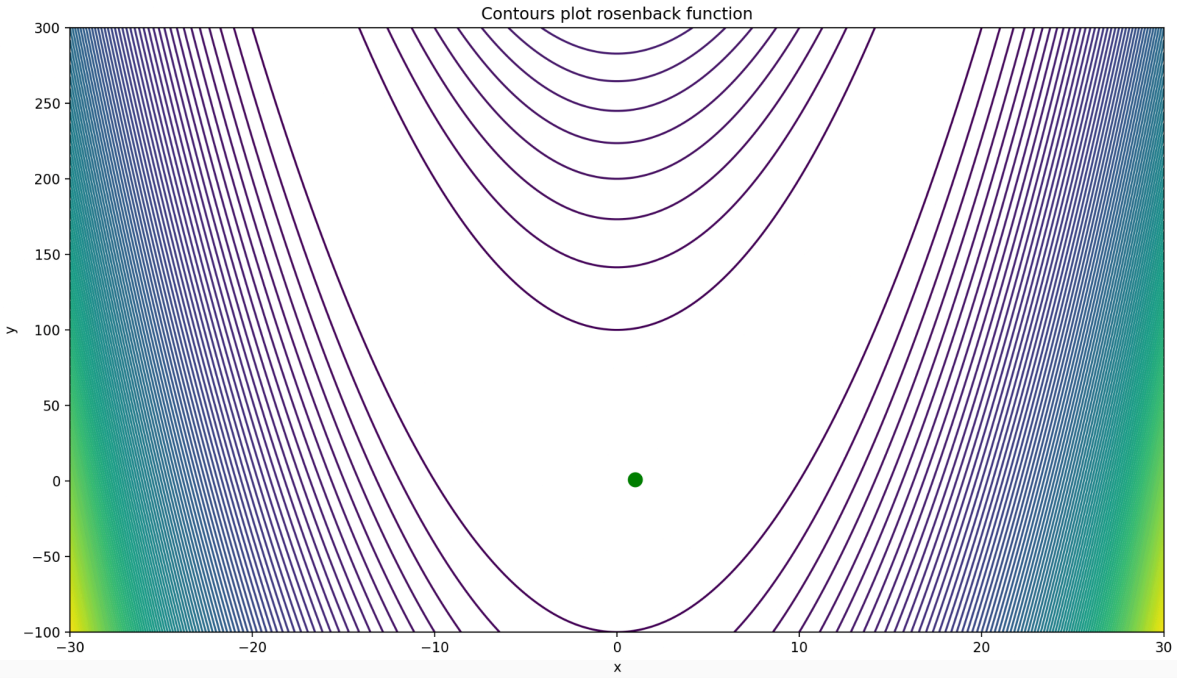
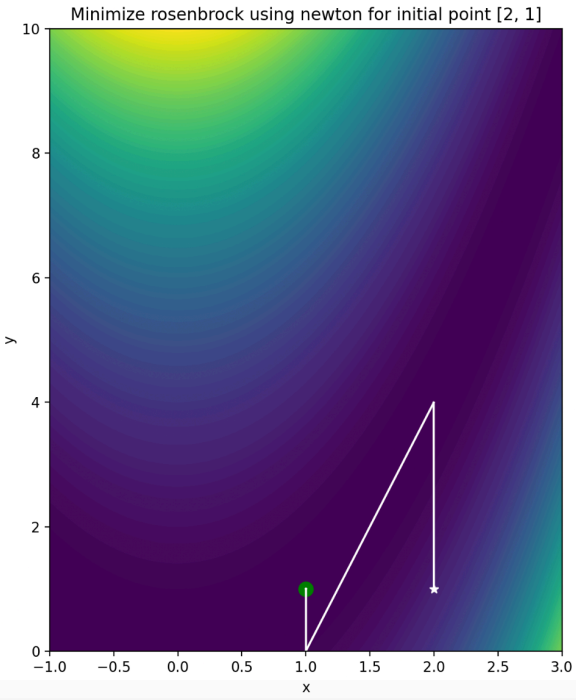


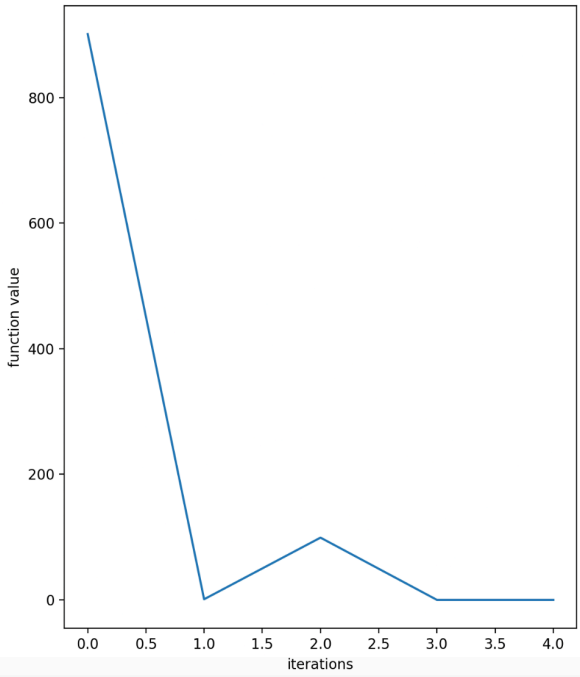

```
import matplotlib.pylab as plt
from matplotlib.pyplot import figure
import numpy as np

def rosenbrock(x, y):
   return 100 * ((y - (x ** 2)) ** 2) + ((1-x) ** 2)

def gradient(x, y):
   return np.array([-400 * x * (y - (x ** 2)) - 2 * (1-x), 200 * (y - (x ** 2))])

def hessian(x, y):
   return np.array([[(-400 * y) + (1200 * (x ** 2)) + 2, -400*x], [-400 * x, 200]])

def plotc(fun, xmin, xmax, ymin, ymax, contour=100, colour=True):
   x = np.linspace(xmin, xmax, 300)
   y = np.linspace(ymin, ymax, 300)
   X, Y = np.meshgrid(x, y)
   Z = fun(X, Y)
   if colour:
       plt.contourf(X, Y, Z, contour)
   else:
       plt.contour(X, Y, Z, contour)
   plt.scatter(1, 1, marker='o', s=100, color='g')
```

