LABORATOIRE DE GÉNIE CHIMIQUE
TOULOUSE • UMR 5503

Formation continue

Python pour l'ingénierie des procédés

python™

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

≈MATLAB

FiPy: A Finite Volume PDE Solver Using Python

≈ COMSOL

Université de Toulouse

Patrice Bacchin, Fabien Chauvet, Yannick Hallez, Formation continue LGC, 2018

Pourquoi Python ?

- **Python : langage de programmation gratuit (libre de droit d'usage) facile à apprendre et à relire (point - : moins rapide que Fortran ou C)**
- **De nombreuses bibliothèques d'outils spécialisés**

  -> équivalent de Matlab ou de Comsol
- **Une communauté d'utilisateurs (forum)**

- **Intérêt croissant pour ce langage (des lycées à Google !)**
- **Un plus important sur le CV (on peut faire bien plus que du calcul)**

-> un élément différenciant pour un recrutement

Comment l'installer et l'utiliser ?

- **Le compilateur / interpréteur**
  - Python : https://www.python.org
- **Le langage + les bibliothèques**
  - Sous windows : plateforme Canopy pour Python sous windows https://www.enthought.com/products/canopy/ (gratuit pour les étudiants et académiques)
  - Sous linux en installant Anaconda
- **En ligne**
  - https://www.pythonanywhere.com/
  - http://www.sagemath.org/fr/
- **Notebook** IPython Interactive Computing
- **IPython notebook server in the cloud as Wakari, Authorea or CoCalc**

Les bibliothèques indispensables

- **Pour le calcul**
  - Numpy
  - Scipy
  - Matplotlib

Equivalent de Scilab ou Matlab

- **Pour le calcul différentiel**
  - Fipy

Equivalent de Comsol

- **Pour l'analyse d'image**
  - PIL (Python Imaging Library)
- **Pour le traitement de données**
  - Pandas, statsmodel, Scikit-learn
- **Pour le contrôle-commande**
  - python-control package

Numpy

- **Création de tableau**

```
In [6]: import numpy as np
        a= np.zeros(5)
        a
Out[6]: array([ 0.,  0.,  0.,  0.,  0.])

In [7]: b=np.linspace(-2,3,6)
        c=np.arange(-2,3,1)
        print b
        print c
        [-2. -1.  0.  1.  2.  3.]
        [-2 -1  0  1  2]

In [12]: d=np.array(range(10))
         d
Out[12]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- **Caractéristiques d'un tableau a**

  a.shape -> dimensions

  a.dtype -> type des éléments

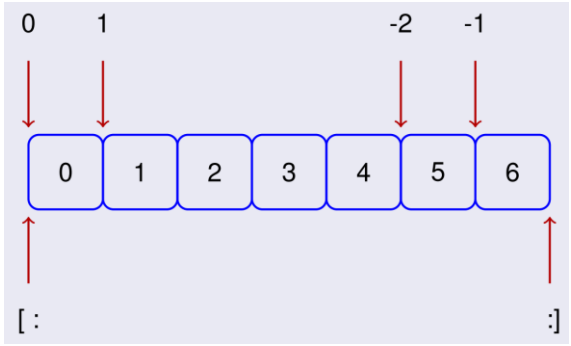
  a.size -> nombre total d'éléments

```
In [11]: x,y=[1,2,3],[4,5,6]
         e=np.array([x,y])
         e
Out[11]: array([[1, 2, 3],
                [4, 5, 6]])

In [20]: print e.shape
         print e.dtype
         print e.size
         (2L, 3L)
         int32
         6
```

Numpy : allocation dans un tableau

- **Tableaux alloués en mémoire selon alignement C ou fortran**
  - ⚠ **Indices commencent à 0**
- **Accès aux éléments**




```
x,y=[1,2,3],[4,5,6]
e=np.array([x,y])
print e
print e[0,0] # la 1er ligne et 1ere colonne
print e[1,2] # la 2eme ligne et 3eme colonne
print e[-1,-1] # la dernière ligne et dernière colonne
print e[:,1] # toutes les lignes et 2eme colonne
print e[e>2]
```

```
[[1 2 3]
 [4 5 6]]
1
6
6
[2 5]
[3 4 5 6]
```

- Modification des éléments

```
e[1,1]=10
print e
```

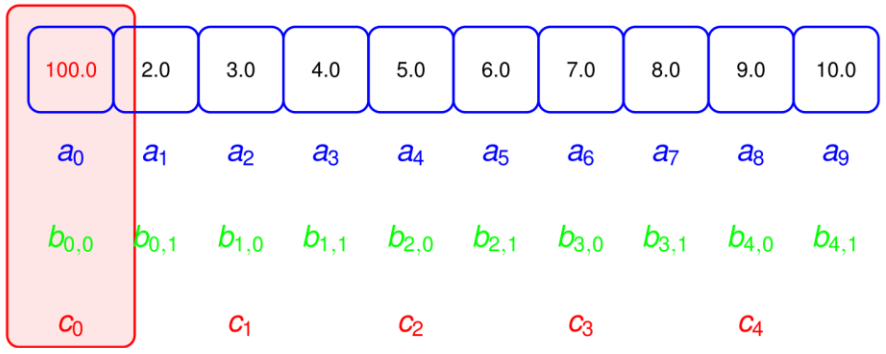
```
[[ 1  2  3]
 [ 4 10  6]]
```

```
e[:,1]=10
print e
```

```
[[ 1 10  3]
 [ 4 10  6]]
```

Numpy : redimensionnement d'un tableau

- **Tableaux alloués en mémoire**



- **Redimensionnement .reshape**

```
a= np.linspace(1,10,10)
b= a.reshape((5,2))
c=b[:,0]
a[0]=100
print a
print b
print c
```

```
[ 100.    2.    3.    4.    5.    6.    7.    8.    9.   10.]
[[ 100.    2.]
 [   3.    4.]
 [   5.    6.]
 [   7.    8.]
 [   9.   10.]]
[ 100.    3.    5.    7.    9.]
```

- **Attention, la modification d'un élément s'applique à tous les tableaux liés**

- Pour créer, un nouveau tableau (nouvelle allocation en mémoire) utiliser.copy()

