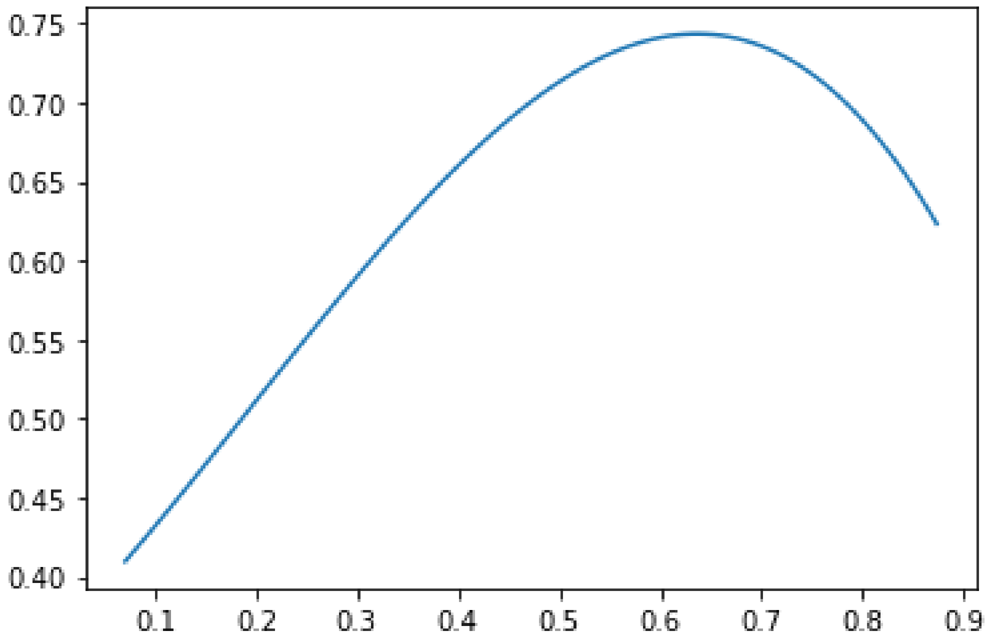
chap4-linear-algebra

June 16, 2022

[1]:
```python
### fitting to polynomials with NumPy

# import required libraries NumPy, polynomial and matplotlib
import numpy as np
import matplotlib.pyplot as plt

# generate two random vectors
v1 = np.random.rand(10)
v2 = np.random.rand(10)

# create a sequence of equally separated values
sequence = np.linspace(v1.min(), v1.max(), num=len(v1)*10)

# fit the data to polynomial fit data with 4 degrees of the polynomial
coefs = np.polyfit(v1, v2, 3)

# evaluate polynomial on given sequence
polynomial_sequence = np.polyval(coefs, sequence)

# plot the polynomial curve
plt.plot(sequence, polynomial_sequence)

# show plot
plt.show()
```



[2]:

```
# determinant

# import numpy
import numpy as np

# create matrix using NumPy
mat = np.mat([[2,4], [5,7]])
print("Matrix:\n", mat)

# calculate determinant
print("Determinant:", np.linalg.det(mat))
```

```
Matrix:
 [[2 4]
 [5 7]]
Determinant: -5.999999999999998
```

[3]:

```
# matrix rank

# import required libraries
import numpy as np
from numpy.linalg import matrix_rank

# create matrix
mat = np.array([[5,3,1],[5,3,1],[1,0,5]])
```

```
# compute rank of matrix
print("Matrix: \n", mat)
print("Rank:", matrix_rank(mat))
```

```
Matrix:
 [[5 3 1]
 [5 3 1]
 [1 0 5]]
Rank: 2
```

[4]:
```
# inverse matrix

# import numpy
import numpy as np

# create matrix using NumPy
mat = np.mat([[2,4], [5,7]])
print("Input Matrix:\n", mat)

# find matrix inverse
inverse = np.linalg.inv(mat)
print("Inverse:\n", inverse)
```

```
Input Matrix:
 [[2 4]
 [5 7]]
Inverse:
 [[-1.16666667  0.66666667]
 [ 0.83333333 -0.33333333]]
```

[5]:
```
# solving linear equations

# create matrix A and vector B using NumPy
A = np.mat([[1,1],[3,2]])
print("Matrix A:\n", A)

B = np.array([200, 450])
print("Vector B:", B)
```

```
Matrix A:
 [[1 1]
 [3 2]]
Vector B: [200 450]
```

[6]:
```
# solve linear equations
solution = np.linalg.solve(A, B)
print("Solution vector x:", solution)
```

```
Solution vector x: [ 50. 150.]
```

[7]:
```python
# check the solution
print("Result:", np.dot(A, solution))
```

```
Result: [[200. 450.]]
```

[9]:
```python
# decomposing a matrix using SVD

# import required libraries
import numpy as np
from scipy.linalg import svd

# create a matrix
mat = np.array([[5,3,1],[5,3,1],[1,0,5]])

# perform matrix decomposition using SVD
U, Sigma, V_transpose = svd(mat)

print("Left Singular Matrix: ", U)
print("Diagonal Matrix: ", Sigma)
print("Right Singular Matrix: ", V_transpose)
```

```
Left Singular Matrix:  [[-6.78452471e-01 -1.99254219e-01 -7.07106781e-01]
 [-6.78452471e-01 -1.99254219e-01  7.07106781e-01]
 [-2.81788019e-01  9.59476687e-01  1.05433742e-16]]
Diagonal Matrix:  [8.61123648e+00 4.67403534e+00 1.72212145e-16]
Right Singular Matrix:  [[-0.82059211 -0.47272129 -0.32119023]
 [-0.22102219 -0.25578012  0.94113002]
 [ 0.52704628 -0.84327404 -0.10540926]]
```

[2]:
```python
## eigenvectors and eigenvalues

# import numpy
import numpy as np

# create matrix using NumPy
mat = np.mat([[2,4], [5,7]])
print("Matrix:\n", mat)
```

```
Matrix:
 [[2 4]
 [5 7]]
```

[3]:
```python
# calculate the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(mat)
print("Eigenvalues: ", eigenvalues)
print("Eigenvectors: ", eigenvectors)
```

```
Eigenvalues:  [-0.62347538  9.62347538]
Eigenvectors:  [[-0.83619408 -0.46462222]
 [ 0.54843365 -0.885509   ]]
```

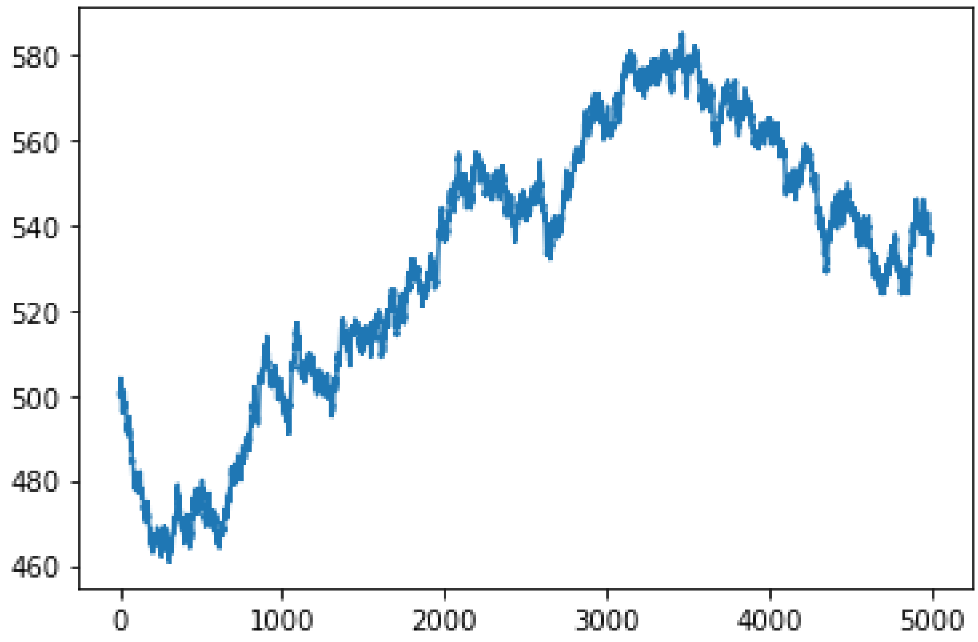
[4]:
```python
# compute eigenvalues
eigenvalues = np.linalg.eigvals(mat)
print("Eigenvalues: ", eigenvalues)
```

```
Eigenvalues:  [-0.62347538  9.62347538]
```

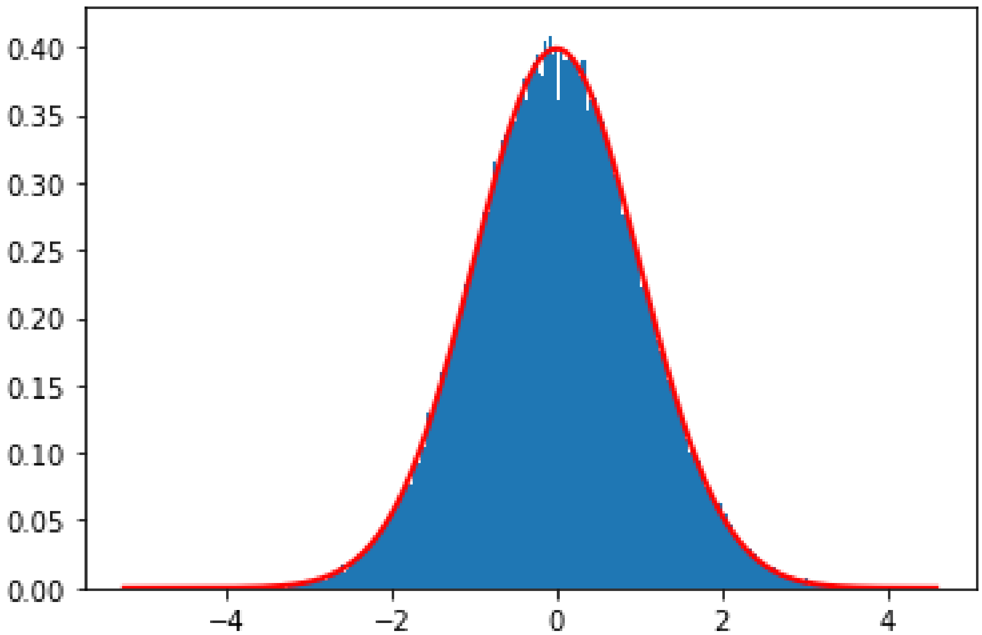
[6]:
```python
# generating random numbers
# import numpy
import numpy as np

# create an array with random values
random_mat = np.random.random((3,3))
print("Random Matrix: \n", random_mat)
```

```
Random Matrix:
 [[0.71291274 0.10722169 0.97383935]
 [0.88774134 0.86616774 0.81355749]
 [0.3693829  0.83366853 0.22663683]]
```

[7]:
```python
# binomal distribution
# import required libraries
import numpy as np
import matplotlib.pyplot as plt

# create an numpy vector of size 5000 with value 0
cash_balance = np.zeros(5000)

cash_balance[0] = 500

# generate random numbers using Binomial
samples = np.random.binomial(9,0.5,size=len(cash_balance))

# update the cash balance
for i in range(1, len(cash_balance)):
    if samples[i] < 5:
        cash_balance[i] = cash_balance[i - 1] - 1
    else:
        cash_balance[i] = cash_balance[i - 1] + 1

# plot the updated cash balance
plt.plot(np.arange(len(cash_balance)), cash_balance)
plt.show()
```

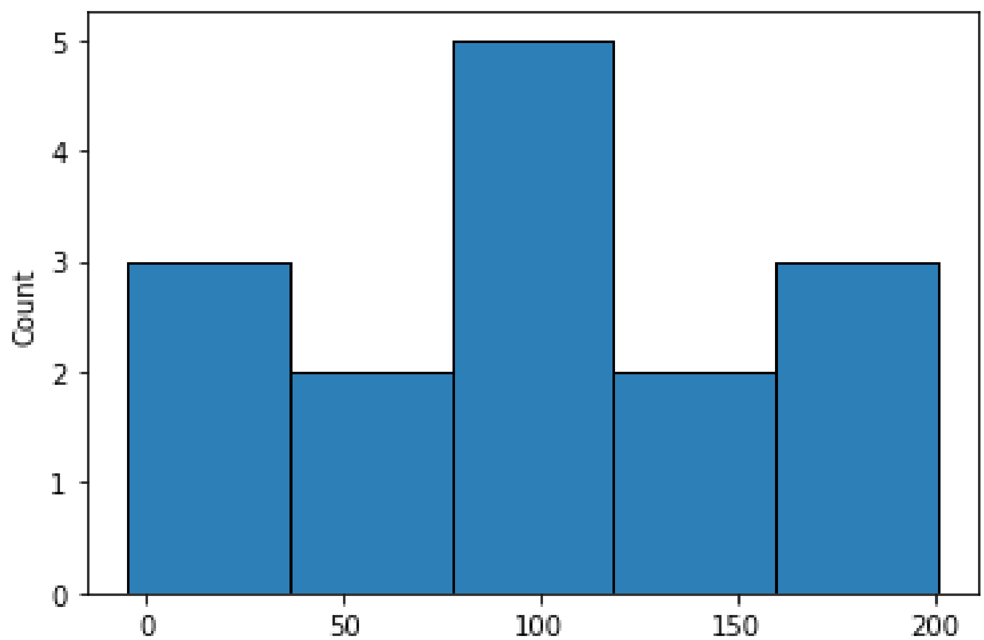


[8]:
```
# normal distribution

# import required library
import numpy as np
import matplotlib.pyplot as plt

sample_size = 225000

# generate random values sample using normal distribution
sample = np.random.normal(size=sample_size)

# create histogram
n, bins, patch_list = plt.hist(sample, int(np.sqrt(sample_size)), density=True)

# set parameters
mu, sigma=0,1

x = bins
y = 1/(sigma * np.sqrt(2 * np.pi)) * np.exp(-(bins - mu)**2 / (2* sigma**2))

# plot line plot (or bell curve)
plt.plot(x,y,color='red', lw=2)
plt.show()
```

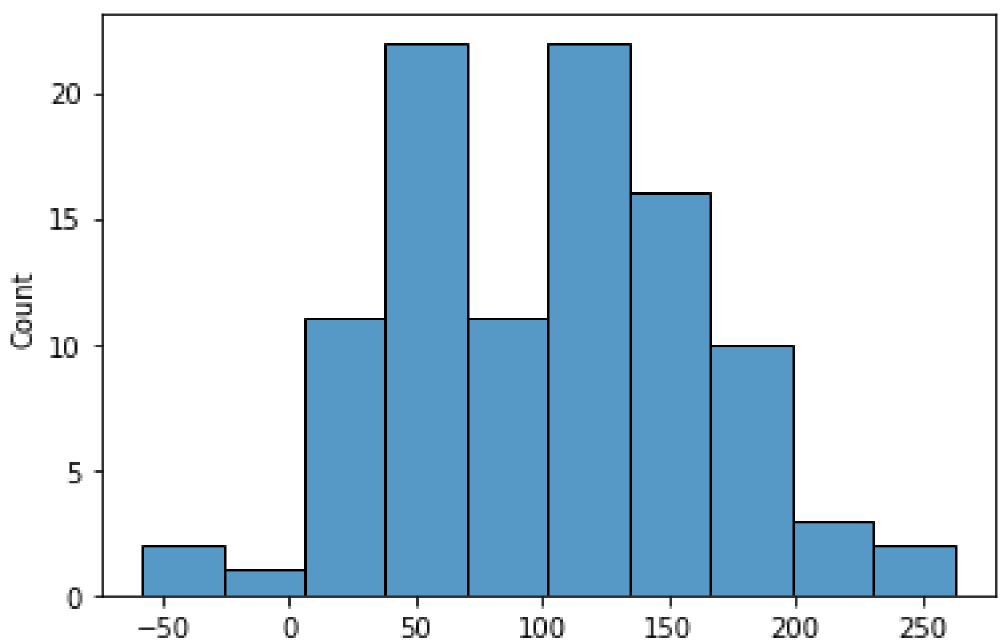


[10]:
```
# testing normality od data using SciPy
import numpy as np

# create small, medium, and large samples for normality test
small_sample = np.random.normal(loc=100, scale=60, size=15)
medium_sample = np.random.normal(loc=100, scale=60, size=100)
large_sample = np.random.normal(loc=100, scale=60, size=1000)
```

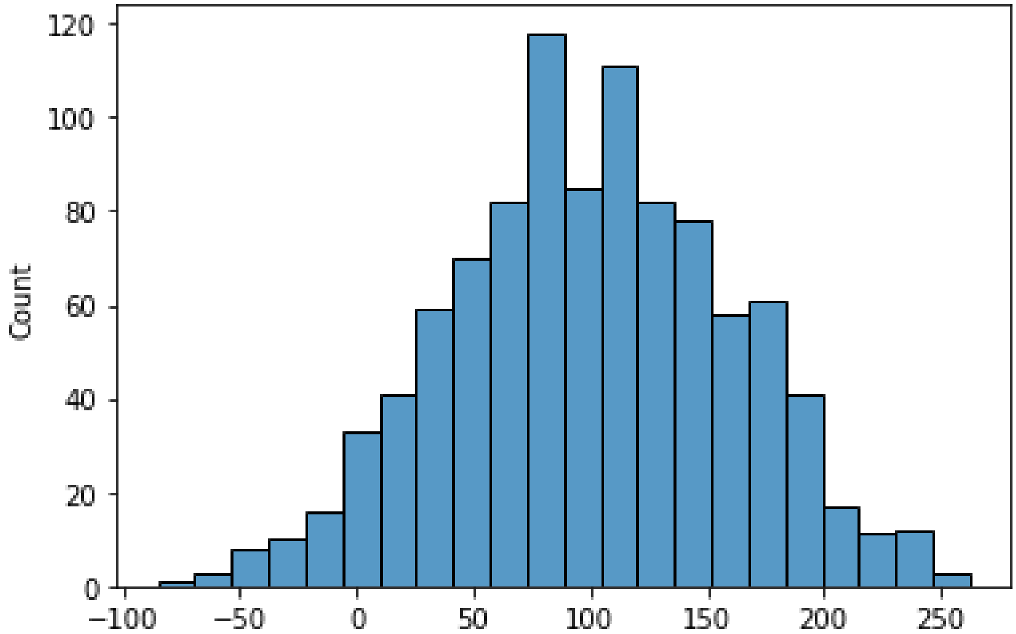
[12]:
```
# histogram for small
import seaborn as sns
import matplotlib.pyplot as plt

# create distribution plot
sns.histplot(small_sample)

sns.histplot(small_sample)

plt.show()
```



[14]:
```
# histogram for medium
sns.histplot(medium_sample)
plt.show()
```



[15]:
```
# histogram for large
sns.histplot(large_sample)
plt.show()
```

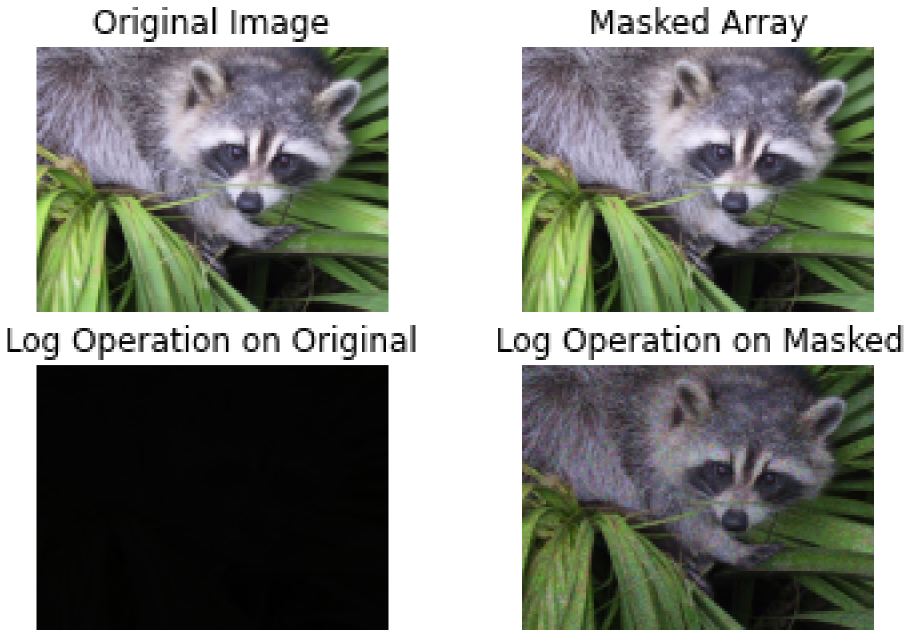


[17]:
```
# shapiro-wilk test

# import shapiro func
from scipy.stats import shapiro

# apply Shapiro-Wilk Test
print("Shapiro-Wilk Test for Small Sample: ", shapiro(small_sample))
print("Shapiro-Wilk Test for Medium Sample: ", shapiro(medium_sample))
print("Shapiro-Wilkt Test for Large Sample: ", shapiro(large_sample))
```

```
Shapiro-Wilk Test for Small Sample:  ShapiroResult(statistic=0.9622419476509094,
pvalue=0.7312947511672974)
Shapiro-Wilk Test for Medium Sample:
ShapiroResult(statistic=0.9901710152626038, pvalue=0.6784243583679199)
Shapiro-Wilkt Test for Large Sample:
ShapiroResult(statistic=0.9978142380714417, pvalue=0.21330703794956207)
```

[29]:
```
# creating a masked array using the numpy.ma subpackage

from scipy.misc import face

face_image = face()
```

```
mask_random_array = np.random.randint(0, 3, size=face_image.shape)

fig, ax = plt.subplots(nrows = 2, ncols = 2)

# display the orginal Image
plt.subplot(2,2,1)
plt.imshow(face_image)
plt.title("Original Image")
plt.axis('off')

# display masked array
masked_array = np.ma.array(face_image, mask=mask_random_array)
plt.subplot(2,2,2)
plt.title("Masked Array")
plt.imshow(masked_array)
plt.axis('off')

# log operation on orginal image
plt.subplot(2,2,3)
plt.title("Log Operation on Original")
plt.imshow(np.ma.log(face_image).astype('uint8'))
plt.axis('off')

# log operation on masked array
plt.subplot(2,2,4)
plt.title("Log Operation on Masked")
plt.imshow(np.ma.log(masked_array).astype('uint8'))
plt.axis('off')

# display the subplots
plt.show()
```


Original Image
Masked Array
Log Operation on Original
Log Operation on Masked

[ ]: