AE 6310-A: Assignment #2

Karl Roush

Due March 07, 2020

Contents

Problem 1

Part A requires plotting the model function and trust region radius

$$m_x(x_k + p) = f(x_k) + g(x_k)^T p + \frac{1}{2} p^T B_k p$$

$$m(p) = [p_1 \quad p_2] g^T + \frac{1}{2} [p_1 \quad p_2] B_k \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

Part B requires finding the Cauchy step.

$$p = -\tau \frac{\Delta}{\|g\|_2} g \ \ and \ \tau = \begin{cases} 1 & g^T B g \ \leq 0 \\ min(1, ||g||_2{}^3 / \Delta g^T B g) & otherwise \end{cases}$$

Part C requires finding the exact trust region step and Lagrange multiplier for the exact solution. This means finding the eigenvalues via

$$det(B - \mu I) = 0$$

then solving

$$(B + \lambda I)p = -g \ must \ satisfy \ \ \lambda \geq +\mu$$

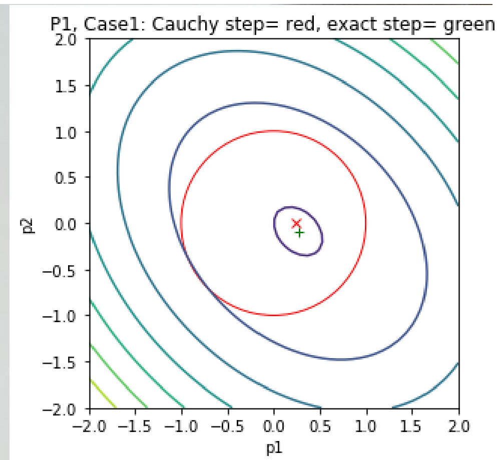
Furthermore, recall the following

$$\|p\|_2{}^2 = p_1{}^2 + p_2{}^2 = \Delta^2$$

Part D requires indicating whether the exact step is on the trust region boundary. This corresponds to if the trust region constraint is active. If the magnitude of p is greater than the trust region radius, it is infeasible for this step.

$$\|p\| = \begin{cases} < \Delta & within \ trust \ region \\ = \Delta & on \ trust \ region \ boundary \end{cases}$$

Case 1



Case 1

$$m(p) = [p_1 \ \ p_2]\begin{bmatrix} \frac{-1}{\sqrt{2}} \\ 0 \end{bmatrix} + \frac{1}{2}[p_1 \ \ p_2]\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

find eigen values of $B - \mu I$

$$\rightarrow \begin{bmatrix} 3-\mu & 1 \\ 1 & 3-\mu \end{bmatrix} \Rightarrow \mu = 2, 4$$

Cauchy step:

$$p^c = -\alpha g = \begin{bmatrix} \frac{\alpha}{\sqrt{2}} \\ 0 \end{bmatrix} \Rightarrow m(-\alpha g) = [\alpha \ \ \alpha]\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix} + \frac{1}{2}[\alpha \ \ \alpha]\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} \alpha \\ \alpha \end{bmatrix}$$

$$p^c = \begin{bmatrix} -1/16 \\ 0 \end{bmatrix} \leftarrow \quad = 4\alpha^2 + \frac{\alpha\sqrt{2}}{2} \rightarrow \min @ \alpha = \frac{-\sqrt{2}}{16}$$

alternatively consider

$$p = -\tau \frac{\Delta}{\|g\|_2} g \qquad \tau = \begin{cases} 1 & g^T B g \le 0 \\ \min\left(1, \frac{\|g\|_2^3}{\Delta g^T B g}\right) & \text{else} \end{cases}$$

check
$$g^T B g = [-1/\sqrt{2} \ \ 0]\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix} = 3/2 \rightarrow \frac{\|g\|_2^3}{\Delta g^T B g} = \frac{(\sqrt{2}/2)^3}{1(3/2)} = \frac{\sqrt{2}}{6}$$

$$p = -\frac{\sqrt{2}}{6}\frac{1}{\sqrt{2}/2}\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix} \Rightarrow p^c = \begin{bmatrix} \sqrt{2}/6 \\ 0 \end{bmatrix} \Rightarrow p^c = \begin{bmatrix} 0.2357 \\ 0 \end{bmatrix}$$

exact step:

$$(B + \lambda I)p = -g$$

$$\begin{bmatrix} 3+\lambda & 1 \\ 1 & 3+\lambda \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = -\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \frac{1}{(3+\lambda)^2 - 1}\begin{bmatrix} 3+\lambda & -1 \\ -1 & 3+\lambda \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} \\ 0 \end{bmatrix} \frac{1}{\lambda^2+6\lambda+8}\begin{bmatrix} 3+\lambda & -1 \\ -1 & 3+\lambda \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\Rightarrow \frac{1}{(\lambda+4)(\lambda+2)}\begin{bmatrix} \frac{\lambda+3\sqrt{2}}{2} \\ -\sqrt{2}/2 \end{bmatrix}$$

$$\left[\frac{1}{(\lambda+4)(\lambda+2)}\right]^2\left(\left(\frac{\lambda+3\sqrt{2}}{2}\right)^2 + \left(\frac{-\sqrt{2}}{2}\right)^2\right) = 1^2 \rightarrow \frac{\lambda^2+6\lambda+10}{2(\lambda+2)^2(\lambda+4)^2} = 1^2 \rightarrow \lambda = -4.51, -3.47, -2.53, -1.49$$

need $\lambda > -\lambda_1 \rightarrow$ neither satisfy

now check $p = -B^{-1}g$
$$= \begin{bmatrix} \frac{3\sqrt{2}}{16} \\ \frac{-\sqrt{2}}{16} \end{bmatrix} \rightarrow \|p\|_2 = 0.279508 \rightarrow \lambda = 0$$

$$p^* = \begin{bmatrix} \frac{3\sqrt{2}}{16} \\ \frac{-\sqrt{2}}{16} \end{bmatrix} \approx \begin{bmatrix} 0.265 \\ -0.088 \end{bmatrix}$$

$\|p^*\|^2 = 0.078125$

$\|p^*\|^2 < \Delta^*$ so not on boundary

Case 2

Case 2

$\Delta = 1/2$

$$m(p) = [p_1 \ p_2]\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix} + \frac{1}{2}[p_1 \ p_2]\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

from case 1: $m = 2,4$

<u>Cauchy step</u>: same as case 1 $\rightarrow p^c = \begin{bmatrix} 0.2357 \\ 0 \end{bmatrix}$

$g^T B g = \sqrt{2}/6$ and $\sqrt{2}/6 < 1/2$ $\rightarrow \tau = \min\left(1, \frac{(\sqrt{2}/2)^3}{\frac{1}{2}(3/2)}\right) = 0.4714$

$\Downarrow$

so <u>exact step</u> is also same

$p = -0.4714 \frac{0.5}{\sqrt{2}/2}\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix}$

$\lambda = 0$

$p^* = \begin{bmatrix} 0.2652 \\ -0.0884 \end{bmatrix} \Rightarrow$ not on boundary

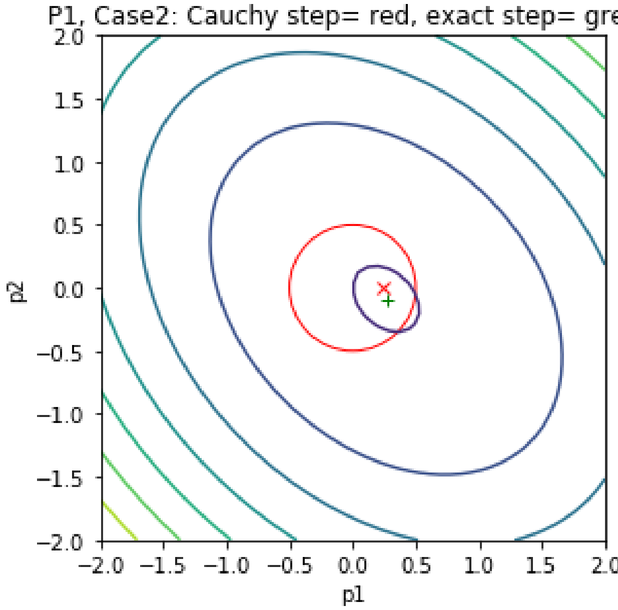
$p^c = \begin{bmatrix} 0.2357 \\ 0 \end{bmatrix}$