```
c=a.copy()
a[1]=200.
print a
print c
```

```
[ 100.  200.    3.    4.    5.    6.    7.    8.    9.   10.]
[ 100.    2.    3.    4.    5.    6.    7.    8.    9.   10.]
```

Numpy : opérations sur les tableaux

- **Somme d'éléments : a.sum() ou np.sum(a)**
- **Somme de tableau : a+a**
- **Multiplication : a*a**
- **Produit vectoriel ou matriciel : np.dot(a,a)**

Fonctions de calcul basiques

Matplotlib

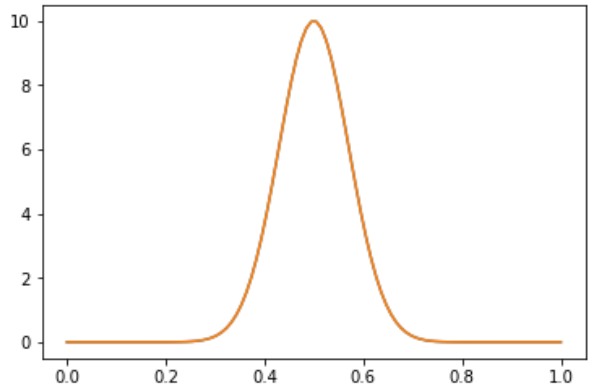
Tracé

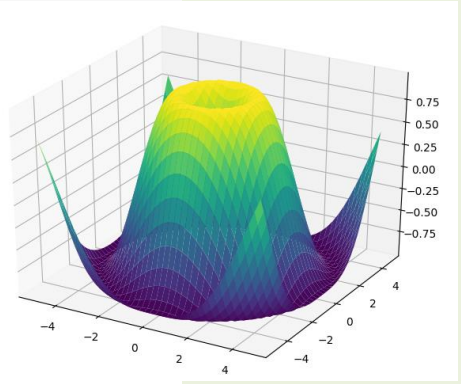
(voir matplotlib pour toutes les options de graphes prévoir plusieurs soirées !)

```
In [11]: from numpy import *
         x= linspace (0.,1., 100)
         f= 10.*exp(-((x-0.5)/0.1)**2)
         print f
```

```
[  1.38879439e-10   3.77474939e-10   1.00525455e-09   2.62302071e-09
   6.70602464e-09   1.67983382e-08   4.12292325e-08   9.91475372e-08
   2.33612654e-07   5.39322510e-07   1.21994003e-06   2.70374774e-06
   5.87126398e-06   1.24920812e-05   2.60420842e-05   5.31929936e-05
   1.06456180e-04   2.08749322e-04   4.01067085e-04   7.54999601e-04
   1.39256082e-03   2.51663004e-03   4.45617596e-03   7.73113109e-03
   1.31420010e-02   2.18885859e-02   3.57200177e-02   5.71141054e-02
   8.94772943e-02   1.37347279e-01   2.06568943e-01   3.04402171e-01
   4.39509449e-01   6.21765240e-01   8.61831599e-01   1.17045882e+00
   1.55749843e+00   2.03065828e+00   2.59408275e+00   3.24689734e+00
   3.98190635e+00   4.78466215e+00   5.63312339e+00   6.49807924e+00
   7.34443672e+00   8.13335512e+00   8.82508158e+00   9.38221701e+00
   9.77304662e+00   9.97452490e+00   9.97452490e+00   9.77304662e+00
   9.38221701e+00   8.82508158e+00   8.13335512e+00   7.34443672e+00
   6.49807924e+00   5.63312339e+00   4.78466215e+00   3.98190635e+00
   3.24689734e+00   2.59408275e+00   2.03065828e+00   1.55749843e+00
   1.17045882e+00   8.61831599e-01   6.21765240e-01   4.39509449e-01
   3.04402171e-01   2.06568943e-01   1.37347279e-01   8.94772943e-02
   5.71141054e-02   3.57200177e-02   2.18885859e-02   1.31420010e-02
   7.73113109e-03   4.45617596e-03   2.51663004e-03   1.39256082e-03
   7.54999601e-04   4.01067085e-04   2.08749322e-04   1.06456180e-04
   5.31929936e-05   2.60420842e-05   1.24920812e-05   5.87126398e-06
   2.70374774e-06   1.21994003e-06   5.39322510e-07   2.33612654e-07
   9.91475372e-08   4.12292325e-08   1.67983382e-08   6.70602464e-09
   2.62302071e-09   1.00525455e-09   3.77474939e-10   1.38879439e-10]
```

