STUDENT

FIRST LETTER OF SURNAME

PERSONAL TUTOR

Coursework Coversheet

This document is the assessed coursework coversheet for all Computer Science modules conducted at UCL for this academic year. Please do the following when submitting coursework:

- Staple a completed and signed copy of this form to every piece of assessed coursework you submit for modules in the Department of Computer Science.

- avoid the use of document containers such as cardboard or plastic covers, document wallets, ring binders or folders (unless otherwise stated by the lecturer concerned).

We do not wish to discourage students from discussing their work with fellow students and collaborating in solving problems. However you should avoid allowing the collaborative phase to approach too close to a final solution which might make it impossible for you to make your own distinctive intellectual contribution. The key point is that you must not present the results of another person's work "as though they were your own". http://www.ucl.ac.uk/current-students/study/plagiarism/

UCL has now signed up to use a sophisticated detection system (JISC Turn-It-In) to scan work for evidence of plagiarism, and the Department intends to use this for assessed coursework. This system gives access to billions of sources worldwide, including websites and journals, as well as work previously submitted to the Department, UCL and other universities

The collection point for "returned marked" coursework will be in a public area. If you wish to have your coursework returned privately. You MUST collect your coursework as soon as you are informed. □

Submission Details

Please ensure that the details you give are accurate and completed to the best of your knowledge.

COURSEWORK TITLE	Homework 1
LECTURER	Dr. Herbster
MODULE CODE	COMPGI07
MODULE NAME	Mathematical Programming & Mathematical Models
DEADLINE	05 / 12 / 2016

Declaration

I have read and understood the UCL and Departmental statements and guidelines concerning plagiarism.
I declare that:

1. This submission is entirely my own unaided work.
2. Wherever published, unpublished, printed, electronic or other information sources have been used as a contribution or component of this work, these are explicitly, clearly and individually acknowledged by appropriate use of quotation marks, citations, references and statements in the text.
3. In preparing this coursework, I discussed the general approach to take with the person(s) named below, however the content of the submission is my own work alone:

Person(s) with whom I have discussed this coursework:

Signature: R.D[illegible]

For Departmental Office Use

DATE AND TIME RECEIVED

INITIALS OR STAMP

Contents

1 Gradient Descent **4**

2 Linear Regression **17**

Appendices **37**

A Gradient Descent **37**

A.1 Question_1.m 37

A.2 Question_2.m 39

B Linear Regression **42**

B.1 Question_1.m 42

B.2 Question_2.m 45

Bibliography **52**

List of Figures

1.1 Plot of f(x,y) 5

1.2 Gradient Descent in 3-D 6

1.3 Gradient Descent in 2-D 7

1.4 Convergence of solution of linear equations 11

1.5 Convergence of solution of linear equations 11

1.6 Plot of f(x) 12

1.7 Plot of f(x) 14

1.8 Plot of f(x) 16

2.1 Polynomial bases of $k = 1,2,3,4$ 18

2.2 $sin^2(2\pi x)$ funcition wth data of $S_{0.07,30}$ 22

2.3 $S_{0.07,30}$ data fitted with basis functions ($k = 2,5,10,14,18$). 22

2.4 Training error $te_k(S)$ with basis polynomial basis ($k = 1,\ldots,18$). . . 24

2.5 Test error $tse_k(S,T)$ with basis polynomial basis ($k = 1,\ldots,18$). . . 25

2.6 Training ($te_k(S)$) and test error $tse_k(S,T)$ for 100 runs. 25

2.7 Training error $te_k(S)$ with basis sinusoidal basis ($k = 1,\ldots,18$). . . . 27

2.8 Test error $tse_k(S,T)$ with basis sinusoidal basis ($k = 1,\ldots,18$). . . . 27

2.9 Training ($te_k(S)$) and test error $tse_k(S,T)$ for 100 runs with sinusoidal basis. 28

2.10 Example Whack-A-Mole in 4×4 grid. 29

List of Tables

2.1 Equations of Basis functions. 18
2.2 MSE of Basis $k = 1, 2, 3, 4$. 20

Question 1

Gradient Descent

The gradient descent method pseudocode was defined as follows, given $x_0 \in$ domain of f and step size $\lambda > 0$.

repeat

$$x_{t+1} = x_t - \lambda \nabla_x f(x_t)$$

until stop criteria is satisfied

Algorithm 1: Gradient Descent Pseudo-code

Where:

- x_0 is the initial starting point.
- λ is the step size of the gradient descent algorithm.
- $f(x)$ is the given function.
- x_{t+1} is the updated x estimate.

[1a.] After defining the pseudo-code for the Gradient Descent (Algorithm 1), the function $f(x,y)$ the for this question was defined as shown in equation 1.1 below.

$$f(x,y) = (x-2)^2 + 2(y-3)^2 \tag{1.1}$$

A complete 3-D rendering of $f(x,y)$ can be seen in Figure 1.1 shown below.

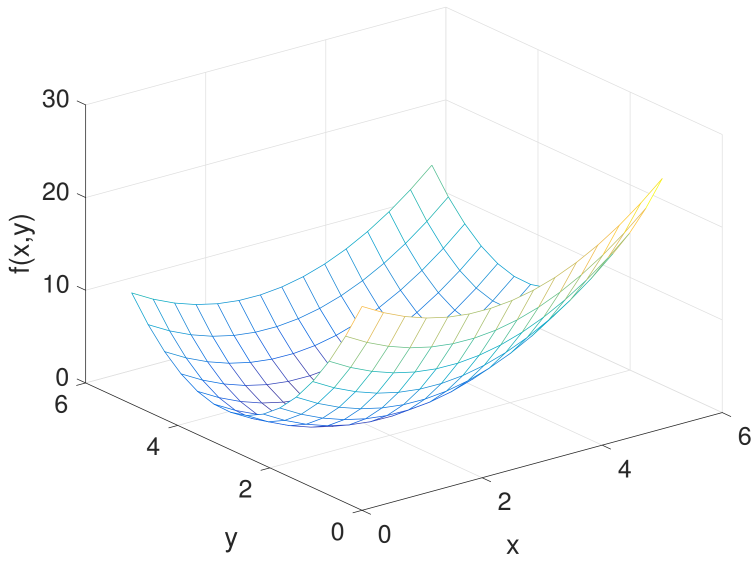

Figure 1.1: Plot of f(x,y)

The MATLAB code utilized in order to produce List 1.1 can be seen below.

Listing 1.1: Code for plotting $f(x,y)$

```
x_gv = linspace(0,5,15);
y_gv = linspace(0,5,15);

% 3-D Plot of f(x,y)
[X,Y] = meshgrid(x_gv,y_gv);
figure;
mesh(X,Y,fcarg(X,Y))
xlabel('x','FontSize',15)
ylabel('y','FontSize',15)
zlabel('f(x,y)','FontSize',15)
set(gca,'fontsize',17);
set(gcf, 'Color', 'w');
```

[1b(i).] The MATLAB function file *graddesc.m* was modified and renamed *graddesc_mod.m* . The MATLAB code for *graddesc_mod.m* can be seen below in Listing 1.2

Listing 1.2: Code for gradient descent of $f(x,y)$

```
function soln = graddesc_mod(f, g, i,e, t)
% gradient descent
% f -- function
% g -- gradient
% i -- initial guess
% e -- step size
% t -- tolerance
soln = [i feval(f,i)];
gi = feval(g,i);
while(norm(gi)>t) % crude termination condition
  i = i - e .* feval(g,i);
  soln = [soln; i feval(f,i)]; % appending histories of vectors to each other.
  gi = feval(g,i)
end
```

[1b(ii).] It can be seen from Figure 1.2 below, the Gradient Descent method transverses along the contour of $f(x,y)$ from the starting point $(x_0,y_0) = (0,0)$ until its terminating condition is satisfied.

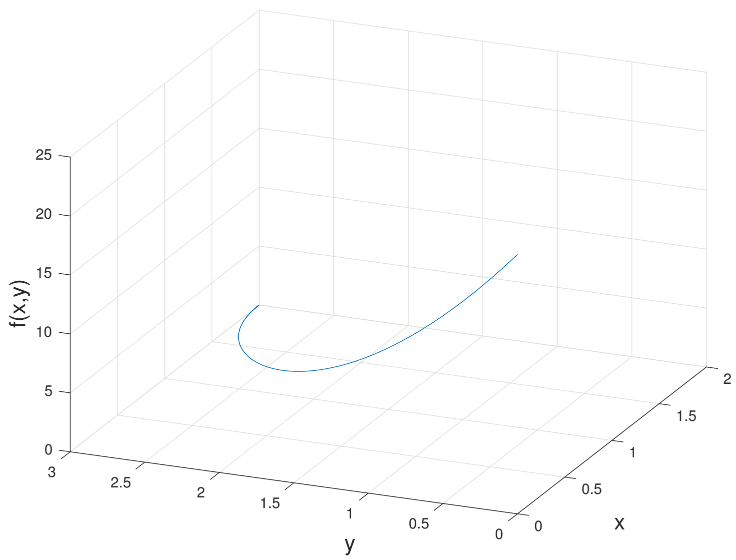

Figure 1.2: Gradient Descent in 3-D

[1b(iii).] The local minimum for $f(x,y)$ was found to be $(x_{min},y_{min}) = (1.9560, 2.9985)$ as shown in Figure 1.3 on the following page.

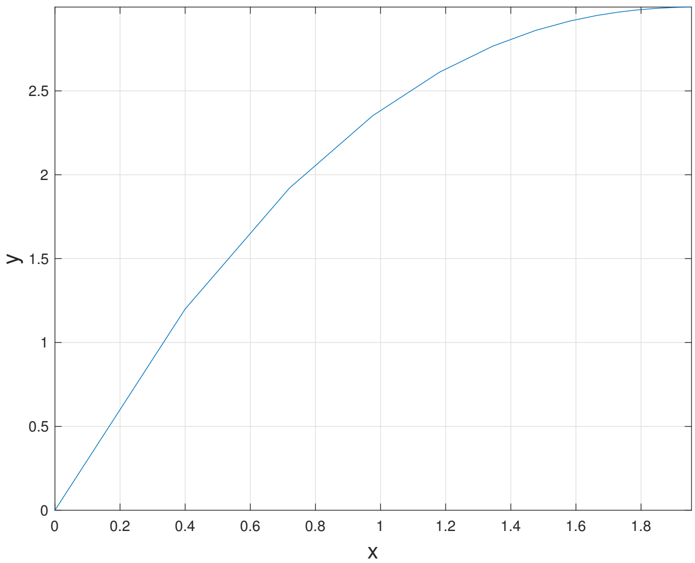

Figure 1.3: Gradient Descent in 2-D

The complete code to produce Figure 1.2 and Figure 1.3 can be seen in Listing 1.3 shown below. The MATLAB complete code for Listing 1.2 can be seen in *Question_1.m* which can be seen in Appendix A.1.

Listing 1.3: Code for plots of gradient descent of $f(x,y)$

```
%% b.
%i - Modified gradient descent function
soln = graddesc_mod('fc','dfc',[0,0],0.1,0.1);

% ii - 3-D rendering of Gradient Descent methods movement to solution.
plot3(soln(:,1),soln(:,2),soln(:,3))
xlabel('x','FontSize',15)
ylabel('y','FontSize',15)
zlabel('f(x,y)','FontSize',15)
grid on;
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/grad_desc_3d', '-eps')

% iii - X-Y Projection of Gradient Descent
figure;
plot(soln(:,1),soln(:,2))
xlabel('x','FontSize',15)
ylabel('y','FontSize',15)
grid on
set(gcf, 'Color', 'w');
```

[2a.] It is widely known that gradient descent is a very ineffiecient method to compute the least square solution to a set of equations. However, the implementation of the gradient descent algorithm required the mathematical understanding of the update procedure of x_{t+1} as highlighted in Algorithm 1 in the earlier question. Nonetheless, the equations take the subsequent forms.

The matrix equation is defined as shown below.

$$Ax \quad = b \tag{1.2}$$

The error column vector takes the form:

$$e \quad = Ax - b \tag{1.3}$$

The sum of square error takes the form:

$$SSE \quad = \quad e^T e \tag{1.4}$$

$$SSE \quad = \quad (Ax - b)^T (Ax - b) \tag{1.5}$$

Therefore if we define the cost (SSE) as a function of x ,therefore it can be written as:

$$J(x) \quad = \quad (Ax - b)^T (Ax - b) \tag{1.6}$$

In order to compute x_{t+1}, the cost $J(x)$ will need to be calculated as shown below.

$$\nabla_x \big((Ax - b)^T (Ax - b) \big) = \begin{pmatrix} \sum_{i=1}^{m} \frac{\delta}{\delta x_1} \left(\sum_{j=1}^{n} a_{ij} x_j - b_j \right)^2 \\ \vdots \\ \sum_{i=1}^{m} \frac{\delta}{\delta x_n} \left(\sum_{j=1}^{n} a_{ij} x_j - b_j \right)^2 \end{pmatrix} \tag{1.7}$$

$$\nabla_x\big((Ax-b)^T(Ax-b)\big) = \begin{pmatrix} \sum_{i=1}^{m} 2\big(\sum_{j=1}^{n} a_{ij}x_i - b_i\big)a_{i1} \\ \vdots \\ \sum_{i=1}^{m} 2\big(\sum_{j=1}^{n} a_{ij}x_i - b_i\big)a_{in} \end{pmatrix} \tag{1.8}$$

$$\nabla_x\big((Ax-b)^T(Ax-b)\big) = \begin{pmatrix} 2\sum_{i=1}^{m} 2\big(Ax-b\big)a_{i1} \\ \vdots \\ 2\sum_{i=1}^{m} 2\big(Ax-b\big)a_{in} \end{pmatrix} \tag{1.9}$$

$$\nabla_x\big(J(x)\big) \quad = \quad 2(A^TAx - A^Tb) \tag{1.10}$$

Once the derivation was completed the subsequent function entitled *mydescent.m* could be created as shown in Listing 1.4 below.

Listing 1.4: Gradient descent function

```
function [x, x_history ,J_history ,iteration_count] = mydescent(A,b,guess ,step ,tol)
% ———— Input & Output Arguments————%
% A – Matrix of coefficients
% b – Output values (Column vector)
% guess – Initial guess
% step – Step size
% tol – Tolerance
% ————————————————————————————————————————%

m = size(b,2); %number of training examples
x = guess; % Initial guess (x_0)
n = size(x,2); % number of features
iteration_count = 0;

J_ini = ((A*x)-b)'*((A*x)-b); %Initial cost/ error of squared summations
J = J_ini;
J_history = [J];
x_history = [x];
grad_j = 2 * ((A'*A*x)-(A'*b));
while (norm(grad_j)>tol) % Repeat until stop criterion is reached.

    grad_j = 2 * ((A'*A*x)-(A'*b)); % Calculated the gradient, Delta_x
    temp = x - (step*grad_j); %Updated estimates of x
    x = temp;
    x_history = [x_history x]; %Append history of x estimate to vector
```

```
    J = ((A*x)-b)'*((A*x)-b);
    J_history = [J_history ;J]; %Append history of J(x) estimate to vector
    iteration_count=iteration_count+1; % Count the number of iterations requried
        to reach convergence.

end
```

[2b.] The function *mydescent* was then utilized in order to solve the given set of equations as shown below:

$$x_1 - x_2 = 1$$

$$x_1 + x_2 = 1$$

$$x_1 + 2 * x_2 = 3$$

The solution was found to be:

$$x_1 = 1.2857$$

$$x_2 = 0.5714$$

The code utilized in order to calculate x_1 and x_2 can be found in Listing 1.5 shown below.

Listing 1.5: Linear Equation Solution

```

% (b.) Solve system of linear equations
A = [1 -1;1 1;1 2];
b = [1;1;3];
tol = 0.0001;
step = 0.01;
guess = [1;1];

[x, x_history,J_history,iteration_count] = mydescent(A,b,guess,step,tol);
```

[2c.] The convergence of the Gradient Descent method to a solution for the given set of linear equations can be seen in Figure 1.4.

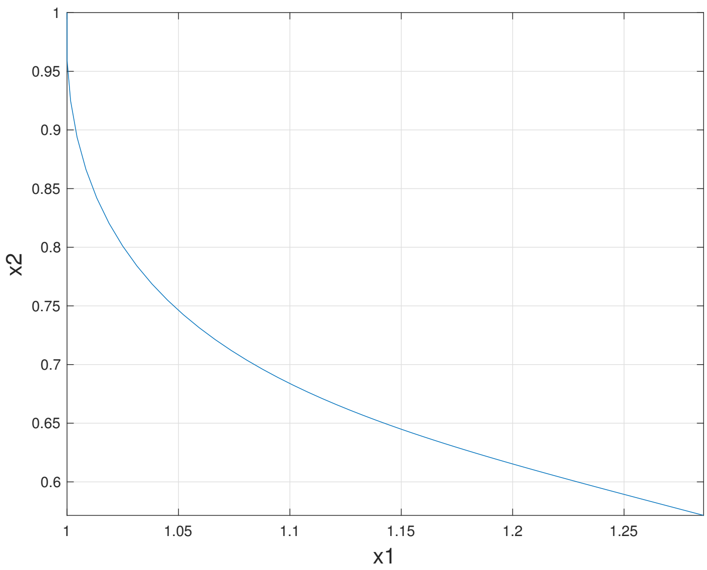

Figure 1.4: Convergence of solution of linear equations

It can be seen by comparing Figure 1.4 shown above with the square error $J(x)$, it decreases as each estimate of x_{t+1} is updated as shown in Figure 1.5 shown below, which is to be expected. This shows the reduction in the square error we aim to achieve.

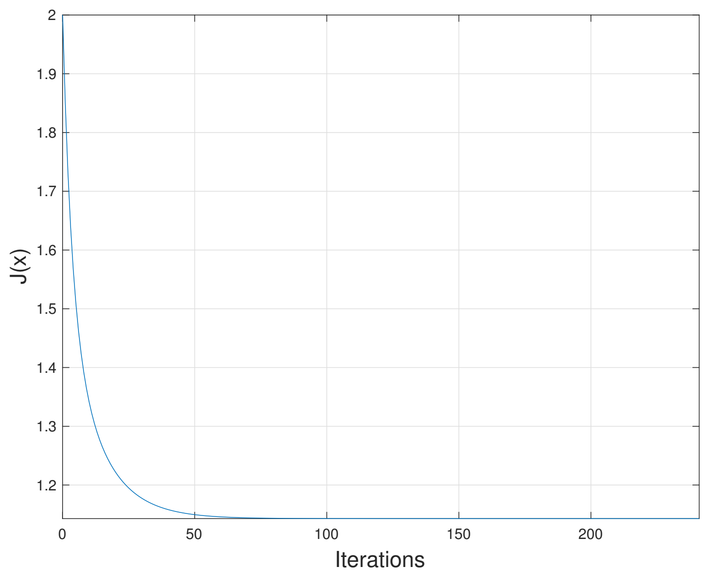

Figure 1.5: Convergence of solution of linear equations

The complete MATLAB code for this quesiton can be found in Appendix A.2.

[3a.] The given function $f(x)$ takes the following form:

$$f(x) = |x-1|^3$$

A graphical display of this function can be seen in Figure 1.6 shown below.

