Ingénierie numérique et simulation

Dans cette partie, nous nous intéressons à des méthodes numériques pour la résolution d'équations sur les réels, de systèmes linéaires et d'équations différentielles. D'une part, nous rappelons les méthodes du cours de mathématiques, qui sont faciles à programmer, et dont la validité peut être démontrée formellement et sans grande difficulté. D'autre part, nous expliquons comment utiliser les fonctions « clés en main » fournies par Python et ses bibliothèques adaptées, avec des exemples rencontrés en mathématiques, sciences physiques ou chimie.

L'idée pénible à garder à l'esprit est que, en calcul numérique, tout est faux !
En effet, les données prises en entrée sont des approximations des données « réelles » (parce qu'issues d'autres calculs ou de mesures physiques).

De plus, on résout des équations qui sont une approximation de la vie réelle (linéarisation d'un phénomène en physique...) et on applique pour cela des schémas qui introduisent une erreur dans le résultat. Enfin, le moindre calcul en arithmétique flottante induit des erreurs d'arrondis, on note le résultat final sur un morceau de papier, et on se trompe en copiant la deuxième décimale.

Nous montrons sous quelles conditions on peut contrôler ces erreurs, et comment choisir les paramètres des méthodes utilisées pour obtenir un résultat satisfaisant.

I - RECHERCHE DE ZERO

Dans les problèmes d'ingénierie, il faut souvent rechercher les solutions d'une équation de la forme f(x)=0, où f est une fonction à valeurs réelles. Le but est donc de trouver une solution approchée en minimisant le temps d'exécution, mais en obtenant quand même une bonne approximation. Trois méthodes classiques de recherche de zéro sont expliquées dans ce cours :

- La recherche par dichotomie
- La méthode de Newton
- La méthode de Lagrange

On considérera par la suite une fonction f continue sur un intervalle [a,b] à valeurs réelles, telle que f(a).f(b)<0. D'après le théorème des valeurs intermédiaires, la fonction f s'annule au moins une fois sur l'intervalle [a,b].

I.1. Méthode de dichotomie

Soit f est une fonction continue et admet au moins une solution sur un intervalle [a, b].

Problème :

Rechercher $\alpha \in$ [a, b] tel que f(α) = 0

Algorithme :

Initialisation : on prend pour x0 le milieu de [a, b].
La racine se trouve alors dans l'un des deux intervalles] a, x0 [ou] x0, b [ou bien elle est égale à x0.

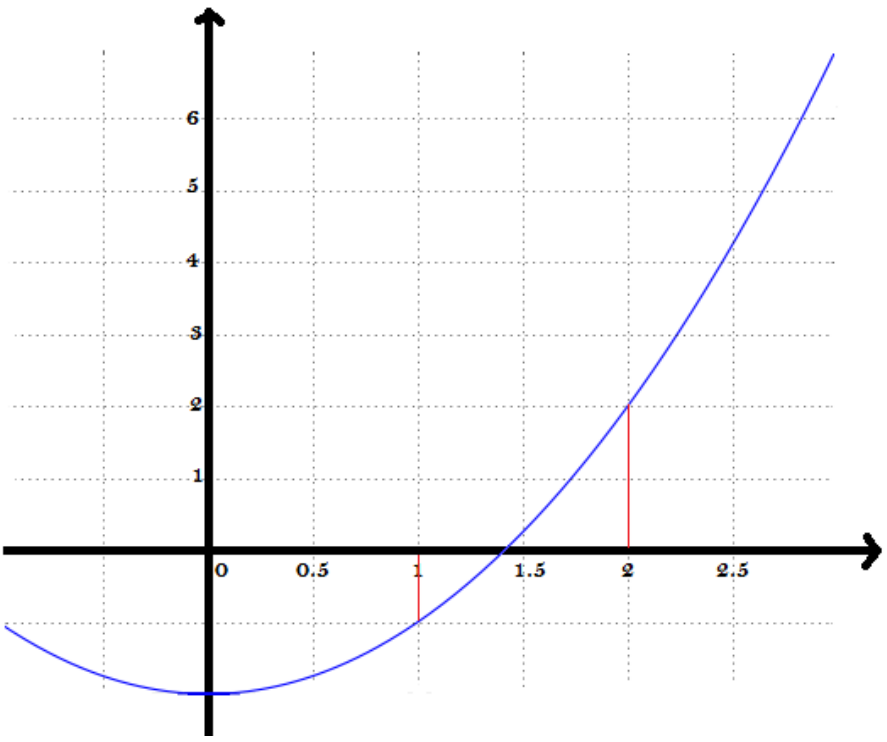
- si f(a).f(x0) < 0, alors $\alpha \in$] a, x0[. On pose a1 = a, b1 = x0.
- si f(a).f(x0) = 0, alors α = x0.
- si f(a).f(x0) > 0, alors $\alpha \in$]x0, b[. On pose a1 = x0, b1 = b.

On prend alors pour x1 le milieu de [a1, b1].

On construit ainsi une suite :
x0 = (a+b)/2, x1 = (a1 + b1)/2, . . ., xn = (an + bn)/2
telle que :
$|\alpha - xn| \leq (b - a)/2n+1$.

Etant donné une précision ε , cette méthode permet d'approcher α en un nombre prévisible d'itérations.

