Wyklad9

December 11, 2019

1 Moduł NumPy

[1]:
```
import time, random
```

[1]:
```
import numpy as np
```

[3]:
```
a=[random.random() for i in range(10**7)] #wektor 1d generowany losowo
b=[random.random() for i in range(10**7)]
```

[2]:
```
x = np.array([3, 1, 2])
```

Chcemy przeprowadzić mnożenie a i b (mnożenie ich elementów)

[4]:
```
t0=time.time()
c0=[]
for i in range(len(a)):
    c0.append(a[i]*b[i])
tk=time.time()
print(tk-t0)
```

```
4.36263632774353
```

[5]:
```
t0=time.time()
c1=[a[i]*b[i] for i in range(len(a))]
tk=time.time()
print(tk-t0)
```

```
2.4249751567840576
```

A teraz wykorzystajmy moduł NumPy

[6]:
```
a1=np.array(a)
b1=np.array(b)
```

[7]:
```
t0=time.time()
c2=a1*b1        #jak widać, składnia jest trywialna
tk=time.time()
print(tk-t0)
```

```
0.6150147914886475
```

[8]:
```python
del(c0,c1,c2)
```

Dlaczego tak jest? I jak to działa?

[9]:
```python
a = np.array([2,3,4])
b = np.array([1.2, 3.5, 5.1])
```

[10]:
```python
print(type(a))
```

```
<class 'numpy.ndarray'>
```

[11]:
```python
a.dtype
```

[11]:
```
dtype('int64')
```

[12]:
```python
b.dtype
```

[12]:
```
dtype('float64')
```

[13]:
```python
c = np.array([(1.5,2,3), (4,5,6)])
c  #elementy: rzutowanie na float
```

[13]:
```
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

[14]:
```python
d = np.array( [ [1,2], [3,4] ], dtype=complex ) #rzutowanie na complex
d
```

[14]:
```
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
```

Inne sposoby definiowania tablic

[15]:
```python
np.arange(0,1.5,0.1) #w przeciwieństwie do range(), tutaj mażemy pracować z float
```

[15]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4])
```

[16]:
```python
a=np.linspace( 0, 2, 9 ) #np.linspace( start, stop, ile_punktow )
a
```

[16]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

[17]:
```python
len(_)
```

[17]:
```
9
```

[18]:
```
np.arange(15).reshape(3, 5)
```

[18]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

[21]:
```
np.zeros((3,4)) #tablica "zer"
```

[21]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[2]:
```
d=np.ones((2,3,4),dtype='int') #tablica "jedynek"
d
```

[2]:
```
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]])
```

[22]:
```
rand_array=np.random.random((4,5)) #tablica "liczb pseudo-losowych"
rand_array
```

[22]:
```
array([[0.42919243, 0.47119496, 0.56110931, 0.18278576, 0.1240371 ],
       [0.25922231, 0.61770581, 0.11814958, 0.22529839, 0.45897044],
       [0.95116095, 0.63537319, 0.11060498, 0.41790037, 0.12641249],
       [0.97768212, 0.52469775, 0.69350203, 0.11076307, 0.44789166]])
```

[23]:
```
rand_array.shape  #(wiersze, kolumny)
```

[23]:
```
(4, 5)
```

[24]:
```
d.shape
```

[24]:
```
(2, 3, 4)
```

[25]:
```
print("Ndim(a) =", a.ndim, "; Ndim(rand_array) =", rand_array.ndim, "; Ndim(d)␣
↪=", d.ndim)
```

```
Ndim(a) = 1 ; Ndim(rand_array) = 2 ; Ndim(d) = 3
```

Rzutowanie

[3]:
```
d.astype(float)
```

[3]:
```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

[4]:
```
d
```

[4]:
```
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]])
```

Sortowanie

[28]:
```
print(rand_array)
```

```
[[0.42919243 0.47119496 0.56110931 0.18278576 0.1240371 ]
 [0.25922231 0.61770581 0.11814958 0.22529839 0.45897044]
 [0.95116095 0.63537319 0.11060498 0.41790037 0.12641249]
 [0.97768212 0.52469775 0.69350203 0.11076307 0.44789166]]
```

[29]:
```
rand_array.sort()
print(rand_array)
```

