Problem 1

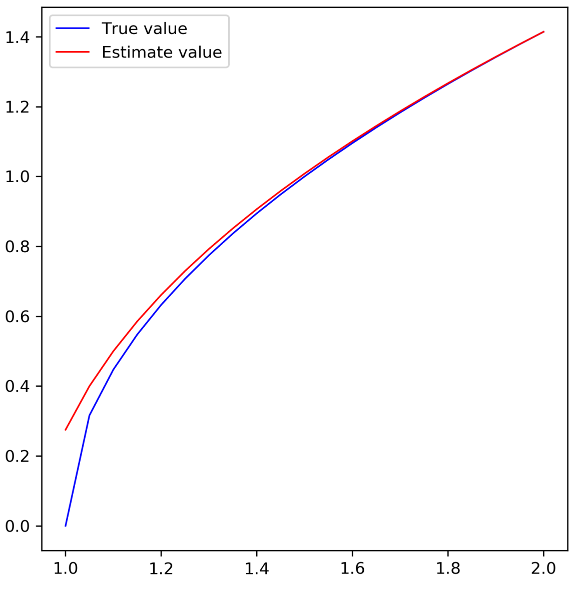
Given function is $\frac{dy}{dx} = \frac{1}{y}$, with $y(2) = \sqrt(2)$

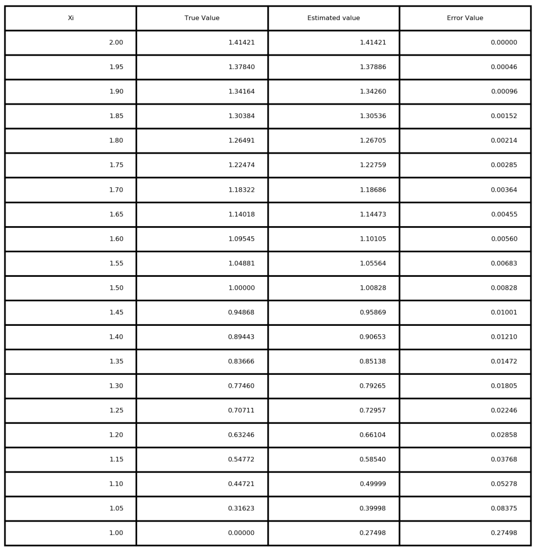
(a) The analytical solution to the above equation is as follows:

$$\frac{dy}{dx} = \frac{1}{y}$$
$$\implies y.dy = dx$$
$$\implies \int y.dy = \int dx$$
$$\implies \frac{y^2}{2} + c_1 = x + c_2$$
$$\implies y = \pm\sqrt{2x + c}$$

now from the initial condition we have:

$$\sqrt{2} = \sqrt{2 * 2 + c}$$
$$2 = 4 + c \implies c = -2$$
$$\implies y = \pm\sqrt{2x - 2}$$

