MATH 140: Mathematical Methods for Optimization
Assignment 4—Spring 2024
Due February 13, 2024
By:Ronald Nap

Note: Hessian's and Derivatives computed using Wolfram Alpha

1. (**10 points**) Consider the unconstrained optimization problem

$$\min f(x,y) \equiv -\cos x \cos(y/10).$$

(a) Find and classify all stationary points in the region $-\pi/2 \le x \le \pi/2, -10\pi/2 \le y \le 10\pi/2$

Solution: The function given is $f(x,y) = -\cos(x)\cos(y/10)$. To find the stationary points, we first compute the gradient of $f(x,y)$:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right).$$

The partial derivatives are:

$$\frac{\partial f}{\partial x} = \sin(x)\cos\left(\frac{y}{10}\right),$$

$$\frac{\partial f}{\partial y} = \frac{1}{10}\cos(x)\sin\left(\frac{y}{10}\right).$$

To find the stationary points, we set each component of the gradient to zero within the region $-\frac{\pi}{2} \le x \le \frac{\pi}{2}, -\frac{10\pi}{2} \le y \le \frac{10\pi}{2}$.

$$\sin(x)\cos\left(\frac{y}{10}\right) = 0,$$

$$\frac{1}{10}\cos(x)\sin\left(\frac{y}{10}\right) = 0.$$

This results in the only solution being $x = 0$ and $y = 0$, as all other potential solutions do not satisfy both conditions simultaneously within the region.

To classify the stationary point, we examine the second derivatives and the Hessian matrix:

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} \cos(x)\cos\left(\frac{y}{10}\right) & -\frac{1}{10}\sin(x)\sin\left(\frac{y}{10}\right) \\ -\frac{1}{10}\sin(x)\sin\left(\frac{y}{10}\right) & \frac{1}{100}\cos(x)\cos\left(\frac{y}{10}\right) \end{bmatrix}.$$

At the point $(0,0)$, the Hessian matrix simplifies to:

$$H(f)_{(0,0)} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{100} \end{bmatrix}.$$

The determinant of this Hessian matrix is positive, as $\det(H(f)_{(0,0)}) = 1 \times \frac{1}{100} - 0 = \frac{1}{100}$. Since both diagonal elements (which are also the eigenvalues in this case) are positive, it indicates that the stationary point at $(0,0)$ is a local minimum.

(b) There is a portion of the problem region within which the Hessian matrix of $f(x,y)$ is positive definite. Give expressions for this portion. You should be able to do this analytically but I recommend using Matlab/python.

Solution: Using 1a), the Hessian matrix $H(f)$ is

$$H(f) = \begin{bmatrix} \cos(x)\cos\left(\frac{y}{10}\right) & -\frac{1}{10}\sin(x)\sin\left(\frac{y}{10}\right) \\ -\frac{1}{10}\sin(x)\sin\left(\frac{y}{10}\right) & \frac{1}{100}\cos(x)\cos\left(\frac{y}{10}\right) \end{bmatrix}.$$

The determinant of $H(f)$ is:

$$\det(H(f)) = 0.005\cos(2x) + 0.005\cos\left(\frac{y}{5}\right).$$

For the Hessian matrix to be positive definite, the determinant must be greater than zero.

$$0.005\cos(2x) + 0.005\cos\left(\frac{y}{5}\right) > 0.$$

This condition is met within the domain where $\cos(x)$ and $\cos\left(\frac{y}{10}\right)$ are both positive or both negative. Given the specified domain $-\pi/2 \le x \le \pi/2, -10\pi/2 \le y \le 10\pi/2$, $\cos(x)$ is positive for $-\pi/2 < x < \pi/2$, and similarly, $\cos\left(\frac{y}{10}\right)$ is positive for $-10\pi/2 < y < 10\pi/2$. Therefore, within the given domain, the Hessian matrix can be positive definite where $\cos(x)$ and $\cos\left(\frac{y}{10}\right)$ maintain their positive values and satisfy the determinant condition.

(c) Derive expressions for the search directions associated with the steepest descent method.

Solution: The steepest descent method moves from the current point in the direction of the most rapid decrease of the function. This direction is opposite to the gradient of the function at the current point. For $f(x,y)$, the gradient is

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right).$$

The search direction at any point (x_k, y_k) is therefore

$$d_k = -\nabla f(x_k, y_k) = -\left(\frac{\partial f}{\partial x}(x_k, y_k), \frac{\partial f}{\partial y}(x_k, y_k)\right).$$

For the function $f(x,y) = -\cos(x)\cos(y/10)$, the partial derivatives are

$$\frac{\partial f}{\partial x} = \sin(x)\cos\left(\frac{y}{10}\right),$$

$$\frac{\partial f}{\partial y} = -\frac{1}{10}\cos(x)\sin\left(\frac{y}{10}\right).$$

Hence, the search direction for steepest descent at (x_k, y_k) is

$$d_k = -\left(\sin(x_k)\cos\left(\frac{y_k}{10}\right), -\frac{1}{10}\cos(x_k)\sin\left(\frac{y_k}{10}\right)\right).$$

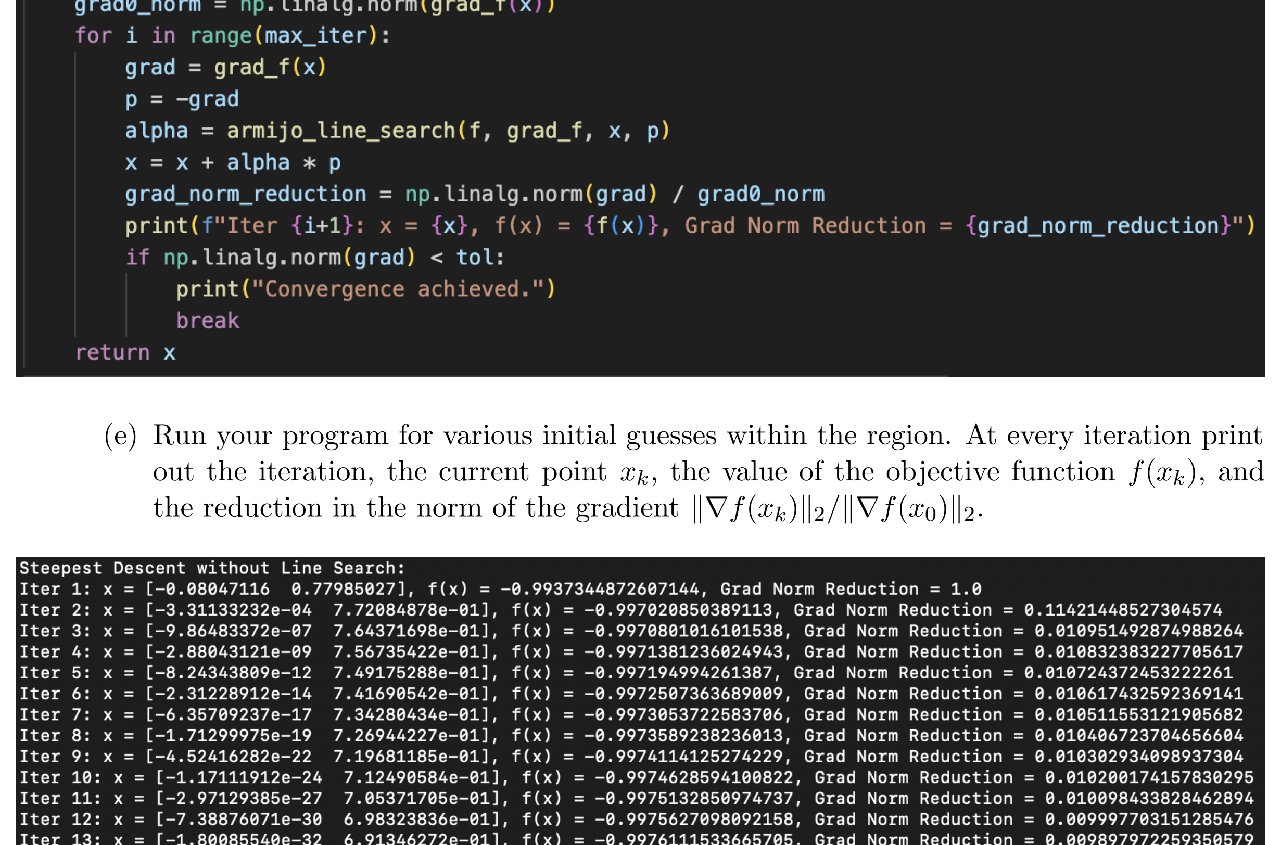
This expression will be used to update the current point to the next point in steepest descent.

(d) Write a program that performs the steepest descent iterations, without a line search, with an exact line search and with Armijo and backtracking (as discussed in class). Note that you will not be able to find the value of the optimal step length analytically; instead, determine it numerically. *Suggestion:* use a built-in one-dimensional minimization function such as `FindMinimum` or `FindRoot` in MATHEMATICA or `fzero` in MATLAB.

