Tracé de courbes et de surfaces avec *Python* et *matplotlib*

Appel de *Python* depuis le *C*

Ni le *C* ni le *C++* ne possèdent d'instructions graphiques permettant de tracer des courbes et des surfaces.
Le but de cette notice est de montrer comment utiliser les possibilités graphiques de *Python* et de *matplotlib* à partir du *C* ou du *C++*.

1 Trois façons de tracer une courbe

On veut tracer la courbe $y = x^2 + 3x + 1$ entre $x = -3$ et $x = 1$.

1.1 Avec l'interpréteur de commandes *Python*

Dans ce qui suit *$* représente l'invite de commande *Linux* et `In [.]` l'invite de commande de l'interpréteur *Python*.
On lance l'interpréteur de commandes *ipython* dans un terminal *Linux* :

$ ipython –pylab

puis on écrit les commandes dans l'interpréteur en appuyant sur la touche Entrée après chaque ligne :

```
In [.] x=linspace(-3.,1.,100) # on demande 100 points équirépartis de -3 à 1 compris
In [.] y=x**2+3*x+1
In [.] plot(x,y)
```

ce qui donne la figure :

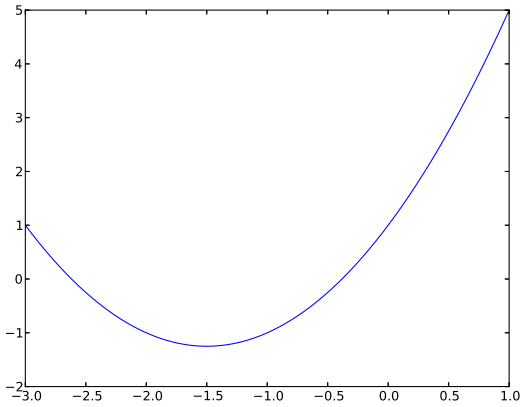

FIGURE 1 –

On sort de l'interpréteur par `quit` ou *Ctrl d*.
Ce mode d'utilisation, analogue à celui d'une calculatrice, est adapté aux tracés les plus simples.

1.2 En écrivant les instructions *Python* dans un fichier

On écrit les instructions *Python* précédentes dans un fichier nommé par exemple *courbe_1_a.py*.
On ouvre le fichier à l'aide d'un éditeur de texte :

$ emacs courbe_1_a.py &

puis on y écrit les instructions :

```
from numpy import *
from matplotlib.pyplot import *
x=linspace(-3.,1.,100)
y=x**2+3*x+1
plot(x,y)
show()
```

et on fait exécuter le programme *Python* écrit dans le fichier *courbe_1_a.py* par la commande :

$ ipython courbe_1_a.py

Ce mode d'utilisation est adapté dès que le nombre d'instructions et supérieur à deux ou trois ou quand on veut conserver ces instructions pour les réutiliser en les modifiant ou complétant si nécessaire.

Ici il faut appeler explicitement les modules `numpy` et `matplotlib.pyplot` et ajouter l'instruction `show()` alors que ce n'était pas nécessaire avec l'interpréteur grâce à l'option *–pylab*. Pour éviter d'avoir à réécrire à chaque fois les lignes :

```
from numpy import *
from matplotlib.pyplot import *
```

et :

```
show()
```

il existe une commande propre au Magistère nommée *ipy* qui les introduit automatiquement.
Si le fichier *courbe_1_b.py* ne contient que les lignes :

```
x=linspace(-3.,1.,100)
y=x**2+3*x+1
plot(x,y)
```

il suffit d'écrire la commande :

$ ipy courbe_1_b.py

au lieu de :

$ ipython courbe_1_b.py

1.3 En écrivant les instructions *Python* à partir d'un programme *C*

1.3.1 Méthode brute

On peut écrire et exécuter le programme *Python* à partir d'un programme *C* comme celui-ci :

```
#include<Python.h>
#include<iostream>
#include<sstream>
using namespace std;
int main()
{
  ostringstream ostpy;
  ostpy
    << "from numpy import *\n"
    << "from matplotlib.pyplot import *\n"
    << "x=linspace(-3.,1.,100)\n"
    << "y=x**2+3*x+1\n"
    << "plot(x,y)\n"
    << "show()"
    ;
  Py_Initialize();
  PyRun_SimpleString(ostpy.str().c_str());
  Py_Finalize();
  return 0;
}
```

Les instructions *Python* sont écrites dans une chaîne de caractères [1] nommée ici `ostpy` puis exécutées par l'instruction *C* :

```
PyRun_SimpleString(ostpy.str().c_str());
```

Ce mode d'utilisation permet de bénéficier des possibilités graphiques de *Python* dans un programme *C*.

1.3.2 En utilisant une fonction

On peut transformer en une fonction (nommée ici `make_plot_py`) la partie du programme précédent qui est toujours la même :

```
#include<Python.h>
#include<iostream>
#include<sstream>
using namespace std;
void make_plot_py(ostringstream &ss)
{
  ostringstream run_py;
  run_py
    << "from numpy import *\n"
    << "from matplotlib.pyplot import *\n"
    << "from mpl_toolkits.mplot3d import Axes3D\n"
    << ss.str()
    << "show()"
    ;
  //  cout << run_py.str() << endl;
  Py_Initialize();
  PyRun_SimpleString(run_py.str().c_str());
  Py_Finalize();
}
int main()
{
  ostringstream pyth;
  pyth
    << "x=linspace(-3.,1.,100)\n"
    << "y=x**2+3*x+1\n"
    << "plot(x,y)\n"
    ;
  make_plot_py(pyth);
  return 0;
}
```

1.3.3 En utilisant la fonction mise dans la bibliothèque du Magistère

La fonction `make_plot_py` étant incluse dans la bibliothèque du Magistère on aura juste à écrire :

```
#include<Python.h>
#include<bibli_fonctions.h>
int main()
{
  ostringstream pyth;
  pyth
    << "x=linspace(-3.,1.,100)\n"
    << "y=x**2+3*x+1\n"
    << "plot(x,y)\n"
    ;
```

1. On utilise ici une chaîne de type `ostringstream` et non du type courant `string` pour des raisons qui apparaissent à la section **2.** Le couple de caractères `\n` indique un passage à la ligne dans la chaîne.

```
  make_plot_py(pyth);
  return 0;
}
```

2 Passage de paramètres du C à $Python$

Supposons qu'on veuille tracer non plus la courbe $y = x^2 + 3x + 1$ mais $y = x^2 + ax + 1$ où a est une variable définie dans le programme C.
Il suffit d'ajouter, dans le `main` du programme précédent, les instructions qui déclarent et attribuent une valeur à la variable a :

```
int main()
{
  double a;
  ...
  a=... // calcul quelconque
  ...
  return 0;
}
```

et de remplacer la ligne :

```
<< "y=x**2+3*x+1\n"
```

par :

```
<< "y=x**2+" << a << "*x+1\n"
```

> À PARTIR D'ICI, DANS CES NOTES, ON N'ÉCRIT PLUS QUE LES INSTRUCTIONS $Python$ PROPREMENT DITES, L'APPEL À PARTIR DU C COMME À LA SOUS-SECTION 1.3.3 ÉTANT SOUS-ENTENDU.

Par exemple dans un cas comme celui du programme de la sous-section 1.3.3, on écrira seulement dans ces notes :

```
x=linspace(-3.,1.,100)
y=x**2+3*x+1
plot(x,y)
```

3 Tracer plusieurs courbes sur le même graphe

```
x=linspace(-2.34,2.73,47)
y1=(2+x)**2
y2=(1-x)**3
y3=y1+y2
plot(x,y1)
plot(x,y2)
plot(x,y3)
```

ce qui donne la figure :

