Librairie mathématique et affichage

Numpy: Array

```
import numpy as np

va=np.array([1,4,8,9])
vb=np.array([-4.1,2.2,1,-5])

vc=va+vb

print(vc)
```

array ressemble aux listes
spécialisé pour les nombres

Numpy: Array

```
va=np.array([1,4,8,9])
vb=np.array([-4.1,2.2,1,-5])
```

```
va+vb
va-vb
2*va
vb*4
np.vdot(va,vb)
```

addition
soustraction
multiplication par un scalaire
produit scalaire
...

Rem. On ne mélange pas "mot" et nombre dans un array

Linspace

```
a,b=1.1,4.8
N=8

x=np.linspace(a,b,N)

print(x)
```

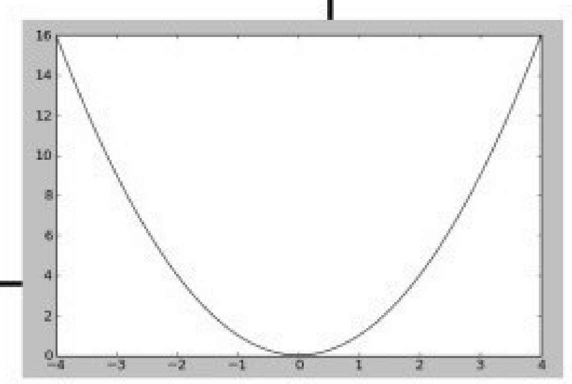
Vecteur uniformément réparti entre [a,b] avec N échantillons

Affichage

Combinaison array + affichage

```
import numpy as np
import matplotlib.pyplot as plt

a,b=-4,4
N=200

x=np.linspace(a,b,N)
y=x**2

plt.plot(x,y)
plt.show()
```

élève au carré
élément à élément



Array-range

arange: similaire à range

```
v=np.arange(1,8,2)
print(v)
```

1 3 5 7

Slicing

```
print(a)
print(a[2:4])
print(a[2:])
print(a[:5])
print(a[1:6:2])
print(a[::3])
```


Vecteurs particuliers

```
va=np.zeros(5)
vb=np.ones(6)
```

va 0 0 0 0 0

vb 1 1 1 1 1 1

Applications

Soit la droite D passant par x0 et de vecteur directeur u

x0=(1,2)
u=(1,4)

Calculer un, le vecteur directeur unitaire de même direction que u

(norme: from numpy.linalg import norm)

Soit a=x0-2un, b=x0+2un

Afficher la droite ab
Afficher un point en x0

Applications

Soit p=(3,3)

Calculer L=<ap,un>

Calculer la projection orthogonale p_proj de p sur la droite D
p_proj=a+L un

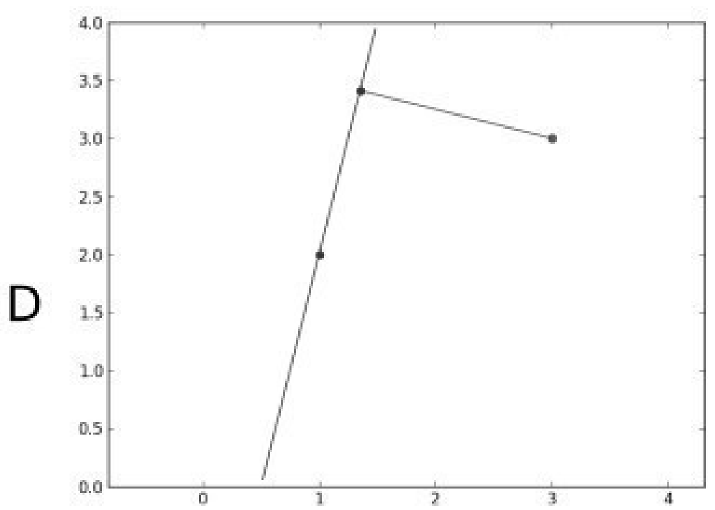
Afficher p et p_proj

Notez que les axes ne sont pas orthogonaux

Ajouter: plt.axis("square")

Afficher le segment [p,p_proj]

Calculer la distance entre p et la droite D



Applications

Soit C le cercle de centre x0=(1,2) et de rayon 4

Construire le vecteur theta contenant N échantillons
également répartis sur [0,2 pi]

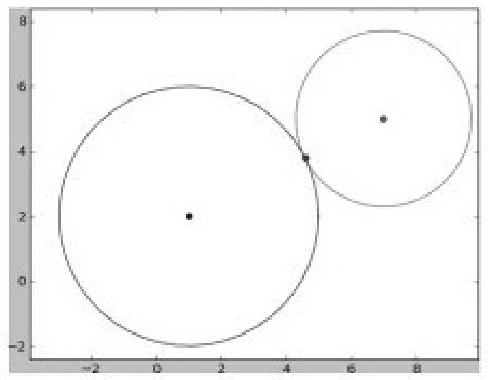
(pi=np.pi)

Stocker dans les vecteurs cx, et cy les coordonnees
de N échantillons du cercle C

Tracer le cercle C

Soit C' un cercle tangent à C de centre x0'=(7,5)

Calculer le rayon de C'



Tracer C', les points x0 et x1 et le point d'intersection entre C et C'

Applications

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm

x0=np.array([1,2])
r=4

N=50
theta=np.linspace(0,2*np.pi,N)

cx=r*np.cos(theta)+x0[0]
cy=r*np.sin(theta)+x0[1]

x1=np.array([7,5])
r1=norm(x0-x1)-r

c1x=r1*np.cos(theta)+x1[0]
c1y=r1*np.sin(theta)+x1[1]


inter=x0+(x1-x0)/norm(x1-x0)*r

plt.plot(cx,cy)
plt.plot(c1x,c1y,"g")
plt.plot(x0[0],x0[1],"bo")
plt.plot(x1[0],x1[1],"go")
plt.plot(inter[0],inter[1],"ro")

plt.axis('equal')
plt.show()
```

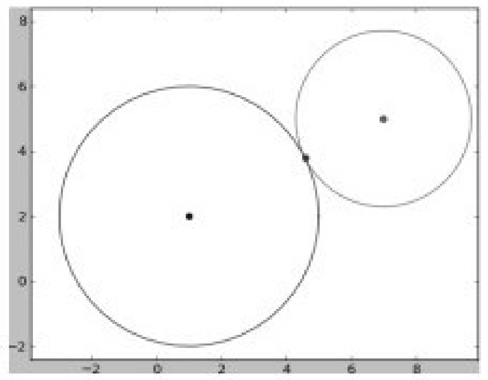
re x0=(1,2) et de rayon 4

eta contenant N échantillons
[0,2 pi]

(pi=np.pi)

rs cx, et cy les coordonnees
rcle C

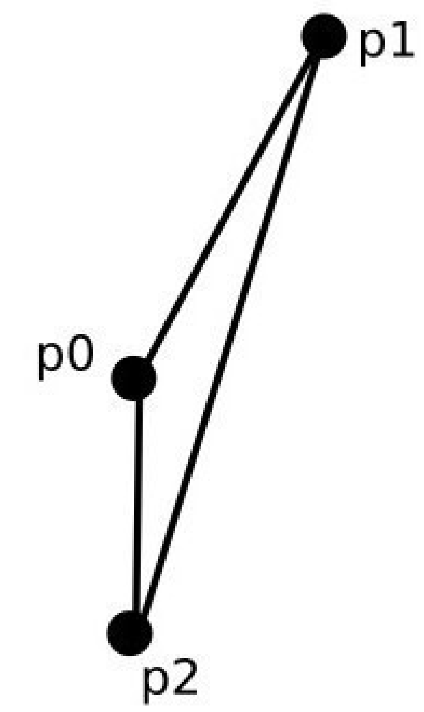
nt à C de centre x0'=(7,5)



Tracer C', les points x0 et x1 et le point d'intersection entre C et C'

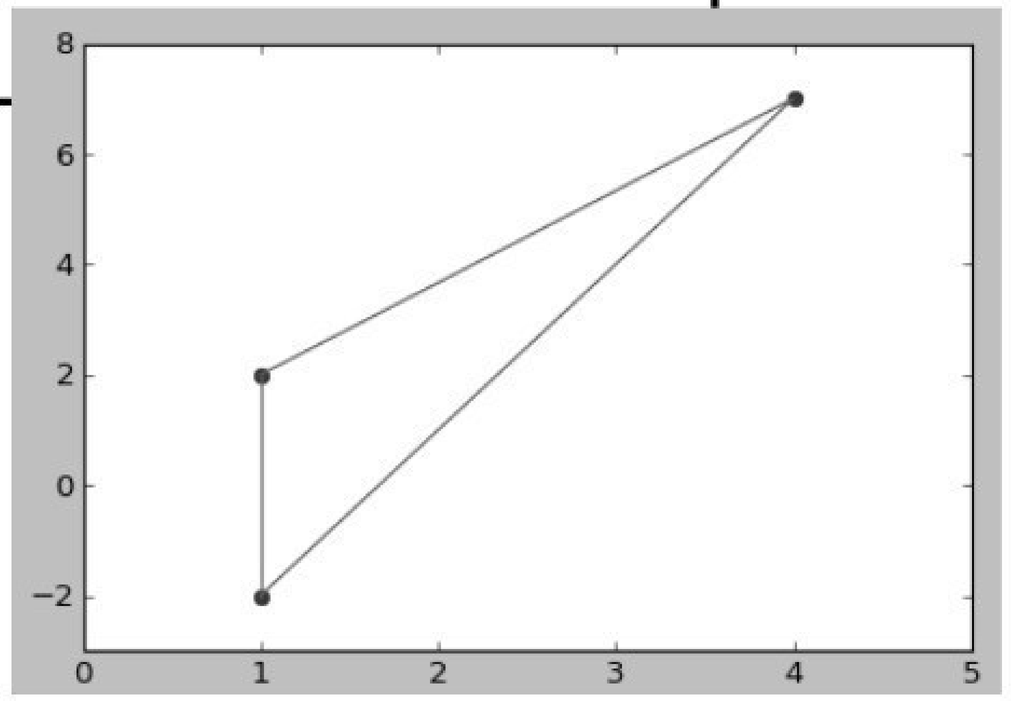
Vecteurs multi-dimensionnels

```
p0=np.array([1,2])
p1=np.array([4,7])
p2=np.array([1,-2])

triangle=np.array([p0,p1,p2])

print(triangle[2,1]) #coord-y p2

print(triangle[0,:]) #point 0
print(triangle[1,:]) #point 1
print(triangle[2,:]) #point 2

print(triangle[:,0]) #coord-x
print(triangle[:,1]) #coord-y
```



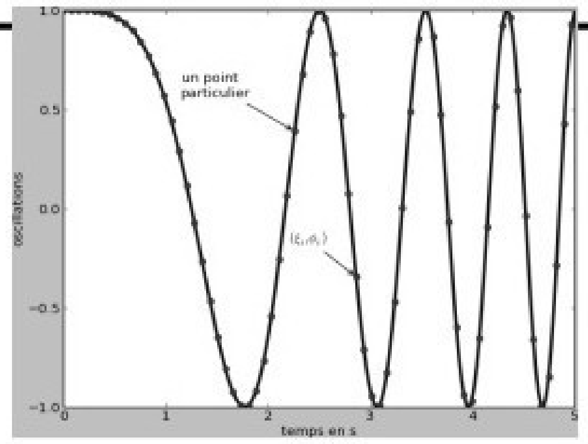
Vecteurs multi-dimentionnels

```
plt.plot(triangle[:,0],triangle[:,1],"ro")
plt.plot(triangle[:,0],triangle[:,1],"g-")
plt.plot([triangle[0,0],triangle[2,0]],
         [triangle[0,1],triangle[2,1]],"g-")
plt.axis([0,5,-3,8])
plt.show()
```

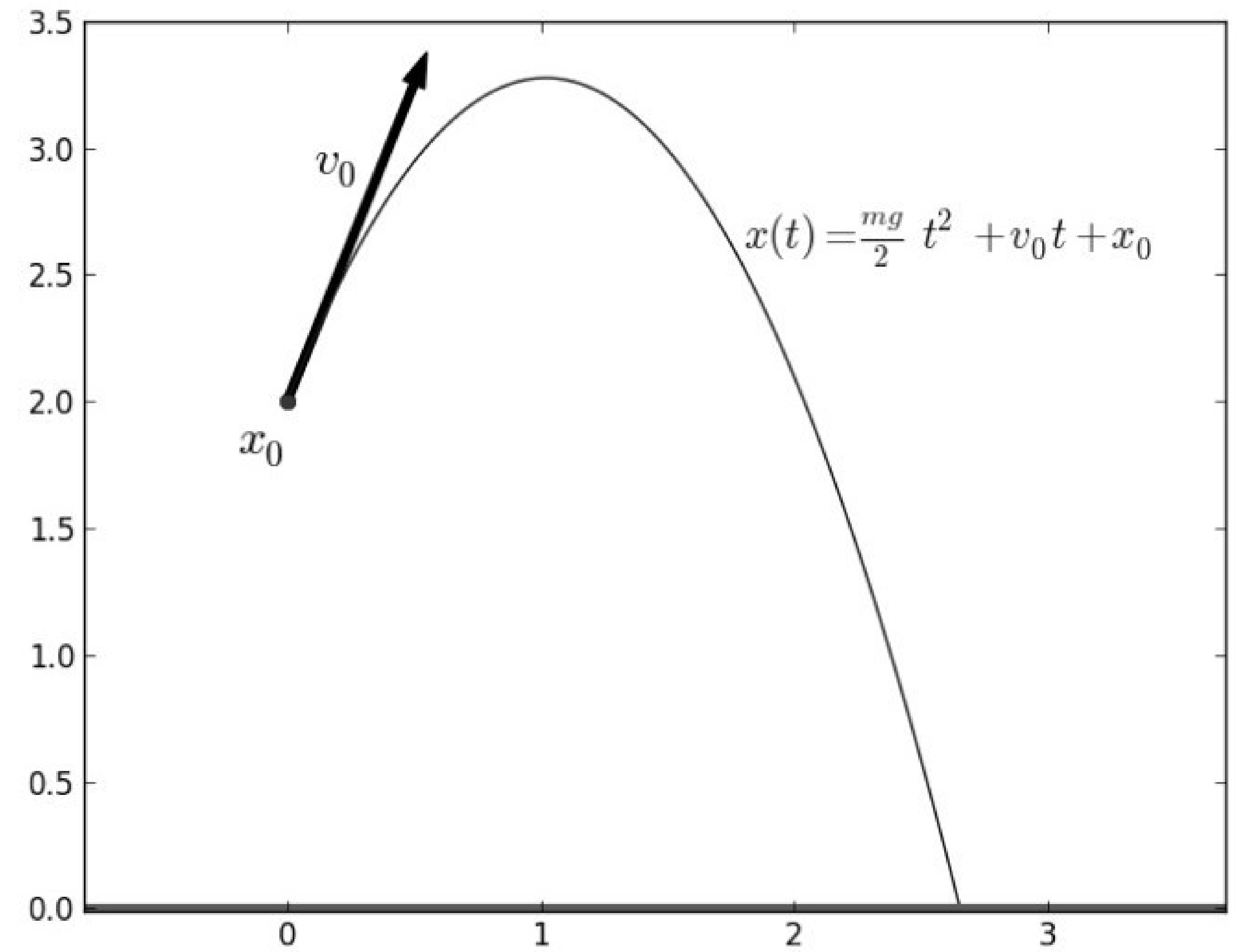


Embellissement graphique

```
N=200
x=np.linspace(0,5,N)
y=np.cos(x*x)



plt.plot(x,y,linewidth=3)
plt.plot(x[::3],y[::3],"ro")
plt.xlabel("temps en s")
plt.ylabel("oscillations")

plt.annotate('un point \nparticulier', xy=(x[90], y[90]),  xycoords='data',
                xytext=(-100, 30), textcoords='offset points',
                arrowprops=dict(arrowstyle="->")
                )

plt.annotate(u'$(\\xi_i,\\phi_i)$', xy=(x[114], y[114]),  xycoords='data',
                xytext=(-60, 30), textcoords='offset points',
                arrowprops=dict(arrowstyle="->")
                )
plt.show()
```

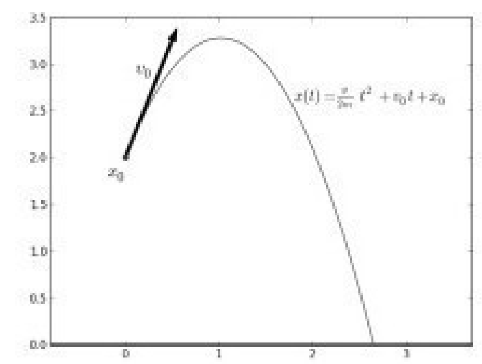


Application, afficher:



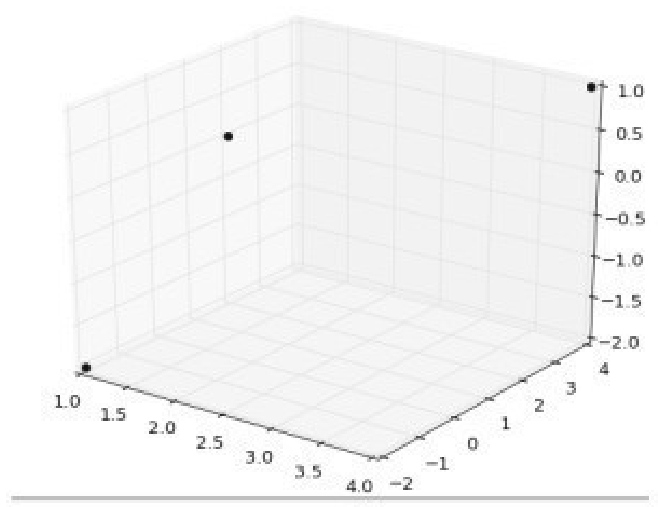
Application, afficher:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm
from mpl_toolkits.mplot3d import Axes3D



x0=np.array([0,2])
v0=np.array([2,5])
g=np.array([0,-9.8])

N=100
t=np.linspace(0,1.4,N)

x=np.array([np.zeros(N),np.zeros(N)]);


for dim in range(2):
  x[dim]=1/2*g[dim]*t**2 + v0[dim]*t+x0[dim]

#plt.arrow(0,0,0.5,0.5,width=0.1)
plt.arrow(x0[0],x0[1],v0[0]/4,v0[1]/4,width=0.03,color="black")

plt.plot(x[0,:],x[1,:],linewidth=1)
plt.plot([-5,5],[0,0],"g-",linewidth=3)
plt.plot(x0[0],x0[1],"ro")
plt.text(x0[0]-0.2,x0[1]-0.2,u"$x_0$",fontsize=20)
plt.text(x0[0]+0.1,x0[1]+0.9,u"$v_0$",fontsize=20)
plt.text(1.8,2.6,u"$x(t)=\\frac{g}{2m}\\,t^2+v_0 t+x_0$",fontsize=17,color="blue")
plt.axis('equal')
plt.axis([-0.1,3,-0.01,3.5])
plt.show()
```



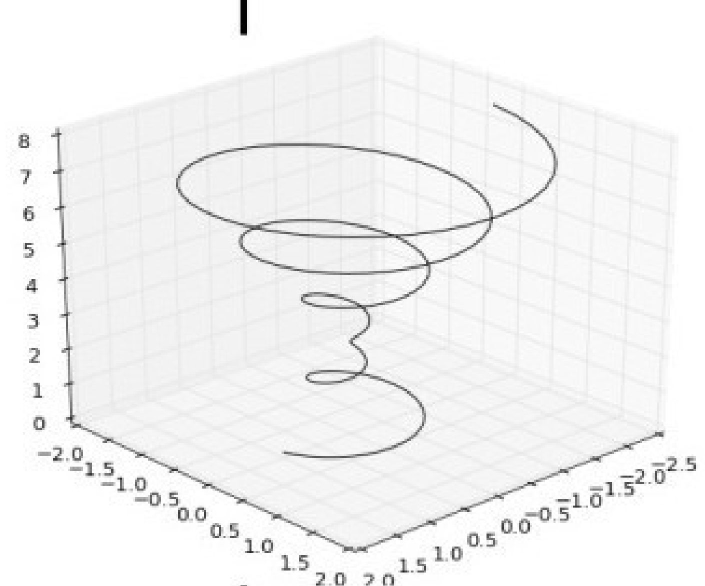
Courbe et points 3D

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

p0=np.array([1,2,0])
p1=np.array([4,4,1])
p2=np.array([1,-2,-2])

triangle=np.array([p0,p1,p2])

x=triangle[:,0]
y=triangle[:,1]
z=triangle[:,2]

print(x,y,z)

fig=plt.figure()
axes3d=fig.gca(projection='3d')
axes3d.plot(triangle[:,0],triangle[:,1],triangle[:,2],"o")
plt.show()
```



Courbe et points 3D

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

N=200
t=np.linspace(0,5*np.pi,N)

x=(1-t/5)*np.cos(2*t)
y=(1-t/5)*np.sin(2*t)
z=t/2

p=np.array([x,y,z])

fig=plt.figure()
axes3d=fig.gca(projection='3d')
axes3d.plot(p[0,:],p[1,:],p[2,:],"-")
plt.show()
```



Cas d'application: Equations différentielles

Dérivée discrète

Soit une fonction réelle dérivable f

La dérivée discrète de f au point x est calculable par la relation

$$\frac{f(x+\epsilon)-f(x)}{\epsilon}$$

Application:

Coder la fonction: f(x)=sin(x)

Calculer la dérivée numérique en 0

Comparer à la vraie valeur (pour différentes valeurs de ϵ)

Dérivée discrète

Rem. On peut définir l'application

$$(\mathcal{F}\times R)\to\mathbb{R}$$
$$D:(f,x)\mapsto f'(x)$$

```
def f(x):
  return np.sin(x)

def D(f,x):
  epsilon=1e-6
  df=(f(x+epsilon)-f(x))/epsilon

  return df

x=0
print(D(f,0))
print(D(f,0.5))
print(D(f,0.8))
```

f est un argument de D

Dérivée discrète

Application:

$$g(x)=\tanh\left(\sin(x)+\tanh(x)\right)$$
$$f(x)=(g\circ g\circ g)(x)$$

Afficher f et f' sur [-30,30]

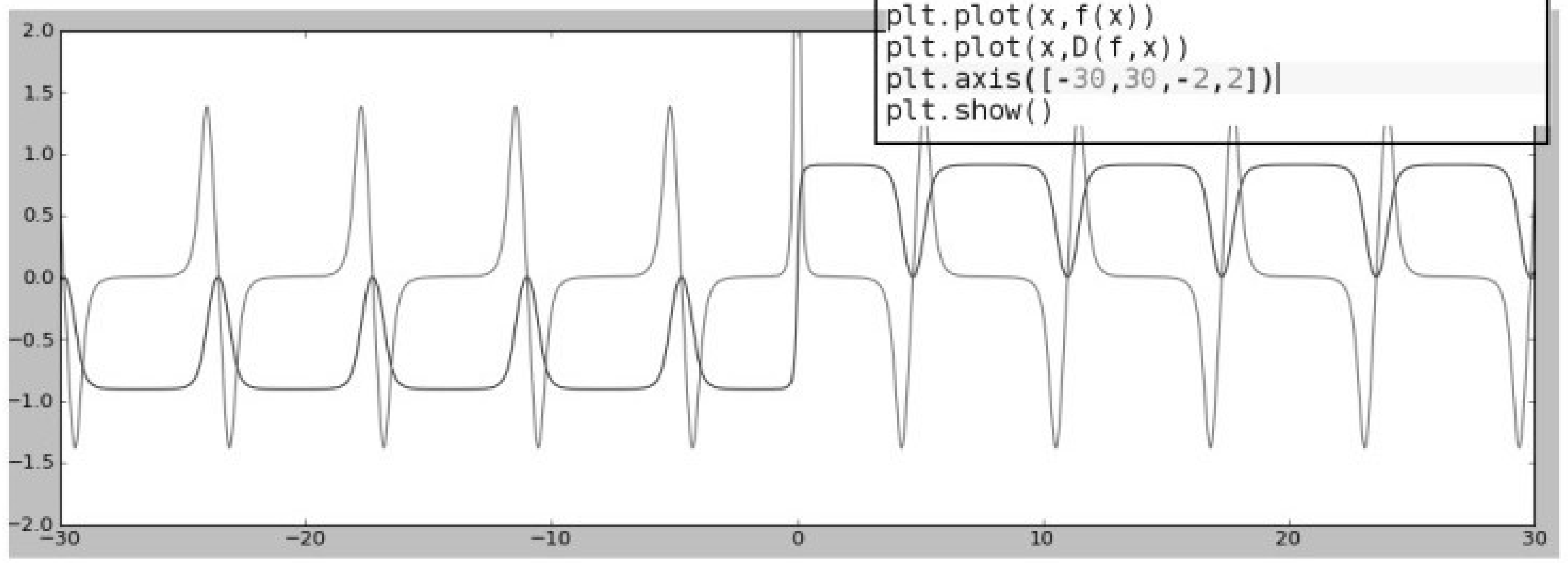
Dérivée discrète

Application:

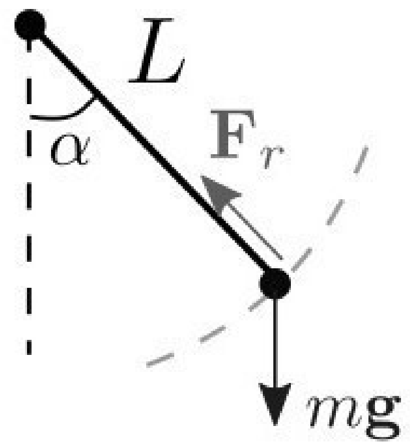
$$g(x)=\tanh\left(\sin(x)+\tanh(x)\right)$$
$$f(x)=(g\circ g\circ g)(x)$$

Afficher f et f' sur [-30,30]

```
def g(x):
  return np.tanh(np.sin(x)+np.tanh(x))

def f(x):
  y=x
  for k in range(3):
    y=g(y)

  return y

def D(f,x):
  epsilon=1e-6
  df=(f(x+epsilon)-f(x))/epsilon

  return df

N=1500
x=np.linspace(-30,30,N)

plt.plot(x,f(x))
plt.plot(x,D(f,x))
plt.axis([-30,30,-2,2])
plt.show()
```



Pendule oscillant



$$\alpha'(t) = \omega(t)$$

Equation de la dynamique:

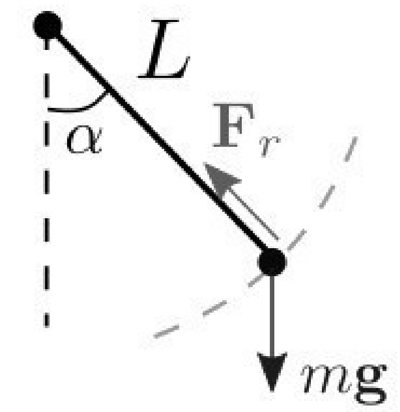
$$\omega'(t) = -\frac{g}{L}\sin(\alpha(t))$$

Pas de solution analytique simple pour $\alpha(t)$ grand

Si frottements, ou pendule couplé, pas de solution analytique du tout

On discrétise la dérivée
(on intègre numériquement suivant t)

Pendule oscillant



$$\alpha'(t) = \omega(t)$$

Equation de la dynamique:

$$\omega'(t) = -\frac{g}{L}\sin(\alpha(t))$$

Equation discrétisée:

$$\alpha^k = \alpha(t + \Delta t)$$
$$\omega^k = \omega(t + \Delta t)$$

$$\left|\begin{array}{l} \omega^{k+1} = \omega^k - \Delta t \dfrac{g}{L}\sin\left(\alpha^k\right) \\ \alpha^{k+1} = \alpha^k + \Delta t\, \underbrace{\omega^{k+1}}_{\text{non divergence}} \\ \alpha^0 = \alpha_0 \\ \omega^0 = \omega_0 \end{array}\right.$$

Afficher $\alpha(t)$ en fonction de t

Considérer $\alpha_0 \simeq \pi$

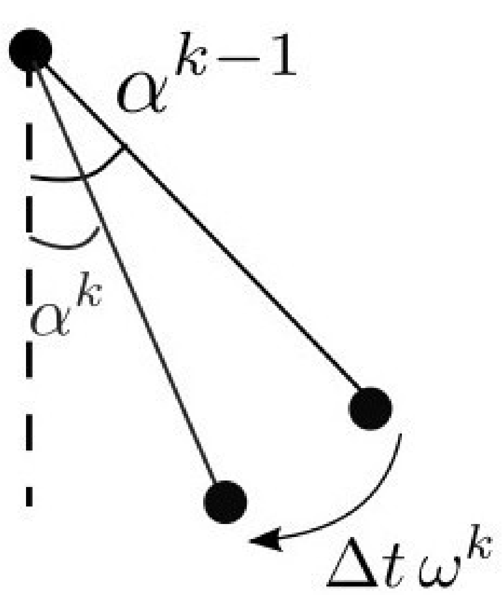
Pendule oscillant

Algorithme:

Initialiser α^0 , ω^0 0

Pour tous k

$$\left|\begin{array}{l} \omega^k = \omega^{k-1} - \Delta t\, \dfrac{g}{L}\sin\left(\alpha^{k-1}\right) \\ \alpha^k = \alpha_{k-1} + \Delta t\, \omega^k \\ \text{Afficher}(t = k\,\Delta t\ ,\ \alpha^k) \end{array}\right.$$



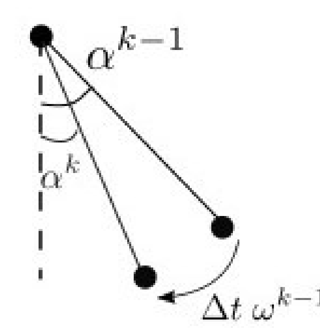
Pendule oscillant

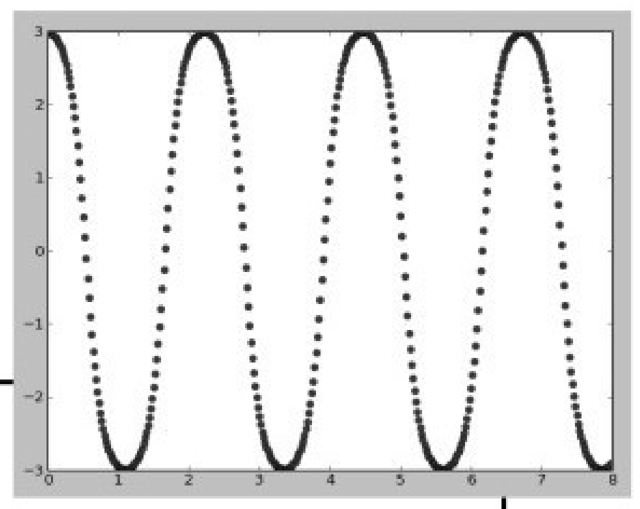
Algorithme:

Initialiser α^0 , ω^0 0

Pour tous k

$$\left|\begin{array}{l} \omega^k = \omega^{k-1} - \Delta t\, \dfrac{g}{L}\sin\left(\alpha^{k-1}\right) \\ \alpha^k = \alpha_{k-1} + \Delta t\, \omega^{k-1} \\ \text{Afficher}(t = k\,\Delta t\ ,\ \alpha^k) \end{array}\right.$$





```
g=9.8
L=0.2
dt=0.02

omega=0
alpha=0.6*np.pi

omega_previous=omega
alpha_previous=alpha

for k in range(900):
    omega=omega_previous-dt*g/L*np.sin(alpha_previous)
    alpha=alpha_previous+omega*dt

    plt.plot(k*dt,alpha,"ro")

    omega_previous=omega
    alpha_previous=alpha

plt.show()
```

Matrices

Matrices

```
import numpy as np

A=np.matrix([[1,2,3],[4,5,6]])

print(A)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Matrices

Produit matriciel

```
import numpy as np

A=np.matrix([[1,2,3],
             [4,5,6]])

B=np.matrix([[1,4],
             [1,2],
             [3,10]])

C=A*B

print(C)
```


Matrices

Matrices carrées

```
A=np.matrix([[1,2,3],
             [4,5,6],
             [7,8,9]])

B=np.matrix([[1,4,2],
             [1,2,2],
             [3,-1,1]])

print(A+B)
print(A-B)
print(A*B)
print(A**2)
```

Matrices

Bloc/slicing

```
A=np.matrix([[1,2,3],
             [4,5,6],
             [7,8,9]])


A[0:2,1:3]
A[1:3,:]
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Operateurs matriciels

```
A=np.matrix([[1,2],
             [4,5]])

np.linalg.det(A)
np.trace(A)
np.linalg.inv(A)
```

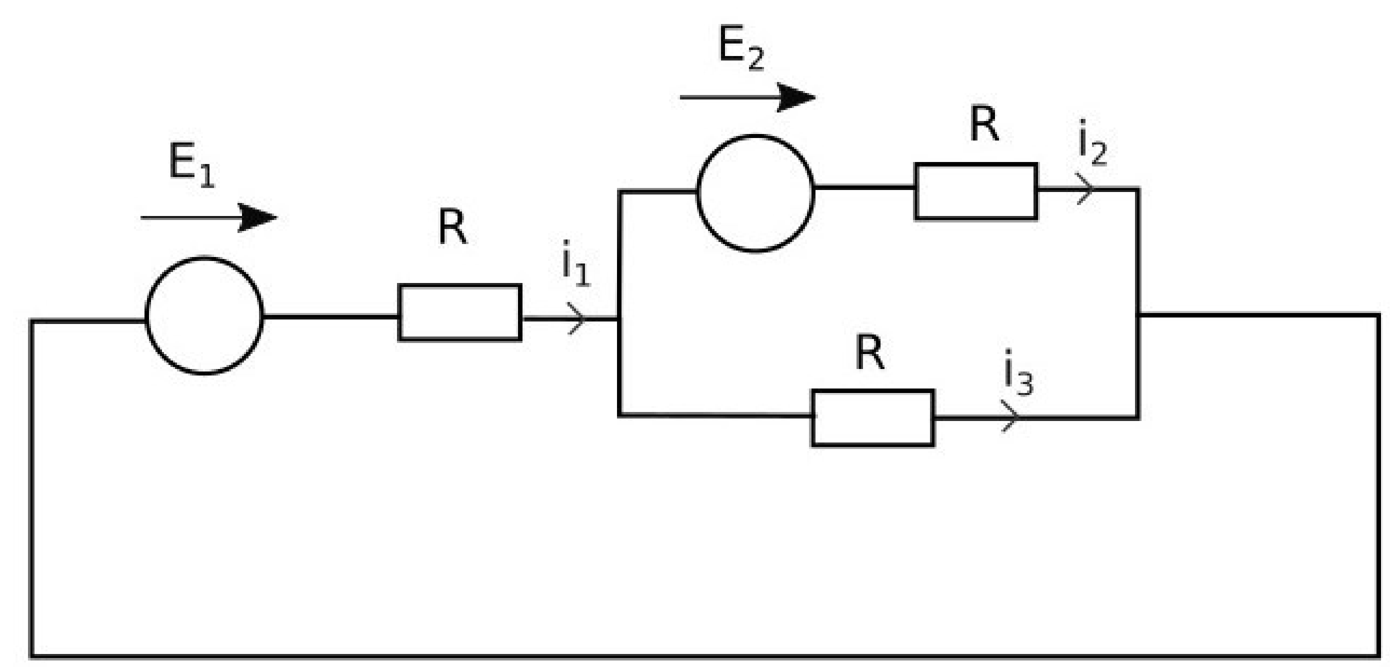
linalg = linear algebra

Algèbre linéaire

```
A=np.matrix([[1,2,5],
             [4,5,9],
             [7,8,4]])

b=np.array([4,8,9])

x=np.linalg.solve(A,b)
```

$$\mathrm{A}\,x = b$$

Application, systeme lineaire

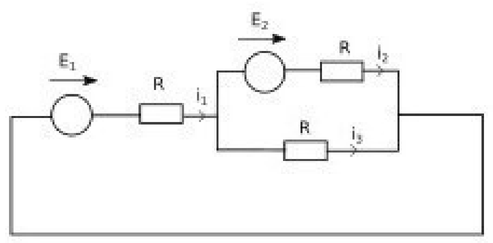


$$\left| \begin{array}{l} -E_1 + Ri_1 - E_2 + Ri_2 = 0 \\ -E_1 + Ri_1 + Ri_3 = 0 \\ \quad i_1 = i_2 + i_3 \end{array} \right.$$

$$\left| \begin{array}{l} R = 10k\Omega \\ E_1 = 1V \\ E_2 = 0.5V \end{array} \right.$$

Calculer i1, i2, i3

Application, systèmes lineaires



$$\left|\begin{array}{l} -E_1 + Ri_1 - E_2 + Ri_2 = 0 \\ -E_1 + Ri_1 + Ri_3 = 0 \\ \quad i_1 = i_2 + i_3 \end{array}\right. \qquad \left|\begin{array}{l} R = 10k\Omega \\ E_1 = 1V \\ E_2 = 0.5V \end{array}\right.$$

Calculer i1, i2, i3

```
A=np.matrix([[R,R,0],
             [R,0,R],
             [1,-1,-1]])

e=np.array([E1+E2,E1,0])

i=np.linalg.solve(A,e)

print(i)
```


Diagonalisation

```
A=np.matrix([[1,2,3],
             [4,7,8],
             [4,7,1]])

w,P=np.linalg.eig(A)

print(w)
print(P)

print(P*np.diag(w)*np.linalg.inv(P))
```


Application, diagonalisation

La matrice compagnon du polynome

$$p(x) = c_0 + c_1\,x + c_2\,x^2 + \cdots + c_{n-1}\,x^{n-1} + x^n$$

est $\quad \mathrm{A} = \begin{pmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{pmatrix}$

Les valeurs propres de A sont les racines du polynome p

Application, diagonalisation

Construire la matrice compagnon des polynomes suivants, et en déduire leurs racines:

$$p_1(x) = x^2 - 1$$

$$p_2(x) = x^2 + 1$$

$$p_3(x) = x^3 + 2x^2 - 1$$

$$p_4(x) = x^6 - 11x^5 + 30x^4 - 12x^3 - 13x^2 - x - 42$$

Application, diagonalisation

Construire la matrice compagnon des polynomes suivants,
et en déduire leurs racines:

$p_1(x) = x^2 - 1$

$p_2(x) = x^2 +$

$p_3(x) = x^3 + 2$

$p_4(x) = x^6 -$

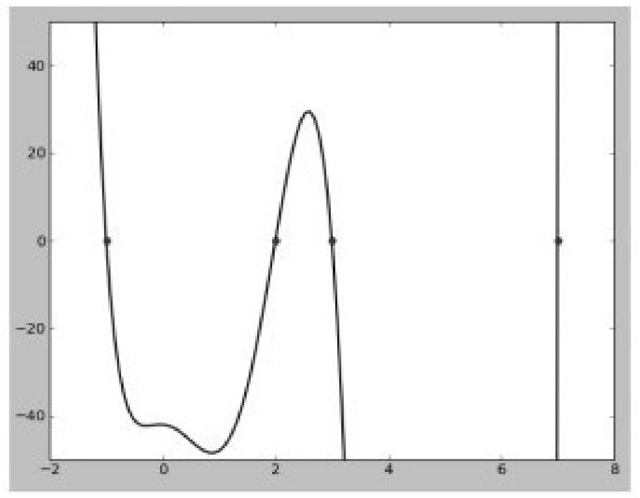
```
c=np.array([-42,-1,-13,-12,+30,-11])

N=len(c)
A=np.zeros([N,N])
for k in range(N-1):
  A[k+1,k]=1

A[:,-1]=-c


w,P=np.linalg.eig(A)
```


Application, diagonalisation

Afficher p3, ainsi que l'ensemble de ses racines réelles

Application, diagonalisation

Afficher p3, ainsi que l'ensemble de ces racines réelles



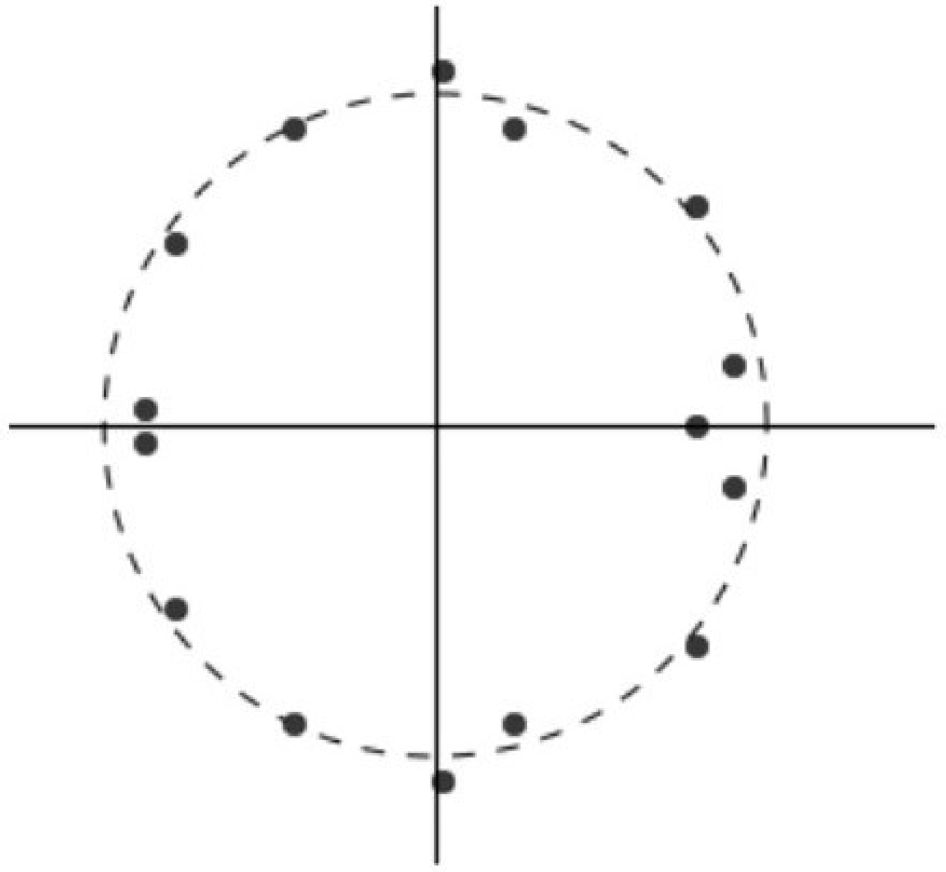

```
def f(x,c):
  p=np.zeros(len(x))
  for k in range(len(x)):
    for i in range(len(c)):
      p[k]+=c[i]*x[k]**i
    p[k]+=x[k]**(len(c))
  return p
```

```
x=np.linspace(-8,8,200)
plt.plot(x,f(x,c),linewidth=2)
epsilon=1e-5
for r in w:
  if(abs(r.imag)<epsilon):
    plt.plot(r.real,0,"ro")
plt.axis([-2,8,-50,50])
plt.show()
```


Application, diagonalisation

Construire un polynome dont les coefficients sont des reels aléatoires

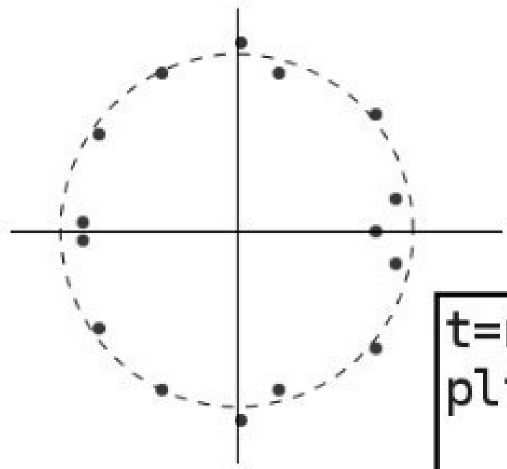
Observez la distribution des racines dans le plan complexe



Application, diagonalisation

Construire un polynome dont les coefficients sont des reels aléatoires

Observez la distribution des racines dans le plan complexe

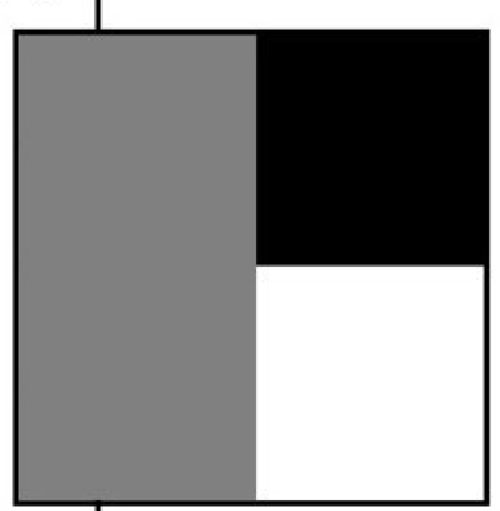


```
t=np.linspace(0,2*np.pi,100)
plt.plot(np.cos(t),np.sin(t),"b--")

for r in w:
  plt.plot(r.real,r.imag,"ro")

plt.plot([-2,2],[0,0],"k")
plt.plot([0,0],[-2,2],"k")
plt.axis("equal")
plt.axis([-2,2,-2,2])
plt.show()
```


Affichage matrices/images

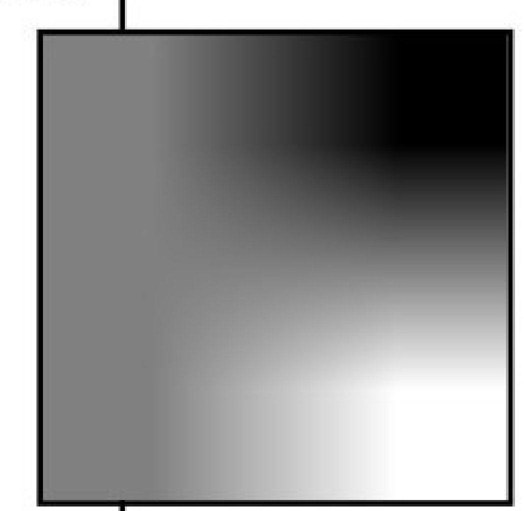
Affichage matrices/images

```
import numpy as np
import matplotlib.pyplot as plt

A=np.matrix([[1,0],
             [1,2]])

im=plt.imshow(A)

im.set_cmap('gray')
im.set_interpolation('nearest')
plt.show()
```



Affichage matrices/images

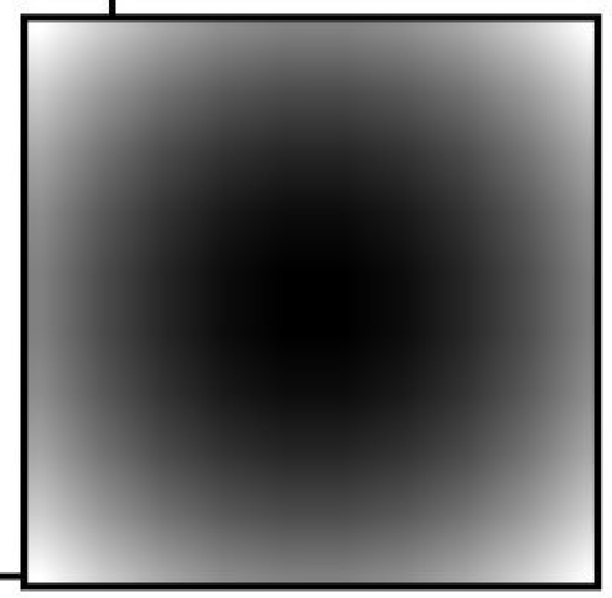
```
import numpy as np
import matplotlib.pyplot as plt

A=np.matrix([[1,0],
             [1,2]])

im=plt.imshow(A)

im.set_cmap('gray')
im.set_interpolation('nearest')
plt.show()
```



Affichage fonctions 2D

```
import numpy as np
import matplotlib.pyplot as plt

N=10
vx=np.linspace(-1.0,1.0,N)
vy=vx

z=np.zeros([N,N])
for kx,x in enumerate(vx):
  for ky,y in enumerate(vy):
    z[kx,ky]=x**2+y**2

plt.imshow(z)
plt.set_cmap("gray")
plt.show()
```

$$f(x,y) = x^2 + y^2$$



Affichage fonctions 2D

Soit:

$$\left| \begin{array}{l} f(x,y) = \cos\left(h(x,y-2)\,h(x,y+2)\right) \\ h(x,y) = 2\,\sqrt{x^2+y^2} \end{array} \right.$$

Afficher f (utiliser la colormap "hot")

Affichage fonctions 2D

Soit:

$$\left| \begin{array}{l} f(x,y) = \cos\left(h(x,y-2)\,h(x,y+2)\right) \\ h(x,y) = 2\,\sqrt{x^2+y^2} \end{array} \right.$$

Afficher f (utiliser la colormap "hot")

```
def h(x,y):
  return 2*(x**2+y**2)**(0.5)

def f(x,y):
  z=np.cos(h(x,y-2)*h(x,y+2))
  return z

N=150
vx,vy=np.linspace(-3,3,N),np.linspace(-3,3,N)

z=np.zeros([N,N])

for kx,x in enumerate(vx):
  for ky,y in enumerate(vy):
    z[kx,ky]=f(x,y)

print(z)
im=plt.imshow(z)
im.set_cmap("hot")
plt.show()
```

Application: Fractale

La fractale de Mandelbrot est obtenue en itérant la formule suivante:

$$\left| \begin{array}{l} z_{k+1} = z_k^2 + c \\ z_0 = c \end{array} \right.$$

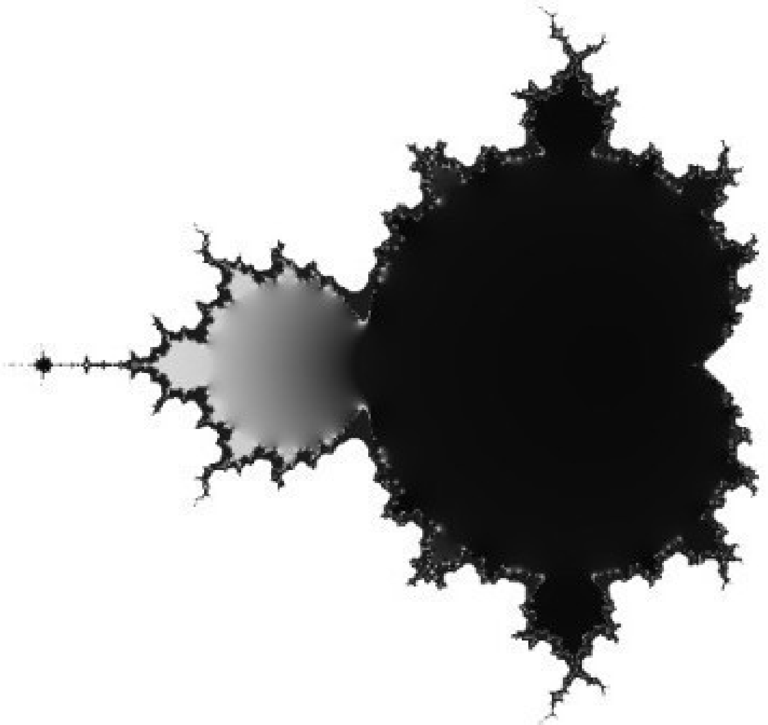
Afficher |z| après N iterations pour $c \in [-2-j, 2+j]$

