Alec Anaya

# The Make or Break Project

This project gave us a dynamical system of nonlinear 2D equations and a corresponding auxiliary function. Our task was to analyze the system from both perspectives, that of the system itself and of the auxiliary function.

$$\dot{x_1} = (1 - x_1^2)x_{2,}$$

$$\dot{x_2} = \frac{x_1}{2} - x_{2.}$$

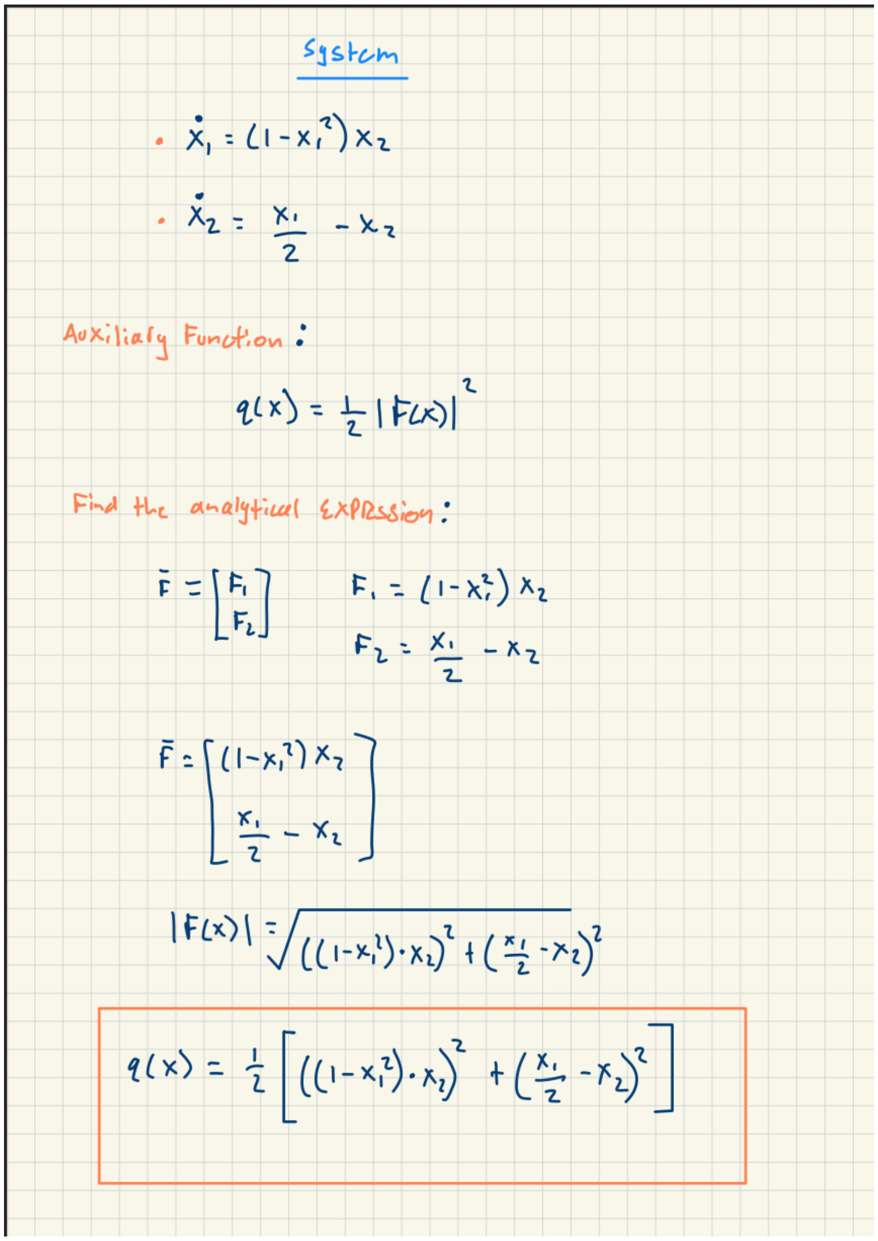
This is the system of equations above, a beautiful one I might add. I just think it's aesthetically pleasing.

**Auxiliary function q(x) Minimization:**

First I needed to analyze the auxiliary function. To do this I needed to understand what this auxiliary function even was. I could see that it was related to each equation in the system by involving the magnitude of the force vector squared multiplied by a factor of one-half as follows.

Define the auxiliary function: $q(x) = \frac{1}{2}|F(x)|^2$

This seemed to me like some type of potential function similar to how a potential function for a harmonic oscillator takes the form of  V(x) = 0.5kx2, but I wasn't sure of the exact meaning. I started with finding the analytical expression for the auxiliary function as follows.



System

- $\dot{x}_1 = (1 - x_1^2) x_2$
- $\dot{x}_2 = \frac{x_1}{2} - x_2$

Auxiliary Function:

$$q(x) = \frac{1}{2} |F(x)|^2$$

Find the analytical expression:

$$\bar{F} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \qquad F_1 = (1 - x_1^2) x_2 \qquad F_2 = \frac{x_1}{2} - x_2$$

$$\bar{F} = \begin{bmatrix} (1 - x_1^2) x_2 \\ \frac{x_1}{2} - x_2 \end{bmatrix}$$

$$|F(x)| = \sqrt{((1-x_1^2) \cdot x_2)^2 + (\frac{x_1}{2} - x_2)^2}$$

$$\boxed{q(x) = \frac{1}{2} \left[ ((1-x_1^2) \cdot x_2)^2 + (\frac{x_1}{2} - x_2)^2 \right]}$$

*Figure 1. The analytical expression for q(x) in terms of the system*

With the analytical expression in hand, I could do now some work. The goal and relation of this function to my project was to locate the minima of the function. In doing so by inspection, I automatically thought about where the function equaled zero. In looking at what values of $x1$ and $x2$ would make both quadratic terms zero and result in a minimum. It became quickly apparent that I should expect three minima at (-1,-0.5), (0,0), and (1,0.5). The next step would be to confirm this. For this project, the given method to do this was gradient descent. To do this I generated 25 random points in the $(x1, x2)$ plane; specifically in the range of [-1.5;1.5] for $x1$ and [-0.8;0.8] for $x2$. At the start for each of the points the gradient of q(x) would be calculated, showing the direction of the fastest rate of change of the function. Since we are trying to locate the minimums of the auxiliary function, we take a step size in the opposite direction the gradient is pointing and then iteratively repeat this process until we land at a minimum. The contour plot q(x) and the trajectories from the gradient descent method are shown as follows.