```
In [13]: import matplotlib.pyplot as plt
         plt.plot (x, f)
         plt.show()
```





Scipy : pour le calcul scientifique

- **Modules de calculs scientifiques**
  - Solveur **optimize**
    - **Bisect**
    - **Newton**
    - **fsolve**
  - Interpolation **interpolate**
  - Integration **integrate**
    - **Equation différentielle ordinaire odeint**

➡ Application à des cas fréquents retrouvés sur les problèmes Chemical Engineering

intégration d'EDO -> cinétique chimique, bilan de population

intégration d'EDO + shooting method -> colonne à contre courant, convection-diffusion-réaction

Autre idée d'applications ?

Conseils pour la programmation

- 80 % des problèmes viennent d'une physique incorrecte : importance d'un modèle (équations + conditions limites) bien posé
- Appliquer le principe KISS (keep it as simple as possible) et compliquer la simulation progressivement (ajouter les ingrédients progressivement)
- Pensez à vérifier votre code régulièrement en l'exécutant (pour identifier facilement des erreurs liées à vos derniers changements)
- Ne pas désespérer : debugger fait partie de la programmation

Une citation bien adaptée à la programmation :

Le succès consiste à aller d'échecs en échecs sans perdre son enthousiasme

Winston Churchill

Recherche de racine

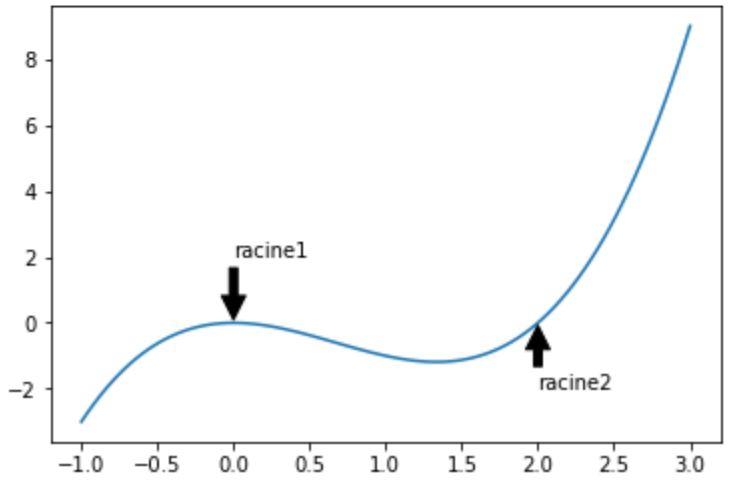
Racine de la fonction

$$x^3-2x^2= x^2(x-2)$$

```
import matplotlib.pyplot as plt
import numpy as np
def f(x):
    return (x**3)-2*(x**2)
x=np.linspace(-1,3,100)
plt.plot(x,f(x))

from scipy.optimize import newton
x1=newton(f,x0=1.)
x2=newton(f,x0=3.)
print x1
print x2
plt.annotate('racine1', xy=(x1, 0), xytext=(x1, 2),arrowprops=dict(facecolor='black', shrink=0.05),)
plt.annotate('racine2', xy=(x2, 0), xytext=(x2, -2),arrowprops=dict(facecolor='black', shrink=0.05),)
plt.show()
```

```
-1.82851760603e-08
2.0
```



Intégration d'EDO

Succession de deux réactions chimiques de premier ordre

$$A \overset{k_1}{\rightarrow} B \overset{k_2}{\rightarrow} C$$

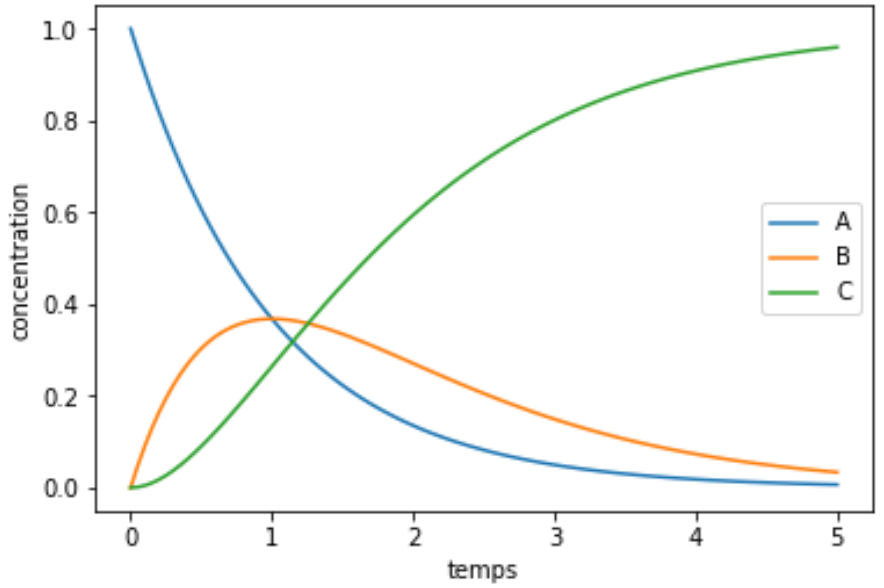
$$\begin{cases} \frac{dA(t)}{dt} = -k_1 A(t) \\ \frac{dB(t)}{dt} = k_1 A(t) - k_2 B(t) \\ \frac{dC(t)}{dt} = k_2 B(t) \end{cases}$$

Résolution en 6 lignes de code !

```
In [27]: from numpy import *
         from scipy import integrate
         k1, k2=1.,1.
         def dX_dt(X, t=0):
             return array([-k1*X[0],k1*X[0]-k2*X[1],k2*X[1]])
         t = linspace(0, 5,  100)              # time
         X0 = array([1, 0, 0])                 # conditions initiales
         X, infodict = integrate.odeint(dX_dt, X0, t, full_output=True)
         infodict['message']

Out[27]: 'Integration successful.'
```

```
In [28]: import matplotlib.pyplot as plt
         [A,B,C]=plt.plot (t, X )
         plt.legend([A,B,C],["A","B","C"], loc='best')
         plt.xlabel ('temps')
         plt.ylabel ('concentration')
         plt.show()
```



Application 1 : réacteur d'ozonation

Cinétique de production d'ozone : $3O_2 \xrightarrow{k} 2O_3$

$$-\frac{\varepsilon}{V}\frac{dn_{O_2}}{dt} = k[O_2]^{\frac{3}{2}} \ , \ \ n_{O_2} = n_0(1-x) \ , \varepsilon = -{}^1/_3$$

$$\boxed{\frac{dx}{dt} = -\frac{k}{\varepsilon}\left(\frac{n_0}{V_0}\right)^{1/2}\frac{(1-x)^{3/2}}{(1+\varepsilon x)^{1/2}}}$$

1- Tracer le taux de conversion en fonction du temps

2 -Trouver le temps pour atteindre 99% de conversion

https://www.suezwaterhandbook.fr/procedes-et-technologies/oxydation-desinfection/oxydation-et-desinfection-par-l-ozone/applications

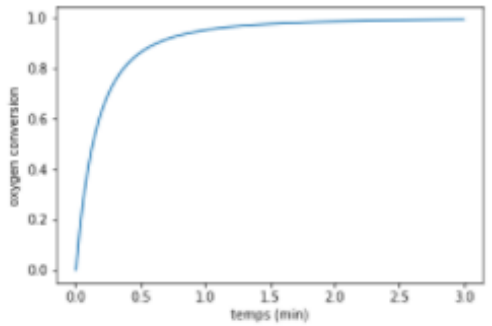
- **Solution :**

http://nbviewer.jupyter.org/url/www.patricebacchin.fr/python/Ozonation.ipynb

