Introduction à l'analyse numérique
Illustrée sous Python

Laurent Kaczmarek

PCSI^2 2013-2014
Lycée Louis Le Grand

Février-mars 2014

I
Introduction

Pourquoi l'analyse numérique ?

- Nous avons donné ici ou là dans le cours de Mathématiques des solutions explicites à certaines équations différentielles, scalaires et des calculs de primitives.
- Toutefois, dans la plupart des cas, des théorèmes nous assurent l'existence et l'unicité d'une solution mais il n'est pas possible de l'expliciter (ceci peut d'ailleurs être parfois démontré).
- L'analyse numérique est la branche des Mathématiques qui a pour objectif de développer des méthodes de résolution approchée à des problèmes complexes.
- Ce contexte oblige à tenir compte de la théorie mais aussi de la pratique : certains algorithmes valables en théorie vont s'avérer inefficaces en pratique car accumulant les erreurs d'arrondis au point de diverger.

Demandez le programme !

Cette introduction à l'analyse numérique consistera en une étude élémentaire des méthodes suivantes :

- Le calcul intégral approché : méthodes des rectangles et des trapèzes.
- Résolution approchée de $f(x) = 0$: méthodes de bissection (dichotomie) et de Newton.
- Résolution approchée d'une équation différentielle : méthode d'Euler.
- Système de Cramer : méthode du pivot de Gauss.

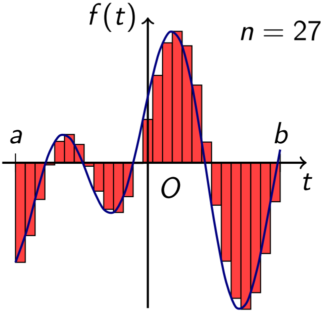
Efficacité d'une méthode numérique

- La complexité d'une méthode d'approximation se mesure en termes de vitesse de convergence. Pour comparer deux méthodes, on calculera l'erreur commise dans le pire des cas.

II
Calcul intégral approché

Méthode des rectangles

- Cadre : $f : [a, b] \to \mathbb{R}$ de classe $\mathscr{C}^1$.
- Principe :



Approximation de

$$\int_{[a,b]} f$$

par une somme de Riemann

$$\frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a + \frac{k(b-a)}{n}\right)$$

- Erreur pour f de classe $\mathscr{C}^1$ (preuve à connaître) :

$$\int_a^b f(t)\mathrm{d}t - \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a + \frac{k(b-a)}{n}\right) = O\left(\frac{1}{n}\right)$$

Méthode des rectangles (suite)

- ```python
  def rectangles(f,a,b,n):
      pas,s=(b-a)/n,0
      for i in range(n):
          s=s+f(a+k*pas)
      return(pas*s)
  ```

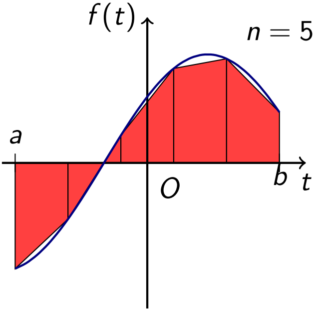
- Exemple : calcul approché de $\int_0^1 \exp\left(-x^2\right) dx$

  ```
  >>> f=lambda x:exp(-x**2)
  >>> rectangles(f,0,1,1000)
  0.7471401317785985
  ```

Méthode des Trapèzes

- Cadre : $f : [a, b] \to \mathbb{R}$ de classe $\mathscr{C}^2$.
- Principe : approximation de $\int_{[a,b]} f$ par une somme d'aires de trapèzes.
  ```

Méthode des Trapèzes (suite)



- Formule :

$$\frac{b-a}{2n}\sum_{k=0}^{n-1}(f(a_k)+f(a_{k+1})) = \frac{b-a}{2n}(f(a)+f(b)) + \frac{b-a}{n}\sum_{k=1}^{n-1}f(a_k)$$

- Erreur pour f de classe $\mathscr{C}^2$:

$$\int_a^b f(t)\mathrm{d}t - \frac{b-a}{2n}\sum_{k=0}^{n-1}(f(a_k)+f(a_{k+1})) = O\left(\frac{1}{n^2}\right)$$

Méthode des Trapèzes (suite)