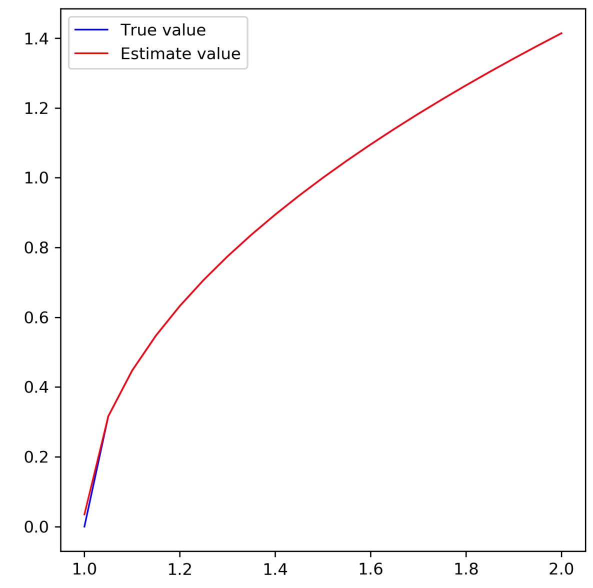


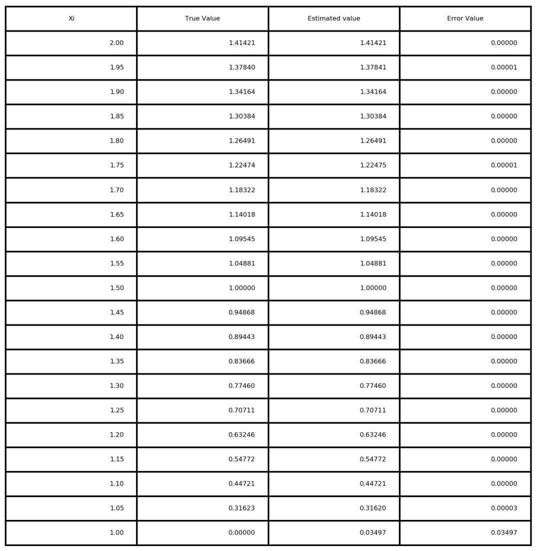


Xi	True Value	Estimated value	Error Value
2.00	1.41421	1.41421	0.00000
1.95	1.37840	1.37886	0.00046
1.90	1.34164	1.34260	0.00096
1.85	1.30384	1.30536	0.00152
1.80	1.26491	1.26705	0.00214
1.75	1.22474	1.22759	0.00285
1.70	1.18322	1.18686	0.00364
1.65	1.14018	1.14473	0.00455
1.60	1.09545	1.10105	0.00560
1.55	1.04881	1.05564	0.00683
1.50	1.00000	1.00828	0.00828
1.45	0.94868	0.95869	0.01001
1.40	0.89443	0.90653	0.01210
1.35	0.83666	0.85138	0.01472
1.30	0.77460	0.79265	0.01805
1.25	0.70711	0.72957	0.02246
1.20	0.63246	0.66104	0.02858
1.15	0.54772	0.58540	0.03768
1.10	0.44721	0.49999	0.05278
1.05	0.31623	0.39998	0.08375
1.00	0.00000	0.27498	0.27498

Figure 1: Graph and table showing the true and estimated values obtained from Euler's method

(b) The code for this part of the problem is given in `q1.py`. The average error for euler's method is:0.0281878 The results for Euler integration are given in Fig 1

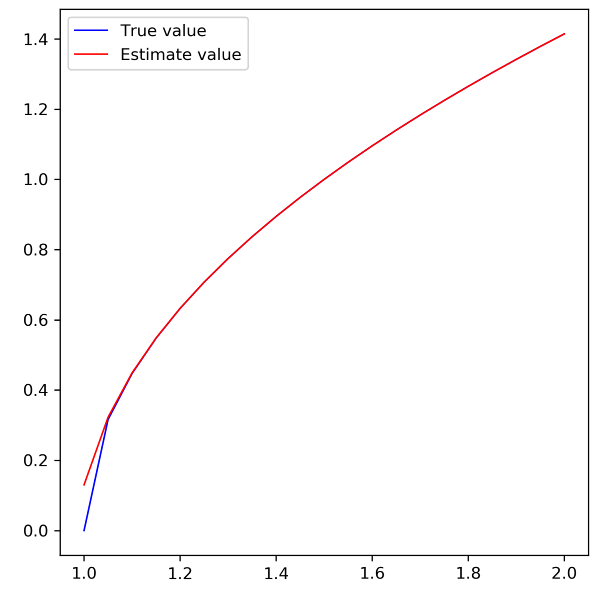


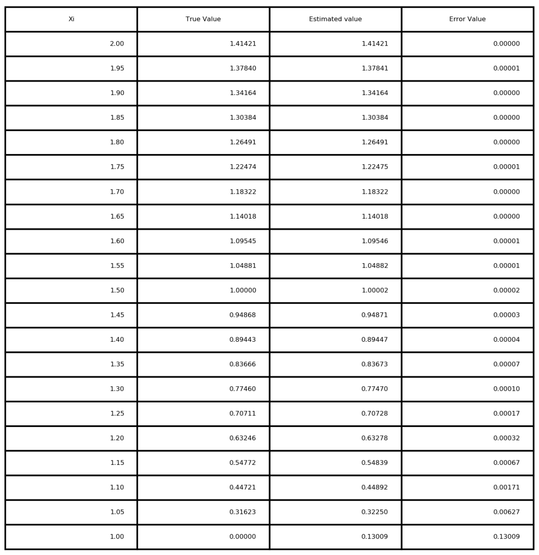


Xi	True Value	Estimated value	Error Value
2.00	1.41421	1.41421	0.00000
1.95	1.37840	1.37841	0.00001
1.90	1.34164	1.34164	0.00000
1.85	1.30384	1.30384	0.00000
1.80	1.26491	1.26491	0.00000
1.75	1.22474	1.22475	0.00001
1.70	1.18322	1.18322	0.00000
1.65	1.14018	1.14018	0.00000
1.60	1.09545	1.09545	0.00000
1.55	1.04881	1.04881	0.00000
1.50	1.00000	1.00000	0.00000
1.45	0.94868	0.94868	0.00000
1.40	0.89443	0.89443	0.00000
1.35	0.83666	0.83666	0.00000
1.30	0.77460	0.77460	0.00000
1.25	0.70711	0.70711	0.00000
1.20	0.63246	0.63246	0.00000
1.15	0.54772	0.54772	0.00000
1.10	0.44721	0.44721	0.00000
1.05	0.31623	0.31620	0.00003
1.00	0.00000	0.03497	0.03497

Figure 2: Graph and table showing the true and estimate values obtained from RK4 method

(c) The code for this part is given in `q1.py`. The average error for 4th order Runge-Kutta method is : 0.0016690. The results for RK4 integration method is as given in Fig 2





Xi	True Value	Estimated value	Error Value
2.00	1.41421	1.41421	0.00000
1.95	1.37840	1.37841	0.00001
1.90	1.34164	1.34164	0.00000
1.85	1.30384	1.30384	0.00000
1.80	1.26491	1.26491	0.00000
1.75	1.22474	1.22475	0.00001
1.70	1.18322	1.18322	0.00000
1.65	1.14018	1.14018	0.00000
1.60	1.09545	1.09546	0.00001
1.55	1.04881	1.04882	0.00001
1.50	1.00000	1.00002	0.00002
1.45	0.94868	0.94871	0.00003
1.40	0.89443	0.89447	0.00004
1.35	0.83666	0.83673	0.00007
1.30	0.77460	0.77470	0.00010
1.25	0.70711	0.70728	0.00017
1.20	0.63246	0.63278	0.00032
1.15	0.54772	0.54839	0.00067
1.10	0.44721	0.44892	0.00171
1.05	0.31623	0.32250	0.00627
1.00	0.00000	0.13009	0.13009

Figure 3: Graph and table showing the true and estimated values obtained from AB4 method

(d) The average for 4th order Adam Bashforth method is: 0.0066446. The results for Adams-Bashforth 4th order integration method is as given in Fig 3

Problem 2

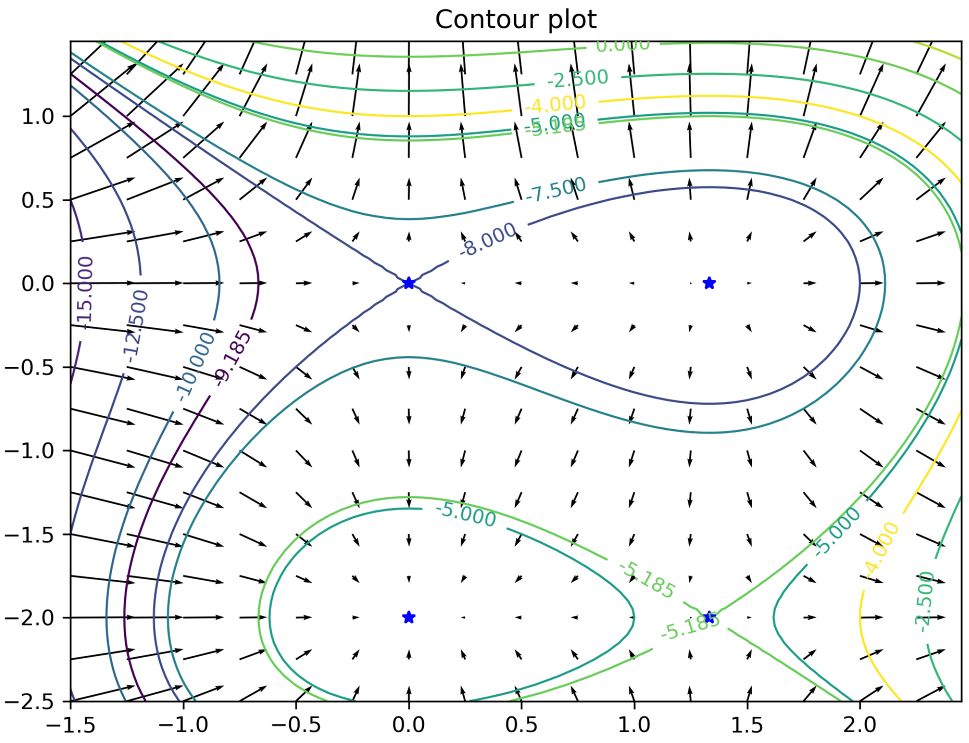


Figure 4: contour plot for the function $f(x, y)$. Black arrows denote the gradient direction, and blue * denote the critical points

(a) The code for sketching the iso-contours of the function is given in `q2.py` We have the function as follows:

$$f(x, y) = x^3 + y^3 - 2x^2 + 3y^2 - 8$$
$$\frac{\partial f}{\partial x} = 3x^2 - 4x = 0$$
$$\implies x_1 = 0, x_2 = \frac{4}{3}$$
$$\frac{\partial f}{\partial y} = 3y^2 + 6y = 0$$
$$\implies y_1 = 0, y_2 = -2$$

The critical points therefore are as follows:

$$(0, 0), (0, -2), (\frac{4}{3}, 0), (\frac{4}{3}, -2)$$

The contour plot of the function is as given in Fig 4

From the graph and gradient map, at the critical points, we can observe that:

- $(0,0)$ and $(\frac{4}{3},-2)$ are saddle points
- $(0,-2)$ is a local minima
- $(\frac{4}{3},0)$ is a local maxima

(b) We know from part (a) that

$$\nabla f = \begin{bmatrix} 3x^2 - 4x \\ 3y^2 + 6y \end{bmatrix}$$

We know that $x^{(0)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ therefore we have $\nabla f(x^{(0)}) = \begin{bmatrix} -1 \\ -3 \end{bmatrix}$ Now, we are to minimize $g(t)$:

$$g(t) = f(x^{(0)} - tu^{(0)})$$
$$g(t) = (1+t)^3 + (-1+3t)^3 - 2(1+t)^2 + 3(-1+3t)^2 - 8$$

We are to find the optimal t^* which minimizes $g(t)$, therefore:

$$\frac{\partial g(t)}{\partial t} = 3(1+t)^2 + 9(-1+3t)^2 - 4(1+t) + 18(-1+3t) = 0$$
$$\implies t_1 = \frac{-5}{14}, t_2 = \frac{1}{3}$$

When $t^* = \frac{1}{3}$, $g(t^*)$ is minimized, therefore, for the next step:

$$x^{(1)} = \left[x^{(0)} - tu^{(0)}\right] = \begin{bmatrix} \frac{4}{3} \\ 0 \end{bmatrix}$$
$$\nabla f(x^{(1)}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This means that gradient descent converges to the local minima, in **one step**, since, after one step the gradient of the function is 0.

Problem 3

(a) Suppose $\mathbf{v_i}$ and $\mathbf{v_j}$ are two eigenvectors of matrix $\mathbf{Q}$ then we have

$$\mathbf{Q}\mathbf{v_i} = \lambda_i \mathbf{v_i} \tag{1}$$
$$\mathbf{Q}\mathbf{v_j} = \lambda_j \mathbf{v_j} \tag{2}$$

Premultiplying 1 and 2 by $\mathbf{v_j}^\top$ and $\mathbf{v_i}^\top$ respectively we have:

$$\mathbf{v_j}^\top \mathbf{Q}\mathbf{v_i} = \lambda_i \mathbf{v_j}^\top \mathbf{v_i} \tag{3}$$
$$\mathbf{v_i}^\top \mathbf{Q}\mathbf{v_j} = \lambda_j \mathbf{v_i}^\top \mathbf{v_j} \tag{4}$$