```
from numpy import *
from scipy import integrate
k1, c0, eps =0.316, 40.09, -1./3.     # mol-1/2.m-3/2,    mol.m-3,    -
kc=-k1*(c0**0.5)/eps
def dX_dt(X, t):
    return kc*((1-X)**1.5)/((1+eps*X)**0.5)
t = linspace(0, 3,  100)         # time
X0 = 0.                          # conditions initiales
X, infodict = integrate.odeint(dX_dt, X0, t, full_output=True)
infodict['message']
```

```
'Integration successful.'
```

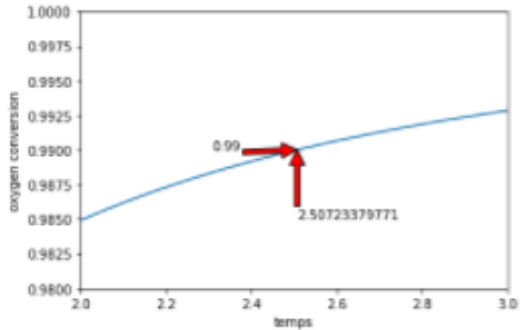
```
import matplotlib.pyplot as plt
plt.plot (t, X )
plt.xlabel ('temps (min)')
plt.ylabel ('oxygen conversion')
plt.show()
```




```
cible=0.99
def dt_dX(X):
    return 1./(kc*((1-X)**1.5)/((1+eps*X)**0.5))
tb=integrate.quad(dt_dX, 0, cible)[0]
print tb, 'min'
```

```
2.50723379771 min
```

```
plt.plot (t, X )
plt.xlim((2,3))
plt.ylim((0.98,1))
plt.xlabel ('temps')
plt.ylabel ('oxygen conversion')
plt.annotate(cible, xy=(tb, cible), xytext=(tb-0.2, cible),arrowprops=dict(facecolor='red'),)
plt.annotate(tb, xy=(tb, cible), xytext=(tb, cible-0.005),arrowprops=dict(facecolor='red'),)
plt.show()
```



Convection-diffusion-réaction

Un problème classique en génie chimique : réacteur piston, transport de polluant dans un sol ou une rivière mais aussi une membrane catalytique, une dialyse avec adsorbant …

$$\frac{dc}{dt} = -\frac{dj}{dz} - kc$$

$$j = uc - D\frac{dc}{dz}$$

$$\Rightarrow \boxed{D\frac{d^2c}{dz^2} - u\frac{dc}{dz} - kc = 0}$$

$$\begin{cases} c = c_0 & z = 0 \\ \frac{dc}{dz} = 0 & z = \delta \end{cases}$$

Système de 2 EDO

$$\begin{cases} \frac{dc}{dz} = v \\ \frac{dv}{dz} = \frac{u}{D}v + \frac{k}{D}c \end{cases} \qquad \begin{cases} \frac{d\hat{c}}{d\hat{z}} = \hat{v} \\ \frac{d\hat{v}}{d\hat{z}} = Pe\hat{v} + Ha^2\hat{c} \end{cases}$$

$$\hat{c} = \frac{c}{c_0} \qquad \hat{z} = \frac{z}{\delta} \qquad Pe = \frac{u\delta}{D} \qquad Ha^2 = \frac{k\delta^2}{D}$$

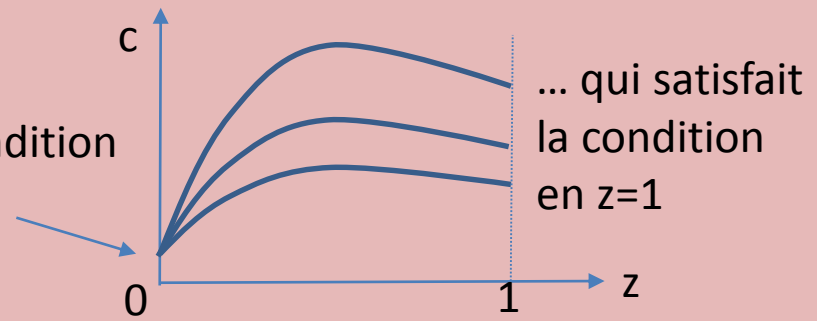
Méthode du tir ou Shooting method

Souvent, les 2 conditions limites s'appliquent aux extrémités du domaine.

Les intégrations doivent s'effectuer avec 1 condition limite fixée de façon aléatoire en z=0 puis déterminée par itérations pour respecter la condition limite en z=1

Intégration **–odeint-** (tir) avec différents $\frac{dc}{dz}|_{z=0}$ puis trouver par optimization **–newton-** la condition en z=0 qui permet d'avoir la condition souhaitée en z=L

Trouver la condition en z=0 …



… qui satisfait la condition en z=1

Application 2 : convection-diffusion-reaction

1- Ecrire le code pour la résolution numérique en régime permanent pour les conditions suivantes

$$c = c_0 \quad \mathrm{z} = 0$$
$$\frac{dc}{dz} = 0 \quad \mathrm{z} = \delta$$

2 -Trouver la concentration en sortie pour un Péclet de 1 et un nombre de Hatta de 2

Convection-diffusion -réaction

Valeur $\frac{dc}{dz}|_{z=1}$ souhaitée

Valeur $\frac{dc}{dz}|_{z=0}$ arbitraire

Fonction cible retournant après intégration $\frac{dc}{dz}|_{z=1}$ calculée - souhaitée

Optimisation pour trouver $\frac{dc}{dz}|_{z=0}$ afin que la function cible soit nulle