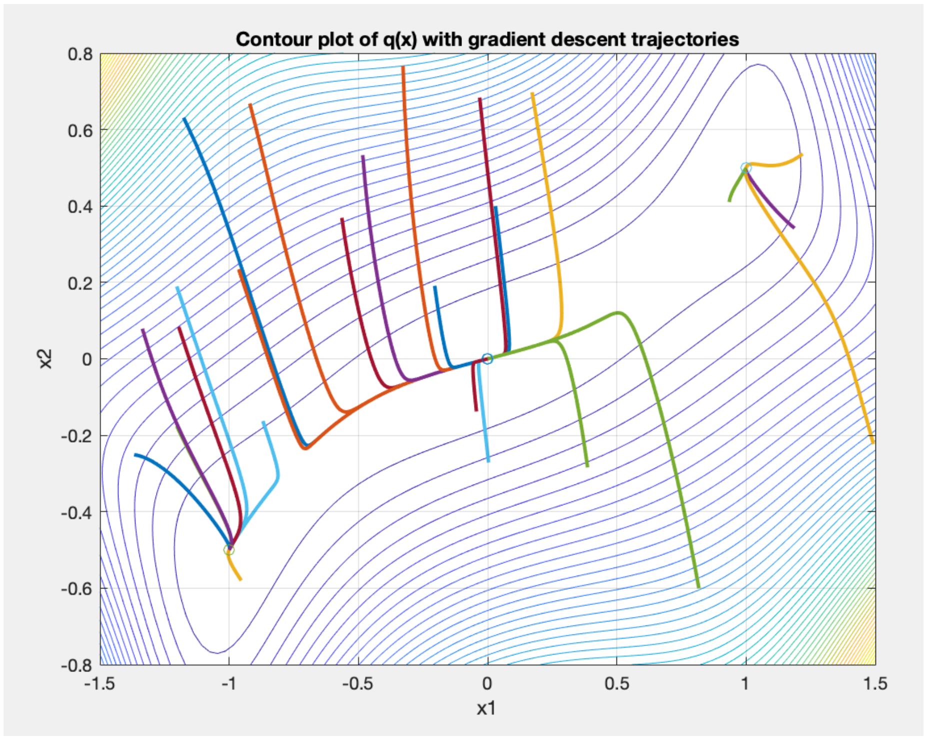


*Figure2. Contour plot and gradient descent trajectories*

From the contour plot, we know the areas of the function that are the steepest based on how close the contour lines are together. The trajectories are represented by the different colored lines that start in the contour lines and end at one of the three minima. We see from this method that we do get three minima at the points expected by inspection; (-1,-0.5), (0,0), and (1,0.5). More analysis on this will come later but for now, I just wanted to show the solution and confirm that this method leads these randomly selected points on a journey to the minimums of the function.

**Dynamics of the System:**

Next, we will analyze the dynamics of the system itself, meaning we will observe what happens in finding solutions to the system of equations and see what the solutions look like. Our method for doing this will be fourth-order runge kutta. We will again generate 25 random points in the $(x1, x2)$ plane but in a square defined by [-2;2] and [-2;2]. The point of this is to essentially create an initial value problem for the system at the randomly generated 25 points. Then follow the iteration of the RK method to see what the solutions look like. If given enough iterations, we should get a good look at what the dynamics of the system enforce in the long run. Meaning we can identify if there is convergence or divergence at any points or identify if the function is stable or unstable.  The following image shows the contour plot of q(x) along with the dynamic trajectories found from solving the system.

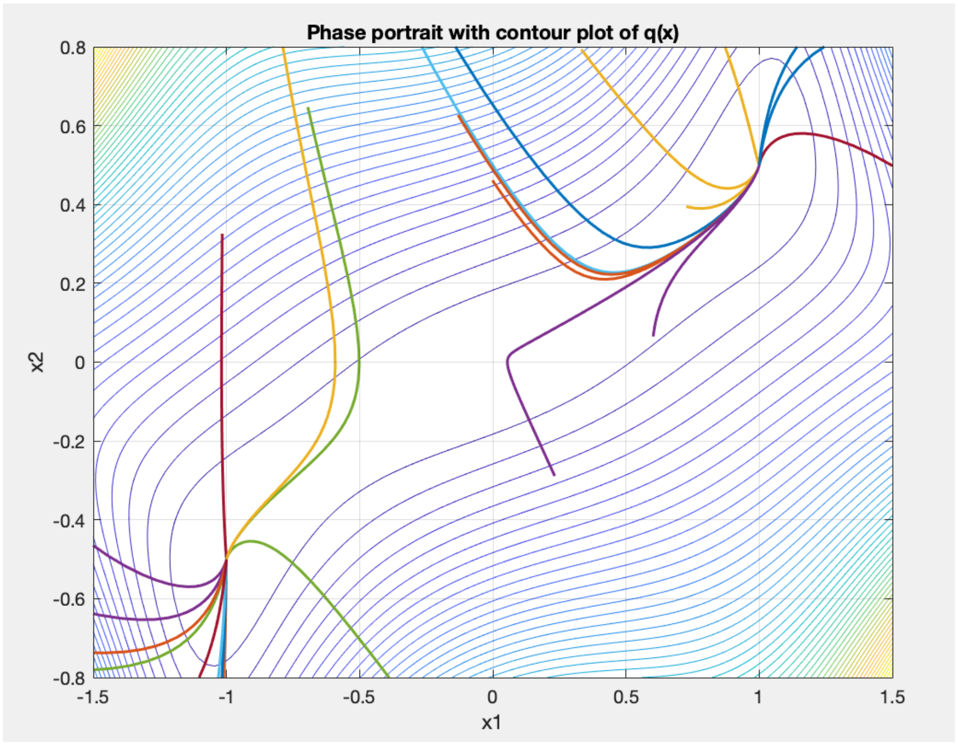


*Figure 3. Contour plot and runge kutta trajectories*

Again, the colored lines are the trajectories. They start somewhere in the square of [-2;2] and [-2;2] and end up at the points (-1,-0.5) and (1,0.5). Interestingly, the trajectories end at two of the same points as the minimums of the auxiliary function.

**Analysis:**

Now that we've displayed the results of the computation, we can analyze the results. So, back to this auxiliary function. After doing some research I did find that it was representative of the potential or a similar Lapynov function that allows us to study the system from a different aspect. It gives insight into the system behavior like in this case, the equilibrium or fixed points of the system. Starting with the solutions shown in *Figure 2,* we can see three clear minimums of the auxiliary function. These points show that if we follow the gradient descent to the stopping point we end here. These points can be understood as points where forces are equal to zero or where the system comes to rest, if we are talking about a physical system. Or at least that's the way that I best understand it. For the runga kutta solutions shown in *Figure 3,* there are points that the trajectories go towards and come to rest at as well. The dynamics of this system over time influence different initial points to the same boundary point, which is extremely interesting. But, the most interesting result to me comes from comparing the two solutions in *Figure 2* and *Figure 3.* In comparing the two figures, we see that the trajectories end up at the points (-1,-0.5) and (1,0.5) but that the runga kutta solutions don't converge at the other minimum of the auxiliary function at (0,0). I was extremely confused by this. I felt that if this was a point where the forces equal zero, or the system comes to rest, then why does this point seem to repel the RK trajectories? We can see in *Figure 3* that there are trajectories that even head towards (0,0) and then seem to diverge at the last second. It made me think about what is going on here and I did what I could to try and figure it out. I had chosen random initialization points for the RK function and kept running the system because I figured there had to be at least one initial point with initial conditions such that the system would come to rest at (0,0) but I could not find one. I eventually realized through some research that these two methods are solving completely different things even though there is a relationship. The gradient descent is specifically looking for minimums of the potential function and the RK method is seeing how the system evolves over time. This is the key difference. Because even though it is likely that the system evolves to the minimum of the potential function it is not always the case. I wondered if my GD solutions were incorrect and that (0,0) was a saddle point that my code didn't account for. So I solved for the eigenvalues of the Hessian matrix for the auxiliary function at the point (0,0) and found them to be 0.1172 and 2.1328. Since both eigenvalues are positive the point is indeed a minimum for the potential function. Next, I did the same thing for the dynamical system by finding the eigenvalues of the Jacobian matrix. I used the Jacobian in this case for the differential equations because this is similar to the second derivative test but with respect to stability. I found the eigenvalues to be -1.3660 and 0.3660. Since one of the values is positive and the other negative, this point in the system is unstable or effectively a saddle point. This answered all my questions. The solutions of the differential equations want to go towards stability. This project shows that we can start at any point in this range and solve the initial value problems and the solutions with be attracted to certain points concerning stability. These attractor points can often be related to the fixed points, where the forces equal zero, but don't always have to. What matters most to the dynamic system at the end of the day is stability. This made me wonder if I make decisions in the same way. Do I tend to always go towards stability no matter my initial starting point? I would say personally that I think I do even though stability is subjective. Even when following a path that should seem like it's heading towards stability I quickly change direction when realizing this is not the case. So do differential equations not take risks?  I like to think I do but do I really? Is it probable that you will skim through this and just give me a decent grade, maybe? But if I were to bank on this then that would be a trajectory towards instability. What I learned from this project is that I want to learn whether things can be probable and unstable at the same time. Of course, I learned the other things already mentioned and am very happy with the things this project made me think about. Thank you. My code will be provided below.

## Table of Contents

## Hessian matrix eigenvalues

```
syms x1 x2

% auxiliary function q(x)
q = 0.5 * (x2^2) * ((x1^2 - 1)^2) + 0.125 * ((x1 - 2*x2)^2);
% making the hessian matrix of the auxilary function
Hessian_q = hessian(q, [x1, x2]);
Hessian_at_zero = subs(Hessian_q, [x1, x2], [0, 0]);
eigenvalues_Hessian = eig(Hessian_at_zero);
disp('The Hessian matrix of q(x) at the point (0,0) is:');
disp(Hessian_at_zero);

% show the eigenvalues of the Hessian matrix
disp('eigenvalues of the Hessian matrix at (0,0) are:');
disp(double(eigenvalues_Hessian));
```

```
The Hessian matrix of q(x) at the point (0,0) is:
[ 1/4, -1/2]
[-1/2,    2]

eigenvalues of the Hessian matrix at (0,0) are:
    0.1172
    2.1328
```

## Finding the eigenvalues of the jacobian

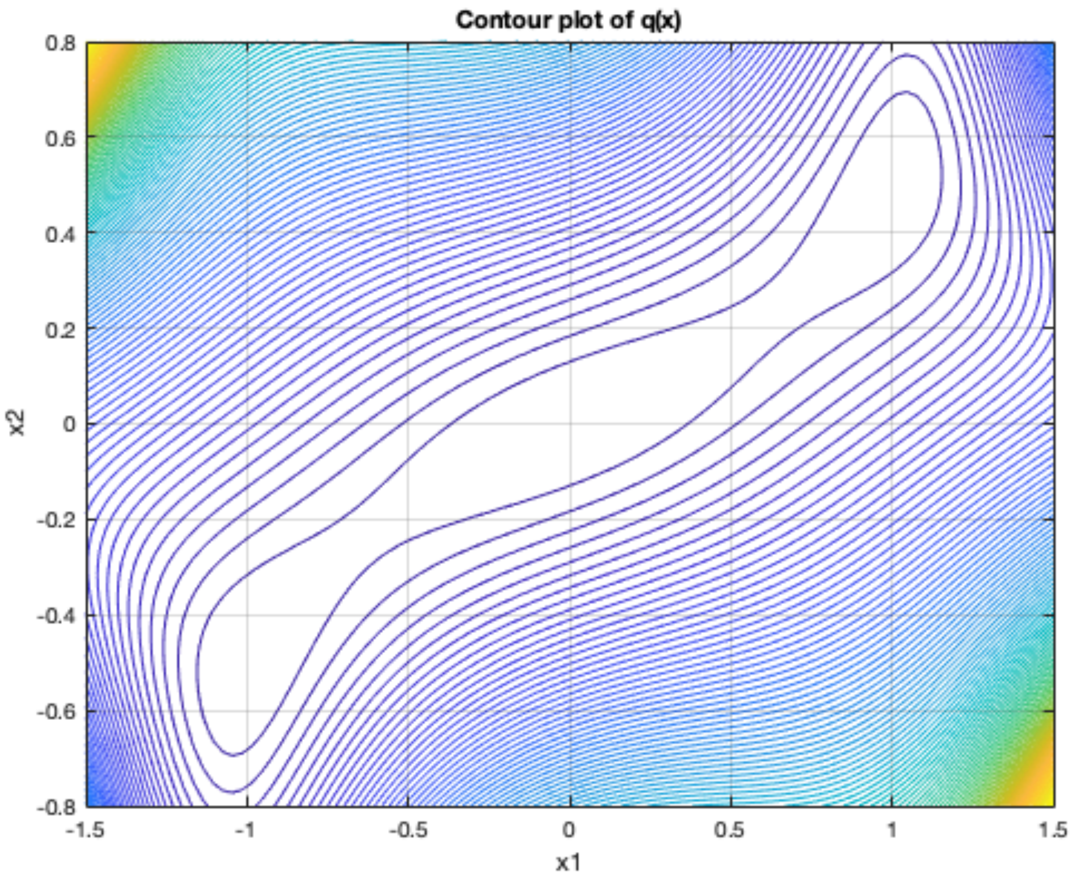
```
syms x1 x2

% system of equations
F = [(1 - x1^2)*x2; (x1/2) - x2];

% find the eigenvalues of the Jacobian matrix of the system by setting it
% up and evaluating at the point (0,0)
J = jacobian(F, [x1, x2]);

J_at_eq = subs(J, [x1, x2], [0, 0]);


eigenvalues = eig(J_at_eq);
```

```
disp('Jacobian matrix at the equilibrium point (0,0) is:');
disp(J_at_eq);

% Display the eigenvalues
disp('eigenvalues of the Jacobian at (0,0) are:');
disp(double(eigenvalues));
```

```
Jacobian matrix at the equilibrium point (0,0) is:
[   0,   1]
[1/2, -1]

eigenvalues of the Jacobian at (0,0) are:
   -1.3660
    0.3660
```

## Contour plot of q(x)

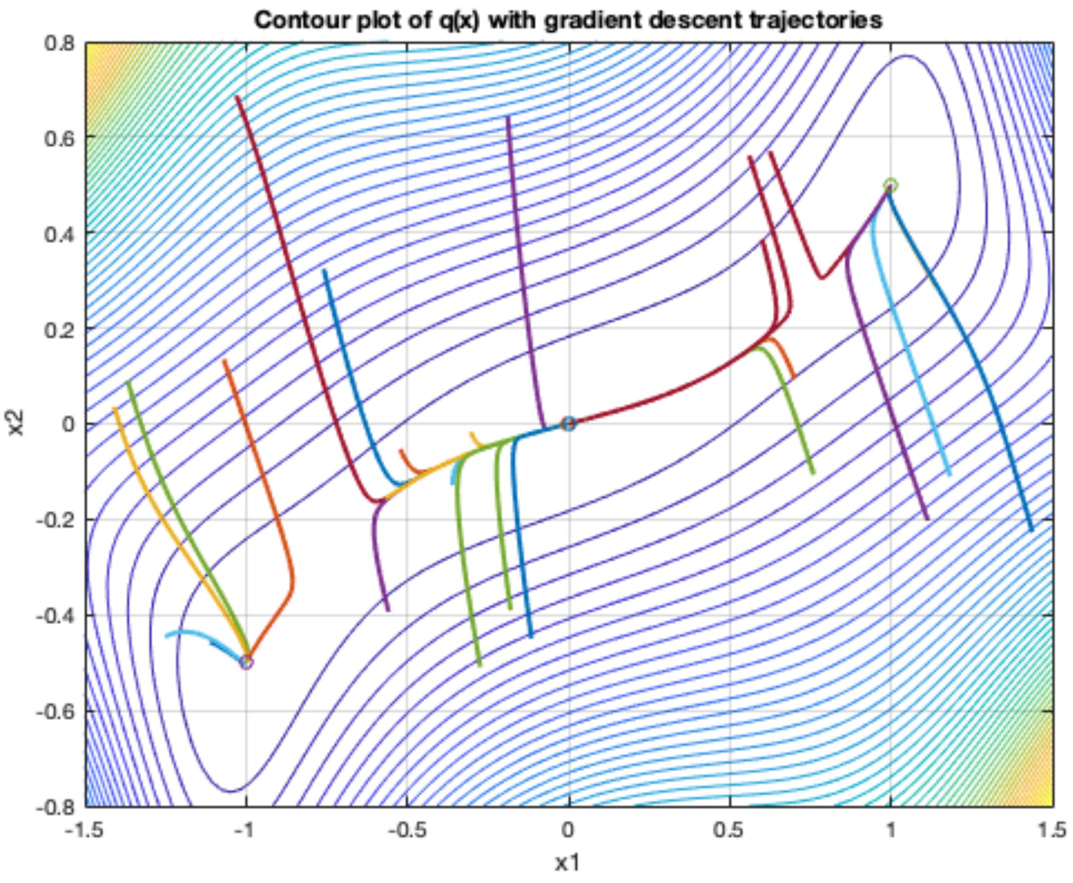
```
x1_range = linspace(-1.5, 1.5, 100);
x2_range = linspace(-0.8, 0.8, 100);
[X1, X2] = meshgrid(x1_range, x2_range);
Q = 0.5 * (X2.^2) .* ((X1.^2 - 1).^2) + 0.125 * ((X1 - 2*X2).^2);
figure;
contour(X1, X2, Q, 100);
xlabel('x1');
ylabel('x2');
title('Contour plot of q(x)');
grid on;
```



## Gradient descent method

```
alpha = 0.1; % Learning rate/ or step size
tolerance = 1e-6;
max_iterations = 1000;

% generating random starting points for the
num_trajectories = 25;
starting_points = [-1.5 + 3*rand(1, num_trajectories); % x1 starting points
                   -0.8 + 1.6*rand(1, num_trajectories)]; % x2 starting points

% gradient of q(x)
grad_q = @(x1, x2) [x1*x2^2*(x1^2 - 1) + 0.125*(x1 - 2*x2);
                    x2*(x1^2 - 1)^2 - 0.25*(x1 - 2*x2)];

% ranges for x1 and x2 in the (x1,x2) plane
x1_range = linspace(-1.5, 1.5, 100);
x2_range = linspace(-0.8, 0.8, 100);

[X1, X2] = meshgrid(x1_range, x2_range);

% auxiliary function q(x)
```

```
Q = 0.5 * (X2.^2) .* ((X1.^2 - 1).^2) + 0.125 * ((X1 - 2*X2).^2);
% contour plot
figure;
contour(X1, X2, Q, 50);
hold on;

% plotting each gradient descent trajectory and keeping track of each
% postistion
for j = 1:num_trajectories

    x = starting_points(:, j);
    x_history = x;

    % creating the loop for the gradient descent method
    for i = 1:max_iterations
        grad = grad_q(x(1), x(2));
        x_new = x - alpha * grad;
        x_history = [x_history, x_new];

        % Check for convergence
        if norm(grad) < tolerance
            break;
        end
        x = x_new;
    end


    plot(x_history(1,:), x_history(2,:), '-', 'LineWidth', 2);
    plot(x_history(1,end), x_history(2,end), 'o');
end


xlabel('x1');
ylabel('x2');
title('Contour plot of q(x) with gradient descent trajectories');
grid on;
hold off;
```

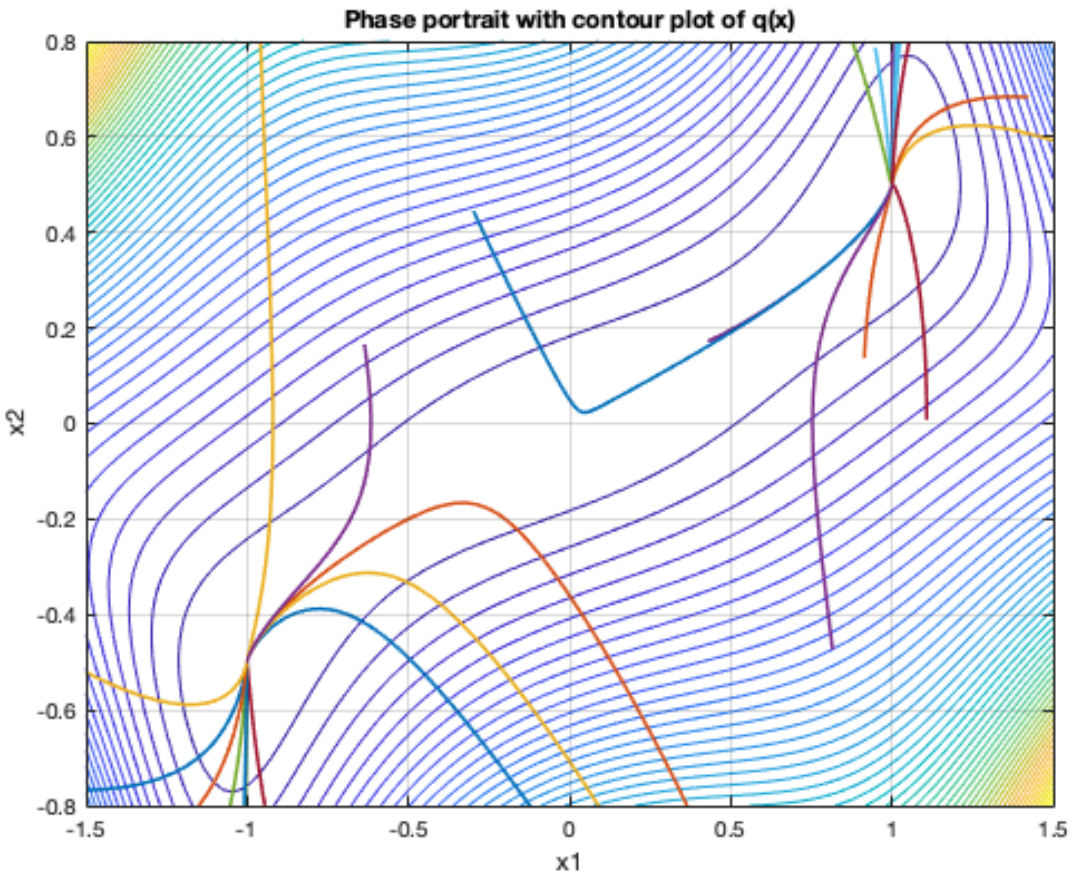


## Runge Kutta

auxiliary function q(x)

```
q = @(x1, x2) 0.5 * (x2.^2) .* ((x1.^2 - 1).^2) + 0.125 * ((x1 - 2*x2).^2);


x1_range = linspace(-1.5, 1.5, 100);
x2_range = linspace(-0.8, 0.8, 100);
[X1, X2] = meshgrid(x1_range, x2_range);


Q = q(X1, X2);

figure;
contour(X1, X2, Q, 50);
hold on;
grid on;

% start of solving using RK
tspan = [0, 20];

% Number of random initial conditions
```

```
num_trajectories = 25;

% random initial conditions in the region [-2,2] x [-2,2]
initial_conditions = 4*rand(2, num_trajectories) - 2;
% solve thesystem for each set of initial conditions and plot the trajectories
for i = 1:num_trajectories
    % initial conditions for each new point
    x0 = initial_conditions(:, i);

    %  4th order Runge Kutta using ode45
    [t, x] = ode45(@system_odes, tspan, x0);
    plot(x(:,1), x(:,2), 'LineWidth', 1.5);
end


xlabel('x1');
ylabel('x2');
title('Phase portrait with contour plot of q(x)');
xlim([-1.5 1.5]);
ylim([-0.8 0.8]);
hold off;

% defining the system of euations to solve
function dx = system_odes(t, x)
    dx = zeros(2,1);
    dx(1) = (1 - x(1)^2) * x(2);
    dx(2) = x(1)/2 - x(2);
end
```



*Published with MATLAB® R2021b*