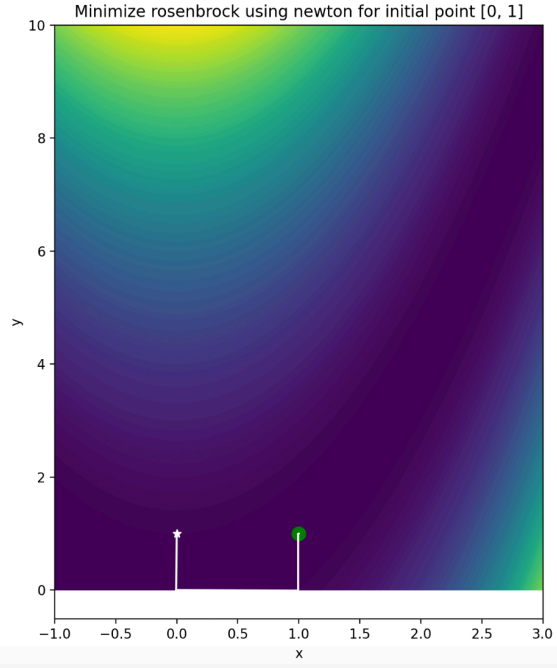
```
figure(figsize=(15, 8))
plotc(rosenbrock, -30,30, -100, 300, colour=False)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Contours plot rosenback function")
# plt.show()
def newton(fun, gradient, hessian, init, tol=1e-5, max_iter=10000):
   a = init
   count = 0
   values = [a]
   f_prev = fun(a[0], a[1])
   f_values = [f_prev]
   while count < max_iter:
       # computer hessian and gradient
       print("current values ", a[0], a[1], f_prev)
       g = gradient(a[0], a[1])
       h = hessian(a[0], a[1])
       a = a - np.linalg.solve(h, g) # Let numpy handle the complex calculations
       current = fun(a[0], a[1])
       if np.isnan(current):
           break
       values.append(a)
       f_values.append(current)
       if abs(current - f_prev) < tol:
           # under tolerance hence end the algorithm
           break
       f_prev = current
       count = count + 1
   return np.array(values), np.array(f_values)
for elem in [[-1, 1], [0, 1], [2, 1]]:
   x_values, y_values = newton(rosenbrock, gradient, hessian, init=elem)
   figure(figsize=(15, 8))
   plt.subplot(1,2,1)
   plotc(rosenbrock, -1,3,0,10)
   plt.xlabel("x")
   plt.ylabel("y")
   title = "Minimize rosenbrock using newton for initial point " + str(elem)
   plt.title(title)
   plt.scatter(x_values[0,0],x_values[0,1],marker="*",color="w")
   for i in range(1,len(x_values)):
           plt.plot((x_values[i-1,0],x_values[i,0]), (x_values[i-1,1],x_values[i,1]) ,
"w")
```

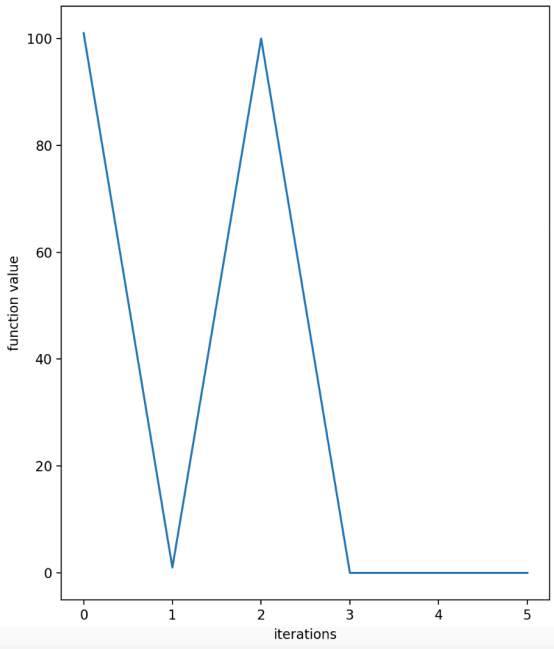
```
    plt.subplot(1,2,2)
    plt.plot(y_values)
    plt.xlabel("iterations")
    plt.ylabel("function value")
    print("DOne", x_values[-1])
plt.show()
```

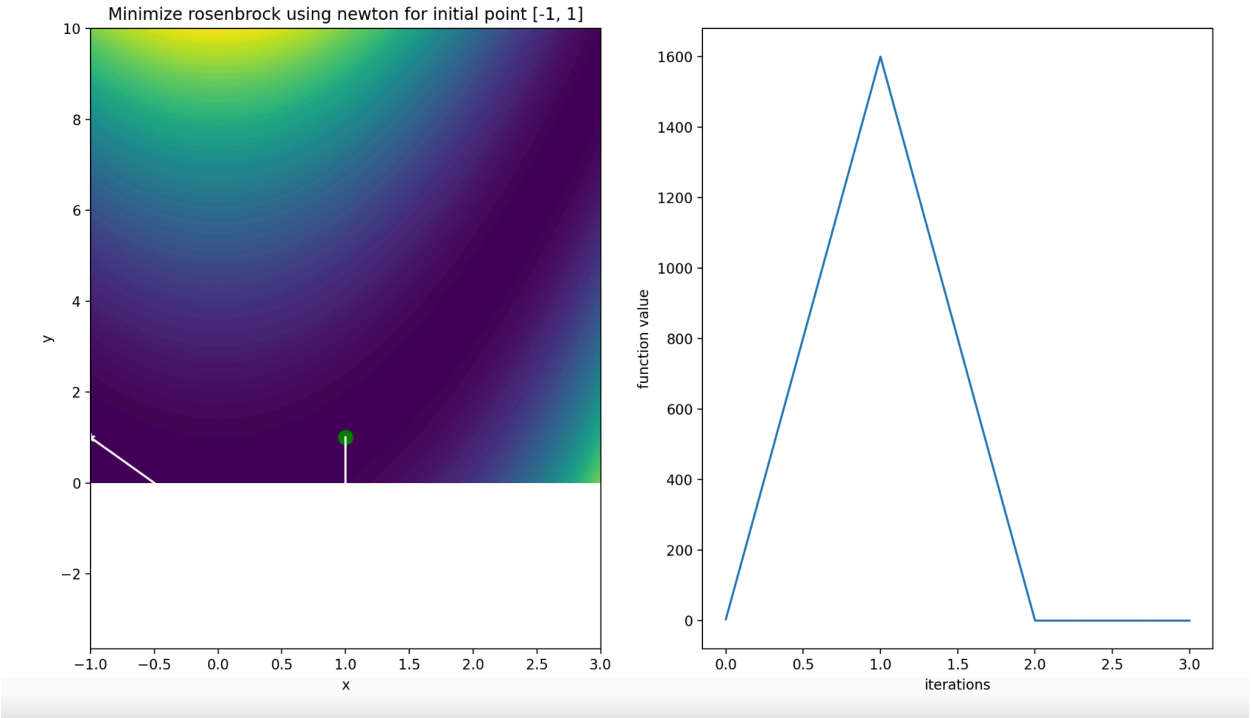
NEWTON’S METHOD PLOTS


Minimize rosenbrock using newton for initial point [2, 1]
x
y


function value
iterations


Minimize rosenbrock using newton for initial point [0, 1]
x
y


function value
iterations

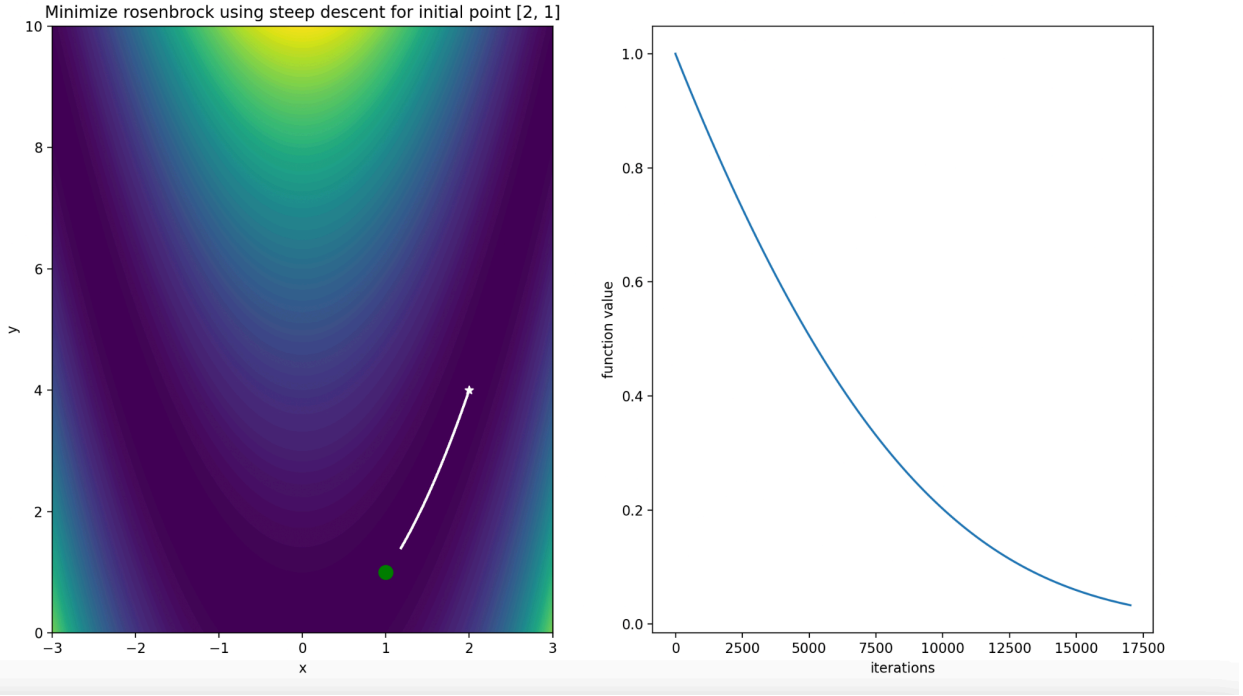


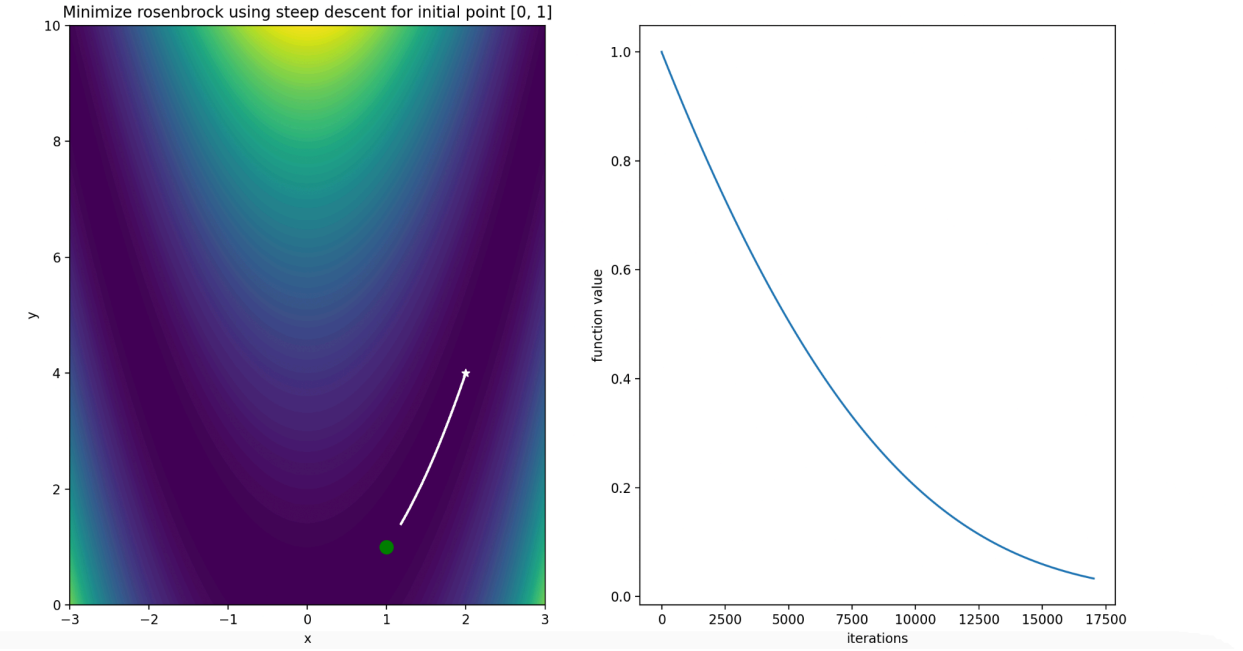
STEEP DESCENT CODE AND PLOT (have used different step size and max iterations for this)