```
# Convection-diffusion-reaction (first order) equation at steady state
# Shooting method to solve the problem for boundary conditions at both end
from numpy import *
from scipy import integrate
from scipy import optimize
Pe, Ha=1.,2.
C0=1.
dC1=0.
n=100
z = linspace(0, 1, n)

#Set of ordinary differential equations
def dX_dt(X, z):
    return array([X[1],Pe*X[1]+Ha*Ha*X[0]])

dC0_try=-Pe
# function giving the gap between dc/dz|z=1 and the targeted boundary conditions
def cible(dC0_try):
    X0 = array([C0, dC0_try])        # boundary conditions at t=0
    X, infodict = integrate.odeint(dX_dt, X0, z, full_output=True)
    return X[n-1,1]-dC1

#find the dc/dz|z=0 to have a cible=0
dC0_found=optimize.newton(cible, dc0_try)
print "Solution found for dc/dz|z=0 at", dc0_found

X0 = array([C0, dC0_found])
X, infodict = integrate.odeint(dX_dt, X0, z, full_output=True)
print "c|z=0     = ", X[0,0], "            c|z=1     =", X[n-1,0]
print "dc/dz|z=0 =", X[0,1], "  dc/dz|z=1 =", X[n-1,1]
```

```
Solution found for dc/dz|z=0 at [-1.52124686]
c|z=0     =  1.0             c|z=1     = 0.334411336713
dc/dz|z=0 = -1.52124686145   dc/dz|z=1 = 1.88003432267e-12
```

http://nbviewer.jupyter.org/url/patricebacchin.fr/python/convection-diffusion-reaction-2.ipynb

Convection-diffusion -réaction

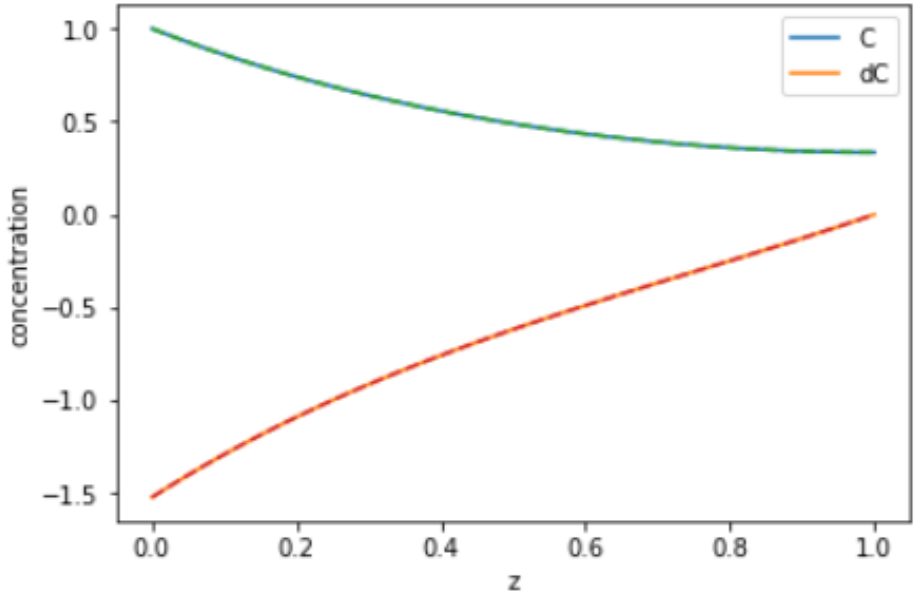
Tracé de la solution numérique

```
import matplotlib.pyplot as plt
[C, dC]=plt.plot (z, X )

#analytical solution
delta=(Pe*Pe+4*Ha*Ha)**0.5
c=exp((Pe-delta)*z/2.)/(1-(Pe-delta)*exp(-delta)/(Pe+delta
dc=((Pe-delta)/2.)*exp((Pe-delta)*z/2.)/(1-(Pe-delta)*exp(

plt.plot(z,c,'--')
plt.plot(z,dc,'--')
plt.legend([C, dC],["C","dC"], loc='best')
plt.xlabel ('z')
plt.ylabel ('concentration')
plt.show()
```