- ```python
  def trapezes(f,a,b,n):
      pas,s=(b-a)/n,0
      for k in range(1,n):
          s=s+f(a+k*pas)
      return(pas*s+pas*(f(a)+f(b))/2)
  ```

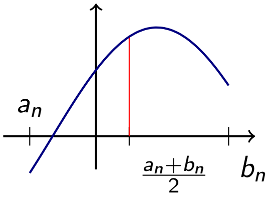
- Exemple : calcul approché de $\int_0^1 \exp\left(-x^2\right) dx$

  ```
  >>> f=lambda x:exp(-x**2)
  >>> trapezes(f,0,1,1000)
  0.74682407149918406
  ```
  ```

III
Résolution approchée d'une équation scalaire

Méthode de bissection (dichotomie)

- Cadre : $f : [a, b] \to \mathbb{R}$ $\mathscr{C}^0$, $f(a) < 0$ et $f(b) > 0$.
- Principe : à partir de $[a_0, b_0] := [a, b]$, on construit une suite de segments emboîtés $[a_n, b_n]$ de longueur $(b-a)/2^n$ tels que $f(a_n) \leqslant 0$ et $f(b_n) \geqslant 0$, $\forall n \in \mathbb{N}$.



Si $f\left(\dfrac{a_n+b_n}{2}\right) \leqslant 0$, on pose

$$a_{n+1} = \frac{a_n+b_n}{2}, \quad b_{n+1} := b_n$$

Si $f\left(\dfrac{a_n+b_n}{2}\right) > 0$, on pose

$$b_{n+1} = \frac{a_n+b_n}{2}, \quad a_{n+1} := a_n$$

Les deux suites (a_n) et (b_n) convergent vers un zéro c de la fonction f.

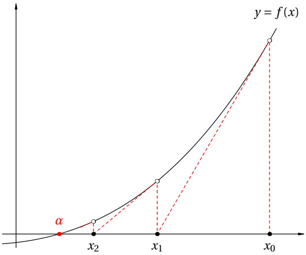
Méthode de bissection (suite)

- $\forall n \in \mathbb{N}, \quad a_n \leqslant c \leqslant b_n, \quad |a_n - c| \leqslant \dfrac{b-a}{2^n}.$
-
```
def bissection(f,a,b,eps):
    an,bn=a,b
    while bn-an>eps:
        if f((an+bn)/2)<0:
            an=(an+bn)/2
        else:
            bn=(an+bn)/2
    return(an)
```
- Exemple : résolution de $x^3 + x - 1 = 0$ sur $[0, 1]$.

```
>>> f=lambda x:x**3+x-1
>>> bissection(f,0,1,0.0001)
0.68231201171875
```

Méthode de Newton

- Cadre : $f : [a, b] \to \mathbb{R}$ $\mathscr{C}^1$ telle que $f' > 0$, $f'' > 0$, $f(a) < 0$ et $f(b) > 0$.
- Principe :



On itère à partir de x_0 tel que $f(x_0) > 0$ le schéma suivant :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- On adapte cette méthode aux cas similaires : f est décroissante avec $f(a) > 0$ et $f(b) < 0$, etc.

Méthode de Newton (suite)

- ```
  def newton(f,df,x0,eps):
      x=x0
      while f(x-eps)>0:
          x=x-f(x)/df(x)
      return(x)
  ```

- Exemple : résolution de $x^3 + x - 1 = 0$ sur $[0, 1]$.

  ```
  >>> f=lambda x:x**3+x-1
  >>> df=lambda x:3x**2+1
  >>> newton(f,df,0.8,0.0001)
  0.6824270546639579
  ```
  ```

Comparaison des méthodes

- L'erreur (en valeur absolue) commise dans la méthode de bissection est dite géométrique :

$$\varepsilon_{n+1} \leqslant \frac{1}{2}\varepsilon_n$$

  Et il existe des fonctions pour lesquelles $\varepsilon_n \sim \dfrac{\alpha}{2^n}$, $\alpha \in \mathbb{R}^*$.

- L'erreur (en valeur absolue) commise dans la méthode de bissection de Newton dite quadratique :

$$\varepsilon_{n+1} \leqslant k\varepsilon_n^2$$

  Et il existe des fonctions pour lesquelles $\varepsilon_n \sim \lambda^{2^n}$, $\lambda \in \mathbb{R}^*$.

- La méthode de Newton est plus efficace que la méthode de bissection.

IV
Résolution approchée d'une équation différentielle scalaire

Méthode d'Euler

- Cadre : on recherche une approximation sur $[a, b]$ de l'unique solution de l'équation différentielle $y' = f(x, y)$ vérifiant la condition initiale $y(a) = \alpha$.
- Soient $n \in \mathbb{N}^*$, $p := (b-a)/n$ puis $(x_0, y_0) := (a, \alpha)$.
- La solution y vérifie

  $$y(x_0 + p) \simeq y_0 + py'(x_0) = y_0 + pf(x_0, y_0)$$

  on choisit $(x_1, y_1) := (x_0 + p, y_0 + pf(x_0, y_0))$ pour approximation de $(x_0 + p, y(x_0 + p)) = (x_1, y(x_1))$. On continue :

  $$y(x_1 + p) \simeq y(x_1) + py'(x_1) \simeq y_1 + pf(x_1, y_1)$$

  on choisit $(x_2, y_2) := (x_1 + p, y_1 + pf(x_1, y_1))$ pour approximation de $(x_1 + p, y(x_1 + p)) = (x_2, y(x_2))$. Et ainsi de suite.

Méthode d'Euler (suite)

- Le schéma de cette méthode est donc :

$$x_k := x_{k-1} + p, \; y_k := y_{k-1} + pf(x_k, y_k)$$

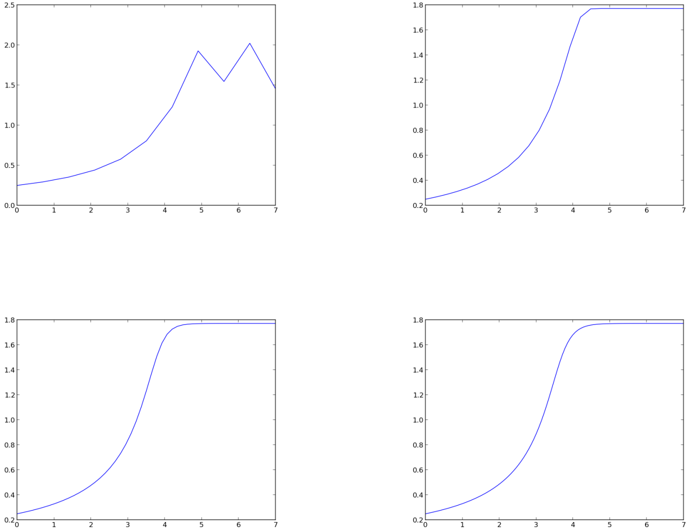
  La ligne brisée $M_0 M_1 \dots M_n$ avec $M_k(x_k, y_k)$, est une approximation du graphe de y sur $[a, b]$ appelée polygône d'Euler d'ordre n.
- On peut s'attendre à ce que cette approximation soit bonne lorsque n est grand mais ce n'est pas toujours le cas. Cette question est très délicate, les hypothèses sur f assurant l'existence et l'unicité de cette solution ainsi que la convergence de la méthode ne sont pas au programme et dépassent de loin le cadre de ce cours de première année. Nous ne les citerons qu'à titre informatif : f est de classe $\mathscr{C}^1$ et lipschitzienne par rapport à la deuxième variable.

Méthode d'Euler (suite)

- ```python
  def euler(f,y0,a,b,n):
      p,x=(b-a)/n,linspace(a,b,n+1)
      y=[y0]
      for k in range(n):
          aux=y[k]+p*f(x[k],y[k])
          y.append(aux)
      plot(x,y),show()
  ```
- Exemple : solution approchée de $y' = \sin\left(x^2\right)$, $y(0) = 0.25$ sur $[0, 7]$ avec $n = 10$, $n = 25$, $n = 50$ et $n = 100$.

  ```
  >>> from pylab import *
  >>> f=lambda t,x:sin(x**2)
  >>> euler(f,0.25,0,7,100)
  ```
  ```

Méthode d'Euler (suite)



V
Résolution d'un système linéaire par la méthode du pivot de Gauss

La méthode du pivot de Gauss

- Cadre : le programme se limite à la résolution de $MX = Y$ où M est une matrice carrée inversible.
- La méthode a été décrite et justifiée dans le cours de mathématiques : on triangularise la matrice augmentée du système $A := (M|Y)$ par opérations élémentaires (transvections et permutations), puis on résout le système triangulaire "*en remontant*".

Le type array de Numpy

- Nous utiliserons le type `array` de **Numpy**. La syntaxe est $A =$ `array([[1,2],[3,4]])` pour une matrice de $\mathfrak{M}_2(\mathbb{R})$. On accède aux coefficients par $A[i,j]$.

Opérations élémentaires

Attention, il faut réaliser une copie de la ligne à modifier. On rappelle que, pour les listes et les matrices, `A=B` ne crée pas en A une copie indépendante de B : si B est modifié dans la suite du programme, A le sera également.

```
def perm(A,i,j):
    aux=copy(A[i,:])
    A[i,:]=A[j,:]
    A[j,:]=aux
def trans(A,i,j,mu):
    A[i,:]=A[i,:]+mu*A[j,:]
```

Choix d'un pivot optimal

Afin de minimiser les erreurs d'arrondis, on fait le choix d'un pivot maximal (en valeur absolue).

```
def trouverPivot(A,col):
    imax,max,n=col,abs(A[col,col]),len(A)
    for i in range(col,n):
        if abs(A[i,col])>max:
            imax,max=i,abs(A[i,col])
    return(imax)
```

Échelonnement

Comme la matrice est inversible, on est certain de trouver un pivot par colonne.

```
def echelon(A):
    n=len(A)
    for i in range(0,n-1):
        ipivot=trouverPivot(A,i)
        perm(A,i,ipivot)
        for j in range(i+1,n):
            trans(A,j,i,-A[j,i]/A[i,i])
```

Remontée

```
def resol(A):
    n,AA=len(A),copy(A)
    sol=[0]*n
    echelon(AA)
    for i in range(n):
        sol[n-1-i]=AA[n-1-i,n]
        for j in range(n-i,n):
            sol[n-1-i]=sol[n-1-i]-sol[j]*AA[n-1-i,j]
        sol[n-1-i]=sol[n-1-i]/AA[n-1-i,n-1-i]
    return(sol)
```

Un exemple

La résolution du système

$$\begin{cases} 0.25x + y + z + t = 3 \\ x + y + 1.4z + t = 4 \\ 2x + 3y + 4z + 5t = 6 \\ x + 2y + 3z + 4t = 1 \end{cases}$$

aboutit à

```
>>> M=array([[0.25,1,1,1,3],[1,1,1.4,1,4],[2,3,4,5,6],
                                        [1,2,3,4,1]],dtype=float)
>>> resol(M)
[2.666666666666667, 6.75, -2.500000000000009,
-1.916666666666661]
```

Complexité de la méthode du pivot

- *L'étape d'échelonnement.* Pour transformer A en une matrice triangulaire par des opérations de pivot, il faut dans le pire des cas à la i-ème étape : une permutation pour échanger L_i avec une ligne de pivot ($2n$ opérations) et $n-i$ transvections pour modifier $L_{i+1}, \ldots, L_n$ ($n + n(n-i) = n(n+1-i)$ opérations en tout). Ainsi, l'étape d'échelonnement admet une complexité dans le pire des cas de l'ordre de

$$\sum_{i=1}^{n-1} (2n + n(n+1-i)) = n\frac{(n-1)(n+6)}{2} = O\left(n^3\right)$$

- *La remontée.* Le calcul de la composante $Y_{i,1}$ de la solution du système nécessite $1 + n - i$ opérations dans le pire des cas.

Complexité de la méthode du pivot (suite)

- La remontée du système triangulaire admet donc une complexité dans le pire des cas de l'ordre de

$$\sum_{i=1}^{n}(n+1-i) = \frac{n(n+1)}{2} = O\left(n^2\right)$$

La complexité globale de l'algorithme de Gauss est en $O\left(n^3\right)$.