Résolution numérique d'équations différentielles (EULER, RK4 et ODEint de scipy)

Dans une épreuve IMSP, on vous présentera le plus souvent une résolution progressive du problème en définissant successivement des fonctions qui ne réalisent qu'une …fonction. Puis le problème sera composé de ces poupées russes s'emboitant jusqu'à la matriochka finale.
La résolution d'équations différentielles ordinaires (ODE) par intégration numérique ne devrait pas échapper à cette règle méthodologique à part peut-être la seule méthode d'Euler.
Cette présentation peut parfois nuire à la visibilité complète de la méthode à la fois dans sa globalité et dans son détail. Parfois même les concepteurs de sujets utilisent la boite noire « scipy.integrate.odeint » dont il convient alors de découvrir la syntaxe.

Je vous propose de présenter sur deux exemples issus du cours de physique (thermique et méca du point matériel) des codes python développant successivement Euler, Runge Kutta et odeint du module Scipy sur des équations différentielles NON linéaires. Le premier exemple a l'intérêt de posséder une solution par intégration exacte et le second adapte les méthodes à une équation différentielle du second ordre.

1.Equation différentielle du premier ordre non linéaire (avec solution littérale exacte)

1.1.Modélisation : Matériau conducteur thermique non-linéaire (conductivité thermique dépendant de T)

MUR EN REGIME PERMANENT AVEC CONDUCTIVITE VARIABLE

Pour de nombreux matériaux soumis à des écarts de température importants, il faut prendre en compte la variation de la conductivité avec la température. Cette variation est donnée généralement par une loi linéaire

$$\lambda = \lambda_0 (1 + \alpha (T - T_0))$$

On considère une plaque d'épaisseur e soumise sur ses deux faces à un contact parfait avec deux milieux de températures $T(0) = T_0$ $T(e) = T_e$

En supposant que la conductivité du matériau constitutif de la plaque varie linéairement avec la température, déterminer la répartition interne des températures, ainsi que la valeur du flux de chaleur traversant cette paroi.

A.N. $e = 5$ cm $\lambda_0 = 1$ W/m.K $\alpha = 2.10^{-3}$ °C^{-1}
$T_0 = 50$°C $T_e = 550$°C.

Cette situation (et ces valeurs numériques) conduisent à une valeur de flux de chaleur (surfacique !) indépendante de x (position dans le matériau) et négative (flux positif dans le sens des hautes vers les basses températures donc dans le sens -x) : $\varphi = -15 kW.m^{-2} = -15000 W.m^{-2}$ (que l'on peut obtenir par intégration entre les deux surfaces extrêmes et la connaissance de leurs températures)
et à une expression numérique exacte de : $\theta(x) \equiv T(x) - T_0 = 500.\left(\sqrt{1+60x} - 1\right)$ avec x en m

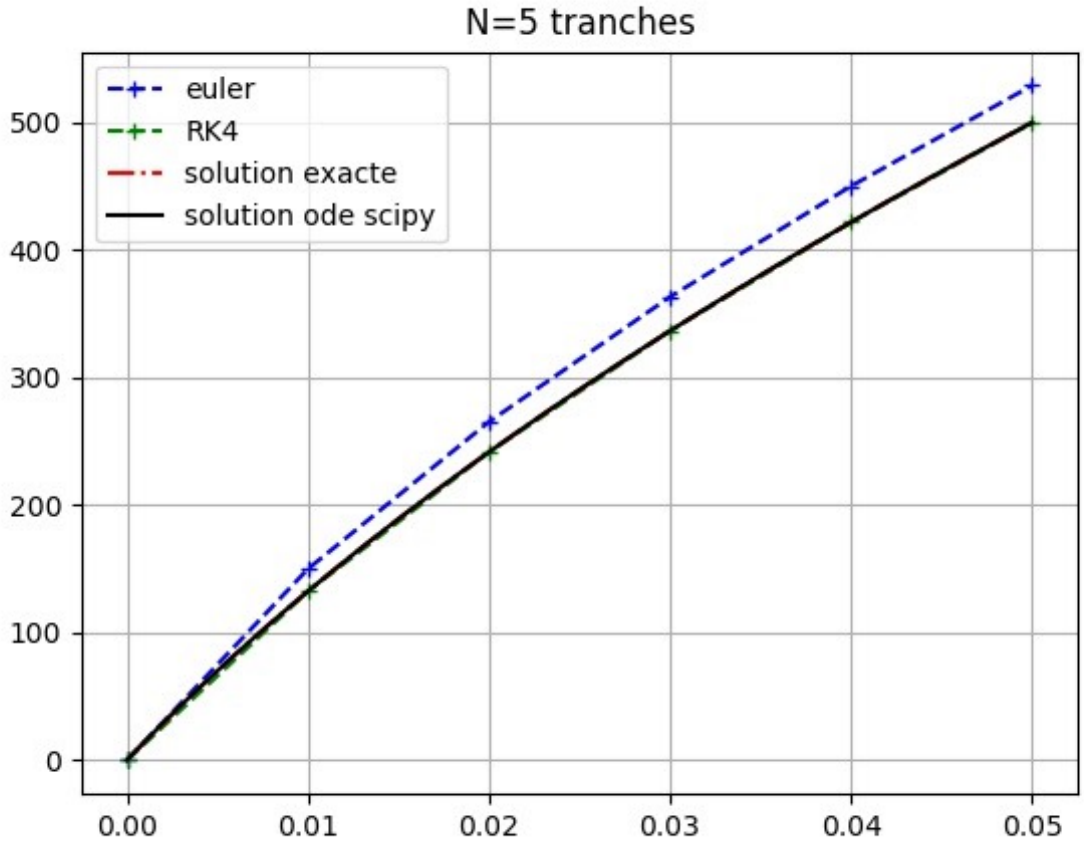
En utilisant l'hypothèse de régime permanent, écrire l'équation diférentielle en $\theta(x) \equiv T(x) - T_0$ avec les paramètres caractéristiques constants φ, λ_0 et α.

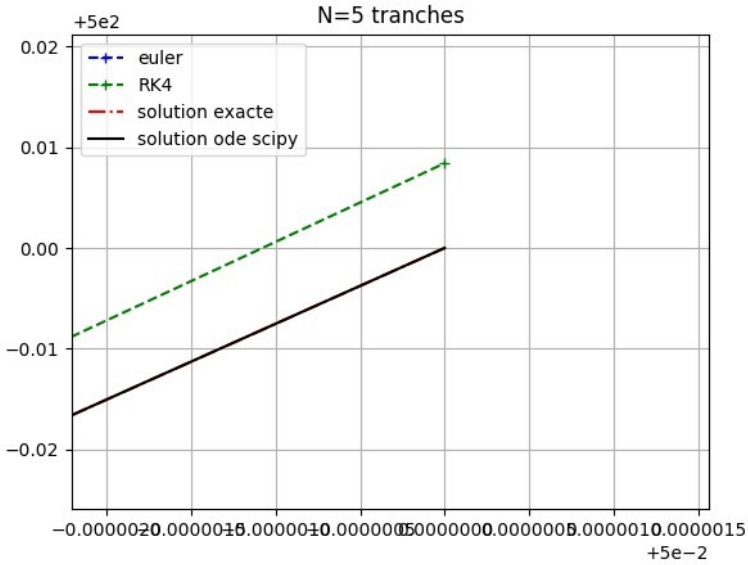
1.2.Code PYTHON

N'hésitez pas à commenter le code à droite !

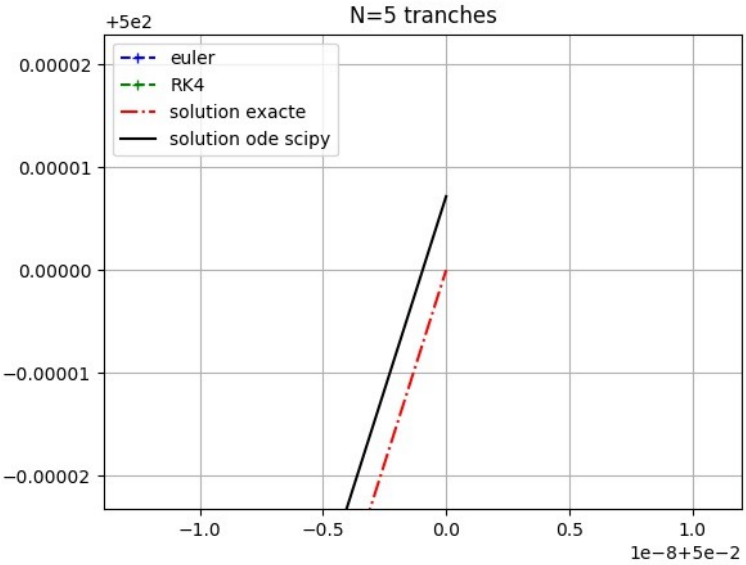
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