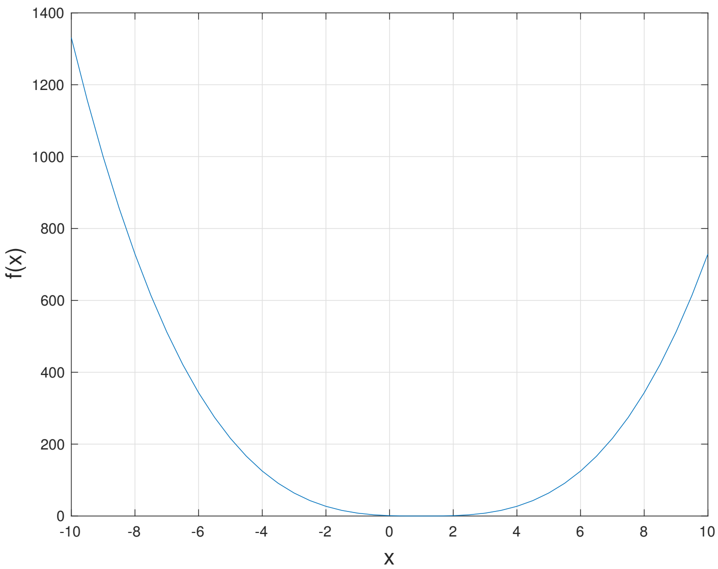

Figure 1.6: Plot of f(x)

Which can be written in its piecewise form as:

$$f(x) = \begin{cases} (x-1)^3 & x \geq 1 \\ -(x-1)^3 & x < 1 \end{cases} \tag{1.11}$$

The first derivative of $f(x)$ takes the form of:

$$f'(x) = \begin{cases} 3(x-1)^2 & x \geq 1 \\ -3(x-1)^2 & x < 1 \end{cases} \tag{1.12}$$

The function $f(x)$ is a convex function as it satisfies the following property:

$$f(\theta x_1 + (1-\theta x_2)) \leq \theta f(x_1) + (1-\theta) f(x_2)$$

Where:

- θ is small scaling factor .
- x_1 and x_2 are points along the x-axis.
- $f(x)$ is the given function.

IIt should be noted at x_0 and λ play an important role in the convergence of gradient descent. When the λ value is reduced to a small number, the number of steps taken to reach a pont o convergence becomes larger. This is also affected if the starting point is chosen to be arbitrarily far away from the point $x = 1$, e.g $x_0 \approx 10^8$ with a $\lambda = 0.0001$. Therefore we shall prove the case for a certain condition in which the given function $f(x)$ with gradient descent applied to it converges to a solution.

Where:

- $lambda = (0.1, 0.01, 0.001)$
- $x_0 = (10, 20, -10, -20)$

The resultant final convergence sequence was then plotted to yield the estimated x^* that minimises $f(x)$. It is hypothesised that it would be $x^* \approx 1$. There does exist a starting point x_0 and λ that would result in a non-trivial solution of $x = 1$. In a more formal sense, this convergence can be proved by the Induction Method.

[3b.] The given function $f(x)$ takes the following form:

$$f(x) = \sqrt{|x-1|}$$

A graphical display of this function can be seen in Figure 1.7 shown below.

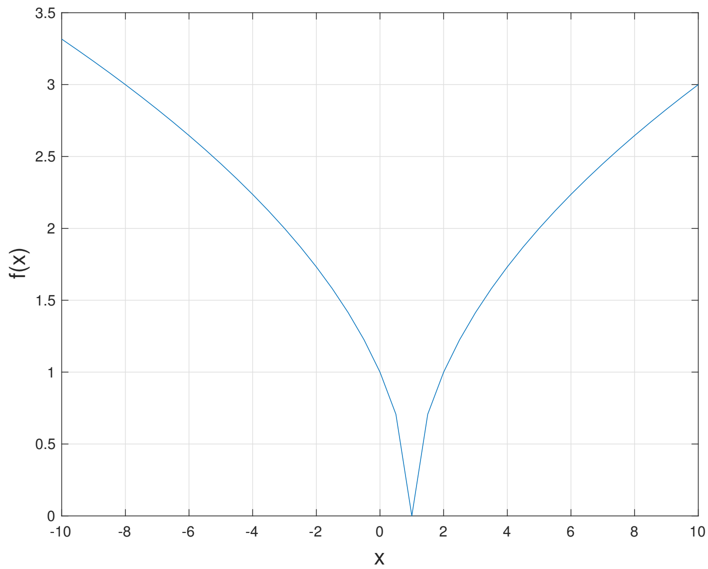

Figure 1.7: Plot of f(x)

Which can be written in its piecewise form as:

$$f(x) = \begin{cases} \sqrt{(x-1)} & x \geq 1 \\ \sqrt{-(x-1)} & x < 1 \end{cases} \tag{1.13}$$

The first derivative of $f(x)$ takes the form of:

$$f'(x) = \begin{cases} \frac{1}{2\sqrt{(x-1)}} & x \geq 1 \\ \frac{1}{-2\sqrt{(x-1)}} & x < 1 \end{cases} \tag{1.14}$$

It is a well known fact that the function $f(x)$ is not differentiable at $x = 1$. Looking at Figure 1.7, starting at a point $x_0 + \varepsilon$ were $\varepsilon > 0$ assuming $x_0 > 1$ not matter how small we make the step size $lambda$ as we $x* -> 1$ there exisit a possibility to get a new updated value $x^* < 1$ and when computing the resultant gradient could result in a gradient at the unknown point that would cause x^* to diverge. As such there exisits no x_0 and fixed λ that would converge to 1. If the convergence of this sequence was to be attempted by Induction, the calculation would diverge.

[3c.] The given function $f(x)$ takes the following form:

$$f(x) = x^4 + 5x^2$$

A graphical display of this function can be seen in Figure 1.8 shown below.

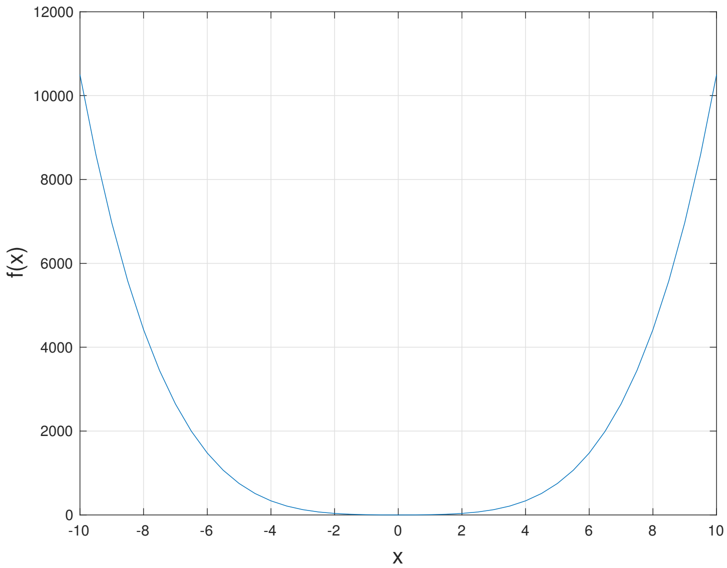

Figure 1.8: Plot of f(x)

This is a convex function.

Question 2

Linear Regression

[1a.] Given a set of data:

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$$

The sum of squared errors is defined as shown in Equation 2.1 below.

$$SSE = \sum_{t=1}^{m} (y_t - \mathbf{w} \cdot \mathbf{x}_t) \tag{2.1}$$

We define **y** and **x** to be vectors:

$$y = (y_1.y_2, \ldots, y_m)$$

$$x = (x_1.x_2, \ldots, x_n)$$

Thus in linear regression, using basis function the data is transformed as shown below:

$$\Phi := \begin{pmatrix} \phi(\mathbf{x}_1) \\ \vdots \\ \phi(\mathbf{x}_m) \end{pmatrix} = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \ldots & \phi_k(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_1) & \ldots & \phi_k(\mathbf{x}_m) \end{pmatrix} \tag{2.2}$$

Therefore our aim is to find a vector **w** that minimises the equation:

$$SSE = (\phi w - y)^T (\phi w - y) = \sum_{t=1}^{m} (y_t - \sum_{i=1}^{k} w_i \phi_i (x_t)) \tag{2.3}$$

Given the data set $S = \{(1,3),(2,2),(3,0),(4,5)\}$ using polynomial bases ($k = 1,2,3,4$) Figure 2.1 was created as shown below:

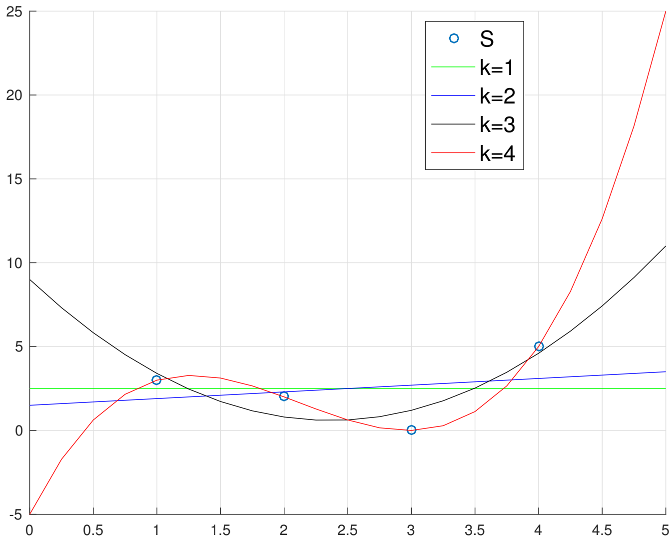

Figure 2.1: Polynomial bases of $k = 1,2,3,4$

It can be seen from inspection of Figure 2.1 above that as the power of the polynomial basis increases, the better fit of the given data is achieved. In saying this, the relevant MSE calculated should therefore decrease.

[1b.] The corresponding equations for each of the polynomial bases can be seen in Table 2.1 below.

k	Equation
1	$y = 2.50$
2	$y = 1.50 + 0.40x$
3	$y = 9.0 - 7.1x + 1.50x^2$
4	$y = -5.0 + 15.1667x - 8.5x^2 + 1.333x^3$

Table 2.1: Equations of Basis functions.

The code required to calculate the equations in Table 2.1 can be seen in Listing 2.1 shown on the following page.

Listing 2.1: Coefficients of Basis

```
%% 1.a

X = [1;2;3;4];
x_vector = 0:0.25:5;
y = [3;2;0;5];

%k=1
PHI_1 = [1;1;1;1];
w_iter_1 = inv(PHI_1'*PHI_1)*PHI_1'*y % Coefficients for eq 3.1b
y_1 = w_iter_1*ones(length(x_vector),1);

%k=2

PHI_2 = [ones(length(X),1) X];
w_iter_2 = inv(PHI_2'*PHI_2)*PHI_2'*y % Coefficients for eq 3.1b
y_2 = w_iter_2(1)+(w_iter_2(2)*x_vector);

%k=3
PHI_3 = [ones(length(X),1) X X.^2];
w_iter_3 = inv(PHI_3'*PHI_3)*PHI_3'*y % Coefficients for eq 3.1b
y_3 = w_iter_3(1)+(w_iter_3(2)*x_vector)+(w_iter_3(3)*x_vector.^2);

%k=4
PHI_4 = [ones(length(X),1) X X.^2 X.^3];
w_iter_4 = inv(PHI_4'*PHI_4)*PHI_4'*y
y_4 = w_iter_4(1)+(w_iter_4(2)*x_vector)+(w_iter_4(3)*x_vector.^2)+(w_iter_4(4)*
    x_vector.^3);
```

[1c.] The MSE is defined as shown in Equation 2.4 below.

$$MSE = \frac{SSE}{m} = \frac{1}{m}\sum_{t=1}^{m}(y_t - \mathbf{w}\cdot\mathbf{x}_t) \tag{2.4}$$

The MSE for each of the curves fitted can be seen in Table 2.2 shown below.

k	**MSE**
1	3.2500
2	3.0500
3	0.8000
4	$6.3305x10^{-24}$

Table 2.2: MSE of Basis $k = 1,2,3,4$.

It can be seen from Table 2.2 above that as the MSE decreases as the polynomial basis increases, which is to be expected. The basis ($k = 3,4$) fit the data well but could susepticble to over-fitting.

The relevant code required to perform the MSE values as listed in Table 2.2 can be seen in Listing 2.2 shown below.

Listing 2.2: MSE Calculation

```
SSE_k1 = sum((y-(PHI_1*w_iter_1)).^2);
MSE_k1 = SSE_k1/length(y) % MSE for k=1

SSE_k2 = sum((y-(PHI_2*w_iter_2)).^2);
MSE_k2 = SSE_k2/length(y) % MSE for k=2

SSE_k3 = sum((y-(PHI_3*w_iter_3)).^2);
MSE_k3 = SSE_k3/length(y) % MSE for k=3

SSE_k4 = sum((y-(PHI_4*w_iter_4)).^2);
MSE_k4 = SSE_k4/length(y) % MSE for k=4
```

The complete MATLAB code for this question can be found in Appendix B.1.

[2a.] In order to demonstrate the phenomena of overfitting, the function *noise_function* was created in order to create a normal distribution with a mean μ and variance σ^2. The equation defining such a distribution can be seen in Equation 2.5 below.

$$N_{\mu,\sigma}(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{2.5}$$

The MATLAB code for this implemetation can be seen in Listing 2.3 below.

Listing 2.3: Noise Function

```
function n = noise_function(mu,sigma)
z = randn;
x = z*sigma+mu;
n = x;
```

[2b.] Let $g_\sigma(x)$ be defined as follows in Equation 2.6.

$$g_\sigma(x) = sin^2(2\pi x) + N_{0,\sigma} \tag{2.6}$$

Using the function $g_\sigma(x)$ the training data set was created as specified in the question:

$$S_{0.07,30} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{30}, g_{0.07}(x_{30}))\}$$

[2b(i).] The function $sin^2(2\pi x)$ was then plotted in the range $0 \leq x \leq 1$ with the data set ($S_{0.07,30}$) superimposed on it as shown on the following page in Figure 2.2.

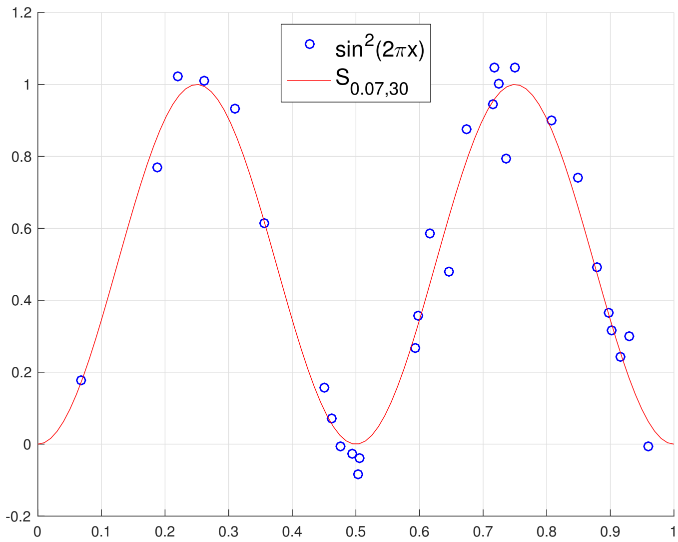

Figure 2.2: $sin^2(2\pi x)$ funcition wth data of $S_{0.07,30}$

[2b(ii).] Polynomial bases of dimension $k = 2, 5, 10, 14, 18$ were applied on the given data ($S_{0.07,30}$). The result can be seen in Figure 2.3 shown below.

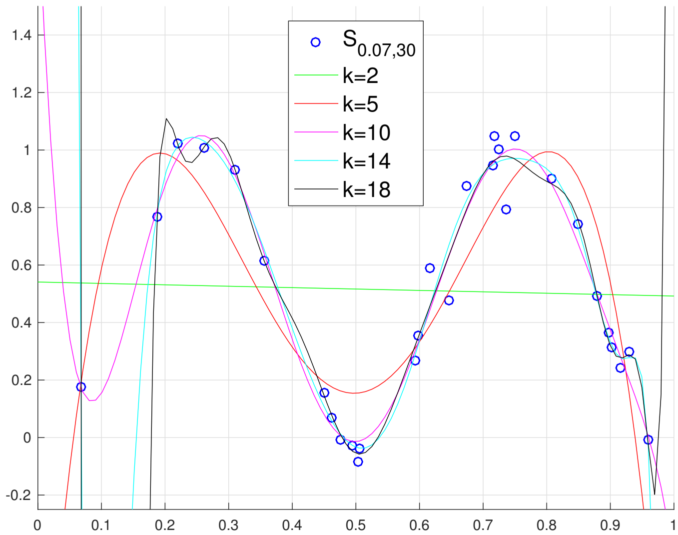

Figure 2.3: $S_{0.07,30}$ data fitted with basis functions ($k = 2, 5, 10, 14, 18$).

It should be noted that as the polynomial matrix was increased an $k- > 18$ the transformed dataset Φ matrix came close to singular value. Therefore as an alternative for this scenario QR Decomposition was applied for Least Squares problem with high dimensional basis. The QR approach was chosen as opposed to LU and Moore-Penrose factorization method as it is more numerically stable [1] . It should also be noted that MATLAB's function *pinv* uses Moore-Penrose method [2] and *inv* uses LU decomposition [3].

It can be seen that as the polynomial basis is increased, the curves pass through more points in $S_{0.07,30}$, however in the intervals $x > 1$ and $x < 0$ the functions diverge rapidly.

[2c.] The training error $te_k(S)$ of fitting the data $S_{0.07,30}$ with polynomial basis of dimension $k = 1,\ldots,18$ can be seen in Figure 2.4 on the following page.

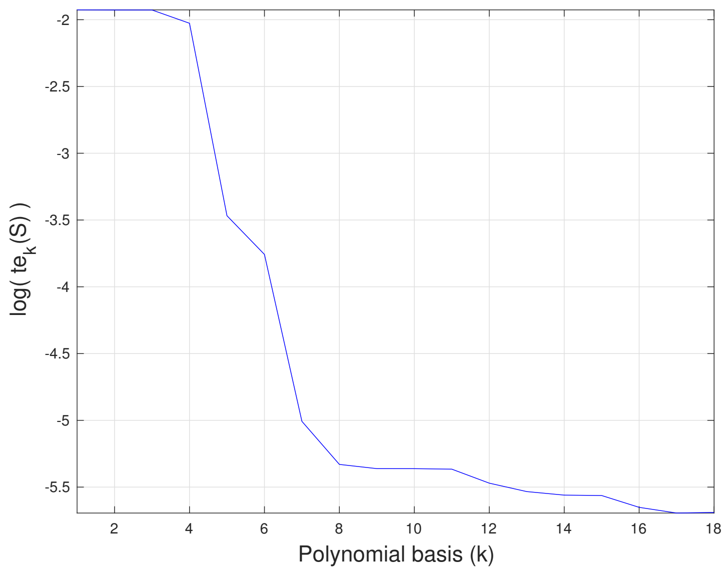

Figure 2.4: Training error $te_k(S)$ with basis polynomial basis ($k = 1,\ldots,18$).

It can be seen from Figure 2.4 above, as $k- > 18$ the MSE decreases which is to be expected.

[2d.] The subsequent test $T_{0.07,1000}$ was then created in order to evaluate the performance of the model created from $S_{0.07,30}$. the test set was defined as:

$$S_{0.07,1000} = \{(x_1, g_{0.07}(x_1)),\ldots,(x_{1000}, g_{0.07}(x_{1000}))\}$$

The test error $tse_k(S,T)$ is therefore defined as, the MSE of the test set T on the polynomial of dimension k fitted from the training set S. The test error $tse_k(S,T)$ can be seen in Figure 2.5 on the following page.

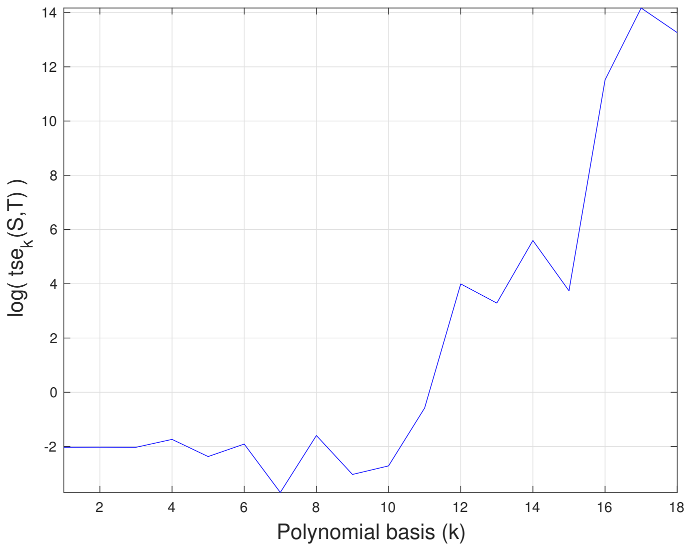

Figure 2.5: Test error $tse_k(S,T)$ with basis polynomial basis ($k = 1,\ldots,18$).

It can be seen that opposed to Figure 2.4, as the polynomial basis is increased $k->18$ the MSE for the test data $tse_k(S,T)$ increases as shown in Figure 2.5. This demonstrates the phenomena of overfitting, as a result we are now fitting noise in the model.

[2e.]As a result of the randomness due to the use of random numbers, each time we run the script slightly different training and test curves would be created. Therefore, the average error results for 100 runs was performed. The resultant average performance can be seen in Figure 2.6 below.

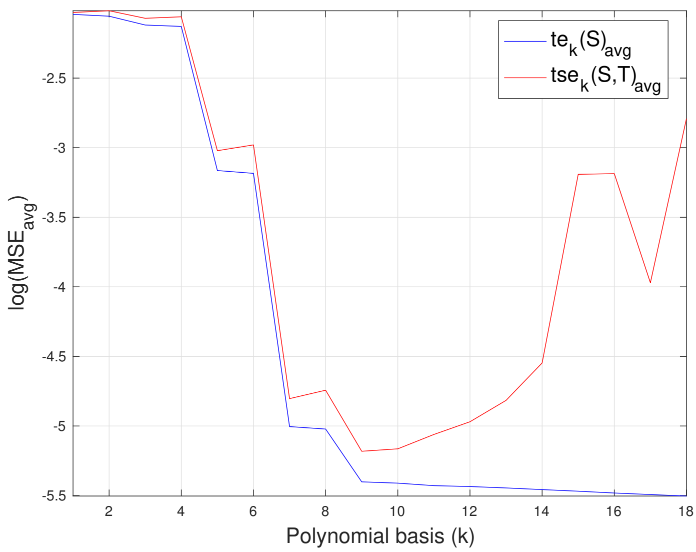

Figure 2.6: Training ($te_k(S)$) and test error $tse_k(S,T)$ for 100 runs.

It can be observed from Figure 2.6 that a similiar trend is present to that seen in Figure 2.5. The phenomena of overfitting is present in Figure 2.6 as the model created using $S_{0.07,30}$ fails to generalise well as the polynomial basis dimension increases when testing on $T_{0.07,1000}$ even for 100 independent simulations.

The complete MATLAB code for this question can be found in Appendix B.2.

[3.] The following basis will now be used in fitting the data on the training ($S_{0.07,30}$) and test set ($T_{0.07,1000}$).

$$sin(1\pi x), sin(2\pi x), \ldots, sin(k\pi x)$$

The training error $te_k(S)$ of fitting the data $S_{0.07,30}$ with sinusoidal basis of dimension $k = 1, \ldots, 18$ can be seen in Figure 2.7 below.

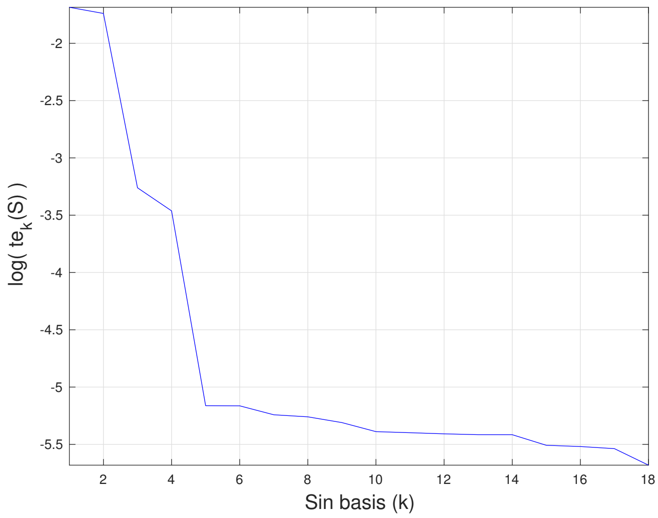

Figure 2.7: Training error $te_k(S)$ with basis sinusoidal basis ($k = 1, \ldots, 18$).

It can be seen that the MSE decreases as the sinusoidal basis dimension $k- > 18$. The test error $tse_k(S,T)$ based on a sinusoidal basis can be seen seen in Figure 2.8 below.

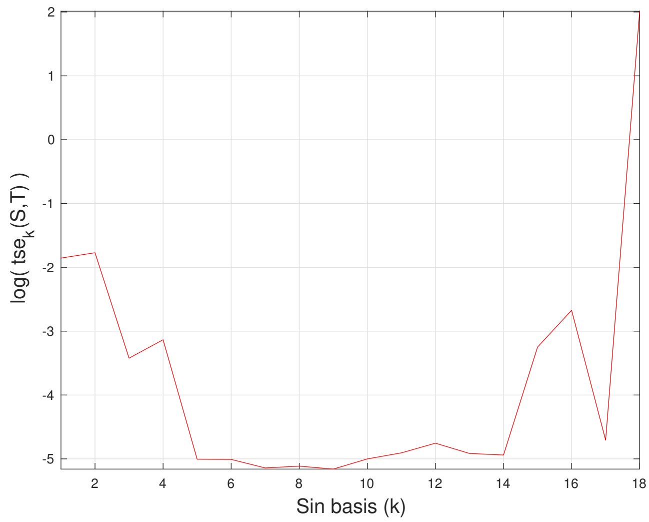

Figure 2.8: Test error $tse_k(S,T)$ with basis sinusoidal basis ($k = 1, \ldots, 18$).

The average MSE of 100 simulation runs were completed and the result can be seen in Figure 2.9

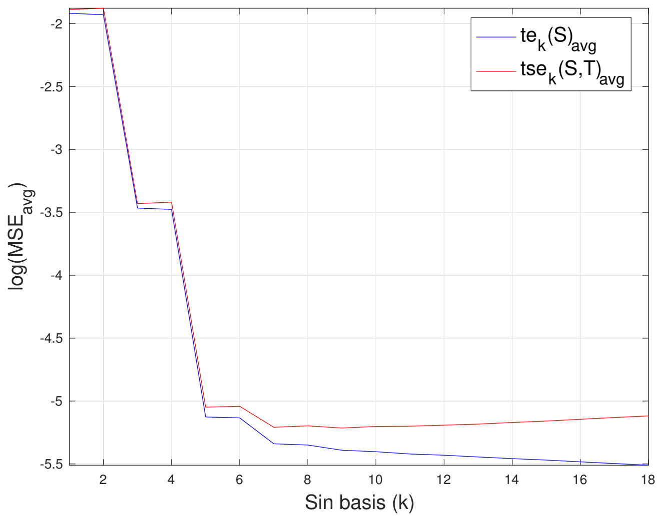

Figure 2.9: Training $(te_k(S))$ and test error $tse_k(S,T)$ for 100 runs with sinusoidal basis.

The complete MATLAB code for this question can be found in Appendix B.2.

[4.] The aim of this task was to design an algorithm that can solve the holes the customer must hit for an $n \times n$ grid of Whack-A-Mole. The problem can be formalised into a mathematical framework that should be solveable given an initial configuration, if such a solution exists. If the game is not winnable given an initial configuraiton, it will be stated that it cannot be won.

Therefore, considering a 4×4 grid, the state of each of holes can be represented in Figure 2.10 below.

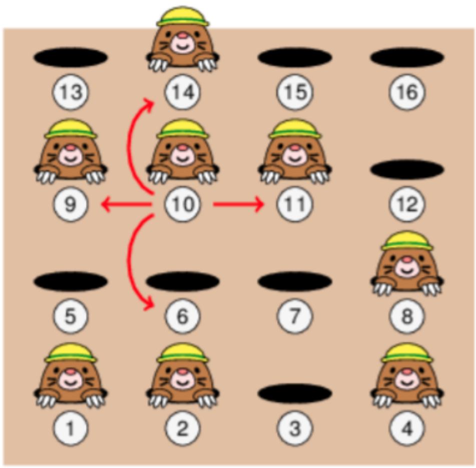

Figure 2.10: Example Whack-A-Mole in 4×4 grid.

Given that the fact that a mole can only be either be hidden in a hole or outside of a hole, each hole shall only take on binary states $(0, 1)$. In saying this, Figure 2.10 can be represented in matrix form shown below in Equation 2.10.

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \tag{2.7}$$

The matrix $\mathbf{M}$ can then be written as a column vector ($\mathbf{b}$) as shown in Equation 2.8 below.

$$\mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_{16} \end{pmatrix} \tag{2.8}$$

Where:

- $\mathbf{b} \in Z_2$ which is the field of integers modulo 2. The state of each hole can only take on the value 0 or 1.

- The holes of the grid ($M_{i,j}$) are elements of the ith row and jth column. In this instance b is a 16×1 column vector.

It is important to understand that state changes within this game. Two key observations can be made from the rules defined in Whack-A-Mole [4] namely:

1. Whacking a certain hole twice is the same as not whacking that hole. In saying this, the order in which the holes are whacked is not important. The only aspect is to consider solutions resulting in a hole being whacked no more than once.

2. The state of a hole not having or having a mole present depends on the number of times the hole and its surrounding neighbours have been whacked. The number of whacks are needed to see if it and its surrounding neighbours have been struck an even or odd number of times. The number of times holes are whacked are important and not the order in which they whacked.

It should be important to list the mathematics of the modulo 2 operations, this will give a clear understanding to the state changes of each hole.

$$0+0=0, \quad 0+1=1, \quad 0+1=1, \quad 1+0=1$$

$$0\times 0=0, \quad 0\times 1=0, \quad 1\times 0=0, \quad 1\times 1=1$$

$$-0=0, \quad -1=1, \quad 1/1=1$$

The action of whacking a hole can be described by the following example action matrices shown below:

$$\mathbf{A_{1,2}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.9}$$

The subscript for matrix $\mathbf{A}$ namely, $\mathbf{A}_{i,j}$ deponds the state changes that would occur by striking hole. Therefore, $\mathbf{A_{1,2}}$ in Equation 2.9 denotes striking $(1,2)$ in the 4×4 grid shown in Equation 2.10. The resultant state change of the grid due to $\mathbf{A_{1,2}}$ added to $\mathbf{M}$ would become:

$$\mathbf{M_1} = \mathbf{M} + \mathbf{A_{1,2}}$$

$$\mathbf{M_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \tag{2.10}$$

Therefore, following a similiar thought process we are able to eliminate the moles through a linear combination of matrices $(A_{4,2}, A_{4,4})$ as shown in Equations 2.11 - 2.13 shown below.

$$\mathbf{A_{4,2}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \tag{2.11}$$

$$\mathbf{A_{4,4}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \tag{2.12}$$

$$\mathbf{M_2} = \mathbf{M_1} + \mathbf{A_{4,4}} + \mathbf{A_{4,2}} = \mathbf{0} \tag{2.13}$$

Where:

- $\mathbf{0}$ denotes the zero matrix.

$$\mathbf{0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.14}$$

A winning combination can therefore be expressed in a more general form as a set of linear combinations for an $n \times n$ grid of Whack-A-Mole as shown in Equation 2.15 [5].

$$\mathbf{M} + \sum_{i,j}^{n} x_{i,j}\mathbf{A}_{i,j} = \mathbf{0} \tag{2.15}$$

$$\sum_{i,j}^{n} x_{i,j}\mathbf{A}_{i,j} = \mathbf{M} \tag{2.16}$$

Where:

- $x_{i,j}$ denotes if a hole needs to be whacked (1) or not (0). This is more formally known in academic literature as the strategy vector $\mathbf{x}$ [4].

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{16} \end{pmatrix} \tag{2.17}$$

The resultant set of linear equations (Equation 2.15) that need to be solved can then be expressed as shown in Equation 2.18 below.

$$\mathbf{Ax} = \mathbf{b} \tag{2.18}$$

Where:

- $\mathbf{A}$ is defined in Equation 2.20 which for a $n \times n$ grid game is, $n^2 \times n^2$ in dimension.
- $\mathbf{x}$ is defined in Equation 2.17 which for a $n \times n$ grid game is, $n^2 \times 1$ in dimension.
- $\mathbf{b}$ is defined in Equation 2.8 which for a $n \times n$ grid game is, $n^2 \times 1$ in dimension..

In case for 4×4 grid the matrices and vectors take the following form.

$$\mathbf{A_{4,4}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \tag{2.19}$$

The complete form of matrix $\mathbf{A}$ for a $n \times n$ grid of Whack-A-Mole can be written as:

$$\mathbf{A} = \begin{pmatrix} F & I & O & O \\ I & F & I & O \\ O & I & F & I \\ O & O & I & F \end{pmatrix} \tag{2.20}$$

Where:

- O is $n \times n$ matrix of all zero elements.
- I is $n \times n$ identity matrix.
- F is $n \times n$ symmetric matrix. This matrix expresses the dependency of all holes based on their actions. Elements of matrix F was expressed in Equations 2.9 and 2.19.

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \tag{2.21}$$

The matric equation defined in Equation 2.18 can be solved using Gauss-Jordan Elimination as stated in [4] and [5]. Using Equation 2.18, 2.8, 2.17,2.20, 2.21. The matrix equation can be wriite as shown below:

$$\mathbf{Ax} = \begin{pmatrix} F & I & O & O \\ I & F & I & O \\ O & I & F & I \\ O & O & I & F \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{16} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_{16} \end{pmatrix} \tag{2.22}$$

In order to determine if a game of Whack-A-Mole is winnable given an initial configuration for a $n \times n$ grid game, Anderson [4] stated two Theorems that would need to be satisfied. This will not considered here, and forthwith it is assumed all given initial configurations are winnable. Thus standard Gauss-Jordan Elimination can be applied on the set of linear equations defined in Equation 2.18 and 2.22. The pseudocode describing Gauss-Jordan elimination can be seen in below.

1: **for** $1 <= k < n$:
2: #Find a pivot at A[k,k]
3: #Operate on all rows below the k-th pivot element
4: **for** k+1< = i < n:
5: $m_{scale} \leftarrow \frac{A[i,k]}{A[k,k]}$
6: #Operate on all column elements for the i-th row (current row)
7: **for** k+1< = j < n:
8: $A[i,j] \leftarrow A[i,j] - (m_{scale} \times A[k,j])$
9: $b[i] \leftarrow b[i] - (m_{scale} \times b[k])$
10: #Perform back-substitution (Populate lower-triangle matrix)
11: $A[i,j] \leftarrow 0$

Algorithm 2: Gauss-Jordan Elimnaion Pseudocode

It can be seen from inspection of the pseudocode in Algorithm 2 above that for a given input state matrix D of dimension $n \times n$, the computational complexity would be $O(n^3)$ which is consistent with the findings of [6].

This is due to the fact that the outer for loop takes complexity $O(n^2)$ as we need to loop through all rows and columns in the $n \times n$ matrix to find a pivot element $A[k,k]$ as stated in line 2 of Algorithm 2. Following this, we need to clear all rows (i) noted in line 7 which are below the pivot, this would require $O(n^2)$ computation to look at all the elements in the matrix. In each step for a given row, the nested for-loop of j needs to elimnate the subsequent columns ($O(n)$). Due to the fact that this is a nested sequence, the dominant computational term would therefore be $O(n^3)$ for Gauss-Jordan elimination.

In the case of Whack-A-Mole (Equation 2.22) for a game of $n \times n$ in grid size the corresponding size of the complete action matrix $\mathbf{A}$ becomes $(n \times n)$ by $(n \times n)$. Therefore, if we perform the following substitution:

$$k = n \times n \tag{2.23}$$

The corresponding size of **A** can be considered to be of size k by k, based on analysis of 2.22 and [7] it can be concluded that the linear Equation 2.18 has complexity

$$O(k^3)$$

.

Using the result in Equation 2.23 and subsituting in for k, the complexity of solving Equation 2.18 becomes $O(n^6)$ as shown below.

$$O(k^3) = O(n \times n^3) = O((n^2)^3) = O(n^6) \tag{2.24}$$

In saying this, the final complexity for the Whack-A-Mole algorithm was found to be $O(n^6)$, which is polynomial in n based on the input grid size $n \times n$.

Appendix A

Gradient Descent

A.1 Question_1.m

```
%-----------------------------------------------------------------------%
% Module: GI07 - Mathematical Programming & Research Methods
% Assignment : Homework 1
% Author : Russel Stuart Daries
% Student ID: 16079408
% Question: 1
% Section: Gradient Descent
% Description: Familiarise with MATLAB, implement Gradient descent and
% understand mathematical reasoning.
% -----------------------------------------------------------------------%

close all
clear all
clc

%% a.
x_gv = linspace(0,5,15);
y_gv = linspace(0,5,15);

% 3-D Plot of f(x,y)
[X,Y] = meshgrid(x_gv,y_gv);
figure;
mesh(X,Y,fcarg(X,Y))
xlabel('x','FontSize',15)
ylabel('y','FontSize',15)
zlabel('f(x,y)','FontSize',15)
set(gca,'fontsize',17);
set(gcf, 'Color', 'w');

```

```
export_fig '/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/f_xy_3d' '-eps'
close
local_min = graddesc('fc','dfc',[0,0],0.1,0.1)


%% b.

%i - Modified gradient descent function
soln = graddesc_mod('fc','dfc',[0,0],0.01,0.01);

% ii - 3-D rendering of Gradient Descent methods movement to solution.
plot3(soln(:,1),soln(:,2),soln(:,3))
xlabel('x','FontSize',15)
ylabel('y','FontSize',15)
zlabel('f(x,y)','FontSize',15)
view([-67.1 28.4]);
grid on;
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/grad_desc_3d', '-eps')
close
% iii - X-Y Projection of Gradient Descent
figure;
plot(soln(:,1),soln(:,2))
xlabel('x','FontSize',15)
ylabel('y','FontSize',15)
grid on
set(gcf, 'Color', 'w');
axis tight;
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/grad_desc_2d', '-eps')
close
```