Now we have:

$$(\mathbf{v_j}^\top \mathbf{Q}\mathbf{v_i})^\top = \mathbf{v_i}^\top \mathbf{Q}^\top \mathbf{v_j} = \mathbf{v_i}^\top \mathbf{Q}\mathbf{v_j} = \lambda_j \mathbf{v_i}^\top \mathbf{v_j}$$

Since $\mathbf{Q}^\top = \mathbf{Q}$, the matrix being symmetric. Also, since the R.H.S is a scalar quantity and can be defined as $\langle \mathbf{v_i}, \mathbf{v_j} \rangle = \mathbf{v_j}^\top \mathbf{v_i}$, we now have:

$$(\lambda_i \mathbf{v_j}^\top \mathbf{v_i})^\top = \lambda_i \mathbf{v_j}^\top \mathbf{v_i} = \lambda_j \mathbf{v_i}^\top \mathbf{v_j}$$

Now we have: $(\lambda_i - \lambda_j)\mathbf{v_j}^\top \mathbf{v_i} = 0$. Since we know that the eigenvalues are distinct, this requires that $\mathbf{v_j}^\top \mathbf{v_i} = 0$, meaning $\mathbf{v_j}$ and $\mathbf{v_i}$ are orthogonal, and this means:

$$\lambda_i \mathbf{v_j}^\top \mathbf{v_i} = 0 = \mathbf{v_j}^\top \mathbf{Q}\mathbf{v_i}$$

This shows that eigen vectors $\mathbf{v_j}$ and $\mathbf{v_i}$ are Q-orthogonal

(b) Since we know that we can find pairwise orthogonal eigen vectors regardless of the eigenvalues being distinct, we know that $\langle \mathbf{v_i}, \mathbf{v_j} \rangle = 0$. Therefore we have:

$$\lambda_i \mathbf{v_j}^\top \mathbf{v_i} = 0 = \mathbf{v_j}^\top \mathbf{Q}\mathbf{v_i}$$

This means that any two orthogonal basis vectors are Q-orthogonal as well

Problem 4

(a) We know from conjugate descent algorithm that:

$$\beta_k = \frac{g_{k+1}^\top Q d_k}{d_k^\top Q d_k}$$
$$d_{k+1} = -g_{k+1} + \beta_k d_k$$
$$\implies d_k = = -g_k + \beta_{k-1} d_{k-1}$$

Now we compute:

$$d_k^\top Q d_k = d_k^\top Q(-g_k + \beta_{k-1} d_{k-1})$$
$$d_k^\top Q d_k = -d_k^\top Q g_k + \cancel{\beta_{k-1} d_k^\top Q d_{k-1}}$$

The last term gets cancelled since d_k and d_{k-1} are Q orthogonal by definition. Therefore we prove that: $d_k^\top Q d_k = -d_k^\top Q g_k$

Further we observe similarly that:

$$-g_k^\top d_k = g_k^\top (g_k - \beta_{k-1} d_{k-1})$$
$$-g_k^\top d_k = g_k^\top g_k - \cancel{\beta_{k-1} g_k^\top d_{k-1}}$$

The second term cancels out, owing to the expanding subspace theorem. We now show that to obtain x_{k+1} from x_k we require Q only to compute g_k and Qg_k. We know from Conjugate descent algorithm that:

$$x_{k+1} = x_k + \alpha_k d_k$$
$$\alpha_k = \frac{-g_k^\top d_k}{d_k^\top Q d_k}$$

Combining the above two equations we have:

$$x_{k+1} = x_k + \frac{-g_k^\top d_k}{d_k^\top Q d_k} d_k$$
$$x_{k+1} = x_k - \frac{g_k^\top g_k}{d_k^\top Q g_k} d_k$$

Also we know that $g_k = b + Qx_k$. Therefore we observe that to compute x_{k+1} from x_k we require Q only to compute g_k and Qg_k.

(b) We know that $y_k = x_k - g_k$ now

$$p_k = \nabla f(y_k)$$
$$p_k = Qy_k + b$$
$$p_k = Q(x_k - g_k) + b$$
$$p_k = (Qx_k + b) - Qg_k$$
$$\implies Qg_k = g_k - p_k$$

Since we know that $Qx_k + b = g_k$

(c) From part (a) and (b) we know that:

$$x_{k+1} = x_k - \frac{g_k^\top g_k}{d_k^\top Q g_k} d_k \tag{5}$$

$$Qg_k = g_k - p_k \tag{6}$$

Substituting, equation 5 in equation 6, we have

$$x_{k+1} = x_k - \frac{g_k^\top g_k}{d_k^\top (g_k - p_k)} d_k \tag{7}$$

Further we observe that to obtain β_k we have:

$$\beta_k = \frac{g_{k+1}^\top Q d_k}{d_k^\top Q d_k}$$

For a general function f, if we consider only its second order taylor's approximation, We can write $g_k = Qx_k + b$ and thus we have:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k d_k \\ Qx_{k+1} &= Qx_k + \alpha_k Q d_k \\ Qx_{k+1} + b &= Qx_k + b + \alpha_k Q d_k \\ \implies g_{k+1} &= g_k + \alpha_k Q d_k \\ \implies Qd_k &= \frac{g_{k+1} - g_k}{\alpha_k} \end{aligned}$$

We can then write

$$\beta_k = \frac{g_{k+1}^\top (g_{k+1} - g_k)}{\alpha d_k^\top Q d_k}$$

From the expanding subspace theorem again we have $g_{k+1} \perp g_k$ therefore we have:

$$\beta_k = \frac{g_{k+1}^\top g_{k+1}}{g_k^\top g_k} \tag{8}$$

Since we observe that $\alpha = \frac{g_k^\top g_k}{d_k^\top Q d_k}$

Problem 5

We are given that we need to maximize the area of a rectangle, subject to the constraint that the perimeter is fixed. This can be written mathematically as:

$$\max(f(x,y)) = xy \implies \min(f(x,y)) = -xy$$
$$h(x,y) = 2x + 2y = c \implies 2x + 2y - c = 0$$

Writing down the Lagrange multiplier for the above two functions we have:

$$\mathcal{L}(x,y,\lambda) = f(x,y) + \lambda h(x,y) \implies \mathcal{L}(x,y,\lambda) = -xy + \lambda(2x + 2y - c)$$

Now we set the $\nabla\mathcal{L} = 0$

$$\nabla\mathcal{L} = \begin{bmatrix} -y + 2\lambda \\ -x + 2\lambda \\ 2x + 2y - c \end{bmatrix} = 0$$

Solving the three equations individually, we get:

$$\begin{aligned} y = 2\lambda &= x \\ 4\lambda + 4\lambda - c = 0 \implies \lambda &= \frac{c}{8} \\ \implies y = x &= \frac{c}{4} \end{aligned}$$

So at $x^* = y^* = \frac{c}{4}$ the rectangle has the highest area.
For the second order sufficiency condition, we have:

$$\nabla^2 f(x^*) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial xy} \\ \frac{\partial^2 f}{\partial xy} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$
$$\nabla^2 h(x^*) = \begin{bmatrix} \frac{\partial^2 h}{\partial x^2} & \frac{\partial^2 h}{\partial xy} \\ \frac{\partial^2 h}{\partial xy} & \frac{\partial^2 h}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

We then have:

$$\nabla^2 f(x^*) + \lambda\nabla^2 h(x^*) = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

Now we find λ satisfying the following:

$$\begin{aligned} \nabla f(x^*) + \lambda^\top \nabla h(x^*) &= 0 \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \lambda^\top \begin{bmatrix} 2 \\ 2 \end{bmatrix} &= 0 \\ \implies \lambda &= \begin{bmatrix} p \\ -p \end{bmatrix} \end{aligned}$$

Where p is any arbitrary real number. Now we require that the following is positive definite i.e.,

$$\lambda^\top(\nabla^2 f(x^*) + \lambda\nabla^2 h(x^*))\lambda = \begin{bmatrix} p & -p \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} p \\ -p \end{bmatrix} = 2p^2 \geq 0$$

Thus the second order sufficiency condition is verified.

Problem 6

(a) The solution to this problem is given in `trajectory_optiimization.py`. Please observe the section marked `Q6(a)` in the comments.

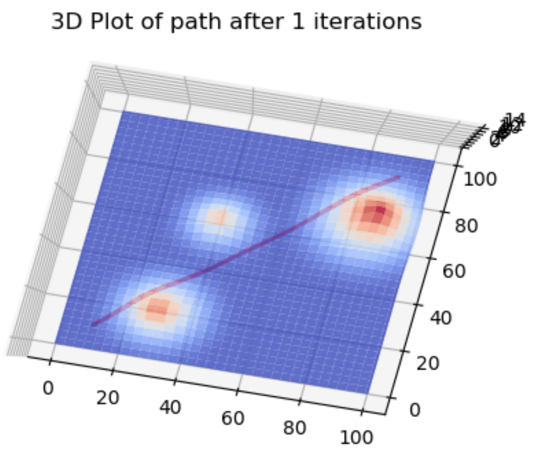
We observe from the path after convergence that the points get unevenly distributed or isolated since the trajectory of points are considered as individual points, and move away from the start and end points to produce a trajectory that is not smooth. Further some of the points get accumulated near the center obstacle.

(b) The solution to this part of the problem is given in `trajectory_optimization.py`. Please observe the section marked `Q6(b)` in the comments. We observe from this that since we do not consider the next path point when computing the cost of the trajectory, the points get accumulated at the front end of the trajectory and do not reach the goal. This happens because the points are optimized to tend to remain close to the previous point.

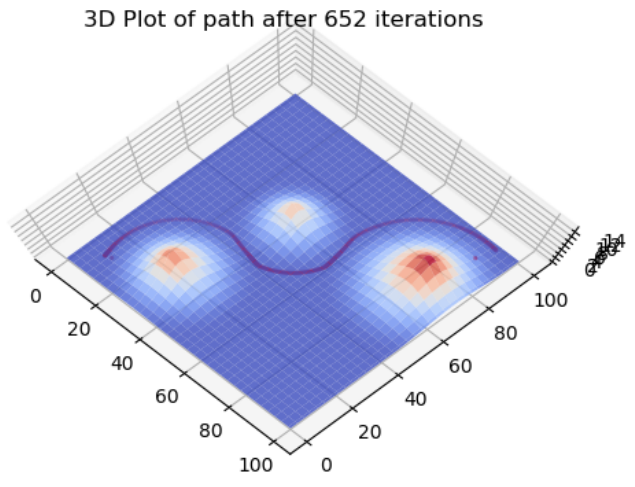
(c) The solution to this part is also provided in `trajectory_optimization.py`. Please observe the section marked as `Q6(c)` in the comments. This version is considerably better than the previous approach because here in the penalty term we optimize the points to remain close to both the previous point and the next point, this ensures that the path remains smooth even near the initial and destination points.

(d) With different initial trajectories we will not obtain the same paths because, the points could be on the other side of the obstacles, which are symmetric about its axis, and could cause gradient descent to move the points to the corresponding side of the obstacle. The premise here is that, with different initial points, gradient descent will cause descent is different directions according to the local topology (obstacle gradient).

(e) When we apply the original cost function we observe that the points move according to the negative gradient in an isolated manner. In the case that the trajectory still passes through some high cost regions, we could randomly perturb the initial trajectory in an iterative manner to obtain a low cost path. On obtaining a path, we could then interpolate splines between the points since the points could be isolated among low cost regions.