```
-1.52124683975
```



Comparaison aux solutions analytiques publiées pour un ordre un :

$c = c_0 \quad z = 0$
$\frac{dc}{dz} = 0 \quad z = \delta$

Applications aux membranes catalytiques

Gu, Y., Bacchin, P., Lahitte, J. F., Remigy, J. C., Favier, I., Gómez, M., ... & Noble, R. D. (2017). Catalytic membrane reactor for Suzuki-Miyaura C– C cross-coupling: Explanation for its high efficiency via modeling. *AIChE Journal*, *63*(2), 698-704.

$c = c_0 \quad z = 0$
$c = c_1 \quad z = \delta$

Applications à la dialyse avec absorbant

Snisarenko, D., Pavlenko, D., Stamatialis, D., Aimar, P., Causserand, C., & Bacchin, P. (2017). Insight into the transport mechanism of solute removed in dialysis by a membrane with double functionality. *Chemical Engineering Research and Design*, *126*, 97-108.

Application 3 : colonne à contre-courant

Transfert local (⚠ change suivant la hauteur)

$$N = -k_{gl}(X - Y/K) \qquad \frac{1}{k_{gl}} = \frac{1}{k_X} + \frac{1}{(K.k_Y)}$$

Equilibre

$$Y = KX$$

Differential mass balance

Entrée – Sortie = 0

$$AX - A(X + dX) - N d\text{Sint} = 0$$

$$A(Y + dY) - AY + N d\text{Sint} = 0$$

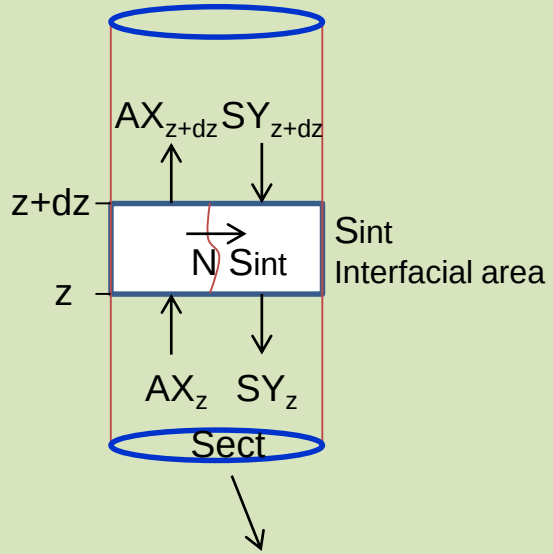
$$A\frac{dX}{dz} = -k_{gl}(X - Y)\ a\ \text{Sect}$$

$$S\frac{dY}{dz} = -k_{gl}(X - Y)\ a\ \text{Sect}$$

Conditions limites

Y=0 z=L

X=X0 z=0



S int=aV=a Sect L

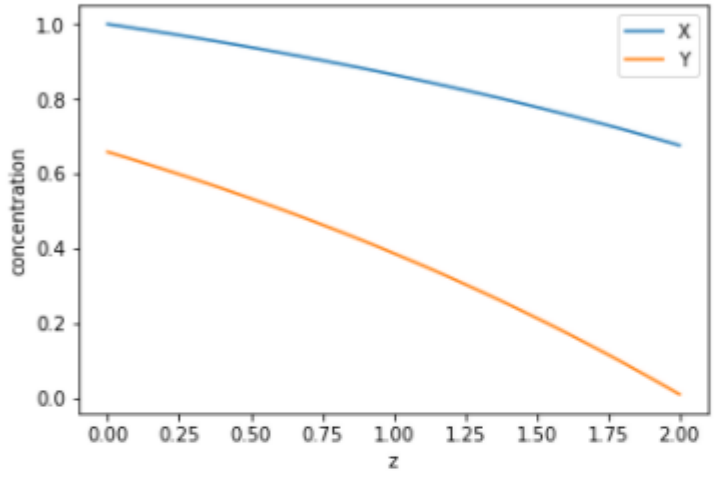
dS int=adV=a Sect dz

Problème 2 EDO avec les conditions limites sur 2 bornes décalées

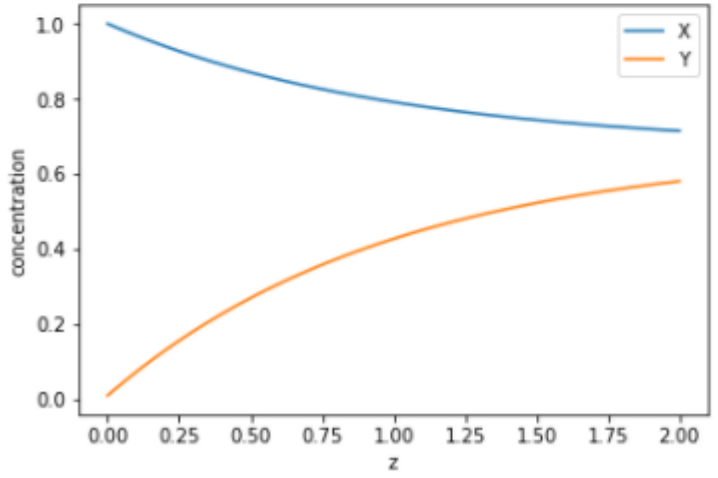
➡ Shooting method

Ou approche ingénieur NUT/HUT

```
Contre-courant
X|z=0      =  1.0              X|z=1      = 0.675956814442
Y|z=0 = 0.658086371115    Y|z=1 = 0.01
A(X0-X1) 6.48086371115e-06 S(Y1-Y0) -6.48086371115e-06
Efficacité dans le raffinat 0.325671543274 d'extraction 0.325671543274
```




```
Co-courant
X|z=0      =  1.0              X|z=1      = 0.714660643197
Y|z=0 = 0.01    Y|z=1 = 0.580678713606
A(X0-X1) 5.70678713606e-06 S(Y1-Y0) 5.70678713606e-06
```



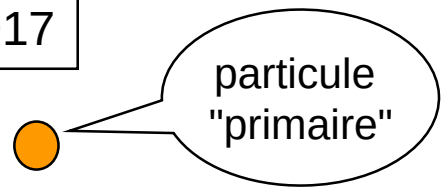