Exemple : On cherche une solution de f : x → x2 -2 dans l'intervalle [1, 2]



- f(1)= -1 < 0 et f(2)= 2 >0, donc l'équation f(x)=0 possède une solution dans [1 ,2]. La valeur médiane de cet intervalle est 1.5.
- f(1.5)= 0.25 >0 et f(1) = -1 <0, donc l'équation f(x)=0 possède une solution dans [1.25, 1.5].
- ...
- f(1.41430664062) ≃ 0.000263273715973 > 0 et f(1.4140625) < 0, donc l'équation f(x) = 0 possède une solution dans [1.4140625, 1.41430664062].

On peut représenter ceci, en notant $[a_n, b_n]$ le segment [c, d] après la n-ième étape.

On arrête le processus lorsque d − c a atteint une valeur correspondant à la précision demandée ε et on renvoie comme résultat $\frac{d-c}{2}$

Travail à faire :

Ecrire la fonction dicho(f,a,b,epsilon) qui recherche et renvoie un zéro par dichotomie.
f : une fonction
a et b : des réels qui délimitent l'intervalle [a,b]
epsilon : un réel indiquant la précision demandées

I.2. Méthode de Lagrange

Dans la méthode de Lagrange, on approxime le graphe d'une fonction f entre les points d'abscisses a et b par sa corde, c'est-à-dire par le segment de droite reliant les points de coordonnées (a, f(a)) et (b, f(b)). L'abscisse x_0 du point d'intersection entre la corde et l'axe des x donne une approximation d'un zéro de la fonction.

La corde a pour fonction :

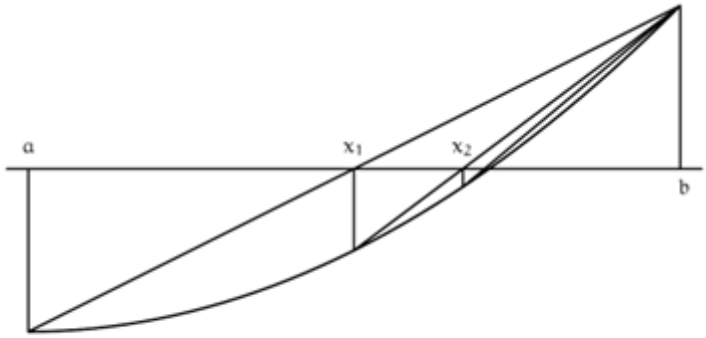
$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a).$$

La valeur de x_0 est :

$$x_0 = a - \frac{b - a}{f(b) - f(a)} \cdot f(a) = \frac{a\,f(b) - b\,f(a)}{f(b) - f(a)}.$$

Il suffit ensuite de réitérer le procédé, en considérant l'intervalle [x0, b] si f(x0) est du même signe que f(a), et l'intervalle [a,x0] si non. La suite ainsi construite converge vers un zéro de f dans l'intervalle [a,b].

Cette méthode est aussi appelée méthode des sécantes et est illustrée par la figure suivante :



Méthode de Lagrange

Travail à faire :

Ecrire la fonction lagrange(f,a,b,epsilon) qui recherche et renvoie un zéro par la méthode de Lagrange.
f : une fonction
a et b : des réels qui délimitent l'intervalle [a,b]
epsilon : un réel indiquant la précision demandées

I.3. Méthode de Newton

On l'appelle aussi méthode de la tangente, la méthode de Newton est une méthode de résolution de l'équation $f(x) = 0$. Si x_0 est proche de la racine r on peut faire un développement de Taylor à l'ordre 1 de la fonction f en x_0 :

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + O((x - x_0)^2)$$

Pour trouver une valeur approchée de r, on ne garde que la partie linéaire du développement, on résout :

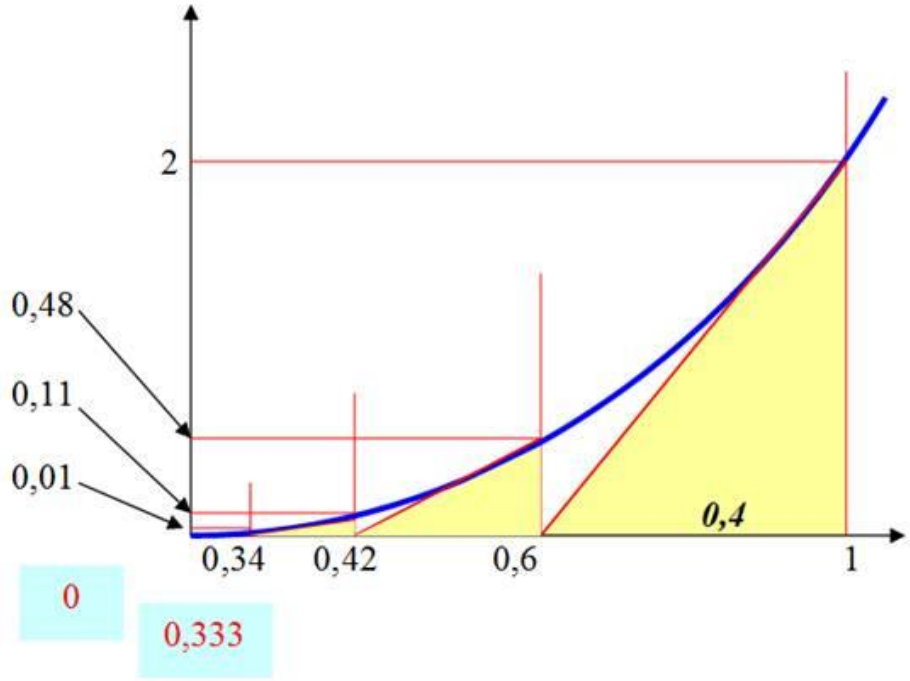
$$f(r) = 0 \approx f(x_0) + (r - x_0)f'(x_0)$$