Solution: Refer to HW4.py for full code

```
def steepest_descent_no_line_search(x0, max_iter=100, tol=1e-6, step_size=0.01):
    x = x0
    grad0_norm = np.linalg.norm(grad_f(x))
    for i in range(max_iter):
        grad = grad_f(x)
        x = x - step_size * grad
        grad_norm_reduction = np.linalg.norm(grad) / grad0_norm
        print(f"Iter {i+1}: x = {x}, f(x) = {f(x)}, Grad Norm Reduction = {grad_norm_reduction}")
        if np.linalg.norm(grad) < tol:
            print("Convergence achieved.")
            break
    return x


def exact_line_search(f, x, p):
    func = lambda alpha: f(x + alpha * p)
    result = minimize_scalar(func)
    return result.x


def steepest_descent_exact_line_search(x0, max_iter=100, tol=1e-6):
    x = x0
    grad0_norm = np.linalg.norm(grad_f(x))
    for i in range(max_iter):
        grad = grad_f(x)
        alpha = exact_line_search(f, x, -grad)
        x = x - alpha * grad
        grad_norm_reduction = np.linalg.norm(grad) / grad0_norm
        print(f"Iter {i+1}: x = {x}, f(x) = {f(x)}, Grad Norm Reduction = {grad_norm_reduction}")
        if np.linalg.norm(grad) < tol:
            print("Convergence achieved.")
            break
    return x


def armijo_line_search(f, grad_f, x, p, alpha=1.0, beta=0.5, sigma=0.1):
    while f(x + alpha * p) > f(x) + sigma * alpha * np.dot(grad_f(x), p):
        alpha *= beta
    return alpha


def steepest_descent_armijo(x0, max_iter=100, tol=1e-6):
    x = x0
    grad0_norm = np.linalg.norm(grad_f(x))
    for i in range(max_iter):
        grad = grad_f(x)
        p = -grad
        alpha = armijo_line_search(f, grad_f, x, p)
        x = x + alpha * p
        grad_norm_reduction = np.linalg.norm(grad) / grad0_norm
        print(f"Iter {i+1}: x = {x}, f(x) = {f(x)}, Grad Norm Reduction = {grad_norm_reduction}")
        if np.linalg.norm(grad) < tol:
            print("Convergence achieved.")
            break
    return x
```

