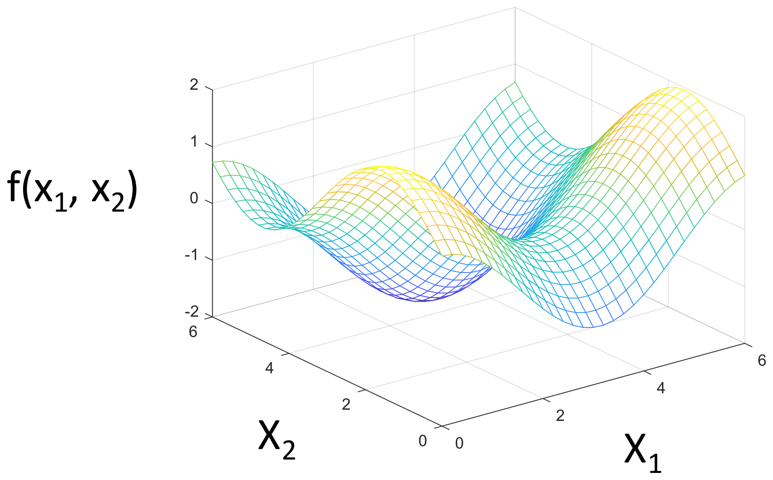
Shubh Raval

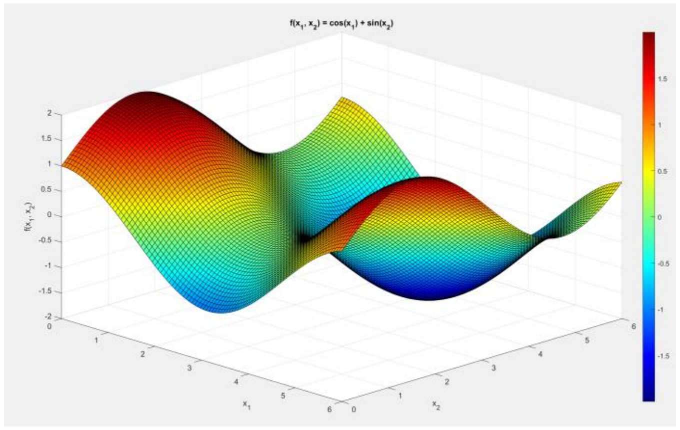
ME 6103 Engineering Optimization

Spring 2025, Dr. Carolyn Conner Seepersad

Assignment 4: Experimentation and Metamodeling

Consider the 2-dimensional function: $f(\boldsymbol{x}) = \cos(x_1) + \sin(x_2)$ for $0 \le x_n \le 6$

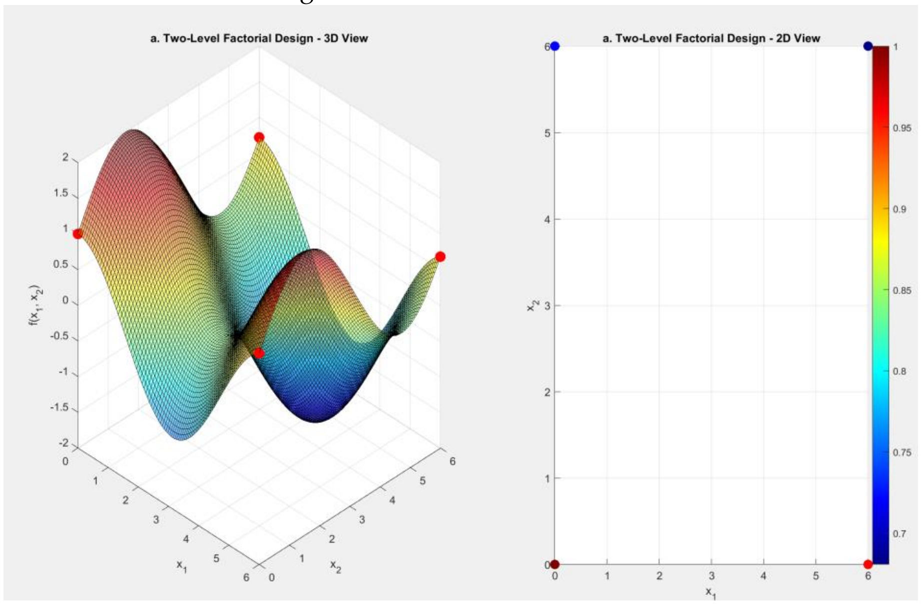




Function Recreated in Matlab

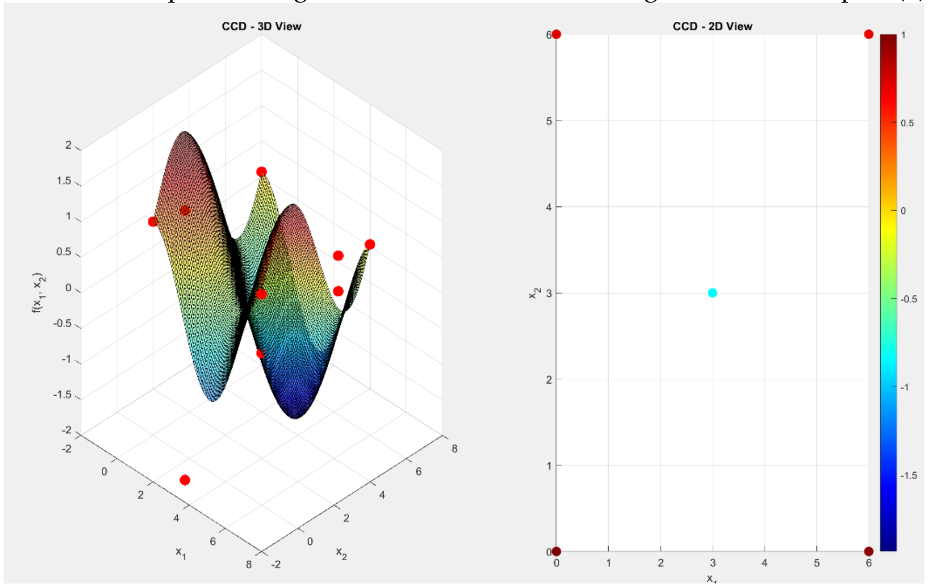
(1) Design and conduct computer experiments on the test function using the following experimental designs:

a. A two-level factorial design.



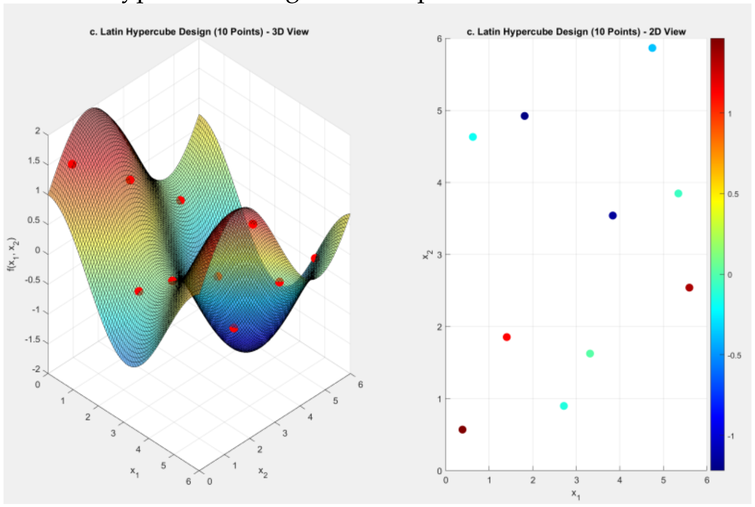
This was done using the factorial combination of the upper and lower bounds of the design variable which was then plotted both on the mesh and shown on a 2d layout

b. A central composite design, based on the factorial design conducted in part (a).



From the factorial experimentation alpha was calculated as sqrt(2) which was then scaled proportionally based on the upper and lower bounds to generate the new points used along with a center point.

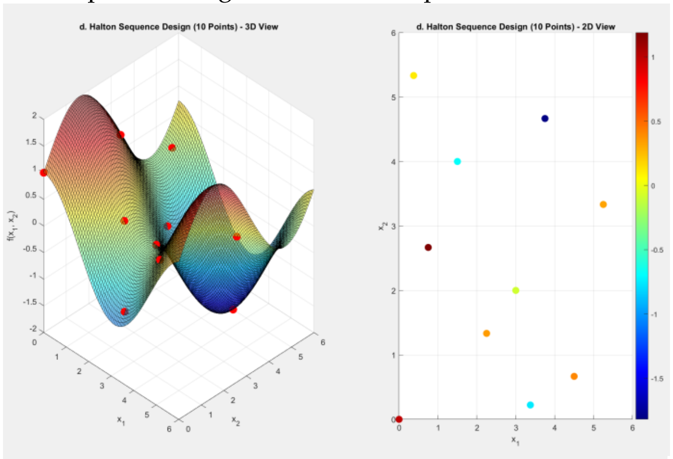
c. A Latin Hypercube design with 10 points.



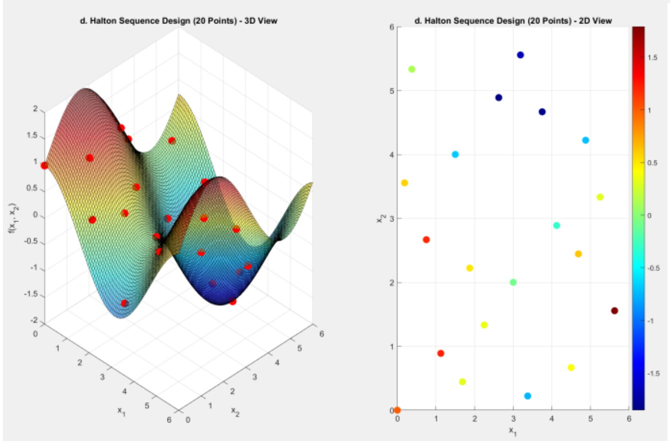
The latin hypercube was implemented with the lhsdesign toolbox from matlab for our 2d design with 10 points desired, each selected point was then evaluated and plotted.

d. Two Halton sequence designs: one with 10 points and one with 20.

i.



ii.



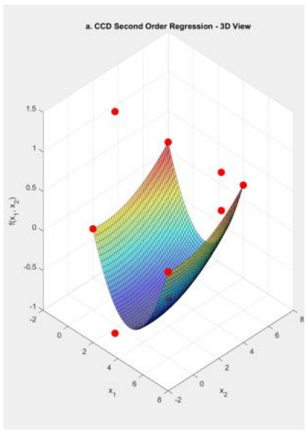
The halton sequence design was done using the haltonset toolbox, where first the set was 2 dimension object was create, then populated with the 20 points, and then these points from a unit base to one based on the upper and lower bound. An initial issue I had with the haltonset was the points not being generated after x1 =

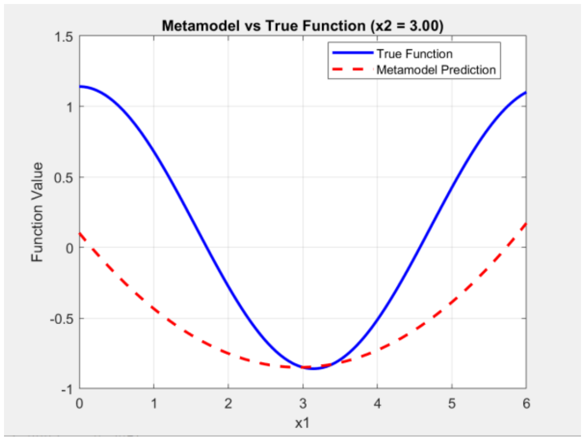
3, which I realized was due to passing in improper parameters to the haltonset object. Specifically from the matworks documentation I had originally implemented the skip and leap parameters caused points to be lost and not properly cover the full design space.

(2) Create the following metamodels based on the experiments in part (1). Plot each metamodel in a 3-d plot (like the one shown above). Quantify the accuracy of each metamodel with a uniform grid of 25 test points and the relative average and relative maximum error metrics introduced in class.

For all the following a uniform grid of the 25 points was fed into the created predictive function and were then evaluated to create the mesh. Additionally similar to lecture a 2d plot to show error was made by hold x2 = 3 was created to show the error between the true function and the prediction. Finally for each the RAAE and RMAE was calculated to determine error against the true function. This was done by creating a function that accepted the uniform gird points, the predictive function and the test responses generated by evaluating the grid. So the absolute error was found between values. Then sigma was calculated and use to the determine the relative error, then the average of these was RAAE, and the max was the RMAE.

a. A second order regression based on the data from (1b).



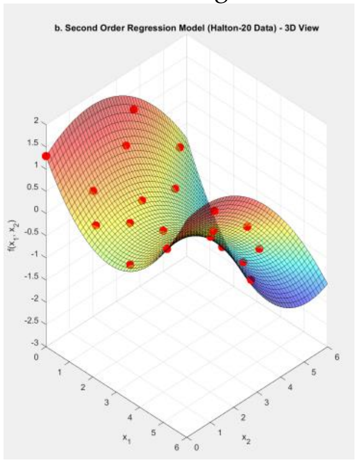


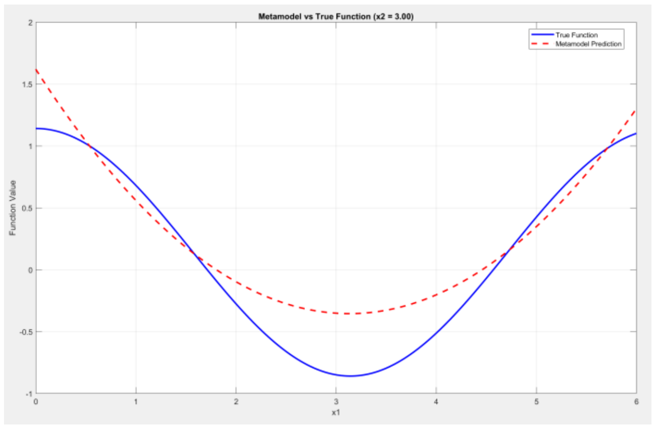
This was implemented by passing the ccd data to the fitlm function to create a function called predict that used the quadratic regression generated to evaluate the points from the uniform grid. Then using the error function the RMAE and the RAAE were calculated:

Relative Average Error (RAAE): 0.7542

Relative Maximum Error (RMAE): 1.9613

b. A second order regression based on the data from (1d).



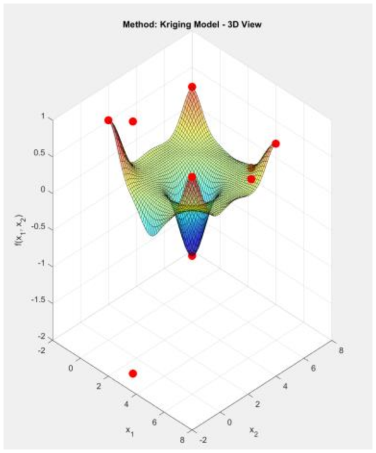


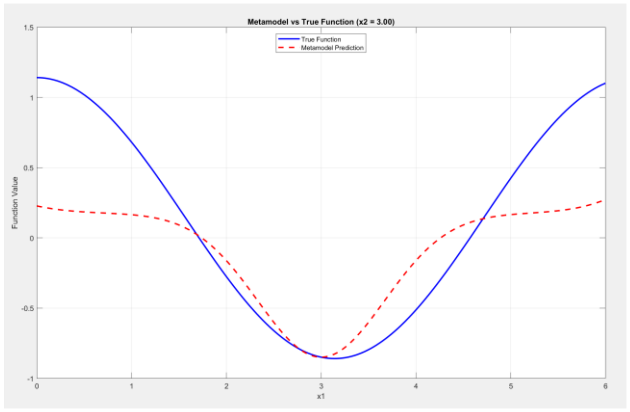
For generating this metamodel a similar implementation was used with fitlm and the RMAE and RAAE were determined to be:

Relative Average Error (RAAE): 0.6431

Relative Maximum Error (RMAE): 2.2540

c. A kriging model based on the data from (1b).[1] Use a zero order polynomial and a Gaussian correlation function.





This was implemented via the kriging toolbox provided in class and essentially following along with the basic script give. The major change was it was turned into a function so that its easier to use in this broader scope. Finally one issue that occurred with kriging was the multiple design sites error based on duplicate results from evaluation, to handle this I simple kept the first unique instance and discarded the other duplicate instances.

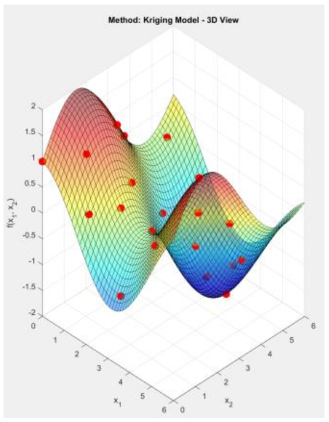
The error found was:

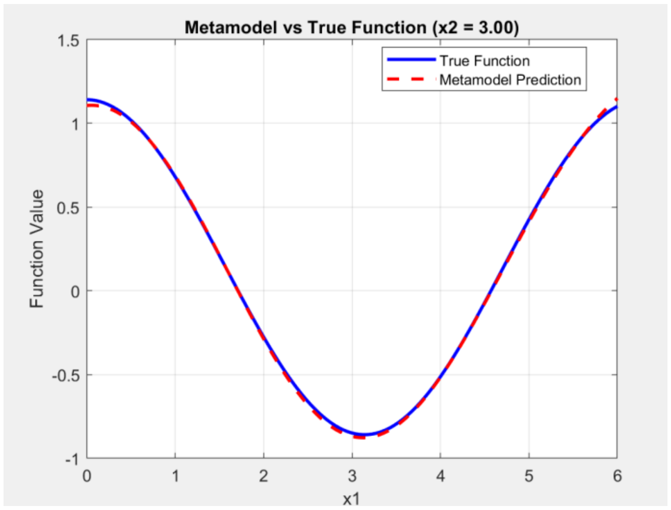
Relative Average Error (RAAE): 0.6311

Relative Maximum Error (RMAE): 2.0466

[1] Use the DACE toolbox for Matlab, available in a zip file on Canvas in the metamodeling module. Start with the .m file labeled "SimpleKrigingforME397.m"

d. A kriging model based on the data from (1d). Use a zero order polynomial and a Gaussian correlation function.





This called the kriging function I wrote in my script and passed in the appropriate data where similarly to handle duplicate points the first unique was kept. Overall it is visually and quantitatively apparent that this combination of halton 20 and the kriging meta model has produced the most accurate result. With the error being:

Relative Average Error (RAAE): 0.0630

Relative Maximum Error (RMAE): 0.4742

e. Compare and contrast the different metamodels and explain any trends you observe.

Overall there were some very interesting takeaways from the metamodels and how they interacted with the method of experimentation. Generally the best combination of experimental evaluation and meta model was kriging and the Halton sequence, the model had a relative average error of 0.063 that was an order magnitude less than the other methods which were between 0.6 - .75, and its relative maximum error of .4742 was less than the average error of the other models approx. 2.

This combination makes sense to be the best overall given that the Halton sequence with 20 points will have produced the best distribution of experimental values to represent the original function. When this is coupled with the kriging method which can well capture non-linearities in the experimental data it allows for the design to be well represented and the benefit of having such a well encapsulated distribution put to use. Now looking at the CCD with kriging we see the response is likely the worst since the limited points results in the metamodel responding too sharply since the boundary and center points are the only provided to the model. Hence the meta model is very poorly related to the true function.

Looking at the 2nd order regression its seen that the performance is still relatively poor however, with the halton-20 the models maximum error is worse. The second order model has an inability to capture any higher order relations that might be present in the better distributed data set, so its performance is not much better enhanced the halton. This is directly noticeable when contrasted to the results from CCD and the 2nd order regression. Which ended up producing a lower maximum error and only a slightly worse average error than the halton-20 input. This pairing makes sense it does not provide anything more than the 2nd order regression would be able to represent until like the halton-20 experimental set. So all 2nd order relationships through the boundaries and the middle of the design are well captured.

Holistically, the choice of design experimentation method is then tied to the metamodeling technique to be used. Something that focuses on space filling in a well representing distribution is a suitable candidate for a more complex metamodel like kriging where the gaussian process better tracks the full non-linear relationships that might be captured. However for a faster and simpler method like a 2nd order regression there isn't a notable benefit for such a broad coverage and it would be better paired to a ccd or factorial design exploration that would focus on representing the boundaries and center given the limitations of the metamodel.

Code:

```
%% Homework 4 Shubh Raval


% Create points
x1_range = linspace(0, 6, 100);
x2_range = linspace(0, 6, 100);
[X1, X2] = meshgrid(x1_range, x2_range);

% Create the Function
F = @(x1,x2) cos(x1) + sin(x2);
Fplot = cos(X1) + sin(X2);

% Create 3D surface plot of the function
figure;
surf(X1, X2, Fplot);
colormap('jet');
colorbar;
title('f(x_1, x_2) = cos(x_1) + sin(x_2)');
xlabel('x_1');
ylabel('x_2');
zlabel('f(x_1, x_2)');
grid on;
view(45, 30);
create = true;
true_func = @(x) F(x(1), x(2));
fixed_x2 = 3;  % for error checking
%% Create Uniform Grid of 25 Test Points
lb = [0, 0];
ub = [6, 6];
[X1_grid, X2_grid] = meshgrid(linspace(lb(1), ub(1), 5), linspace(lb(2), ub(2), 5));
test_points = [X1_grid(:), X2_grid(:)];
test_responses = zeros(25, 1);
```

```
    for i = 1:25
        test_responses(i) = F(test_points(i,1), test_points(i,2));
    end

    %% PROBLEM 1
    disp('CREATE EXPERIMENTAL DATA SETS FOR PROBLEM 1')

    [ccd, f_eval_ccd,halton20, f_eval_halton20] = createExperimentalsets(create);


    %% PROBLEM 2
    disp('BEGIN META MODELING FOR PROBLEM 2')


    %% a. Second Order Regression Model based on CCD Data
    mdl = fitlm(ccd, f_eval_ccd, 'quadratic');

    % Create prediction function from the model
    predict_fitlm = @(x) predict(mdl, x);

    [X1_vis, X2_vis] = meshgrid(linspace(lb(1), ub(1), 50), linspace(lb(2), ub(2), 50));
    reg_fitlm_vis = zeros(50, 50);
    for i = 1:50
        for j = 1:50
            reg_fitlm_vis(i,j) = predict_fitlm([X1_vis(i,j), X2_vis(i,j)]);
        end
    end

    ccd_predictions_fitlm = predict(mdl, ccd);

    plotDesignPoints(X1_vis, X2_vis, reg_fitlm_vis, ccd, ccd_predictions_fitlm, 'a. CCD Second
Order Regression');
    [RAE_ccd_fitlm, RME_ccd_fitlm] = evaluateMetamodel(predict_fitlm, test_points,
test_responses, fixed_x2, true_func);

    %% b. Second Order Regression Model based on Halton-20 Data using fitlm
    h_20_mdl = fitlm(halton20, f_eval_halton20, "quadratic");

    % Create prediction function from the model
    predict_h_20 = @(x) predict(h_20_mdl, x);

    reg_fith20_vis = zeros(50, 50);
    for i = 1:50
        for j = 1:50
            reg_fith20_vis(i,j) = predict_h_20([X1_vis(i,j), X2_vis(i,j)]);
        end
    end
    h_20_predictions = predict(h_20_mdl, halton20);

    plotDesignPoints(X1_vis, X2_vis, reg_fith20_vis, halton20, h_20_predictions, 'b. Second Order
Regression Model (Halton-20 Data)');

    [RAE_h20_fitlm, RME_h20_fitlm] = evaluateMetamodel(predict_h_20, test_points,
test_responses, fixed_x2,true_func);

    %% c. Kriging Model based on CCD Data using 0 order polynomial and Gaussian Fit
    d_ccdmodel = buildKrigingModel(ccd, f_eval_ccd, X1_vis, X2_vis);
    predict_kriging_ccd = @(x) predictor(x, d_ccdmodel);
    [RAE_ccd_kriging, RME_ccd_kriging] = evaluateMetamodel(predict_kriging_ccd,
test_points, test_responses,fixed_x2,true_func);

    %% d. Kriging Model based on Halton 20 Data using 0 order polynomial and Gaussian Fit
    d_h_20model = buildKrigingModel(halton20, f_eval_halton20, X1_vis, X2_vis);
    predict_kriging_h20 = @(x) predictor(x, d_h_20model);
    [RAE_h20_kriging, RME_h20_kriging] = evaluateMetamodel(predict_kriging_h20,
test_points, test_responses,fixed_x2,true_func);


    function [RAAE, RMAE] = evaluateMetamodel(predict_func, test_points, test_responses,
fixed_x2,true_func)

      test_predictions = zeros(size(test_points,1),1);
      for i = 1:size(test_points,1)
        test_predictions(i) = predict_func(test_points(i,:));
      end

      % Calculate errors
      errors = test_predictions - test_responses;
      abs_errors = abs(errors);

      response_std = std(test_responses);
      rel_errors = abs_errors / response_std;

      RAAE = mean(rel_errors);  % Relative Average Error
      RMAE = max(rel_errors);   % Relative Maximum Error


      fprintf('\n======= Metamodel Accuracy Metrics =======\n');
      disp(RAAE);
      disp(RMAE);


      x1_range = linspace(min(test_points(:,1)), max(test_points(:,1)), 200)';
      x2_const = fixed_x2 * ones(size(x1_range));
      input_grid = [x1_range, x2_const];
      true_vals = arrayfun(@(i) true_func(input_grid(i,:)), 1:length(x1_range))';
      pred_vals = arrayfun(@(i) predict_func(input_grid(i,:)), 1:length(x1_range))';

      % Plotting
      figure;
      plot(x1_range, true_vals, 'b-', 'LineWidth', 2); hold on;
      plot(x1_range, pred_vals, 'r--', 'LineWidth', 2);
      legend('True Function', 'Metamodel Prediction', 'Location', 'Best');
      xlabel('x1');
      ylabel('Function Value');
      title(sprintf('Metamodel vs True Function (x2 = %.2f)', fixed_x2));
      grid on;

    end
    function plotDesignPoints(X1, X2, Fplot, designPoints, f_eval, title_text)
      figure;

      % First subplot: 3D view
      subplot(1, 2, 1);
      surf(X1, X2, Fplot, 'FaceAlpha', 0.5);
      hold on;
      scatter3(designPoints(:,1), designPoints(:,2), f_eval, 100, 'r', 'filled');
      colormap('jet');
      title([title_text, ' - 3D View']);
      xlabel('x_1');
      ylabel('x_2');
      zlabel('f(x_1, x_2)');
      grid on;
      view(45, 30);

      % Second subplot: 2D view
      subplot(1, 2, 2);
      scatter(designPoints(:,1), designPoints(:,2), 80, f_eval, 'filled');
      colorbar;
      title([title_text, ' - 2D View']);
      xlabel('x_1');
      ylabel('x_2');
      axis([0 6 0 6]);
      grid on;

    end
    function natural = scaleFromCoded(coded, lb, ub)
      natural = lb + (coded + 1) .* (ub - lb) / 2;
    end
    function [ccd, f_eval_ccd,halton20, f_eval_halton20] = createExperimentalsets(create)
      if create
        % Define the domain boundaries
        x1_range = linspace(0, 6, 100);
```

```
x2_range = linspace(0, 6, 100);

% Create meshgrid for 3D surface plot
[X1, X2] = meshgrid(x1_range, x2_range);

% Create the Function to be Called in the metamodeling
F = @(x1,x2) cos(x1) + sin(x2);

% Compute function values
Fplot = cos(X1) + sin(X2);
%% a. Two-Level Factorial Design
lb = [0, 0];
ub = [6, 6];
factorialPoints = [lb(1), lb(2);
            lb(1), ub(2);
            ub(1), lb(2);
            ub(1), ub(2)];

f_eval_factorial = zeros(size(factorialPoints,1),1);
for i = 1:size(factorialPoints,1)
   f_eval_factorial(i) = F(factorialPoints(i,1), factorialPoints(i,2));
end

plotDesignPoints(X1, X2, Fplot, factorialPoints, f_eval_factorial, 'a. Two-Level Factorial 
Design');

%% b. Central Composite Design (CCD)
alpha = sqrt(2);
axial_coded = [ alpha,  0;
           -alpha,  0;
            0,    alpha;
            0,   -alpha];
ccd_axial = scaleFromCoded(axial_coded, lb, ub);
ccd_center = (lb + ub) / 2;
ccd = [factorialPoints; ccd_axial; ccd_center];
f_eval_ccd = zeros(size(ccd,1), 1);
for i = 1:size(ccd, 1)
   f_eval_ccd(i) = F(ccd(i,1), ccd(i,2));
end

plotDesignPoints(X1, X2, Fplot, ccd, f_eval_ccd, 'CCD');

%% c. Latin Hypercube Design with 10 Points
lhsPoints = lhsdesign(10, 2);
% Need to scale the points
lhsPoints = lb + lhsPoints .* (ub - lb);

f_eval_lhs = zeros(size(lhsPoints,1),1);
```

```
        for i = 1:size(lhsPoints,1)
            f_eval_lhs(i) = F(lhsPoints(i,1), lhsPoints(i,2));
        end

        % Plot the Latin Hypercube design
        plotDesignPoints(X1, X2, Fplot, lhsPoints, f_eval_lhs, 'c. Latin Hypercube Design (10
Points)');

        %% d. Halton Sequence Designs
        % For 10 points:
        pHalton10 = haltonset(2);
        halton10 = net(pHalton10,10);
        % Scale like lhs
        halton10 = lb + halton10 .* (ub - lb);

        f_eval_halton10 = zeros(size(halton10,1),1);
        for i = 1:size(halton10,1)
            f_eval_halton10(i) = F(halton10(i,1), halton10(i,2));
        end

        % Plot the Halton-10 design
        plotDesignPoints(X1, X2, Fplot, halton10, f_eval_halton10, 'd. Halton Sequence Design
(10 Points)');

        % For 20 points:
        pHalton20 = haltonset(2);
        halton20 = net(pHalton20,20);

        % Scale like lhs
        halton20 = lb + halton20 .* (ub - lb);

        f_eval_halton20 = zeros(size(halton20,1),1);
        for i = 1:size(halton20,1)
            f_eval_halton20(i) = F(halton20(i,1), halton20(i,2));
        end

        % Plot the Halton-20 design
        plotDesignPoints(X1, X2, Fplot, halton20, f_eval_halton20, 'd. Halton Sequence Design
(20 Points)');
    end
  end
  function dmodel = buildKrigingModel(design_points, f_eval, X1_vis, X2_vis)

    % Create the model
    xTrain = design_points;
    yTrain = f_eval;

    [xUnique, ia, ~] = unique(xTrain, 'rows', 'stable');
    yUnique = yTrain(ia);
    regr   = @regpoly0;
    corr   = @corrgauss;
    theta0 = 0.1;
    lob    = 1e-6;
    upb    = 10;

    [dmodel, ~] = dacefit(xUnique, yUnique, regr, corr, theta0, lob, upb);

    testPoints = [X1_vis(:), X2_vis(:)];

    predictions = predictor(testPoints, dmodel);
    krigingVis = reshape(predictions, size(X1_vis));

    uniquePredictions = predictor(xUnique, dmodel);

    % Plot the design and the kriging model predictions.
    plotDesignPoints(X1_vis, X2_vis, krigingVis, xUnique, uniquePredictions, ...
        'Method: Kriging Model');
end
```