```python
# Transfert in a counter-current column
# Shooting method to solve the problem for boundary conditions at both end
from numpy import *
from scipy import integrate
from scipy import optimize
kx, ky=1.e-4, 1.e-4
K=2.
sect, a =  0.001,100.
A, S= 0.00002, 0.00001
X_0, Y_1 =1., 0.01
n=100
L=2
z = linspace(0, L, n)
#Set of ordinary differential equations
def dX_dz(X, z):
    kgl=1./((1./kx)+(1./(ky*K)))
    N=-kgl*(X[0]-X[1])*sect*a
    return array([N/A,N/S])
Y0_try=1.
# function giving the gap between dc/dz|z=1 and the targeted boundary conditions
def cible(Y_0):
    X0 = array([X_0, Y_0])        # boundary conditions at t=0
    X, infodict = integrate.odeint(dX_dz, X0, z, full_output=True)
    return X[n-1,1]-Y_1
#find the dc/dz|z=0 to have a cible=0
Y0_found=optimize.newton(cible, Y0_try)
X0 = array([X_0, Y0_found])
X, infodict = integrate.odeint(dX_dz, X0, z, full_output=True)
print "Contre-courant"
print "X|z=0      = ", X[0,0], "            X|z=1      =", X[n-1,0]
print "Y|z=0 =", X[0,1], "  Y|z=1 =", X[n-1,1]
print "A(X0-X1)",A*(X[0,0]-X[n-1,0]),"S(Y1-Y0)", -S*(X[0,1]-X[n-1,1])
print "Efficacité dans le raffinat", (X[0,0]-X[n-1,0])/(X[0,0]-(X[n-1,1]/K)), "d'extrac
import matplotlib.pyplot as plt
[X, Y]=plt.plot (z, X )
plt.legend([X, Y],["X","Y"], loc='best')
plt.xlabel ('z')
plt.ylabel ('concentration')
plt.show()
# Transfert in a co-current column
# Shooting method to solve the problem for boundary conditions at both end
#Set of ordinary differential equations
def dX_dz_co(X, z):
    kgl=1./((1./kx)+(1./(ky*K)))
    N=-kgl*(X[0]-X[1])*sect*a
    return array([N/A,-N/S])
Y_0=Y_1
X0 = array([X_0, Y_0])
X, infodict = integrate.odeint(dX_dz_co, X0, z, full_output=True)
print "Co-courant"
print "X|z=0      = ", X[0,0], "            X|z=1      =", X[n-1,0]
print "Y|z=0 =", X[0,1], "  Y|z=1 =", X[n-1,1]
print "A(X0-X1)",A*(X[0,0]-X[n-1,0]),"S(Y1-Y0)", -S*(X[0,1]-X[n-1,1])
```

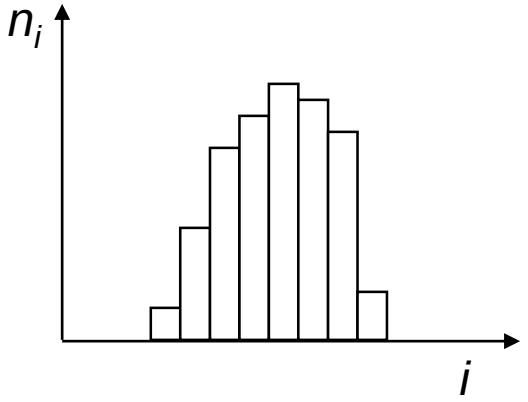
http://nbviewer.jupyter.org/url/patricebacchin.fr/python/Colonne.ipynb

Agrégation et bilan de population

Problème : distribution de taille modifiée au cours du temps
(approche précédente valide pour les premiers temps de l'agrégation)

Smoluchowski 1917

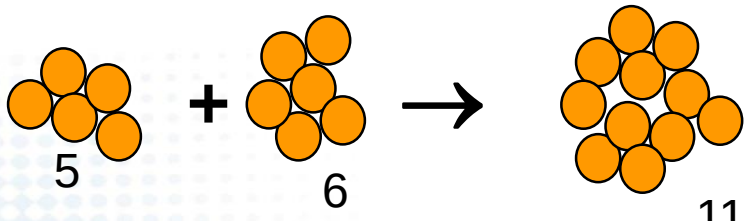




n_i = nombre de particules contenant *i* particules primaires

Il est nécessaire de tenir compte de l'agrégation d'agrégats !

Collisions entre
2 corps



Nombre de collisions entre particules de taille *i* et particules de taille *j* ($m^{-3}\ s^{-1}$)

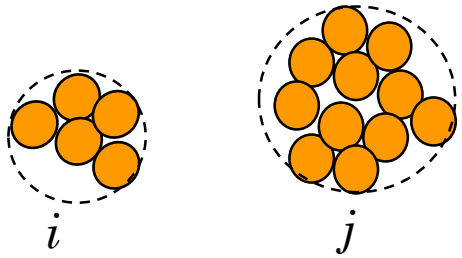
constante de cinétique de 2e ordre

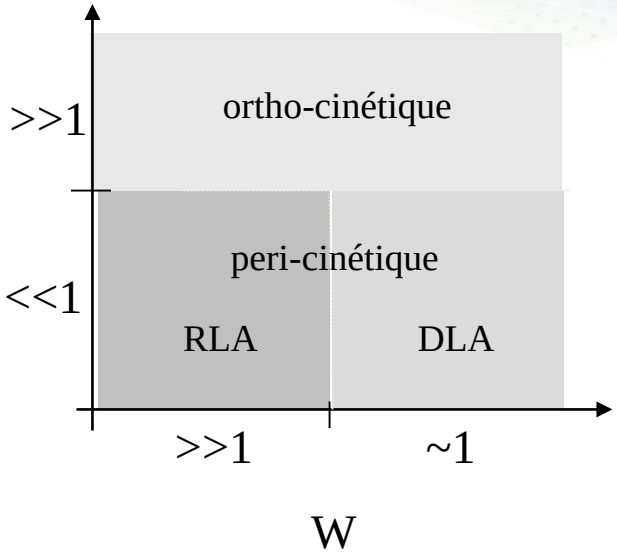
$$\dot{n}_{ij} = k_{ij} n_i n_j$$

Comparaison des modes d'agrégation

$$Pe = \frac{F_{hydr}}{F_{br}} = \frac{6\pi\mu_w \dot{\gamma} a_i^3}{kT}$$

$$= 4.6.10^{18}\, \dot{\gamma} a_i^3$$





•diffusion Brownienne (agrégation péricinétique), Pe <<1

$$k_{ij} = \frac{1}{W}\frac{2k_B T}{3\mu}\frac{(a_i + a_j)^2}{a_i a_j}$$

• mouvement du fluide (agrégation orthocinétique), Pe>>1

$$k_{ij} = \frac{4}{3}\dot{\gamma}\left(a_i + a_j\right)^3$$

• sédimentation différentielle

$$k_{ij} = \left(\frac{2\pi g}{9\mu_w}\right)(\rho_s - \rho)\left(a_i + a_j\right)^3\left(a_i - a_j\right)$$

Agrégation et bilan de population

$$\frac{dn_k}{dt} = \frac{1}{2}\sum_{\substack{i+j=k \\ i=1}}^{i=k-1} k_{ij} n_i n_j - n_k \sum_{k=1}^{\infty} k_{ik} n_i$$

- toutes les collisions sont "efficaces"
- chaque agrégation est irréversible