(e) Run your program for various initial guesses within the region. At every iteration print out the iteration, the current point x_k, the value of the objective function $f(x_k)$, and the reduction in the norm of the gradient $\|\nabla f(x_k)\|_2 / \|\nabla f(x_0)\|_2$.



```
Steepest Descent without Line Search:
Iter 1: x = [-0.08047116  0.77985027], f(x) = -0.9937344872607144, Grad Norm Reduction = 1.0
Iter 2: x = [-3.31133232e-04  7.72084878e-01], f(x) = -0.997020850389113, Grad Norm Reduction = 0.11421448527304574
Iter 3: x = [-9.86483372e-07  7.64371698e-01], f(x) = -0.9970801016101538, Grad Norm Reduction = 0.010951492874988264
Iter 4: x = [-2.88043121e-09  7.56735422e-01], f(x) = -0.9971381236024943, Grad Norm Reduction = 0.010832383227705617
Iter 5: x = [-8.24343809e-12  7.49175288e-01], f(x) = -0.997194994261387, Grad Norm Reduction = 0.010724372453222261
Iter 6: x = [-2.31228912e-14  7.41690542e-01], f(x) = -0.9972507363689009, Grad Norm Reduction = 0.010617432592369141
Iter 7: x = [-6.35709237e-17  7.34280434e-01], f(x) = -0.9973053722583706, Grad Norm Reduction = 0.010511553121905682
Iter 8: x = [-1.71299975e-19  7.26944227e-01], f(x) = -0.9973589238236013, Grad Norm Reduction = 0.010406723704656604
Iter 9: x = [-4.52416282e-22  7.19681185e-01], f(x) = -0.9974114125274229, Grad Norm Reduction = 0.010302934098937304
Iter 10: x = [-1.17111912e-24  7.12490584e-01], f(x) = -0.9974628594100822, Grad Norm Reduction = 0.010200174157830295
Iter 11: x = [-2.97129385e-27  7.05371705e-01], f(x) = -0.9975132850974737, Grad Norm Reduction = 0.010098433828462894
Iter 12: x = [-7.38876071e-30  6.98323836e-01], f(x) = -0.9975627098092158, Grad Norm Reduction = 0.009997703151285476
Iter 13: x = [-1.80085540e-32  6.91346272e-01], f(x) = -0.9976111533665705, Grad Norm Reduction = 0.009897972259350579
Iter 14: x = [-4.30196736e-35  6.84438315e-01], f(x) = -0.9976586352002141, Grad Norm Reduction = 0.009799231377592987
Iter 15: x = [-1.00724749e-37  6.77599274e-01], f(x) = -0.9977051743578582, Grad Norm Reduction = 0.009701470822110952
Iter 16: x = [-2.31145738e-40  6.70828466e-01], f(x) = -0.9977507895117249, Grad Norm Reduction = 0.009604680999448752
Iter 17: x = [-5.19895418e-43  6.64125211e-01], f(x) = -0.99779549896588, Grad Norm Reduction = 0.009508852405880718
Iter 18: x = [-1.14610999e-45  6.57488840e-01], f(x) = -0.9978393206634243, Grad Norm Reduction = 0.009413975626696904
Iter 19: x = [-2.47637616e-48  6.50918688e-01], f(x) = -0.9978822721935482, Grad Norm Reduction = 0.009320041335490513
Iter 20: x = [-5.24429066e-51  6.44414096e-01], f(x) = -0.9979243707984506, Grad Norm Reduction = 0.009227040293447250
Iter 21: x = [-1.08852028e-53  6.37974415e-01], f(x) = -0.997965633380125, Grad Norm Reduction = 0.00913496334863671
Iter 22: x = [-2.21444933e-56  6.31598997e-01], f(x) = -0.9980060765070146, Grad Norm Reduction = 0.009043801435305934
Iter 23: x = [-4.41544254e-59  6.25287206e-01], f(x) = -0.9980457164205407, Grad Norm Reduction = 0.008953545573175287
Iter 24: x = [-8.62902686e-62  6.19038408e-01], f(x) = -0.9980845690415044, Grad Norm Reduction = 0.008864186866736696
Iter 25: x = [-1.65283052e-64  6.12851976e-01], f(x) = -0.9981226499763658, Grad Norm Reduction = 0.008775716504554469

Steepest Descent with Exact Line Search:
Iter 1: x = [6.14500817e-05 7.79216462e-01], f(x) = -0.99696564243836, Grad Norm Reduction = 1.0
Iter 2: x = [-0.00603374  0.00474875], f(x) = -0.9999816842796219, Grad Norm Reduction = 0.011042677518328242
Iter 3: x = [3.70243097e-07 4.70126081e-03], f(x) = -0.9999998894906676, Grad Norm Reduction = 0.008559333360528238
Iter 4: x = [-3.64283336e-05  2.86627036e-05], f(x) = -0.9999999993323806, Grad Norm Reduction = 6.669145573192538e-05
Convergence achieved.

Steepest Descent with Armijo Line Search:
Iter 1: x = [-0.08047116  0.77985027], f(x) = -0.9937344872607144, Grad Norm Reduction = 1.0
Iter 2: x = [-3.31133232e-04  7.72084878e-01], f(x) = -0.997020850389113, Grad Norm Reduction = 0.11421448527304574
Iter 3: x = [-9.86483372e-07  7.64371698e-01], f(x) = -0.9970801016101538, Grad Norm Reduction = 0.010951492874988264
Iter 4: x = [-2.88043121e-09  7.56735422e-01], f(x) = -0.9971381236024943, Grad Norm Reduction = 0.010832383227705617
Iter 5: x = [-8.24343809e-12  7.49175288e-01], f(x) = -0.997194994261387, Grad Norm Reduction = 0.010724372453222261
Iter 6: x = [-2.31228912e-14  7.41690542e-01], f(x) = -0.9972507363689009, Grad Norm Reduction = 0.010617432592369141
Iter 7: x = [-6.35709237e-17  7.34280434e-01], f(x) = -0.9973053722583706, Grad Norm Reduction = 0.010511553121905682
Iter 8: x = [-1.71299975e-19  7.26944227e-01], f(x) = -0.9973589238236013, Grad Norm Reduction = 0.010406723704656604
Iter 9: x = [-4.52416282e-22  7.19681185e-01], f(x) = -0.9974114125274229, Grad Norm Reduction = 0.010302934098937304
Iter 10: x = [-1.17111912e-24  7.12490584e-01], f(x) = -0.9974628594100822, Grad Norm Reduction = 0.010200174157830295
Iter 11: x = [-2.97129385e-27  7.05371705e-01], f(x) = -0.9975132850974737, Grad Norm Reduction = 0.010098433828462894
Iter 12: x = [-7.38876071e-30  6.98323836e-01], f(x) = -0.9975627098092158, Grad Norm Reduction = 0.009997703151285476
Iter 13: x = [-1.80085540e-32  6.91346272e-01], f(x) = -0.9976111533665705, Grad Norm Reduction = 0.009897972259350579
Iter 14: x = [-4.30196736e-35  6.84438315e-01], f(x) = -0.9976586352002141, Grad Norm Reduction = 0.009799231377592987
Iter 15: x = [-1.00724749e-37  6.77599274e-01], f(x) = -0.9977051743578582, Grad Norm Reduction = 0.009701470822110952
Iter 16: x = [-2.31145738e-40  6.70828466e-01], f(x) = -0.9977507895117249, Grad Norm Reduction = 0.009604680999448752
Iter 17: x = [-5.19895418e-43  6.64125211e-01], f(x) = -0.99779549896588, Grad Norm Reduction = 0.009508852405880718
Iter 18: x = [-1.14610999e-45  6.57488840e-01], f(x) = -0.9978393206634243, Grad Norm Reduction = 0.009413975626696904
Iter 19: x = [-2.47637616e-48  6.50918688e-01], f(x) = -0.9978822721935482, Grad Norm Reduction = 0.009320041335490513
Iter 20: x = [-5.24429066e-51  6.44414096e-01], f(x) = -0.9979243707984506, Grad Norm Reduction = 0.009227040293447250
Iter 21: x = [-1.08852028e-53  6.37974415e-01], f(x) = -0.997965633380125, Grad Norm Reduction = 0.00913496334863671
Iter 22: x = [-2.21444933e-56  6.31598997e-01], f(x) = -0.9980060765070146, Grad Norm Reduction = 0.009043801435305934
Iter 23: x = [-4.41544254e-59  6.25287206e-01], f(x) = -0.9980457164205407, Grad Norm Reduction = 0.008953545573175287
Iter 24: x = [-8.62902686e-62  6.19038408e-01], f(x) = -0.9980845690415044, Grad Norm Reduction = 0.008864186866736696
Iter 25: x = [-1.65283052e-64  6.12851976e-01], f(x) = -0.9981226499763658, Grad Norm Reduction = 0.008775716504554469
```

Solution: The approach utilizing exact line search exhibited the quickest convergence to the function's minimum, demonstrating substantial efficacy in optimization. Next was the Armijo line search, which also effectively adjusted step sizes adaptively but at a marginally slower rate. The version without a line search showed the slowest convergence, highlighting the role of line search in enhancing the steepest descent method's convergence speed.

(f) Verify that the Steepest descent method converges to the minimum x^* for any starting point within the region.

Solution: Each approach—without line search, with exact line search, and with Armijo line search—converged towards the function's minimum. This convergence behavior across different starting points confirms the robustness of the steepest descent method in finding the minimum x^* within the specified region.

```
Iter 1: x = [-0.77834889  0.78534268], f(x) = -0.7098789819866576, Grad Norm Reduction = 1.0
Iter 2: x = [-0.77134949  0.78528682], f(x) = -0.7147610358828453, Grad Norm Reduction = 0.9929272507450424
Iter 3: x = [-0.76439994  0.78523057], f(x) = -0.7195736811385084, Grad Norm Reduction = 0.9858557244468117
Iter 4: x = [-0.75750022  0.78517396], f(x) = -0.7243174431734963, Grad Norm Reduction = 0.9787867946655945
Iter 5: x = [-0.75065032  0.78511697], f(x) = -0.7289928596702338, Grad Norm Reduction = 0.971721805011597
Iter 6: x = [-0.74385019  0.78505961], f(x) = -0.7336004799122335, Grad Norm Reduction = 0.9646620691332755
Iter 7: x = [-0.73709978  0.7850019 ], f(x) = -0.738140864133974, Grad Norm Reduction = 0.9576088707328713
Iter 8: x = [-0.73039905  0.78494384], f(x) = -0.7426145828825158, Grad Norm Reduction = 0.9505634636081087
Iter 9: x = [-0.72374792  0.78488543], f(x) = -0.7470222163911996, Grad Norm Reduction = 0.9435270717190128
Iter 10: x = [-0.71714633  0.78482668], f(x) = -0.7513643539657264, Grad Norm Reduction = 0.9365008892788087
Iter 11: x = [-0.71059419  0.78476759], f(x) = -0.7556415933828987, Grad Norm Reduction = 0.9294860808678659
Iter 12: x = [-0.70409143  0.78470816], f(x) = -0.7598545403022542, Grad Norm Reduction = 0.9224837815696675
Iter 13: x = [-0.69763793  0.78464841], f(x) = -0.7640038076908072, Grad Norm Reduction = 0.9154950971277833
Iter 14: x = [-0.6912336   0.78458834], f(x) = -0.7680900152610703, Grad Norm Reduction = 0.9085211041228451
Iter 15: x = [-0.68487833  0.78452796], f(x) = -0.7721137889225084, Grad Norm Reduction = 0.9015628501685319
Iter 16: x = [-0.678572    0.78446726], f(x) = -0.7760757602465417, Grad Norm Reduction = 0.8946213541255853
Iter 17: x = [-0.67231449  0.78440625], f(x) = -0.7799765659451969, Grad Norm Reduction = 0.8876976063328924
Iter 18: x = [-0.66610565  0.78434494], f(x) = -0.7838168473634707, Grad Norm Reduction = 0.8807925688546948
Iter 19: x = [-0.65994536  0.78428334], f(x) = -0.78759724998545, Grad Norm Reduction = 0.873907175742985
Iter 20: x = [-0.65383347  0.78422144], f(x) = -0.7913184229542104, Grad Norm Reduction = 0.867042333314192
Iter 21: x = [-0.64776983  0.78415926], f(x) = -0.7949810186054919, Grad Norm Reduction = 0.8601989204392544
Iter 22: x = [-0.64175427  0.78409679], f(x) = -0.7985856920151263, Grad Norm Reduction = 0.8533777888462205
Iter 23: x = [-0.63578665  0.78403404], f(x) = -0.8021331005601775, Grad Norm Reduction = 0.8465797634345155
Iter 24: x = [-0.62986678  0.78397103], f(x) = -0.8056239034937297, Grad Norm Reduction = 0.8398056426000561
Iter 25: x = [-0.62399451  0.78390774], f(x) = -0.8090587615332484, Grad Norm Reduction = 0.833056198570401
Iter 1: x = [ 0.77834889 -0.78534268], f(x) = -0.7098789819866576, Grad Norm Reduction = 1.0
Iter 2: x = [ 0.77134949 -0.78528682], f(x) = -0.7147610358828453, Grad Norm Reduction = 0.9929272507450424
Iter 3: x = [ 0.76439994 -0.78523057], f(x) = -0.7195736811385084, Grad Norm Reduction = 0.9858557244468117
Iter 4: x = [ 0.75750022 -0.78517396], f(x) = -0.7243174431734963, Grad Norm Reduction = 0.9787867946655945
Iter 5: x = [ 0.75065032 -0.78511697], f(x) = -0.7289928596702338, Grad Norm Reduction = 0.971721805011597
Iter 6: x = [ 0.74385019 -0.78505961], f(x) = -0.7336004799122335, Grad Norm Reduction = 0.9646620691332755
Iter 7: x = [ 0.73709978 -0.7850019 ], f(x) = -0.738140864133974, Grad Norm Reduction = 0.9576088707328713
Iter 8: x = [ 0.73039905 -0.78494384], f(x) = -0.7426145828825158, Grad Norm Reduction = 0.9505634636081087
Iter 9: x = [ 0.72374792 -0.78488543], f(x) = -0.7470222163911996, Grad Norm Reduction = 0.9435270717190128
Iter 10: x = [ 0.71714633 -0.78482668], f(x) = -0.7513643539657264, Grad Norm Reduction = 0.9365008892788087
Iter 11: x = [ 0.71059419 -0.78476759], f(x) = -0.7556415933828987, Grad Norm Reduction = 0.9294860808678659
Iter 12: x = [ 0.70409143 -0.78470816], f(x) = -0.7598545403022542, Grad Norm Reduction = 0.9224837815696675
Iter 13: x = [ 0.69763793 -0.78464841], f(x) = -0.7640038076908072, Grad Norm Reduction = 0.9154950971277833
Iter 14: x = [ 0.6912336  -0.78458834], f(x) = -0.7680900152610703, Grad Norm Reduction = 0.9085211041228451
Iter 15: x = [ 0.68487833 -0.78452796], f(x) = -0.7721137889225084, Grad Norm Reduction = 0.9015628501685319
Iter 16: x = [ 0.678572   -0.78446726], f(x) = -0.7760757602465417, Grad Norm Reduction = 0.8946213541255853
Iter 17: x = [ 0.67231449 -0.78440625], f(x) = -0.7799765659451969, Grad Norm Reduction = 0.8876976063328924
Iter 18: x = [ 0.66610565 -0.78434494], f(x) = -0.7838168473634707, Grad Norm Reduction = 0.8807925688546948
Iter 19: x = [ 0.65994536 -0.78428334], f(x) = -0.78759724998545, Grad Norm Reduction = 0.873907175742985
Iter 20: x = [ 0.65383347 -0.78422144], f(x) = -0.7913184229542104, Grad Norm Reduction = 0.867042333314192
Iter 21: x = [ 0.64776983 -0.78415926], f(x) = -0.7949810186054919, Grad Norm Reduction = 0.8601989204392544
Iter 22: x = [ 0.64175427 -0.78409679], f(x) = -0.7985856920151263, Grad Norm Reduction = 0.8533777888462205
Iter 23: x = [ 0.63578665 -0.78403404], f(x) = -0.8021331005601775, Grad Norm Reduction = 0.8465797634345155
Iter 24: x = [ 0.62986678 -0.78397103], f(x) = -0.8056239034937297, Grad Norm Reduction = 0.8398056426000561
Iter 25: x = [ 0.62399451 -0.78390774], f(x) = -0.8090587615332484, Grad Norm Reduction = 0.833056198570401
/Users/ronald/Desktop/MATH140/HW4/HW4.py:33: RuntimeWarning: invalid value encountered in scalar divide
  grad_norm_reduction = np.linalg.norm(grad) / grad0_norm
Iter 1: x = [0. 0.], f(x) = -1.0, Grad Norm Reduction = nan
Convergence achieved.
Iter 1: x = [6.14500817e-05 7.79216462e-01], f(x) = -0.99696564243836, Grad Norm Reduction = 1.0
Iter 2: x = [-0.00603374  0.00474875], f(x) = -0.9999816842796219, Grad Norm Reduction = 0.011042677518328242
Iter 3: x = [3.70243097e-07 4.70126081e-03], f(x) = -0.9999998894906676, Grad Norm Reduction = 0.008559333360528238
Iter 4: x = [-3.64283336e-05  2.86627036e-05], f(x) = -0.9999999993323806, Grad Norm Reduction = 6.669145573192538e-05
Convergence achieved.
Iter 1: x = [-6.14500817e-05 -7.79216462e-01], f(x) = -0.99696564243836, Grad Norm Reduction = 1.0
Iter 2: x = [ 0.00603374 -0.00474875], f(x) = -0.9999816842796219, Grad Norm Reduction = 0.011042677518328242
Iter 3: x = [-3.70243097e-07 -4.70126081e-03], f(x) = -0.9999998894906676, Grad Norm Reduction = 0.008559333360528238
Iter 4: x = [ 3.64283336e-05 -2.86627036e-05], f(x) = -0.9999999993323806, Grad Norm Reduction = 6.669145573192538e-05
Convergence achieved.
/Users/ronald/Desktop/MATH140/HW4/HW4.py:54: RuntimeWarning: invalid value encountered in scalar divide
  grad_norm_reduction = np.linalg.norm(grad) / grad0_norm
Iter 1: x = [0. 0.], f(x) = -1.0, Grad Norm Reduction = nan
Convergence achieved.
```

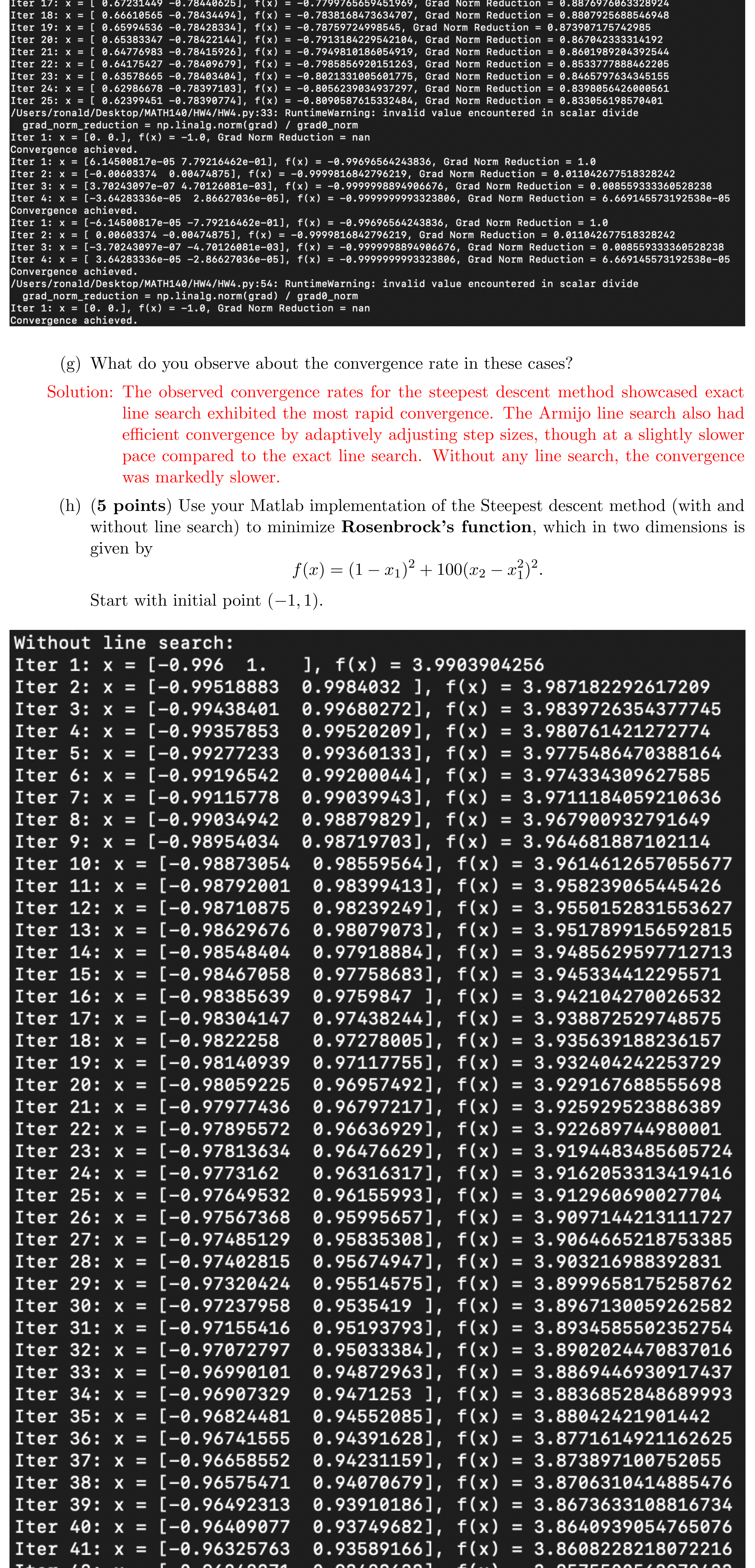
(g) What do you observe about the convergence rate in these cases?

Solution: The observed convergence rates for the steepest descent method showcased exact line search exhibited the most rapid convergence. The Armijo line search also had efficient convergence by adaptively adjusting step sizes, though at a slightly slower pace compared to the exact line search. Without any line search, the convergence was markedly slower.

(h) (**5 points**) Use your Matlab implementation of the Steepest descent method (with and without line search) to minimize **Rosenbrock's function**, which in two dimensions is given by

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2.$$

Start with initial point $(-1, 1)$.



```
Without line search:
Iter 1: x = [-0.996  1.   ], f(x) = 3.9903904256
Iter 2: x = [-0.99518883  0.9984032 ], f(x) = 3.987182292617209
Iter 3: x = [-0.99438401  0.99680272], f(x) = 3.9839726354377745
Iter 4: x = [-0.99357853  0.99520209], f(x) = 3.980761421272774
Iter 5: x = [-0.99277233  0.99360133], f(x) = 3.9775486470388164
Iter 6: x = [-0.99196542  0.99200044], f(x) = 3.974334309627585
Iter 7: x = [-0.99115778  0.99039943], f(x) = 3.9711184059210636
Iter 8: x = [-0.99034942  0.98879829], f(x) = 3.967900932791649
Iter 9: x = [-0.98954034  0.98719703], f(x) = 3.964681887102114
Iter 10: x = [-0.98873054  0.98559564], f(x) = 3.9614612657055677
Iter 11: x = [-0.98792001  0.98399413], f(x) = 3.958239065445426
Iter 12: x = [-0.98710875  0.98239249], f(x) = 3.9550152831553627
Iter 13: x = [-0.98629676  0.98079073], f(x) = 3.9517899156592815
Iter 14: x = [-0.98548404  0.97918884], f(x) = 3.9485629597712713
Iter 15: x = [-0.98467058  0.97758683], f(x) = 3.945334412295571
Iter 16: x = [-0.98385639  0.9759847 ], f(x) = 3.942104270026532
Iter 17: x = [-0.98304147  0.97438244], f(x) = 3.938872529748575
Iter 18: x = [-0.9822258   0.97278005], f(x) = 3.935639188236157
Iter 19: x = [-0.98140939  0.97117755], f(x) = 3.932404242253729
Iter 20: x = [-0.98059225  0.96957492], f(x) = 3.929167688555698
Iter 21: x = [-0.97977436  0.96797217], f(x) = 3.925929523886389
Iter 22: x = [-0.97895572  0.96636929], f(x) = 3.922689744980001
Iter 23: x = [-0.97813634  0.96476629], f(x) = 3.9194483485605724
Iter 24: x = [-0.9773162   0.96316317], f(x) = 3.9162053313419416
Iter 25: x = [-0.97649532  0.96155993], f(x) = 3.912960690027704
Iter 26: x = [-0.97567368  0.95995657], f(x) = 3.9097144213111727
Iter 27: x = [-0.97485129  0.95835308], f(x) = 3.9064665218753385
Iter 28: x = [-0.97402815  0.95674947], f(x) = 3.903216988392831
Iter 29: x = [-0.97320424  0.95514575], f(x) = 3.8999658175258762
Iter 30: x = [-0.97237958  0.9535419 ], f(x) = 3.8967130059262582
Iter 31: x = [-0.97155416  0.95193793], f(x) = 3.893458550235275
Iter 32: x = [-0.97072797  0.95033384], f(x) = 3.8902024470837016
Iter 33: x = [-0.96990101  0.94872963], f(x) = 3.8869446930917437
Iter 34: x = [-0.96907329  0.9471253 ], f(x) = 3.8836852848689993
Iter 35: x = [-0.96824481  0.94552085], f(x) = 3.8804242190144
Iter 36: x = [-0.96741555  0.94391628], f(x) = 3.8771614921162625
Iter 37: x = [-0.96658552  0.94231159], f(x) = 3.873897100752055
Iter 38: x = [-0.96575471  0.94070679], f(x) = 3.8706310414885476
Iter 39: x = [-0.96492313  0.93910186], f(x) = 3.8673633108816734
Iter 40: x = [-0.96409077  0.93749682], f(x) = 3.8640939054765076
Iter 41: x = [-0.96325763  0.93589166], f(x) = 3.8608228218072216
Iter 42: x = [-0.96242371  0.93428638], f(x) = 3.8575500563970433
Iter 43: x = [-0.961589    0.93268098], f(x) = 3.854275605758213
Iter 44: x = [-0.96075351  0.93107547], f(x) = 3.850999466391938
Iter 45: x = [-0.95991724  0.92946984], f(x) = 3.847721634788356
Iter 46: x = [-0.95908017  0.92786409], f(x) = 3.8444421074264823
Iter 47: x = [-0.95824231  0.92625823], f(x) = 3.841160880774175
Iter 48: x = [-0.95740366  0.92465225], f(x) = 3.837877951288086
Iter 49: x = [-0.95656422  0.92304615], f(x) = 3.834593315413618
Iter 50: x = [-0.95572398  0.92143994], f(x) = 3.831306969584884

With Armijo line search:
Iter 1: x = [1. 1.], f(x) = 0.0
Convergence after 2 iterations.
```

Solution: The output indicates that Steepest Descent method without line search is showing very slow convergence towards the minimum of the function. In contrast, the Steepest Descent with Armijo line search shows rapid convergence, achieving the desired tolerance after just two iterations.

(i) (**5 points**) **Tentative Project Plan**:

i. If you decide to work with your peers, please submit the names of your team members. It's perfectly acceptable to work alone on the project as well.

Team Members: Ronald Nap, Cristian Espinosa

ii. (If applicable) Meet with your group initially to start discussing a topic. Submit a tentative title.

Tentative Title: Edge-Aware Image Smoothing via Weighted Least Squares Optimization and Deep Learning

iii. Give a short description of why you are interested on this topic.

Description: My interest in edge promoting stems from enhancing computer vision tasks by preserving edges in image processing.

iv. Include 2-3 references for the topic you chose.

References:

A. "Total Variation Optimization Layers for Computer Vision"
Paper, Code

B. "Fast Global Image Smoothing Based on Weighted Least Squares"
Paper

C. "Deep Edge-Aware Filters"
Paper

v. You can meet with me to discuss the project plan and help further ideas - come prepared to this meeting with some topics you would be interested in. Send an email to set up an appointment.

Preparation: We intend to prepare a detailed project outline, including a literature review and proposed methodology