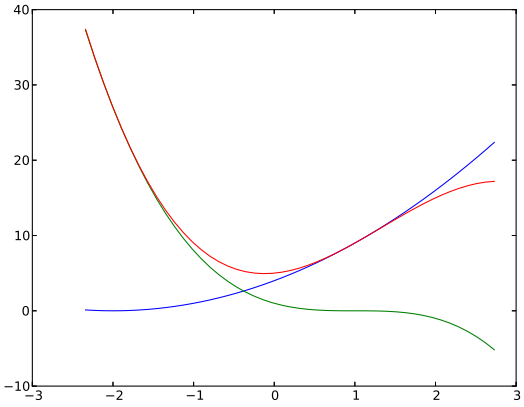

FIGURE 2 –

4 Tracer des courbes à partir de points écrits dans un fichier

4.1 Une seule courbe

On veut tracer une courbe définie non plus par une expression mathématique mais par une liste de points dont les coordonnées x et y sont écrites dans un fichier. Ces coordonnées peuvent par exemple être le résultat d'un calcul effectué par un programme $C/C++$ ou des valeurs expérimentales. Sur chaque ligne du fichier figure un couple x y comme sur l'exemple suivant :

```
1.82776e+08 1.38477e+08
1.82777e+08 1.36734e+08
1.82778e+08 1.34974e+08
...
```

Les instructions *Python* sont :

```
l=loadtxt('courbe_6.dat')
x=l[:,0]
y=l[:,1]
plot(x,y)
```

où `courbe_6.dat` est le nom du fichier contenant les coordonnées des points. Cela donne la figure :

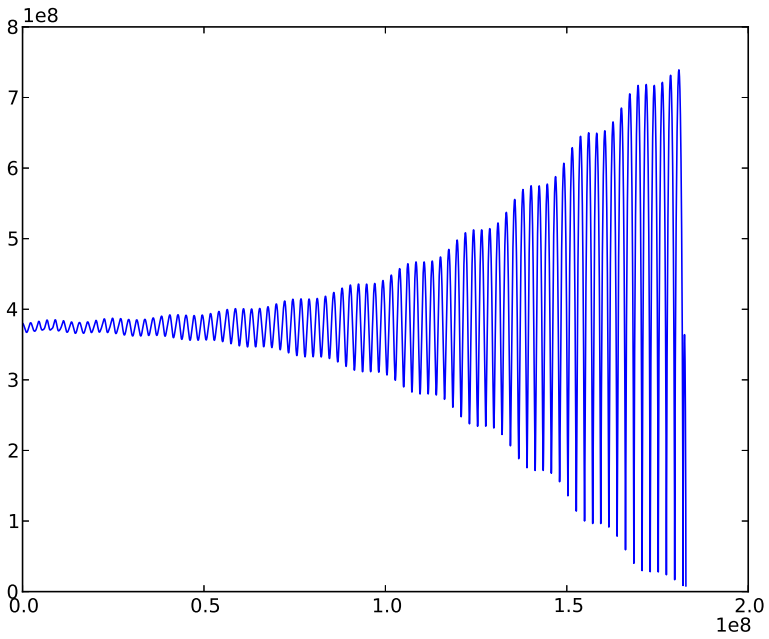

FIGURE 3 –

4.2 Plusieurs courbes

On a deux courbes définies respectivement par deux listes de couples de points (x_i, y_{i0}) et (x_i, y_{i1}).
Le fichier doit écrit de la façon suivante :

x_0 y_{00} y_{01}
x_1 y_{10} y_{11}
x_2 y_{20} y_{21}
...
x_{p-1} $y_{p-1,0}$ $y_{p-1,1}$

Par exemple :

0.5 0.833325 0.420871
1 0.833298 0.425152
1.5 0.833253 0.429512
...