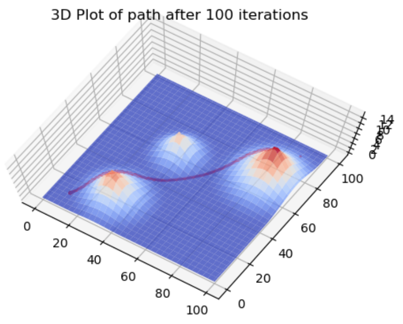


(a) Plot of the path after 1 iteration

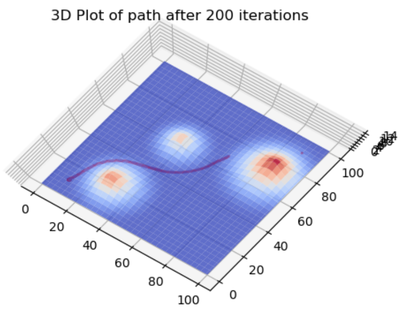


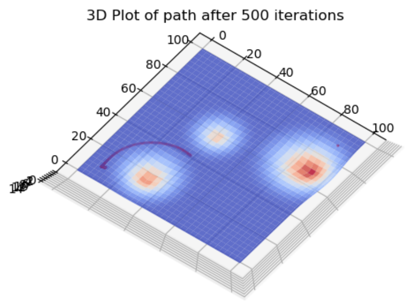
(b) Plot of the path after convergence

Figure 5: Plot for the paths obtained in section `Q6(a)`



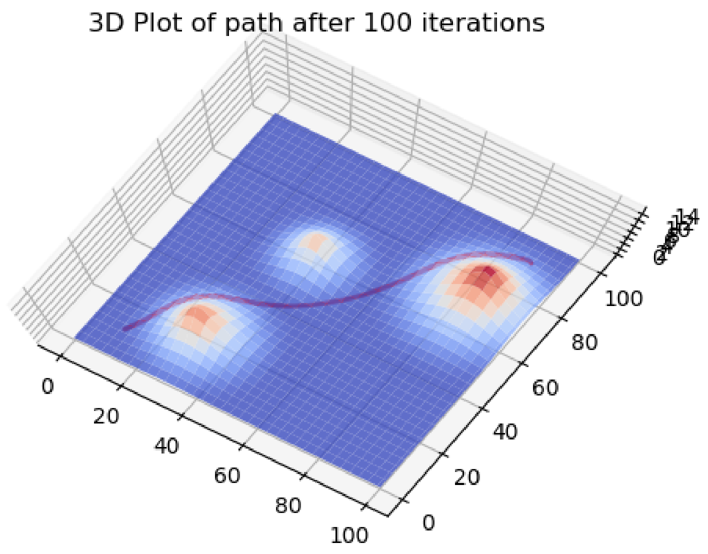
(a) Plot of the path after 1 iteration



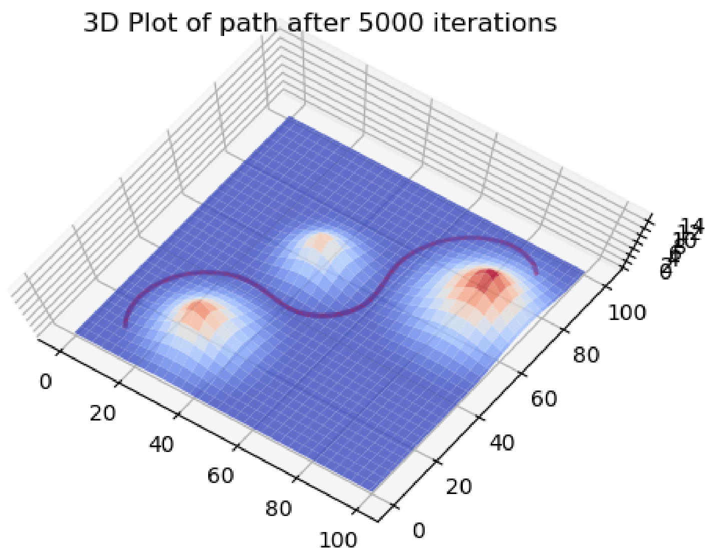


(b) Plot of the path after 200 iterations (c) Plot of the path after 500 iterations

Figure 6: Plot for the path obtained in section `Q6(b)`



(a) Plot of the path after 1 iteration



(b) Plot of the path after 5000 iterations

Figure 7: Plots for the path obtained in section `Q6(c)`

Discussion Citation

For this Homework set, I have discussed with *Tarasha Khurana* andrewID: `tkhurana@andrew.cmu.edu`, and *Viraj Parimi* andrewID: `vparimi@andrew.cmu.edu`

References

[1] Remez Algorithm
`https://en.wikipedia.org/wiki/Remez_algorithm`

[2] Fischler, Martin A. and Bolles, Robert C, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography* Commun. ACM June 1981 vol. 24 num 6