<u>if we instead consider $\Delta = 1/4$</u>

$\tau = \min\left(1, \frac{(\sqrt{2}/2)^3}{\frac{1}{4}(3/2)}\right) \simeq 0.9428 \rightarrow p = -0.9428 \frac{0.25}{\sqrt{2}/2}\begin{bmatrix} -1/\sqrt{2} \\ 0 \end{bmatrix} \Rightarrow p^c = \begin{bmatrix} 0.2357 \\ 0 \end{bmatrix}$

check $p = -B^{-1}g = \begin{bmatrix} \frac{3\sqrt{2}}{16} \\ \frac{-\sqrt{2}}{16} \end{bmatrix} \rightarrow$ same $p^* \Rightarrow$ $\lambda = 0$ $p^* = \begin{bmatrix} 0.2652 \\ -0.0884 \end{bmatrix}$ not on boundary (same as case 1)

P1, Case2: Cauchy step= red, exact step= green



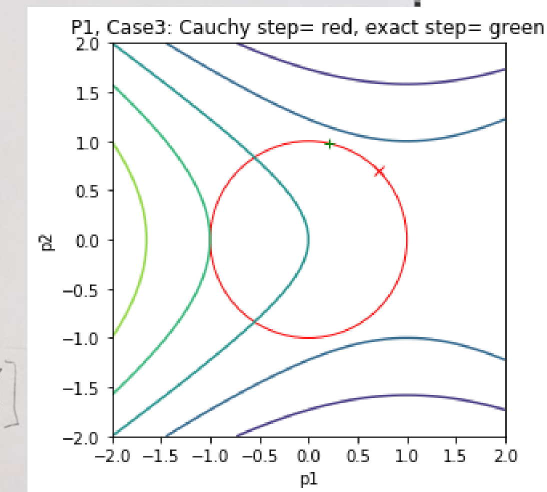
Case 3

Case 3

$m(p) = [p_1 \ \ p_2]\begin{bmatrix}-1\\-1\end{bmatrix} + \frac{1}{2}[p_1 \ \ p_2]\begin{bmatrix}1 & 0\\0 & -2\end{bmatrix}\begin{bmatrix}p_1\\p_2\end{bmatrix}$ $\quad \Delta = 1$

find eigenvalues of $B - \mu I$

$\begin{bmatrix}1-\mu & 0\\0 & -2-\mu\end{bmatrix} \Rightarrow \mu = -2, 1$



Cauchy step:

check $g^T B g = [-1 \ -1]\begin{bmatrix}1 & 0\\0 & -2\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} = -1 \le 0 \rightarrow \tau = 1$

$p^c = -1 \frac{\Delta}{\|g\|_2} g = -\frac{1}{\sqrt{2}}\begin{bmatrix}-1\\-1\end{bmatrix} \rightarrow p^c = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \approx \begin{bmatrix}0.707\\0.707\end{bmatrix}$

exact step:

check $p = -B^{-1}g = -\begin{bmatrix}1 & 0\\0 & -2\end{bmatrix}^{-1} g = \begin{bmatrix}1\\-1/2\end{bmatrix}$ $\quad \|p\|_2 = \frac{\sqrt{5}}{2} \ge \Delta$ $\quad (1.1180 > 1)$

now → solve $\|p(\lambda)\|_2^2 - \Delta^2 = 0$

$(B + \lambda I)p = -g$

$\begin{bmatrix}1+\lambda & 0\\0 & -2+\lambda\end{bmatrix}\begin{bmatrix}p_1\\p_2\end{bmatrix} = -\begin{bmatrix}-1\\-1\end{bmatrix} \Rightarrow \begin{bmatrix}p_1\\p_2\end{bmatrix} = \frac{1}{(1+\lambda)(-2+\lambda)}\begin{bmatrix}-2+\lambda & 0\\0 & 1+\lambda\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}$

$\|p\|_2^2 = \frac{1}{((1+\lambda)(-2+\lambda))^2}\left((\lambda-2)^2 + (\lambda+1)^2\right) \leftarrow = \frac{1}{\lambda^2-\lambda-2}\begin{bmatrix}\lambda-2\\\lambda+1\end{bmatrix}$

$= \frac{2\lambda^2 - 2\lambda + 5}{(\lambda-2)^2(\lambda+1)^2} \longrightarrow$ solve $\|p\|_2^2 - \Delta^2 = 0$ $\quad \frac{2\lambda^2-2\lambda+5}{(\lambda-2)^2(\lambda+1)^2} - 1^2 = 0$

$\lambda = -0.0743, 1.165, 2.835, 4.074$

we need $\lambda \ge 0$ & $\lambda > -\lambda_1$ $\quad \lambda_1 = -2$

↓

$\lambda = 2.835 \Rightarrow p^* = \begin{bmatrix}0.261\\1.198\end{bmatrix} \Rightarrow$ need to normalize by Δ $\Rightarrow p^* = \frac{\Delta p^*}{\|p\|} = \begin{bmatrix}0.2127\\0.9771\end{bmatrix}$

$\Rightarrow \|p^*\| = \Delta$ so on trust region boundary

Problem 2

The trust subroutines are located in a separate section after the answers below.

Part A: Polynomial coefficients

$$\|p^*(\lambda)\|_2{}^2 = \sum_{i=1}^{n} \frac{(q_i{}^T g)^2}{(\lambda_i + \lambda)^2} \leq \Delta^2$$

For the case of n=2

$$0 = \sum_{i=1}^{n} \frac{(q_i{}^T g)^2}{(\lambda_i + \lambda)^2} - \Delta^2$$

$$0 = \frac{(q_1{}^T g)^2}{(\lambda_1 + \lambda)^2} + \frac{(q_2{}^T g)^2}{(\lambda_2 + \lambda)^2} - \Delta^2$$

The numerator terms reduce to constants, so the bottom terms need to be expanded.

$$0 = \frac{(q_1{}^T g)^2}{\lambda^2 + 2\lambda\lambda_1 + \lambda_1{}^2} + \frac{(q_2{}^T g)^2}{\lambda^2 + 2\lambda\lambda_2 + \lambda_2{}^2} - \Delta^2$$

Re-arranging:

$$\frac{1}{\Delta^2} = \frac{\lambda^2 + 2\lambda\lambda_1 + \lambda_1{}^2}{(q_1{}^T g)^2} + \frac{\lambda^2 + 2\lambda\lambda_2 + \lambda_2{}^2}{(q_2{}^T g)^2}$$

Since λ_1 and λ_2 are constants:

$$\frac{1}{\Delta^2} = (q_1{}^T g)^{-2}(\lambda^2 + 2\lambda\lambda_1 + \lambda_1{}^2) + (q_2{}^T g)^{-2}(\lambda^2 + 2\lambda\lambda_2 + \lambda_2{}^2)$$

$$\frac{1}{\Delta^2} = ((q_1{}^T g)^{-2} + (q_2{}^T g)^{-2})\lambda^2 + (2(q_1{}^T g)^{-2}\lambda_1 + 2(q_2{}^T g)^{-2}\lambda_2)\lambda + (q_1{}^T g)^{-2}\lambda_1{}^2 + (q_2{}^T g)^{-2}\lambda_2{}^2$$

Further re-arranging to use numpy.roots:

$$0 = [(q_1{}^T g)^{-2} + (q_2{}^T g)^{-2}]\lambda^2 + [2(q_1{}^T g)^{-2}\lambda_1 + 2(q_2{}^T g)^{-2}\lambda_2]\lambda + \left[(q_1{}^T g)^{-2}\lambda_1{}^2 + (q_2{}^T g)^{-2}\lambda_2{}^2 - \frac{1}{\Delta^2}\right]$$

Note that is it useful to avoid the possible divide by zero with the negative exponents, though this requires using sympy.solve instead:

$$0 = (q_1{}^T g)^2(\lambda^2 + 2\lambda\lambda_1 + \lambda_1{}^2)^{-1} + (q_2{}^T g)^2(\lambda^2 + 2\lambda\lambda_2 + \lambda_2{}^2)^{-1} - \Delta^2$$

