de A par B.

TP 3 – Résolution de systèmes

Exercice 3.1 – Intersection de deux droites – Écrire une fonction `intersection_droites(D1,D2)` qui renvoie le point d'intersection des droites $\mathscr{D}_1$ et $\mathscr{D}_2$ ou alors si les droites sont parallèles, ou alors confondues. Une droite $\mathscr{D}$ sera donnée par une équation $ax + by + c = 0$ que l'on stockera informatiquement par la liste `D = [a,b,c]`.

Exercice 3.2 – Initiation à Numpy II : calcul matriciel –

1. On importe toujours la librairie `numpy` et on détermine ce qu'effectuent les commandes suivantes :

```
import numpy as np
np.zeros(7)
np.ones(6)
np.identity(3)
np.random.rand(3,4)
np.random.rand(-4,3,size=(3,5),dtype=int)
```

2. On définit deux matrices puis on teste différentes commandes : lesquelles ont un sens en algèbre linéaire ?

```
a = np.array([[1,3],[0,4]])
b = np.array([[4,0],[-1,1]])
a+b
a+4
a*b
3*a
a*3
np.add(a,b)
a.dot(b)
a @ b
np.linalg.matrix_power(a,2)
np.linalg.inv(a)
```

```
a.shapes
a.sum()
a.sum(axis=0)
a.sum(axis=1)
a.min()
a.max()
a[1]
a[0,1]
a[0][1]
```

3. Écrire une fonction `somme(A)` qui fait la somme des coefficients de la matrice A. Comparer le temps d'execution de la fonction `somme` avec celui de `A.sum()`. (Pour cela, on pourra utiliser la librairie `time` et la fonction `time.time()` avec A et B de grandes matrices tirées aléatoirement.)
4. Écrire une fonction `produit(A,B)` qui, à partir de deux matrices A et B, renvoie le produit $A \times B$ si celui-ci existe et renvoie la phrase "le produit est impossible" sinon. Comparer le temps d'execution de la fonction `produit` avec la fonction `A.dot(B)`.

Exercice 3.3 – L'algorithme du pivot de Gauss – On a présenté en cours et en TD l'algorithme du pivot de Gauss. Cet algorithme s'effectue bien sur de petits systèmes mais devient rapidement très fastidieux lorsque la taille du système devient grande. Nous allons donc coder cet algorithme : pour cela, on va le découper en plusieurs fonctions.

Dans tout cet exercice, on considère le système

$$AX = b$$

où $A = (a_{ij})$ est une matrice carrée de taille n et tel que b est une matrice colonne ayant le même nombre de lignes que A.

Le choix de pivot en numérique : on a vu en cours que lors de l'algorithme du pivot de Gauss, on doit choisir nos pivots non-nuls. Mais y-a-t-il d'autres contraintes ? D'un point de vue algébrique, non. Mais d'un point de vue numérique, où l'on fait des approximations, cela peut avoir une importance. Nous allons constater cela sur un exemple. On considère le système suivant :

$$S: \begin{cases} 10^{-4}x & +y & = & 1 \\ x & +y & = & 2 \end{cases}$$

1. Résoudre à la main ce système de manière exacte.
2. On considère à présent que les calculs se font avec 3 chiffres significatifs. Résoudre le système S en choisissant $a_{1,1} = 10^{-4}$ comme pivot.
3. Toujours avec la même précision de calcul, résoudre le système S en choisissant comme premier pivot le coefficient $a_{2,1} = 1$.
4. Que constatez-vous ?

La partie de descente :

5. Rédiger une fonction `recherche_pivot(A, b, i)` qui détermine le coefficient a_{ij} le plus grand en valeur absolue parmi $a_{jj}, \dots, a_{nj}$ puis qui permute les lignes L_i et L_j de A et b. On fera attention au cas où les coefficients $a_{jj}, \dots, a_{nj}$ sont tous nuls.
6. Rédiger une fonction `elimination_bas(A, b, j)` qui effectue les éliminations successives des coefficients situés sous a_{jj} en supposant ce coefficient non nul. On effectuera en parallèle les mêmes opérations sur b.
7. En déduire une fonction `descente(A, b)` qui, par opérations élémentaires sur les lignes des matrices A et b, réalise l'étape de descente de l'algorithme du pivot de Gauss.

La partie de remontée :

8. Rédiger une fonction `elimination_haut(A, b, j)` qui effectue les éliminations successives des coefficients situés au-dessus du coefficient a_{jj}, en supposant ce coefficient non nul. On effectuera en parallèle les mêmes opérations sur b.
9. En déduire une fonction `remontee(A, b)` qui, par opérations élémentaires sur les lignes des matrices A et b réalise l'étape de remontée de l'algorithme du pivot de Gauss.

