1. Compute the eigenvalues and right eigenvectors of a given square array.

```
import math
import numpy as np
def mat_vec_multiple(matrix, vector):
  result = [] # initialize an empty list to store the elements of the resulting vector

  for row in matrix:
    # dot_product = 0 # initialize a variable to hold the dot product for the current row.
    dot_product = sum(row[i] * vector[i] for i in range(len(vector)))
    result.append(dot_product)
  return result

# normalize vector which represents the direction of the original vector with unit magnitude
def vec_norm(vector):
  magnitude = sum(x ** 2 for x in vector) ** 0.5 # summing the square of the each elements and taki
  normalized_v = [x / magnitude for x in vector]
  return normalized_v

def power_iterate(matrix, num_loop=1000, tol=1e6):
  n = len(matrix)
  v = [1] *n

  for _ in range(num_loop):
    v_new = mat_vec_multiple(matrix,v)
    v_new = vec_norm(v_new)  #normalize the new vector

    # this is just to checkthe convergence
    diff = sum((v[i] - v_new[i]) **2 for i in range(n))
    if diff > tol:
      break
    v = v_new


  eigenvalue = sum(v[i] * matrix[i][j] * v[j] for i in range(n) for j in range(n))
  return eigenvalue, v

if __name__ == '__main__':
  matrix = [[3, -2],
            [1, 0]]
  eigenvalue, eigenvector = power_iterate(matrix)
  for row in matrix:
    print(row)
    print("\nEigenvalue: " )
    print(eigenvalue)
    print("\nEigenvector: " )
    print(eigenvector)
```

```
[3, -2]

Eigenvalue:
2.000000000000001

Eigenvector:
[0.894427190999916, 0.44721359549995787]
[1, 0]

Eigenvalue:
```

```
2.000000000000001

Eigenvector:
[0.894427190999916, 0.44721359549995787]
```

2. Compute the factor of a given array by Singular Value Decomposition

```
import numpy as np

#define an array
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]], dtype=np.float32)
# U, s, Vt where U represents left singular vector
# s represents the singular values of 'A'
# Vt the transpose of the right vectors.

U, s, Vt = np.linalg.svd(A)

# reconstruct the original matrix 'A' using the SVD components
# Singular values are 1D array so we convert it into diagonal matrix
# using np.daig
Sigma = np.diag(s)
reconstruct_A = np.dot(U, np.dot(Sigma, Vt))

print("Provided Matrix: ")
print(A)

print("\nSingular Value Decomposition: ")
print("U =\n", U)
print("Sigma =\n", Sigma)
print("\nReconstructed Matrix: ")
print(reconstruct_A)
```

```
Provided Matrix:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]

Singular Value Decomposition:
U =
 [[-0.21483724  0.8872307   0.4082483 ]
 [-0.5205874   0.24964395 -0.8164966 ]
 [-0.8263375  -0.3879428   0.4082483 ]]
Sigma =
 [[1.6848103e+01 0.0000000e+00 0.0000000e+00]
 [0.0000000e+00 1.0683695e+00 0.0000000e+00]
 [0.0000000e+00 0.0000000e+00 4.4184248e-16]]

Reconstructed Matrix:
[[1.        1.9999999 3.       ]
 [4.        5.        6.       ]
 [7.        7.9999995 8.999999 ]]
```

3. compute the determinant of an array

```
# define a matrix
matrix = [[1, 2],
          [3, 4]]
# compute the determinant using formula ad -bc
determinant = (matrix[0][0] * matrix[1][1]) - (matrix[1][0] * matrix[0][1])
```

```
# print original matrix
print("Original Array: ")
for row in matrix:
  print(row)

print("Determinant of the Array: ")
print(determinant)
```

```
Original Array:
[1, 2]
[3, 4]
Determinant of the Array:
-2
```