collisions de particule de taille *i*
avec particule de taille *j*
pour former une particule de taille *k*

collisions de particules de taille *k*
qui forment des particules
d'autres tailles

$$\frac{dn_1}{dt} = -k_{11} n_1^2 - k_{12} n_1 n_2 - k_{13} n_1 n_3 \ldots$$

$$\frac{dn_2}{dt} = \frac{1}{2} k_{11} n_1^2 - k_{12} n_1 n_2 - k_{23} n_2 n_3 \ldots$$

$$\frac{dn_3}{dt} = \frac{1}{2} k_{12} n_1 n_2 - k_{13} n_1 n_3 - k_{23} n_2 n_3 \ldots$$

Concentration totale en particules:

$$\frac{dn}{dt} = -k_a n^2 \qquad \text{si } k_{ij} = k_{11} = 2k_a$$

$$n = \frac{n_0}{1 + k_a n_0 t} \qquad \text{à } t = 0\text{: } n = n_1 = n_0$$

Exemple à débugger !

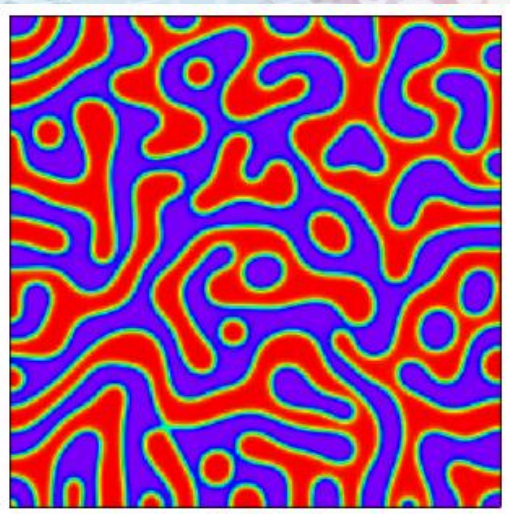
```
# Transfert in a counter-current column
# Shooting method to solve the problem for boundary conditions at both end
from numpy import *
from scipy import integrate
from scipy import optimize
import matplotlib.pyplot as plt
nc=10                #nombre de classe
X0=zeros(nc)
dX_dt_array=zeros(nc)
k=zeros([nc,nc])
for i in range(nc):
    for j in range(nc) :
         k[i,j]=1.
X0[1]=1.
tf=1.
n=10.
t = linspace(0, tf, n)
#Set of ordinary differential equations
def dX_dt(X, t):
    for ik in range(nc):
        dX_dt_array[ik]=0.
        for i in range(ik-1):
            dX_dt_array[ik]=dX_dt_array[ik]-(k[i,ik]*X[i]*X[ik])
            for j in range(nc):
                if i+j==ik:
                    dX_dt_array[ik]=dX_dt_array[ik]+0.5*(k[i,j]*X[i]*X[j])
    return dX_dt_array
X, infodict = integrate.odeint(dX_dt, X0, t, full_output=True)
print X
plt.plot (t, X )
plt.xlabel ('t')
plt.ylabel ('concentration')
plt.show()
```

http://nbviewer.jupyter.org/url/patricebacchin.fr/python/bilan_de_population.ipynb

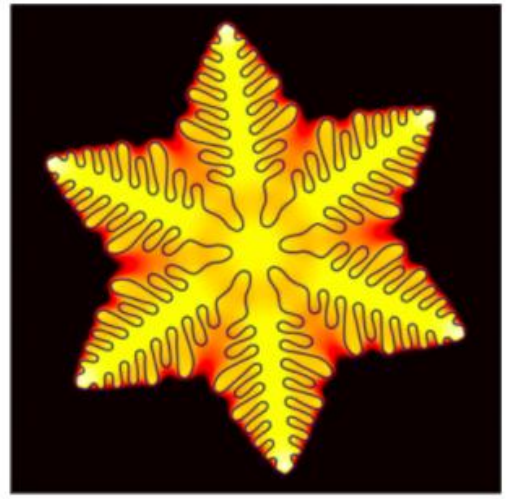
FIPY

Résolution d'équations différentielles partielles EDP

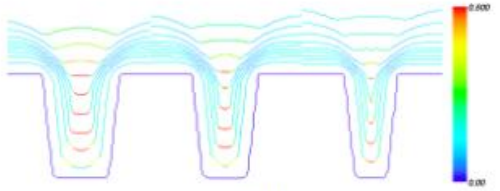
- Transport de matière en 2D-3D en instationnaire
- Navier Stokes



Cahn-Hilliard



Phase Field



Level Set

[Exemples]

Pour une future formation ?

Références

- Documentation en ligne : python , numpy, matplotlib, scipy, fipy
- http://code.activestate.com/recipes/langs/python/
- https://docs.scipy.org/doc/scipy/reference/
- Tutoriel en Français : https://docs.python.org/fr/3/tutorial/index.html
- Pour le traitement de données : https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-tutor1-python-start.pdf

- Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, *9*(3).

- Guyer, J. E., Wheeler, D., & Warren, J. A. (2009). FiPy: Partial differential equations with Python. *Computing in Science & Engineering*, *11*(3).

Données

- **Liste [,] : serie d'objets non nécessairement identiques**
- **Tuple (,) : liste non modifiable**
- **Dictionnaire { , } : clé/valeur non ordonnées**

Commandes de tests

indentation

```
If <condition>:
    <commandes>
Elif <condition>:
    <commandes>
Else :
    <commandes>
```

```
Try:
    <commandes>
Except <condition>:
    <commandes>
Else :
    <commandes>
Finally:
    <commandes>
```

Commandes de boucles

indentation

For <variables> in <serie>:
<commandes>
Else :
<commandes>

While <conditions> :
<commandes>
Else :
<commandes>

Fonction

Def <nom> (<arg>, ……, <arg1>=<valeur>, …):

<commandes>

return <valeur>

Classes et objets

Permet de réaliser une structuration spéciale de données (association de données et de fonction)

Outils intéressant pour structurer des concepts physiques et la logique de traitement

Class <nom>: