HW3

Lifan Yu lifany

October 2023

Question 1

a

We know:

$$\frac{dy}{dx} = \frac{2}{3\sqrt{y}}, y(1) = 1$$

We rearrange the equation:

$$\int 3\sqrt{y}dy = \int 2dx$$

$$3\int \sqrt{y}dy = 2\int 1dx$$

$$\frac{2}{3} * 3 * y^{\frac{3}{2}} = 2 * x + C$$

$$y^{\frac{3}{2}} = x + C$$

Substitute $y(1) = 1$ into the above:

$$1^{\frac{3}{2}} = 1 + C$$

$$C = 0$$

$$y^{\frac{3}{2}} = x$$

$$y = \sqrt[3]{x^2}$$

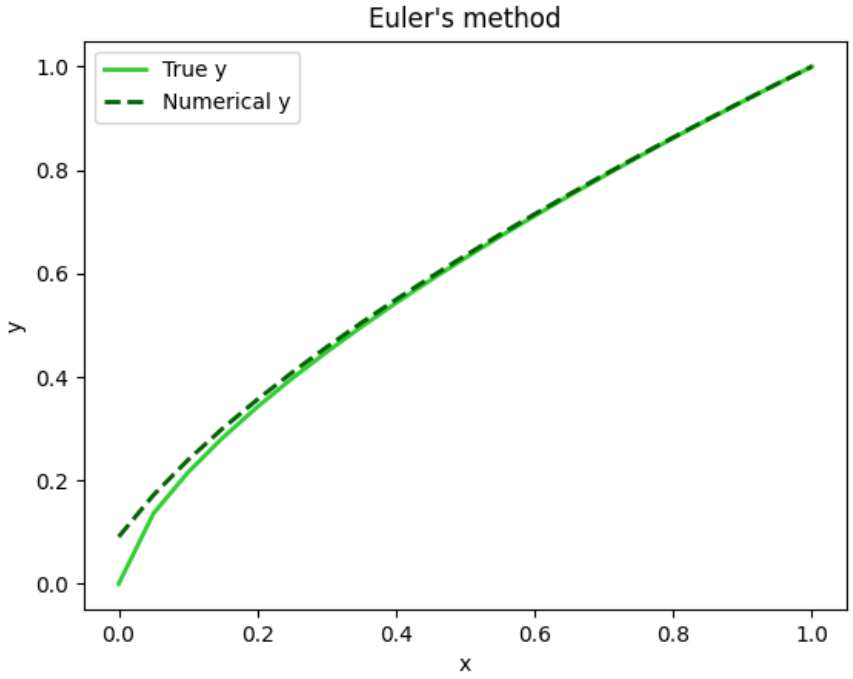
We have found the exact $\boxed{y(x) = y = \sqrt[3]{x^2}}$

We can verify that when $x = 1$, $y(1) = \sqrt[3]{1^2} = 1$ which satisfies the differential equation.

b

To run:

```
python q1.py
```



Program output

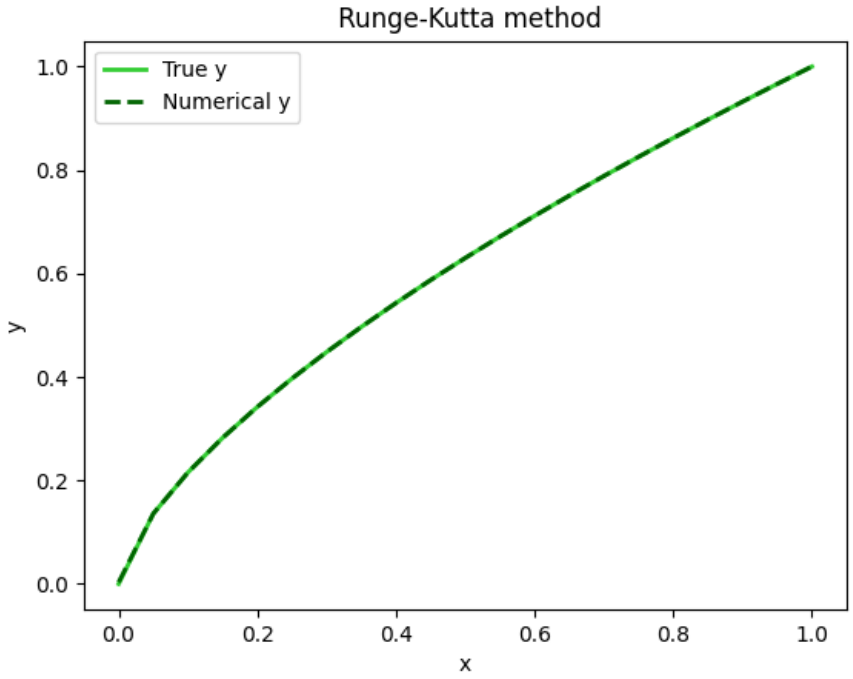
```
Table for Euler s method :
      x_i  True y(x_i)  Numerical y_i  Err y(x_i) - y_i
0    1.00     1.000000       1.000000          0.000000
1    0.95     0.966383       0.966667         -0.000284
2    0.90     0.932170       0.932763         -0.000594
3    0.85     0.897317       0.898250         -0.000933
4    0.80     0.861774       0.863079         -0.001305
5    0.75     0.825482       0.827199         -0.001717
6    0.70     0.788374       0.790549         -0.002175
7    0.65     0.750370       0.753059         -0.002689
8    0.60     0.711379       0.714647         -0.003269
9    0.55     0.671287       0.675217         -0.003929
10   0.50     0.629961       0.634651         -0.004691
11   0.45     0.587230       0.592809         -0.005579
12   0.40     0.542884       0.549516         -0.006632
13   0.35     0.496644       0.504550         -0.007905
14   0.30     0.448140       0.457622         -0.009482
15   0.25     0.396850       0.408347         -0.011497
16   0.20     0.341995       0.356184         -0.014189
17   0.15     0.282311       0.300332         -0.018021
18   0.10     0.215443       0.239507         -0.024064
19   0.05     0.135721       0.171396         -0.035675
20   0.00     0.000000       0.090881         -0.090881
```

```
Error metric: mean absolute error
 Maximum error 0.09088067557348478
 Mean error 0.011690998407682295
```

c

To run:

```
python q1.py
```



Program output

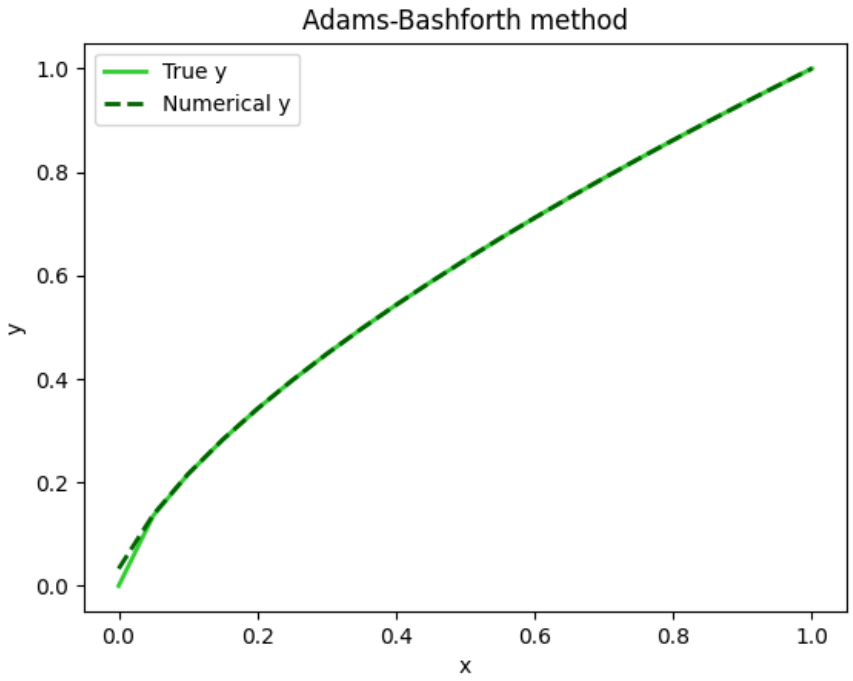
```
Table for Runge-Kutta method :
      x_i  True y(x_i)  Numerical y_i  Err y(x_i) - y_i
0    1.00     1.000000       1.000000      0.000000e+00
1    0.95     0.966383       0.966383      1.623409e-10
2    0.90     0.932170       0.932170      3.690975e-10
3    0.85     0.897317       0.897317      6.353017e-10
4    0.80     0.861774       0.861774      9.823542e-10
5    0.75     0.825482       0.825482      1.441357e-09
6    0.70     0.788374       0.788374      2.058575e-09
7    0.65     0.750370       0.750370      2.904666e-09
8    0.60     0.711379       0.711379      4.090859e-09
9    0.55     0.671287       0.671287      5.798494e-09
10   0.50     0.629961       0.629961      8.335596e-09
11   0.45     0.587230       0.587230      1.225149e-08
12   0.40     0.542884       0.542884      1.858517e-08
13   0.35     0.496644       0.496644      2.944940e-08
14   0.30     0.448140       0.448140      4.955214e-08
15   0.25     0.396850       0.396850      9.071641e-08
16   0.20     0.341995       0.341995      1.878990e-07
17   0.15     0.282311       0.282310      4.722343e-07
18   0.10     0.215443       0.215442      1.666788e-06
19   0.05     0.135721       0.135709      1.231157e-05
20   0.00     0.000000       0.003153     -3.152629e-03
```

```
Error metric: mean absolute error
 Maximum error 0.003152629431472015
 Mean error 0.00015083310718636266
```

d

To run:

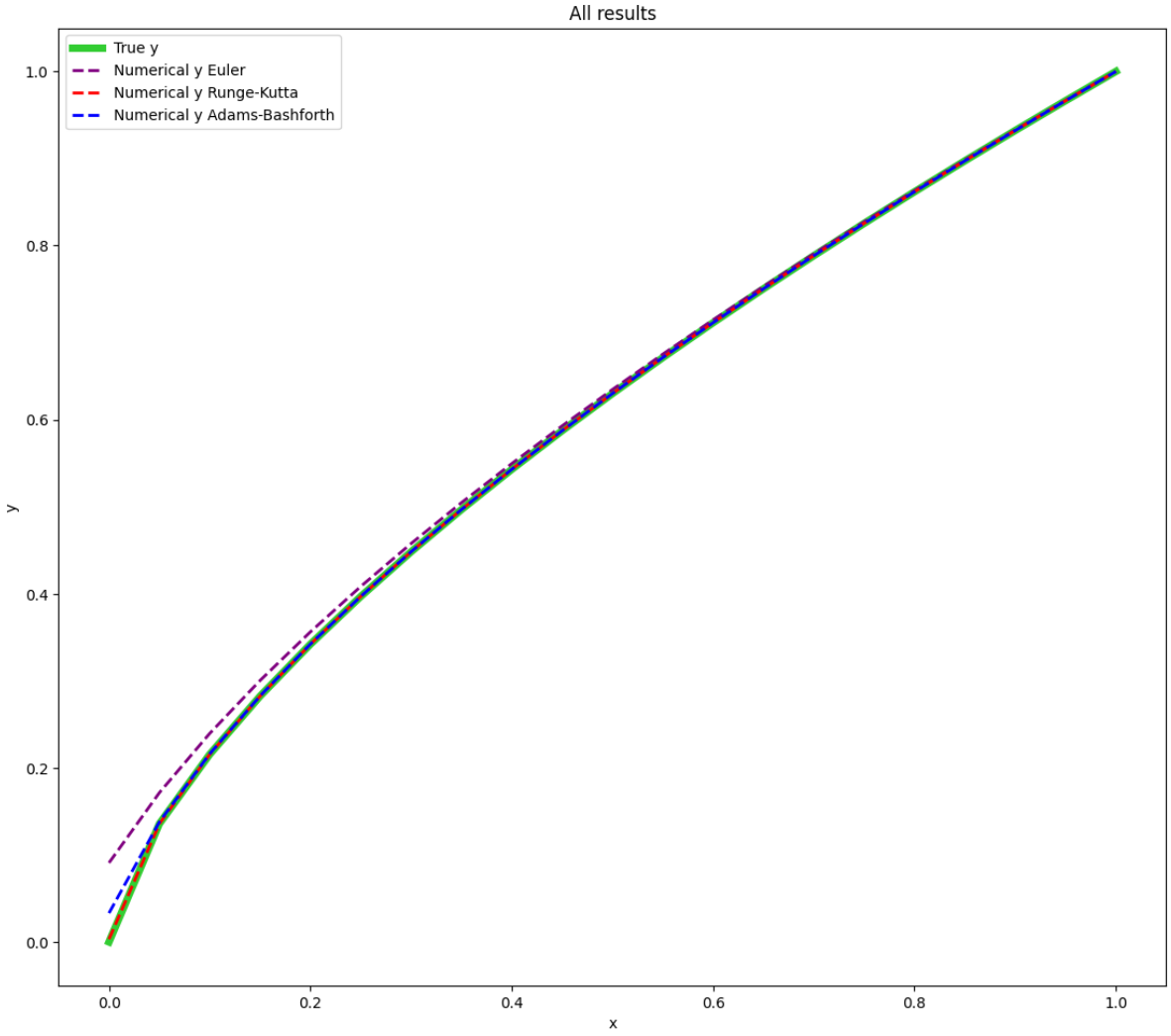
```
python q1.py
```



Program output:

```
Table for Adams-Bashforth method :
      x_i  True y(x_i)  Numerical y_i  Err y(x_i) - y_i
0    1.00     1.000000       1.000000      0.000000e+00
1    0.95     0.966383       0.966383     -2.025644e-07
2    0.90     0.932170       0.932170     -4.610275e-07
3    0.85     0.897317       0.897318     -7.845845e-07
4    0.80     0.861774       0.861775     -1.199788e-06
5    0.75     0.825482       0.825484     -1.737959e-06
6    0.70     0.788374       0.788376     -2.445161e-06
7    0.65     0.750370       0.750374     -3.389670e-06
8    0.60     0.711379       0.711383     -4.674870e-06
9    0.55     0.671287       0.671294     -6.462045e-06
10   0.50     0.629961       0.629970     -9.011508e-06
11   0.45     0.587230       0.587243     -1.276040e-05
12   0.40     0.542884       0.542902     -1.847826e-05
13   0.35     0.496644       0.496672     -2.759911e-05
14   0.30     0.448140       0.448183     -4.298957e-05
15   0.25     0.396850       0.396921     -7.091319e-05
16   0.20     0.341995       0.342122     -1.267532e-04
17   0.15     0.282311       0.282566     -2.549259e-04
18   0.10     0.215443       0.216062     -6.189447e-04
19   0.05     0.135721       0.137862     -2.141575e-03
20   0.00     0.000000       0.033116     -3.311562e-02
```

```
Error metric: mean absolute error
 Maximum error 0.03311561596078856
 Mean error 0.0017362345107478116
```


All results
True y
Numerical y Euler
Numerical y Runge-Kutta
Numerical y Adams-Bashforth
y
x
0.0
0.2
0.4
0.6
0.8
1.0

Question 2

a

Critical points:

$$\frac{\partial f}{\partial x} = 3x^2 + 12x = 0$$
$$\frac{\partial f}{\partial y} = 3y^2 - 6y = 0$$

we have $x = 0$ or $x = -4$, $y = 0$ or $y = 2$

Critical points are $\boxed{(0,0), (-4,0), (0,2), (-4,2)}$

We construct the Hessian matrix:

$$\begin{bmatrix} \frac{\partial^2 f}{\partial^2 x} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial^2 y} \end{bmatrix} = \begin{bmatrix} 6x + 12 & 0 \\ 0 & 6y - 6 \end{bmatrix}$$

$\boxed{\text{Local minima: } (0,2)}$

Because for $(0,2)$, the Hessian matrix is $\begin{bmatrix} 12 & 0 \\ 0 & 6 \end{bmatrix}$, its determinant is $72 > 0$ abd eigen values 12 and 6 are both positive. The matrix is positive definite.

$\boxed{\text{Local maxima: } (-4,0)}$

Because for $(-4,0)$, the Hessian matrix is $\begin{bmatrix} -12 & 0 \\ 0 & -6 \end{bmatrix}$, its determinant is -72 > 0 and eigen values -12 and -6 are both negative.

$\boxed{\text{Saddle points: (0, 0) and (-4, 2)}}$

Because their respective Hessian matrices are $\begin{bmatrix} 12 & 0 \\ 0 & -6 \end{bmatrix}$ and $\begin{bmatrix} -12 & 0 \\ 0 & 6 \end{bmatrix}$ and both have a negative determinant.

b

Gradient of f:

$$\nabla f = \begin{bmatrix} 3x^2 + 12x \\ 3y^2 - 6x \end{bmatrix}$$

First iteration

Gradient of f at (-3, 1):

$$\mathbf{u} = \nabla f(-3, 1) = \begin{bmatrix} 3 * (-3)^2 + 12 * (-3) \\ 3 * 1^2 - 6 * 1 \end{bmatrix} = \begin{bmatrix} -9 \\ -3 \end{bmatrix}$$

$$g(t) = f(x^{(m)} - t * \mathbf{u}) = f(-3 + 3t, 1 + t) = (-3 + 9t)^3 + (1 + 3t)^3 + 6(-3 + 9t)^2 - 3(1 + 3t)^2 + 2$$

$$g(t) = 756t^3 - 243t^2 - 90t + 27 = 0$$

To minimize $g(t)$

$$g'(t) = 18(126t^2 - 27t - 5) = 0$$

Take the positive t closest to 0:

$$t = \frac{1}{3}$$

Now update x

$$x^{(m+1)} = x^m - t * \mathbf{u} = \begin{bmatrix} -3 - \frac{1}{3} * (-9) \\ 1 - \frac{1}{3} * (-3) \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

Second iteration

Gradient of f at (0, 2):

$$\mathbf{u} = \nabla f(0, 2) = \begin{bmatrix} 0 \\ 3 * 2^2 - 6 * 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Stop.

Only **1 step** is taken to reach the overall local minimum (0, 2)

Question 3

Let Q be a symmetric positive definite matrix, λ, v denote eigen values and eigen vectors. We take any two corresponding eigen values and eigen vectors. By definition of eigen values:

$$Qv_i = \lambda_i v_i$$
$$Qv_j = \lambda_j v_j$$

Left multiply v_j to $Qv_i = \lambda_i v_i$

$$v_j^T Q v_i = v_j \lambda_i v_i$$

$$(Q^T v_j)^T v_i = \lambda_i v_j^T v_i$$

Because Q is symmetric:

$$(Qv_j)^T v_i = \lambda_i v_j^T v_i$$

$$\lambda_j v_j^T v_i = \lambda_i v_j^T v_i$$

When $\lambda_i \neq \lambda_j$, we have $v_j^T v_i = 0$. Then:

$$v_j^T Q v_i = v_j \lambda_i v_i = \lambda_i v_j^T v_i = 0$$

Thus any two eigenvectors of Q corresponding to distinct eigenvalues are Q-orthogonal.

Question 4

a

From the conjugate gradient method, we know that $d_k \longleftarrow -g_k - \beta_{k-1} d_{k-1}$

$d_k^T Q d_k$

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

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

$= -d_k^T Q g_k - \beta_{k-1} d_k^T Q d_{k-1}$

Because d_i's are Q-orthogonal, $d_k^T Q d_{k-1} = 0$. Therefore we have $d_k^T Q d_k = -d_k^T Q g_k$

If we know Qg_k, we do not need to explicitly know Q to express x_{k+1} from x_k

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

$$x_{k+1} \longleftarrow x_k + \frac{g_k^T d_k}{d_k^T Q g_k} d_k$$

b

We are given that $y_k = x_k - g_k$, $p_k = \nabla f(y_k)$, therefore

$$Qy_k = Qx_k - Qg_k$$

$$Qy_k + b = Qx_k + b - Qg_k$$

Because we know that $\nabla f(x) = Qx + b$, the above becomes

$$\nabla f(y_k) = \nabla f(x_k) - Qg_k$$

$$Qg_k = \nabla f(x_k) - \nabla f(y_k) = g_k - p_k$$

c

Combining (a) and (b), the conjugate gradient method can be written as

Let $d_0 = -g_0 = -\nabla f(x_0)$

for $k = 0, 1, ..., n-1$, do:

$p_k \longleftarrow \nabla f(x_k - g_k)$

$\boxed{\alpha_k \longleftarrow -\frac{g_k^T d_k}{d_k^T Q d_k} = \frac{g_k^T d_k}{d_k^T Q g_k} = \frac{g_k^T d_k}{d_k^T (g_k - p_k)}}$

$x_{k+1} \longleftarrow x_k + \alpha_k d_k$

$\boxed{\beta_k \longleftarrow \frac{g_{k+1}^T Q d_k}{d_k^T Q d_k} = \frac{g_{k+1}^T Q d_k}{-d_k^T Q g_k} = -\frac{g_{k+1}^T Q d_k}{d_k^T (g_k - p_k)} = -\frac{(Q g_{k_1})^T d_k}{d_k^T (g_k - p_k)} = -\frac{(g_k - p_k)^T d_k}{d_k^T (g_k - p_k)}}$

$d_{k+1} \longleftarrow -g_{k+1} + \beta_k d_k$

return x_n

Question 5

First order necessary condition

We want to maximize the area, which is xy, so we need to minimize $- = xy$. We write

$$f(x, y) = -xy$$

Subject to the constant perimeter constraint

$$g(x, y) = 2x + 2y - C = 0, C \in R, C > 0$$

$$F(x, y, \lambda) - -xy + \lambda(2x + 2y - C) = 0, \lambda \in R$$

$$\nabla F = \begin{bmatrix} -y + 2\lambda \\ -x + 2\lambda \\ 2x + 2y - C \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$x = \frac{C}{4}, y = \frac{C}{4}, \lambda = \frac{C}{8}$$

The greatest area is $\boxed{f(x, y) = (\frac{C}{4})^2 = \frac{C^2}{16}}$ at $\boxed{x* = (\frac{C}{4}, \frac{C}{4}, \frac{C}{8})^T}$

Second order sufficient condition

$$\nabla h = (2, 2)^T$$

$$\nabla^2 h = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\nabla f = (-y, -x)^T$$

$$\nabla^2 h = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

$x*$ denotes the local minimum of f subject to h

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

We need to prove that the matrix $L(x*)$ is positive definite.on $m = \{y : \nabla h(x*)y = 0\}$. Therefore, we need to find a vector $w \in m, w \neq 0$ such that $w^T \nabla^2 L(x*, \lambda*)w > 0$ is true for all $w \in F(\lambda*)$

To find w, let

$$\nabla h(x, y)^T w = 0$$

$$[2, 2] \begin{bmatrix} w_1 \\ w2 \end{bmatrix} = 0$$

$$w_1 = -w_2$$

Denote w as

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} c \\ -c \end{bmatrix}, c \in R, c \neq 0$$

We now evaluate whether or not $L(x*)$ is positive definite

$$w^T \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} w = \begin{bmatrix} c & -c \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} c \\ -c \end{bmatrix} = \begin{bmatrix} c & -c \end{bmatrix} \begin{bmatrix} c \\ -c \end{bmatrix} = c^2 + c^2 = 2c^2$$

Because $c \neq 0$, $\boxed{2c^2 \text{ is always greater than } 0}$. Therefore, $\boxed{L(x*) \text{ is always a positive definite matrix for } x*}$. The second order sufficient condition is proved.

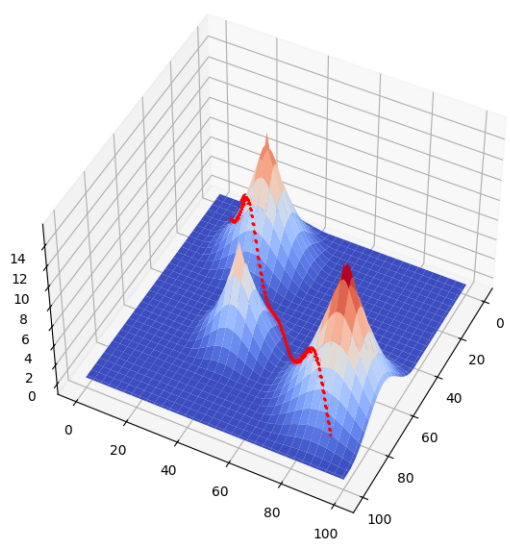
Question 6

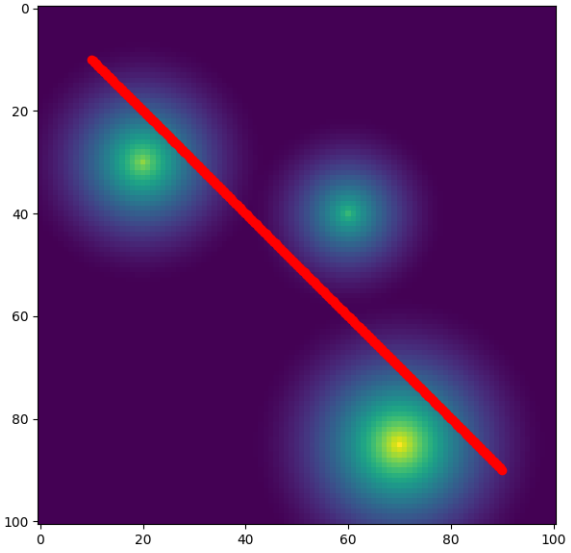
a

Convergence criteria: Take the L2-norm of the difference between the updated path and the original path, if this value is lower than a predefined threshold, the convergence is reached.

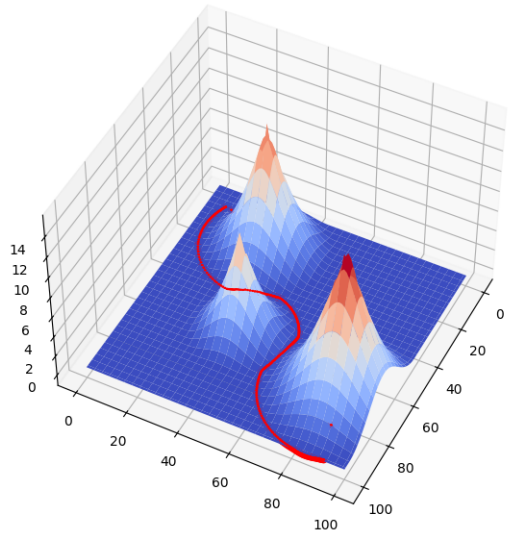
Problem: In this vanilla approach, each point is updated independently. We don't consider its distance from other points on the path. Therefore, an updated point can move really far from the rest of the points, just like the two points close to the starting and ending point in the two images at convergence.

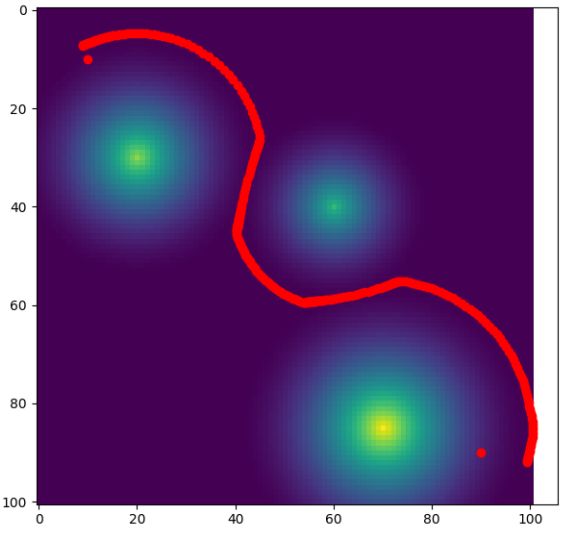
After 1 iteration





At convergence:



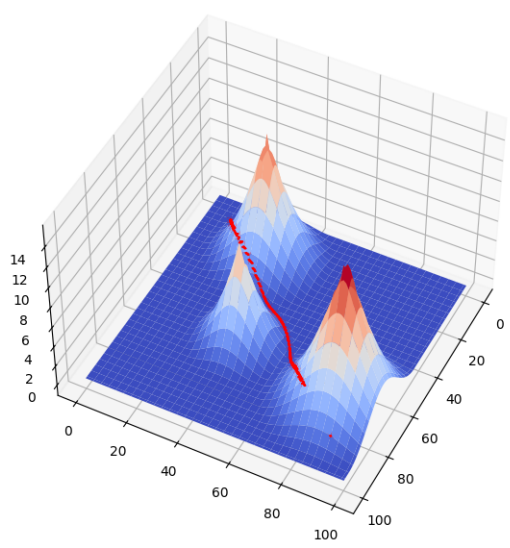


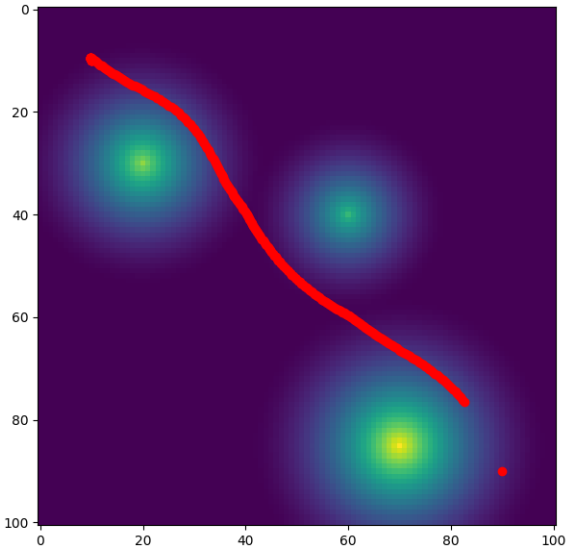
b

Why it doesn't work:

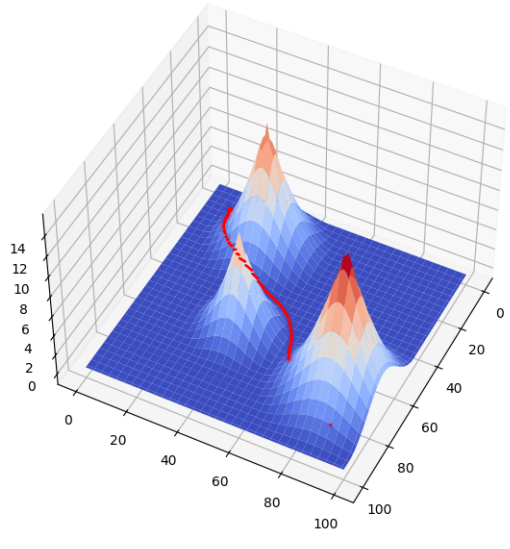
Although we are considering the point's distance from the previous point, we fail to take into account its distance from the next point. Every point now only stays close to the previous one, but can move really far away from the next one. Therefore we see an increasing gap between the end of the path and the target destination, because all points tend to shift towards their previous point.

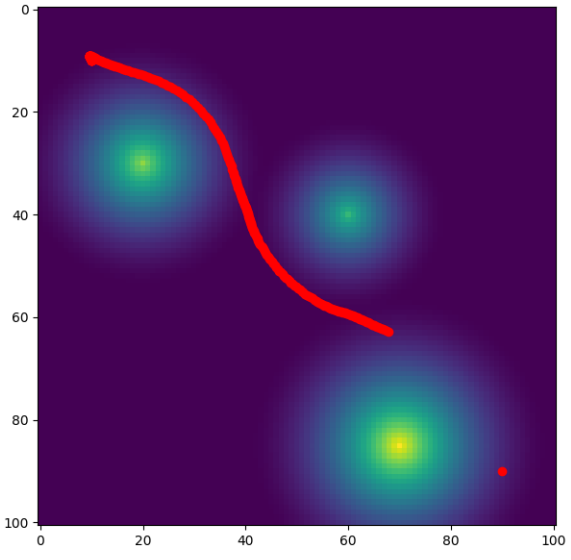
After 100 iterations



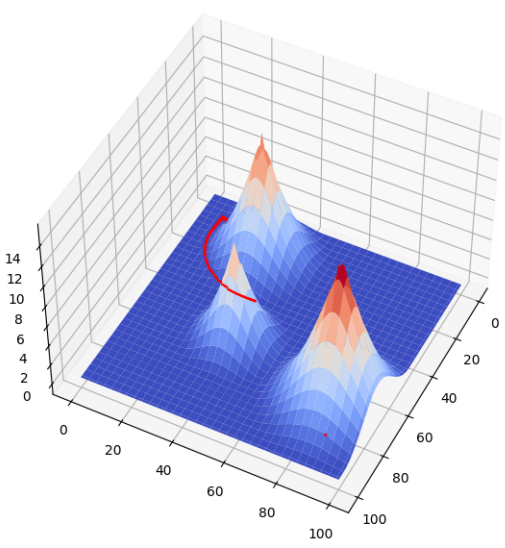


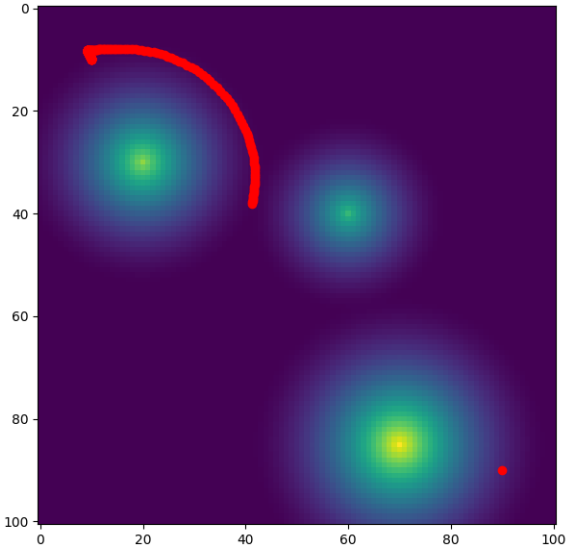
After 200 iterations





After 500 iterations



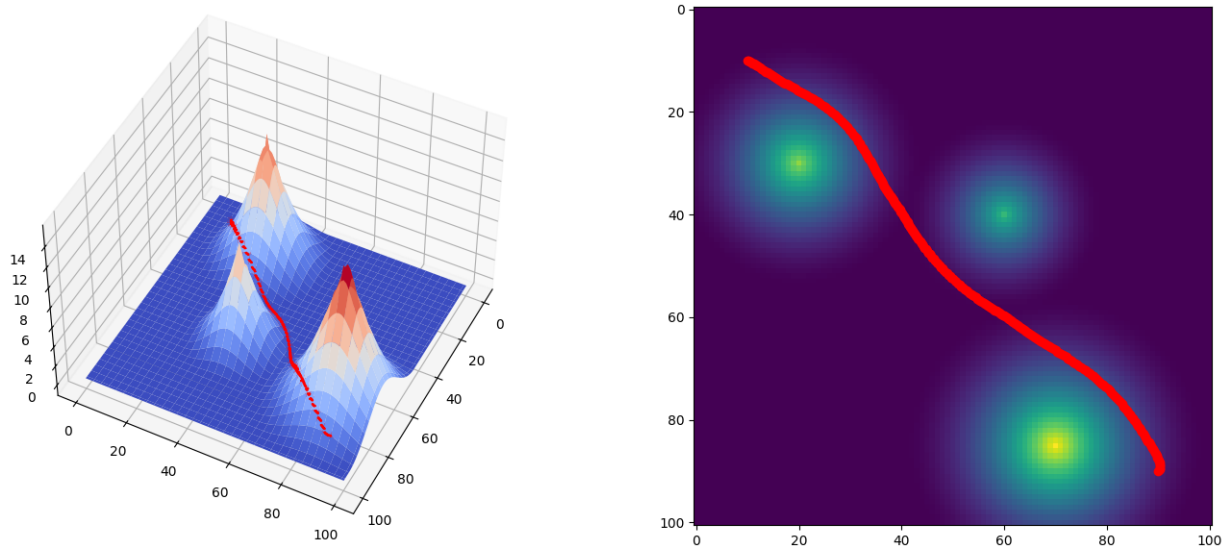


c

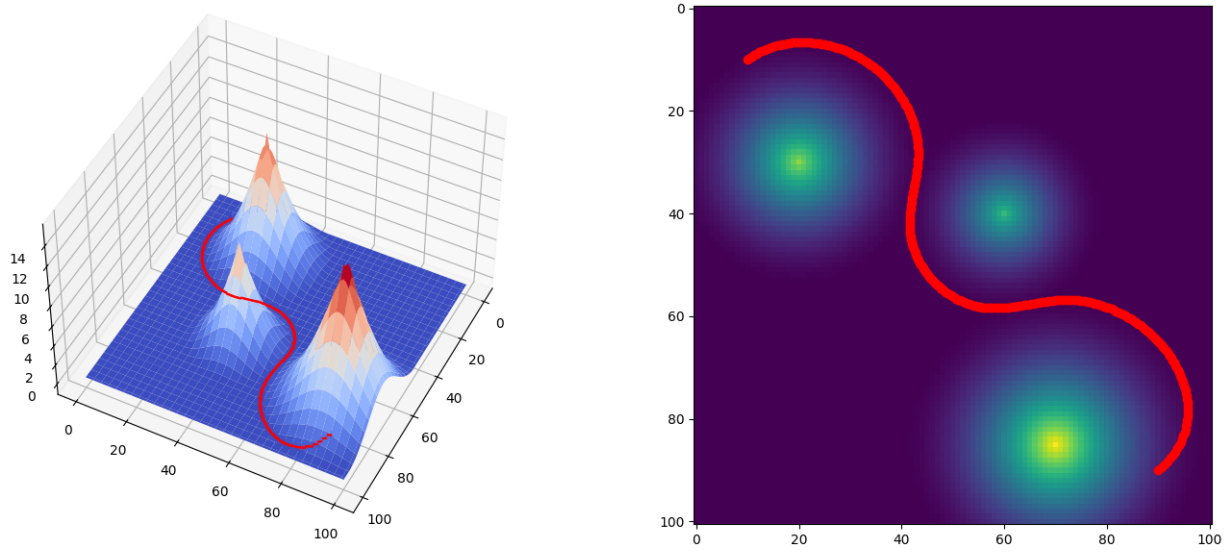
Why this version is better

This version takes into account a point's distance from both its previous and next point. Adding this into the cost to minimize will minimize its distance from both its neighboring points. Therefore, the path does not shift all towards the previous points and produce a gap at the end of the pat. The specified vector field now bonds each neighboring points to each other, thus keeping the path smooth.

After 100 iterations



After 5000 iterations



d

With different initial paths, the final path will not be the same. The algorithm tries to find a path that both minimize the cost and stays rather close to the initial path defined. For example, in this question, the given initial path goes through the middle of the square area, then the optimized path also travels through the middle, turning to avoid obstacles.

However, if we defined a path that travels first along the upper edge to go around the obstacles, and then down to the target in the lower right corner, the optimized path will also follow this trend to travel around the obstacles rather than travelling between them.

This is because we only update the points in the "down hill" direction on the cost function rather than the "up hill". The updates will not move a point over the peaks in the cost function, so the updated path will stay on the same side of an obstacle as the predefined initial path.

Therefore, with a different initial path, the final answer can be different every time.

e

Some strategies

(1) We can let the initial path travel on another side of certain obstacles, because we are given the cost function and can do some simple searches and tests to determine a roughly feasible initial path.

(2) Another way is to do parameter tuning. We can change the weights for the gradient and the weights for the smoothing cost. The step size should be big enough to converge fast, but small enough to not skip over a minima.