Homework 4 Josh Boehm

December 4, 2022

Joshua Boehm December 1, 2022 Math 3423 Homework 4

1 Libraries/Imports

[ ]:
```
import numpy as np
import scipy as sp
from numpy.linalg import solve
from numpy.linalg import qr
from numpy.linalg import svd
from scipy.linalg import diagsvd
from scipy.linalg import pinv
from scipy.optimize import minimize
from statistics import mean
import matplotlib.pyplot as plt
```

2 Question 1

Given the data points

$$(2, 3), (3, 2), (5, 1), (6, 0)$$

derive the equation of the least-squares line

$$y = mx + b$$

that best fits the given data points. Solve the system using the QR factorization, the SVD, and the Normal Equations.

[ ]:
```
A = np.matrix([[2,1],[3,1],[5,1],[6,1]])
b = np.matrix([3,2,1,0]).T
```

2.1 QR Factorization

The matrix created from the data points:

$$\begin{bmatrix} 2, 1 \\ 3, 1 \\ 5, 1 \\ 6, 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \end{bmatrix}$$

To solve using QR factorization, we first factor A as QR. Next, we pre-multiply with Q^T, giving $Rx = Q^T b$ ($Q^T Q$ is the identity matrix) Now, we can call `solve()` with R and $Q^T b$

[ ]:
```
Q, R = qr(A)

x = solve(R, Q.T @ b)
x
```

2.1.1 Solution

$$x = \begin{bmatrix} -0.7 \\ 4.3 \end{bmatrix}$$

2.2 SVD Factorization

To solve using SVD factorization, we first factor A as SVD.
Now, we create the pseudo-inverse by taking the transpose of the the parts in reverse order, except with Σ, we take the take the reciprocals before transposing.
Last, we can call pre-multiply b with $A^\dagger$, as $A^\dagger A$ is the identity matrix

$$Ax = bU\Sigma V^T x = bA^\dagger = V\Sigma^\dagger U^T A^T A = IIx = A^\dagger bx = A^\dagger b \text{or} V\Sigma^\dagger U^T b$$

[ ]:
```
U, sigma, VT = svd(A)
Sigma = diagsvd(sigma, len(A), len(A.T))
PseudoA = (VT).T @ pinv(Sigma) @ U.T

print(PseudoA @ A)
x = PseudoA @  b
x
```

2.2.1 Solution

$$x = \begin{bmatrix} -0.7 \\ 4.3 \end{bmatrix}$$

2.3 Normal Equations

The normal equation is that which minimizes the sum of the square differences between the left and right sides:

$$A^T A x = A^T b$$

In this case, I would just use `solve(A.T @ A, A.T @ b)` and see. You could utilize QR factorizations (or any other for that matter).

2.3.1 Solution

```
[ ]: solvex = solve(A.T @ A, A.T @ b)
     solveqr = solve(R.T @ Q.T @ Q @ R, R.T @ Q.T @ b)
     solvesvd = solve(VT.T @ Sigma.T @ U.T @ U @ Sigma @ VT, VT.T @ Sigma.T @ U.T @␣
      ↪b)

     print(f"{solvex}\n\n{solveqr}\n\n{solvesvd}")
```

```
[ ]: X1 = np.array(A[:,0])
     Y1 = np.array(b)
     x = np.linspace(0,7,35)
     y = (-0.7)*x+(4.3)
     plt.scatter(X1,Y1, edgecolor='k',c='none',s=50)
     plt.plot(x,y, "r--")
```

3 Question 2

Use functions from `scipy.optimize.minimize` to minimize: The Rosenbrock function

$$f(x_1, x_2) = (7 - x_1)^2 + 100(x_2 - x_1^2)^2 + 10$$

3.1 Solution

The Rosenbrock function is defined as:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2 + c$$

In the case of our example, $a = 7$, $b = 100$, and $c = 10$.

3.1.1 Gradiant

In order to derive the gradiant, let's first define it:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix} \quad \text{OR} \quad \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \end{bmatrix}$$

First with regards to x_1:

$$\frac{\partial f}{\partial x_1}(7 - x_1)^2 + 100(x_2 - x_1^2)^2 + 10 = 400x_1^3 - 400x_1x_2 + 2x_1 - 14$$

Next with regards to x_2:

$$\frac{\partial f}{\partial x_2}(7 - x_1)^2 + 100(x_2 - x_1^2)^2 + 10 = 200x_2 - 200x_1^2$$

Thus:

$$\nabla f = \begin{bmatrix} 400x_1^3 - 400x_1x_2 + 2x_1 - 14 \\ 200x_2 - 200x_1^2 \end{bmatrix}$$

3.1.2 Hessian

Again, to derive the Hessian, it might help to define it.

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

So our second partials are as follows:

$$\begin{aligned} \frac{\partial^2 f}{\partial x_1^2} &= 1200x_1^2 - 400x_2 + 2 \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} &= -400x_1 \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} &= -400x_1 \\ \frac{\partial^2 f}{\partial x_2^2} &= 200 \end{aligned}$$

This makes the Hessian:

$$\mathbf{H_f} = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$$

3.1.3 Definitions

[ ]:
```
a = 7; b = 100; c = 10
def f(x):
   return (a - x[0])**2 + b*(x[1] - x[0]**2)**2 + c

def gradient(x):
   return np.array([-2*(a - x[0]) - 4.*b*x[0]*(x[1] - x[0]**2),
                    2*b*(x[1] - x[0]**2)])

def hessian(x):
   return np.array([[2 - 4*b*x[1] + 12*b*x[0]**2, -4*b*x[0] ],
                  [-4*b*x[0], 2*b]])

optimization_methods = {'newton-cg':{'function evals': [], 'function iters':␣
 ↪[]},
'nelder-mead':{'function evals': [], 'function iters': []},
'powell':{'function evals': [], 'function iters': []},
'bfgs':{'function evals': [], 'function iters': []},
```

```
'dogleg':{'function evals': [], 'function iters': []}}

starting_points = [np.random.randint(-20,20, size = (1,2)) for i in range(3)]
section_break = "============================================================"
```

3.1.4 Method 1: Newton-CG

```
[ ]: for x in enumerate(starting_points):
         result = minimize(f, x[1], method = 'Newton-CG',jac = gradient, hess =␣
      ↪hessian, tol = 1.e-7)

         print(section_break)
         print('Test Run', x[0] + 1, ':')
         print(section_break)

         print('Starting Value Used: ', x[1])
         print("The Minimum Occurs at (x, y) = ", result.x)
         print("The Minimum Value = ", f(result.x).round(3))

         print("Other Statistics:")
         print(result)
         print(section_break)
         print(section_break)
         print('\n')


         optimization_methods['newton-cg']['function evals'].append(result['nfev'])
         optimization_methods['newton-cg']['function iters'].append(result['nit'])


     # Displaying a summary of both function evaluations and iterations
     print(f"\
     Summary Statistics of the Method:\n\n\
     Function Evaluations:\n\
     Minimum Function Evaluations: {min(optimization_methods['newton-cg']['function␣
      ↪evals'])}\n\
     Maximum Function Evaluations: {max(optimization_methods['newton-cg']['function␣
      ↪evals'])}\n\
     Mean Function Evaluations: {mean(optimization_methods['newton-cg']['function␣
      ↪evals'])}\n\n\
     Function Iterations:\n\
     Minimum Function Iterations:  {min(optimization_methods['newton-cg']['function␣
      ↪iters'])}\n\
     Maximum Function Iterations:  {max(optimization_methods['newton-cg']['function␣
      ↪iters'])}\n\
```

```
Mean Function Iterations:  {mean(optimization_methods['newton-cg']['function␣
↪iters'])}\n")
```

3.1.5 Method 2: Nelder-Mead

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'Nelder-Mead',tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)

    print(section_break)
    print(section_break)
    print('\n')

    optimization_methods['nelder-mead']['function evals'].append(result['nfev'])
    optimization_methods['nelder-mead']['function iters'].append(result['nit'])

# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations:␣
↪{min(optimization_methods['nelder-mead']['function evals'])}\n\
Maximum Function Evaluations:␣
↪{max(optimization_methods['nelder-mead']['function evals'])}\n\
Mean Function Evaluations: {mean(optimization_methods['nelder-mead']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations: ␣
↪{min(optimization_methods['nelder-mead']['function iters'])}\n\
Maximum Function Iterations: ␣
↪{max(optimization_methods['nelder-mead']['function iters'])}\n\
Mean Function Iterations:  {mean(optimization_methods['nelder-mead']['function␣
↪iters'])}\n")
```

3.1.6 Method 3: Powell

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'Powell',tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)

    print(section_break)
    print(section_break)
    print('\n')

    optimization_methods['powell']['function evals'].append(result['nfev'])
    optimization_methods['powell']['function iters'].append(result['nit'])

# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations: {min(optimization_methods['powell']['function␣
↪evals'])}\n\
Maximum Function Evaluations: {max(optimization_methods['powell']['function␣
↪evals'])}\n\
Mean Function Evaluations: {mean(optimization_methods['powell']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations:  {min(optimization_methods['powell']['function␣
↪iters'])}\n\
Maximum Function Iterations:  {max(optimization_methods['powell']['function␣
↪iters'])}\n\
Mean Function Iterations:  {mean(optimization_methods['powell']['function␣
↪iters'])}\n")
```

3.1.7 Method 4: BFGS

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'BFGS',tol = 1.e-7)

    print(section_break)
```

```
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)

    print(section_break)
    print(section_break)
    print('\n')

    optimization_methods['bfgs']['function evals'].append(result['nfev'])
    optimization_methods['bfgs']['function iters'].append(result['nit'])

# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations: {min(optimization_methods['bfgs']['function␣
↪evals'])}\n\
Maximum Function Evaluations: {max(optimization_methods['bfgs']['function␣
↪evals'])}\n\
Mean Function Evaluations: {mean(optimization_methods['bfgs']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations:  {min(optimization_methods['bfgs']['function␣
↪iters'])}\n\
Maximum Function Iterations:  {max(optimization_methods['bfgs']['function␣
↪iters'])}\n\
Mean Function Iterations:  {mean(optimization_methods['bfgs']['function␣
↪iters'])}\n")
```

3.1.8 Method 5: Dogleg

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'dogleg',jac = gradient, hess =␣
↪hessian, tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
```

```
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)
    print(section_break)
    print(section_break)
    print('\n')


    optimization_methods['dogleg']['function evals'].append(result['nfev'])
    optimization_methods['dogleg']['function iters'].append(result['nit'])


# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations: {min(optimization_methods['dogleg']['function␣
↪evals'])}\n\
Maximum Function Evaluations: {max(optimization_methods['dogleg']['function␣
↪evals'])}\n\
Mean Function Evaluations: {mean(optimization_methods['dogleg']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations:  {min(optimization_methods['dogleg']['function␣
↪iters'])}\n\
Maximum Function Iterations:  {max(optimization_methods['dogleg']['function␣
↪iters'])}\n\
Mean Function Iterations:  {mean(optimization_methods['dogleg']['function␣
↪iters'])}\n")
```

3.2 Conclusion

It seems the `dogleg` method is the most efficient method at find the minimum with fewest of every summary statistic escribed.

4 Question 3

Use functions from `scipy.optimize.minimize` to minimize: The Booth function

$$f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

4.1 Solution

In this case, the function is defined as it's base form. We need only to find the Gradiant and the Hessian for the function.

4.1.1 Gradiant

Again, the definition of the gradiant is:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix} \quad \text{OR} \quad \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \end{bmatrix}$$

First with regards to x_1:

$$\frac{\partial f}{\partial x_1}(x_1 + 2x_2 – 7)^2 + (2x_1 + x_2 – 5)^2 = 10x_1 + 8x_2 - 34$$

Next with regards to x_2:

$$\frac{\partial f}{\partial x_2}(x_1 + 2x_2 – 7)^2 + (2x_1 + x_2 – 5)^2 = 8x_1 + 10x_2 - 38$$

Thus:

$$\nabla f = \begin{bmatrix} 10x_1 + 8x_2 - 34 \\ 8x_1 + 10x_2 - 38 \end{bmatrix}$$

4.1.2 Hessian

As before, the Hessian definition:

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

So our second partials are as follows:

$$\begin{aligned} \frac{\partial^2 f}{\partial x_1^2} &= 10 \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} &= 8 \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} &= 8 \\ \frac{\partial^2 f}{\partial x_2^2} &= 10 \end{aligned}$$

This makes the Hessian:

$$\mathbf{H_f} = \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}$$

4.1.3 Definitions

[ ]:
```
def booth(x):
    return (x[0] + 2*x[1] - 7)**2 + (2*x[0] + x[1] - 5)**2

def booth_gradient(x):
    return np.array([10*x[0] + 8*x[1] - 34, 8*x[0] + 10*x[1] - 38])

def booth_hessian(x):
    return np.array([[10, 8], [8, 10]])

booth_optimization_methods = {'newton-cg':{'function evals': [], 'function␣
↪iters': []},
'nelder-mead':{'function evals': [], 'function iters': []},
'powell':{'function evals': [], 'function iters': []},
'bfgs':{'function evals': [], 'function iters': []},
'dogleg':{'function evals': [], 'function iters': []}}

starting_points = [np.random.randint(-20,20, size = (1,2)) for i in range(3)]
```

4.1.4 Method 1: Newton-CG

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(booth, x[1], method = 'Newton-CG',jac = booth_gradient,␣
↪hess = booth_hessian, tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)
    print(section_break)
    print(section_break)
    print('\n')


    booth_optimization_methods['newton-cg']['function evals'].
↪append(result['nfev'])
    booth_optimization_methods['newton-cg']['function iters'].
↪append(result['nit'])
```

```
# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations:␣
↪{min(booth_optimization_methods['newton-cg']['function evals'])}\n\
Maximum Function Evaluations:␣
↪{max(booth_optimization_methods['newton-cg']['function evals'])}\n\
Mean Function Evaluations:␣
↪{mean(booth_optimization_methods['newton-cg']['function evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations: ␣
↪{min(booth_optimization_methods['newton-cg']['function iters'])}\n\
Maximum Function Iterations: ␣
↪{max(booth_optimization_methods['newton-cg']['function iters'])}\n\
Mean Function Iterations: ␣
↪{mean(booth_optimization_methods['newton-cg']['function iters'])}\n")
```

4.1.5 Method 2: Nelder-Mead

```
[ ]: for x in enumerate(starting_points):
    result = minimize(booth, x[1], method = 'Nelder-Mead', tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)
    print(section_break)
    print(section_break)
    print('\n')


    booth_optimization_methods['nelder-mead']['function evals'].
↪append(result['nfev'])
    booth_optimization_methods['nelder-mead']['function iters'].
↪append(result['nit'])


# Displaying a summary of both function evaluations and iterations
print(f"\
```

```
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations:␣
↪{min(booth_optimization_methods['nelder-mead']['function evals'])}\n\
Maximum Function Evaluations:␣
↪{max(booth_optimization_methods['nelder-mead']['function evals'])}\n\
Mean Function Evaluations:␣
↪{mean(booth_optimization_methods['nelder-mead']['function evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations: ␣
↪{min(booth_optimization_methods['nelder-mead']['function iters'])}\n\
Maximum Function Iterations: ␣
↪{max(booth_optimization_methods['nelder-mead']['function iters'])}\n\
Mean Function Iterations: ␣
↪{mean(booth_optimization_methods['nelder-mead']['function iters'])}\n")
```

4.1.6 Method 3: Powell

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'Powell',tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)

    print(section_break)
    print(section_break)
    print('\n')

    booth_optimization_methods['powell']['function evals'].
↪append(result['nfev'])
    booth_optimization_methods['powell']['function iters'].append(result['nit'])

# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations:␣
↪{min(booth_optimization_methods['powell']['function evals'])}\n\
```

```
Maximum Function Evaluations:␣
↪{max(booth_optimization_methods['powell']['function evals'])}\n\
Mean Function Evaluations: {mean(booth_optimization_methods['powell']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations: ␣
↪{min(booth_optimization_methods['powell']['function iters'])}\n\
Maximum Function Iterations: ␣
↪{max(booth_optimization_methods['powell']['function iters'])}\n\
Mean Function Iterations:  {mean(booth_optimization_methods['powell']['function␣
↪iters'])}\n")
```

4.1.7 Method 4: BFGS

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'BFGS',tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)

    print(section_break)
    print(section_break)
    print('\n')

    booth_optimization_methods['bfgs']['function evals'].append(result['nfev'])
    booth_optimization_methods['bfgs']['function iters'].append(result['nit'])

# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations: {min(booth_optimization_methods['bfgs']['function␣
↪evals'])}\n\
Maximum Function Evaluations: {max(booth_optimization_methods['bfgs']['function␣
↪evals'])}\n\
Mean Function Evaluations: {mean(booth_optimization_methods['bfgs']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
```

```
Minimum Function Iterations:  {min(booth_optimization_methods['bfgs']['function␣
↪iters'])}\n\
Maximum Function Iterations:  {max(booth_optimization_methods['bfgs']['function␣
↪iters'])}\n\
Mean Function Iterations:  {mean(booth_optimization_methods['bfgs']['function␣
↪iters'])}\n")
```

4.1.8 Method 5: Dogleg

[ ]:
```
for x in enumerate(starting_points):
    result = minimize(f, x[1], method = 'dogleg',jac = gradient, hess =␣
↪hessian, tol = 1.e-7)

    print(section_break)
    print('Test Run', x[0] + 1, ':')
    print(section_break)

    print('Starting Value Used: ', x[1])
    print("The Minimum Occurs at (x, y) = ", result.x)
    print("The Minimum Value = ", f(result.x).round(3))

    print("Other Statistics:")
    print(result)
    print(section_break)
    print(section_break)
    print('\n')


    booth_optimization_methods['dogleg']['function evals'].
↪append(result['nfev'])
    booth_optimization_methods['dogleg']['function iters'].append(result['nit'])


# Displaying a summary of both function evaluations and iterations
print(f"\
Summary Statistics of the Method:\n\n\
Function Evaluations:\n\
Minimum Function Evaluations:␣
↪{min(booth_optimization_methods['dogleg']['function evals'])}\n\
Maximum Function Evaluations:␣
↪{max(booth_optimization_methods['dogleg']['function evals'])}\n\
Mean Function Evaluations: {mean(booth_optimization_methods['dogleg']['function␣
↪evals'])}\n\n\
Function Iterations:\n\
Minimum Function Iterations: ␣
↪{min(booth_optimization_methods['dogleg']['function iters'])}\n\
```

```
Maximum Function Iterations: ␣
↪{max(booth_optimization_methods['dogleg']['function iters'])}\n\
Mean Function Iterations:  {mean(booth_optimization_methods['dogleg']['function␣
↪iters'])}\n")
```

4.2 Conclusion

It seems the `dogleg` method is the most efficient method at find the minimum with fewest of every summary statistic escribed.