Les outils mathématiques

Eléments d'analyse

Intégrer ou dériver une fonction

Pour intégrer une fonction, on utilise la commande `quad(fonction, borne inférieure, borne supérieure)` de la bibliothèque `scipy.integrate`.

```
>>> from scipy.integrate import quad
>>> a = 0 # borne inférieure
>>> b = 1 # borne supérieure
>>> def f(x): # Fonction à intégrer
        return x**3
>>> J, err = quad(f, a, b)

>>> print "l'integrale vaut ", J
l'integrale vaut 0.25
>>> print "erreur = ", err
erreur =  2.77555756156e-15
```

Pour plus de détails, on pourra se reporter à la documentation de **Scipy p.11-13**.

Pour dériver une fonction, on utilise la commande `derivative(fonction, x0, dx=1.0, n=1, args=(), order=3)` de la bibliothèque `scipy.misc`.

`x0` est le point où la dérivée doit être calculée, dx=1.0 est l'espacement, n est l'ordre de la dérivée, args() sont les arguments et order est l'ordre de la dérivée.

```
>>> from scipy.misc import derivative
>>> def f(x): # Fonction à dériver
        return x**3

>>> derivative(f, 2)
13 # Il y aurait-il une erreur ???
>>> derivative(f, 2, 0.01)
12.000000999998317 # Il faut diminuer le pas pour calculer la dérivée...
```

Pour plus de détails, on pourra se reporter à la documentation de **Scipy p.320**.

Racine(s) d'une équation

On recherche les zéros de la fonction \(f:x \to x^4-5x^2+4\) qui se factorise en \((x^2-4)(x^2-1)\) dont les racines sont \(-2, \, -1, \, 1\) et \(2\). On utilise pour ce faire la fonction `bisect(fonction, borne inférieure, borne supérieure)` de la bibliothèque `scipy.optimize`.

Sujet précédent

Les animations

Sujet suivant

Aide mémoire

Recherche rapide

Go

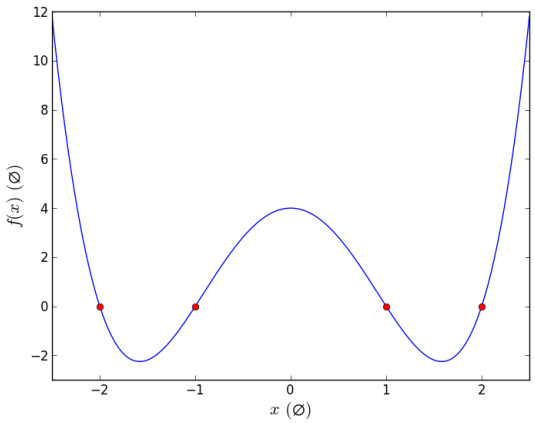
Saisissez un mot clef ou un nom de module, classe ou fonction.

```
# -*- coding: utf-8 -*-

"""
Programme type de recherche de zéros d'une fonction
"""

from __future__ import division
from scipy import *
from pylab import *
from scipy.optimize import bisect            # Module de recherche des zéros

x = linspace(-2.5,2.5,300)                   # Abscisses


def f(x):                                    # Fonction à étudier
    res = x**4-5*x**2+4
    return res

zero1 = bisect(f, -2.5, -1.5)                # Recherche du premier zéro
print(zero1)
zero2 = bisect(f, -1.5, -0.5)                # Recherche du second zéro
print(zero2)
zero3 = bisect(f, -0.5,  1.5)                # Recherche du troisième zéro
print(zero3)
zero4 = bisect(f,  1.5,  2.5)                # Recherche du quatrième zéro
print(zero4)

plot(x, f(x))                                # Graphique

# Annotations
plot([zero1,zero2,zero3,zero4],[f(zero1),f(zero2),f(zero3),f(zero4)],'or')


xlim(-2.5, 2.5)                              # Limites de l'axe des abscisses
xlabel(ur"$x \, (\varnothing)$", fontsize=16)   # Label de l'axe des abscisses

ylim(-3, 12)                                 # Limites de l'axe des ordonnées
ylabel(ur"$f(x) \, (\varnothing)$", fontsize=16)# Label de l'axe des ordonnées

show()
```

ce qui donne

```
>>>
-2.0
-1.0
1.0
2.0
```

et un graphique cohérent



Pour plus de détails, on pourra se reporter à la documentation de **Scipy p.420-421**.

Recherche de minimum

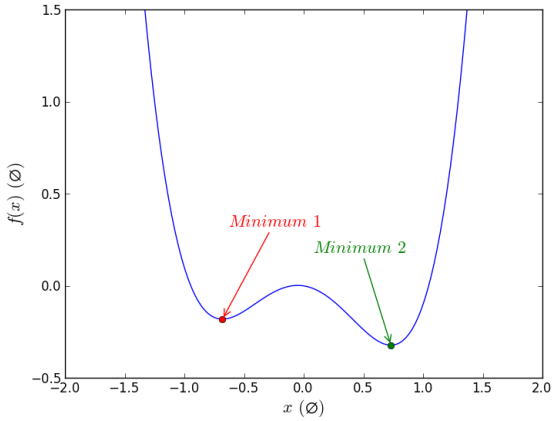
Pour rechercher le miniumum d'une fonction, on utilise la fonction `fmin(fonction, point de recherche)` de la bibliothèque `scipy.optimize`. Dans l'exemple suivant, on cherche les minimas de la fonction \(f:x \to x^4-x^2-0,1x\) proche de \(-0,5\) et \(0,5\)

```
# -*- coding: utf-8 -*-

"""
Programme type de recherche du minimum d'une fonction
"""

from __future__ import division
from scipy import *
from pylab import *
from scipy.optimize import fmin              # Module de recherche des minima

x = linspace(-2.,2.,300)                     # Abscisses


def f(x):                                    # Fonction à étudier
    res = x**4 - x**2 - 0.1*x
    return res

minimum1 = fmin(f, -0.5)                     # Recherche du premier minimum local
print(minimum1)
print(f(minimum1))
minimum2 = fmin(f, 0.5)                      # Recherche du premier minimum local
print(minimum2)
print(f(minimum2))

plot(x, f(x))                                # Graphique

# Annotations
annotate(ur"$Minimum \,1$", fontsize=16,color='r',xy=(minimum1,f(minimum1)), xyc
annotate(ur"$Minimum \,2$", fontsize=16,color='g',xy=(minimum2,f(minimum2)), xyc
plot([minimum1],[f(minimum1)],'or')
plot([minimum2],[f(minimum2)],'og')

xlim(-2, 2)                                  # Limites de l'axe des abscisses
xlabel(ur"$x \, (\varnothing)$", fontsize=16)   # Label de l'axe des abscisses

ylim(-0.5, 1.5)                              # Limites de l'axe des ordonnées
ylabel(ur"$f(x) \, (\varnothing)$", fontsize=16)# Label de l'axe des ordonnées

show()
```

ce qui donne

```
>>>
Optimization terminated successfully.
Current function value: -0.180587
Iterations: 13
Function evaluations: 26
[-0.68066406]
[-0.18058697]
Optimization terminated successfully.
 Current function value: -0.321919
 Iterations: 14
 Function evaluations: 28
[ 0.73085938]
[-0.32191934]
```

et un graphique cohérent



Pour plus de détails, on pourra se reporter à la documentation de **Scipy p.394-395**.

Résoudre un système d'équations

Pour résoudre un système d'équations, on utilise la commande `fsolve(système, initialisation)` de la bibliothèque `scipy.optimize`.

Pour déterminer les solutions du système suivant (\(x=2, \, y=0\) et \(z=-1\)) :

\[\begin{split}\left\{\begin{array}{l} x+10y-3z=5 \quad[1] \\ 2x-y+2z=2 \quad[2] \\ -x+y+z=-3 \quad[3] \end{array}\right.\end{split}\]

Remarque importante : Les équation seront réécrites sous la forme système(variables) = 0.

```
>>> from scipy.optimize import fsolve
>>> def syst(var): # définition du système
        x, y, z = var[0], var[1], var[2] # définition des variables
        eq1 = x +10*y-3*z-5
        eq2 = 2*x-y+2*z-2
        eq3 = -x+y+z+3
        res = [eq1, eq2, eq3]
        return res

x0, y0, z0 = 0, 0, 0 # Initialisation de la recherche des solutions numériques
sol_ini = [x0, y0, z0]

>>> fsolve(syst, sol_ini)
array([  2.00000000e+00,  -1.20797142e-17,  -1.00000000e+00])
```

Pour plus de détails, on pourra se reporter à la documentation de **Scipy p.420-421**.

Une application de résolution de tels systèmes est le dosage d'un triacide par une base forte. Le système à résoudre est le suivant

\[\begin{split}\left\{\begin{array}{l} Equation1=0 \quad[1] \\ Equation2=0 \quad[2] \\ Equation3=0 \quad[3] \end{array}\right.\end{split}\]

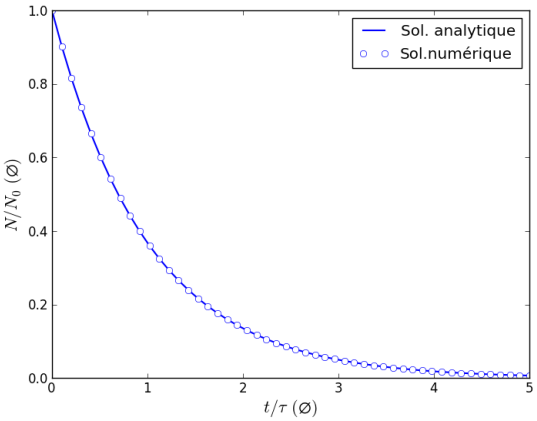
Les équations différentielles

Équation différentielle du premier ordre

La population d'un corps radioactif évolue suivant la loi de désintégration \(\dfrac{dN}{dt}=-\dfrac{N}{\tau}\), où \(N(t)\) est le nombre d'atomes à l'instant \(t\) et \(\tau\) le temps caractéristique de désintégration du corps considéré. Résolvons numériquement cette équation avec \(\tau=1\) et \(N(t=0)=1\) à l'aide de la fonction `odeint` de la bibliothèque `scipy.integrate`.

```
# -*- coding: utf-8 -*-

"""
Exemple de résolution d'équations différentielles
1 équation d'ordre 1 : dN/dt=N/tau
"""

from __future__ import division
from scipy import *
from pylab import *
from scipy.integrate import odeint     # Module de résolution des équations différe

tau = 1

def deriv(syst, t):
    N = syst[0]                    # Variable1 N(t)
    dNdt=-N/tau                    # Equation différentielle
    return [dNdt]                  # Dérivées des variables

# Paramètres d'intégration
start = 0
end = 5
numsteps = 50
t = linspace(start,end,numsteps)

# Conditions initiales et résolution
N0=1
syst_CI=array([N0])                # Tableau des CI
Sols=odeint(deriv,syst_CI,t)       # Résolution numérique de l'équation différentiell

# Récupération des solutions
N = Sols[:, 0]

# Graphiques des solutions
plot(t, exp(-t), '-b', lw=1.5, label=u"Sol. analytique")        # Solution analytique
plot(t, N, 'o', ms=6, mfc='w', mec='b',label=u"Sol.numérique")  # Solution numérique

xlabel(ur"$t/\tau \, (\varnothing)$", fontsize=16)     # Label de l'axe des abscisses
ylabel(ur"$N/N_{0} \, (\varnothing)$", fontsize=16)    # Label de l'axe des ordonnées

legend()                                # Appel de la légende
show()
```

ce qui donne



Systèmes d'Équa. diff. du premier ordre

De la même façon que précedemment, on veut résoudre le système suivant de filation radioactive

\[_{83}^{210}Bi\overset{\tau_{1}}{\rightarrow}{}_{84}^{210}Po\overset{\tau_{2}}{\rightarrow}{}_{82}^{206}Pb\]

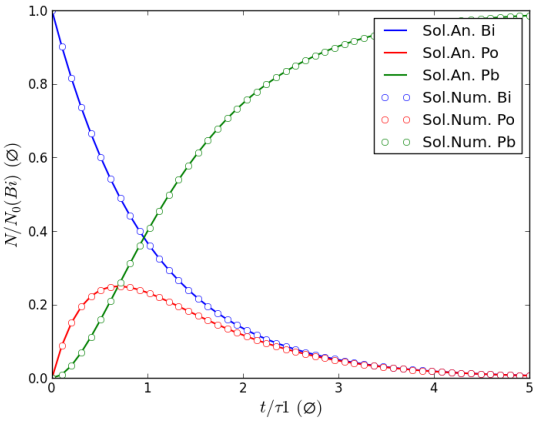
qui suit le système suivant d'équations différentielles (3 équations différentielles d'ordre 1)

\[\begin{split}\left\{\begin{array}{l} \frac{dN\left(Bi\right)}{dt}=-\frac{N\left(Bi\right)}{\tau_{1}} \\ \frac{dN\left(Po\right)}{dt}=\frac{N\left(Bi\right)}{\tau_{1}}-\frac{N\left(Po\right)}{\tau_{2}} \\ \frac{dN\left(Pb\right)}{dt}=\frac{N\left(Po\right)}{\tau_{2}} \end{array}\right.\end{split}\]

On résouds le sytème suivant avec la même procédure que celle utilisée pour une équation différentielle d'ordre 1. On complète juste le nombre de variable et le nombre de conditions initiales nécessaires

```
# -*- coding: utf-8 -*-

"""
Exemple de résolution d'équations différentielles
3 équation d'ordre 1 :
dNBi/dt=-NBi/tau1
dNPo/dt=NBi/tau1-NPo/tau2
dNPb/dt=NPo/tau2
"""

from __future__ import division
from scipy import *
from pylab import *
from scipy.integrate import odeint     # Module de résolution des équations différe

tau1 = 1
tau2 = 0.5

def deriv(syst, t):
    NBi = syst[0]                  # Variable1 NBi(t)
    NPo = syst[1]                  # Variable2 NPo(t)
    NPb = syst[2]                  # Variable3 NPb(t)
    dNBidt=-NBi/tau1               # Equation différentielle 1
    dNPodt=NBi/tau1-NPo/tau2       # Equation différentielle 2
    dNPbdt=NPo/tau2                # Equation différentielle 3
    return [dNBidt,dNPodt,dNPbdt]      # Dérivées des variables

# Paramètres d'intégration
start = 0
end = 5
numsteps = 50
t = linspace(start,end,numsteps)

# Conditions initiales et résolution
NBi0=1
NPo0=0
NPb0=0
syst_CI=array([NBi0,NPo0,NPb0])        # Tableau des CI
Sols=odeint(deriv,syst_CI,t)           # Résolution numérique des équations différentie

# Récupération des solutions
NBi = Sols[:, 0]
NPo = Sols[:, 1]
NPb = Sols[:, 2]

# Graphiques des solutions
# Solutions analytiques
plot(t, exp(-t/tau1), '-b', lw=1.5, label=u"Sol.An. Bi")
plot(t, (tau2*NBi0/(tau1-tau2))*(exp(-t/tau1)-exp(-t/tau2))+NPo0*exp(-t/tau2), '-r', lw=
plot(t, NBi0*(1-exp(-t/tau1)-(tau2/(tau1-tau2))*(exp(-t/tau1)-exp(-t/tau2))) + NPo0*(1-e
# Solutions numériques
plot(t, NBi, 'o', ms=6, mfc='w', mec='b',label=u"Sol.Num. Bi")
plot(t, NPo, 'o', ms=6, mfc='w', mec='r',label=u"Sol.Num. Po")
plot(t, NPb, 'o', ms=6, mfc='w', mec='g',label=u"Sol.Num. Pb")


xlabel(ur"$t/\tau1 \, (\varnothing)$", fontsize=16)     # Label de l'axe des abscisses
ylabel(ur"$N/N_{0}(Bi) \, (\varnothing)$", fontsize=16) # Label de l'axe des ordonnée

legend()                                    # Appel de la légende
show()
```

ce qui donne



Équation différentielle du second ordre

Pour résoudre une telle équation différentielle, il suffit de réécrire l'équation

différentielle du second ordre en système de deux équations différentielles du premier ordre. Par exemple, prenons l'équation de l'oscillateur harmonique amorti

$$\ddot{x}+\frac{\omega_{0}}{Q}\dot{x}+\omega_{0}^{2}x=0$$

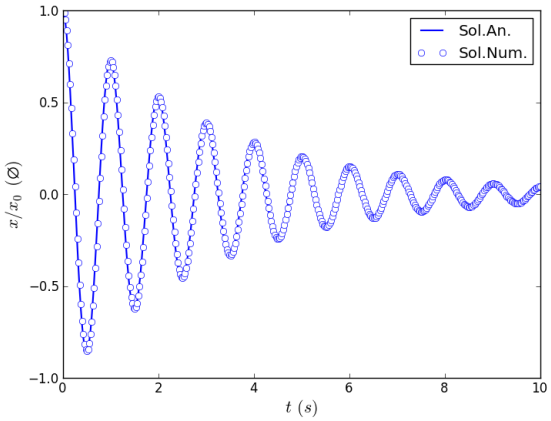
On peut réécrire cette équation en un système de deux équations différentielles d'ordre 1

$$\begin{split}\left\{\begin{array}{l} \frac{dx}{dt}=v \\ \frac{dv}{dt}=-\frac{\omega_{0}}{Q}v-\omega_{0}^{2}x \end{array}\right.\end{split}$$

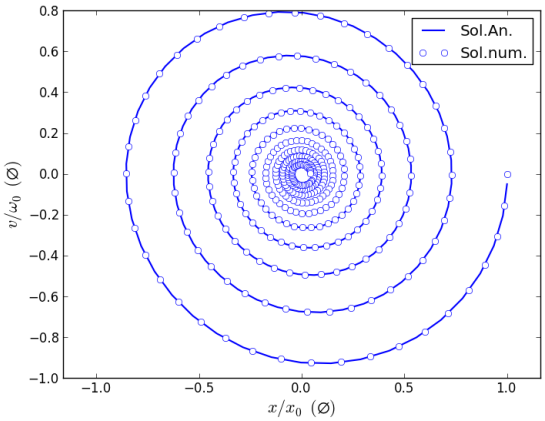
On résouds le sytème précédent avec la même procédure que celle utilisée pour un système d'équations différentielles d'ordre 1.

```
# -*- coding: utf-8 -*-

"""
Exemple de résolution d'équation différentielle
1 équation d'ordre 2
d^2x/dt^2+omega0/Q*dx/dt+omega0^2*x=0
en décomposant en un système de deux équations du premier ordre
dx/dt = v
dv/dt = -omega0/Q*v+omega0^2*x
"""

from __future__ import division
from scipy import *
from pylab import *
from scipy.integrate import odeint     # Module de résolution des équations différe

omega0 = 2*pi
Q = 10

def deriv(syst, t):
    x = syst[0]                    # Variable1 x
    v = syst[1]                    # Variable2 v
    dxdt = v                       # Equation différentielle 1
    dvdt = -omega0/Q*v-omega0**2*x     # Equation différentielle 2
    return [dxdt,dvdt]             # Dérivées des variables

# Paramètres d'intégration
start = 0
end = 10
numsteps = 400
t = linspace(start,end,numsteps)

# Conditions initiales et résolution
x0=1
v0=0
syst_CI=array([x0,v0])             # Tableau des CI
Sols=odeint(deriv,syst_CI,t)       # Résolution numérique des équations différentie

# Récupération des solutions
x = Sols[:, 0]
v = Sols[:, 1]

# Solutions analytiques simplifiées
xanalytique = exp(-omega0/Q/2*t)*cos(omega0*t)
vanalytique_sur_omega0 = -exp(-omega0/Q/2*t)*(1/Q/2*cos(omega0*t)+sin(omega0

# Graphiques des solutions
# Evolution temporelle
figure1 = figure()
plot(t, xanalytique, '-b', lw=1.5, label=u"Sol.An.")          # Solution analytique simplif
plot(t, x, 'o', ms=6, mfc='w', mec='b',label=u"Sol.Num.")      # Solution numérique
xlim(0, 10)                                  # Limites de l'axe des abscisses
xlabel(ur"$t \, (s)$", fontsize=16)                  # Label de l'axe des abscisses
ylim(-1, 1)                                  # Limites de l'axe des ordonnées
ylabel(ur"$x/x_0 \, (\varnothing)$", fontsize=16)            # Label de l'axe des ordonn
legend()

# Portrait de phase
figure2 = figure()
plot(xanalytique, vanalytique_sur_omega0, '-b', lw=1.5, label=u"Sol.An.") # Solution
plot(x, v/omega0, 'o', ms=6, mfc='w', mec='b',label=u"Sol.num.")        # Solution nu
xlim(-1.25, 1.25)                            # Limites de l'axe des abscisses
xlabel(ur"$x/x_0 \, (\varnothing)$", fontsize=16)      # Label de l'axe des abscisses
ylim(-1, 1)                                  # Limites de l'axe des ordonnées
ylabel(ur"$v/\omega_0 \, (\varnothing)$", fontsize=16)  # Label de l'axe des ordonné
axis('equal')
legend()
show()
```

