Leçon 1 : EDO - Application à COVI19-TN

Ines Abdeljaoued Tej (ESSAI - UCAR et BIMS - IPT)

SciPy fournit deux façons de résoudre les équations différentielles ordinaires (EDO) : Une API basée sur la fonction odeint, et une API orientée-objet basée sur la classe ode. odeint est plus simple pour commencer. Commençons par l'importer :

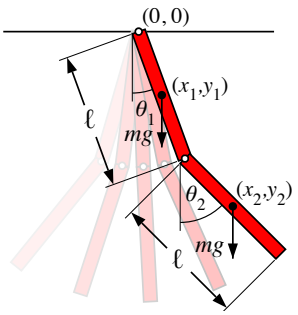
In [1]:
```
import numpy as np
from scipy.integrate import odeint
```

Un système d'EDO se formule de la façon standard : $y' = f(y, t)$ avec $y = [y_1(t), y_2(t), \ldots, y_n(t)]$ et f est une fonction qui fournit les dérivées des fonctions $y_i(t)$. Pour résoudre une EDO il faut spécifier f et les conditions initiales, y(0). Une fois définies, on peut utiliser odeint : `y_t = odeint(f, y_0, t)` où `t` est un NumPy array des coordonnées en temps où résoudre l'EDO. `y_t` est un array avec une ligne pour chaque point du temps `t`, et chaque colonne correspond à la solution `y_i(t)` à chaque point du temps.

Exemple: double pendule Description: http://en.wikipedia.org/wiki/Double_pendulum (http://en.wikipedia.org/wiki/Double_pendulum)

In [2]:
```
from IPython.core.display import Image
Image(url='http://upload.wikimedia.org/wikipedia/commons/c/c9/Double-compound-pendulum-dimensioned.svg')
```

Out[2]:



In [3]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

Les équations du mouvement du pendule sont données sur la page wikipedia :

$$\dot{\theta}_1 = \frac{6}{ml^2}\frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)}$$
$$\dot{\theta}_2 = \frac{6}{ml^2}\frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}.$$

et :

$$\dot{p}_{\theta_1} = \frac{\partial L}{\partial \theta_1} = -\tfrac{1}{2}ml^2\left(\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + 3\frac{g}{l}\sin\theta_1\right)$$
$$\dot{p}_{\theta_2} = \frac{\partial L}{\partial \theta_2} = -\tfrac{1}{2}ml^2\left(-\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + \frac{g}{l}\sin\theta_2\right)$$

où les p_{θ_i} sont les moments d'inertie.

In [4]:
```
g = 9.82
L = 0.5
m = 0.1
def dx(x, t):
    """The right-hand side of the pendulum ODE"""
    x1, x2, x3, x4 = x[0], x[1], x[2], x[3]
    dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * np.cos(x1-x2) * x4)/(16 - 9 * np.cos(x1-
x2)**2)
    dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * np.cos(x1-x2) * x3)/(16 - 9 * np.cos(x1-
x2)**2)
    dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * np.sin(x1-x2) + 3 * (g/L) * np.sin(x
1))
    dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * np.sin(x1-x2) + (g/L) * np.sin(x2))
    return [dx1, dx2, dx3, dx4]
```

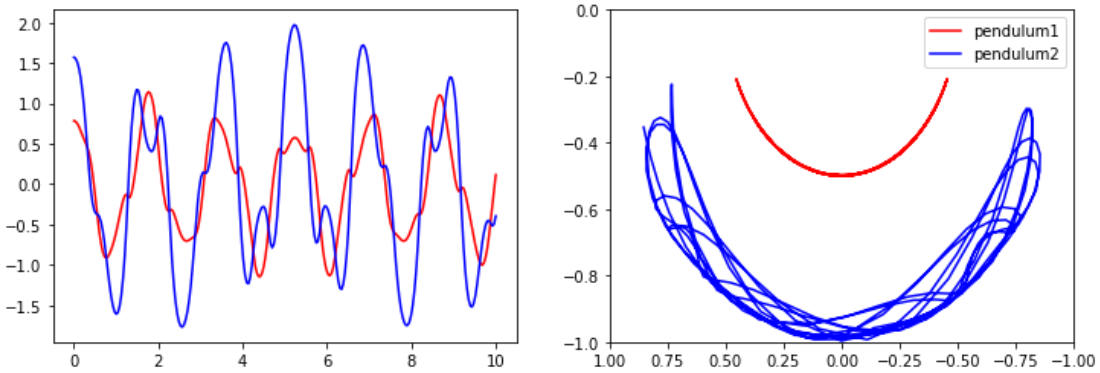
In [5]:
```
# on choisit une condition initiale
x0 = [np.pi/4, np.pi/2, 0, 0]
```