4.2.1 Tracé brut

```
A=loadtxt('courbe_7.dat')
plot(A[:,0],A[:,1])
plot(A[:,0],A[:,2])
ylim(-30,30)
```

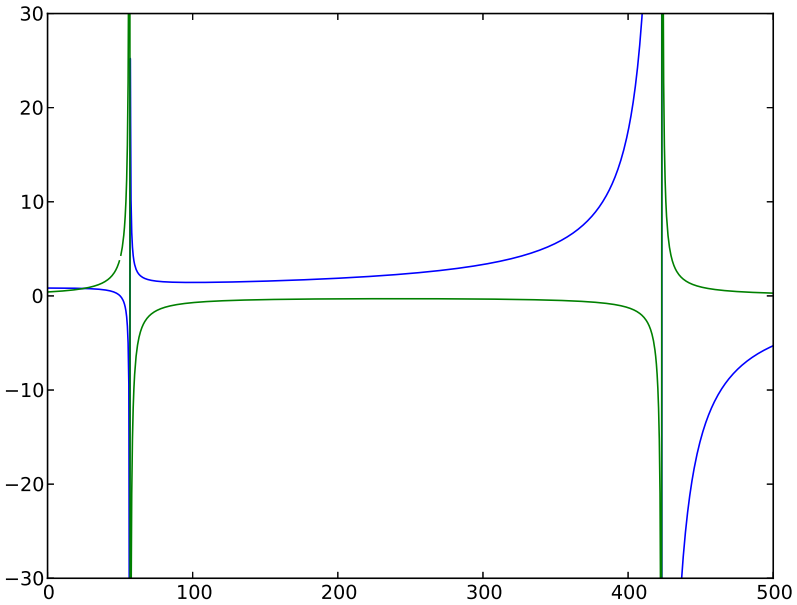

FIGURE 4 –

4.2.2 Tracé habillé

```
A=loadtxt('courbe_7.dat')
plot(A[:,0],A[:,1],label='y(u)')
plot(A[:,0],A[:,2],label='grandissement *10')
plot(A[:,0],0*ones(A.shape[0]),label='0')
plot(A[:,0],2*ones(A.shape[0]),label='2')
ylim(-30,30)
xlabel('u')
legend(loc=9)
```

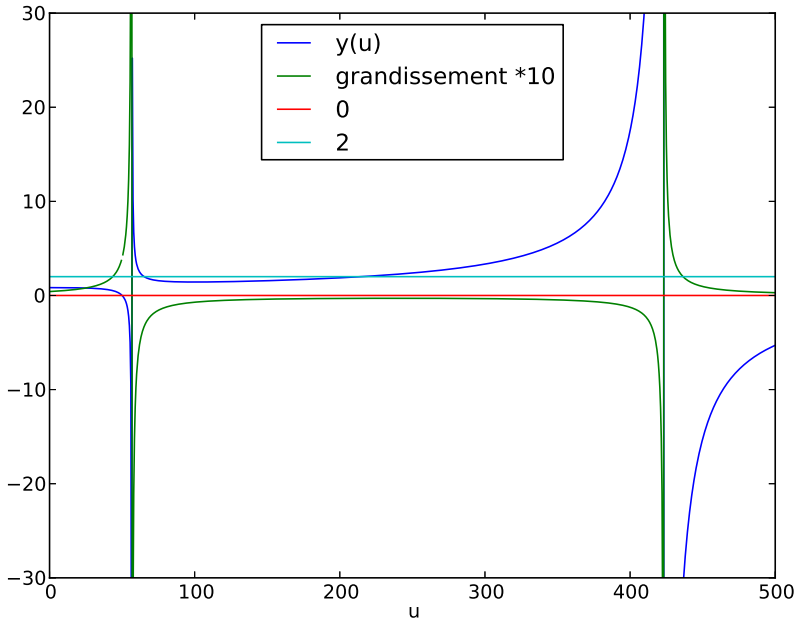

FIGURE 5 –

On peut facilement généraliser à un nombre quelconque q de courbes. Le fichier est écrit de la façon suivante :

$x_0\ y_{0,0}\ y_{0,1}\ \cdots\ y_{0,j}\ \cdots\ y_{0,q-1}$
$x_1\ y_{1,0}\ y_{1,1}\ \cdots\ y_{1,j}\ \cdots\ y_{1,q-1}$
$\cdots$
$x_i\ y_{i,0}\ y_{i,1}\ \cdots\ y_{i,j}\ \cdots\ y_{i,q-1}$
$\cdots$
$x_{p-1}\ y_{p-1,0}\ y_{p-1,1}\ \cdots\ y_{p-1,j}\ \cdots\ y_{p-1,q-1}$

et le tracé est fait par une boucle en Python :

```
A=loadtxt('courbe_8.dat')
for i in range(1,102) : # il y a 102 colonnes et 101 courbes dans la fichier courbe_8.dat
    plot(A[:,0],A[:,i])
```

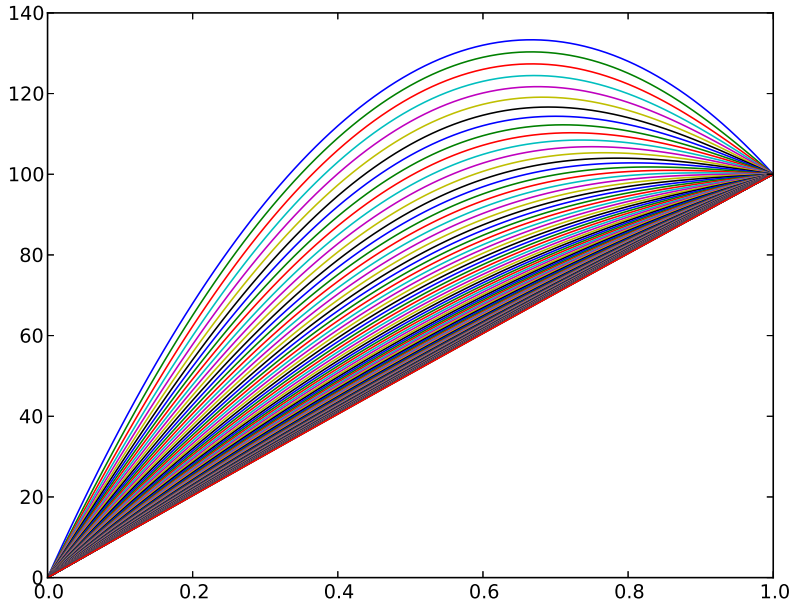

FIGURE 6 –

5 Tracer une courbe dans l'espace

5.1 Courbe dans l'espace définie par une expression mathématique

5.1.1 Tracé brut

```
theta=linspace(-16*pi,16*pi,800)
z=linspace(-2,2,800)
r=z*z+1
x=r*sin(theta)
y=r*cos(theta)
gca(projection='3d').plot(x,y,z)
```

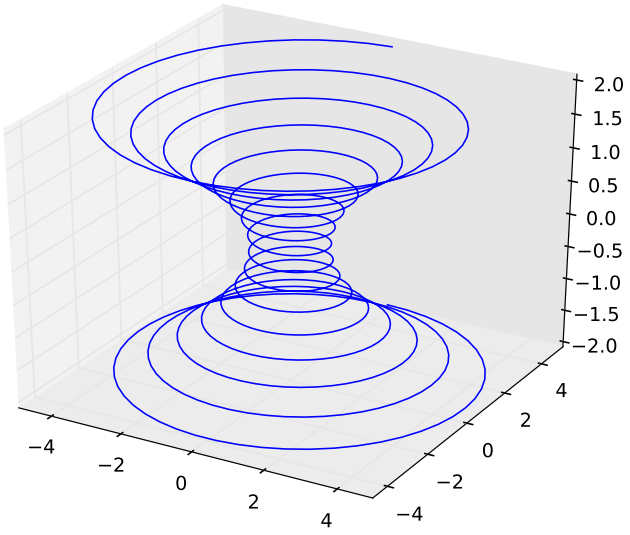

FIGURE 7 –

5.1.2 Tracé habillé

```
rcParams['legend.fontsize']=10
ax=figure().gca(projection='3d')
theta=linspace(-16*pi,16*pi,800)
z=linspace(-2,2,800)
r=z*z+1
x=r*sin(theta)
y=r*cos(theta)
ax.plot(x,y,z,label='Courbe en parametrique')
ax.legend()
```

