3X03
Assignment 2

Johnson Ji

400499564

P1:

Finished in Julia file.

P2:

P2:

$$\|v\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = 1$$

$V \triangleq v v^T = \|v\|_2^2$ and $V \in$ symmetric matrix

Note: $(vv^T)^T = vv^T$ $\therefore A = Q \Lambda Q^{-1}$, $Q \in$ Orthogonal matrix

$Vv = \lambda v \iff (vv^T)v = \lambda v$

$LHS = v(v^T v) = v$

$\therefore \lambda = 1$ is an eigenvalue of v, v is an eigenvector

Note, Since for all column vectors of Q, are the eigenvec of the matrix A and they are orthogonal.

Since v is one of the eigenvectors, so other eigenvetors u should $\perp$ with v

$\therefore (vv^T)u = \lambda u$

$v(v^T u) = \lambda u$, $v^T u = 0$.

$\therefore$ for any $\vec{u} \neq \vec{0}$, $\lambda = 0$.

$\therefore$ For all eigenvectors of A and their eigenvalues:

- E.V = V, E.value = 1
- E.V = u that $uv^T = 0$, E.value = 0

P3:

Finished in Julia.

P4:

P4:

$$F(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix} \quad \text{while } f(x) = \|x - p\| - d$$

Note: $$J(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \vdots \end{bmatrix} \quad \text{while } f_i(x) = \left(\sum_{k=1}^{n} (x_k - p_{ik})^2\right)^{\frac{1}{2}} - d_i$$

$$\nabla f_i(x) = \begin{bmatrix} \frac{\partial f_i}{\partial x_1} \\ \vdots \\ \frac{\partial f_i}{\partial x_n} \end{bmatrix}, \quad \frac{\partial f_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\sum_{k=1}^{n} (x_k - p_{ik})^2\right)^{\frac{1}{2}} - d_i\right]$$

$$= \frac{1}{2}\left(\sum_{1}^{n} (x_k - p_{ik})^2\right)^{-\frac{1}{2}} \cdot \frac{\partial}{\partial x_j}\left(\sum_{1}^{n} (x_k - p_{ik})^2\right)$$

$$= \frac{1}{2}\left(\sum_{1}^{n} (x_k - p_{ik})^2\right)^{-\frac{1}{2}} \cdot 2(x_j - p_{ij})$$

$$= \frac{1}{2} \frac{2}{\|x - p_i\|} \cdot 2(x_j - p_{ij})$$

$$= \frac{x_j - p_{ij}}{\|x - p_i\|} \rightarrow 2 \text{ norm}$$

$$\therefore \nabla f_i(x) = \frac{x - p_i}{\|x - p_i\|_2}$$

$$\therefore J(x) = \begin{bmatrix} \frac{(x - p_1)^T}{\|x - p_1\|_2} \\ \vdots \\ \frac{(x - p_1)^T}{\|x - p_n\|_2} \end{bmatrix} \text{ until } n.$$

P5:

Finished in Julia.

P6:

P6:

$$f(x) \triangleq \sum_{i=1}^{m} |f_i(x)|^2 = \sum_{i=1}^{m} (f_i(x))^2, \quad f_i(x) = \|x - p_i\|_2 - d_i$$

$$\nabla f(x) = \nabla \left(\sum_{1}^{m} f_i(x)^2 \right)$$

$$= \sum_{1}^{m} \nabla f_i(x)^2 = 2 \sum_{1}^{m} \nabla f_i(x) f_i(x)$$

$$= 2 \sum_{1}^{m} \left((\|x - p_i\|_2 - d_i) \cdot \frac{x - p_i}{\|x - p_i\|_2} \right)$$

$$\nabla^2 f(x) = J(\nabla f)$$

$$= 2 \sum_{1}^{m} J(f_i(x) \nabla f_i(x))$$

$$= 2 \sum_{1}^{m} \left[\nabla f_i(x) (\nabla f_i(x))^T + f_i(x) J(\nabla f_i(x)) \right]$$

$$= 2 \sum_{1}^{m} \left[\nabla f_i(x) (\nabla f_i(x))^T + f_i(x) + H_i(f) \right]$$

$$H_i(f) = J(\nabla f_i) = J\left(\frac{x - p_i}{\|x - p_i\|_2} \right)$$

$$= \frac{1 \cdot \|x - p_i\|_2 - (x - p_i) \left(\frac{x - p_i}{\|x - p_i\|_2} \right)^T}{\|x - p_i\|_2^2}$$

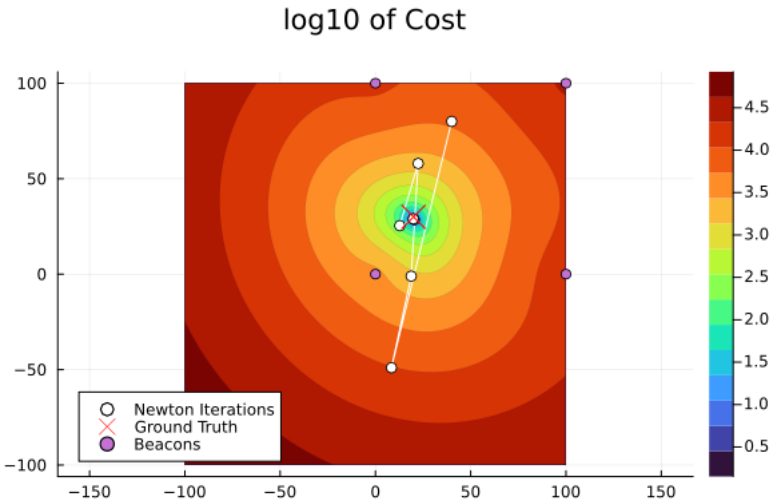
$$\therefore \nabla^2 f(x) = 2 \sum_{1}^{m} \left[\frac{(x - p_i)(x - p_i)^T}{\|x - p_i\|_2^2} + (\|x - p_i\|_2 - d_i) \cdot \left(\frac{I}{\|x - p_i\|_2} - \frac{(x - p_i)(x - p_i)^T}{\|x - p_i\|_2^3} \right) \right]$$

P7:

The function newton_optimizer() is finished in Julia.

Note: the definition of the local min should satisfy: gradient is 0 and Hessian matrix is PD,

(a)



In order to give the numerical evidence, I add these code lines under the test file:

```
# Test out the Newton optimizer (Problem 7)
println("\n--- Problem 7 Test ---")
x_gt = rand(3) * 10 .- 5.0
x0 = x_gt + 0.5 * rand(3)
P = rand(3, 3) * 10.0 .- 5.0
d = [norm(x_gt - P[:, i]) for i in 1:size(P, 2)]
x_trace = newton_optimizer(x0, P, d, newton_tol, 100)

n = length(x0)
gradient_f(xc) = begin
    g = zeros(n)
    m = size(P, 2)
    @inbounds for i in 1:m
        dx = xc .- P[:, i]
        nd = norm(dx)
        if nd > 1e-12
            g .+= 2 * ((nd - d[i]) * (dx / nd))
        end
    end
    g
end
```

```
hessian_f(xc) = begin
    H = zeros(n, n)
    m = size(P, 2)
    I_n = Matrix{Float64}(I, n, n)
    @inbounds for i in 1:m
        dx = xc .- P[:, i]
        nd = norm(dx)
        if nd > 1e-12
            term1 = (dx * dx') / (nd^2)
            term2 = (nd - d[i]) * ((I_n / nd) - (dx * dx') / (nd^3))
            H .+= 2 * (term1 + term2)
        end
    end
    H
end

g_final = gradient_f(x_trace[end])
println("Final gradient norm ||∇f(x)|| = ", norm(g_final))
println("Final gradient value = ", g_final)

H_final = hessian_f(x_trace[end])
eigvals_H = eigvals(H_final)
println("Eigenvalues of final Hessian H(x̂):")
println(eigvals_H)
println("Smallest eigenvalue λ_min = ", minimum(eigvals_H))
println("Largest eigenvalue λ_max = ", maximum(eigvals_H))
```