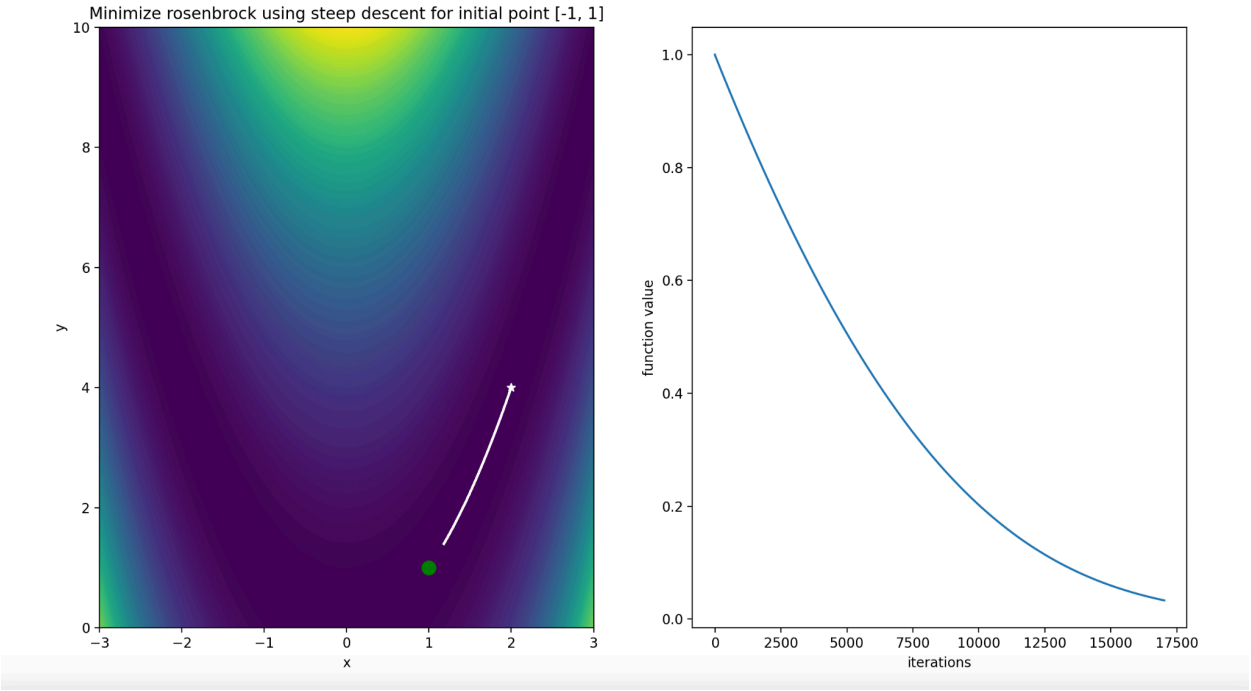
```
import matplotlib.pylab as plt
from matplotlib.pyplot import figure
import numpy as np

def rosenbrock(x, y):
   return 100 * ((y - (x ** 2)) ** 2) + ((1-x) ** 2)

def gradient(x, y):
   return np.array([-400 * x * (y - (x ** 2)) - 2 * (1-x), 200 * (y - (x ** 2))])

def hessian(x, y):
   return np.array([[(-400 * y) + (1200 * (x ** 2)) + 2, -400*x], [-400 * x, 200]])

def plotc(fun, xmin, xmax, ymin, ymax, contour=100, colour=True):
   x = np.linspace(xmin, xmax, 300)
   y = np.linspace(ymin, ymax, 300)
   X, Y = np.meshgrid(x, y)
   Z = fun(X, Y)
   if colour:
       plt.contourf(X, Y, Z, contour)
   else:
```

```
        plt.contour(X, Y, Z, contour)
    plt.scatter(1, 1, marker='o', s=100, color='g')

figure(figsize=(15, 8))
plotc(rosenbrock, -30,30, -100, 300, colour=False)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Contours plot rosenback function")
# plt.show()
def steep(fun, gradient, hessian, init, tol=1e-5, maxiter=30000, steplength=0.0005):
    a = init
    count=0
    values = [a]
    f_prev = fun(a[0],a[1])
    f_values = [f_prev]
    while count < maxiter:
        print("current values ", a[0], a[1], f_prev)
        a = a - steplength*gradient(a[0],a[1])
        cur = fun(a[0], a[1])
        if np.isnan(cur):
            break
        values.append(a)
        f_values.append(cur)
        if abs(cur-f_prev)<tol:
            break
        f_prev = cur
        count += 1
    return np.array(values), np.array(f_values)
for elem in [[-1, 1], [0, 1], [2, 1]]:
    x_values, y_values = steep(rosenbrock, gradient, hessian, init=[2,4])
    figure(figsize=(15, 8))
    plt.subplot(1,2,1)
    plotc(rosenbrock, -3,3,0,10)
    plt.xlabel("x")
    plt.ylabel("y")
    title = "Minimize rosenbrock using steep descent for initial point " + str(elem)
    plt.title(title)
    plt.scatter(x_values[0,0],x_values[0,1],marker="*",color="w")
    for i in range(1,len(x_values)):
            plt.plot((x_values[i-1,0],x_values[i,0]), (x_values[i-1,1],x_values[i,1]) ,
"w")
    plt.subplot(1,2,2)
```

```
    plt.plot(y_values)
    plt.xlabel("iterations")
    plt.ylabel("function value")
    print("DOne")
plt.show()
```


Minimize rosenbrock using steep descent for initial point [2, 1]
x
y
function value
iterations


Minimize rosenbrock using steep descent for initial point [0, 1]
x
y
function value
iterations

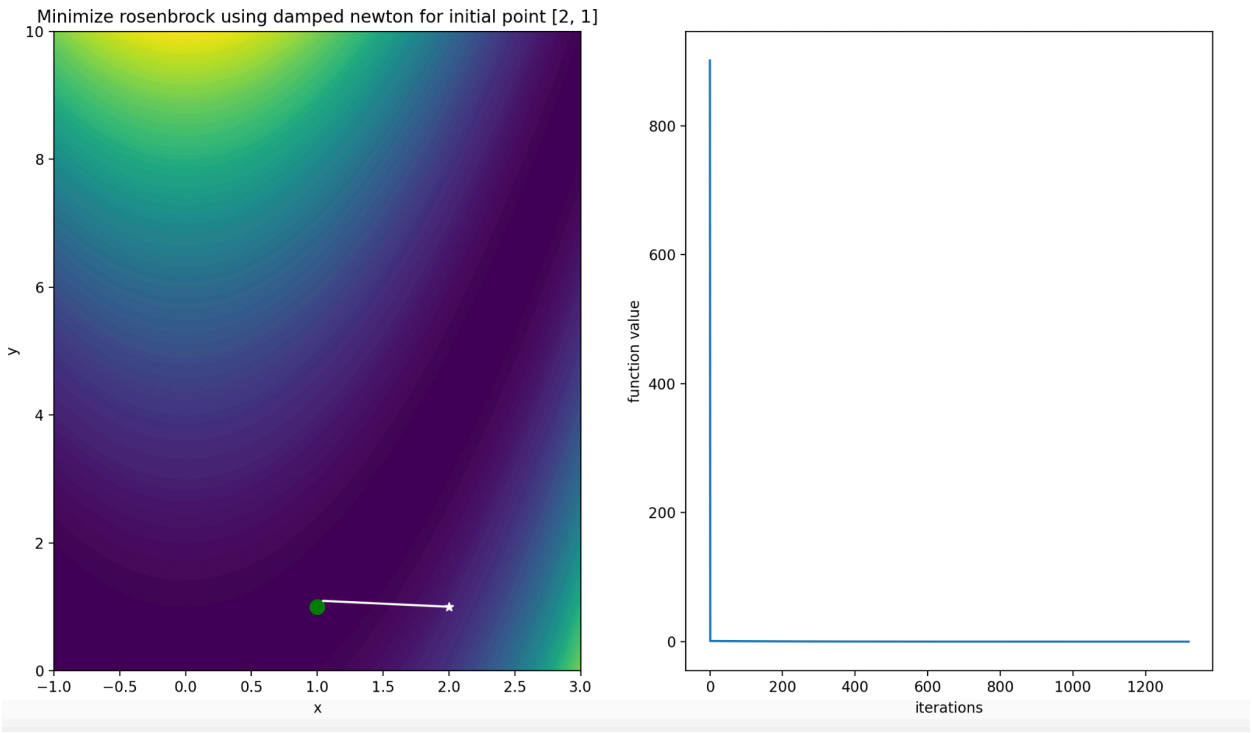


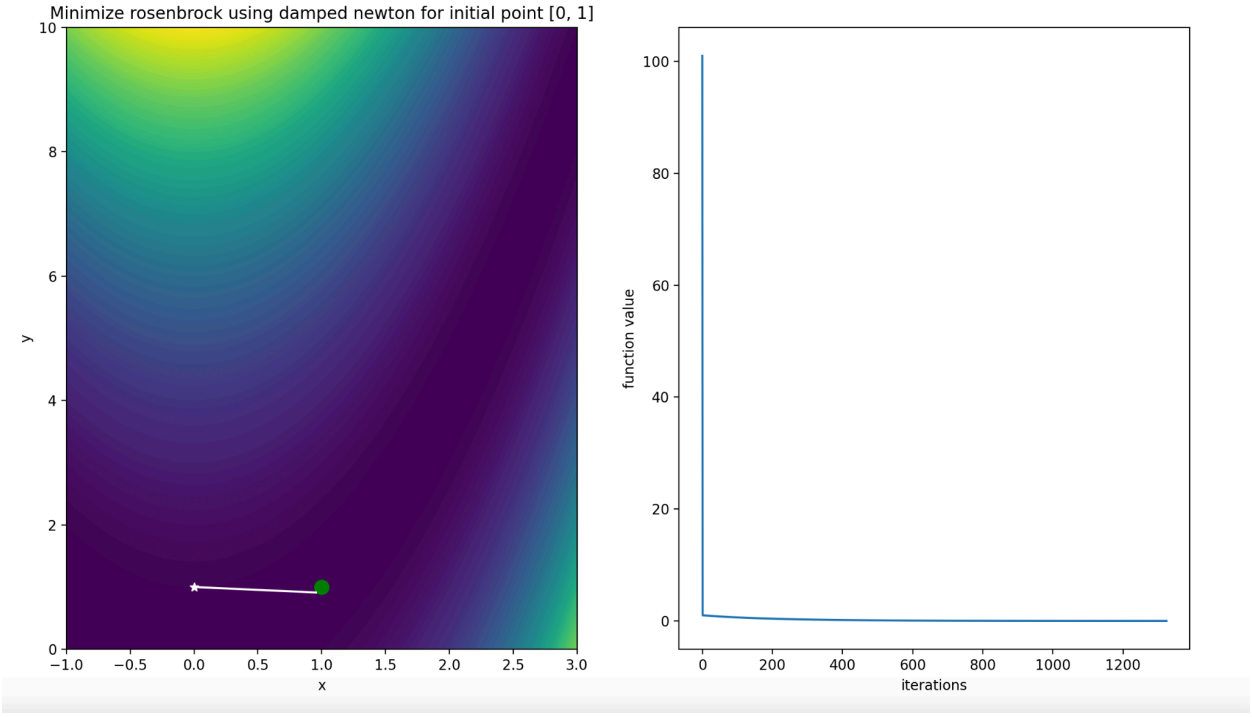
DAMPED NEWTON DESCENT (have changed max iter, alpha, beta)

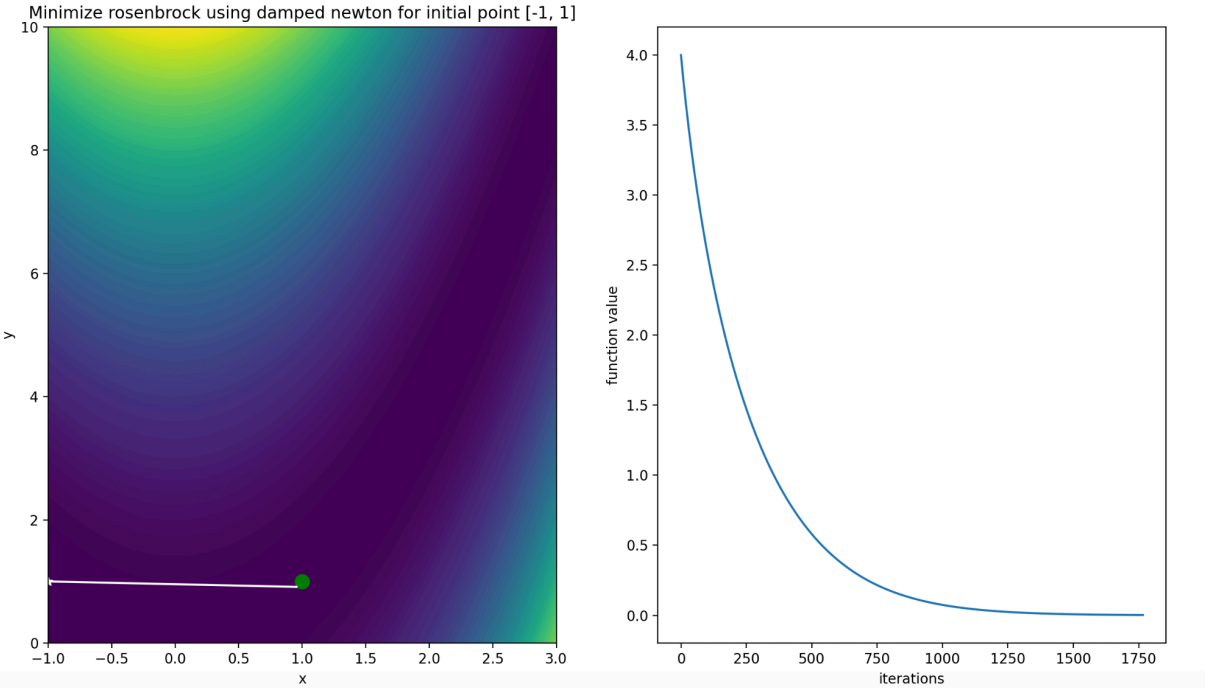
```
import matplotlib.pylab as plt
from matplotlib.pyplot import figure
import numpy as np

def rosenbrock(x, y):
   return 100 * ((y - (x ** 2)) ** 2) + ((1-x) ** 2)

def gradient(x, y):
   return np.array([-400 * x * (y - (x ** 2)) - 2 * (1-x), 200 * (y - (x ** 2))])

def hessian(x, y):
   return np.array([[(-400 * y) + (1200 * (x ** 2)) + 2, -400*x], [-400 * x, 200]])

def plotc(fun, xmin, xmax, ymin, ymax, contour=100, colour=True):
   x = np.linspace(xmin, xmax, 300)
   y = np.linspace(ymin, ymax, 300)
   X, Y = np.meshgrid(x, y)
   Z = fun(X, Y)
   if colour:
       plt.contourf(X, Y, Z, contour)
   else:
       plt.contour(X, Y, Z, contour)
```

```
   plt.scatter(1, 1, marker='o', s=100, color='g')

figure(figsize=(15, 8))
plotc(rosenbrock, -30,30, -100, 300, colour=False)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Contours plot rosenback function")
def newton_line(fun, gradient, hessian, init, tol=1e-5, max_iter=10000, alpha=0.01,
beta=0.05):
   a = init
   t = 1 #step size
   count = 0
   values = [a]
   f_prev = fun(a[0], a[1])
   f_values = [f_prev]
   while count < max_iter:
       # computer hessian and gradient
       g = gradient(a[0], a[1])
       h = hessian(a[0], a[1])
       dir = -np.linalg.solve(h, g) # # Let numpy handle the complex calculations,
negative gradient direction
       val = a + t*dir
       while(fun(val[0], val[1]) > (fun(a[0], a[1]) + alpha * t * np.dot(g.T, dir))):
#alpha and beta for backtracking
           val = a + t*dir
           t *= beta
       a += t * dir # Update
       current = fun(a[0], a[1])
       if np.isnan(current):
           break
       values.append(a)
       f_values.append(current)
       if abs(current - f_prev) < tol:
           # under tolerance hence end the algorithm
           break
       f_prev = current
       count = count + 1
   return np.array(values), np.array(f_values)
for elem in [[-1, 1], [0, 1], [2, 1]]:
   x_values, y_values = newton_line(rosenbrock, gradient, hessian, init=elem)
   figure(figsize=(15, 8))
   plt.subplot(1,2,1)
```

```
   plotc(rosenbrock, -1,3,0,10)
   plt.xlabel("x")
   plt.ylabel("y")
   title = "Minimize rosenbrock using damped newton for initial point " + str(elem)
   plt.title(title)
   plt.scatter(x_values[0,0],x_values[0,1],marker="*",color="w")
   for i in range(1,len(x_values)):
           plt.plot((x_values[i-1,0],x_values[i,0]), (x_values[i-1,1],x_values[i,1]) ,
"w");
   plt.subplot(1,2,2)
   plt.plot(y_values)
   plt.xlabel("iterations")
   plt.ylabel("function value")
plt.show()
```


Minimize rosenbrock using damped newton for initial point [2, 1]
y
x
function value
iterations


Minimize rosenbrock using damped newton for initial point [0, 1]
y
x
function value
iterations


Minimize rosenbrock using damped newton for initial point [-1, 1]
10
8
6
4
2
0
y
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
function value
0
250
500
750
1000
1250
1500
1750
iterations

(10) Given

$f: R^n \to R$

$A \to n \times n$ symmetric positive definite

$b \to n \times 1$ vector

$c \to$ scalar

(1) $\leftarrow f(x) = \frac{1}{2} x^T A x - x^T b + C$

(a) Starting point x_0

$H_f(x_0) s_0 = A s_0$ [Because $\nabla f(x) = Ax - b$, $H_f(x) = A$]

$= b - A x_0$

$\Rightarrow -\nabla f(x_0)$

Since $x_1 = x_0 + s_0$

(11) $\leftarrow Ax_1 = A(x_0 + s_0) \Rightarrow Ax_0 + As_0 \Rightarrow (b - Ax_0) + As_0 = b$

Since we know $\nabla f(x) = Ax - b$

$\Rightarrow$ Hence $Ax = b$ is an inflection point

Since $H_f(x) = A$ [Positive Semidefinite]

$\therefore Ax = b$ is minima

Hence eq (11) directly gives us x^*

(b) To compute $\min_{\alpha} f(x_0 + \alpha s_0)$ where

$s_0 = -\nabla f(x) \Rightarrow b - Ax$

For min α we get the general formula

$$\alpha = \frac{s_k^T s_k}{s_k^T A s_k} \quad [\text{From Q9}] \rightarrow (III)$$

$x_0 - x^*$ is eigenvector of $A \Rightarrow$ Given

$$\lambda(x_0 - x^*) = A(x_0 - x^*) \Rightarrow Ax_0 - b \rightarrow (IV)$$

$\therefore A s_0 = \lambda A(x^* - x_0) \Rightarrow \lambda s_0$

Hence s_0 is also an eigenvector

Putting $As_0 \Rightarrow \lambda s_0$ in (III) we get

$$\alpha = \frac{s_0^T s_0}{s_0^T \lambda s_0} \Rightarrow \frac{1}{\lambda}$$

Hence $x_1 = x_0 - \frac{1}{\lambda} \nabla f(x_0)$

$$\Rightarrow x_0 + \frac{(b - Ax_0)}{\lambda}$$

From (IV)

$$x_0 - \frac{\lambda(x_0 - x^*)}{\lambda}$$

$\Rightarrow x^*$

Hence Steepest Descent Converges in one iteration

(8) Given $f: \mathbb{R}^2 \to \mathbb{R}$

$$f(x) = \frac{1}{2}(x_1^2 - x_2)^2 + \frac{1}{2}(1 - x_1)^2$$

(a) Clearly f is of the form $\frac{X^2}{2} + \frac{Y^2}{2}$

where $X, Y \in \mathbb{R}$

So min value of $X^2, Y^2 = 0$ {Both} $\to$ (1)

Since x_1, x_2 are independent {No constraints} we can choose them in such a way that (1) holds

$\therefore x_1 = 1, x_2 = 1$ are the values

$\Rightarrow f([1,1]) = \frac{1}{2}(1^2 - 1)^2 + \frac{1}{2}(1-1)^2 \Rightarrow 0$ is the minimum value

(b) Newton's Method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

For multivariable

$$\bar{x}_{n+1} = \bar{x}_n - \left[J_f(\bar{x}_n)\right]^{-1} f(\bar{x}_n) \qquad \bar{x} = [x_1, x_2]$$

$$J_f(\bar{x}_n) = \left[\frac{\partial f(\bar{x})}{\partial x} \quad \frac{\partial f(\bar{x})}{\partial y}\right]^T$$

$$\Rightarrow \left[\frac{\partial f(\bar{x})}{\partial x_1} \quad \frac{\partial f(\bar{x})}{\partial x_2}\right]^T$$

$$\Rightarrow \left[(x_1^2 - x_2) \times 2x_1 + (1 - x_1)(-1), -(x_1^2 - x_2)\right]^T$$

$$\Rightarrow \left[2x_1^3 - 2x_1x_2 + x_1 - 1, x_2 - x_1^2\right]^T$$

$$H_f(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 6x_1^2 - 2x_2 + 1 & -2x_1 \\ -2x_1 & 1 \end{bmatrix}$$

Now, to solve

$$H_f(\bar{x}) s_0 = -\nabla f(x_0)$$

and $x_1 = x_0 + s_0$

$$\begin{bmatrix} 21 & -4 \\ -4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = -\begin{bmatrix} 9 \\ -2 \end{bmatrix}$$

$$\therefore s_0 = \begin{bmatrix} -1/5 \\ 6/5 \end{bmatrix}$$

$$\Rightarrow x_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} -1/5 \\ 6/5 \end{bmatrix}$$

$\therefore$

$$x_1 \Rightarrow \begin{bmatrix} 9/5 \\ 16/5 \end{bmatrix}$$

(c) Since the function ≥ 0, newton iteration will be good if $f(x_{k+1}) < f(x_k)$

(d) Since $|x_1 - x^*| > |x_0 - x^*|$ it goes further away in this one step

(9) $\phi(x) = \frac{1}{2} x^T A x - b^T x + c$ → Given

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 3 \end{bmatrix} \leftarrow (1)$$

$$b = [1 \; 0 \; 0]^T$$

$$x_0 = [0 \; -1 \; 0]^T$$

(a) Gradient Direction g at x_0

$$\nabla f(x_0) = Ax_0 - b$$

$$H_f(x_0) = A \Rightarrow$$

Given the value of A at (1) we obtain the eigenvalues of 2, $2-\sqrt{3}$, $2+\sqrt{3}$ all +ve hence its positive definite

∴ Gradient is "curving upwards"

(b) To find

Step size α which minimizes $\phi(x_0 + \alpha d)$

$d = -g$

$\phi(x_0 + \alpha d)$

$$\Rightarrow \frac{1}{2}(x_0 + \alpha d)^T A(x_0 + \alpha d) - b^T(x_0 + \alpha d) + c$$

$$x_0 + \alpha d = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -3 \end{bmatrix}$$

For choosing such an alpha we do

$$\frac{d\phi(x_1)}{d\alpha} \Rightarrow \nabla\phi(x_1)^T \frac{d}{d\alpha} x_1 = 0$$

$$\Rightarrow (Ax_1 - b)^T \frac{d}{d\alpha}(x_0 + \alpha d) = 0$$

Now, using residual direction

$$r_1 = b - Ax_0 \Rightarrow b - A(x_0 + \alpha d)$$

$$\therefore \alpha = -r_1^T d / d^T A d$$