```
[[0.1240371  0.18278576 0.42919243 0.47119496 0.56110931]
 [0.11814958 0.22529839 0.25922231 0.45897044 0.61770581]
 [0.11060498 0.12641249 0.41790037 0.63537319 0.95116095]
 [0.11076307 0.44789166 0.52469775 0.69350203 0.97768212]]
```

[30]:
```
rand_array.min()
```

[30]:
```
0.1106049797261186
```

[31]:
```
rand_array.max()
```

[31]:
```
0.9776821179906275
```

[32]:
```
rand_array.sum()
```

[32]:
```
8.443654722808894
```

[33]:
```
rand_array.sum(axis=0)
```

[33]: array([0.46355473, 0.98238831, 1.63101287, 2.25904062, 3.10765819])

[34]:
```
rand_array.sum(axis=1)
```

[34]: array([1.76831957, 1.67934654, 2.24145198, 2.75453663])

Podstawowe operacje

[35]:
```
a = np.array([[1,2],[0,3]])
b = np.array([[2,0],[3,4]])
```

[36]:
```
print(a*b)
```

```
[[ 2  0]
 [ 0 12]]
```

[37]:
```
print(a+b)
```

```
[[3 2]
 [3 7]]
```

[38]:
```
print(a@b) #alternatywnie np.dot(a,b) lub a.dot(b)
```

```
[[ 8  8]
 [ 9 12]]
```

[6]:
```
d+=3.    #mogę dodać 3 ale nie 3.!!
print(d)
```

```
␣
↪---------------------------------------------------------------------------

    UFuncTypeError                            Traceback (most recent call last)

    <ipython-input-6-ab02371dc90d> in <module>
----> 1 d+=3.    #mogę dodać 3 ale nie 3.!!
      2 print(d)


    UFuncTypeError: Cannot cast ufunc 'add' output from dtype('float64') to␣
↪dtype('int64') with casting rule 'same_kind'
```

[7]:
```
d*=2
print(d)
```

```
[[[8 8 8 8]
  [8 8 8 8]
  [8 8 8 8]]

 [[8 8 8 8]
  [8 8 8 8]
  [8 8 8 8]]]
```

[44]:
```python
np.vstack((a,b)) #np.concatenate((a,b),axis=0)
```

[44]:
```
array([[1, 2],
       [0, 3],
       [2, 0],
       [3, 4]])
```

[45]:
```python
np.hstack((a,b)) #np.concatenate((a,b),axis=1)
```

[45]:
```
array([[1, 2, 2, 0],
       [0, 3, 3, 4]])
```

[46]:
```python
np.concatenate((a,b),axis=1)
```

[46]:
```
array([[1, 2, 2, 0],
       [0, 3, 3, 4]])
```

[47]:
```python
d.reshape(6,4)
```

[47]:
```
array([[8, 8, 8, 8],
       [8, 8, 8, 8],
       [8, 8, 8, 8],
       [8, 8, 8, 8],
       [8, 8, 8, 8],
       [8, 8, 8, 8]])
```

Funkcje uniwersalne

[48]:
```python
x = np.linspace(-2, 2, 6)
print(x)
```

```
[-2.  -1.2 -0.4  0.4  1.2  2. ]
```

[49]:
```python
print(1/x)
```

```
[-0.5        -0.83333333 -2.5         2.5         0.83333333  0.5       ]
```

[50]:
```python
print(np.exp(x))
```

```
[0.13533528 0.30119421 0.67032005 1.4918247  3.32011692 7.3890561 ]
```

[51]:
```python
from math import exp as fexp
fexp(-2)
```

[51]: 0.1353352832366127

[52]:
```python
h=np.ones_like(x)
print(h)
```

```
[1. 1. 1. 1. 1. 1.]
```

[53]:
```python
h[x<-0.5]=0.
h[x>0.5]=0.
print(h)
```

```
[0. 0. 1. 1. 0. 0.]
```

[54]:
```python
def f(x,y):
    return 10*x+y

b = np.fromfunction(f,(5,4),dtype=int)
print(b)
```

```
[[ 0  1  2  3]
 [10 11 12 13]
 [20 21 22 23]
 [30 31 32 33]
 [40 41 42 43]]
```

Algebra liniowa

[55]:
```python
a = np.array([[1.0, 2.0], [3.0, 4.0]])
```

[56]:
```python
a.transpose()
```

[56]:
```
array([[1., 3.],
       [2., 4.]])
```

[57]:
```python
a_inv=np.linalg.inv(a)
a_inv
```

[57]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

[58]:
```python
print(a.dot(a_inv))
```