For a, I print the norm of the gradient to prove the point is a critical point, notice that the result is 2.56e-7, which is very closed to 0. So this is a local min.

```
--- Problem 7 Test ---
Final gradient norm ||∇f(x)|| = 2.5633010779930474e-7
Final gradient value = [1.4093543772355096e-7, 9.193227281011238e-8, -1.933669666050573e-7]
Eigenvalues of final Hessian H(x̂):
[0.004728943215221599, 1.6134113214303103, 4.38185986648365]
Smallest eigenvalue λ_min = 0.004728943215221599
Largest eigenvalue λ_max = 4.38185986648365
jihaoran@SdKfz-Marx 3X03_A2 %
```

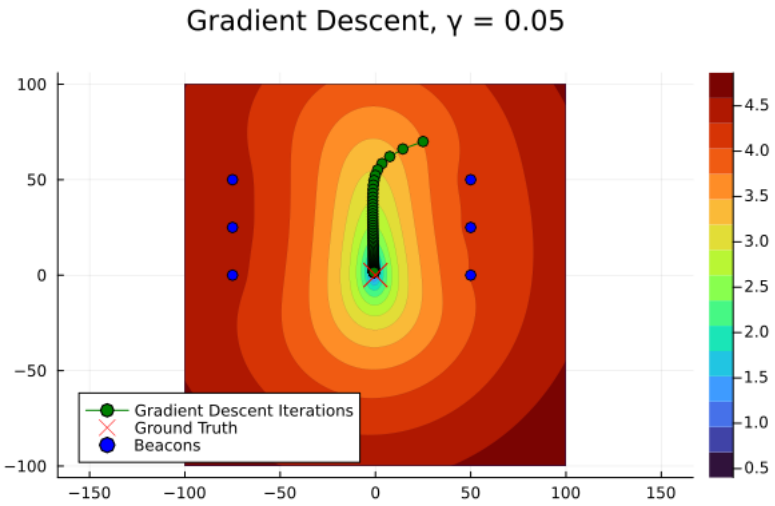
Git Graph Julia env: v1.11 Not Committed Yet Ln 131, Col 59 (1275 selected) Spaces: 4 UTF-8

(b)

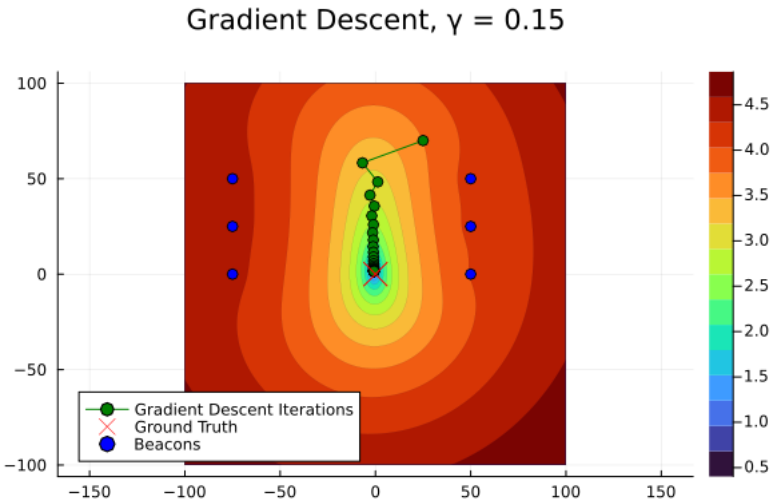
The second derivative for a multivariable function is Hessian matrix, so we measure all of the eigenvalue of the Hessian matrix, we find all of the eigenvalues are positive so which means the local behavior near this critical point is concave up so this point is local min (since H is positive definite).

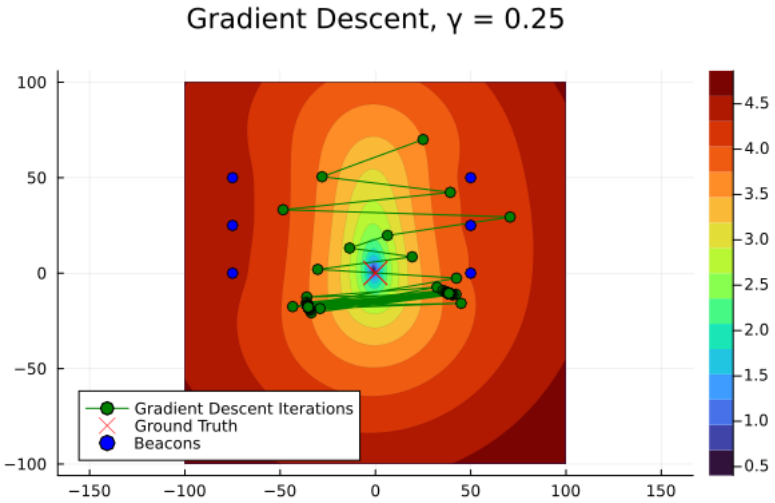
P8:

(a) Finished in Julia.



(b)



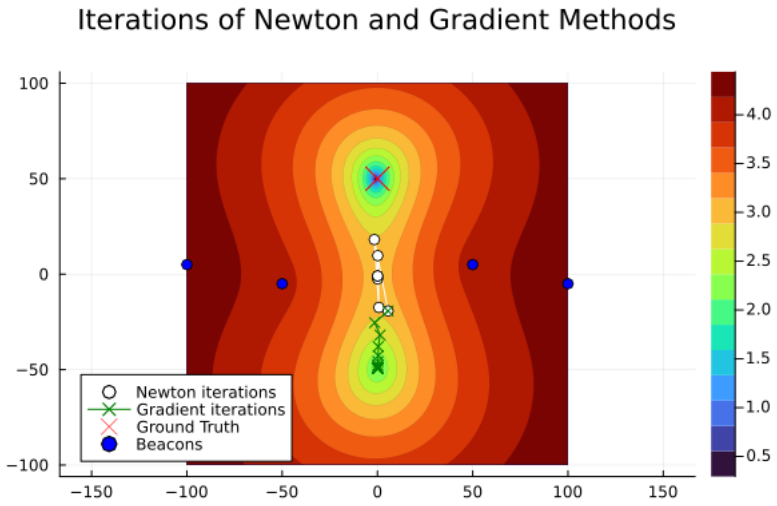


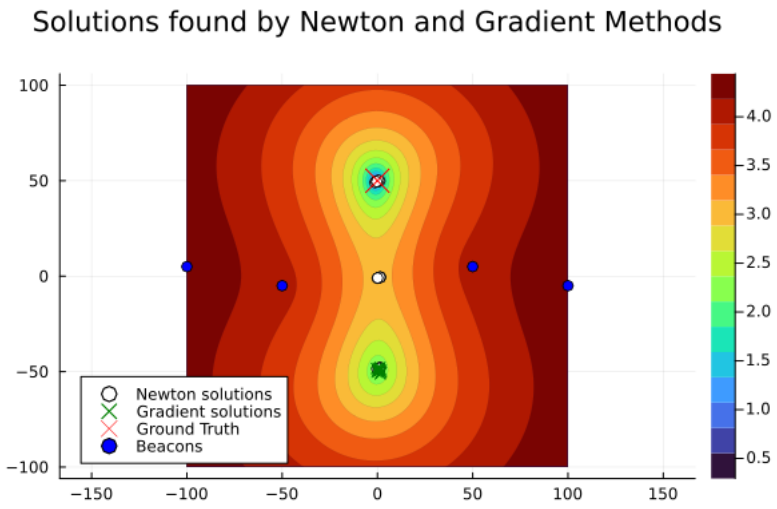
Compare these 3 plots, we find that for the first plot, since the step length is too small so although the gradient method will convergent, but it is very slow (note the green points are much more than the second picture.

The second plot is convergent and converged to a good result with less points than first plot.

The third plot uses large step length, so it is not convergent to a good result.

P9:





(a)

Since the Newton method needs to compute the Hessian matrix and needs to compute its inverse, but when the H is singular (hard to compute the inverse), any small noise will be amplified and that inlfuence the itertaion, so the step will be misleaded. However, the gradient method just simply use the gradient to calculate and guide the next step's direction, which is safe and stable since gradient is always well-defined. So that is the reason why even in the extreme cases, the gradient method convergent but not for Newton's method. We can use condition number to examinate the status of the Hessain matrix; if the conditon number is

very high which means the next step might has some problems.

(b)

We should compare the convergence order for each; the order of convergence for Newton method is quadratic and for gradient method is first order, which means $e_{k+1} \leq C * e_k^2$ for Newton method and $e_{k+1} \leq C * e_k$ for gradient method, so that is the reason why Newton method converges faster.