A.2 Question_2.m

```
%-----------------------------------------------------------------------%
% Module: GI07 - Mathematical Programming & Research Methods
% Assignment : Homework 1
% Author : Russel Stuart Daries
% Student ID: 16079408
% Question: 2
% Section: Gradient Descent
% Description: Implement Gradient descent main file.
% ----------------------------------------------------------------------%

close all
clear all
clc


%% 2.
% (a.) Complete function mydescent
% Shown in mydescent.m file

% (b.) Solve system of linear equations
A = [1 -1;1 1;1 2];
b = [1;1;3];
tol = 0.0001;
step = 0.01;
guess = [1;1];

[x, x_history,J_history,iteration_count] = mydescent(A,b,guess,step,tol);


%(c.)
% Plot of Iteration count vs cost (J(x))
iteration_history = 0:1:iteration_count;
figure
plot(iteration_history,J_history)
xlabel('Iterations','FontSize',15)
ylabel('J(x)','FontSize',15)
% zlabel('F(X,Y)','FontSize'?,15)
grid on;
set(gcf, 'Color', 'w');
axis tight;
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/iteration_vs_cost', '-eps')
close
```

```

% Plot of convergence of x vector to final x_1 and x_2 values
figure
plot(x_history(1,:),x_history(2,:))
xlabel('x1','FontSize',15)
ylabel('x2','FontSize',15)
grid on;
axis tight;
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/converge', '-eps')
close

% Plots of x_0,x_1,x_2 values vs Iteration count
figure
plot(iteration_history,x_history(1,:))
xlabel('Iterations','FontSize',15)
ylabel('x1','FontSize',15)
grid on;
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/x0_plot', '-eps')
close
%
figure
plot(iteration_history,x_history(2,:))
xlabel('Iterations','FontSize',15)
ylabel('x2','FontSize',15)
grid on;
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/x1_plot', '-eps')
close
%
% figure
% plot(iteration_history,x_history(3,:))
% xlabel('Iterations','FontSize',15)
% ylabel('x3','FontSize',15)
% grid on;
% set(gcf, 'Color', 'w');
% export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/x2_plot', '-eps')
```

```
% close


%% 3. Plotting the functions given in each of the subsequent questions.
%a.
x = -10:0.5:10;
figure(1)
f1 = abs(x-1).^3;
plot(x,f1)
grid on
xlabel('x','FontSize',15)
ylabel('f(x)','FontSize',15)
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/f1_plot', '-eps')
close

%b.
figure(2)
f2 = sqrt(abs(x-1));
plot(x,f2)
grid on
xlabel('x','FontSize',15)
ylabel('f(x)','FontSize',15)
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/f2_plot', '-eps')
close

%c.
figure(3)
f3 = x.^4+(5*x.^2);
plot(x,f3)
grid on
xlabel('x','FontSize',15)
ylabel('f(x)','FontSize',15)
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/f3_plot', '-eps')
close
```

Appendix B

Linear Regression

B.1 Question_1.m

```
%--------------------------------------------------------------------------------%
% Module: GI07 - Mathematical Programming & Research Methods
% Assignment : Homework 1
% Author : Russel Stuart Daries
% Student ID: 16079408
% Question: 1
% Section: Linear Regression
% Description: Linear Regression implementation
% -------------------------------------------------------------------------------%

close all
clear all
clc

%% 1.a

X = [1;2;3;4];
x_vector = 0:0.25:5;
y = [3;2;0;5];

%k=1
PHI_1 = [1;1;1;1];
w_iter_1 = inv(PHI_1'*PHI_1)*PHI_1'*y % Coefficients for eq 3.1b
y_1 = w_iter_1*ones(length(x_vector),1);

%k=2

PHI_2 = [ones(length(X),1) X];
w_iter_2 = inv(PHI_2'*PHI_2)*PHI_2'*y % Coefficients for eq 3.1b
y_2 = w_iter_2(1)+(w_iter_2(2)*x_vector);
```

```

%k=3
PHI_3 = [ones(length(X),1) X X.^2];
w_iter_3 = inv(PHI_3'*PHI_3)*PHI_3'*y % Coefficients for eq 3.1b
y_3 = w_iter_3(1)+(w_iter_3(2)*x_vector)+(w_iter_3(3)*x_vector.^2);

%k=4
PHI_4 = [ones(length(X),1) X X.^2 X.^3];
w_iter_4 = inv(PHI_4'*PHI_4)*PHI_4'*y
y_4 = w_iter_4(1)+(w_iter_4(2)*x_vector)+(w_iter_4(3)*x_vector.^2)+(w_iter_4(4)*
    x_vector.^3);

%1a.
figure;
scatter(X,y)
hold on
plot(x_vector,y_1,'g')
hold on
plot(x_vector,y_2,'b')
hold on
plot(x_vector,y_3,'k')
hold on
plot(x_vector,y_4,'r')
grid on;
axis tight;
set(gcf, 'Color', 'w');
leg=legend('S','k=1','k=2','k=3','k=4','Location','Best')
set(leg,'FontSize',15)
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/simple_basis_4', '-eps')
close

%1b.
% Coefficients calculated in lines above when w_iter calculated

% 1c.

SSE_k1 = sum((y-(PHI_1*w_iter_1)).^2);
MSE_k1 = SSE_k1/length(y) % MSE for k=1

SSE_k2 = sum((y-(PHI_2*w_iter_2)).^2);
MSE_k2 = SSE_k2/length(y) % MSE for k=2

SSE_k3 = sum((y-(PHI_3*w_iter_3)).^2);
MSE_k3 = SSE_k3/length(y) % MSE for k=3
```

```

SSE_k4 = sum((y-(PHI_4*w_iter_4)).^2);
MSE_k4 = SSE_k4/length(y) % MSE for k=4
```

B.2 Question_2.m

```
%---------------------------------------------------------------------------%
% Module: GI07 - Mathematical Programming & Research Methods
% Assignment : Homework 1
% Author : Russel Stuart Daries
% Student ID: 16079408
% Question: 2
% Section: Linear Regression
% Description: Linear Regression implementation
% --------------------------------------------------------------------------%

close all
clear all
clc

%%
%2a - Function defined as noise_function

%2b.
mu = 0;
sigma = 0.07;
x_rand = rand([30,1]);
x_smooth = linspace(0,1);

for i=1:length(x_rand)
    g_x_noise(i) = (sin(2*pi*x_rand(i)))^2 + noise_function(mu,sigma);
end
g_x_noise = g_x_noise';
g_x_smooth = (sin(2*pi*x_smooth)).^2;

%2.b.i
figure;
scatter(x_rand,g_x_noise,'b') % Generated Data set S
hold on
plot(x_smooth,g_x_smooth,'r') % Smoothed Sin curve
grid on
set(gcf, 'Color', 'w');
leg=legend('sin^2(2\pix)','S_{0.07,30}','Location','Best')
set(leg,'FontSize',15)
export_fig('/Users/russeldaries/Documents/University␣College␣London/Computer␣Science/Courses/Mathematical␣Methods␣for␣Machine␣Learning/Assignments/Assignment␣1/Report/LaTeX/report/Figures/sin_with_data', '-eps')
close

% 2b.ii
```

```
% k=2
figure;
scatter(x_rand,g_x_noise,'b')
hold on
PHI_k2 = poly_basis(2,x_rand);
w_iter_k2 = (PHI_k2'*PHI_k2)\(PHI_k2'*g_x_noise); % Coefficients for eq 2b.ii
y_k2 = basis_calc(w_iter_k2,x_smooth');
hold on
plot(x_smooth,y_k2,'g')

% k=5
PHI_k5 = poly_basis(5,x_rand);
w_iter_k5 = (PHI_k5'*PHI_k5)\(PHI_k5'*g_x_noise); % Coefficients for eq eq 2b.ii
y_k5 = basis_calc(w_iter_k5,x_smooth');
plot(x_smooth,y_k5,'r')

% k=10
PHI_k10 = poly_basis(10,x_rand);
[Q_10,R_10] = qr(PHI_k10); % QR decomposition
w_iter_k10 = (R_10)\(Q_10'*g_x_noise); % Coefficients for eq eq 2b.ii
y_k10 = basis_calc(w_iter_k10,x_smooth');
plot(x_smooth,y_k10,'m')

% k=14
PHI_k14 = poly_basis(14,x_rand);
[Q_14,R_14] = qr(PHI_k14); % QR decomposition
w_iter_k14 = (R_14)\(Q_14'*g_x_noise); % Coefficients for eq eq 2b.ii
y_k14 = basis_calc(w_iter_k14,x_smooth');
plot(x_smooth,y_k14,'c')

% k=18
PHI_k18 = poly_basis(18,x_rand);
[Q_18,R_18] = qr(PHI_k18); % QR decomposition
w_iter_k18 = (R_18)\(Q_18'*g_x_noise); % Coefficients for eq eq 2b.ii
y_k18 = basis_calc(w_iter_k18,x_smooth');
plot(x_smooth,y_k18,'k')
ylim([-0.25 1.5])
hold off
grid on;
set(gcf, 'Color', 'w');
leg=legend('S_{0.07,30}','k=2','k=5','k=10','k=14','k=18','Location','North')
set(leg,'FontSize',15)
export_fig('/Users/russeldaries/Documents/University␣College␣London/Computer␣
    Science/Courses/Mathematical␣Methods␣for␣Machine␣Learning/Assignments/
    Assignment␣1/Report/LaTeX/report/Figures/k_dim_ini', '-eps')
close
```

```

%2c.
for i=1:18
    PHI = poly_basis(i,x_rand);
    [Q,R] = qr(PHI); % QR decomposition
    w_iter = (R)\(Q'*g_x_noise);
    SSE_k(i) = sum((g_x_noise-(PHI*w_iter)).^2);
    MSE_k(i) = SSE_k(i)/length(g_x_noise);
end
% Plot of MSE for Polynomial basis k=1,...,18 on training set
dim_k = 1:18;
plot(dim_k,log(MSE_k),'b')
xlabel('Polynomial␣basis␣(k)','FontSize',15)
ylabel('log(␣te_{k}(S)␣)','FontSize',15)
axis tight;
grid on
hold off
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University␣College␣London/Computer␣
    Science/Courses/Mathematical␣Methods␣for␣Machine␣Learning/Assignments/
    Assignment␣1/Report/LaTeX/report/Figures/k_dim_18', '-eps')
close
%2d.

x_rand_1000 = rand([1000,1]);
for i=1:length(x_rand_1000)
    g_x_noise_1000(i) = (sin(2*pi*x_rand_1000(i)))^2 + noise_function(mu,sigma); %
        Generated test set T
end
g_x_noise_1000 = g_x_noise_1000';

for i=1:18
    PHI_train = poly_basis(i,x_rand);
    [Q,R] = qr(PHI_train);
    w_iter_train = (R)\(Q'*g_x_noise); %Coefficients of w based on training data
    PHI_1000 = poly_basis(i,x_rand_1000);
    SSE_k_1000(i) = sum((g_x_noise_1000-(PHI_1000*w_iter_train)).^2);
    MSE_k_1000(i) = SSE_k_1000(i)/length(g_x_noise_1000);
end
% Plot of MSE for Polynomial basis k=1,...,18 on test set
figure;
plot(dim_k,log(MSE_k_1000),'b')
xlabel('Polynomial␣basis␣(k)','FontSize',15)
ylabel('log(␣tse_{k}(S,T)␣)','FontSize',15)
axis tight;
grid on
```

```
hold off
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University␣College␣London/Computer␣
    Science/Courses/Mathematical␣Methods␣for␣Machine␣Learning/Assignments/
    Assignment␣1/Report/LaTeX/report/Figures/k_dim_18_test', '-eps')
close
%2e.
MSE_100 = [];
MSE_1000 = [];

% Resultant MSE averaged for 100 runs
for j = 1:100
    x_rand_repeat_100 = rand([100,1]);
    x_rand_repeat_1000 = rand([1000,1]);
    g_x_noise_repeat_100 = [];

    for k=1:length(x_rand_repeat_100)
        g_x_noise_repeat_100(k) = (sin(2*pi*x_rand_repeat_100(k)))^2 +
            noise_function(mu,sigma);
    end
    g_x_noise_repeat_100 = g_x_noise_repeat_100';

    g_x_noise_repeat_1000 = [];
    for m=1:length(x_rand_repeat_1000)
        g_x_noise_repeat_1000(m) = (sin(2*pi*x_rand_repeat_1000(m)))^2 +
            noise_function(mu,sigma);
    end
    g_x_noise_repeat_1000 = g_x_noise_repeat_1000';
    SSE_k_repeat_100 = [];
    MSE_k_repeat_100 = [];
    SSE_k_repeat_1000 = [];
    MSE_k_repeat_1000 = [];

    for i=1:18
        PHI_repeat_100 = poly_basis(i,x_rand_repeat_100);
        [Q,R] = qr(PHI_repeat_100);
        w_iter_repeat_100 = (R)\(Q'*g_x_noise_repeat_100);

        SSE_k_repeat_100(i) = sum((g_x_noise_repeat_100-(PHI_repeat_100*
            w_iter_repeat_100)).^2);
        MSE_k_repeat_100(i) = SSE_k_repeat_100(i)/length(g_x_noise_repeat_100);

        PHI_repeat_1000 = poly_basis(i,x_rand_repeat_1000);
        SSE_k_repeat_1000(i) = sum((g_x_noise_repeat_1000-(PHI_repeat_1000*
            w_iter_repeat_100)).^2);
        MSE_k_repeat_1000(i) = SSE_k_repeat_1000(i)/length(g_x_noise_repeat_1000);
```

```
    end
    MSE_100 = [MSE_100; MSE_k_repeat_100];
    MSE_1000 = [MSE_1000; MSE_k_repeat_1000];

end
% Resultant MSE for 100 runs for Training and Test set
figure;
MSE_100_avg = mean(MSE_100);
MSE_1000_avg = mean(MSE_1000);
plot(dim_k,log(MSE_100_avg),'b')
hold on
plot(dim_k,log(MSE_1000_avg),'r')
xlabel('Polynomial basis (k)','FontSize',15)
ylabel('log(MSE_{avg})','FontSize',15)
axis tight;
grid on
hold off
set(gcf, 'Color', 'w');
leg=legend('te_{k}(S)_{avg}','tse_{k}(S,T)_{avg}','Location','Best')
set(leg,'FontSize',15)
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/k_dim_100', '-eps')
close

%3.
for i=1:18
    PHI_sin = sin_basis(i,x_rand);
    [Q,R] = qr(PHI_sin);
    w_iter_sin = (R)\(Q'*g_x_noise);
    SSE_k_sin(i) = sum((g_x_noise-(PHI_sin*w_iter_sin)).^2);
    MSE_k_sin(i) = SSE_k_sin(i)/length(g_x_noise);
end
% Plot of MSE for Sin basis k=1,...,18 on training set
dim_k = 1:18;
figure;
plot(dim_k,log(MSE_k_sin),'b')
xlabel('Sin basis (k)','FontSize',15)
ylabel('log( te_{k}(S) )','FontSize',15)
axis tight;
grid on
hold off
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/k_dim_18_sin', '-eps')
```

```
close
% Expanding basis for k->18
for i=1:18
    PHI_train_sin = sin_basis(i,x_rand);
    [Q,R] = qr(PHI_train_sin);
    w_iter_train_sin = (R)\(Q'*g_x_noise); %Coefficients of w based on training
        data
    PHI_1000 = sin_basis(i,x_rand_1000);
    SSE_k_1000_sin(i) = sum((g_x_noise_1000-(PHI_1000*w_iter_train_sin)).^2);
    MSE_k_1000_sin(i) = SSE_k_1000_sin(i)/length(g_x_noise_1000);
end
% Plot of MSE for Sin basis k=1,...,18 on test set
figure;
plot(dim_k,log(MSE_k_1000_sin),'r')
xlabel('Sin basis (k)','FontSize',15)
ylabel('log( tse_{k}(S,T) )','FontSize',15)
axis tight;
grid on
hold off
set(gcf, 'Color', 'w');
export_fig('/Users/russeldaries/Documents/University College London/Computer
    Science/Courses/Mathematical Methods for Machine Learning/Assignments/
    Assignment 1/Report/LaTeX/report/Figures/k_dim_1000_sin', '-eps')
close

MSE_100_sin = [];
MSE_1000_sin = [];

% Computing results for 100 simulation runs
for j = 1:100
    x_rand_repeat_100_sin = rand([100,1]);
    x_rand_repeat_1000_sin = rand([1000,1]);
    g_x_noise_repeat_100_sin = [];

    for k=1:length(x_rand_repeat_100_sin)
        g_x_noise_repeat_100_sin(k) = (sin(2*pi*x_rand_repeat_100_sin(k)))^2 +
            noise_function(mu,sigma);
    end
    g_x_noise_repeat_100_sin = g_x_noise_repeat_100_sin';

    g_x_noise_repeat_1000_sin = [];
    for m=1:length(x_rand_repeat_1000_sin)
        g_x_noise_repeat_1000_sin(m) = (sin(2*pi*x_rand_repeat_1000_sin(m)))^2 +
            noise_function(mu,sigma);
    end
    g_x_noise_repeat_1000_sin = g_x_noise_repeat_1000_sin';
```

```
    SSE_k_repeat_100_sin = [];
    MSE_k_repeat_100_sin = [];
    SSE_k_repeat_1000_sin = [];
    MSE_k_repeat_1000_sin = [];

    for i=1:18
        PHI_repeat_100_sin = sin_basis(i,x_rand_repeat_100_sin);
        [Q,R] = qr(PHI_repeat_100_sin);
        w_iter_repeat_100_sin = (R)\(Q'*g_x_noise_repeat_100_sin);

        SSE_k_repeat_100_sin(i) = sum((g_x_noise_repeat_100_sin-(PHI_repeat_100_sin*w_iter_repeat_100_sin)).^2);
        MSE_k_repeat_100_sin(i) = SSE_k_repeat_100_sin(i)/length(g_x_noise_repeat_100_sin);

        PHI_repeat_1000_sin = sin_basis(i,x_rand_repeat_1000_sin);
        SSE_k_repeat_1000_sin(i) = sum((g_x_noise_repeat_1000_sin-(PHI_repeat_1000_sin*w_iter_repeat_100_sin)).^2);
        MSE_k_repeat_1000_sin(i) = SSE_k_repeat_1000_sin(i)/length(g_x_noise_repeat_1000_sin);
    end
    MSE_100_sin = [MSE_100_sin; MSE_k_repeat_100_sin];
    MSE_1000_sin = [MSE_1000_sin; MSE_k_repeat_1000_sin];

end
% Resultant MSE for 100 runs for Training and Test set with Sin basis
figure;
MSE_100_sin_avg = mean(MSE_100_sin);
MSE_1000_sin_avg = mean(MSE_1000_sin);
plot(dim_k,log(MSE_100_sin_avg),'b')
hold on
plot(dim_k,log(MSE_1000_sin_avg),'r')
xlabel('Sin basis (k)','FontSize',15)
ylabel('log(MSE_{avg})','FontSize',15)
axis tight;
grid on
hold off
set(gcf, 'Color', 'w');
leg=legend('te_{k}(S)_{avg}','tse_{k}(S,T)_{avg}','Location','Best')
set(leg,'FontSize',15)
export_fig('/Users/russeldaries/Documents/University College London/Computer Science/Courses/Mathematical Methods for Machine Learning/Assignments/Assignment 1/Report/LaTeX/report/Figures/k_dim_100_sin', '-eps')
close
```

Bibliography

[1] LEE DO Q. Numerically efficient methods for solving least squares problems. http://math.uchicago.edu/~may/REU2012/REUPapers/Lee.pdf, 2012.

[2] Moorse-penrose pseudoinverse. https://uk.mathworks.com/help/matlab/ref/pinv.html. Accessed: 2016-12-3.

[3] Matrix inverse. http://www.keithschwarz.com/interesting/code/lights-out/LightsOut.hh.html. Accessed: 2016-12-3.

[4] Marlow Anderson and Todd Feil. Turning lights out with linear algebra. *Mathematics Magazine*, 71(4):300–303, 1998.

[5] Lights out puzzle. http://mathworld.wolfram.com/LightsOutPuzzle.html. Accessed: 2016-11-25.

[6] Kendell Atkinson. *Introduction to Numerical Analysis*. John-Wiley-Sons, Canada, 1989.

[7] Lights out. http://www.keithschwarz.com/interesting/code/lights-out/LightsOut.hh.html. Accessed: 2016-11-23.