phi=-15000
alpha=0.002
e=0.05
Nx=5

dx=e/Nx
x=np.arange(0,e+dx,dx)
x2=np.arange(0,e+dx/10,dx/10)
theta=np.zeros(Nx+1)

plt.figure()
for i in range(Nx):
    theta[i+1]=theta[i]-phi*dx/(1+alpha*theta[i])
ecarteuler=theta[Nx]-500
print ("écart euler",ecarteuler)
plt.plot(x,theta,'b+--',label="euler")


for i in range(Nx):
    k1=-phi/(1+alpha*theta[i])
    k2=-phi/(1+alpha*(theta[i]+k1*dx/2))
    k3=-phi/(1+alpha*(theta[i]+k2*dx/2))
    k4=-phi/(1+alpha*(theta[i]+k3*dx))
    theta[i+1]=theta[i]+(dx/6)*(k1+2*k2+2*k3+k4)
ecartRK=theta[Nx]-500
print ("écart RK",ecartRK)
plt.plot(x,theta,'g+--',label="RK4")

thetaexact=500*(np.sqrt(1+60*x2)-1)
plt.plot(x2,thetaexact,'r-.',label="solution exacte")


def derivee (Theta,x2):
    return -phi/(1+alpha*Theta)
solutionscipy=odeint(derivee,0,x2)
print ("Valeur obtenues par ODE de scipy ",solutionscipy)
plt.plot(x2,solutionscipy,'k',label="solution ode scipy")
plt.grid()
plt.legend(loc='upper left')
plt.title("N=%1.0f tranches"%Nx)
plt.show()
```

1.3.Visualisation et comparaison des 4 profils des températures





ZOOM



SUPER ZOOM

2.Equation différentielle du second ordre non linéaire (sans solution littérale exacte)

2.1.Modélisation : Fluctuations d'éloignement de Mercure relativement au Soleil sur son orbite elliptique de forte excentricité et sur plusieurs périodes.

L'obtention de l'équation polaire exacte de la trajectoire ne nous affranchit pas de la résolution temporelle quand il s'agit de trouver la position instantanée d'une planète sur son orbite (pour justifier une rétrogradation relative par exemple). La trajectoire et la périodicité du mouvement permet bien sûr de se limiter à une révolution mais nous chercherons ici au contraire à visualiser l'écart cumulé au bout de plusieurs révolutions lors de l'utilisation de méthodes numériques que nous allons comparer.

i. Les équations différentielles horaires d'une planète du système solaire s'écrivent :

$$\frac{\partial^2 r}{\partial t^2} = -\frac{G.M}{r^2} + \frac{G.M.a.(1-e^2)}{r^3} \text{ et } \frac{\partial \theta}{\partial t} = \frac{\sqrt{G.M.a.(1-e^2)}}{r^2}$$

avec : a le demi-grand axe de l'ellipse
