Homework #3 과제(01)분반 20171703 정태원

p57

2.9.1 넘파이의 이용

In [1]:

```
import numpy as np
```

2.9.2 벡터의 정의

In [3]:

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

```
[1 2 3]
```

Out[3]:

```
array([1, 2, 3])
```

In [4]:

```
y = [1,2,3]
print(y)
```

```
[1, 2, 3]
```

In [5]:

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

Out[5]:

```
array([1, 2, 3])
```

In [6]:

```
x+y
```

Out[6]:

```
array([2, 4, 6])
```

In [7]:

```
print(x+y)
```

```
[2 4 6]
```

In [8]:

```
type(x+y)
```

Out[8]:

```
numpy.ndarray
```

In [9]:

```
x[0]
```

Out[9]:

```
1
```

In [10]:

```
x[0]=100
print(x)
```

```
[100   2   3]
```

2.9.5 연속된 정수 벡터의 생성

In [13]:

```
print(np.arange(10))
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [14]:

```
print(np.arange(3,19))
```

```
[ 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

In [16]:

```
j = np.array([4,5,6])
for i in np.arange(1, 4):
    print(i)
    print(i+j)
str(j)
```

```
1
[5 6 7]
2
[6 7 8]
3
[7 8 9]
```

Out[16]:

```
'[4 5 6]'
```

2.9.6 ndarray 형의 주의점

ndarray를 사용하려면 = 이 아닌 .copy()를 사용하도록 한다. = 을 사용하면 참조 주소(이는 array가 불가변수이기 때문이다.)를 복사하는 것이다. 같은 주소를 쓰기에 =를 받은 새로운 변수의 값은 서로에게 영향을 준다.

In [17]:

```
a = np.array([1,1])
b = a
print('a='+str(a))
print('b='+str(b))
b[0]=100
print('b=' + str(b))
print('a='+str(a))
```

```
a=[1 1]
b=[1 1]
b=[100   1]
a=[100   1]
```

In [18]:

```
a = np.array([1,1])
b = a.copy()
print('a='+str(a))
print('b='+str(b))
b[0]= 100
print('a='+str(a))
print('b='+str(b))
```

```
a=[1 1]
b=[1 1]
a=[1 1]
b=[100   1]
```

2.10 행렬

2.10.1 행렬의 정의

In [19]:

```
x = np.array([[1,2,3],[4,5,6]])
print(x)
```

```
[[1 2 3]
 [4 5 6]]
```

In [20]:

```
x
```

Out[20]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

2.10.2 행렬의 크기

In [25]:

```
x = np.array([[1,2,3],[4,5,6]])
x.shape
```

Out[25]:

```
(2, 3)
```

In [26]:

```
w,h = x.shape
print(w)
print(h)
```

```
2
3
```

In [27]:

```
type(x.shape)
```

Out[27]:

```
tuple
```

2.10.3 요소의 참조

In [30]:

```
x = np.array([[1,2,3],[4,5,6]])
print(x[1,2])
print(x[0,2])
```

```
6
3
```

2.10.4 요소의 수정

In [2]:

```
import numpy as np
x = np.array([[1, 2, 3],[4, 5, 6]])
x[1, 2] = 100
```

2.10.5 요소가 0과 1인 ndarray만들기

In [3]:

```
print(np.zeros(10))
```

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

In [7]:

```
print(np.zeros((2,10)))
```

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

In [8]:

```
print(np.ones((2, 10)))
```

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

In [9]:

```
print(np.ones((2,3,4)))
```

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

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

2.10.6 요소가 랜덤인 행렬 생성

In [11]:

```
np.random.rand(2, 3)
```

Out[11]:

```
array([[0.22945588, 0.7869296 , 0.88480885],
       [0.6263588 , 0.06807567, 0.0414397 ]])
```

In [12]:

```
np.random.randn(2,3)
```

Out[12]:

```
array([[ 1.26961716, -0.64021638,  0.24607543],
       [ 1.41715796,  0.33305623, -0.23230727]])
```

In [14]:

```
np.random.randint(1,10,3)
```

Out[14]:

```
array([2, 7, 3])
```

2.10.7 행렬의 크기 변경

In [15]:

```
a= np.arange(10)
print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [16]:

```
a.reshape(2,5)
```

Out[16]:

```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

In [17]:

```
a.reshape(5,2)
```

Out[17]:

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

2.11 행렬(ndarray)의 사칙 연산

In [23]:

```
x = np.array([[4, 4, 4],[8, 8, 8]])
y = np.array([[1, 1, 1],[2 ,2 ,2]])
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x+2)
print(x/4)
```

```
[[ 5  5  5]
 [10 10 10]]
[[3 3 3]
 [6 6 6]]
[[ 4  4  4]
 [16 16 16]]
[[4. 4. 4.]
 [4. 4. 4.]]
[[ 6  6  6]
 [10 10 10]]
[[1. 1. 1.]
 [2. 2. 2.]]
```

2.11.2 스칼라 * 행렬

In [24]:

```
x = np.array([[4, 4, 4],[8, 8, 8]])
print(10*x)
```

```
[[40 40 40]
 [80 80 80]]
```

2.11.3 산술 함수

In [25]:

```
x = np.array([[4,4,4],[8,8,8]])
print(np.exp(x))
```

```
[[  54.59815003   54.59815003   54.59815003]
 [2980.95798704 2980.95798704 2980.95798704]]
```

2.11.4 행렬 곱의 계산

In [26]:

```
v = np.array([[1,2,3],[4,5,6]])
w = np.array([[1,1],[2,2],[3,3]])
print(v.dot(w))
```

```
[[14 14]
 [32 32]]
```

2.12 슬라이싱

2.12.1 슬라이싱의 이용

In [29]:

```
x = np.arange(10)
print(x)
print(x[:5])
print(x[5:])
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4]
[5 6 7 8 9]
```

In [30]:

```
print(x[3:8])
```

```
[3 4 5 6 7]
```

In [31]:

```
print(x[3:8:2])
```

```
[3 5 7]
```

In [34]:

```
print(x[::-1])#역순
```

```
[9 8 7 6 5 4 3 2 1 0]
```

In [35]:

```
y = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(y)
print(y[:2, 1:2])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[2]
 [5]]
```

2.13 조건을 만족하는 데이터의 수정

2.13.1 bool 배열 사용

In [37]:

```
x = np.array([1, 1, 2, 3, 5, 8, 13])
x > 3
```

Out[37]:

```
array([False, False, False, False,  True,  True,  True])
```

In [38]:

```
x[x>3]
```

Out[38]:

```
array([ 5,  8, 13])
```

In [42]:

```
x[x>3]= 999
print(x)
```

```
[  1   1   2   3 999 999 999]
```

2.14 Help

2.14.1 Help 사용

In [45]:

```
help(np.random.randint)
np.random.randint(5)
np.random.randint(5,10,(2,3))
```

```
Help on built-in function randint:

randint(...) method of numpy.random.mtrand.RandomState instance
    randint(low, high=None, size=None, dtype='l')

    Return random integers from `low` (inclusive) to `high` (exclusi
ve).

    Return random integers from the "discrete uniform" distribution
of
    the specified dtype in the "half-open" interval [`low`, `high`).
If
    `high` is None (the default), then results are from [0, `low`).

    .. note::
        New code should use the ``integers`` method of a ``default_r
ng()``
        instance instead; see `random-quick-start`.

    Parameters
```

2.15 함수

2.15.1 함수의 사용

In [47]:

```
def my_func1():
    print('hi')
#함수 my_func1()의 정의는 여기까지
my_func1() # 함수의 실행
```

```
hi
```

In [48]:

```
def my_func2(a,b):
    c = a + b
    return c
my_func2(1, 2)
```

Out[48]:

```
3
```

2.15.2 인수와 반환값

In [51]:

```
def my_func3(D):
    m = np.mean(D)
    s = np.std(D)
    return m, s
```

In [80]:

```
data = np.random.randn(100)
data_mean, data_std = my_func3(data)
print(data_mean, data_std,
      'mean:{0:3.2f}, std:{1:3.2f}'.format(data_mean, data_std))
```

```
0.08893441922599521 0.9358307316602601 mean:0.09, std:0.94
```

In [81]:

```
output = my_func3(data)
print(output)
print(type(output))
print('mean:{0:3.2f}, std:{1:3.2f}'.format(output[0],output[1]))
```

```
(0.08893441922599521, 0.9358307316602601)
<class 'tuple'>
mean:0.09, std:0.94
```

2.16 파일 저장

In [83]:

```
data = np.random.randn(5)
print(data)
np.save("randn.npy",data)
data = []
print(data)
data = np.load("randn.npy")
print(data)
```

```
[-0.47501627 -0.42275021  0.87746272  0.26815715 -1.18991604]
[]
[-0.47501627 -0.42275021  0.87746272  0.26815715 -1.18991604]
```

2.16.2 여러 ndarray 형을 저장

In [90]:

```
data1 = np.array([1,2,3])
data2 = np.array([10, 20, 30])
np.savez('datafilez.npz',data1=data1,data2=data2)
data1 = []
data2 = []
outfile=np.load('datafilez.npz')
print(outfile)
print(outfile.files)
print(type(outfile))
print(type(outfile.files))
data1 = outfile['data1']
data2 = outfile['data2']
print(data1)
print(data2)
```

```
<numpy.lib.npyio.NpzFile object at 0x7fca787ead90>
['data1', 'data2']
<class 'numpy.lib.npyio.NpzFile'>
<class 'list'>
[1 2 3]
[10 20 30]
```

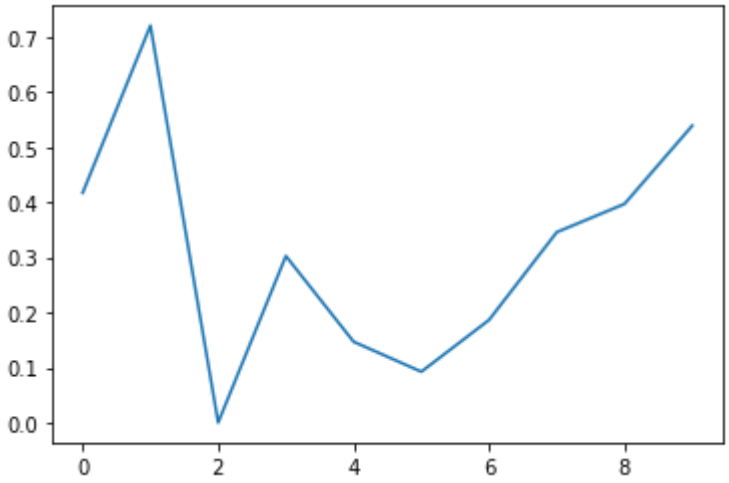
chapter 3 그래프 그리기

3.1 2차원 그래프 그리기

3.1.1 임의의 그래프 그리기

In [100]:

```
#list1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#data 작성
np.random.seed(1)
x = np.arange(10)
y = np.random.rand(10)

#그래프 표시
plt.plot(x, y) #꺾은선 그래프를 등록
plt.show()
```



3.1.2 프로그램 리스트 규칙

In [101]:

```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

3.1.3 3차 함수 f(x) =(x-2)x(x+2)그리기

In [102]:

```
#리스트 2-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
def f(x):
    return (x-2)*x*(x+2)
```

In [103]:

```
print(f(1))
```

```
-3
```

In [105]:

```
#리스트 2-(3)
print(f(np.array([1,2,3])))
```

```
[-3  0 15]
```

3.1.4 그리는 범위를 결정하기

In [107]:

```
help(np.arange)
```

```
Help on built-in function arange in module numpy:

arange(...)
    arange([start,] stop[, step,], dtype=None)

    Return evenly spaced values within a given interval.

    Values are generated within the half-open interval ``[start, stop)
``
    (in other words, the interval including `start` but excluding `sto
p`).
    For integer arguments the function is equivalent to the Python bui
lt-in
    `range` function, but returns an ndarray rather than a list.

    When using a non-integer step, such as 0.1, the results will often
not
    be consistent.  It is better to use `numpy.linspace` for these cas
es.

    Parameters
    ----------
    start : number, optional
        Start of interval.  The interval includes this value.  The def
ault
        start value is 0.
    stop : number
        End of interval.  The interval does not include this value, ex
cept
        in some cases where `step` is not an integer and floating poin
t
        round-off affects the length of `out`.
    step : number, optional
        Spacing between values.  For any output `out`, this is the dis
tance
        between two adjacent values, ``out[i+1] - out[i]``.  The defau
lt
        step size is 1.  If `step` is specified as a position argumen
t,
        `start` must also be given.
    dtype : dtype
        The type of the output array.  If `dtype` is not given, infer
the data
        type from the other input arguments.

    Returns
    -------
    arange : ndarray
        Array of evenly spaced values.

        For floating point arguments, the length of the result is
        ``ceil((stop - start)/step)``.  Because of floating point over
flow,
        this rule may result in the last element of `out` being greate
r
        than `stop`.
```

```
    See Also
    --------
    numpy.linspace : Evenly spaced numbers with careful handling of en
dpoints.
    numpy.ogrid: Arrays of evenly spaced numbers in N-dimensions.
    numpy.mgrid: Grid-shaped arrays of evenly spaced numbers in N-dime
nsions.

    Examples
    --------
    >>> np.arange(3)
    array([0, 1, 2])
    >>> np.arange(3.0)
    array([ 0.,  1.,  2.])
    >>> np.arange(3,7)
    array([3, 4, 5, 6])
    >>> np.arange(3,7,2)
    array([3, 5])
```

In [109]:

```
#리스트 2-(4)
x = np.arange(-3, 3.5, 0.5)#주의 - -3, 3까지 하면 2.5까지 다루게 된다.
print(x)
```

```
[-3.  -2.5 -2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2.   2.5  3. ]
```

In [112]:

```
#리스트 2-(5)
x = np.linspace(-3,3,10)
print(np.round(x, 2))
```

```
[-3.   -2.33 -1.67 -1.   -0.33  0.33  1.    1.67  2.33  3.  ]
```

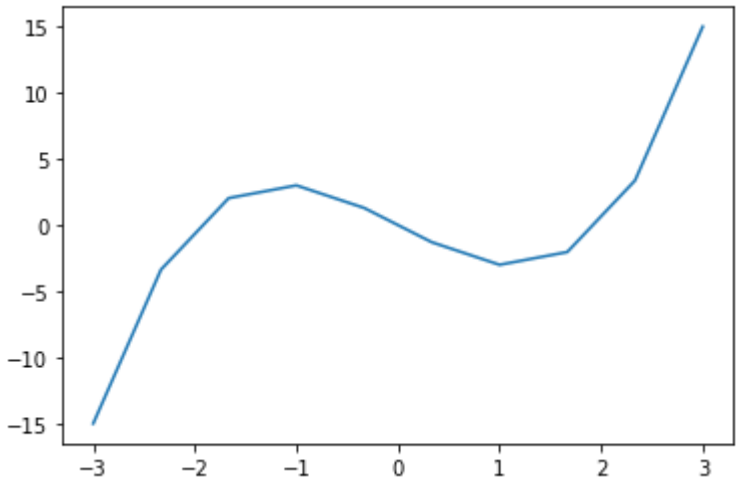
3.1.5. 그래프 그리기

In [113]:

```
#리스트 2-(6)
plt.plot(x,f(x))
```

Out[113]:

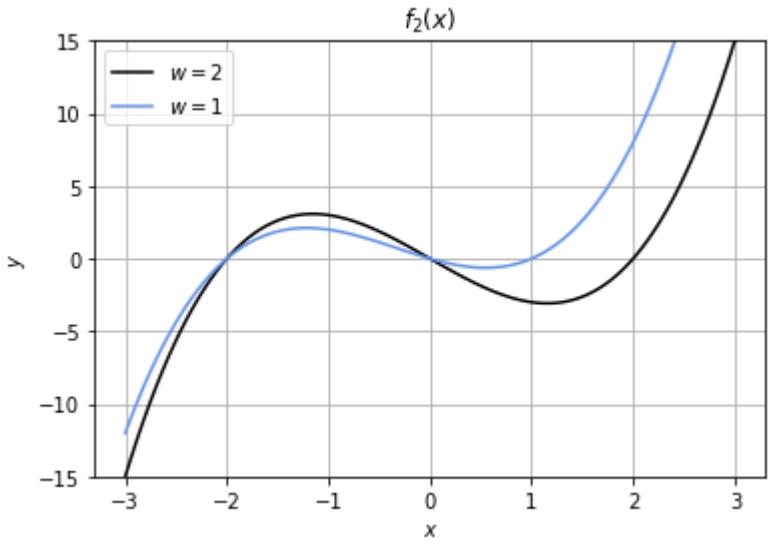
```
[<matplotlib.lines.Line2D at 0x7fca6c0c18d0>]
```



3.1.6 그래프를 장식하기

In [115]:

```
#2-(7)
#함수를 정의


def f2(x, w):
    return ( x - w)*x*(x+2) #(A) 함수 정의

#x를 정의
x = np.linspace(-3,3,100)#(B)) x를 100분할하기

#차트 묘사
plt.plot(x, f2(x, 2), color='black', label = '$w=2$')#(C)
plt.plot(x, f2(x, 1), color='cornflowerblue', label = '$w = 1$') #(D)
plt.legend(loc = "upper left") #(E) 범례 표시
plt.ylim(-15, 15) #(F) y축의 범위
plt.title('$f_2(x)$') #(G) 제목
plt.xlabel('$x$') #(H) x라벨
plt.ylabel('$y$') #(I) y라벨
plt.grid(True) #(J) 그리드
plt.show()
```



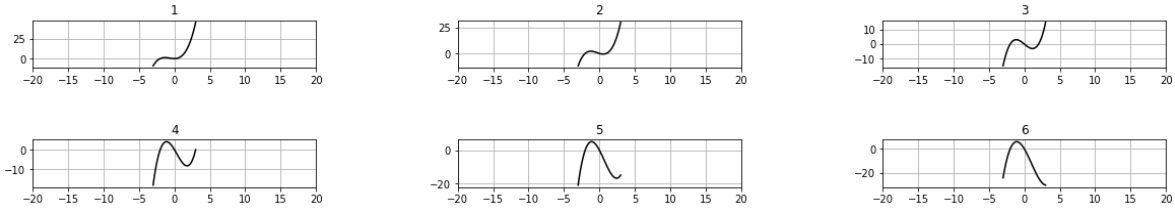
In [116]:

```
#리스트 2-(8)
import matplotlib
matplotlib.colors.cnames
```

```
 'dimgrey': '#696969',
 'dodgerblue': '#1E90FF',
 'firebrick': '#B22222',
 'floralwhite': '#FFFAF0',
 'forestgreen': '#228B22',
 'fuchsia': '#FF00FF',
 'gainsboro': '#DCDCDC',
 'ghostwhite': '#F8F8FF',
 'gold': '#FFD700',
 'goldenrod': '#DAA520',
 'gray': '#808080',
 'green': '#008000',
 'greenyellow': '#ADFF2F',
 'grey': '#808080',
 'honeydew': '#F0FFF0',
 'hotpink': '#FF69B4',
 'indianred': '#CD5C5C',
 'indigo': '#4B0082',
 'ivory': '#FFFFF0',
```

In [122]:

```
# 리스트 2-(9)
plt.figure(figsize=(20,3)) #(A) figure 지정
plt.subplots_adjust(wspace=0.5, hspace=1.5) #(B) 그래프의 간격을 지정
for i in range(6):
    plt.subplot(2, 3, i+1) # (C)그래프 묘사의 위치를 지정
    plt.title(i+1)
    plt.plot(x, f2(x, i), 'k')
    plt.xlim(-20, 20)
    plt.grid(True)
plt.show()
```



3.2 3차원 그래프 그리기

3.2.1 이변수 함수

In [3]:

```
#리스트 3-(1)
import numpy as np
import matplotlib.pyplot as plt

# 함수 f3를 정의
def f3(x0, x1):
    r = 2 * x0 **2 + x1 **2
    ans =  r*np.exp(-r)
    return ans
# x0, x1에서 각각 f3를 계산
xn = 50
x0 = np.linspace(-2,2,xn) #(A)
x1 = np.linspace(-2,2,xn) #(B)
y = np.zeros((len(x0), len(x1))) #(C)
for i0 in range(xn):
    for i1 in range(xn):[[]]
        y[i1, i0]= f3(x0[i0], x1[i1]) #(D)
```

```
  File "<ipython-input-3-dc3118b44f23>", line 17
    y[i1, i0]= f3(x0[i0], x1[i1]) #(D)
    ^
IndentationError: unexpected indent
```

In [127]:

```
#리스트 3-(2)
print(x0)
print(y)# 복잡함
```

```
[-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
[[7.37305482e-05 1.72948114e-03 1.48725131e-02 4.99904844e-02
  7.32625556e-02 4.99904844e-02 1.48725131e-02 1.72948114e-03
  7.37305482e-05]
 [3.62414384e-04 7.90343744e-03 6.06229941e-02 1.75801618e-01
  2.37148255e-01 1.75801618e-01 6.06229941e-02 7.90343744e-03
  3.62414384e-04]
 [1.11068824e-03 2.24772429e-02 1.49361205e-01 3.34695240e-01
  3.67879441e-01 3.34695240e-01 1.49361205e-01 2.24772429e-02
  1.11068824e-03]
 [2.15538310e-03 4.10955522e-02 2.37148255e-01 3.54274915e-01
  1.94700196e-01 3.54274915e-01 2.37148255e-01 4.10955522e-02
  2.15538310e-03]
 [2.68370102e-03 4.99904844e-02 2.70670566e-01 3.03265330e-01
  0.00000000e+00 3.03265330e-01 2.70670566e-01 4.99904844e-02
  2.68370102e-03]
 [2.15538310e-03 4.10955522e-02 2.37148255e-01 3.54274915e-01
  1.94700196e-01 3.54274915e-01 2.37148255e-01 4.10955522e-02
  2.15538310e-03]
 [1.11068824e-03 2.24772429e-02 1.49361205e-01 3.34695240e-01
  3.67879441e-01 3.34695240e-01 1.49361205e-01 2.24772429e-02
  1.11068824e-03]
 [3.62414384e-04 7.90343744e-03 6.06229941e-02 1.75801618e-01
  2.37148255e-01 1.75801618e-01 6.06229941e-02 7.90343744e-03
  3.62414384e-04]
 [7.37305482e-05 1.72948114e-03 1.48725131e-02 4.99904844e-02
  7.32625556e-02 4.99904844e-02 1.48725131e-02 1.72948114e-03
  7.37305482e-05]]
```

In [128]:

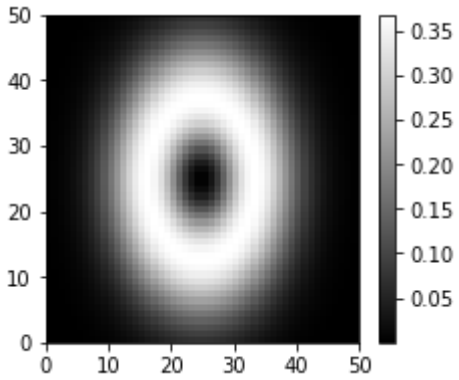
```
#리스트 3-(3)
print(np.round(y,1))
```

```
[[0.  0.  0.  0.  0.1 0.  0.  0.  0. ]
 [0.  0.  0.1 0.2 0.2 0.2 0.1 0.  0. ]
 [0.  0.  0.1 0.3 0.4 0.3 0.1 0.  0. ]
 [0.  0.  0.2 0.4 0.2 0.4 0.2 0.  0. ]
 [0.  0.  0.3 0.3 0.  0.3 0.3 0.  0. ]
 [0.  0.  0.2 0.4 0.2 0.4 0.2 0.  0. ]
 [0.  0.  0.1 0.3 0.4 0.3 0.1 0.  0. ]
 [0.  0.  0.1 0.2 0.2 0.2 0.1 0.  0. ]
 [0.  0.  0.  0.  0.1 0.  0.  0.  0. ]]
```

3.2.2 수치를 색으로 표현하기: pcolor

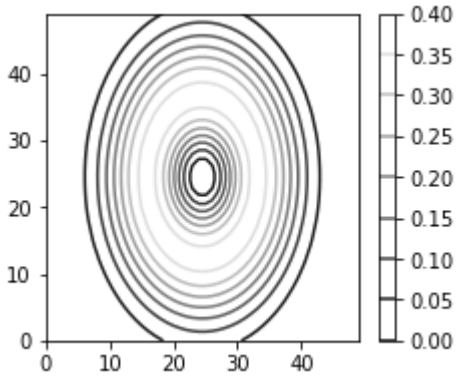
In [141]:

```
#리스트 3-(4)
plt.figure(figsize=(3.5,3))
plt.gray() #(A)
plt.pcolor(y) #(B)
plt.colorbar()#(C)

plt.show()
```



In [138]:

```
#리스트 3-(4)
xn= 50
plt.figure(figsize=(3.5,3))
plt.gray() #(A)
plt.contour(y)
plt.colorbar()#(C)

plt.show()
```

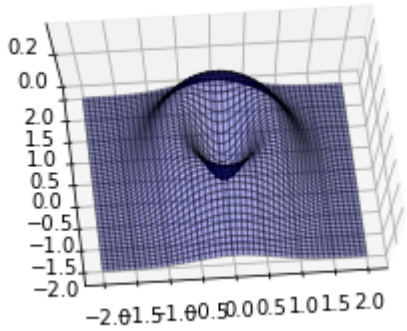


3.2.3 함수의 표면을 표시: surface

In [145]:

```
#리스트 3-(5)
from mpl_toolkits.mplot3d import Axes3D # (A)
xx0, xx1 = np.meshgrid(x0, x1) # (B)
print(xx0)


plt.figure(figsize =(5, 3.5))
ax = plt.subplot(1,1,1, projection = '3d') # (C)
ax.plot_surface(xx0, xx1, y, rstride=1, cstride=1, alpha=0.3, color='blue',edgecolo
ax.set_zticks((0, 0.2)) #(E)
ax.view_init(75, -95) # (F)
plt.show()
```

```
[[-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 ...
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]]
```



In [146]:

```
#리스트 3-(6)
print(x0)
print(x1)
```

```
[-2.         -1.91836735 -1.83673469 -1.75510204 -1.67346939 -1.591836
73
 -1.51020408 -1.42857143 -1.34693878 -1.26530612 -1.18367347 -1.102040
82
 -1.02040816 -0.93877551 -0.85714286 -0.7755102  -0.69387755 -0.612244
9
 -0.53061224 -0.44897959 -0.36734694 -0.28571429 -0.20408163 -0.122448
98
 -0.04081633  0.04081633  0.12244898  0.20408163  0.28571429  0.367346
94
  0.44897959  0.53061224  0.6122449   0.69387755  0.7755102   0.857142
86
  0.93877551  1.02040816  1.10204082  1.18367347  1.26530612  1.346938
78
  1.42857143  1.51020408  1.59183673  1.67346939  1.75510204  1.836734
69
  1.91836735  2.        ]
[-2.         -1.91836735 -1.83673469 -1.75510204 -1.67346939 -1.591836
73
 -1.51020408 -1.42857143 -1.34693878 -1.26530612 -1.18367347 -1.102040
82
 -1.02040816 -0.93877551 -0.85714286 -0.7755102  -0.69387755 -0.612244
9
 -0.53061224 -0.44897959 -0.36734694 -0.28571429 -0.20408163 -0.122448
98
 -0.04081633  0.04081633  0.12244898  0.20408163  0.28571429  0.367346
94
  0.44897959  0.53061224  0.6122449   0.69387755  0.7755102   0.857142
86
  0.93877551  1.02040816  1.10204082  1.18367347  1.26530612  1.346938
78
  1.42857143  1.51020408  1.59183673  1.67346939  1.75510204  1.836734
69
  1.91836735  2.        ]
```

In [147]:

```
print(xx0)
```

```
[[-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 ...
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]]
```

In [148]:

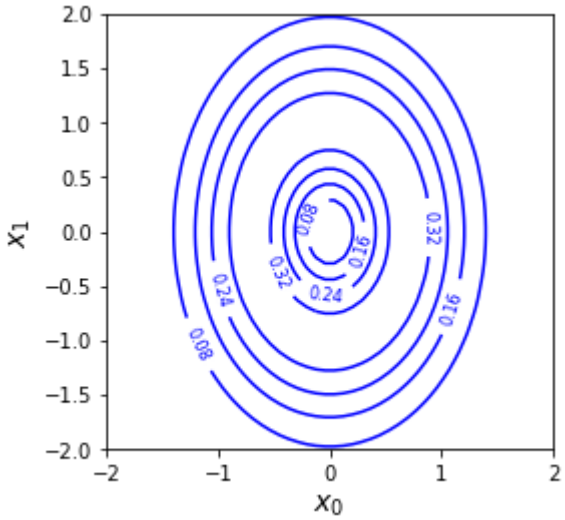
```
print(xx1)
```

```
[[-2.         -2.         -2.         ... -2.         -2.
  -2.        ]
 [-1.91836735 -1.91836735 -1.91836735 ... -1.91836735 -1.91836735
  -1.91836735]
 [-1.83673469 -1.83673469 -1.83673469 ... -1.83673469 -1.83673469
  -1.83673469]
 ...
 [ 1.83673469  1.83673469  1.83673469 ...  1.83673469  1.83673469
   1.83673469]
 [ 1.91836735  1.91836735  1.91836735 ...  1.91836735  1.91836735
   1.91836735]
 [ 2.          2.          2.         ...  2.          2.
   2.        ]]
```

3.2.4 등고선으로 표시: contour

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt
# 함수 f3를 정의
def f3(x0, x1):
    r = 2 * x0 **2 + x1 **2
    ans =  r*np.exp(-r)
    return ans
#리스트 3-(9)
xn = 50
x0 = np.linspace(-2, 2, xn)
x1 = np.linspace(-2, 2, xn)

y = np.zeros((len(x0),len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0] = f3(x0[i0], x1[i1])

xx0, xx1 = np.meshgrid(x0, x1)# (A)

plt.figure(1, figsize=(4, 4))
cont = plt.contour(xx0, xx1, y, 5, colors='blue') # (B)
cont.clabel(fmt='%3.2f', fontsize=8) # (C)
plt.xlabel('$x_0$', fontsize=14)
plt.ylabel('$x_1$', fontsize=14)
plt.show()
#정확도를 꼭 높여야만 등고선이 출력된다.
```

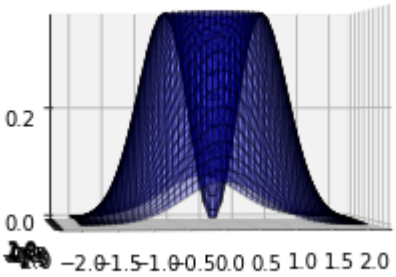


Note 옮긴이의 조언

In [14]:

```
#리스트 3-(5) ax.view_init()함수에 인수1과 인수2를 변경하여 다양한 차트를 생성해보기-3차원 그래프의
from mpl_toolkits.mplot3d import Axes3D # (A)
xx0, xx1 = np.meshgrid(x0, x1) # (B)
print(xx0)


plt.figure(figsize =(5, 3.5))
ax = plt.subplot(1,1,1, projection = '3d') # (C)
ax.plot_surface(xx0, xx1, y, rstride=1, cstride=1, alpha=0.3, color='blue',edgecolo
ax.set_zticks((0, 0.2)) #(E)
ax.view_init(0, -95) # (F)-기울이지않아 옆에서 본 그래프
plt.show()
```

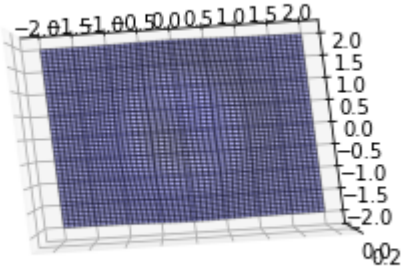
```
[[-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 ...
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]]
```



In [15]:

```
#리스트 3-(5) ax.view_init()함수에 인수1과 인수2를 변경하여 다양한 차트를 생성해보기-3차원 그래프의
from mpl_toolkits.mplot3d import Axes3D # (A)
xx0, xx1 = np.meshgrid(x0, x1) # (B)
print(xx0)


plt.figure(figsize =(5, 3.5))
ax = plt.subplot(1,1,1, projection = '3d') # (C)
ax.plot_surface(xx0, xx1, y, rstride=1, cstride=1, alpha=0.3, color='blue',edgecolo
ax.set_zticks((0, 0.2)) #(E)
ax.view_init(90, -95) # (F)-위에서 본 그래프
plt.show()
```

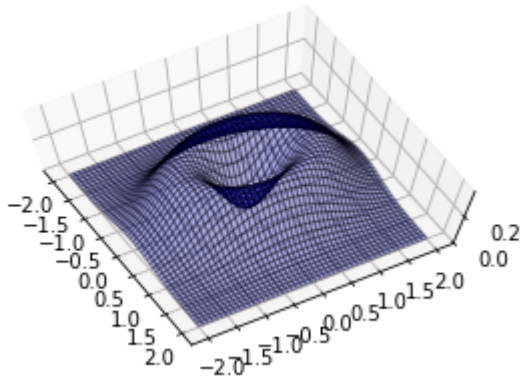
```
[[-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 ...
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]]
```



In [16]:

```
#리스트 3-(5) ax.view_init()함수에 인수1과 인수2를 변경하여 다양한 차트를 생성해보기-3차원 그래프의
from mpl_toolkits.mplot3d import Axes3D # (A)
xx0, xx1 = np.meshgrid(x0, x1) # (B)
print(xx0)


plt.figure(figsize =(5, 3.5))
ax = plt.subplot(1,1,1, projection = '3d') # (C)
ax.plot_surface(xx0, xx1, y, rstride=1, cstride=1, alpha=0.3, color='blue',edgecolo
ax.set_zticks((0, 0.2)) #(E)
ax.view_init(75, -30) # (F)
plt.show()
```

```
[[-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 ...
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]
 [-2.         -1.91836735 -1.83673469 ...  1.83673469  1.91836735
   2.        ]]
```



90pg까지의 소감문

초반 부분은 numpy의 array,arange같은 자료형들과 함수를 생성하는 방법 등 파이썬에서 자료를 관리하고 파일을 다루는 것에 대한 예시를 보여주었고 후반부에는 matplotlib의 함수들을 통해 실제로 그래프를 그리고 linspace나 저번 시간에 사용했던 figure를 함께해서 시각적으로 표현하는 방법에 대해서 배웠습니다. 상당히 많은 기능을 담고 있고 이를 능숙하게 다루기에는 많은 과정과 시간이 필요할 듯합니다. x,y값을 직접 배열에 담아 그래프를 통해 표현하는 내부기능을 전부 이해하진 못했지만 그래프를 코딩을 통해 직접 눈으로 시각화하는 과정은 즐거웠습니다. 이상입니다.

In [ ]:

In [ ]:

```

```

In [ ]:

```

```