Application: Fractale

La fractale de Mandelbrot est obtenue en itérant la formule suivante:

$$\begin{cases} z_{k+1} = z_k^2 + c \\ z_0 = c \end{cases}$$

Afficher |z| après N iterations pour $c \in [-2-j, 2+j]$



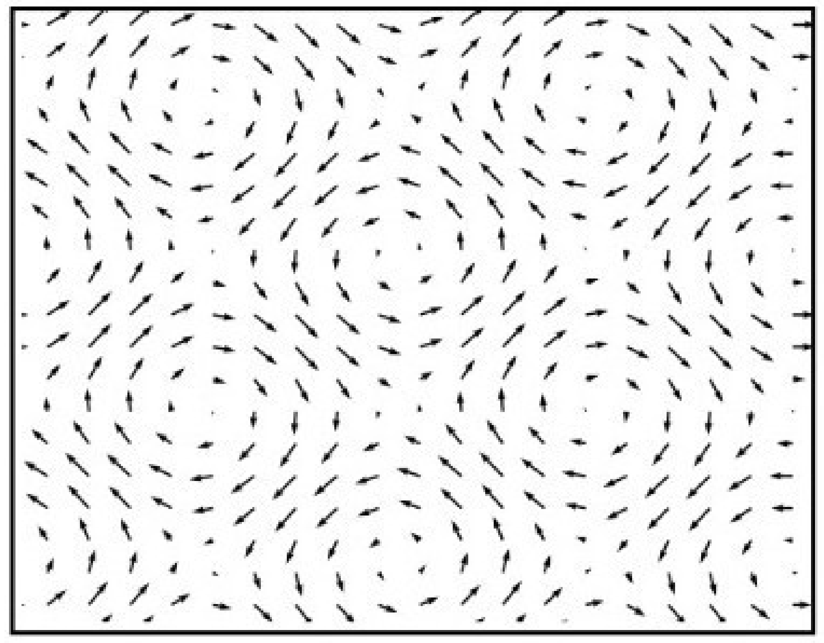
```
import numpy as np
import matplotlib.pyplot as plt

N=1500
u=np.linspace(-2,2,N)

xx,yy=np.meshgrid(u,u)
zz=xx+1j*yy
c=zz

for k in range(20):
  zz=zz**2+c

Z=zz.real**2+zz.imag**2

im=plt.imshow(Z)
im.set_clim(0,4)
plt.show()
```


Champs de vecteurs

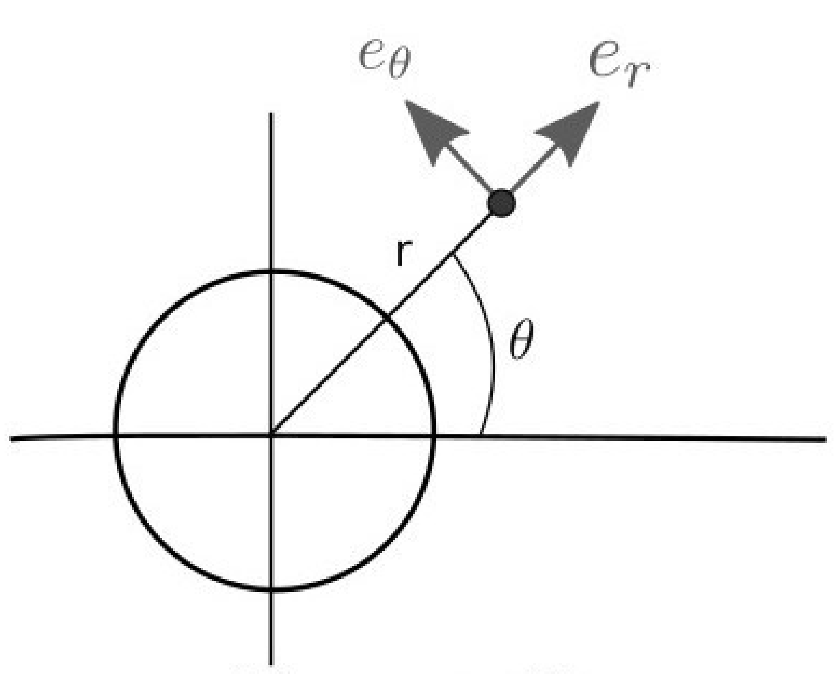
Champs de vecteurs : quiver

$$f : (x, y) \mapsto (\cos(x), \sin(y))$$

```
import numpy as np
import matplotlib.pyplot as plt

N=20
u=np.linspace(-1,1,N)

Vx=np.zeros([N,N])
Vy=np.zeros([N,N])
for kx,x in enumerate(u):
  for ky,y in enumerate(u):
    Vx[kx,ky]=np.cos(2*np.pi*x)
    Vy[kx,ky]=np.sin(2*np.pi*y)

plt.quiver(Vx,Vy)
plt.show()
```



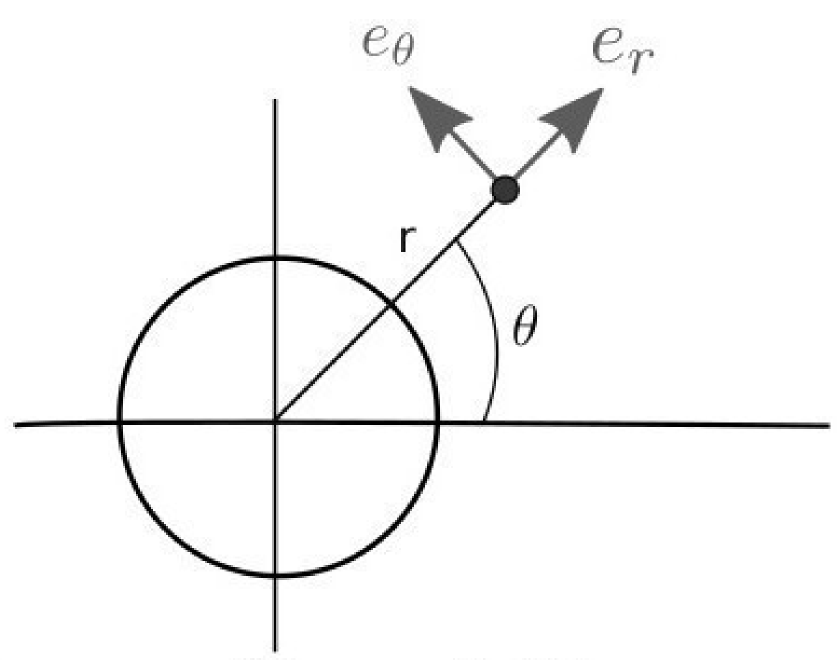
Changement de coordonnées



$$\begin{cases} e_x = \cos(\theta)\, e_r - \sin(\theta)\, e_\theta \\ e_y = \sin(\theta)\, e_r + \cos(\theta)\, e_\theta \end{cases}$$

Afficher e_θ et e_r pour $(x, y) \in [-1, 1]^2$

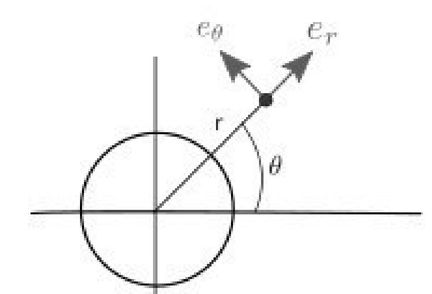
Changement de coordonnées



$$\left| \begin{array}{l} e_x = \cos(\theta)\, e_r - \sin(\theta)\, e_\theta \\ e_y = \sin(\theta)\, e_r + \cos(\theta)\, e_\theta \end{array} \right.$$

Afficher $\left| \begin{array}{l} f_r : (r,\theta) \mapsto (1,0) \\ f_\theta : (r,\theta) \mapsto (0,1) \end{array} \right.$

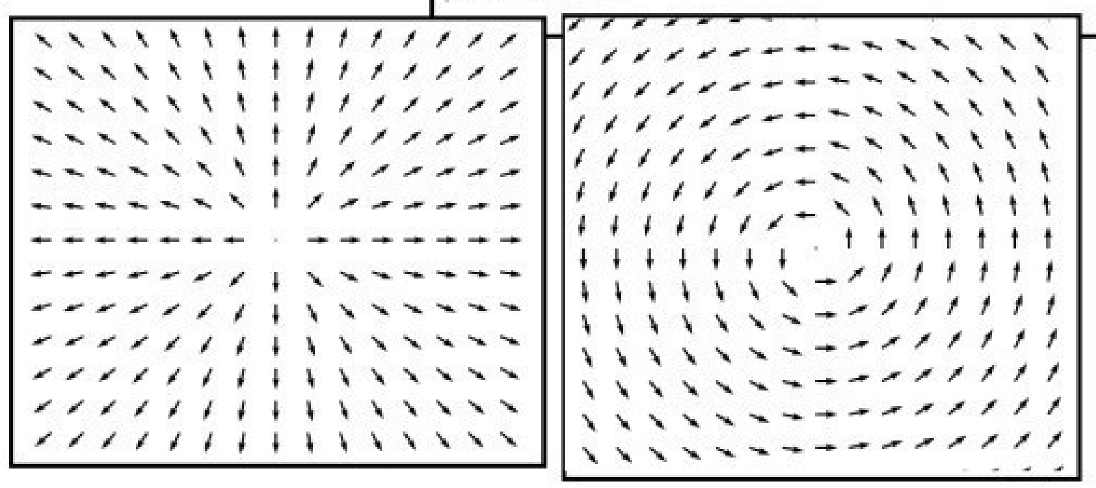
Changement de coordonnées



$$\left| \begin{array}{l} e_x = \cos(\theta)\, e_r - \sin(\theta)\, e_\theta \\ e_y = \sin(\theta)\, e_r + \cos(\theta)\, e_\theta \end{array} \right.$$

Afficher $\left| \begin{array}{l} f_r : (r,\theta) \mapsto (1,0) \\ f_\theta : (r,\theta) \mapsto (0,1) \end{array} \right.$


```
xx,yy=np.meshgrid(u,u)
Vx=np.zeros([N,N])
Vy=np.zeros([N,N])

for kx,x in enumerate(u):
  for ky,y in enumerate(u):

    r=(x**2+y**2)**0.5
    theta=math.atan2(y,x)

    if(r>=0.1):
      ur=1
      ut=0

      Vx[ky,kx]=np.cos(theta)*ur-np.sin(theta)*ut
      Vy[ky,kx]=np.sin(theta)*ur+np.cos(theta)*ut

im=plt.quiver(xx,yy,Vx,Vy)

plt.show()
```

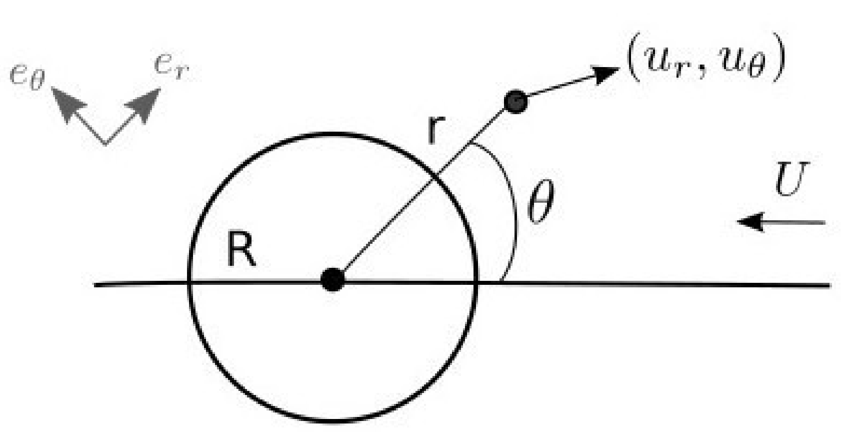


Application: mécanique des fluides

La vitesse d'un fluide (incompressible, non visqueux) autour d'une sphere de rayon R est donnée par

$$\left| \begin{array}{l} u_r(r,\theta) = -U\ \cos(\theta) \left(1 - \dfrac{3R}{2r} + \dfrac{R^3}{2r^3}\right) \\ u_\theta(r,\theta) = \quad U\ \sin(\theta) \left(1 - \dfrac{3R}{4r} - \dfrac{R^3}{4r^3}\right) \end{array} \right.$$

Afficher le champ de vecteur correspondant



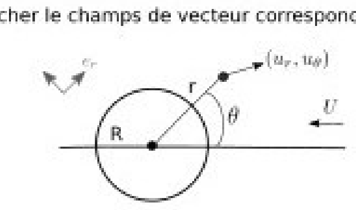
Application: mécanique des fluides

La vitesse d'un fluide (imcompressible, non visceux) autour d'une sphere de rayon R est donné par

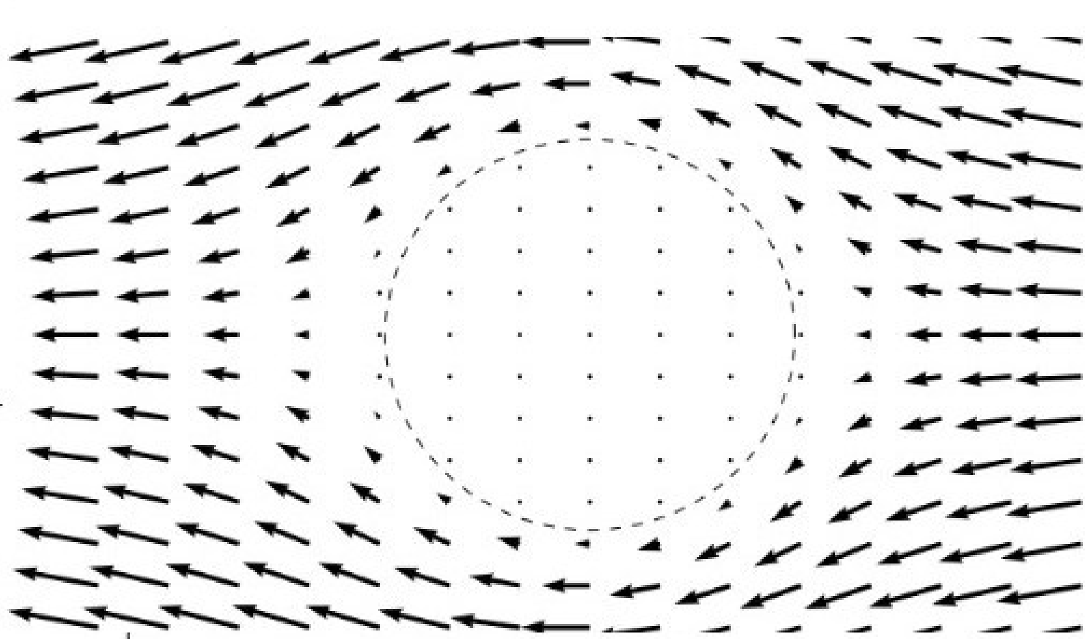
$$u_r(r,\theta) = -U\ \cos(\theta) \left(1 - \frac{3R}{2r} + \frac{R^3}{2r^3}\right)$$

$$u_\theta(r,\theta) = \quad U\ \sin(\theta) \left(1 - \frac{3R}{4r} - \frac{R^3}{4r^3}\right)$$

Afficher le champs de vecteur correspondant




```
xx,yy=np.meshgrid(ux,uy)
Vx=np.zeros([N,N])
Vy=np.zeros([N,N])

for kx,x in enumerate(ux):
  for ky,y in enumerate(uy):

    r=(x**2+y**2)**0.5
    theta=math.atan2(y,x)

    if(r>=R):
      ur=-U*np.cos(theta)*(1-3*R/(2*r)+R**3/(2*r**3))
      ut= U*np.sin(theta)*(1-3*R/(4*r)-R**3/(4*r**3))

      n=np.linalg.norm([ur,ut])

      Vx[ky,kx]=-np.sin(theta)*ut+np.cos(theta)*ur
      Vy[ky,kx]= np.cos(theta)*ut+np.sin(theta)*ur

im=plt.quiver(xx,yy,Vx,Vy)

t=np.linspace(0,2*np.pi,100)
plt.plot(R*np.cos(t),R*np.sin(t),"b--")
plt.show()
```



Lignes de champs

```
plt.streamplot(xx, yy, Vx, Vy, density=1,  color='b')
```