IA_proyecto2

September 20, 2020

[144]:
```
import numpy as np
import math
import pandas
np.set_printoptions(suppress=True)
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from random import random, sample, randint

from sympy import *
%matplotlib inline
#%matplotlib qt
```

1 De Jong's function

1.1

$$f(x) = \sum_{i=1}^{n} x_i^2$$

1.2 Gráfica de la función en caso bidimensional

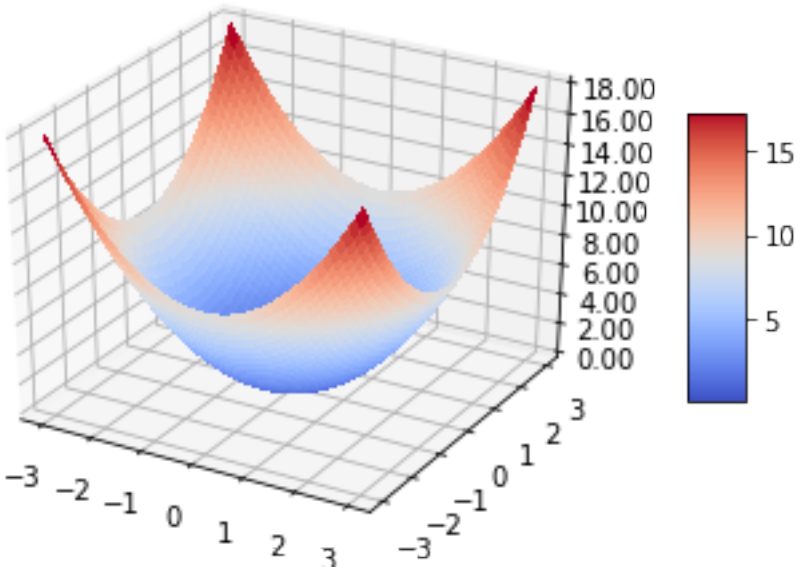
Para la función

1.3

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

[145]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(-3, 3, 0.01)
Y = np.arange(-3, 3, 0.01)
X, Y = np.meshgrid(X, Y)

Z = X**2 + Y**2
```

```
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



1.4 Expresión matemática del gradiente.

Para la función

$$f(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2$$

tenemos

$$\nabla f = (2x_1, 2x_2, 2x_3, 2x_4, 2x_5).$$

El área de prueba generalmente está restringida al hipercubo $ -5.12 \leq xi \leq 5.12$

[146]:
```
gradientes=[]
for i in range (20):
    der = np.ones(5)
    a = np.ones(5)
```

```
    for i in range(5) :
        a[i]=round(uniform(-5.12, 5.12), 3)
   # print("elementos en el arreglo: ",a)
    landa = .001
    aux = np.ones(len(a))
    cont=0
    while cont < 1000:
        cont=cont+1
        der = 2 * a
        a = a - (landa * der)
    #print("número de iteraciones: ",cont)
    #print("landa utilizada: ",landa)
    fx=a[0]**2+a[1]**2+a[2]**2+a[3]**2+a[4]**2
    print("La mejor solución encontrada es: ",fx)

    gradientes += [fx]
```

```
La mejor solución encontrada es:  0.8617698507819773
La mejor solución encontrada es:  1.2284187549295982
La mejor solución encontrada es:  1.005636073915007
La mejor solución encontrada es:  0.9940143355928588
La mejor solución encontrada es:  0.48678025442810435
La mejor solución encontrada es:  0.7640122695232425
La mejor solución encontrada es:  0.8479674859182886
La mejor solución encontrada es:  1.083085895535858
La mejor solución encontrada es:  0.8749420128238907
La mejor solución encontrada es:  0.7542687165748616
La mejor solución encontrada es:  1.2309829102190508
La mejor solución encontrada es:  1.1060280985275812
La mejor solución encontrada es:  0.8148507161974237
La mejor solución encontrada es:  1.028402803018501
La mejor solución encontrada es:  0.4376184508138162
La mejor solución encontrada es:  0.9676847878734701
La mejor solución encontrada es:  0.9333028512360633
La mejor solución encontrada es:  0.6164612771616964
La mejor solución encontrada es:  1.0266361335901322
La mejor solución encontrada es:  0.7638363943016646
```

[147]:
```
#función de Jongs
def Aptitude(I):
 [x1,x2,x3,x4,x5] = Ind2Number(I,v_min,v_max,n_var,n_bits)
 r = x1**2 + x2**2 + x3**2 + x4**2 + x5**2
 return 1.0/(1.0+0.1*r)

genetico = []
for i in range(20):
    y = 0
```

```
    N = 100
    n_var = 5
    n_bits = 10
    P0 = IniciaPob(N,n_var,n_bits)
    v_min = [-5.12,-5.12,-5.12,-5.12,-5.12]
    v_max = [5.12,5.12,5.12,5.12,5.12]
    n_gens = 10
    P = Evolution(P0,0.01,n_gens)
    best = Pop_Aptitude(P)
    index = np.argmax(best)
    a = Ind2Number(P[index],v_min,v_max,n_var,n_bits)

    fx=a[0]**2+a[1]**2+a[2]**2+a[3]**2+a[4]**2
    print("la mejor solucion encontrada es: ", fx)
    genetico += [fx]
```

```
la mejor solucion encontrada es:  0.15382529332679706
la mejor solucion encontrada es:  0.236386466882427
la mejor solucion encontrada es:  0.07867859409533828
la mejor solucion encontrada es:  0.2930971759024355
la mejor solucion encontrada es:  1.318498935815443
la mejor solucion encontrada es:  0.29189482871473177
la mejor solucion encontrada es:  0.4285616257170131
la mejor solucion encontrada es:  0.20051644244927702
la mejor solucion encontrada es:  1.6419303293076442
la mejor solucion encontrada es:  1.054984510510459
la mejor solucion encontrada es:  0.2638400610016533
la mejor solucion encontrada es:  0.19450470651075993
la mejor solucion encontrada es:  0.2389915524557842
la mejor solucion encontrada es:  0.38307282378223767
la mejor solucion encontrada es:  0.47625473082924563
la mejor solucion encontrada es:  0.6608150241417106
la mejor solucion encontrada es:  0.47164573327638315
la mejor solucion encontrada es:  0.4984981538017582
la mejor solucion encontrada es:  1.1058838747899011
la mejor solucion encontrada es:  0.24941189474921371
```

[148]:
```
data = {'Algoritmos genéticos':pandas.Series(genetico),'Descenso de gradiente':
 ↪pandas.Series(gradientes)}
data_frame = pandas.DataFrame(data)
print(data_frame.describe())
```

```
       Algoritmos genéticos  Descenso de gradiente
count             20.000000              20.000000
mean               0.512065               0.891335
std                0.430105               0.215864
min                0.078679               0.437618
```

25%	0.238340	0.763968
50%	0.338085	0.904122
75%	0.539077	1.027078
max	1.641930	1.230983

2 Axis parallel hyper-ellipsoid function

2.1

$$f(x) = \sum_{i=1}^{n} (i * x_i^2)$$

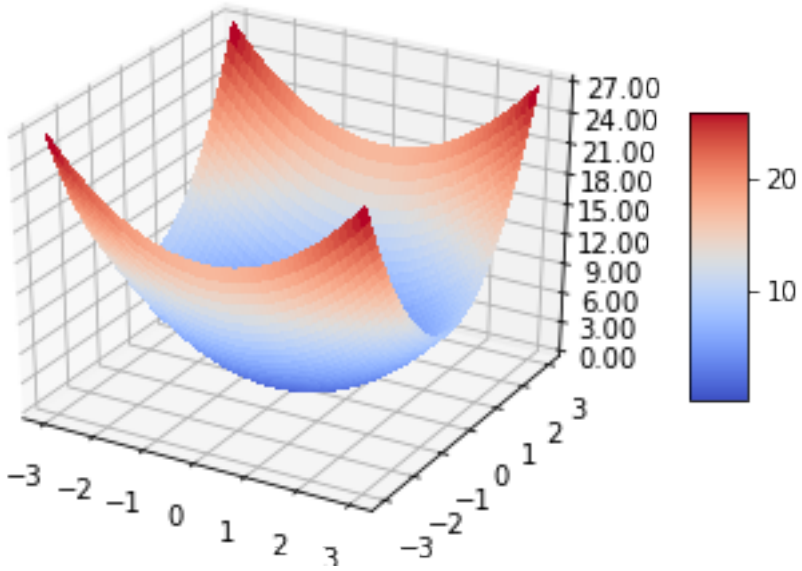
2.2 Gráfica de la función en caso bidimensional

Para la función

2.3

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

[149]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(-3, 3, 0.01)
Y = np.arange(-3, 3, 0.01)
X, Y = np.meshgrid(X, Y)

Z = X**2 + 2 * Y **2

surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



2.4 Expresión matemática del gradiente.

Para la función

$$f(x,y) = x_1^2 + 2x_2^2 + 3x_3^2 + 4x_4^2 + 5x_4^2$$

tenemos

$$\nabla f = (2x_1, 4x_1, 6x_3, 8x_4, 10x_5,).$$

El área de prueba generalmente está restringida al hipercubo $ -5.12 \leq xi \leq 5.12$

[150]:
```
gradientes=[]
for i in range (20):
    der = np.ones(5)
    a = np.ones(5)
    for i in range(5) :
        a[i]=round(uniform(-5.12, 5.12), 3)
   # print("elementos en el arreglo: ",a)
    landa = .001
    aux = np.ones(len(a))
    cont=0
    while cont < 1000:
        cont=cont+1
        der[0] =  2* a[0]
        der[1] =  4* a[1]
```

```
        der[2] =  6* a[2]
        der[3] =  8* a[3]
        der[4] =  10* a[4]
        a = a - (landa * der)
    #print("número de iteraciones: ",cont)
    #print("landa utilizada: ",landa)
    fx=a[0]**2+2*a[1]**2+3*a[2]**2+4*a[3]**2+5*a[4]**2
    print("La mejor solución encontrada es: ",fx)

    gradientes += [fx]
```

```
La mejor solución encontrada es:  0.20068192194449122
La mejor solución encontrada es:  0.017645552288221063
La mejor solución encontrada es:  0.1010274619162005
La mejor solución encontrada es:  0.21243855693813182
La mejor solución encontrada es:  0.04724322379782161
La mejor solución encontrada es:  0.020314236897749156
La mejor solución encontrada es:  0.0777428071387163
La mejor solución encontrada es:  0.02840458343098141
La mejor solución encontrada es:  0.15716824406718238
La mejor solución encontrada es:  0.026529397373904747
La mejor solución encontrada es:  0.060112673252633496
La mejor solución encontrada es:  0.18684801708802545
La mejor solución encontrada es:  0.052128337376997395
La mejor solución encontrada es:  0.07131035367817107
La mejor solución encontrada es:  0.4525973541639501
La mejor solución encontrada es:  0.29719298607117745
La mejor solución encontrada es:  0.42987919183733897
La mejor solución encontrada es:  0.0029051836231150743
La mejor solución encontrada es:  0.018622330239136835
La mejor solución encontrada es:  0.39632549377363685
```

[151]:
```
def Aptitude(I):
 [x1,x2,x3,x4,x5] = Ind2Number(I,v_min,v_max,n_var,n_bits)
 r = x1**2 + 2*x2**2 + 3*x3**2 + 4*x4**2 + 5*x5**2
 return 1.0/(1.0+0.1*r)

genetico = []
for i in range(20):
    y = 0
    N = 100
    n_var = 5
    n_bits = 10
    P0 = IniciaPob(N,n_var,n_bits)
    v_min = [-5.12,-5.12,-5.12,-5.12,-5.12]
    v_max = [5.12,5.12,5.12,5.12,5.12]
    n_gens = 10
```

```
    P = Evolution(P0,0.001,n_gens)
    best = Pop_Aptitude(P)
    index = np.argmax(best)
    a = Ind2Number(P[index],v_min,v_max,n_var,n_bits)

    fx=a[0]**2+2*a[1]**2+3*a[2]**2+4*a[3]**2+5*a[4]**2
    print("la mejor solucion encontrada es: ", fx)
    genetico += [fx]
```

```
la mejor solucion encontrada es:  0.8774880069257491
la mejor solucion encontrada es:  1.996873238677571
la mejor solucion encontrada es:  1.3073271265296993
la mejor solucion encontrada es:  1.763417493065172
la mejor solucion encontrada es:  2.17001123370685
la mejor solucion encontrada es:  0.9848976890272498
la mejor solucion encontrada es:  4.177730645973505
la mejor solucion encontrada es:  2.653154411965652
la mejor solucion encontrada es:  1.7586081043143609
la mejor solucion encontrada es:  5.16145103671279
la mejor solucion encontrada es:  1.0698635569582846
la mejor solucion encontrada es:  0.42841133231854994
la mejor solucion encontrada es:  3.216654460602623
la mejor solucion encontrada es:  1.8678213071974061
la mejor solucion encontrada es:  1.987454852373892
la mejor solucion encontrada es:  1.1063347549852902
la mejor solucion encontrada es:  1.781252309682771
la mejor solucion encontrada es:  0.42019529320257687
la mejor solucion encontrada es:  0.2608842908318835
la mejor solucion encontrada es:  1.2516183734994444
```

[152]:
```
data = {'Algoritmos genéticos':pandas.Series(genetico),'Descenso de gradiente':
 ↪pandas.Series(gradientes)}
data_frame = pandas.DataFrame(data)
print(data_frame.describe())
```

```
       Algoritmos genéticos  Descenso de gradiente
count             20.000000              20.000000
mean               1.812072               0.142856
std                1.238224               0.145889
min                0.260884               0.002905
25%                1.048622               0.027936
50%                1.761013               0.074527
75%                2.040158               0.203621
max                5.161451               0.452597
```

3 Rotated hyper-ellipsoid function

3.1

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{i} (x_j^2)$$

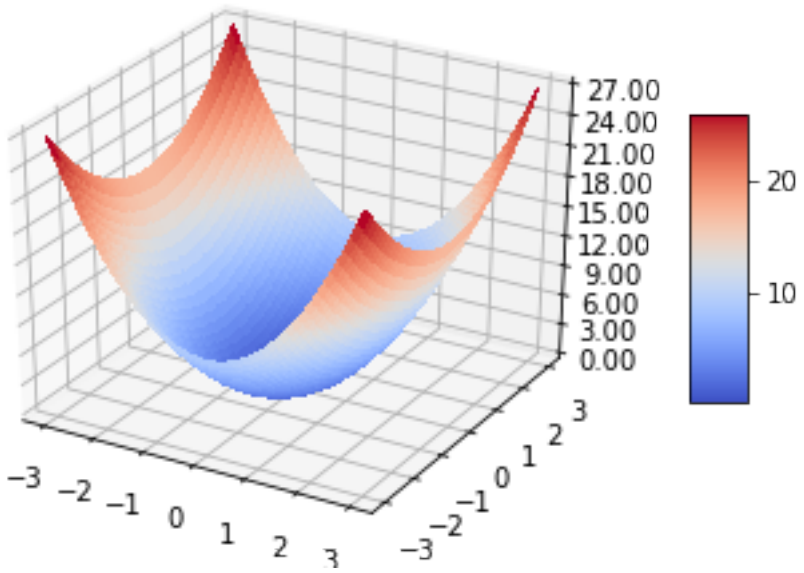
3.2 Gráfica de la función en caso bidimensional

Para la función

3.3

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

[153]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(-3, 3, 0.01)
Y = np.arange(-3, 3, 0.01)
X, Y = np.meshgrid(X, Y)

Z = 2*X**2 +  Y **2

surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



3.4 Expresión matemática del gradiente.

Para la función

$$f(x) = 5x_1^2 + 4x_2^2 + 3x_2^2 + 2x_2^2 + x_5^2$$

tenemos

$$\nabla f = (10x_1, 8x_2, 6x_3, 4x_4, 2x_5).$$

El área de prueba generalmente está restringida al hipercubo $ -65.536 \leq xi \leq 65.536$

[154]:
```
gradientes=[]
for i in range (20):
    der = np.ones(5)
    a = np.ones(5)
    for i in range(5) :
        a[i]=round(uniform(-65.536, 65.536), 3)
   # print("elementos en el arreglo: ",a)
    landa = .01
    aux = np.ones(len(a))
    cont=0
    while cont < 1000:
        cont=cont+1
        der[0] =  10* a[0]
        der[1] =  8* a[1]
```

```
        der[2] =  6* a[2]
        der[3] =  4* a[3]
        der[4] =   2*a[4]
        a = a - (landa * der)
    #print("número de iteraciones: ",cont)
    #print("landa utilizada: ",landa)
    fx=5*a[0]**2+4*a[1]**2+3*a[2]**2+2*a[3]**2+a[4]**2
    print("La mejor solución encontrada es: ",fx)

    gradientes += [fx]
```

```
La mejor solución encontrada es:  1.8385779678195 03e-15
La mejor solución encontrada es:  1.1214249799829389e-14
La mejor solución encontrada es:  3.0566701329491514e-15
La mejor solución encontrada es:  6.876251728921491e-15
La mejor solución encontrada es:  1.3932700669715958e-15
La mejor solución encontrada es:  1.2695580034972251e-16
La mejor solución encontrada es:  6.173061795483676e-16
La mejor solución encontrada es:  7.777033307507878e-15
La mejor solución encontrada es:  1.1689326549102645e-14
La mejor solución encontrada es:  4.665561161790604e-15
La mejor solución encontrada es:  3.0214147169084686e-15
La mejor solución encontrada es:  1.5285765565349937e-15
La mejor solución encontrada es:  6.233421380779448e-16
La mejor solución encontrada es:  2.2831968556242015e-15
La mejor solución encontrada es:  9.275517860985942e-15
La mejor solución encontrada es:  6.3456986992027314e-15
La mejor solución encontrada es:  1.1404679417479484e-14
La mejor solución encontrada es:  1.609500129140002e-15
La mejor solución encontrada es:  1.0831543465303537e-16
La mejor solución encontrada es:  7.264492353818654e-16
```

[155]:
```
def Aptitude(I):
 [x1,x2,x3,x4,x5] = Ind2Number(I,v_min,v_max,n_var,n_bits)
 r = 5*x1**2 + 4*x2**2 +3*x3**2 +2*x4**2 + x5**2
 return 1.0/(1.0+0.1*r)

genetico = []
for i in range(20):
    y = 0
    N = 100
    n_var = 5
    n_bits = 10
    P0 = IniciaPob(N,n_var,n_bits)
    v_min = [-5.12,-5.12,-5.12,-5.12,-5.12]
    v_max = [5.12,5.12,5.12,5.12,5.12]
    n_gens = 10
```

```
    P = Evolution(P0,0.001,n_gens)
    best = Pop_Aptitude(P)
    index = np.argmax(best)
    a = Ind2Number(P[index],v_min,v_max,n_var,n_bits)

    fx=5*a[0]**2+4*a[1]**2+3*a[2]**2+2*a[3]**2+a[4]**2
    print("la mejor solucion encontrada es: ", fx)
    genetico += [fx]
```

```
la mejor solucion encontrada es:  0.6181817967777282
la mejor solucion encontrada es:  1.2119409163052304
la mejor solucion encontrada es:  0.3977514790321153
la mejor solucion encontrada es:  1.1546290336913734
la mejor solucion encontrada es:  1.5902794980358905
la mejor solucion encontrada es:  0.9209728968810218
la mejor solucion encontrada es:  1.9361547056985484
la mejor solucion encontrada es:  1.0546338259140446
la mejor solucion encontrada es:  2.353569571029564
la mejor solucion encontrada es:  0.43322072106936316
la mejor solucion encontrada es:  0.8570481047347948
la mejor solucion encontrada es:  2.817875976681013
la mejor solucion encontrada es:  2.4291170526569235
la mejor solucion encontrada es:  0.6288025302691082
la mejor solucion encontrada es:  0.6540518212108773
la mejor solucion encontrada es:  0.794526050974218
la mejor solucion encontrada es:  2.0706171995233724
la mejor solucion encontrada es:  0.8247851218647556
la mejor solucion encontrada es:  0.5775023835937645
la mejor solucion encontrada es:  1.345802236536205
```

[156]:
```
data = {'Algoritmos genéticos':pandas.Series(genetico),'Descenso de gradiente':
 ↪pandas.Series(gradientes)}
data_frame = pandas.DataFrame(data)
print(data_frame.describe())
```

```
       Algoritmos genéticos  Descenso de gradiente
count             20.000000           2.000000e+01
mean               1.233573           4.309095e-15
std                0.726049           4.053557e-15
min                0.397751           1.083154e-16
25%                0.647739           1.226565e-15
50%                0.987803           2.652306e-15
75%                1.676748           7.101447e-15
max                2.817876           1.168933e-14
```

4 Rastrigin's function

4.1

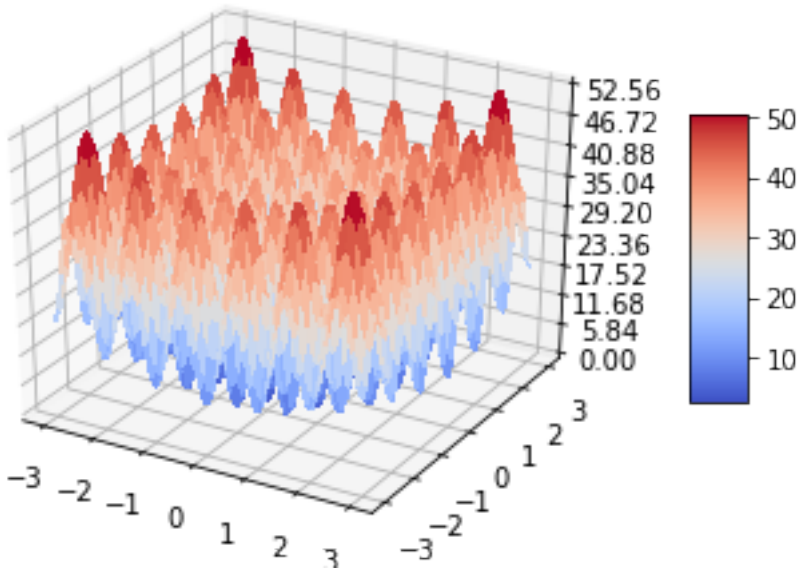
$$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i]$$

4.2 Gráfica de la función en caso bidimensional

4.3

$$10 * 2 + [x^2 - 10\cos(2\pi x)] + [y^2 - 10\cos(2\pi y)]$$

[157]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(-3, 3, 0.01)
Y = np.arange(-3, 3, 0.01)
X,Y = np.meshgrid(X, Y)

Z = 10 * 2 + X**2 - 10 * np.cos(2*np.pi* X)+ Y**2 - 10 * np.cos(2*np.pi*Y)

surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



4.4 Expresión matemática del gradiente.

Para la función

$$f(x) = 10 * 5 + [x_1^2 - 10\cos(2\pi x_1)] + [x_2^2 - 10\cos(2\pi x_2)] + [x_3^2 - 10\cos(2\pi x_3) + [x_4^2 - 10\cos(2\pi x_4) + [x_5^2 - 10\cos(2$$

tenemos

$$\nabla f = (2x_1 + 20\pi\sin(2\pi x_1), 2x_2 + 20\pi\sin(2\pi x_2), 2x_3 + 20\pi\sin(2\pi x_3), 2x_4 + 20\pi\sin(2\pi x_4), 2x_5 + 20\pi\sin(2\pi x_5),$$

El área de prueba generalmente está restringida al hipercubo $ -5.12 \leq xi \leq 5.12$

[158]:
```
gradientes=[]
for i in range (20):
    der = np.ones(5)
    a = np.ones(5)
    for i in range(5) :
        a[i]=round(uniform(-5.12, 5.12), 3)
   # print("elementos en el arreglo: ",a)
    landa = .001
    aux = np.ones(len(a))
    cont=0
    while cont < 1000:
        cont=cont+1
        der[0] = 2*a[0] + 20 * np.pi * np.sin(2*np.pi*a[0])
        der[1] = 2*a[1] + 20 * np.pi * np.sin(2*np.pi*a[1])
        der[2] = 2*a[2] + 20 * np.pi * np.sin(2*np.pi*a[2])
        der[3] = 2*a[3] + 20 * np.pi * np.sin(2*np.pi*a[3])
        der[4] = 2*a[4] + 20 * np.pi * np.sin(2*np.pi*a[4])
        a = a - (landa * der)
    #print("número de iteraciones: ",cont)
    #print("landa utilizada: ",landa)
    fx= (10*5 + (a[0] ** 2 - 10* np.cos(2*np.pi*a[0]))+
                (a[1] ** 2 - 10* np.cos(2*np.pi*a[1]))+
                (a[2] ** 2 - 10* np.cos(2*np.pi*a[2]))+
                (a[3] ** 2 - 10* np.cos(2*np.pi*a[3]))+
                (a[4] ** 2 - 10* np.cos(2*np.pi*a[4])))



    print("La mejor solución encontrada es: ",fx)
```

```
    gradientes += [fx]
```

```
La mejor solución encontrada es:  53.727399159044865
La mejor solución encontrada es:  49.74758816520523
La mejor solución encontrada es:  26.863823921175488
La mejor solución encontrada es:  69.6466429515064
La mejor solución encontrada es:  17.909222679688476
La mejor solución encontrada es:  68.65160740915177
La mejor solución encontrada es:  27.858737364123797
La mejor solución encontrada es:  65.6668117609523
La mejor solución encontrada es:  21.889053870242563
La mejor solución encontrada es:  43.77784389828363
La mejor solución encontrada es:  43.77798115658532
La mejor solución encontrada es:  83.57589214452001
La mejor solución encontrada es:  14.92435054622768
La mejor solución encontrada es:  13.92941168584884
La mejor solución encontrada es:  53.727521258451176
La mejor solución encontrada es:  46.762716031744425
La mejor solución encontrada es:  49.74769006789709
La mejor solución encontrada es:  40.79310902312453
La mejor solución encontrada es:  43.777843898283635
La mejor solución encontrada es:  18.904120963741406
```

[159]:
```
def Aptitude(I):
 [x1,x2,x3,x4,x5] = Ind2Number(I,v_min,v_max,n_var,n_bits)
 r = (10*5 +    (x1 ** 2 - 10* np.cos(2*np.pi*x1))+
                (x2 ** 2 - 10* np.cos(2*np.pi*x2))+
                (x3 ** 2 - 10* np.cos(2*np.pi*x3))+
                (x4 ** 2 - 10* np.cos(2*np.pi*x4))+
                (x5 ** 2 - 10* np.cos(2*np.pi*x5)))
 return 1.0/(1.0+0.1*r)

genetico = []
for i in range(20):
    y = 0
    N = 100
    n_var = 5
    n_bits = 10
    P0 = IniciaPob(N,n_var,n_bits)
    v_min = [-5.12,-5.12,-5.12,-5.12,-5.12]
    v_max = [5.12,5.12,5.12,5.12,5.12]
    n_gens = 10
    P = Evolution(P0,0.001,n_gens)
    best = Pop_Aptitude(P)
    index = np.argmax(best)
    a = Ind2Number(P[index],v_min,v_max,n_var,n_bits)
```

```
    fx= (10*5 + (a[0] ** 2 - 10* np.cos(2*np.pi*a[0]))+
                (a[1] ** 2 - 10* np.cos(2*np.pi*a[1]))+
                (a[2] ** 2 - 10* np.cos(2*np.pi*a[2]))+
                (a[3] ** 2 - 10* np.cos(2*np.pi*a[3]))+
                (a[4] ** 2 - 10* np.cos(2*np.pi*a[4])))
    print("la mejor solucion encontrada es: ", fx)
    genetico += [fx]
```

```
la mejor solucion encontrada es:  7.771503707805914
la mejor solucion encontrada es:  10.720637328552257
la mejor solucion encontrada es:  5.700143084590367
la mejor solucion encontrada es:  5.280505297070686
la mejor solucion encontrada es:  8.83618996742405
la mejor solucion encontrada es:  4.955627264753282
la mejor solucion encontrada es:  10.05027419581887
la mejor solucion encontrada es:  12.593744110033658
la mejor solucion encontrada es:  7.902585655158408
la mejor solucion encontrada es:  7.9973299780474
la mejor solucion encontrada es:  5.319341155369544
la mejor solucion encontrada es:  8.821749451378588
la mejor solucion encontrada es:  5.473460555088259
la mejor solucion encontrada es:  7.207961783129489
la mejor solucion encontrada es:  6.890487492248305
la mejor solucion encontrada es:  5.625966842983861
la mejor solucion encontrada es:  10.829965213225067
la mejor solucion encontrada es:  3.4608952501665087
la mejor solucion encontrada es:  5.743009065723114
la mejor solucion encontrada es:  7.406588242623109
```

[160]:
```
data = {'Algoritmos genéticos':pandas.Series(genetico),'Descenso de gradiente':
 ↪pandas.Series(gradientes)}
data_frame = pandas.DataFrame(data)
print(data_frame.describe())
```

```
       Algoritmos genéticos  Descenso de gradiente
count             20.000000              20.000000
mean               7.429398              42.782968
std                2.347537              20.063270
min                3.460895              13.929412
25%                5.587840              25.620131
50%                7.307275              43.777913
75%                8.825360              53.727430
max               12.593744              83.575892
```

