Gauss–Seidel method:

# Iterative Methods GAUSS SEIDEL Method  
*'''  
 EXAMPLE: Solve the following system of linear equations using Gauss–Seidel method using a  
 predefined threshold = 0.01. Remember to check if the converge condition is satisfied or not.  
 8x1 +3x2 −3x3 = 14,  
 −2x1 −8x2 +5x3 = 5,  
 3x1 + 5x2 +10x3 = −8.  
'''*import numpy as np  
  
##### Let us first check if the coefficient matrix is diagonally dominant or not.  
a = [[8, 3, -3], [-2, -8, 5], [3, 5, 10]]  
# Find diagonal coefficients  
diag = np.diag(np.abs(a))  
  
# Find row sum without diagonal  
off\_diag = np.sum(np.abs(a), axis=1) - diag  
if np.all(diag > off\_diag):  
 print("matrix is diagonally dominant")  
else:  
 print("NOT diagonally dominant")  
########## first set intial values for (x1,x2,x3)  
x1 = 0  
x2 = 0  
x3 = 0  
####### set threshold value equal .01  
epsilon = .01  
##### set converged =false and set value of x\_old equal to intial values for (x1,x2,x3)  
converged = False  
x\_old = np.array([x1, x2, x3])  
print("iteration result")  
print("k, x1, x2, x3")  
  
'''  
Since it is guaranteed to converge, we can use Gauss–Seidel method to solve the system.  
'''  
####### Repeat 50 times to calculate a new value of (x1, x2, x3) each time  
for k in range(1, 50):  
 x1 = (14 - 3 \* x2 + 3 \* x3) / 8  
 x2 = (5 + 2 \* x1 - 5 \* x3) / (-8)  
 x3 = (-8 - 3 \* x1 - 5 \* x2) / (-5)  
 x = np.array([x1, x2, x3])  
 ''' if difference between new values for (x1,x2,x3) and last iteration values for (x1,x2,x3) less than threshold value  
 set converged = true and exit code  
  
 '''  
 # check if it is smaller than threshold  
 dx = np.sqrt(np.dot(x - x\_old, x - x\_old))  
 print("%d, %.4f, %.4f, %.4f" % (k, x1, x2, x3))  
 if dx < epsilon:  
 converged = True  
 print("converged")  
 break  
  
 '''if the difference between new values for (x1,x2,x3) and last iteration values for (x1,x2,x3) more than threshold value  
 set intial values for (x1,x2,x3) equal to last iteration values for (x1,x2,x3) and continnue to execute next iteration  
 '''  
 # assign the latest x value to the old value  
 x\_old = x  
  
####### if the loop ends without converged = true print("Not converged, increase the number of iterations")  
if not converged:  
 print("Not converged, increase the number of iterations")

Use numpy.linalg.solve to solve equation:

# solving system of linear equation

#####1

'''

TRY IT! Use numpy.linalg.solve to solve the following equations:

4x1 +3x2 −5x3 = 2,

−2x1 −4x2 +5x3 = 5,

8x1 +8x2 = −3.

'''

import numpy as np

a = np.array([[4, 3, -5],[-2,-4,5],[8,8,0]])

y = np.array([2,5,3])

x = np.linalg.solve(a,y)

print(x)

solving system of linear equation:

import numpy as np  
from scipy.linalg import lu  
A = np.array([[4, 3, -5], [-2, -4, 5], [8, 8, 0]])  
P, L, U = lu(A)  
print("p", P)  
print("l", L)  
print("u:", U)  
print("lu:", np.dot(L, U))

plot the vector:

import numpy as np  
import matplotlib.pyplot as plt  
  
plt.style.use("seaborn-poster")  
  
  
# %matplotlib inline  
def plot\_vect(x, b, xlim, ylim):  
 plt.figure(figsize=(10, 6))  
 plt.quiver(0, 0, x[0], x[1], color="k", angles="xy", scale\_units="xy", scale=1, label="original vector")  
 plt.quiver(0, 0, b[0], x[1], color="g", angles="xy", scale\_units="xy", scale=1, label="transformed vector")  
 plt.xlim(xlim)  
 plt.ylim(ylim)  
 plt.xlabel("X")  
 plt.ylabel("Y")  
 plt.legend()  
 plt.show()  
  
  
a = np.array([[2, 0], [0, 1]])  
x = np.array([[1], [1]])  
b = np.dot(a, x)  
plot\_vect(x, b, (0, 3), (0, 2))  
  
x = np.array([[1], [0]])  
b = np.dot(a, x)  
plot\_vect(x, b, (0, 3), (-0.5, 0.5))

power method:

######In[1]  
import numpy as np  
#####In [2]:   
def normalize(x):  
 fac = abs(x).max()  
 x\_n = x / x.max()  
 return fac, x\_n  
######In [3]:  
x = np.array([1, 1])  
a = np.array([[0, 2],[2, 3]])  
for i in range(8):  
 x = np.dot(a, x)  
 lambda\_1, x = normalize(x)  
print("Eigenvalue:", lambda\_1)  
print("Eigenvector:", x)

The smallest eigenvalue:

from numpy.linalg import inv  
import numpy as np  
a = np.array([[0, 2], [2, 3]])  
x =np.array([1, 1])  
a\_inv = inv(a)  
for i in range(8):  
 x = np.dot(a\_inv, x)  
 lambda\_l, x = normalize(x)  
print("eigenvalue:", lambda\_l)  
print("eigenvector: ", x)

QR method:

import numpy as np  
from numpy.linalg import qr  
##########In [2]:  
a = np.array([[0, 2], [2, 3]])  
q, r = qr(a)  
print("Q:", q)  
print("R:", r)  
b = np.dot(q, r)  
print("QR:", b)  
#########In [3]:  
a = np.array([[0, 2],[2, 3]])  
p = [1, 5, 10, 20]  
for i in range(20):  
 q, r = qr(a)  
 a = np.dot(r, q)  
 if i+1 in p:  
 print(f"Iteration {i+1}:")  
 print(a)

Calculate the eigenvalues:

import numpy as np  
from numpy.linalg import eig  
  
a = np.array([[0, 2], [2, 3]])  
w, v = eig(a)  
print("E=value", w)  
print("E-vector", v)

Do a least squares regression:

import numpy as np  
import matplotlib.pyplot as plt  
plt.style.use("seaborn-poster")  
x = np.linspace(0, 1, 101)  
y = 1 + x + x \* np.random.random(len(x))  
a = np.vstack([x, np.ones(len(x))]).T  
y = y[:, np.newaxis]  
alpha = np.dot((np.dot(np.linalg.inv(np.dot(a.T,a)),a.T)),y)  
print(alpha)  
plt.figure(figsize=(10, 8))  
plt.plot(x, y, "b")  
plt.plot(x, alpha[0]\*x + alpha[1], "r")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.show()

Do a least squares regression USING NUMPY.LINALG.LSTSQ:

import numpy as np  
import matplotlib.pyplot as plt  
plt.style.use("seaborn-poster")  
x = np.linspace(0, 1, 101)  
y = 1 + x + x \* np.random.random(len(x))  
a = np.vstack([x, np.ones(len(x))]).T  
y = y[:, np.newaxis]  
alpha = np.linalg.lstsq(a, y, rcond=None)[0]  
print(alpha)  
plt.figure(figsize=(10, 8))  
plt.plot(x, y, "b")  
plt.plot(x, alpha[0]\*x + alpha[1], "r")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.show()

OPTIMIZE.CURVE\_FIT FROM SCIPY:

import numpy as np  
from scipy import optimize  
x = np.linspace(0, 1, 101)  
y = 1 + x + x \* np.random.random(len(x))  
  
def func(c, a, b):  
 z = a\*c + b  
 return z  
  
alpha = optimize.curve\_fit(func, xdata=x, ydata=y)[0]  
print(alpha)

plot the real function:

import numpy as np  
import matplotlib.pyplot as plt  
  
plt.style.use("seaborn-poster")  
x = np.linspace(0, 10, 101)  
y = 0.1 \* np.exp(0.3\*x) + 0.1 \* np.random.random(len(x))  
plt.figure(figsize=(10, 8))  
plt.plot(x, y, "b")  
plt.xlabel("x")  
plt.ylabel("y")  
plt.show()

LOG TRICKS FOR EXPONENTIAL FUNCTIONS:

import numpy as np  
from scipy import optimize  
import matplotlib.pyplot as plt  
plt.style.use("seaborn-poster")  
#In [2]: # let’s generate x and y, and add some noise into y  
x = np.linspace(0, 10, 101)  
y = 0.1\*np.exp(0.3\*x) + 0.1\*np.random.random(len(x))  
#In [3]: # Let’s have a look of the data  
plt.figure(figsize = (10,8))  
plt.plot(x, y, "b.")  
plt.xlabel("x")  
plt.ylabel("y")  
plt.show()  
##Once the log trick has been applied, we can fit the data.  
#In [4]:   
A = np.vstack([x, np.ones(len(x))]).T  
beta, log\_alpha = np.linalg.lstsq(A, np.log(y), rcond = None)[0]  
alpha = np.exp(log\_alpha)  
print(f"alpha={alpha}, beta={beta}")  
  
#In [5]: # Let’s have a look of the data  
plt.figure(figsize = (10,8))  
plt.plot(x, y, "b.")  
plt.plot(x, alpha\*np.exp(beta\*x), "r")  
plt.xlabel("x")  
plt.ylabel("y")  
plt.show()

POLYNOMIAL REGRESSION:

import numpy as np  
from scipy import optimize  
import matplotlib.pyplot as plt  
plt.style.use("seaborn-poster")  
x\_d = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])  
y\_d=np.array([0,0.8,0.9,0.1,-0.6,-0.8,-1,-0.9,-0.4])  
plt.figure(figsize = (12, 8))  
for i in range(1, 7):  
 # get the polynomial coefficients  
 y\_est = np.polyfit(x\_d, y\_d, i)  
 plt.subplot(2,3,i)  
 plt.plot(x\_d, y\_d, "o")  
 # evaluate the values for a polynomial  
 plt.plot(x\_d, np.polyval(y\_est, x\_d))  
 plt.title(f"Polynomial order {i}")  
plt.tight\_layout()  
plt.show()

USING OPTIMIZE.CURVE\_FIT FROM SCIPY:

import numpy as np

from scipy import optimize

import matplotlib.pyplot as plt

plt.style.use("seaborn-poster")

#In [2]: # let’s generate x and y, and add some noise into y

x = np.linspace(0, 10, 101)

y = 0.1\*np.exp(0.3\*x) + 0.1\*np.random.random(len(x))

##In [7]: # let’s define the function form

def func(x, a, b):

    y = a\*np.exp(b\*x)

    return y

alpha, beta = optimize.curve\_fit(func, xdata = x, ydata = y)[0]

print(f"alpha={alpha}, beta={beta}")

#In [8]: # Let’s have a look of the data

plt.figure(figsize = (10,8))

plt.plot(x, y, "b.")

plt.plot(x, alpha\*np.exp(beta\*x), "r")

plt.xlabel("x")

plt.ylabel("y")

plt.show()