La résolution du système linéaire :

10. Rédiger une fonction `solve_diagonal(A, b)` qui prend en arguments une matrice diagonale inversible A et un vecteur b et qui retourne l'unique vecteur x solution de l'équation $Ax = b$.
11. En déduire une fonction `gauss(A, b)` qui retourne
    — l'unique solution du système $Ax = b$ si le système est de Cramer ;
    — qui affiche si le système admet une infinité de solutions ou aucune solution.

Comparaison avec `numpy.linalg` :

12. Faire une fonction `compare_temps_solve(A, b)` qui renvoie le temps d'exécution de `gauss(A, b)` et le temps d'exécution la fonction `numpy.linalg.solve(A, b)`. Pour cela, on pourra utiliser la librairie `time` et la fonction `time.time()`.

Exercice 3.4 – Stabilité de la résolution de système – On considère les trois systèmes suivants :

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}, \quad \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}, \quad \begin{pmatrix} 10 & 7 & 8.01 & 7.02 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

Résoudre numériquement ces trois systèmes. Que constatez-vous ?

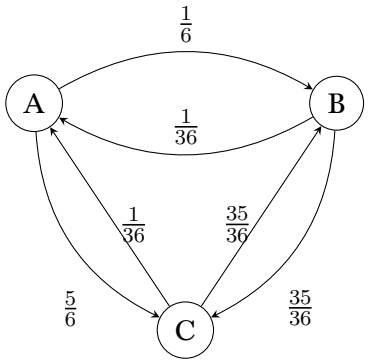
Exercice 3.5 – Calcul d'inverse –

1. Reprendre le code de l'exercice 3.3 afin de rédiger une fonction `inverse(A)` qui renvoie l'inverse de A si celle-ci existe et qui affiche "La matrice est non inversible" sinon.
2. Faire une fonction `compare_temps_inv(A)` qui renvoie le temps d'execution de `inverse(A)` et le temps d'execution la fonction `numpy.linalg.inv(A)`.

TP 4 – Matrices et géométrie

Exercice 4.1 – Matrice liée à un graphe –

Dans une équipe de football, on étudie les passes que se font trois attaquants A, B et C.
Les probabilités qu'un attaquant passe le ballon à un autre sont représentées sur le schéma suivant. Par exemple, la probabilité que l'attaquant A passe le ballon à fl'attaquant B est égale à $\frac{1}{6}$.
Au départ, l'attaquant A possède le ballon.



1. On définit les suites (a_n), (b_n) et (c_n) comme étant les probabilités que le ballon soit avec l'attaquant A, B ou C après n passes. On pose
$$X_n = \begin{pmatrix} a_n \\ b_n \\ c_n \end{pmatrix}.$$
   a) Déterminer une définition par récurrence (croisée) des suites (a_n), (b_n) et (c_n).
   b) En déduire une matrice M telle que $X_{n+1} = M \times X_n$.
2. Quelle est la probabilité que C possède le ballon après 2 passes ?
   On pose
$$P = \begin{pmatrix} 34 & \frac{71}{6} & 0 \\ -29 & 211 & 1 \\ -5 & 215 & -1 \end{pmatrix}$$
3. a) À l'aide de Python, montrer que $D = P^{-1}MP$ est diagonale.
   b) Que vaut D^n lorsque n tend vers $+\infty$.
   c) En déduire M^n lorsque n tend vers $+\infty$.
4. a) Au bout d'un temps suffisamment long, quelle est la probabilité que le ballon soit avec le joueur A, B ou C ?
   b) Cela reste-il vrai si au départ le ballon est possédé par B ou C ?

Exercice 4.2 – Triangles –

1. Écrire une fonction `scalaire(u,v)` qui calcule le produit scalaire de deux vecteurs du plan u et v.
2. Écrire une fonction `norme(u)` qui calcule la norme de du vecteur du plan u.
3. Écrire une fonction `nature_triangle(A,B,C)` qui détermine la nature du triangle $[A, B, C]$, où les points A, B et C sont des points à coéfficients entiers.
4. Écrire une fonction `nombre_triangle_rectangle(x_min,x_max,y_min,y_max)` qui détermine le nombre de triangles rectangles dont les sommets $S_i = (x_i, y_i)$ sont à coordonnées entières et tels que `x_min` $\leqslant x_i \leqslant$ `x_max` et `y_min` $\leqslant y_i \leqslant$ `y_max`.
5. Écrire une fonction `rotation(omega,theta,P)` qui renvoie l'image du point P par la rotation de centre `omega` et d'angle `theta`.

Email address: johan.leray@univ-nantes.fr

Département d'informatique – IUT de Nantes