```
[[1.00000000e+00 1.11022302e-16]
 [0.00000000e+00 1.00000000e+00]]
```

Podstawowy problem algebry liniowej : a*x=b, znajdź x

[60]:
```
a=np.array([[3,2,1],[5,5,5],[1,4,6]])
b=np.array([[5,1],[5,0],[-3,-7.0/2]])
```

[61]:
```
x=np.linalg.solve(a,b)
```

[62]:
```
a.dot(x)-b
```

[62]:
```
array([[ 0.00000000e+00, -6.66133815e-16],
       [ 0.00000000e+00,  8.88178420e-16],
       [-1.77635684e-15,  1.33226763e-15]])
```

[63]:
```
print(x)
```

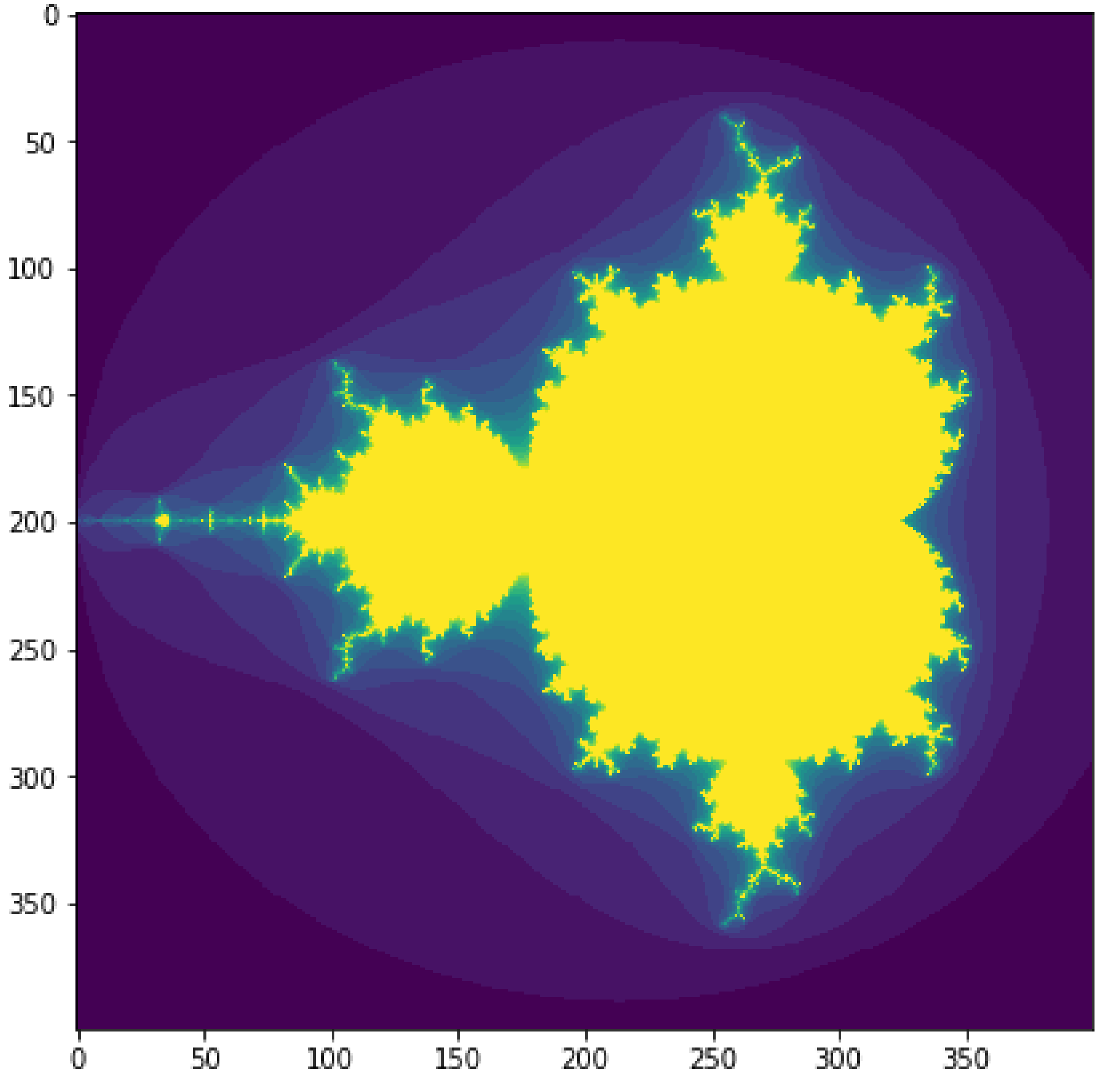
```
[[ 1.   1.5]
 [ 2.  -2. ]
 [-2.   0.5]]
```

2 Zastosowanie - czyli matplotlib (lub pylab)

[64]:
```
import matplotlib.pyplot as plt
```

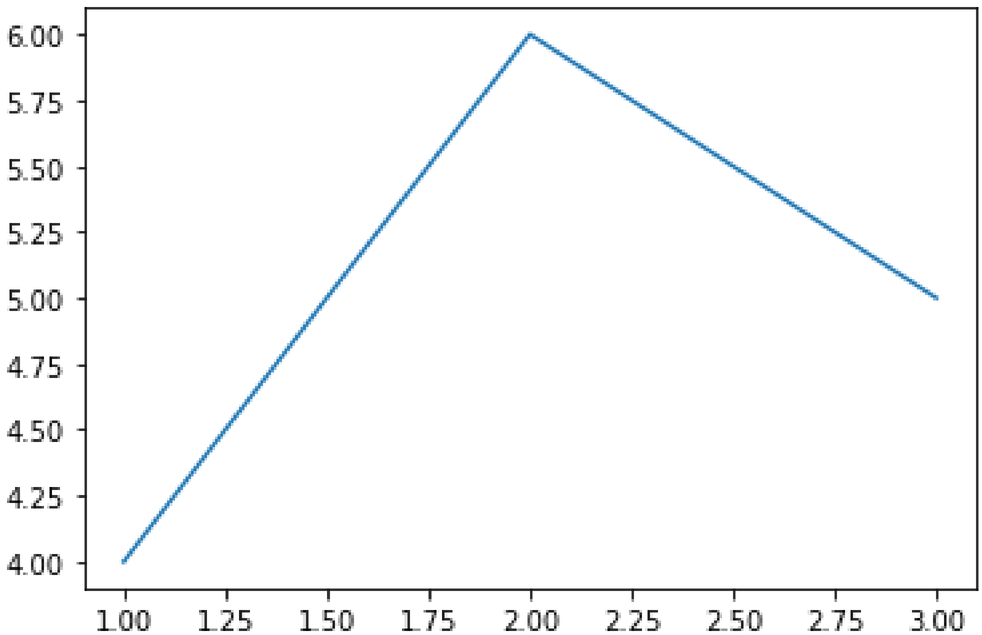
[65]:
```
def mandelbrot( h,w, maxit=20 ):
    y,x = np.ogrid[ -1.4:1.4:h*1j, -2:0.8:w*1j ]
    c = x+y*1j
    z = c
    divtime = maxit + np.zeros(z.shape, dtype=int)