In [6]:
```
# les instants du temps: de 0 à 10 secondes
t = np.linspace(0, 10, 250)
```

In [7]:
```
# On résout
x = odeint(dx, x0, t)
print(x.shape)
```

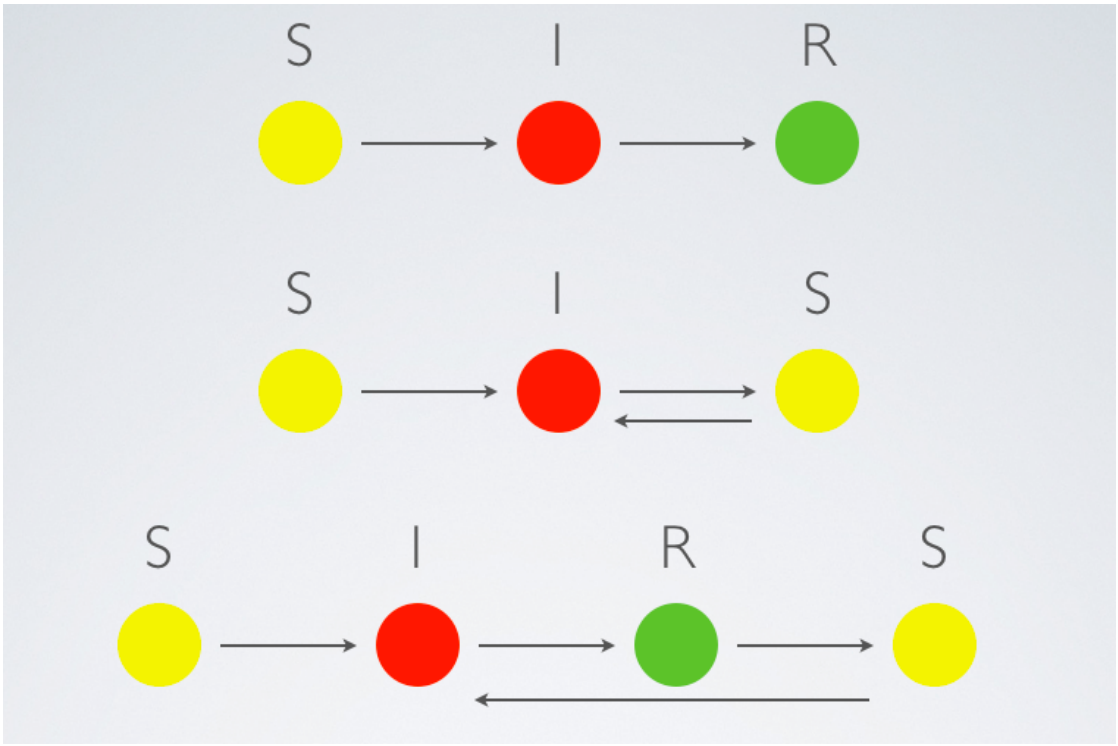
```
(250, 4)
```

In [8]:
```
# affichage des angles en fonction du temps
fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].plot(t, x[:, 0], 'r', label="theta1")
axes[0].plot(t, x[:, 1], 'b', label="theta2")
x1 = + L * np.sin(x[:, 0])
y1 = - L * np.cos(x[:, 0])
x2 = x1 + L * np.sin(x[:, 1])
y2 = y1 - L * np.cos(x[:, 1])
axes[1].plot(x1, y1, 'r', label="pendulum1")
axes[1].plot(x2, y2, 'b', label="pendulum2")
axes[1].set_ylim([-1, 0])
axes[1].set_xlim([1, -1])
plt.legend()
plt.show()
```



In [9]:
```
Image('sirs.png')
```

Out[9]:



On va appliquer ces connaissances pour étudier l'évolution du nombre d'infectés avec COVID19 en Tunisie. Nous avons choisi un modèle simple, connu et éprouvé, avec peu de paramètres. C'est le modèle SIR pour Susceptible/Infected /Recovered.

Le premier groupe est une population susceptible d'être infectée. C'est une population qui n'est pas encore infectée par la maladie, mais qui pourrait basculer à tout moment vers un autre état qui est I i.e. infecté. R représente le nombre de personnes ayant guéri (et qui sont supposé ne plus tomber malade).

$$\frac{dS}{dt} = -\frac{\beta IS}{N},$$

$$\frac{dI}{dt} = \frac{\beta IS}{N} - \gamma I,$$

$$\frac{dR}{dt} = \gamma I,$$

On est devant une utilisation très réduite de librairies. La fonction `odeint` permet de résoudre des systèmes d'EDO et ceci en se basant sur les algorithmes vus en cours. Parmi ces méthodes, on peut citer celle vue en cours comme Runge-Kutta d'ordre 2, ou d'autres méthodes plus sophistiquées. Voir par exemple https://github.com/scipy/scipy/blob/v0.19.0/scipy/integrate/odepack/readme (https://github.com/scipy/scipy/blob/v0.19.0/scipy/integrate/odepack/readme) pour plus d'informations sur les méthodes utilisées.

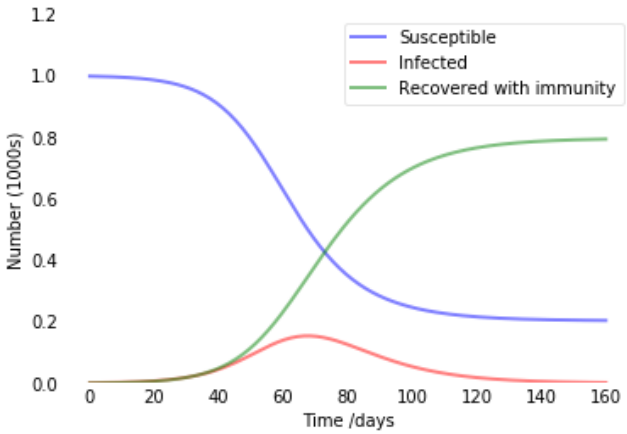
In [10]:
```
# Total population, N.
N = 1000
# Initial number of infected and recovered individuals, I0 and R0.
I0, R0 = 1, 0
# Everyone else, S0, is susceptible to infection initially.
S0 = N - I0 - R0
# Contact rate, beta, and mean recovery rate, gamma, (in 1/days).
beta, gamma = 0.2, 1./10
```

Les hypothèses du modèles sont les suivantes : on part d'une population totale de 1000 personnes. Et on suppose qu'il y a 1 seul infecté. Ici tout le monde est susceptible d'être infecté et aucun guéri pour le moment. Ca représente notre population à l'instant t=0.

On suppose deux valeurs : β et γ deux taux pour passer d'un état à un autre (voir équations plus haut). On verra plus bas comment trouver ces deux valeurs en se basant sur des données réelles.