Part B: Verify that your code

The output of my code is below, matching the results from the Question 1.

```
Cauchy step, case 1=
 [[ 0.23570226]
 [-0.        ]]
B is postive definite, and ||p||2 <=delta
Exact step, case 1=
 [[ 0.26516504]
 [-0.08838835]]
Lagrange multiplier, case1=  0
onBound=  False


Cauchy step, case 2=
 [[ 0.23570226]
 [-0.        ]]
B is postive definite, and ||p||2 <=delta
Exact step, case 2=
 [[ 0.26516504]
 [-0.08838835]]
Lagrange multiplier, case2=  0
onBound=  False


Cauchy step, case 3=
 [[0.70710678]
 [0.70710678]]
B is not postive definite, or ||p||2 >delta
Exact step, case 3=
 [[0.212746894185773]
 [0.977107342626340]]
Lagrange multiplier, case3=  2.83499961812447
onBound=  True
```

Part C: Create your own model functions

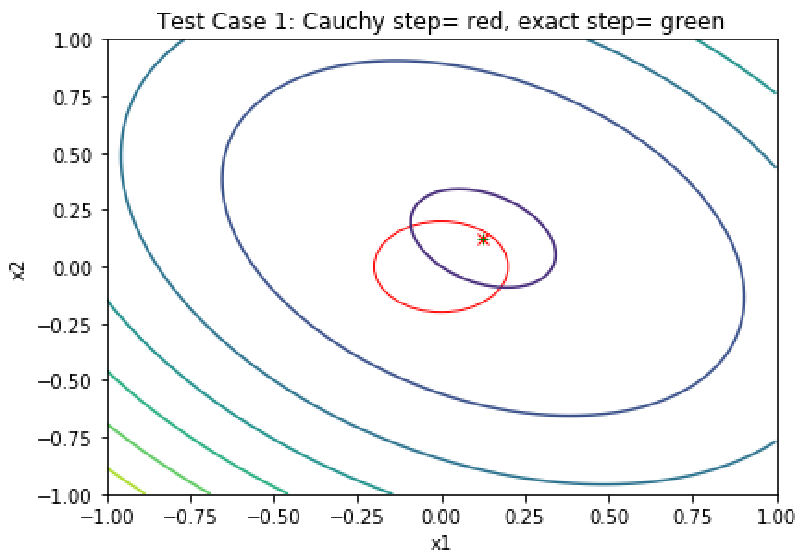
Case 1: Minimizer in the interior of the trust region

$f(x_1,x_2) = 3{x_1}^2 + 2x_1x_2 + 3{x_2}^2 - x_1 - x_2$

$$\nabla f = \begin{bmatrix} 6x_1 + 2x_2 - 1 \\ 2x_1 + 6x_2 - 1 \end{bmatrix} \text{ and the Hessian } H(x) = \begin{bmatrix} 6 & 2 \\ 2 & 6 \end{bmatrix}$$

$$\boxed{x^* = \begin{bmatrix} 1/8 \\ 1/8 \end{bmatrix}}$$

The trust region radius was selected to be delta= 0.2 around [0,0]



Case 2: Positive definite model, but minimizer is constrained

$f(x_1,x_2) = {x_1}^2 + x_1x_2 + {x_2}^2 - x_1 - x_2$

$$\text{t } \nabla f = \begin{bmatrix} 2x_1 + x_2 - 1 \\ x_1 + 2x_2 - 1 \end{bmatrix} \text{ and the Hessian } H(x) = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\boxed{x^* = \begin{bmatrix} 1/3 \\ 1/3 \end{bmatrix}}$$

The trust region radius was selected to be delta= 0.2 around [0,0]

Case 3: Indefinite/negative definite Hessian

$f(x_1, x_2) = 4{x_1}^2 + 6x_1x_2 + 2{x_2}^2 + x_1 + x_2$

saddle point at $\begin{bmatrix} -1/2 \\ 1/2 \end{bmatrix}$

$\nabla f = \begin{bmatrix} 8x_1 + 6x_2 + 1 \\ 6x_1 + 4x_2 + 1 \end{bmatrix}$ and the Hessian $H(x) = \begin{bmatrix} 8 & 6 \\ 6 & 4 \end{bmatrix}$

The trust region radius was selected to be delta= 0.2 around [0,0]

Problem 2 Code

```python
def cauchy_step(g,B,delta):
    if np.transpose(g).dot(B.dot(g)) <= 0:
        tau= 1
    else:
        temp= (np.linalg.norm(g,ord=2)**3)/(delta*(np.transpose(g)).dot(B.dot(g)))
        tau= min(1, temp)
    p= -tau*(delta/np.linalg.norm(g,ord=2))*g
    return p
```

```python
def trust_region_step(g,B,delta):
    onBound= False
    #convert from sympy to numeric values
    a2= np.float64(B[0][0])
    b2= np.float64(B[0][1])
    c2= np.float64(B[1][0])
    d2= np.float64(B[1][1])
    B=np.array([[a2,b2],[c2,d2]])

    lam, Q = np.linalg.eigh(B) #find eigenvalues and eigenvectors of B
    VV= np.diag([lam[0],lam[1]])
    p= -1* (np.linalg.inv(B)).dot(g)
    p_mag= (p[0,0]**2 + p[1,0]**2)**0.5 #linalg norm doesnt work due to sympy/numpy conflict
    if lam[0] >0 and p_mag <= delta: #smallest eigenvalue >0 = postive definite B
        #print('B is postive definite, and ||p||2 <=delta')
        lagrange_mult= 0
        p_star= p
    else:
        #print('B is not postive definite, or ||p||2 >delta')
        #Q[:,1] is second eigenvector
        q1= Q[:,0]
        q2= Q[:,1]

        L= smp.symbols('L',real=True)
        coeff1= (np.dot(q1,g)**2)*(L**2 +2*L*lam[0]+lam[0]**2)**(-1)
        coeff2= (np.dot(q2,g)**2)*(L**2 +2*L*lam[0]+lam[1]**2)**(-1)
        solution= smp.solve(coeff1+coeff2-delta**2, L)
        lams=[]
        # print(lam)
        lagrange_mult= []
        for i in solution:
            lams.append(i[0])
        # print(lams)
        for i in lams:
            if i>0 and i>-(lam[0]):
                lagrange_mult.append(i)
        lagrange_mult= lagrange_mult[0]

        A= np.array(VV+ lagrange_mult*np.identity(2))
        A= smp.Matrix(A).inv()
        A= np.array(A)
        p_star= np.dot(-Q, np.dot(A, np.dot(np.transpose(Q),g)))
        p_star_mag= (p_star[0,0]**2 + p_star[1,0]**2)**0.5
        # print(p_star)
        if p_star_mag >= delta: #if exact step is outside trust region
            p_star[0,0]=(p_star[0,0]*delta)/p_star_mag
            p_star[1,0]=(p_star[1,0]*delta)/p_star_mag
            onBound= True
    return p_star, lagrange_mult, onBound
```

Problem 3

The trust region algorithm and its components are located in a separate section after the answers below

Part A: Applying the algorithm

In order to apply the trust region algorithm, we need a general form of the gradient

$$f1(x_1, x_2) = -10x_1^{\,2} + 10x_2^{\,2} + 4\sin(x_1 x_2) - 2x_1 + x_1^{\,4}$$

$$\nabla f1(x_1, x_2) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -20x_1 + 4x_2\cos(x_1 x_2) - 2 + 4x_1^{\,3} \\ 20x_2 + 4x_1\cos(x_1 x_2) \end{bmatrix}$$

Likewise, for the other function:

$$f2(x_1, x_2) = -100(x_2 - x_1^{\,2})^2 + (1 - x_1)^2$$

$$\nabla f2(x_1, x_2) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -400x_1^{\,3} + 2x_1(200x_2 + 1) - 2 \\ 200x_1^{\,2} - 200x_2 \end{bmatrix}$$

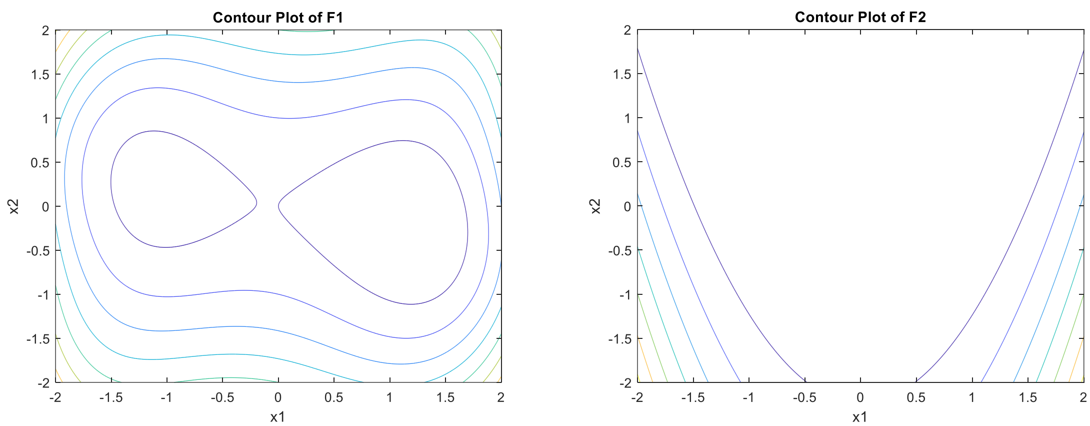


Figure 1: Contour plots of f1 and f2 respectively

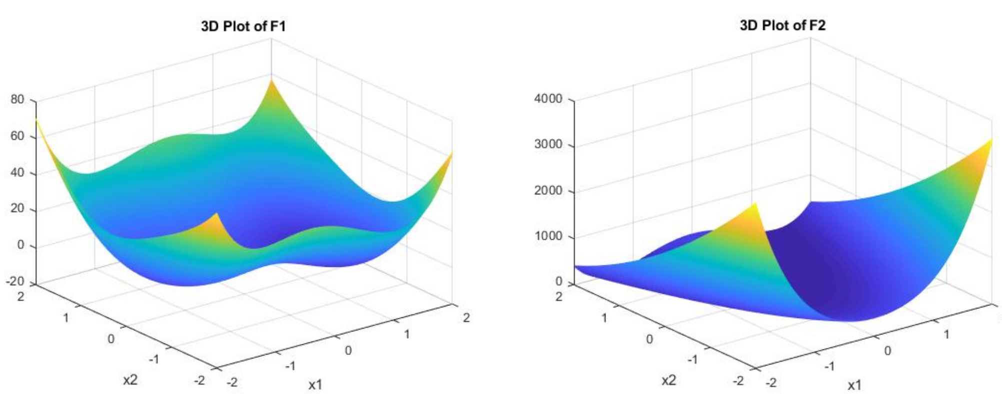


Figure 2: 3D plots of f1 and f2 respectively

Part B: Compare Cauchy vs. exact trust region step

The Cauchy point essentially functions like a steepest decent method. In contrast, the exact trust region step is the solution to the constrained minimization problem.

Note that the code can be run where it breaks out upon encountering a negative rho_k (convergence ratio). This is indicated in the table below with parenthesis. Otherwise, the algorithm breaks out upon encouraging a NaN, divide by zero, or complex value for `rho_k`.

Table 1: Evaluation of trust region methods (tol= 1e-9, delta= 1, delta_max= 2, eta= 0.125)

Starting point [x1,x2]	Function	Cauchy step function evals	Exact trust region step function evals
[0,0]	Function 1	5 (32)	4 (31)
	Function 2	1 (33)	1 (31)
[0,1]	Function 1	6 (32)	22 (48)
	Function 2	1 (29)	1 (29)
[0,-1]	Function 1	5 (31)	5 (30)
	Function 2	1 (29)	1 (29)
[-1,0]	Function 1	4 (30)	5 (54)
	Function 2	1 (29)	1 (29)
[-2,-2]	Function 1	14 (39)	3 (30)
	Function 2	1 (28)	1 (28)

In most of the test cases, the performance of the two models is very close. However, in cases where the starting point is far from the minimum, the exact trust region converges much faster (like in the case of [-2,-2]). Interestingly, around "flatter" regions like a saddle point, the Cauchy step method steepest converges faster because the saddle point is right on the opposite direction of the starting point's gradient.

Part C: Performance at different starting points

This question was partially answered above, referring to the data in **Table 1**. Essentially, the exact trust region step performs better than the Cauchy step at distances farther from the min. However, the Cauchy step converges faster under starting conditions near "flatter" regions.

Both algorithms were rather sensitive to the starting point in reference to the location of the min. For example, function 1 at [5,5] caused both variants to converge around [3.97, 4.68] which is obviously not the minimum. This could be tweaked through changing `delta_max` and/or `eta`, but not enough to shift the result to the min.

Additionally, both algorithms struggled in regions of low gradient values (i.e. in the Rosenbrock function). This is because the difference in the gradient at x_k vs x_{k+1}, or y_k, was not very large. This in turn, did not significantly update B so the algorithm did not move too much from point to point.

Problem 3: Code

```
def trust_algo(desc,fobj, x0,cauchy, tol, delta, delta_max, eta, breakout):
    # x0= np.array([[0,0]])
    f_x, g_x = fobj(x0)
    f_xk= None
    g_xk= None
    b= np.identity(2)
    k=1
    norm2_gk= (g_x[0][0]**2 + g_x[1][0]**2)**0.5
    xk= x0 #if you start at a point closer to the min than the tolerance
    # while k<3:
    while norm2_gk > tol:
        #find the amount to move to
        if cauchy:
            pk= cauchy_step(g_x,b,delta)
        else:
            pk, lagrange_mult, onBound = trust_region_step(g_x,b,delta)
        xk= x0+pk #move to new point
        # f_x, g_x = fobj(x0) #function value and grad at OLD point
        f_xk, g_xk = fobj(xk) #function value and grad at NEW point
        m_x= p3_model(x0,g_x,b) #model value at OLD point
        m_xk= p3_model(xk,g_xk,b) #model value at NEW point

        rho_k= (f_x - f_xk)/((m_x-m_xk)[0][0]) #convergance parameter
        # print(rho_k)
        if breakout:
            if rho_k <0:
                break
        if str(rho_k)=='zoo' or isinstance(rho_k, complex) or math.isnan(rho_k):
            break

        yk= g_xk - g_x #yk= grad@newPoint - grad@oldPoint
        sk= pk
        #update B using SR-1 formula
        b= b+ (np.dot((yk- np.dot(b,sk)), np.transpose(yk-np.dot(b,sk)))/ (np.dot(np.transpose(sk),(yk- np.dot(b,sk)))))
        x0= set_new_point(rho_k,eta, x0, xk) #set the new x0
        delta= set_new_radius(rho_k, delta, pk, delta_max)

        f_x= f_xk
        g_x=g_xk
        norm2_gk= (g_x[0][0]**2 + g_x[1][0]**2)**0.5
        k+=1 #update iterations

    print(desc)
    print('Iterations=',k)
    print('Min at:', xk,'\n')
```

```
def p3_model(p,g,b):
    #p=[p1;p2]
    return np.dot(np.transpose(p),g) +0.5*(np.dot(np.transpose(p),np.dot(b,p)))

def set_new_point(rho_k,eta, x0, xk):
    if rho_k >= eta:
        x0=xk
    else:
        x0=x0
    return x0

def set_new_radius(rho_k, delta, pk, delta_max):
    pk_norm= (pk[0][0]**2 + pk[1][0]**2)**0.5 #that damn numpy/sympy conversion; use lambdify
    if rho_k <0.25:
        delta= 0.25*delta
    # elif (rho_k > 0.75) and (np.linalg.norm(pk,2)== delta):
    elif (rho_k > 0.75) and (pk_norm== delta):
        delta= min(2*delta, delta_max)
    else:
        delta= delta
    return delta
```

```
def problem3(x0, tol, delta, delta_max, eta):
    breakout= True #change this to modify breakout behavior
    print('Starting point=', x0)
    print('Tolerance=',tol ,'\n')
    trust_algo('f1, Cauchy Step',p3_f1,x0,True,tol, delta, delta_max, eta, breakout) #Function 1, cauchy; works
    trust_algo('f1, Exact Step', p3_f1,x0,False,tol, delta, delta_max, eta, breakout) #Function 1, exact step
    trust_algo('f2, Cauchy Step',p3_f2,x0,True,tol, delta, delta_max, eta, breakout) #Function 2, cauchy
    trust_algo('f2, Exact Step', p3_f2,x0,False,tol, delta, delta_max, eta, breakout) #Function 2, exact step
```