    for i in range(maxit):
        z = z**2 + c
        diverge = z*np.conj(z) > 2**2
        div_now = diverge & (divtime==maxit)
        divtime[div_now] = i
        z[diverge] = 2

    return divtime
```

[66]:
```
plt.figure(figsize=(7,7))
plt.imshow(mandelbrot(400,400))
plt.show()
```



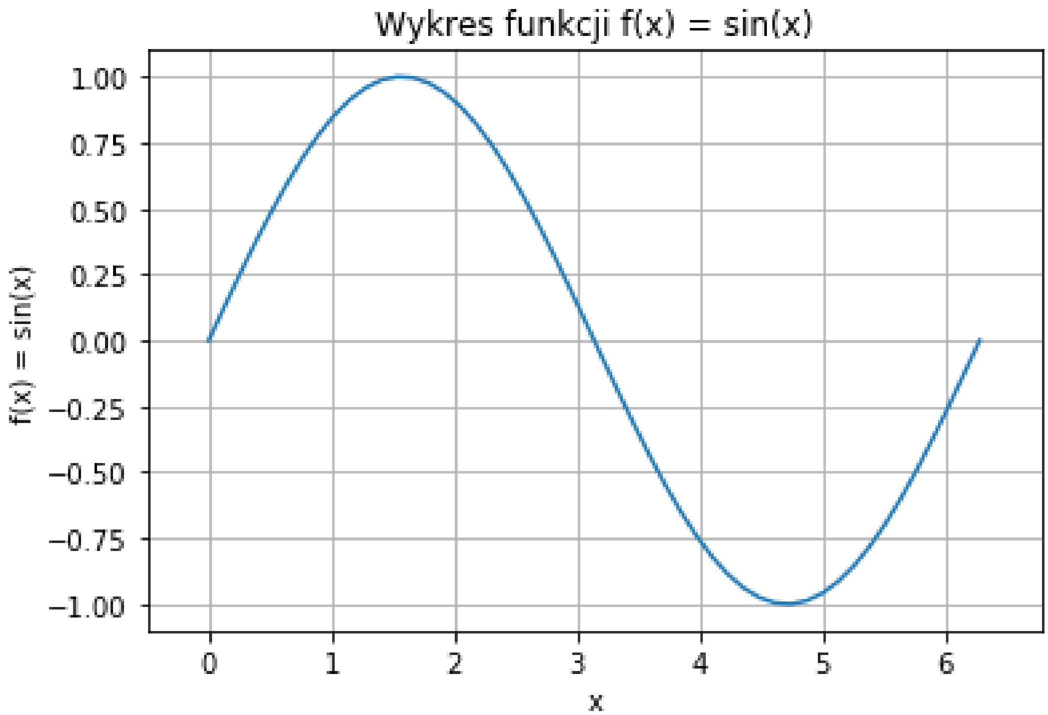
Zacznijmy od czegoś prostszego

[67]:
```
x = [1,2,3]
y = [4,6,5]

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

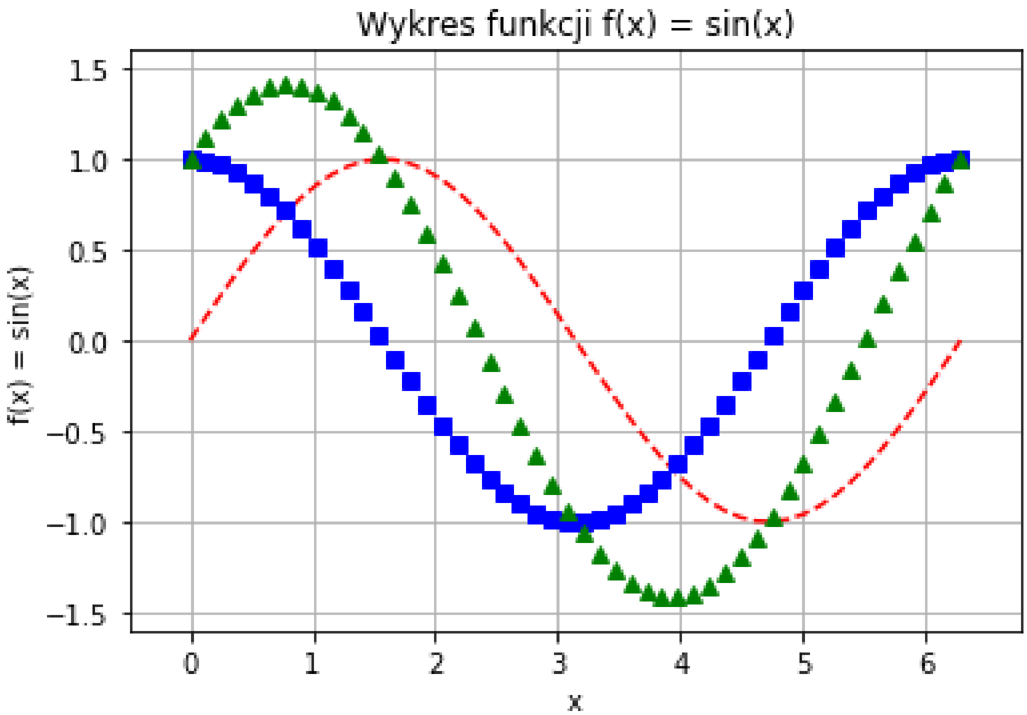


[68]:
```
x = np.linspace(0, np.pi * 2, 50)
y = np.sin(x)
z = np.cos(x)
```

[69]:
```
plt.plot(x, y)
plt.grid(True)
plt.xlim(-0.5, np.pi * 2+0.5)
plt.ylim(-1.1, 1.1)
plt.xlabel("x")
plt.ylabel("f(x) = sin(x)")
plt.title("Wykres funkcji f(x) = sin(x)")
#plt.savefig("fig1.jpg", dpi = 72)
plt.show()
```

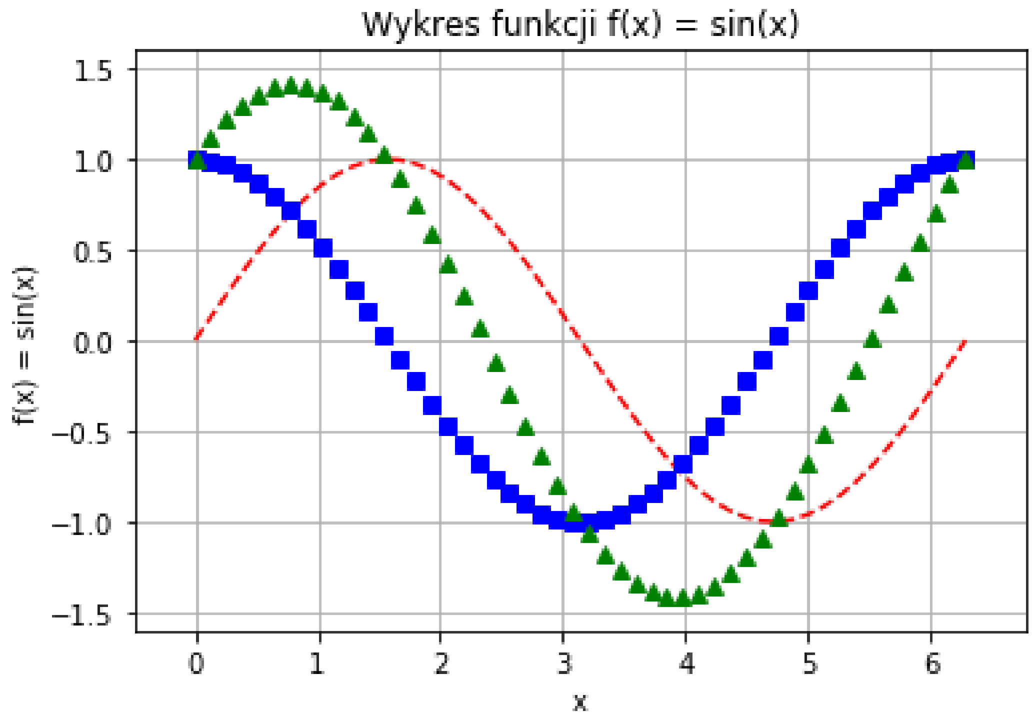


[70]:
```
plt.plot(x, y, 'r--', x, z, 'bs', x, y+z, 'g^')
plt.grid(True)
#plt.xlim(-0.5, np.pi * 2+0.5)
#plt.ylim(-1.6, 1.6)
plt.axis([-0.5, np.pi * 2+0.5, -1.6, 1.6])
plt.xlabel("x")
plt.ylabel("f(x) = sin(x)")
plt.title("Wykres funkcji f(x) = sin(x)")
#plt.savefig("fig1.jpg", dpi = 72)
plt.show()
```

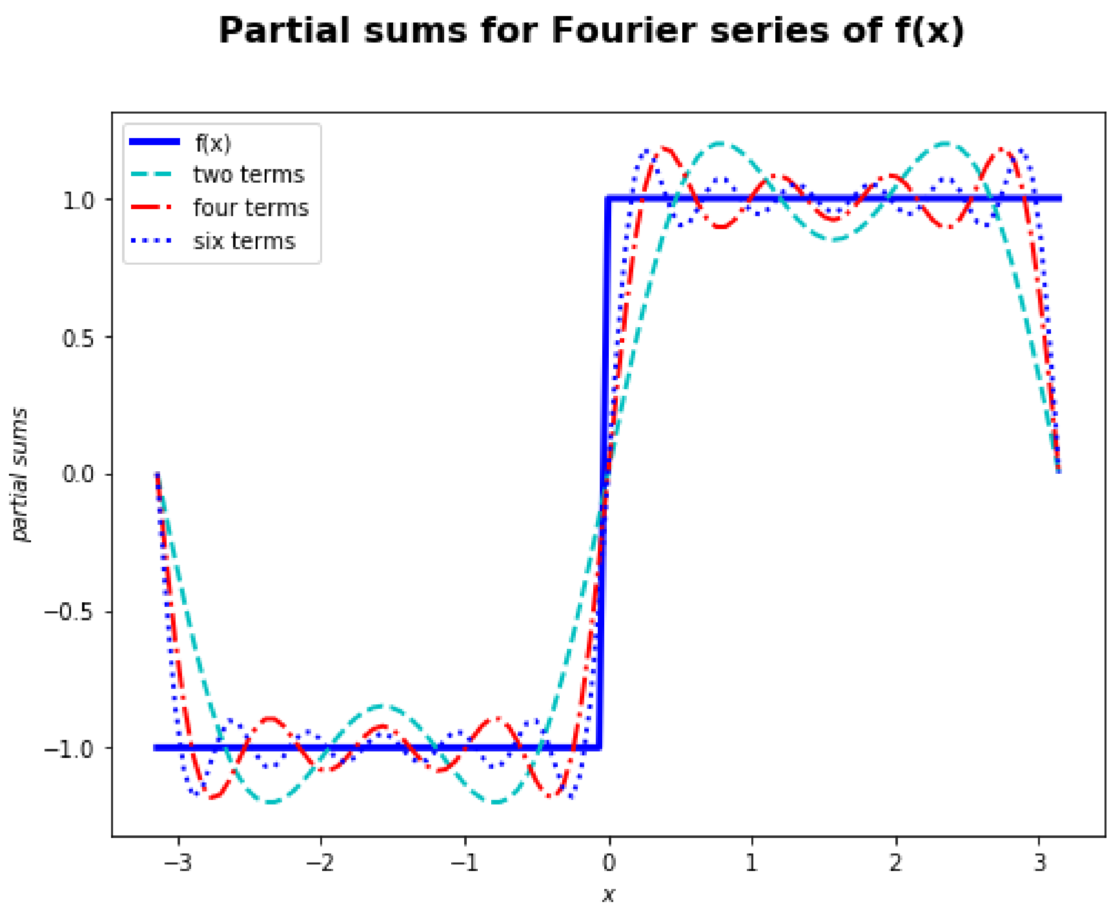



```
[ ]: help(plt.plot) #warto sprawdzic **Markers**, **Line Styles**, **Colors**
```

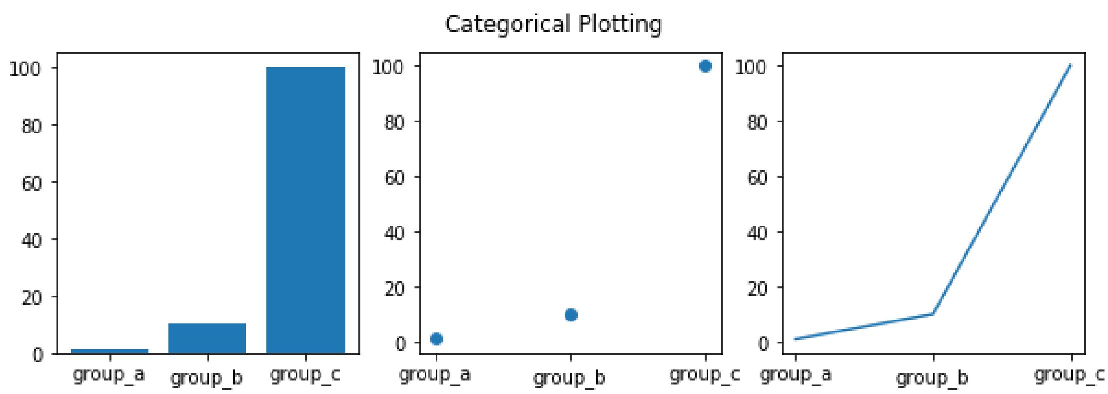
```
[72]: plt.plot(x, y, 'r--')
plt.plot(x, z, 'bs')
plt.plot(x, y+z, 'g^')
plt.grid(True)
#plt.xlim(-0.5, np.pi * 2+0.5)
#plt.ylim(-1.6, 1.6)
plt.axis([-0.5, np.pi * 2+0.5, -1.6, 1.6])
plt.xlabel("x")
plt.ylabel("f(x) = sin(x)")
plt.title("Wykres funkcji f(x) = sin(x)")
#plt.savefig("fig1.jpg", dpi = 72)
plt.show()
```



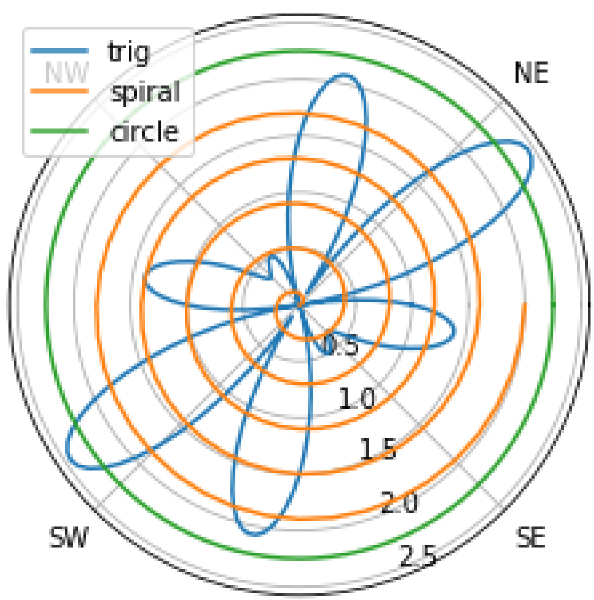
[73]:
```
x=np.linspace(-np.pi,np.pi,101)
f=np.ones_like(x)
f[x<0]=-1
y1=(4/np.pi)*(np.sin(x)+np.sin(3*x)/3.0)
y2=y1+(4/np.pi)*(np.sin(5*x)/5.0+np.sin(7*x)/7.0)
y3=y2+(4/np.pi)*(np.sin(9*x)/9.0+np.sin(11*x)/11.0)

plt.figure(figsize=(8,6))
plt.plot(x,f,'b-',lw=3,label='f(x)')
plt.plot(x,y1,'c--',lw=2,label='two terms')
plt.plot(x,y2,'r-.',lw=2,label='four terms')
plt.plot(x, y3,'b:',lw=2,label='six terms')
plt.legend(loc='best')
plt.xlabel('x',style='italic')
plt.ylabel('partial sums',style='italic')
plt.suptitle('Partial sums for Fourier series of f(x)',size=16,weight='bold')
plt.show()
```



[74]:
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(10,3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```

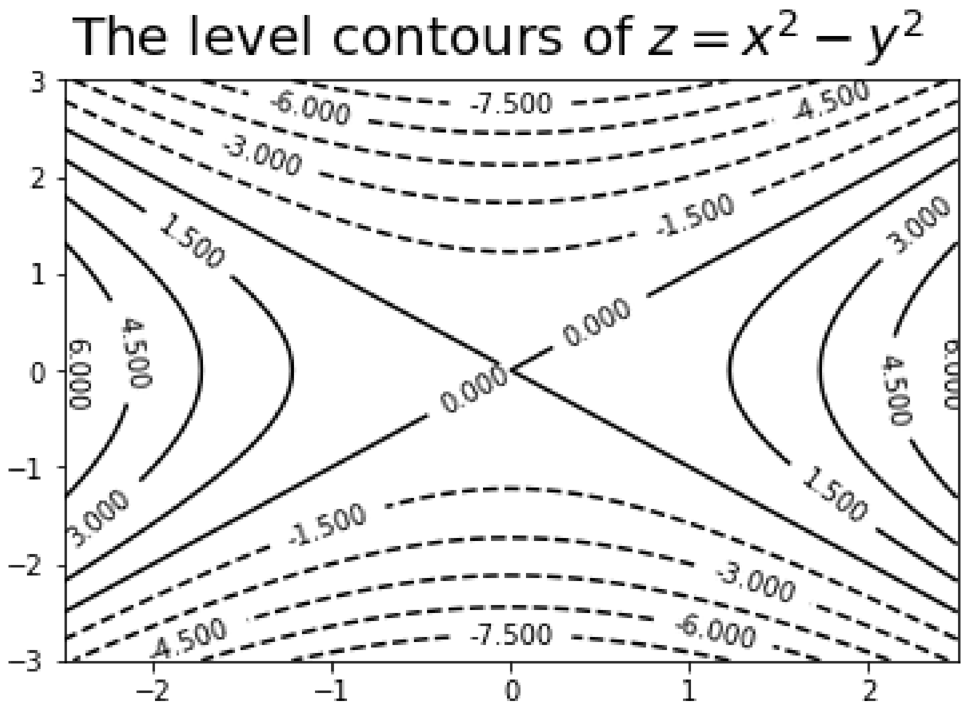


[76]:
```
theta=np.linspace(0,2*np.pi,201)
r1=np.abs(np.cos(5.0*theta) - 1.5*np.sin(3.0*theta))
r2=theta/np.pi
r3=2.25*np.ones_like(theta)

#plt.figure(figsize=(6,6))
plt.polar(theta, r1,label='trig')
plt.polar(5*theta, r2,label='spiral')
plt.polar(theta, r3,label='circle')
plt.thetagrids(np.arange(45,360,90), ('NE','NW','SW','SE'))
plt.rgrids((0.5,1.0,1.5,2.0,2.5),angle=290)
plt.legend(loc='best')
plt.show()
```



[77]:

```
[X,Y] = np.mgrid[-2.5:2.5:51j,-3:3:61j]
Z=X**2-Y**2
curves=plt.contour(X,Y,Z,12,colors='k')
plt.clabel(curves)
plt.suptitle('The level contours of $z=x^2-y^2$',fontsize=20)
plt.show()
```



[ ]: