Irzam Hardiansyah
irzam@kth.se

FSF3581: Homework 4

Problem 1

Assume we seek the minimal expected value,

$$\min_{\theta\in\mathbb{R}^{(d+2)K}} \mathbb{E}[f(\theta,y)] \tag{2}$$

where $Y \sim N(0,1)$ *and* $f(\theta,y) := |\theta - y|^2$, $\theta \in \mathbb{R}$ *and* $y \in \mathbb{R}$.

Part 1a

Consider the iterations,

$$\begin{aligned} \theta_0 &= 1 \\ \theta_{n+1} &= \theta_n - \Delta t \frac{\partial f}{\partial \theta}(\theta_n,Y_n), \quad n = 0,1,2,... \end{aligned} \tag{3}$$

and show that there is a constant C *such that* $\mathbb{E}[|\theta_n|^2] \leq C$, *for* $n = 0,1,2,...$ *for a suitable choice of* $\Delta t \in \mathbb{R}$. *Determine this* Δt. *Here,* $Y_n, n = 0,1,2,...$ *are independent stochastic variables which are all standard normal distributed and* $\mathbb{E}[|\theta_n|^2]$ *denotes the expected value of* $|\theta_n|^2$. *Determine also* θ_* *where,*

$$\mathbb{E}[f(\theta_*,Y)] = \min_{\theta\in\mathbb{R}} \mathbb{E}[f(\theta,Y)]$$

and establish the convergence rate of $lim_{n\to\infty}\, \mathbb{E}[|\theta_n - \theta_*|^2]$.

Answer: First, we note that $Y_t\Delta t \approx Y_{t+1} - Y_t \equiv \Delta Y_t \sim N(0,\Delta t)$ and that

$$\begin{aligned} \frac{\partial f}{\partial \theta}(\theta,Y) &= \frac{\partial}{\partial \theta}|\theta - Y|^2 \\ &= 2\cancel{|\theta - Y|}\frac{(\theta - Y)}{\cancel{|\theta - Y|}} \\ &= 2(\theta - Y) \end{aligned}$$

Then, the iteration in (3) can be re-written as (changing the index, without the loss of generalization),

$$\begin{aligned} \theta_n &= \theta_{n-1} - \Delta t\,(2(\theta_{n-1} - Y_{n-1})) \\ &= \theta_{n-1} - 2\theta_{n-1}\Delta t + 2Y_{n-1}\Delta t \\ &= (1 - 2\Delta t)\theta_{n-1} + 2\Delta Y_{n-1} \\ &= (1 - 2\Delta t)^n\theta_0 + 2\sum_{k=0}^{n-1}(1 - 2\Delta t)^k\Delta Y_k \\ &= (1 - 2\Delta t)^n + 2\sum_{k=0}^{n-1}(1 - 2\Delta t)^k\Delta Y_k \end{aligned}$$

Then,

$$\begin{aligned}\|\theta_n\|_{L2}^2 &= (1-2\Delta t)^{2n} + \left(2\sum_{k=0}^{n-1}(1-2\Delta t)^k \Delta Y_k\right)\left(2\sum_{l=0}^{n-1}(1-2\Delta t)^l \Delta Y_l\right) \\ &= (1-2\Delta t)^{2n} + 4\left(2\sum_{k=l}(1-2\Delta t)^{2k}\Delta Y_k^2 + 2\sum_{k<l}(1-2\Delta t)^{k+l}\Delta Y_k \Delta Y_l\right)\end{aligned}$$

Next, taking the expectation of both sides,

$$\begin{aligned}\mathbb{E}[|\theta_n|^2] &= (1-2\Delta t)^{2n} + 8\sum_{k=l}(1-2\Delta t)^{2k}\underbrace{\mathbb{E}[\Delta Y_k^2]}_{=\Delta t} + 8\sum_{k<l}(1-2\Delta t)^{k+l}\underbrace{\mathbb{E}[\Delta Y_k \Delta Y_l]}_{=0} \\ &= (1-2\Delta t)^{2n} + 8\sum_{k=0}^{n-1}(1-2\Delta t)^{2k}\Delta t\end{aligned}$$

which will be finite if the sum on the right-hand side is a convergent series, i.e., with $|1-2\Delta t| < 1$ (in which case the first term on the right-hand side will also vanish as $n \to \infty$). Therefore,

$$\mathbb{E}[|\theta_n|^2] = (1-2\Delta t)^{2n} + 8\sum_{k=0}^{n-1}(1-2\Delta t)^{2k}\Delta t \le C \quad \iff 0 < \Delta t < 1$$

The optimal solution θ_* is obtained at the stationary point of the gradient of the objective, i.e., $\frac{\partial f}{\partial \theta}(\theta, Y) = 2(\theta - Y) = 0$; thus, $\theta_* = Y$.

Now, to establish the convergence rate of $\mathbb{E}[|\theta_n - \theta_*|^2]$ as $n \to \infty$ (where k, l, k', and l' are

indices from the same partition),

$$\begin{aligned}
\|\theta_n - \theta_*\|_{L2}^2 &= \left((1-2\Delta t)^n + 2\sum_{k=0}^{n-1}(1-2\Delta t)^k \Delta Y_k - Y_n\right)\left((1-2\Delta t)^n + 2\sum_{l=0}^{n-1}(1-2\Delta t)^l \Delta Y_l - Y_n\right) \\
&= \left((1-2\Delta t)^n + 2\sum_{k=0}^{n-1}(1-2\Delta t)^k \Delta Y_k - (Y_0 + \sum_{k'=0}^{n-1}\Delta Y_{k'})\right) \\
&\qquad \left((1-2\Delta t)^n + 2\sum_{l=0}^{n-1}(1-2\Delta t)^l \Delta Y_l - (Y_0 + \sum_{l'=0}^{n-1}\Delta Y_{l'})\right) \\
&= (1-2\Delta t)^{2n} + 4\left(2\sum_{k=l}(1-2\Delta t)^{2k}\Delta Y_k^2 + 2\sum_{k<l}(1-2\Delta t)^{k+l}\Delta Y_k \Delta Y_l\right) \\
&\quad - (1-2\Delta t)^n\left(2Y_0 + 2\sum_{k'=l'}\Delta Y_{k'}\right) - 2\sum_{k=0}^{n-1}(1-2\Delta t)^k \Delta Y_k\left(Y_0 + \sum_{l'=0}^{n-1}\Delta Y_{l'}\right) \\
&\quad - 2\sum_{l=0}^{n-1}(1-2\Delta t)^l \Delta Y_l\left(Y_0 + \sum_{k'=0}^{n-1}\Delta Y_{k'}\right) \\
&\qquad + \left(Y_0^2 + 2Y_0\sum_{k'=l'}\Delta Y_{k'} + 2\sum_{k'=l'}\Delta Y_{k'}^2 + 2\sum_{k'<l'}\Delta Y_{k'}\Delta Y_{l'}\right) \\
&= (1-2\Delta t)^{2n} + 8\sum_{k=l}(1-2\Delta t)^{2k}\Delta Y_k^2 + 8\sum_{k<l}(1-2\Delta t)^{k+l}\Delta Y_k \Delta Y_l \\
&\quad - 2(1-2\Delta t)^n Y_0 - 2(1-2\Delta t)^n \sum_{k'=l'}\Delta Y_{k'} - 4Y_0\sum_{k=l}(1-2\Delta t)^k \Delta Y_k \\
&\quad - 2\left(2\sum_{k=l'}(1-2\Delta t)^k \Delta Y_k^2 + 2\sum_{k<l'}(1-2\Delta t)^k \Delta Y_k \Delta Y_{l'}\right) \\
&\qquad - 2\left(2\sum_{l=k'}(1-2\Delta t)^l \Delta Y_l^2 + 2\sum_{l<k'}(1-2\Delta t)^l \Delta Y_l \Delta Y_{k'}\right) \\
&\quad + \left(Y_0^2 + 2Y_0\sum_{k'=l'}\Delta Y_{k'} + 2\sum_{k'=l'}\Delta Y_{k'}^2 + 2\sum_{k'<l'}\Delta Y_{k'}\Delta Y_{l'}\right)
\end{aligned}$$

Next, taking the expectation of both sides,

$$\begin{aligned}
\mathbb{E}[|\theta_n-\theta_*|^2] =&(1-2\Delta t)^{2n}+8\sum_{k=l}(1-2\Delta t)^{2k}\underbrace{\mathbb{E}[\Delta Y_k^2]}_{=\Delta t}+8\sum_{k<l}(1-2\Delta t)^{k+l}\underbrace{\mathbb{E}[\Delta Y_k\Delta Y_l]}_{=0}\\
&-2(1-2\Delta t)^n\underbrace{\mathbb{E}[Y_0]}_{=0}-2(1-2\Delta t)^n\sum_{k'=l'}\underbrace{\mathbb{E}[\Delta Y_{k'}]}_{=0}-4\underbrace{\mathbb{E}[Y_0]}_{=0}\sum_{k=l}(1-2\Delta t)^k\underbrace{\mathbb{E}[\Delta Y_k]}_{=0}\\
&-2\left(2\sum_{k=l'}(1-2\Delta t)^k\underbrace{\mathbb{E}[\Delta Y_k^2]}_{=\Delta t}+2\sum_{k<l'}(1-2\Delta t)^k\underbrace{\mathbb{E}[\Delta Y_k\Delta Y_{l'}]}_{=0}\right)\\
&-2\left(2\sum_{l=k'}(1-2\Delta t)^l\underbrace{\mathbb{E}[\Delta Y_l^2]}_{=\Delta t}+2\sum_{l<k'}(1-2\Delta t)^l\underbrace{\mathbb{E}[\Delta Y_l\Delta Y_{k'}]}_{=0}\right)\\
&+\left(\underbrace{\mathbb{E}[Y_0^2]}_{=1}+2\underbrace{\mathbb{E}[Y_0]}_{=0}\sum_{k'=l'}\underbrace{\mathbb{E}[\Delta Y_{k'}]}_{=0}+2\sum_{k'=l'}\underbrace{\mathbb{E}[\Delta Y_{k'}^2]}_{=\Delta t}+2\sum_{k'<l'}\underbrace{\mathbb{E}[\Delta Y_{k'}\Delta Y_{l'}]}_{=0}\right)\\
=&(1-2\Delta t)^{2n}+8\sum_{k=0}^{n-1}(1-2\Delta t)^{2k}\Delta t-8\sum_{k=0}^{n-1}(1-2\Delta t)^k\Delta t+1+2\underbrace{\sum_{k'=0}^{n-1}\Delta t}_{=T}
\end{aligned}$$

Then, taking the asymptotic limit, assuming $0<\Delta t<1$,

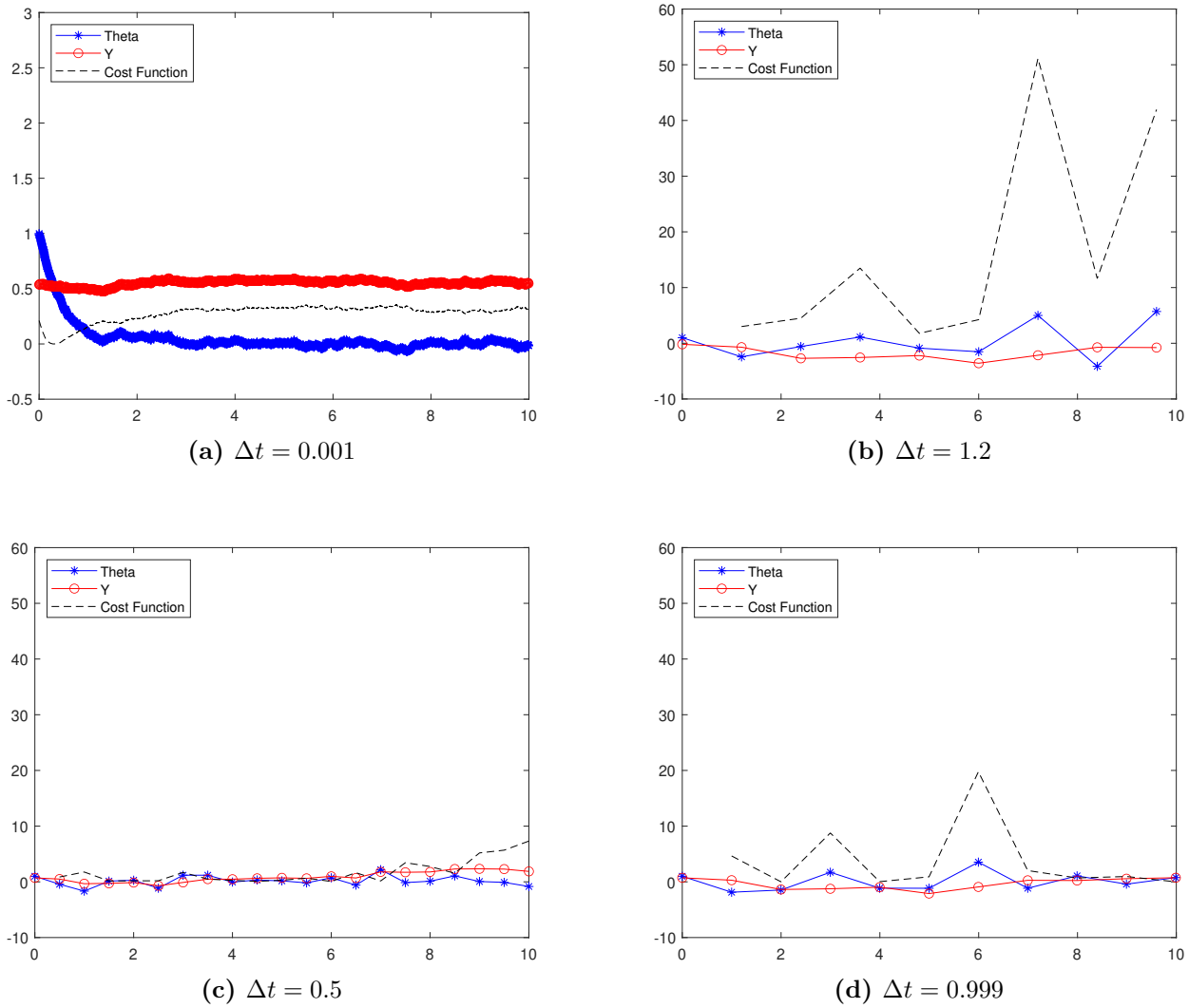
$$\begin{aligned}
\lim_{n\to\infty}\mathbb{E}[|\theta_n-\theta_*|^2] &= 1+2T+8\sum_{k=0}^{n-1}(1-2\Delta t)^{2k}\Delta t-8\sum_{k=0}^{n-1}(1-2\Delta t)^k\Delta t\\
&= 1+2T+8\Delta t\left(\frac{1}{1-(1-2\Delta t)^2}\right)-8\Delta t\left(\frac{1}{1-(1-2\Delta t)}\right)\\
&= 1+2T+8\cancel{\Delta t}\left(\frac{1}{4\cancel{\Delta t}(1-\Delta t)}\right)-8\cancel{\Delta t}\left(\frac{1}{2\cancel{\Delta t}}\right)\\
&= 2T-3+\frac{2}{1-\Delta t}\to\mathcal{O}\left(\frac{1}{|1-\Delta t|}\right)
\end{aligned}$$

Thus, the convergence rate is $\mathcal{O}\left(\frac{1}{1-\Delta t}\right)$ for $0<\Delta t<1$.

Part 1b

Write a Matlab program that performs the iterations in problem 1a and plots θ_n, n = 0,1,2,...,N.

Answer: We run the simulation from $t_0=0$ to $t_N=T=10$ with 8 different step lengths, including lengths very close to both 0 and 1, a length greater than 1, as well three intermediate lenghts, to see if the variance and rate of convergence (and stability) vary as a function of Δt. The results are presented in the table below as well as by plots in Figure 1, showing four example step lengths ($\Delta t = 0.001, 1.2, 0.5, 0.999$, respectively). Notice that the plot for $\Delta t = 0.001$ (Fig. 1a) is on a different scale of Y-axis (1/20th of the other 3 plots) due to the extremely small variations of all the series in this plot. MATLAB code for this simulation can be found in the Appendix.



(a) $\Delta t = 0.001$ **(b)** $\Delta t = 1.2$

(c) $\Delta t = 0.5$ **(d)** $\Delta t = 0.999$

Figur 1: Iterations of Y vs θ for 4 different step lengths

Δt	N (rounded)	$\frac{1}{1-\Delta t}$	$\mathbb{E}[\lvert\theta_n\rvert^2]$	$Var(f)$	$\mathbb{E}[\lvert\theta_n - \theta_*\rvert^2]$	Run Time (msec)
.001	10,000	1.001	.022	.006	.022	15.68
.1	100	1.111	.069	.841	.069	4.57
.3	33	1.429	.327	1.731	.327	1.87
.5	20	2.000	.769	4.510	.769	0.41
.7	14	3.333	2.162	7.427	2.162	3.22
.9	11	10.000	2.117	58.695	2.117	0.10
.999	10	1,000.00	2.782	39.144	2.782	0.06
1.2	8	-5.000	10.633	367.615	10.633	0.06

As expected, and confirming the theoretical analysis, for $\Delta t > 1$ (Fig. 1b), the estimates (θ) became unstable, gradually oscillating and increasing in absolute magnitude (diverging away from the target value). For all $\Delta t < 1$, θ converged to the target value (Figs. 1a, c, d). Results in the table above also showed that, in general, the larger the Δt, the larger the variance of both θ and the cost function f.

Part 1c

Show that the SGD iterations (3) are in fact forward Euler steps for an Ornstein-Uhlenbeck process. Formulate also the SGD based on computing the expected value exactly to solve the minimization problem (2), determine its convergence rate, and compare with the result in problem 1a.

Answer: As shown above in the answer to problem 1a, we showed that,

$$\begin{aligned}\theta_n &= \theta_{n-1} - \Delta t \frac{\partial f}{\partial \theta}(\theta_{n-1}, Y_{n-1}) \\ &= \theta_{n-1} - \Delta t \left(2(\theta_{n-1} - Y_{n-1})\right) \\ &= \theta_{n-1} - 2\theta_{n-1}\Delta t + 2Y_{n-1}\Delta t\end{aligned}$$

which conform to the general form of,

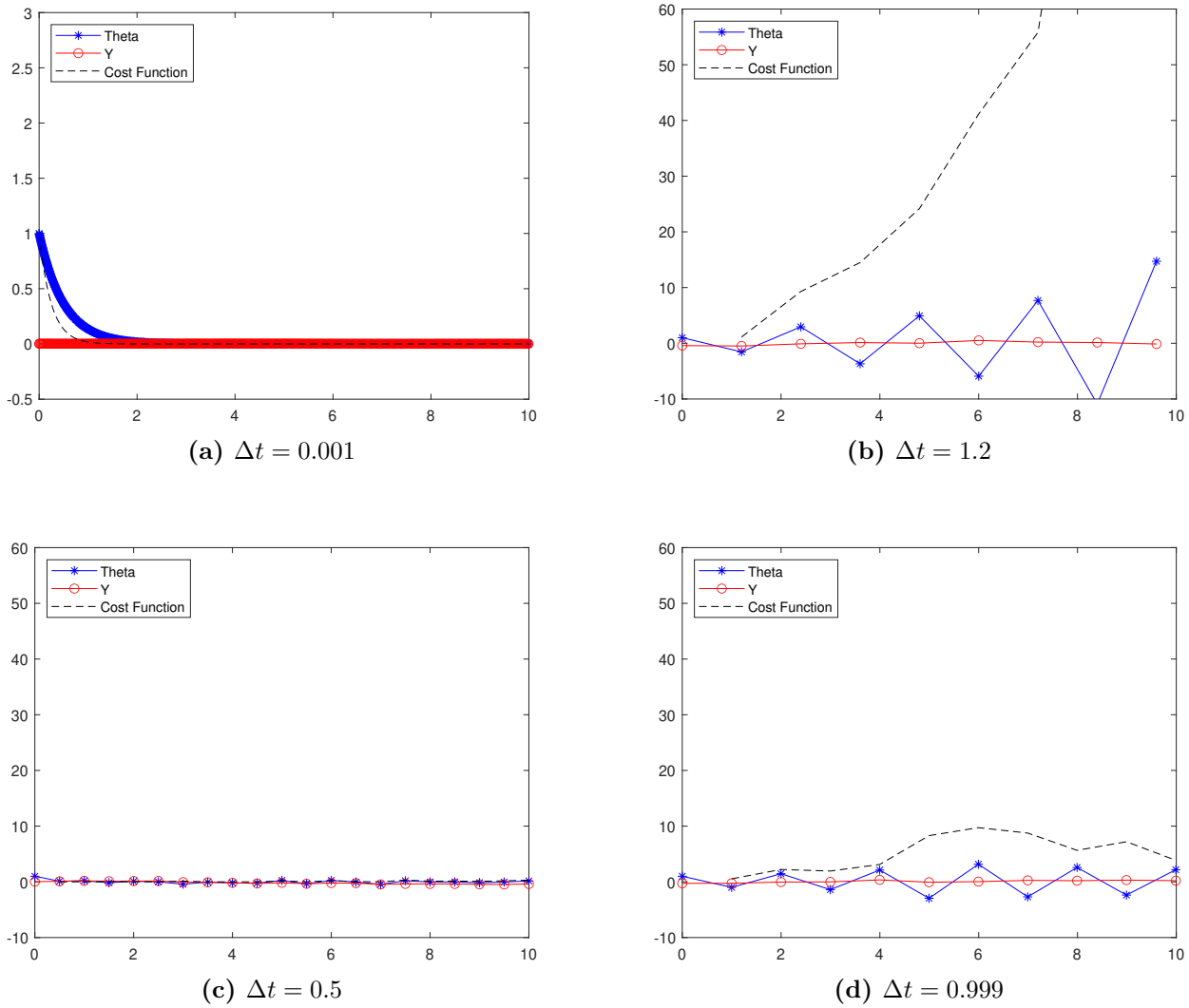
$$dx_t = -\alpha x_t dt + \sigma dW_t$$

that characterizes an Ornstein-Uhlenbeck process, with $\alpha = 2$, $\sigma = 2$, and $dW_t = Y dt = dY_t$, in discretized form (i.e., $\Delta t \approx dt$ and $\Delta Y_t \approx dY_t$).

We ran again the simulation with the same parameters as above, but this time using the mean of the $N = 100$ data points $Y_t \sim N(0,1)$ as the target value at each iteration. The results are shown in the table below and the plots in Figure 2. MATLAB code for this simulation can be found in the Appendix.

Δt	N (rounded)	$\frac{1}{1-\Delta t}$	$\mathbb{E}[\lvert\theta_n\rvert^2]$	$Var(f)$	$\mathbb{E}[\lvert\theta_n - \theta_*\rvert^2]$	Run Time (msec)
.001	10,000	1.001	.023	.012	.023	949.89
.1	100	1.111	.025	.006	.025	6.09
.3	33	1.429	.047	.002	.047	3.07
.5	20	2.000	.094	.011	.094	0.51
.7	14	3.333	.173	.084	.173	4.58
.9	11	10.000	.349	.149	.349	0.12
.999	10	1,000.00	5.317	10.480	5.317	0.08
1.2	8	-5.000	58.730	5,621	58.730	0.12

We observed similar behaviour as in Part 1b but with the following differences: i) the variance of θ, variance of f, and convergence rate were substantially smaller; ii) it seemed to be much more unstable for $\Delta t > 1$ and step lengths closer to this threshold; iii) the running times were all notably larger. Thus, although improving accuracy (in the sense of $L2$-convergence), this approach entails a large increase in computational cost due to the averaging operation in each iteration.



(a) $\Delta t = 0.001$ **(b)** $\Delta t = 1.2$

(c) $\Delta t = 0.5$ **(d)** $\Delta t = 0.999$

Figur 2: As in Figure 1, but now using $\mathbb{E}[Y_t]$ as target

Problem 2

Consider the minimization problem,

$$\min_{\theta \in \mathbb{R}^{3xK}} \mathbb{E}[|\alpha_\theta(X) - f(X)|^2] \tag{4}$$

$$f(x) = \left| x - \frac{1}{2} \right|^2, \quad x \in \mathbb{R}$$

where $X \sim \mathcal{U}(-4,4)$,

$$\alpha_\theta = \sum_{k=1}^{K} \theta_k^1 \sigma(\theta_k^2 x + \theta_k^3), \quad x \in \mathbb{R}, \quad \theta = (\theta_k^i) \in \mathbb{R}^{3xK}$$

and,

$$\sigma(x) = \frac{1}{1 + exp(-x)}, \quad x \in \mathbb{R}$$

The term $E_c(\theta) := \mathbb{E}[|\alpha_\theta(X) - f(X)|^2]$ *is commonly called the loss function or cost function. A solution can be approximated using the following Stochastic Gradient Descent (SGD) optimizer:*

- *Generate an initial guess* $\theta_0 \in \mathbb{R}^{3xK}$ *from some distribution.*
- *Generate* N *data points* $x_1, x_2,..., x_N$ *independently from the uniform distribution* $\mathcal{U}(-4,4)$ *and compute the corresponding function values* $f_n = f(x_n), n = 1, 2,..., N$.
- *Take* $T \in (0, \infty)$ *and denote* $\Delta t = \frac{T}{M}$ *the step length for some positive integer* M.
- *For* $m = 1, 2, ..., M$ *do:*
  - *Take a random* $i \in 1,2,..., N$.
  - $\theta_{m+1} = \theta_m - 2\Delta t \nabla_\theta |\alpha_{\theta_m}(x_i) - f_i|^2$.

An approximate solution to the minimization problem (4) is then given by θ_M.

The SGD optimizer algorithm tries to minimize the approximation $\mathbb{E}(\theta) := \sum_{n=1}^{N} \frac{|f(x_n)-\alpha_\theta(x_n)|^2}{N}$, *of the loss function, for the set* $\{(x_n, f(x_n))\}_{n=1}^{N}$ *of training points. It hence called* ***training error****.*

It is important to know how the neural network generalizes to data outside the set of training data. Thus, one can compute the ***generalization error*** *(also called* ***test error****)* $E_g(\theta) := \sum_{n=1}^{\tilde{N}} \frac{|f(\tilde{x}_n)-\alpha_\theta(\tilde{x}_n)|^2}{\tilde{N}}$, *where* $\{(\tilde{x}_n, f(\tilde{x}_n))\}_{n=1}^{\tilde{N}}$ *is roughly disjoint from the set of training points. A small training error but with a large generalization error is a symptom of* ***overfitting****.*

Parts 2.3.1 and 2.3.2

1. Gradually first increase and then gradually decrease dt. Explain what you see. A suitable range to test is $0.05 > dt$, *cf. Part 1.*
2. Plot the training error.

Answer: The results of the simulation are presented in Figure 3. Python code for this simulation is given in Appendix.

1. As can be seen from the plots, using different step lengths (dt) has the effect on the quality of learning by the algorithm, where smaller dts resulted in better and less variable test errors, while larger dts, up to dt = 0.05, resulted in more variable test errors. For all dt > 0.05, the algorithm failed to learn the function properly, resulting in much larger test errors that were practically flat (non-decreasing) even after many iterations.

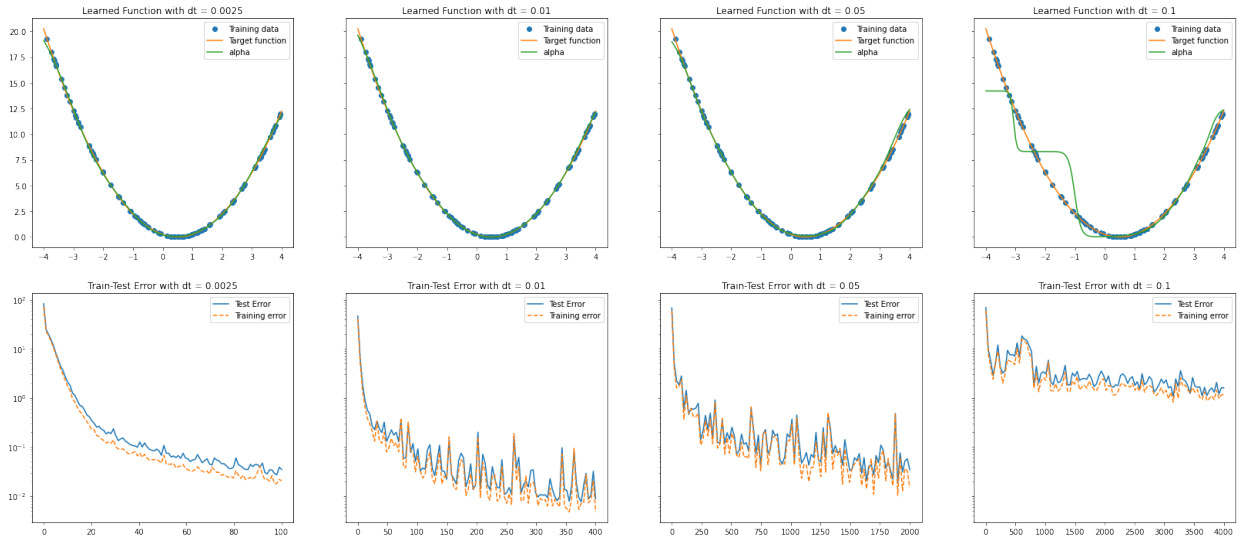
2. The training errors can be seen to be quite close to (and below) the test errors for all the different step lengths tried in our simulation.

Part 2.3.3

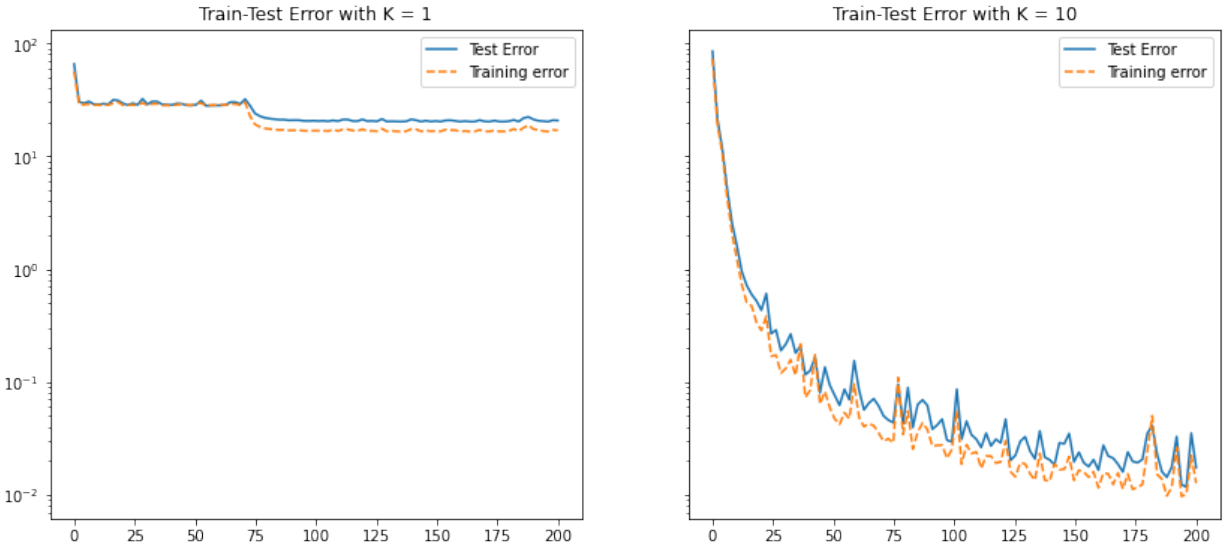
Set $K = 1$. *What do you see?*

Answer: The results of the simulation are presented in Figure 4. Python code for this simulation is given in Appendix.

As can be seen on the left plot of Fig. 4, with only a single hidden neuron (K = 1), the algorithm failed to learn the function properly, indicating a lack of expressive power in the



Figur 3: Learned functions (top row) and training-test errors (bottom row) for different step lengths (dt). All plots made with K = 10 and N = 100.



Figur 4: Training-test errors for different number of hidden neurons (K). All plots made with dt = 0.005 and N = 100.

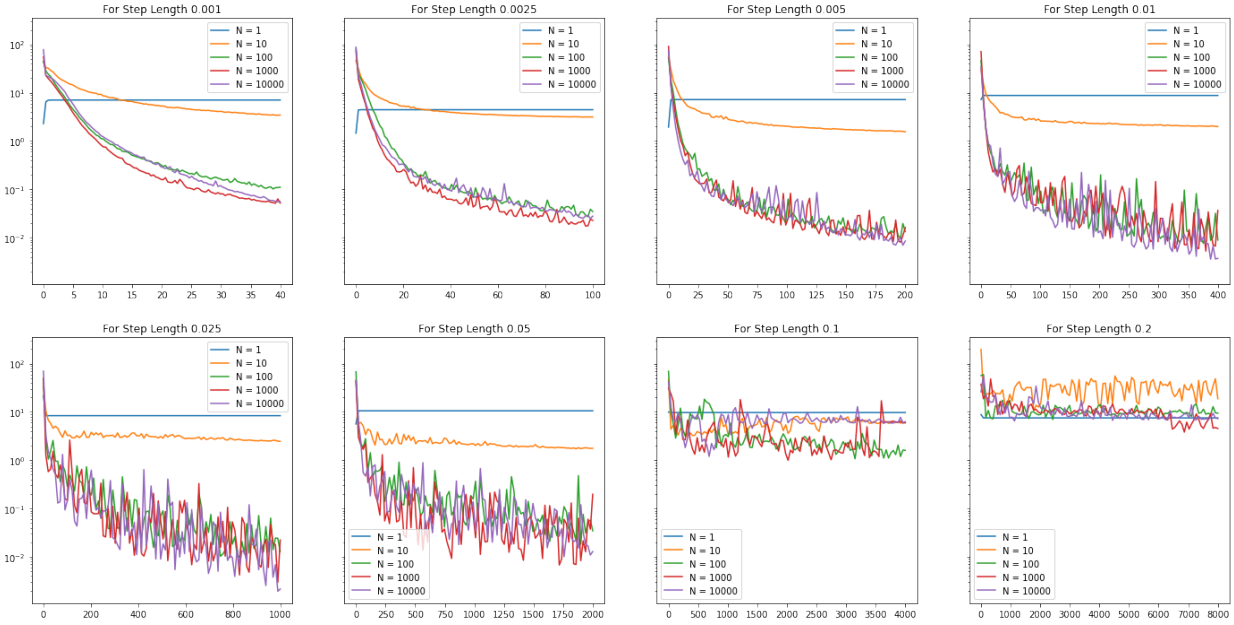
model to cope with the function complexity. This resulted in much larger errors during both training and testing, which did not progress meaningfully even after many iterations.

Part 2.3.4

For K = 10, what is a suitable choice of N? Motivate your answer.

Answer: Using $N \in \{1, 10, 100, 10{,}00, 10{,}000\}$, we ran the simulation again by keeping all the other parameters the same. The results are presented in Figure 5. Python code for this simulation is given in Appendix.

As can be seen from the plots in Fig. 5, for all sample sizes (that we tried) $N < 100$, the test errors were much higher than for any case where $N \geq 100$. For $N \geq 100$ and $dt \leq 0.05$, the



Figur 5: Test errors for different number of data points / sample sizes (N) and step lengths (dt). All plots made with K = 10.

test errors were roughly of the same magnitude, only with different variances. Therefore, we conclude that a suitable choice of sample size would be $N \geq 100$.

Part 2.3.5

Give an example of parameter settings that gives a small value of training error but a large value of the generalization error.

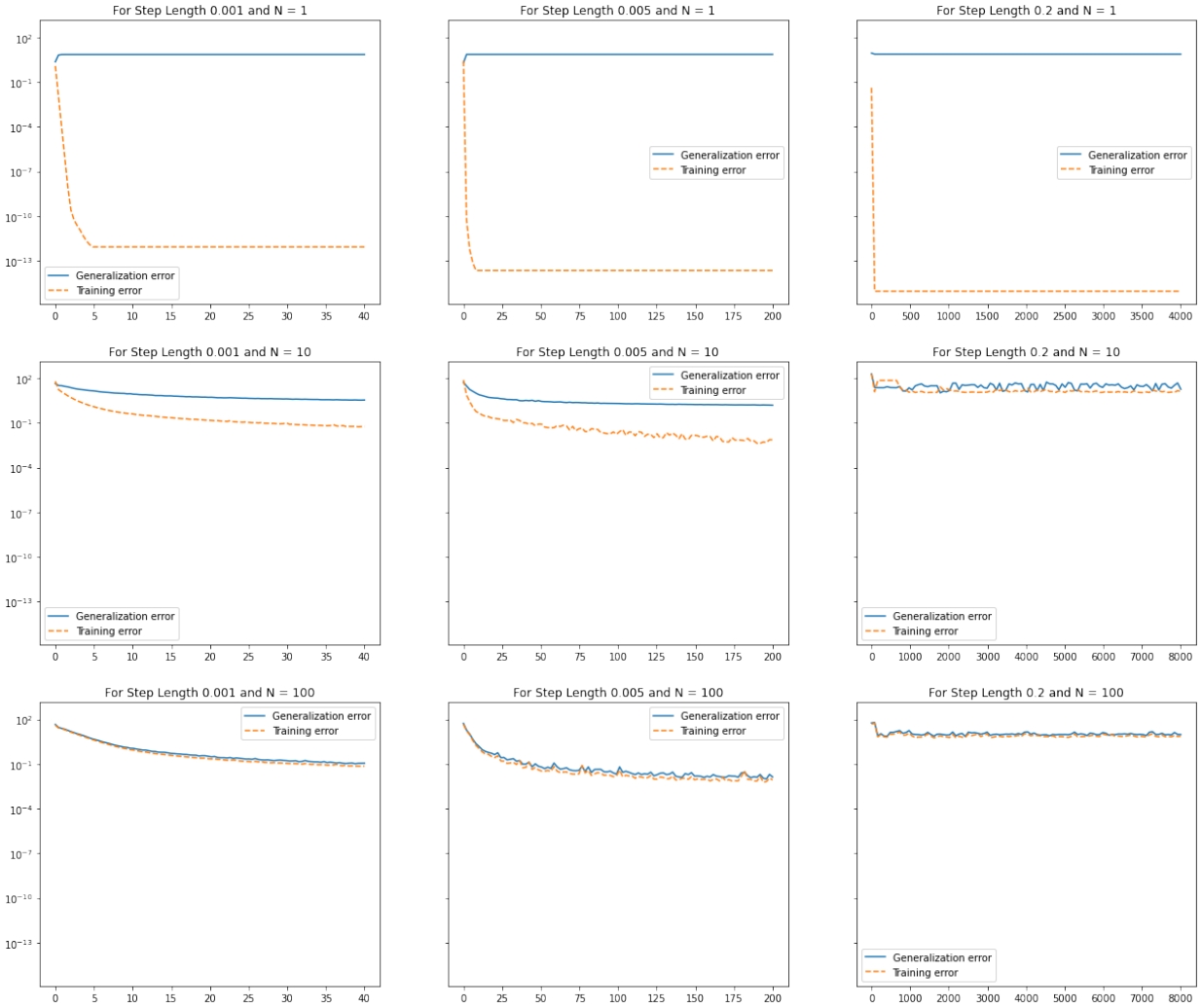
Answer: We plotted some examples of such cases in Figure 6: where the training errors could keep improving (getting lower and lower) while the test errors stagnated - the cases of overfitting.

As seen in the plots, using a very small sample size, e.g., $N = 1$ or $N = 10$, resulted in a large divergence between training and generalization errors, where the former decreased with a pace much greater than the latter. Therefore, all parameter settings which involve a small sample size, e.g. $N \leq 10$ will potentially lead to overfitting. Python code for this simulation is given in Appendix.

Part 2.3.6

This problem aims to show how using the sinus function instead of the sigmoid function as activation function can affect outcome for a given bias initialization.

Add the parameter:
`bias_initializer = bias_init_unif(minval=20*np.pi, maxval=22*np.pi)`
as an argument to the `keras.layers.Dense` *when defining* `Hidden_layer` *and don't forget to define:*



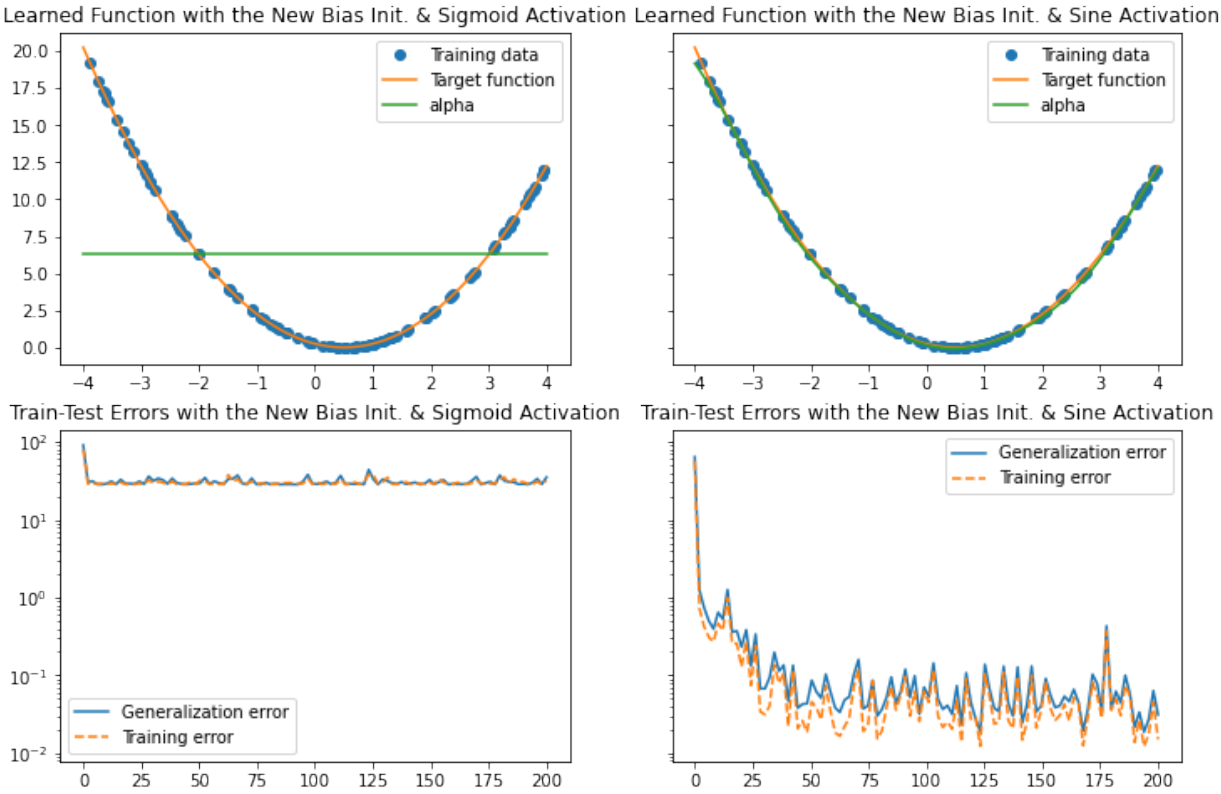
Figur 6: Train-Test errors for different step lengths (dt) and sample sizes (N). All plots made with K = 10.

`bias_init_unif=tf.initializers.random.uniform`.

Does the SGD converge? Another activation function that can be used instead of sigmoid function is the sinus function. Change the activation function in ***`Hidden_layer`*** *from 'sigmoid' to* ***`tf.sin`****. Does the SGD converge now? Explain the result.*

Answer: The results of the simulation are presented in Figure 7. Python code for this simulation is given in Appendix.

As can be seen from the plots in Fig. 7, using the new bias initialization while keeping the sigmoid activation function for the hidden layer resulted in the complete failure of the algorithm to learn the target function (as the alpha is just a straight line on the plot). The reason for this failure was that such large bias values produced a very small $exp(-x)$ term in the sigmoid function, resulting in a virtually flat activation with a value (very close to) 1. Changing the activation function to a sine function resolved this problem as the bias values were just squeezed to a value between -1 and 1, thus actually normalizing the activations which in turn facilitated more stable (numerically) backpropagation gradient estimation.



Figur 7: Learned function and train-test errors for the algorithm with the new bias initialization and with sigmoid (left) and sine (right) activation functions

Part 2.3.7

Problem (4) is a problem in one dimension. Reformulate problem (4) to be in dimension d, where d is a positive integer. Also, extend the program so that it approximates a solution for a general positive integer d.

Answer: The following provides a reformulation of (4) for general data space in d dimension, where $d > 0$, assuming that $f : \mathbb{R}^d \to \mathbb{R}$ (i.e., the function produces a scalar output and thus can still be approximated by a single neuron in the output layer of NN),

$$\min_{\theta} \mathbb{E}[|\alpha_\theta(\mathbf{X}) - f(\mathbf{X})|^2] \tag{5}$$

$$f(\mathbf{x}) = \left\langle \left(\mathbf{x} - \frac{1}{2}\right), \left(\mathbf{x} - \frac{1}{2}\right) \right\rangle, \quad \mathbf{x} \in \mathbb{R}^d$$

where $\mathbf{X} \sim \mathcal{U}_d((-4,4)^d)$,

$$\alpha_\theta = \sum_{k=1}^{K} \theta_k^1 \sigma(\theta_k^2 \cdot \mathbf{x} + \theta_k^3), \quad \mathbf{x} \in \mathbb{R}^d, \quad \theta_k^1, \theta_k^3 \in \mathbb{R}, \quad \theta_k^2 \in \mathbb{R}^d$$

and,

$$\sigma(x') = \frac{1}{1 + exp(-x')}, \quad x' = \theta_k^2 \cdot \mathbf{x} + \theta_k^3 \in \mathbb{R}, \quad \mathbf{x}, \theta_k^2 \in \mathbb{R}^d, \quad \theta_k^3 \in \mathbb{R}$$

Appendix: MATLAB Code

Problem 1

```
%% (1b) ML F-E iterations
clear;

steps = [1e-3,.1,.3,.5,.7,.9,.999,1.2]; %number of different (equidistant)
    step lengths

durations = {}; fs = {}; Ys ={}; thetas = {}; devs = {}; convrgs = {}; t =
    {};
N = []; MSE_f = []; MSE_theta = []; MSE_convrg = []; Big_O = [];

rng('default')
s = rng;
for i=1:length(steps)

    tStart = tic;
    T = 10; dt = steps(i); t{i} = 0:dt:T;
    N(i) = round(T/dt);
    r = -2; sigma = 2;

    theta = []; Y = []; f = []; dev = [];

    theta(1) = 1;
    Y(1) = randn(1,1);
    %rng(s)
    randn('state',0);
    for k=2:(N(i)+1)
        draw_Y = randn(1,1);
        dY = draw_Y*dt; % Wiener increments
        %or
        %dY = sqrt(dt)*randn(1,1); % Wiener increments
        theta(k) = theta(k-1) + theta(k-1)*(r*dt) + sigma*dY; % processes
            at next time step
        Y(k) = Y(k-1) + dY;
        f(k-1) = (abs(theta(k) - Y(k)))^2;
        dev(k-1) = theta(k) - Y(k);
    end
    convrg = theta - Y(end);

    thetas{i} = theta;
    Ys{i} = Y;
    fs{i} = f;
    devs{i} = dev;
    convrgs{i} = convrg;
    MSE_f(i) = var(f);
```

```
    MSE_theta(i) = var(theta);
    MSE_convrg(i) = var(convrg);
    Big_O(i) = 1/(1-dt);
    durations{i} = toc(tStart);
end
toc(tStart)

idx = 1 %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-.5,3]);
legend('Theta','Y','Cost Function','Location','northwest')
%print -depsc hw4Fig1a

idx = length(steps) %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-10,60]);
legend('Theta','Y','Cost Function','Location','northwest')
%print -depsc hw4Fig1b

idx = 4 %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-10,60]);
legend('Theta','Y','Cost Function','Location','northwest')
%print -depsc hw4Fig1c

idx = length(steps)-1 %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-10,60]);
legend('Theta','Y','Cost Function','Location','northwest')
%print -depsc hw4Fig1d

figure;
plot(steps,log(MSE_f),'x-',steps,log(MSE_theta),'o-',steps,log(MSE_convrg),'*-',steps,
legend('Variance of Cost','Variance of Theta','Convergence
   Rate','Big-O','Location','northwest')
%print -depsc hw4Fig1e

%% (1c) ML F-E iterations with expectation of Y
clear;

steps = [1e-3,.1,.3,.5,.7,.9,.999,1.2]; %number of different (equidistant)
   step lengths

durations = {}; fs = {}; Ys ={}; thetas = {}; devs = {}; convrgs = {}; t =
   {};
```

```
N = []; MSE_f = []; MSE_theta = []; MSE_convrg = []; Big_O = [];

rng('default')
s = rng;
for i=1:length(steps)

    tStart = tic;
    T = 10; dt = steps(i); t{i} = 0:dt:T;
    N(i) = round(T/dt);
    r = -2; sigma = 2;

    theta = []; Y = []; f = []; dev = [];

    theta(1) = 1;
    Y(1) = mean(randn(1,N(i)));
    %rng(s)
    randn('state',0);
    for k=2:(N(i)+1)
        mean_Y = mean(randn(1,N(i)));
        dY = mean_Y*dt; % Wiener increments
        theta(k) = theta(k-1) + theta(k-1)*(r*dt) + sigma*dY; % processes
            at next time step
        Y(k) = Y(k-1) + dY;
        f(k-1) = (abs(theta(k) - Y(k)))^2;
        dev(k-1) = theta(k) - Y(k);
    end
    convrg = theta - Y(end);

    thetas{i} = theta;
    Ys{i} = Y;
    fs{i} = f;
    devs{i} = dev;
    convrgs{i} = convrg;
    MSE_f(i) = var(f);
    MSE_theta(i) = var(theta);
    MSE_convrg(i) = var(convrg);
    Big_O(i) = 1/(1-dt);
    durations{i} = toc(tStart);
end
toc(tStart)

idx = 1 %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-.5,3]);
legend('Theta','Y','Cost Function','Location','northwest')
print -depsc hw4Fig2a

idx = length(steps) %choose which step length to use
```

```
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-10,60]);
legend('Theta','Y','Cost Function','Location','northwest')
print -depsc hw4Fig2b

idx = 4 %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-10,60]);
legend('Theta','Y','Cost Function','Location','northwest')
print -depsc hw4Fig2c

idx = length(steps)-1 %choose which step length to use
figure;
plot(t{idx},thetas{idx},'*-blue',t{idx},Ys{idx},'o-red',t{idx}(2:end),fs{idx},'--black
ylim([-10,60]);
legend('Theta','Y','Cost Function','Location','northwest')
print -depsc hw4Fig2d

figure;
plot(steps,log(MSE_f),'x-',steps,log(MSE_theta),'o-',steps,log(MSE_convrg),'*-',steps,
legend('Variance of Cost','Variance of Theta','Convergence
   Rate','Big-O','Location','northwest')
%print -depsc hw4Fig2e
```

Problem 2

Please see the accompanying Python script file (FSF3581_HW4_Q2.py) or Jupyter notebook file (FSF3581_HW4_Q2.ipynb) for all the simulation codes. All codes originally made in Jupyter notebook.