ce qui donne



et pour le portrait de phase



Systèmes d'Équa. diff. du second ordre

La méthode reste la même que pour une équation différentielle du second ordre : Les équations différentielles du second ordre sont réécrites en système d'équations différentielles du premier ordre. Par exemple, pour le problème à 2 corps

\[\begin{split}\left\{\begin{array}{l} m_A\ddot{x}_A=-K\frac{x_A-x_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ m_A\ddot{y}_A=-K\frac{y_A-y_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ m_B\ddot{x}_B=K\frac{x_A-x_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ m_B\ddot{y}_B=K\frac{y_A-y_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \end{array}\right.\end{split}\]

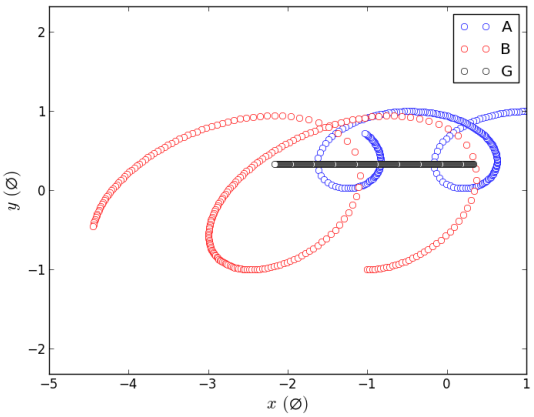
que l'on réécrit en

\[\begin{split}\left\{\begin{array}{l} m_A\frac{dv_{x,A}}{dt}=-K\frac{x_A-x_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ m_A\frac{dv_{y,A}}{dt}=-K\frac{y_A-y_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ m_B\frac{dv_{x,B}}{dt}=K\frac{x_A-x_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ m_B\frac{dv_{y,B}}{dt}=K\frac{y_A-y_B}{((x_A-x_B)^2+(y_A-y_B)^2)^{3/2}} \\ v_{x,A}=\frac{dx_A}{dt} \\ v_{y,A}=\frac{dy_A}{dt} \\ v_{x,B}=\frac{dx_B}{dt} \\ v_{y,B}=\frac{dy_B}{dt} \end{array}\right.\end{split}\]

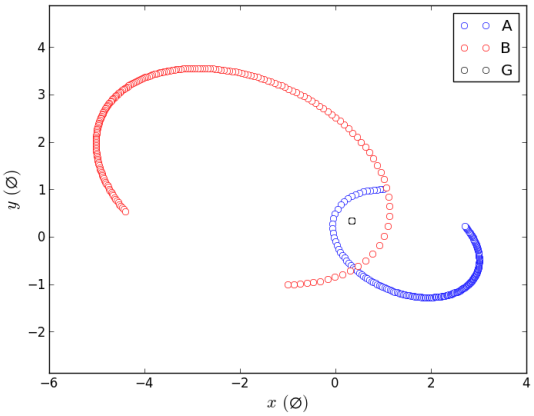
Soit un système de 8 équations différentielles à 8 fonctions temporelles (\(x_A(t),y_A(t),x_B(t), y_B(t),v_{x,A}(t),v_{y,A}(t),v_{x,B}(t),v_{y,A}(t)\)). On résouds le sytème précédent avec la même procédure que celle utilisée pour une équation différentielle d'ordre 2.

```
# -*- coding: utf-8 -*-

"""
Exemple de résolution d'équations différentielles
4 équation d'ordre 2 que l'on réécrit en un système de 8 équations du premier ordre
mA*dvxA/dt+K*xe/de**3
mA*dvyA/dt+K*ye/de**3
mB*dvxB/dt-K*xe/de**3
mB*dvyB/dt-K*ye/de**3
dxAdt=vxA
dyAdt=vyA
dxBdt=vxB
dyBdt=vyB
xe=xA(t)-xB(t)
ye=yA(t)-yB(t):
de=sqrt(xe^2+ye^2):
"""

from __future__ import division
from scipy import *
from pylab import *
from scipy.integrate import odeint      # Module de résolution des équations différe

mA=2
mB=1
K=3

def deriv(syst, t):
    [xA,yA,xB,yB,vxA,vyA,vxB,vyB] = syst              # Variables
    dxAdt=vxA                                 # Equation différentielle 1
    dyAdt=vyA                                 # Equation différentielle 2
    dxBdt=vxB                                 # Equation différentielle 3
    dyBdt=vyB                                 # Equation différentielle 4
    dvxAdt=-K*(xA-xB)/sqrt((xA-xB)**2+(yA-yB)**2)**3/mA # Equation différentielle 5
    dvyAdt=-K*(yA-yB)/sqrt((xA-xB)**2+(yA-yB)**2)**3/mA # Equation différentielle 6
    dvxBdt=K*(xA-xB)/sqrt((xA-xB)**2+(yA-yB)**2)**3/mB  # Equation différentielle 7
    dvyBdt=K*(yA-yB)/sqrt((xA-xB)**2+(yA-yB)**2)**3/mB  # Equation différentielle 8
    return [dxAdt, dyAdt, dxBdt, dyBdt,  dvxAdt, dvyAdt, dvxBdt, dvyBdt]       # Dériv

# Paramètres d'intégration
start = 0
end = 15
numsteps = 250
t = linspace(start,end,numsteps)

# Conditions initiales et résolution
xA0, yA0 = 1, 1
vxA0, vyA0 = -0.5, 0
xB0, yB0 = -1, -1
vxB0, vyB0 = 0.5, 0
syst_CI=array([xA0, yA0, xB0, yB0,  vxA0, vyA0, vxB0, vyB0])   # Tableau des CI
Sols=odeint(deriv,syst_CI,t)           # Résolution numérique des équations différentie

# Récupération des solutions
[xA,yA,xB,yB,vxA,vyA,vxB,vyB] = Sols . T       # Décomposition du tableau des solu
xG = (mA*xA+mB*xB)/(mA+mB)                  # Abscisse du barycentre
yG = (mA*yA+mB*yB)/(mA+mB)                  # Ordonnée du barycentre


# Graphiques des solutions
plot(xA, yA, 'o', ms=6, mfc='w', mec='b', label=u"A")   # Solution numérique
plot(xB, yB, 'o', ms=6, mfc='w', mec='r', label=u"B")   # Solution numérique
plot(xG, yG, 'o', ms=6, mfc='w', mec='k', label=u"G")   # Solution numérique
#xlim(0, 10)                                # Limites de l'axe des abscisses
xlabel(ur"$x \, (\varnothing)$", fontsize=16)          # Label de l'axe des abscisses
#ylim(-1, 1)                                # Limites de l'axe des ordonnées
ylabel(ur"$y \, (\varnothing)$", fontsize=16)          # Label de l'axe des ordonnées
axis('equal')
legend()
show()
```

ce qui donne



ou suivant les conditions initiales



Régressions

Linéaire

Non linéaire

Regression chez les complexes

Transformée de Fourier (fft)

Fichier Rodolphe

Bruit et valeurs aléatoires

Random Uniformément sur 0,1