e : l'excentricité
G : la constante de gravitation universelle
M: la masse du soleil

Mercure : a := 57909081161; G := $6.674287*10^{11}$; M := $1.9891*10^{30}$; e := 0.2056307

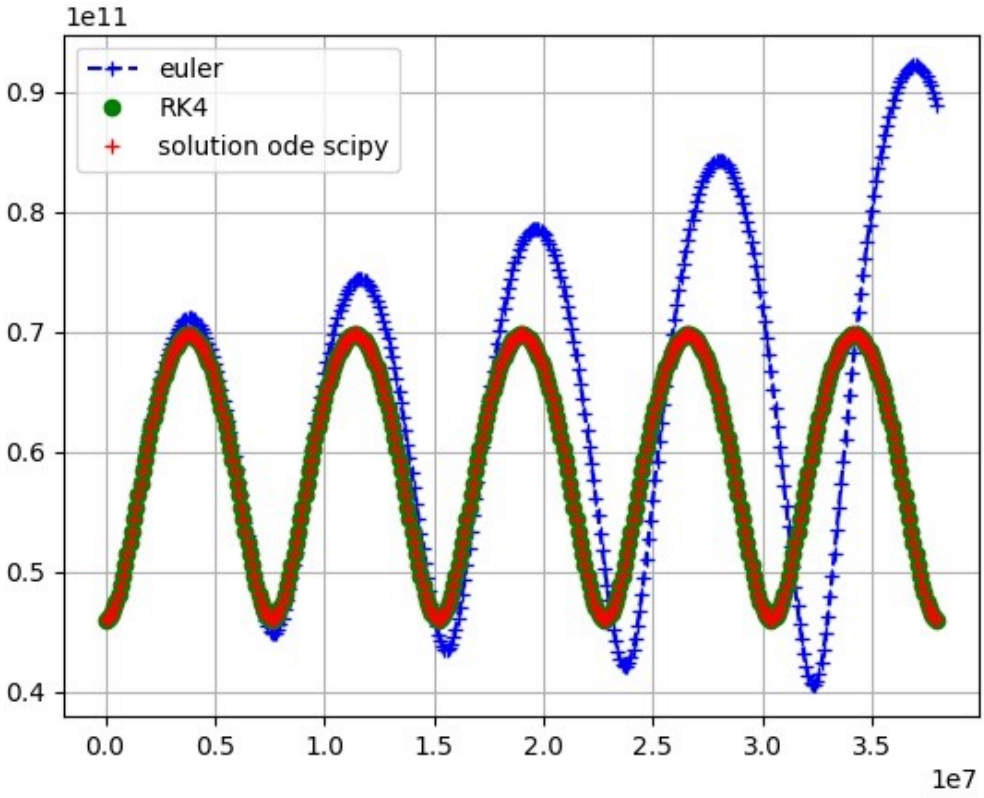
2.2. Code PYTHON

N'hésitez pas à commenter le code à droite !

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

a=57909176000
e=0.2056307
r0=a*(1-e)
rmin=a*(1+e)
G=6.67408e-11
M=1.9884e30
GM=G*M
dt=100000
duree=20*88*3600*24
N=int(duree/dt)
t=np.arange(0,duree,dt)
dist=np.concatenate(([r0],N*[1]))
vitrad=np.zeros(N+1)
accrad=np.zeros(N+1)

plt.figure()

for i in range(N):
    accrad[i]=GM*(((-1/(dist[i]**2))+(a*(1-e**2)/(dist[i]**3))))
    dist[i+1]=dist[i]+(vitrad[i])*dt+(accrad[i]*(dt**2))/2
    vitrad[i+1]=accrad[i]*dt+vitrad[i]
plt.plot(t,dist,'b+--',label="euler")
```

```
dist=np.concatenate(([r0],N*[1]))
vitrad=np.zeros(N+1)
a1,v1=np.zeros(N+1),np.zeros(N+1)
a2,v2=np.zeros(N+1),np.zeros(N+1)
a3,v3=np.zeros(N+1),np.zeros(N+1)
a4,v4=np.zeros(N+1),np.zeros(N+1)

for i in range(N):
    a1[i]=GM*((-1/(dist[i]**2))+(a*(1-e**2)/(dist[i]**3)))
    v1[i]=vitrad[i]
    a2[i]=GM*((-1/((dist[i]+v1[i]*dt/2)**2))+(a*(1-e**2)/((dist[i]+v1[i]*dt/2)**3)))
    v2[i]=vitrad[i]+a1[i]*dt/2
    a3[i]=GM*((-1/((dist[i]+v2[i]*dt/2)**2))+(a*(1-e**2)/((dist[i]+v2[i]*dt/2)**3)))
    v3[i]=vitrad[i]+a2[i]*dt/2
    a4[i]=GM*((-1/((dist[i]+v3[i]*dt)**2))+(a*(1-e**2)/((dist[i]+v3[i]*dt)**3)))
    v4[i]=vitrad[i]+a3[i]*dt
    vitrad[i+1]=vitrad[i]+(dt/6)*(a1[i]+2*a2[i]+2*a3[i]+a4[i])
    dist[i+1]=dist[i]+(dt/6)*(v1[i]+2*v2[i]+2*v3[i]+v4[i])
plt.plot(t,dist,'go',label="RK4")

def deriveeseconde (rpoint,r,t):
    return -(GM/(r**2))+(GM*r0*(1+e)/r**3)
def vecteur2derivees (u,t):
    return [u[1],deriveeseconde(u[1],u[0],t)]
solutionscipy=odeint(vecteur2derivees,[r0,0],t)
plt.plot(t,solutionscipy[:,0],'r+',label="solution ode scipy")

plt.grid()
plt.legend(loc='upper left')
plt.show()
```

2.3. Visualisation et comparaison des 3 intégrations numériques



Sur 20 périodes avec un pas d'environ 28 heures terrestres soit (1/76 de période mercurienne)…

<u>…Mercure se libère de l'attraction solaire d'après la méthode d'EULER !</u>