In [11]:
```
# A grid of time points (in days)
t = np.linspace(0, 160, 160)

# The SIR model differential equations.
def deriv(y, t, N, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt

# Initial conditions vector
y0 = S0, I0, R0
# Integrate the SIR equations over the time grid, t.
ret = odeint(deriv, y0, t, args=(N, beta, gamma))
S, I, R = ret.T
```

In [12]:
```
# Plot the data on three separate curves for S(t), I(t) and R(t)
fig = plt.figure(facecolor='w')
ax = fig.add_subplot(111, axisbelow=True)
ax.plot(t, S/1000, 'b', alpha=0.5, lw=2, label='Susceptible')
ax.plot(t, I/1000, 'r', alpha=0.5, lw=2, label='Infected')
ax.plot(t, R/1000, 'g', alpha=0.5, lw=2, label='Recovered with immunity')
ax.set_xlabel('Time /days')
ax.set_ylabel('Number (1000s)')
ax.set_ylim(0,1.2)
ax.yaxis.set_tick_params(length=0)
ax.xaxis.set_tick_params(length=0)
ax.grid(b=True, which='major', c='w', lw=2, ls='-')
legend = ax.legend()
legend.get_frame().set_alpha(0.5)
for spine in ('top', 'right', 'bottom', 'left'):
    ax.spines[spine].set_visible(False)
plt.show()
```



Le pic d'infectés est atteint au bout de 70 jours. Regardons ensemble ce qui se passe si les chiffres émanent de donénes tunisiennes (annoncées par le ministère de la santé). Voir aussi ces liens pour plus de détails : https://covid19-map-tunisie.com/ (https://covid19-map-tunisie.com/) https://covid-19.tn/fr/statistiques-fr/ (https://covid-19.tn/fr/statistiques-fr/) https://ageos-tunisie.maps.arcgis.com/apps/opsdashboard/index.html#/e41552b7c6554dc7b6ab565273c24d71 (https://ageos-tunisie.maps.arcgis.com/apps/opsdashboard/index.html#/e41552b7c6554dc7b6ab565273c24d71) https://rpubs.com/inestej/COVID19_TN (https://rpubs.com/inestej/COVID19_TN)

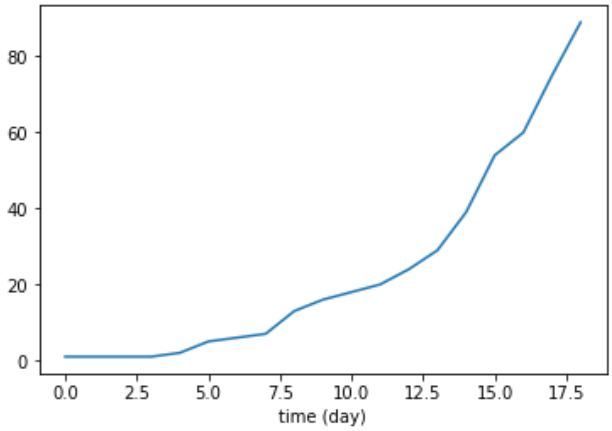
In [13]:
```
# Total population, N.
N = 11694720
# Initial number of infected and recovered individuals, I0 and R0.
I0, R0 = 1, 0
# Everyone else, S0, is susceptible to infection initially.
S0 = N - I0 - R0
```

C'est grâce à une approximation que nous obtenons le modèle qui colle au mieux aux données. Quelle est l'approximation choisie ici ?

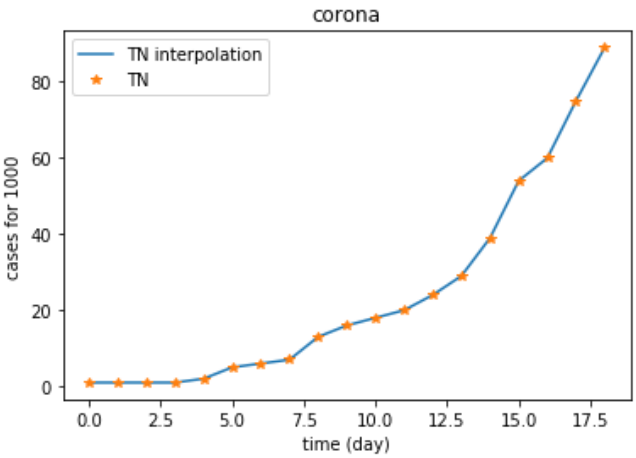
In [14]:
```
def f_Interpolate (x,datax,datay):
        f = interpolate.interp1d(datax, datay, fill_value="extrapolate")
        return(np.float(f(x)))
```

In [15]:
```
import scipy.sparse as sp # sparse matrix
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import  scipy.integrate as integ
import scipy as sci
from scipy import interpolate

import math as math
from sklearn.metrics import mean_squared_error

import pandas as pd
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

In [16]:
```
cases_Tunisia = [1,1,1,1,2,5,6,7,13,16,18,20,24,29,39,54,60,75,89]
timeSet = np.arange(0, len(cases_Tunisia),1)
date = timeSet.reshape(-1,1)
CasesSet = [f_Interpolate(t, date.reshape(len(cases_Tunisia)), cases_Tunisia) f
or t in timeSet]

p = plt.plot(timeSet, cases_Tunisia)
plt.xlabel('time (day)')
plt.show(p)
```

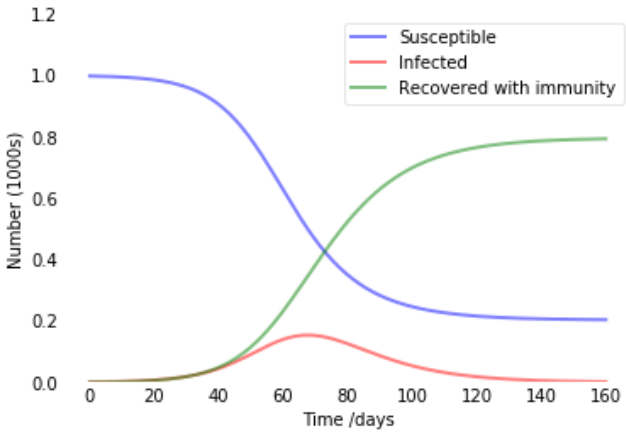


In [17]:
```
fig = plt.subplots()

line1,= plt.plot(timeSet, CasesSet ,label='TN interpolation')
line2,=plt.plot(date, cases_Tunisia,'*',label='TN')#,TempSet,PrSet ,TempSet,sel
f.pr_field,'g^')

plt.legend()
plt.xlabel('time (day)')
plt.ylabel('cases for 1000')
plt.title('corona')