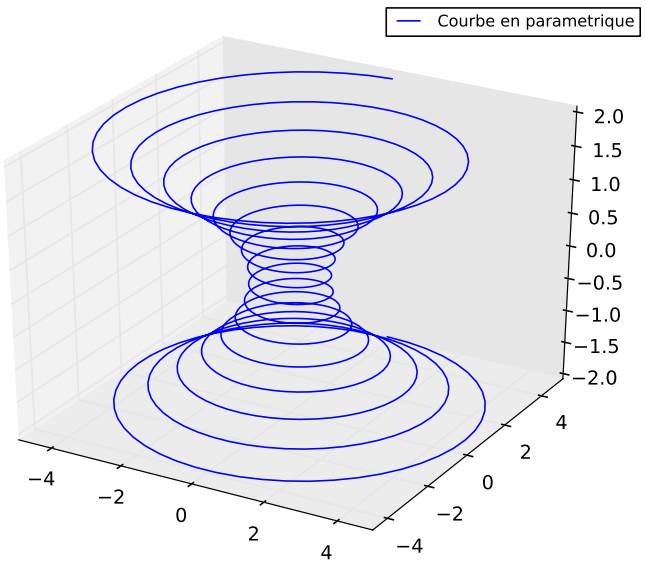

FIGURE 8 –

5.2 Courbe dans l'espace définie par des points

Les points sont définis par des triplets de coordonnées x, y, z qui sont écrits dans le fichier de données de la façon suivante :

x_0 y_0 z_0
x_1 y_1 z_1
x_2 y_2 z_2
...

Par exemple :

1 0 0
0.982287 0.187381 0.03
0.929776 0.368125 0.06
...

5.2.1 Tracé brut

```
l=loadtxt('courbe_10.dat')
x=l[:,0]
y=l[:,1]
z=l[:,2]
gca(projection='3d').plot(x,y,z)
# Pour ne tracer que les points :
#gca(projection='3d').scatter(x,y,z)
```

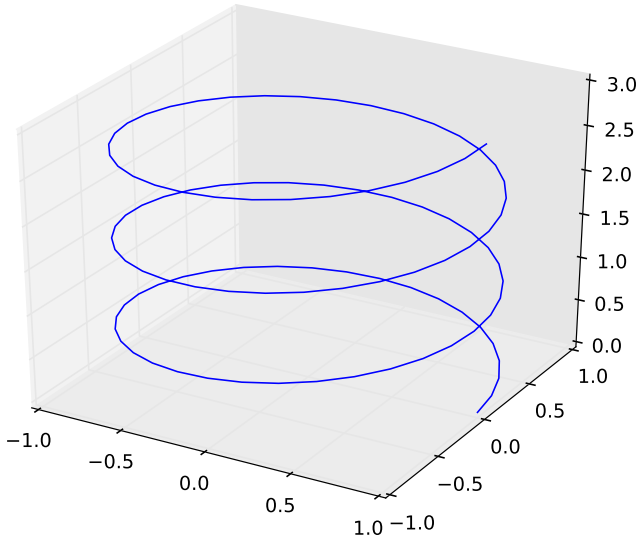

FIGURE 9 –

6 Tracer une surface

6.1 Surface définie par une expression mathématique

6.1.1 Tracé brut

```
X=linspace(-5,5,40)
Y=linspace(-5,5,40)
X,Y=meshgrid(X,Y)
Z=sin(sqrt(X**2+Y**2))
gca(projection='3d').plot_surface(X,Y,Z.T,rstride=1,cstride=1,
                                  cmap=cm.nipy_spectral_r,linewidth=0,antialiased=False)
```

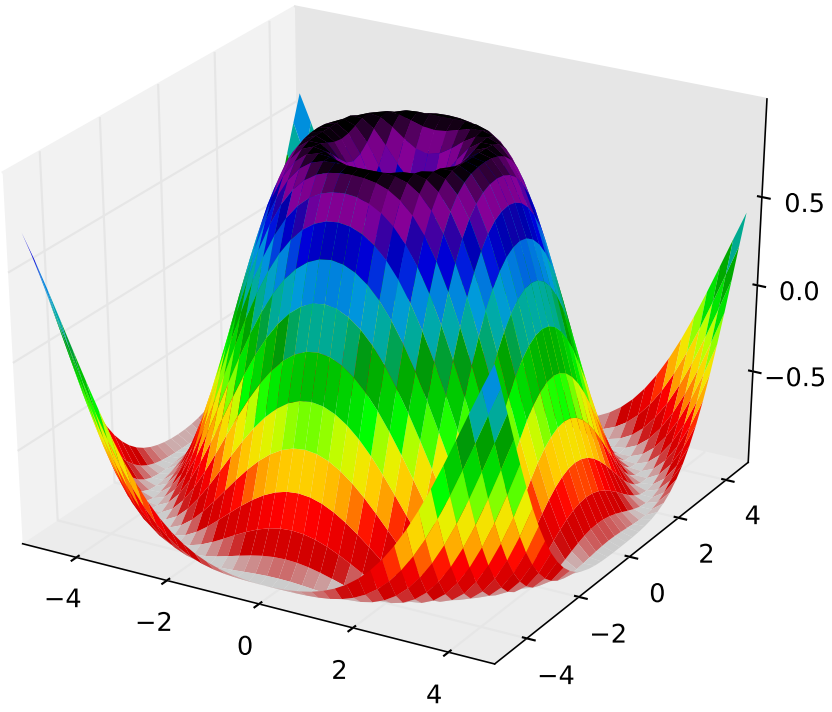

FIGURE 10 –

6.1.2 Tracé habillé

```
X=linspace(-5,5,40)
Y=linspace(-5,5,40)
X,Y=meshgrid(X,Y)
Z=sin(sqrt(X**2+Y**2))
surf=gca(projection='3d').plot_surface(X,Y,Z.T,rstride=1,cstride=1,
                                       cmap=cm.nipy_spectral_r,
                                       linewidth=0,antialiased=False)
gca().set_zlim(-1.5,1.5)
colorbar(surf,shrink=0.5,aspect=5)
```

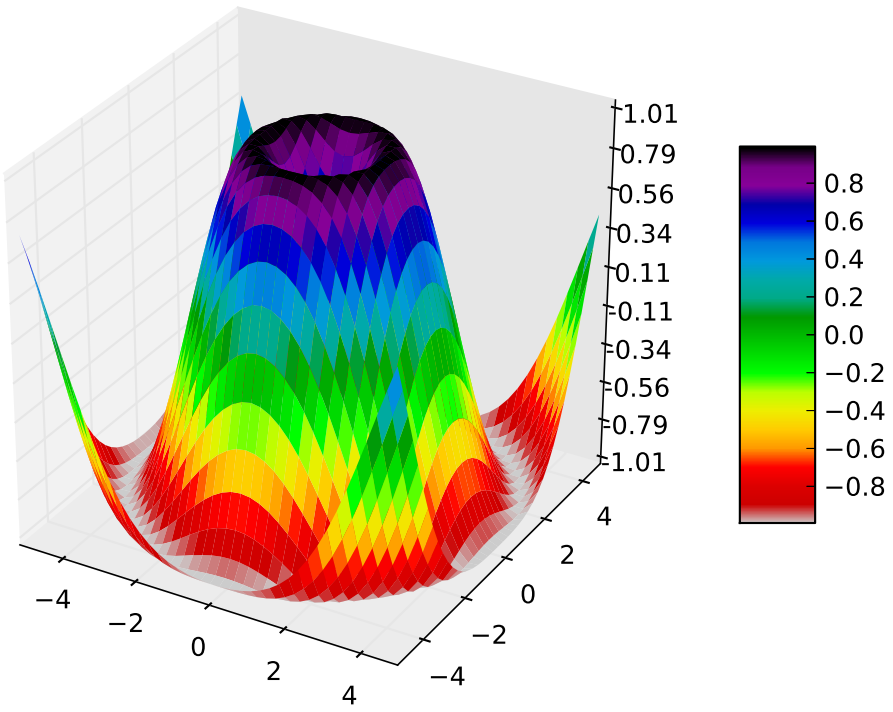

FIGURE 11 –

6.2 Surface définie par des points

Les valeurs de x et y doivent constituer un réseau régulier.
Dans le fichier chaque ligne contient les valeurs de z pour un x donné et toutes les valeurs de y.

6.2.1 Tracé brut

```
Z=loadtxt('surface_2.res')
X=arange(Z.shape[0])
Y=arange(Z.shape[1])
X,Y=meshgrid(X,Y)
gca(projection='3d').plot_surface(X,Y,Z.T,rstride=1,cstride=1,linewidth=0,
cmap=cm.nipy_spectral_r,antialiased=False)
```

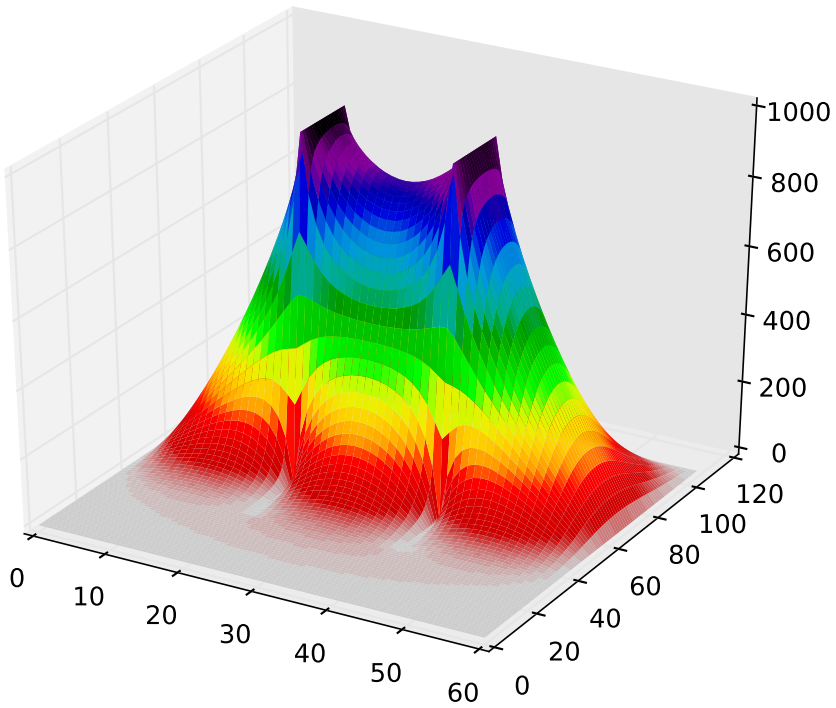

FIGURE 12 –

6.2.2 Tracé habillé

```
figure(1,figsize=(16,6))
subplot(1,2,1,projection='3d')
Z=loadtxt('surface_2.res')
X=arange(Z.shape[0])
Y=arange(Z.shape[1])
X,Y=meshgrid(X,Y)
gca().plot_surface(X,Y,Z.T,rstride=1,cstride=1,linewidth=0,
                   cmap=cm.nipy_spectral_r,antialiased=False)
xlabel('x')
ylabel('y')
title('Potentiel electrostatique')
subplot(1,2,2)
cont=contour(Z.T,linspace(0,1000,11),cmap=cm.nipy_spectral_r)
clabel(cont,fmt='%d')
xlabel('x')
ylabel('y')
title('Courbes de niveau')
```

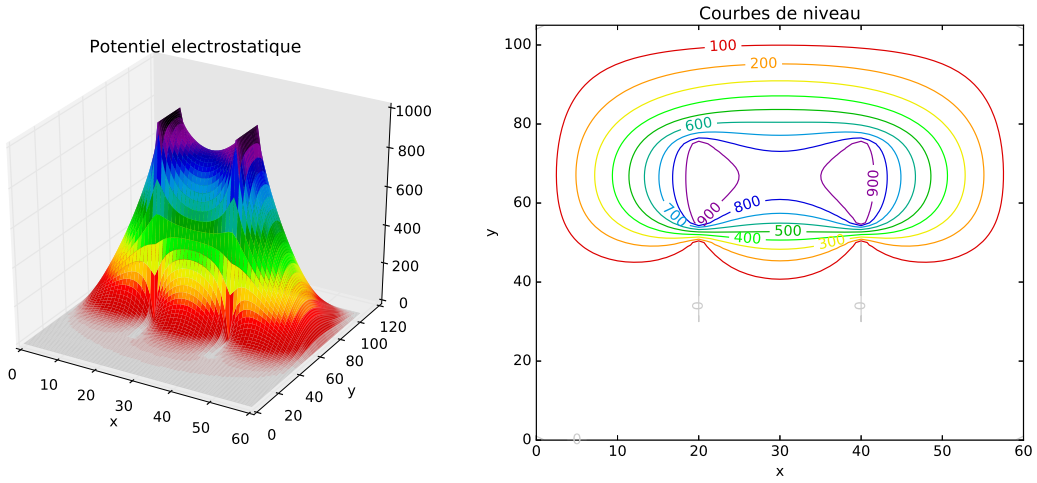

FIGURE 13 –