5 Sum of different power functions

5.1

$$f(x) = \sum_{i=1}^{n} |x_i|^{i+1}$$

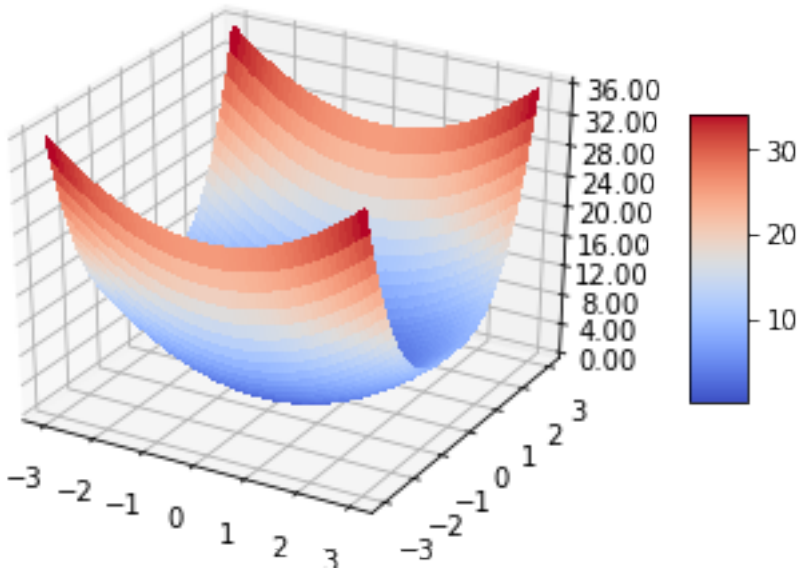
5.2 Gráfica de la función en caso bidimensional

Para la función

5.3

$$f(x,y) = |x|^2 + |y|^3$$

[161]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(-3, 3, 0.01)
Y = np.arange(-3, 3, 0.01)
X, Y = np.meshgrid(X, Y)

Z = abs(X)**2 + abs(Y) **3

surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



5.4 Expresión matemática del gradiente.

Para la función

$$f(x) = |x_1|^2 + |x_2|^3 + |x_3|^4 + |x_4|^5 + |x_5|^6$$

tenemos

$$\nabla f = (2x_1, 3x_2|x_2|, 4x_3^3, 5x_4|x_4|, 6x_5^5)$$

El área de prueba generalmente está restringida al hipercubo:

$$1x_i1$$

[162]:
```
gradientes=[]
for i in range (20):
    der = np.ones(5)
    a = np.ones(5)
    for i in range(5) :
        a[i]=round(uniform(-1, 1), 3)
   # print("elementos en el arreglo: ",a)
    landa = .001
    aux = np.ones(len(a))
    cont=0
    while cont < 1000:
        cont=cont+1
```

```
        der[0] =  2*a[0]
        der[1] =  3*a[1]*abs(a[1])
        der[2] =  4*(a[2]**3)
        der[3] =  5*a[3]*abs(a[3])
        der[4] =  6*(a[4]**5)
        a = a - (landa * der)
    #print("número de iteraciones: ",cont)
    #print("landa utilizada: ",landa)
    fx= abs(a[0])**2+abs(a[1])**3+abs(a[2])**4+abs(a[3])**5+abs(a[4])**6




    print("La mejor solución encontrada es: ",fx)

    gradientes += [fx]
```

```
La mejor solución encontrada es:  0.027258624295806045
La mejor solución encontrada es:  0.024003739279870137
La mejor solución encontrada es:  0.009457570667124642
La mejor solución encontrada es:  0.007148203367226052
La mejor solución encontrada es:  0.021103473564113302
La mejor solución encontrada es:  0.013776369705217563
La mejor solución encontrada es:  0.013766162710678749
La mejor solución encontrada es:  0.03744667382273599
La mejor solución encontrada es:  0.025791866639566077
La mejor solución encontrada es:  0.02678475569063394
La mejor solución encontrada es:  0.019706706840919207
La mejor solución encontrada es:  0.027846762621551446
La mejor solución encontrada es:  0.01000671358888692
La mejor solución encontrada es:  0.011378962775138128
La mejor solución encontrada es:  0.030863873025320683
La mejor solución encontrada es:  0.016158676170896696
La mejor solución encontrada es:  0.02962782928471965
La mejor solución encontrada es:  0.020915458084855704
La mejor solución encontrada es:  0.048320002858519326
La mejor solución encontrada es:  0.023631745266745694
```

[163]:
```
def Aptitude(I):
 [x1,x2,x3,x4,x5] = Ind2Number(I,v_min,v_max,n_var,n_bits)
 r = abs(x1)**2 + abs(x2)**3 + abs(x3)**4 + abs(x4)**5 + abs(x5)**6
 return 1.0/(1.0+0.1*r)

genetico = []
for i in range(20):
    y = 0
    N = 100
```

```
    n_var = 5
    n_bits = 10
    P0 = IniciaPob(N,n_var,n_bits)
    v_min = [-5.12,-5.12,-5.12,-5.12,-5.12]
    v_max = [5.12,5.12,5.12,5.12,5.12]
    n_gens = 10
    P = Evolution(P0,0.001,n_gens)
    best = Pop_Aptitude(P)
    index = np.argmax(best)
    a = Ind2Number(P[index],v_min,v_max,n_var,n_bits)

    fx= abs(a[0])**2+abs(a[1])**3+abs(a[2])**4+abs(a[3])**5+abs(a[4])**6
    print("la mejor solucion encontrada es: ", fx)
    genetico += [fx]
```

```
la mejor solucion encontrada es:  0.03052114999504249
la mejor solucion encontrada es:  0.07191635230323133
la mejor solucion encontrada es:  0.1440756253598673
la mejor solucion encontrada es:  0.11818442964047454
la mejor solucion encontrada es:  0.04629345030046988
la mejor solucion encontrada es:  0.22969538375150533
la mejor solucion encontrada es:  0.017444577326652957
la mejor solucion encontrada es:  0.12970413767463346
la mejor solucion encontrada es:  0.18967381822009563
la mejor solucion encontrada es:  0.001977402456556079
la mejor solucion encontrada es:  0.1758431287790453
la mejor solucion encontrada es:  0.9710247025724302
la mejor solucion encontrada es:  0.23606438221723217
la mejor solucion encontrada es:  0.16120122330048858
la mejor solucion encontrada es:  0.14139562425473867
la mejor solucion encontrada es:  0.04440574328000966
la mejor solucion encontrada es:  0.025894814153292513
la mejor solucion encontrada es:  0.7990529631147743
la mejor solucion encontrada es:  2.4848730415966283
la mejor solucion encontrada es:  0.2656772750593379
```

[164]:
```
data = {'Algoritmos genéticos':pandas.Series(genetico),'Descenso de gradiente':
 ↪pandas.Series(gradientes)}
data_frame = pandas.DataFrame(data)
print(data_frame.describe())
```

```
       Algoritmos genéticos  Descenso de gradiente
count             20.000000              20.000000
mean               0.314246               0.022250
std                0.568268               0.010219
min                0.001977               0.007148
25%                0.045822               0.013774
```

```
50%         0.142736        0.022368
75%         0.231288        0.027406
max         2.484873        0.048320
```

6 Conclusión

Como pudimos obervar en las comparaciones, los algoritmos de búsqueda que se usan depende enteramente de la función a las que se les implementamos, aunque el algoritmo genético tarda más que el descenso de gradiente y ocupa un poco más de recursos, nomrlamente encuentra el mínimo global con buena preciosión, en cambio, el descenso de gradiente no siempre encuentra el mínimo global, pero si la encuentra, la precisión es mayor, es por ello que ninguna es mejor que otra, son herramientas que ocupamos en un trabajo diferente, no porque no puedas martillar con un destornillador significa que el destornillador sea inútil.

[47]:
```
def IniciaPob(N,n_var,n_bits):
    Pob = []
    for i in range(N):
        I = ''
        for v in range(n_var):
            for b in range(n_bits):
                I = I+sample(['0','1'],1)[0]
        Pob.append(I)
    return Pob
```

[48]:
```
def Ind2Number(I, v_min, v_max, n_vars, n_bits):
    numb = []
    for i in range(n_vars):
        cod_v = I[i*n_bits:(i+1)*n_bits]
        d_i = int(cod_v,2)
        val = v_min[i]+d_i*(v_max[i]-v_min[i])/float(2**n_bits-1)
        numb.append(val)
    return numb
```

[49]:
```
def Pop_Aptitude(P):
    apt_v = []
    for I in P:
        ap = Aptitude(I)
        #print 'Individuo ' + I + ', aptitud = '  + str(ap)
        apt_v.append(ap)
    return apt_v
```

[50]:
```
def CrossOver(I1,I2):
    n = len(I1)
    p = randint(1,n-2)
    b1 = I1[:p]
    b2 = I2[p:]
    return b1+b2
```

[51]:
```
#selecciona un individuo de la población de acuerdo a su aptitud
def select(P,Aptitude):
    apt_acum = sum(Aptitude)
    r = random()*apt_acum
    s = 0
    for i,I in enumerate(P):
        s = s+Aptitude[i]
        if s >= r:
            return I
    return I
```

[52]:
```
#probabilidad de cruza = 100%
def CrossOverPop(P,Aptitude):
    CrossP = []
    for I in P:
        I1 = select(P,Aptitude)
        I2 = select(P,Aptitude)
        nI = CrossOver(I1,I2)
        CrossP.append(nI)
    return CrossP
```

[53]:
```
def Mutation(P,pm):
    PM = []
    cs = {'0':'1','1':'0'}
    for I in P:
        nI = ''
        for b in I:
            if random() < pm:
                nI = nI+cs[b]
            else:
                nI = nI+b
        PM.append(nI)
    return PM
```

[54]:
```
def SelectP(P1,P2,A1,A2):
    PS = []
    for i in range(len(P1)):
        if A1[i] > A2[i]:
            PS.append(P1[i])
        else:
            PS.append(P2[i])
    return PS
```

[55]:
```
def Evolution(P0,pm,gnrtns):
    P = P0
    for g in range(gnrtns):
        AP = Pop_Aptitude(P)
```

```
        PC = CrossOverPop(P,AP)
        PM = Mutation(PC,pm)
        AM = Pop_Aptitude(PM)
        P = SelectP(P,PM,AP,AM)
    return P
```