plt.show()
```



In [18]:
```
# Contact rate, beta, and mean recovery rate, gamma, (in 1/days).
beta, gamma = 0.63, 0.37
```

In [19]:
```
# Plot the data on three separate curves for S(t), I(t) and R(t)
fig = plt.figure(facecolor='w')
ax = fig.add_subplot(111, axisbelow=True)
ax.plot(t, S/1000, 'b', alpha=0.5, lw=2, label='Susceptible')
ax.plot(t, I/1000, 'r', alpha=0.5, lw=2, label='Infected')
ax.plot(t, R/1000, 'g', alpha=0.5, lw=2, label='Recovered with immunity')
ax.set_xlabel('Time /days')
ax.set_ylabel('Number (1000s)')
ax.set_ylim(0,1.2)
ax.yaxis.set_tick_params(length=0)
ax.xaxis.set_tick_params(length=0)
ax.grid(b=True, which='major', c='w', lw=2, ls='-')
legend = ax.legend()
legend.get_frame().set_alpha(0.5)
for spine in ('top', 'right', 'bottom', 'left'):
    ax.spines[spine].set_visible(False)
plt.show()
```



Le nombre d'infectés atteint son maximum au bout de 2 mois.

Quelles sont les limites de ce modèles ?

1. Ici on a supposé que l état R ne peut pas retourner dans l état S ou I, ce qui n est pas forcément le cas pour COVID19 (voir littérature sur le sujet)
2. Il y a des modèles qui tiennenet compte de plus de paramètres, comme celui détaillé ci-dessous : S représente le nombre de susceptibles d'attraper COVID19, A ceux qui ont la maladie mais qui n'ont pas de symptomes (asymptomatiques), I les infectés, Q ceux qui sont en quarantaine, R ceux qui ont guéri et D le nombre de décès.

$$\frac{dS}{dt} = -f_1(t)SA - f_2(t)SI,$$

$$\frac{dA}{dt} = f_2(t)SI + (f_1(t)S - \beta - \gamma_1)A,$$

$$\frac{dI}{dt} = \beta A - (\gamma_2 + \tau + \delta)I,$$

$$\frac{dQ}{dt} = \gamma_1 A + \gamma_2 I - (\delta + \tau)Q,$$

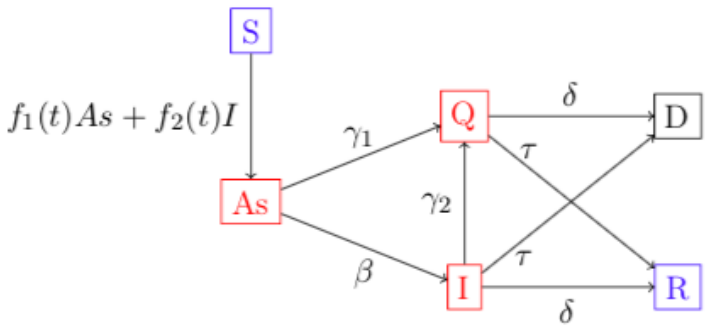
$$\frac{dR}{dt} = \tau(I + Q),$$

$$\frac{dD}{dt} = \delta(I + Q).$$

Quelles sont les valeurs choisies pour les paramètres $f_1, f_2, \beta, \gamma_1, \gamma_2, \tau, \delta$ de ce modèles (voir code ci-dessous) ?

In [20]:
```
Image('sirq.png')
```

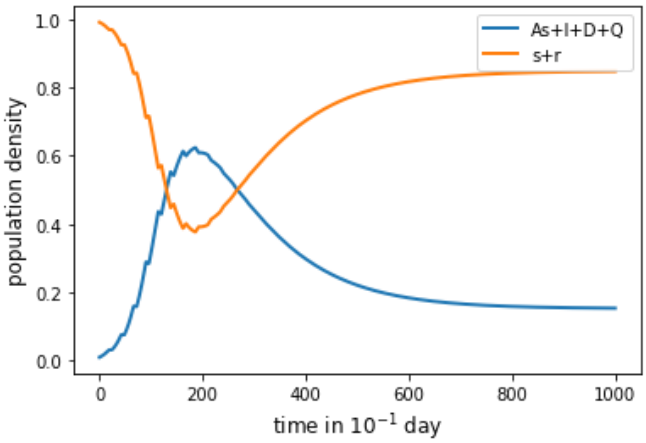
Out[20]:



La simulation suivante montre l'importance du confinement (Source `BenDhiafi et al., BIMS, IPT 2019`) :

In [21]:
```
beta=1/50#1./10#0.01 #taux de passage de l'etat asymptomatique a infecte
gama1=0.001#0.0 # taux de mise en quarantaine
gama2=10#0.0 # taux de mise en quarantaine



tau1= 1./150# taux de gerison des quarantaine
delta1=tau1/5#0.1 # taux de mortalité des infectées

tau2=tau1#0.5 # taux de gerison des quarantaine
delta2=delta1#0.4 # taux de mortalité des quarantene

a=.2#0.332
confinement_hour=6
k=24-confinement_hour
def f1(s,ass,i,t): # taux d'infecté par un assym
    tt=t/24-np.floor(t/24)
    if ((tt>=0) and (tt<k/24)):
        b=a/2.7
        y=float(b*s)
    else:
        y=0
    return y#a*s**2/(0.0*(i+ass)+s)

def f2(s,ass,i,t): # taux d'infecté par un infecté
    #tt=t/24-np.floor(t/24)
    #if ((tt>=0) and (tt<8/24)):
    b=a
    y=float(b*s)
    #else:
     #   y=0
    return y#a*s**2/(0.0*(i+ass)+s)
####################################""

def func(y,t):#S,beta,T_l,T_n): #Matrice de chaque stade
    ss,asss,ii,qq,dd,rr=y # param=[si,beta(si),T_l(si), T_n(si)]
    s2=-ii*f2(ss,asss,ii,t)-asss*f1(ss,asss,ii,t)
    ass2=ii*f2(ss,asss,ii,t) + asss*(f1(ss,asss,ii,t)-beta-gama1)
    i2= beta*asss-(delta2+gama2+tau2)*ii
    q2=gama1*asss+gama2*ii-(delta1+tau1)*qq
    d2=delta1*qq+delta2*ii
    r2=-(s2+ass2+i2+q2+d2) #tau1*qq+tau2*ii #

    return([s2,ass2,i2,q2,d2,r2])
```

In [22]:
```
tmax=1000# tmax=10 = 1 days #  in day
dt=float(1./24) # par tranche de 6 h

timeSet=np.arange(0,tmax,dt)

ass0=0.01#1/100
i0= 0
q0=0
d0=0
r0=0
s0=0.99 #1.-(ass0+i0+q0+d0+r0) #if s<0.2 else 0

##################
param=(beta,gama1,gama2)
y0=[s0,ass0,i0,q0,d0,r0]



#sol = ode(func).set_integrator('lsoda')#,atol=1e-10, rtol=1e-10
#sol.set_initial_value(y0,0.0).set_f_params(param) #param: doit etre un tableau
#sol=odeint(func, y0, temps, args=param)

#res=sol.integrate(10)
sol = integ.odeint(func, y0, timeSet)

#integ.odeint(func, 0, y0, 10)
#, rtol=0.001, atol=1e-06
#res1=[] # tableau des resultats
#timeSet=[] # tableau des temps
#while sol.successful() and sol.t < tmax:
#    ss=sol.integrate(sol.t+dt)
#    if sol.t>0:#tmax-36:
#        timeSet.append(sol.t+dt)
#        res1.append(ss)


#timeSet=sol.t
#s=sol.y[0]
#ass=sol.y[1]
#i=sol.y[2]
#q=sol.y[3]
#d=sol.y[4]
#r=sol.y[5]

s=np.array([sol[k][0] for k in range(len(timeSet))])
ass=np.array([sol[k][1] for k in range(len(timeSet))])
i=np.array([sol[k][2] for k in range(len(timeSet))])
q=np.array([sol[k][3] for k in range(len(timeSet))])
d=np.array([sol[k][4] for k in range(len(timeSet))])
r=np.array([sol[k][5] for k in range(len(timeSet))])

##################################################
fig, ax = plt.subplots()
#line= plt.plot(paramSet,erreur_adult,'ro', label='Dashes set retroactively',pa
ramSet,erreur_nymp,'bv', label='Dashes set retroactively',paramSet,erreur_larv,
'g^', label='Dashes set retroactively')
#line1,=ax.plot(month_field,Larv_total,label='Larvea')
#line2,=ax.plot(month_field,Nymp_total,label='Nymph')


#line1,=ax.plot(timeSet,s,label='s')
#line2,=ax.plot(timeSet,ass,label='ass')
line3,=ax.plot(timeSet,i+ass+d+q,lw=2,label='As+I+D+Q')
line4,=ax.plot(timeSet,s+r,lw=2,label='s+r')

#line4,=ax.plot(timeSet,q,label='q')
#line5,=ax.plot(timeSet,d,label='d')
#line6,=ax.plot(timeSet,r,label='r')
```



Pour plus de lecture

M. J. Keeling and P. Rohani, Modeling Infectious Diseases in Humans and Animals, Pr
inceton (2007).
R. M. Anderson and R. M. May, Infectious Diseases of Humans: Dynamics and Control,
OUP (1992).

Une équipe de BIMS travaille actuellement sur des modèles plus performants et plus précis. Voir par exemple https://www.gisagents.org/p/disease-modeling.html (https://www.gisagents.org/p/disease-modeling.html) (des stages proposés à Pasteur, à distance).