donc (si $f'(x_0) \neq 0$) :

$$r \approx x_0 - \frac{f(x0)}{f\prime(x0)}$$

Graphiquement, cela revient à tracer la tangente à la courbe représentative de f et à chercher où elle coupe l'axe des x. On considère donc la suite récurrente définie par une valeur u_0 proche de la racine et par la relation :

$$U_{n+1} = U_n - \frac{f(Un)}{f\prime(Un)}$$



Il y a deux théorèmes importants, l'un d'eux prouve que si u_0 est "assez proche'' de *r* alors la suite u_n converge vers *r*, malheureusement il est difficile de savoir en pratique si on est ``assez proche'' de u_0 pour que ce théorème s'applique. Le second théorème donne un critère pratique facile à vérifier qui assure la convergence, il utilise les propriétés de convexité de la fonction.

Théorème : *Soit f une fonction de classe C^2 (2 fois continument dérivable) sur un intervalle fermé I. Soit r une racine simple de f située à l'intérieur de I (telle que f (r) = 0 et f '(r) ≠ 0). Alors il existe $\varepsilon > 0$ tel que la suite définie par :*

$$u_{n+1} = u_n - \frac{f(Un)}{f'(Un)}, \quad |u_n - r| \leq \varepsilon$$

converge vers r.

Définition : (convexité)
Une fonction f continument dérivable sur un intervalle I de $\mathbb{R}$ est dite convexe si son graphe est au-dessus de la tangente en tout point de I.

Il existe un critère simple permettant de savoir si une fonction de classe C^2 est convexe :
Théorème : *Si f est C^2 et $f'' \geq 0$ sur I alors f est convexe.*

Travail à faire :

Ecrire la fonction newton(f, fp, x0,epsilon) qui recherche et renvoie un zéro par la méthode de Newton.
f : une fonction
fp : la dérivé de la fonction f
x0 : le point de départ
epsilon : un réel indiquant la précision demandées

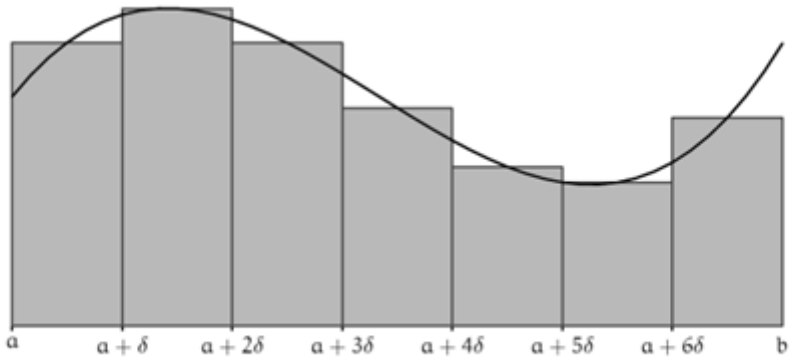
II -INTEGRATION NUMERIQUE

Soit f une fonction à valeurs réelles, continue par morceaux sur un intervalle [a, b].
On souhaite calculer une valeur approchée de l'intégrale $\int_a^b f(t)dt$

II.1. Méthode des rectangles

La méthode des rectangles consiste à approximer la fonction f par une fonction en escalier. On considère un entier n et un pas de subdivision $\frac{b-a}{n}$. Pour tout entier k de $[\![0, n]\!]$, on pose $a_k = a + k \cdot \frac{b-a}{n}$. Sur l'intervalle $[a_k, a_{k+1}]$, on approxime f par la fonction constante égale à $f(\frac{a_k + a_{k+1}}{2})$ (voir figure 4.3).
On prend, comme valeur approchée de l'intégrale de f sur $[a, b]$, l'intégrale de la fonction en escalier ainsi construite, c'est-à-dire la somme des aires des rectangles (les rectangles ont tous une base de longueur $\frac{(b-a)}{n}$).

$$\int_a^b f(t)\,dt \simeq \frac{(b-a)}{n} \sum_{k=0}^{n-1} f\left(a + \frac{(b-a)}{2n} + k\frac{(b-a)}{n}\right)$$



Méthode des rectangles

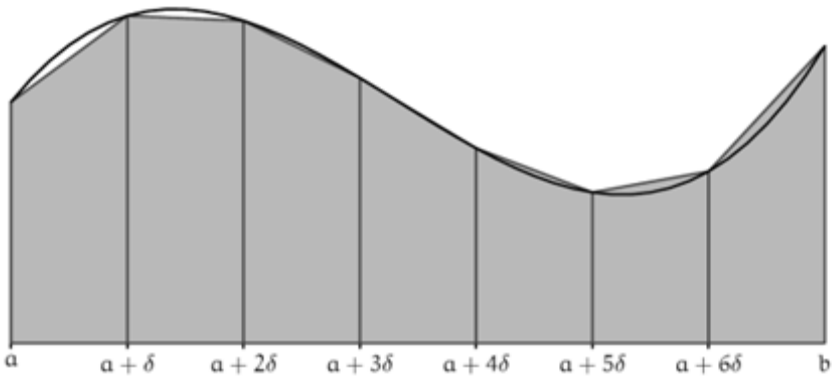
Travail à faire :

Ecrire la fonction rectangles(f, a, b, n) qui calcule et renvoie $\int_a^b f(t)dt$ par la subdivision de l'intervalle [a,b] en n intervalles.
f : une fonction
a et b : Les bornes de l'intervalle
n : le nombre de subdivision

II.2. Méthode des trapèzes

La méthode des trapèzes consiste à approximer la fonction f par une fonction continue affine par morceaux. Ceux-ci coïncident avec la fonction f aux points de la subdivision (voir figure 4.4). En notant $a_k = a + k \cdot \frac{b-a}{n}$ les points de la subdivision du segment $[a, b]$, l'aire de chaque trapèze est égale à $\frac{(b-a)}{n} \cdot \frac{(f(a_k)+f(a_{k+1}))}{2}$, c'est-à-dire, en sommant,

$$\int_a^b f(t)\,dt \simeq \frac{b-a}{n} \cdot \left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a_k) \right).$$



Méthode des trapèzes

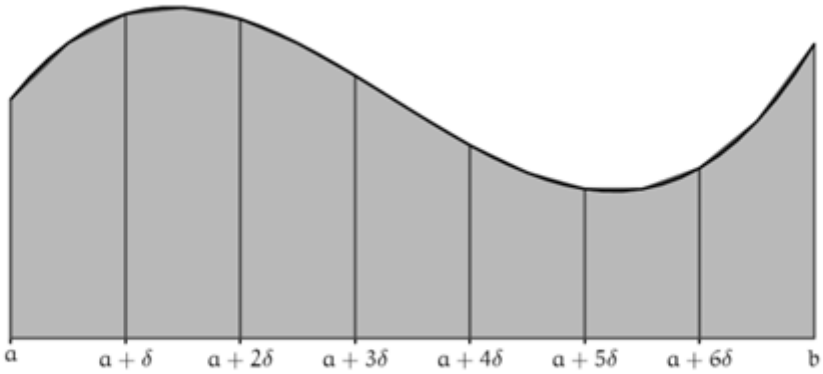
Travail à faire :

Ecrire la fonction trapezes(f, a, b, n) qui calcule et renvoie $\int_a^b f(t)dt$ par la méthode des trapèzes.
f : une fonction
a et b : Les bornes de l'intervalle
n : le nombre de subdivision

II.3. Méthode de Simpson

On note toujours $a_k = a + k \cdot \frac{b-a}{n}$. La méthode de Simpson consiste à approximer le graphe de la fonction sur chaque segment $[a + k \cdot \text{pas}, a + (k+1) \cdot \text{pas}]$ par un arc de parabole qui coïncide avec le graphe de la fonction aux points d'abscisses a_k, a_{k+1} et $\frac{a_k + a_{k+1}}{2}$. En notant f_k la fonction parabolique obtenue pour le segment $[a_k, a_{k+1}]$ et $m_k = \frac{a_k + a_{k+1}}{2}$, on peut montrer que l'aire sous l'arc de la parabole ainsi construite est égale à

$$\int_{a_k}^{a_{k+1}} f_k(t)\,dt = \frac{a_{k+1} - a_k}{6}\big(f(a_k) + 4\,f(m_k) + f(a_{k+1})\big).$$



Méthode de Simpson

Travail à faire :

Ecrire la fonction simpson(f, a, b, n) qui calcule et renvoie $\int_a^b f(t)dt$ la méthode de Simpson.
f : une fonction
a et b : Les bornes de l'intervalle
n : le nombre de subdivision

III - PIVOT DE GAUSS ET RESOLUTION DES SYSTEMES

III.1. Définition

La méthode du « pivot de Gauss », ou « élimination de Gauss-Jordan », est un algorithme efficace permettant de résoudre — lorsque c'est possible — un système d'équations linéaires.

L'algorithme du pivot de Gauss sait résoudre des systèmes à **n** équations et **p** inconnues. Pour simplifier, on se placera pour l'essentiel de l'exposé dans le cas où **n=p** avec l'hypothèse supplémentaire de l'existence d'une et une seule solution. On parle de système de ***Cramer***.

Numériquement, l'implémentation sur ordinateur de cet algorithme donne généralement de mauvais résultats (même s'il est rapide) : les erreurs d'arrondi se cumulent et faussent généralement la solution. Néanmoins, il n'utilise que des additions et multiplications, ce qui en fait le meilleur du point de vue du rapport simplicité/efficacité disponible en calcul manuel.

1. Principe de la méthode

L'objectif du pivot de Gauss est de ramener le système d'équations linéaires à un système étagé (dont on sait qu'il est soluble), c'est-à-dire de la forme « triangulaire » suivante :

$$\begin{cases} 3x + 5y - 1z &= 3 \\ 5y - 3z &= 4 \\ 7z &= 2 \end{cases}$$

Il suffit en effet d'en déduire z avec la dernière ligne, de le remplacer par sa valeur dans la ligne au-dessus, d'en déduire y, de le remplacer par sa valeur dans la ligne au-dessus, d'en déduire x... et c'est fini ! La résolution est complètement machinale une fois que le système est mis sous cette forme.

Voyons maintenant comment s'y prendre :

Pour décrire l'algorithme, nous allons prendre un exemple, plutôt qu'une définition formelle :

$$\begin{cases} x - y + 2z = 5 \\ 3x + 2y + z = 10 \\ 2x - 3y - 2z = -10 \end{cases}$$

- **Étape 1** : choix du pivot : on choisit un « pivot », c'est-à-dire l'un des monômes du système. Le premier pivot est le premier monôme de la première ligne, le second est le second monôme de la seconde ligne, etc. On commence donc avec « x » pour pivot.

- **Étape 2** : élimination : on soustrait aux lignes suivantes la ligne du pivot un nombre suffisant de fois pour que tous les termes en « x » (1er pivot), en « y » (2e pivot) etc. s'annulent. Dans notre exemple, la première étape :

$$\begin{cases} x - y + 2z = 5 \\ 3x + 2y + z = 10 \\ 2x - 3y - 2z = -10 \end{cases}$$

Il faut soustraire 3 fois la première ligne (ligne du pivot) à la seconde, et 2 fois la première ligne à la troisième. Cela donne :

$$\begin{cases} x - y + 2z = 5 \\ 0 + 5y - 5z = -5 \\ 0 - y - 6z = -20 \end{cases}$$

Retour à l'étape 1 avec le pivot suivant.

- **Fin de l'algorithme** : l'algorithme se termine :
  - lorsqu'il a atteint le n-ième coefficient de la n-ième ligne (le système admet une unique solution), ou
  - lorsqu'il atteint un pivot nul. (Le système n'admet pas une unique solution.)

Dans les cas moins favorables, on peut rencontrer en cours de résolution d'un système ces trois problèmes :

- Le pivot n'est pas là où on veut : si, à la première étape, le coefficient en x de la première ligne est nul, on peut échanger la première équation avec la deuxième ou la troisième. De même, si à la seconde étape, le coefficient en y (futur pivot) est nul, on peut échanger la deuxième équation avec la troisième, mais pas la première (se souvenir qu'on veut arriver à un système triangulaire, il ne faut pas faire réapparaître x dans les deux dernières équations).
- Il n'y a plus de pivot en une variable : si tous les coefficients en x sont nuls (c'est rare : cela revient à dire que x n'apparaît pas dans le système...), on peut prendre y ou z comme première variable. De même, si après la première étape, y n'apparaît ni dans la deuxième ni dans la troisième équation, on peut prendre z comme deuxième inconnue.

- Il n'y a plus de pivot : cela signifie que les membres de gauche des équations restantes sont nuls. Selon que les membres de droite correspondants sont nuls ou pas, ces équations vont disparaître ou bien rendre le système incompatible.

Les deux dernières situations ne se produiront pas sur un système de Cramer.

2. Le formalisme matriciel

Pour simplifier l'implémentation de l'algorithme du pivot de Gauss, on va décrire rapidement le pivot de Gauss dans le cadre matriciel.

Un système linéaire peut être vu comme une équation matricielle : le système de l'exemple précédent se traduit par AX = Y, avec :

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & 1 \\ 2 & -3 & -2 \end{pmatrix}, X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ et } Y = \begin{pmatrix} 5 \\ 10 \\ -10 \end{pmatrix}$$

Il est équivalent au système UX = Y', avec :

$$U = \begin{pmatrix} 1 & -1 & 2 \\ 0 & 5 & -5 \\ 0 & 0 & -35 \end{pmatrix} \quad \text{et} \quad Y' = \begin{pmatrix} 5 \\ -5 \\ -105 \end{pmatrix}$$

Maintenant il suffit de calculer z et remonter pour calculer la valeur de y puis de x.

3. Algorithme

On commence par réfléchir aux bons outils, à savoir les fonctions et programmes auxiliaires à l'aide desquels l'écriture du programme principal sera quasiment une traduction en anglais de l'algorithme ! Il faudra déléguer les opérations suivantes :

- <u>La recherche d'un pivot :</u>

On commence par écrire une fonction « **def chercher_pivot (M, i)** » qui prend en paramètre une matrice M et l'indice d'une ligne de la matrice M et renvoie un indice de ligne j>=i tel que $|M_{ji}|$ soit maximal.

- <u>Les échanges de lignes :</u>

On écrit maintenant une fonction « **def echange_lignes(M, i, j)** » qui permet d'échanger la $i^{ème}$ et la $j^{ème}$ ligne de la matrice M.

- Les transvections

On écrit une fonction « **def transvection_ligne(M, i, j, mu)** » qui permet d'appliquer une transvection de la j ème ligne à la i ème ligne avec le scalaire mu.
Lj = Li + mu*Lj

- Copie d'une matrice :

On écrit une fonction « **def copie_matrice(M)**» qui renvoie une copie de la matrice M.

- Recoller les morceaux :

Grâce à tous ces fonctions, l'écriture du programme Python qui fait la résolution d'un système linéaire devient comme prévu un simple exercice de traduction. On commence par faire une copie de la matrice fournie en entrée puis on applique l'algorithme décrit en dessus.

On écrit une fonction « **def resolution(A0, Y0)**» qui prend en paramètre la matrice A0 et la matrice Y0 et retourne le vecteur X contenant la solution du système.

IV - Résolution numériques des systèmes différentielles (Méthode d'Euler)

Hormis quelques cas d'école simples, on ne sait pas déterminer d'expressions analytiques pour les solutions d'équations différentielles. Le but de ce chapitre est de présenter la méthode d'Euler, qui sert à en calculer des approximations.

De nombreux phénomènes physiques se modélisent à l'aide d'équations différentielles pour lesquelles on ne dispose pas de solutions analytiques : un pendule amorti nous amène à étudier l'équation $\ddot{\theta} = -k_1 \sin\theta - k_2\dot{\theta}$, les problèmes de cinétique chimique conduisent à des systèmes différentiels non linéaires décrivant les évolutions de différents réactifs au cours du temps, et les phénomènes qu'on observe en mécanique des fluides sont en partie décrits par les équations aux dérivées partielles non linéaires de type Navier-Stokes.
En mathématiques, ces équations ont leur intérêt propre et étudier le comportement qualitatif de solutions est nettement plus aisé si on peut visualiser une approximation raisonnable de celles-ci.

1. Principe de la méthode d'Euler

Le théorème de Cauchy-Lipschitz assure que sous des conditions raisonnables, il existe une unique application y de classe C^1 sur [a, b] dont la valeur est imposée en a et qui vérifie une équation de la forme y'(t) = F(t, y(t)) pour tout t de [a, b]. L'objet des schémas numériques est d'obtenir des approximations de ces solutions dont la théorie donne l'existence de façon non constructive. En pratique, on tente en général d'approcher **y** en un certain nombre de points répartis sur l'intervalle [a, b].

Il s'agit de calculer une approximation y_k des $y(t_k)$, avec $t_k = a + kh$, où $h = (b – a)/n$ est un **pas** qu'il conviendra d'ajuster. De façon très simple, si on écrit :

$$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} y'(u)du = \int_{t_k}^{t_{k+1}} F(u, y(u))du \simeq hF(t_k, y(t_k)),$$

alors on obtient la méthode d'Euler : les approximations sont calculées de proche en proche via la formule suivante : $y_{k+1} = y_k + hF(t_k, y_k)$. On initialise bien entendu avec $y_0 = y(a)$, qui sera la seule valeur « exacte » calculée.

Graphiquement, cela revient à faire des approximations successives de courbes par des tangentes.

2. Choix du pas

Le choix du pas est une étape obligée lors de la mise en place d'une méthode numérique de résolution.

- Si on choisit ce pas trop petit, le temps de calcul sera élevé.

- Si au contraire h est trop grand, l'erreur de consistance sera trop importante.

Il s'agit donc, comme toujours en calcul scientifique, de faire un compromis entre le temps de calcul et la qualité de l'approximation, et ce n'est pas toujours simple.

3. Implémentation de la méthode

On demande d'écrire la fonction « **def euler(F, a, b, y0, h) »** va prendre en entrée une fonction F, les bornes a et b de l'intervalle d'étude, la condition initiale y_0 et le pas **h**. Plus précisément : avec ces données, la fonction va déterminer les approximations de la solution de y'(t) = F(t, y(t)) avec la condition initiale y(a) = y_0, en rendant un tableau de temps et un tableau de valeurs approchées par la méthode d'Euler. Les temps sont les a + kh majorés par b. Ces tableaux sont construits à l'aide de listes auxquelles on adjoint les nouveaux termes calculés.

V - Présentation des bibliothèques

1. Notion des modules

Un module est un fichier ayant pour extension .py contenant des définitions de classes, constantes et fonctions. Tout programmeur python peut réaliser un module.
Exemple de modules : math, random, time, numpy, scypi, mathplotlib ...

2. Importation des modules dans un programme

Pour importer un module dans un programme il y a deux méthodes :

Méthode 1 : Permet d'importer tous les éléments du module

```
import nom_module
```

Exemple :

```
import math
a = math.sqrt(2)   # Pour calculer la racine carrée de 2
```

Méthode 2 : Permet d'importer une fonction du module

```
from nom_module import fonction
# pour plusieurs fonctions
from nom_module import fonction1, fonction2
```

Exemple :

```
from math import sqrt
a = sqrt(2)
```

3. Modules de simulation numérique

Dans cette troisième partie du cours on va étudier trois nouveaux modules dans cette partie :
numpy , **scipy** , **matplotlib**

NumPy :

Pour le calcul scientifique, on est souvent amené à manipuler de grands ensembles de nombres. Les listes Python sont des conteneurs hétérogènes ne sont pas adaptées pour cela car :

- les opérations arithmétiques (addition, multiplication) ne sont pas réalisable facilement (car le + pour les listes est une concaténation)

- une liste Python ne stocke pas son contenu de façon efficace (i.e. contiguë) en mémoire.

Pour manipuler des tableaux ou les matrices de nombres, il y a le module **numpy**.

SciPy :

Le module **scipy** regroupe un certain nombre de sous modules qui sont autant de boîtes à outils de routines courantes de calcul numérique, regroupées par fonction : fonctions spéciales, interpolation, intégration, optimisation, traitement d'images.

MatPlotLib :

Matplotlib est une bibliothèque en Python très utilisée pour tracer des graphiques en deux et trois dimensions.
En combinaison avec les bibliothèques scientifiques NumPy ou SciPy , nous obtenons un outil de prototypage très pratique.

4. Utilisation du module numpy

Fonction	Description
array(L)	crée un tableau à partir d'une liste L.
arange(a,b,k)	crée un vecteur dont les coefficients sont les a+k. entre a (inclu) et b (exclu).
linspace(a,b,n)	crée un vecteur de n valeurs régulièrement espacées entre a et b (inclus).
zeros(p)	crée un tableau de taille p rempli de zéros.
zeros((p,q))	crée une matrice de taille (p,q) rempli de zéros
ones(p)	crée un tableau de taille p rempli de uns
ones((p,q))	crée une matrice de taille (p,q) rempli de uns
shape(M)	pour obtenir la taille d'un tableau (= type d'une matrice)
reshape(M, (dimensions))	pour redimensionner un tableau

dot(M, N)	pour effectuer un produit matriciel de 2 matrices
vdot(v1, v2)	pour effectuer un produit scalaire de 2 "vecteurs"
transpose(M)	pour transposer une matrice
rank(M)	rang d'une matrice
mean(T)	valeur moyenne d'un tableau
sum(T)	La somme des valeurs d'un tableau
linalg.inv(M)	inversion d'une matrice
linalg.det(M)	déterminant d'une matrice
linalg.solve(A,b)	résolution du système linéaire A:X = b
roots([a,b,c,d])	détermine les racines d'un polynôme donné par la liste de ses coefficients $aX^3 + bX^2 + cX + d = 0$. Elle fournit même les racines complexes, le nombre imaginaire i étant noté j.

5. Utilisation du module scipy

optimize.bisect(f,a,b)	Cherche un zéro d'une fonction dans un intervalle [a,b] par la méthode de dochotomie.
optimize.newton(f,x0 ,fp=None)	La méthode de Newton est programmée dans scipy.optimize.newton. Il est à noter que si on ne donne pas la dérivée, c'est en fait la méthode de la sécante qui est appliquée. Dans le cas opposé où on lui fournit f' et f'', c'est la méthode de Halley qui sera en fait mise en oeuvre.
integrate.quad(f, a, b)	Intégrer une fonction f sur un intervalle [a,b]

integrate.odeint(f,y0, t) t : liste de valeur de t pour lesquelles on calcule y(t)	Résolution numérique des équations différentielles. y'(t)= f(t, y(t)) y(0)=y0

6. Quelques fonctions de matplotlib

Pour le simple tracé de courbes nous n'utiliserons que le sous-module pyplot, importé, avec alias, à l'aide de l'instruction:
import matplotlib.pyplot as pp

Pour plus de documentation : http://www.matplotlib.org.

Les fonctions essentielles de pyplot sont :

1. plot() pour le tracé de points, de courbes, et
2. show() pour afficher le graphique créé.

Utiliser plot() avec :

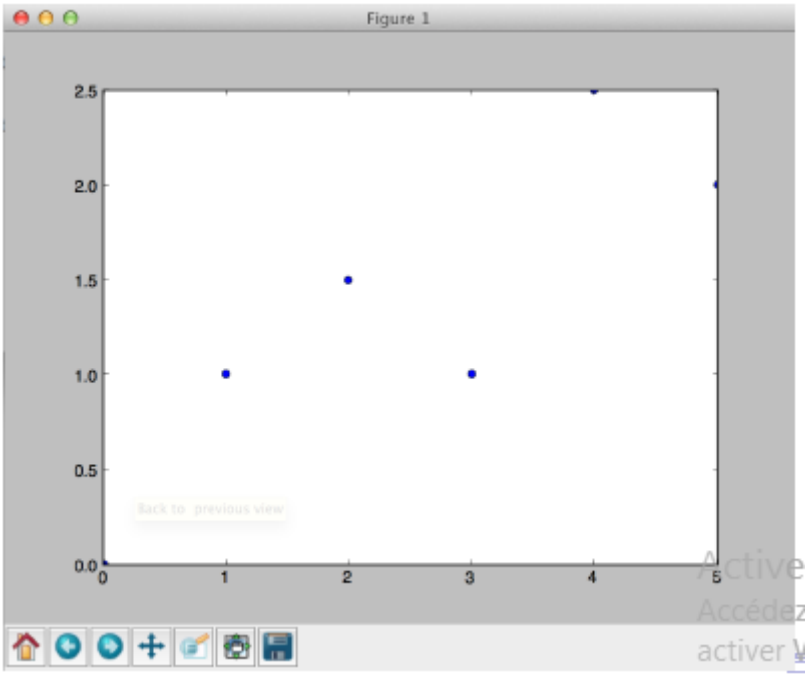
1. en 1er argument la liste des abscisses,
2. en 2eme argument la liste des ordonnées,
3. en 3eme argument (optionnel) le motif des points :
   - '.' pour un petit point,
   - 'o' pour un gros point,
   - '+' pour une croix,
   - '*' pour une étoile,
   - '-' points reliés par des segments
   - '--' points reliés par des segments en pointillés
   - '-o' gros points reliés par des segments (on peut combiner les options)
   - 'b', 'r', 'g', 'y' pour de la couleur (bleu, rouge, vert, jaune, etc...)
   - consulter http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot pour plus d'options.

Exemples :

- Exemple pour le tracé d'un nuage de points :

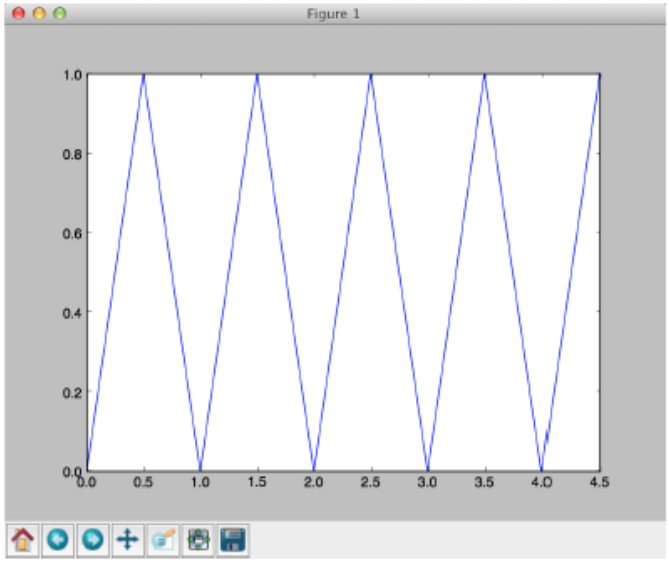
```
>>> import matplotlib.pyplot as pp
>>> abs = [0, 1, 2, 3, 4, 5]
>>> ord = [0, 1, 1.5, 1, 2.5, 2]
>>> pp.plot(abs, ord, 'o')
[<matplotlib.lines.Line2D object at 0x10c6610d0>]
>>> pp.show()
```

produit un graphique (au format .png) :



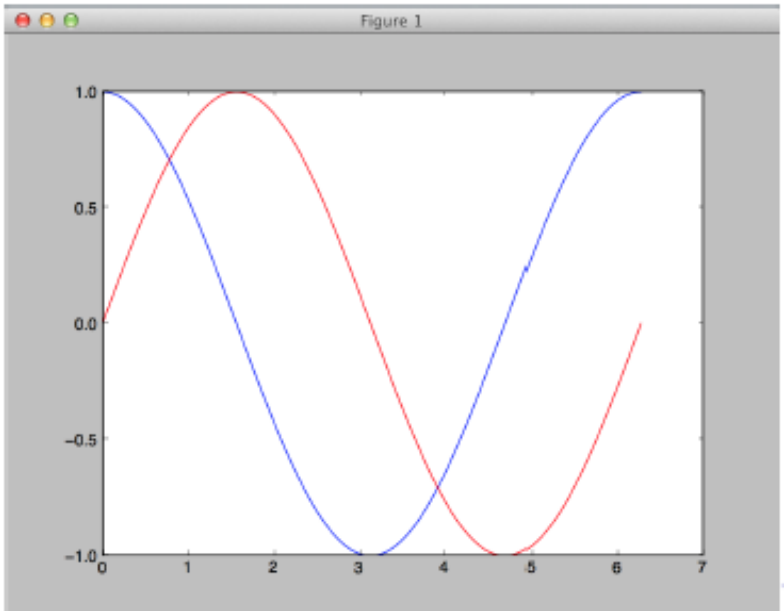
- Exemple pour le tracé d'une ligne brisée :

```
import matplotlib.pyplot as pp
abs = [n/2. for n in range(10)]
ord = [n % 2 for n in range(10)]
pp.plot(abs,ord,'-b')
pp.show()
```



- Exemple pour le tracé de courbes représentatives des fonctions réelles:

```
import matplotlib.pyplot as pp
import numpy as np # pour linspace() et les fonctions mathématiques
X = np.linspace(0, 2*np.pi, 256) # X = 256 pts régulièrement espacées
Ycos = np.cos(X) # image directe de X par cos
Ysin = np.sin(X) # image directe de X par sin
pp.plot(X,Ycos,'b') # trace de la courbe de cos en bleu
pp.plot(X,Ysin,'r') # trace de la courbe de sin en rouge
pp.show()
```



- On améliore le tracé en remplissant quelques options :

```
import matplotlib.pyplot as pp
import numpy as np # pour linspace() et les fonctions mathématiques
pp.plot(X, Ycos, 'b', X, Ysin, 'r') # Tracée simultanée des 2 courbes
pp.grid(True) # Affiche la grille
pp.legend(('cos','sin'), 'upper right', shadow = True) # Légende
pp.xlabel('axe des x') # Label de l'axe des abscisses
pp.ylabel('axe des y') # Label de l'axe des ordonnées
pp.title('Fonctions cosinus et sinus') # Titre
pp.show()
```

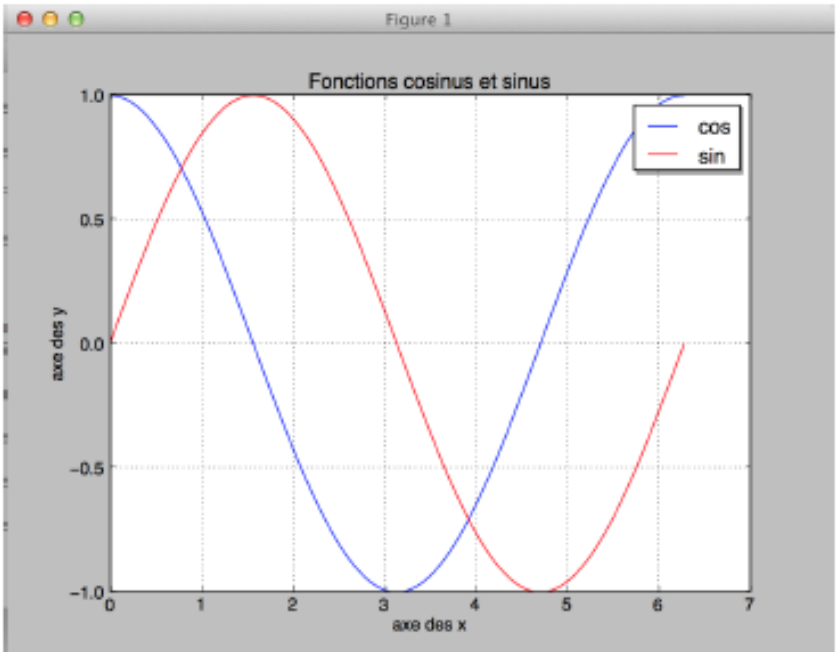


Figure 1
Fonctions cosinus et sinus
cos
sin
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6
7
axe des y
axe des x