38 Monte Carlo and Grid-Based Techniques for Stochastic Simulation

In this problem you will compare the performance of random vs. regular sampling on a specific stochastic dynamics problem.

The system we are considering is a simple rotary mass, controlled by a motor:

$$J\ddot{\phi} = \tau = k_t i,$$

where J is the mass moment of inertia, ϕ is its angular position, τ is the control torque, k_t is the torque constant of the motor, and i is the electrical current applied. While this is a simple control design problem for given values of J and k_t, the situation we study here is when these are each only known within a *range* of values. In particular, J is described as a uniform random variable in the range $[5, 15] kg \cdot m^2$, and k_t is a uniform random variable in the range $[4, 6] Nm/A$. The basic question we ask is: if the control system is designed for a nominal condition, say $J = 10 kg \cdot m^2$ and $k_t = 5 Nm/A$, how will the closed-loop system vary in its response, for all the possible J and k_t?

This is a question of stochastic simulation, that is, finding the statistics of a function output, given the statistics of its input. The code fragment provided below applies Monte Carlo and grid-based approaches to find the mean and variance of the function $\cos(y)$, when y is uniformly distributed in the range $[2, 5]$. Try running this a few times and notice the effects of changing N. The grid-based approach is clearly giving a good result with far less work than MC - for this example with only one random dimension. In general, the grid-based methods suffer greatly as the d dimension increases; for trapezoidal integration, the error goes as $1/N^{2/d}$, whereas for Monte Carlo it is simply $1/N^{1/2}$ for any d!

1. For the nominal system model (as above) design a proportional-derivative controller so that the closed-loop step response reaches the commanded angle for the first time in about one second and the maximum overshoot is twenty percent. The closed-loop system equation is

$$\begin{aligned} J\ddot{\phi} &= k_t(-k_p(\phi - \phi_{desired}) - k_d\dot{\phi}) \longrightarrow \\ J\ddot{\phi} + k_t k_d \dot{\phi} + k_t k_p \phi &= k_t k_p \phi_{desired}. \end{aligned}$$

   Remember that if you write the left-hand side of the equation as $\ddot{\phi} + 2\zeta\omega_n + \omega_n^2$, you can tune this up quite easily because the overshoot scales directly with damping ratio ζ, and you can then adjust ω_n to get the right rise time. Show a plot of the step response and list your two gains k_p and k_d.

   The step response for the nominal system is shown, along with the "four corners" of the parameter space, that is, at the max and min combinations of J and k_t. The gains I used are derived from $\zeta = 0.455$ and $\omega_n = 2.3 rad/s$; they are $k_p = 10.58$ and $k_d = 4.19$.

2. Keeping your controller for the nominal system, use the Monte Carlo technique to calculate the mean and the variance of the overshoot z, over the random domain that

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
N = 1000; % how many trials to run

%%%%%%%%%%%%%%
% Monte Carlo
%%%%%%%%%%%%%%
for i = 1:N,
    q = 2 + 3*rand ;        % random sample from the random domain
    z(i) = cos(q);          % evaluate the function
end;
meanzMC = sum(z)/N ;               % calculate mean
varzMC = sum((z-meanzMC).^2)/N ;   % calculate variance

%%%%%%%
% Grid
%%%%%%%
for i = 1:N,
    q = 2 + 3/N/2 + (i-1)*3/N ;  % regular sample from the random domain
    z(i) = cos(q) ;
end;
meanzGrid = sum(z)/N ;
varzGrid = sum((z-meanzGrid).^2)/N ;

disp(sprintf('Means       MC: %7.4g   Grid: %7.4g   EXACT:  %7.4g', ...
    meanzMC, meanzGrid, (sin(5)-sin(2))/3 ));
disp(sprintf('Variances   MC: %7.4g   Grid: %7.4g', varzMC, varzGrid));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

covers all the possible J and k_t values. Show a plot of mean $\bar{z}$ vs. N, and a plot of variance σ_z^2 vs. N, for $N = [1, 2, 5, 10, 20, 50, 100, 200, ...]$. About how high does N have to be to give two significant digits?

The MC version is pretty noisy, and you'd need at least some hundreds of trials to say with confidence that $\bar{z}$ is between 0.19 and 0.20; ditto for the variance. Clearly a thousand or more trials is preferable.

3. Keeping your controller for the nominal system, use the trapezoidal rule to calculate the mean and variance of z. Let n_1 and n_2 be the number of points in the J and the k_t dimensions, and set $n_1 = n_2$, so that $N = n_1 n_2$. Show plots of $\bar{z}$ and σ_z^2 vs. N, to achieve at least two significant digits.

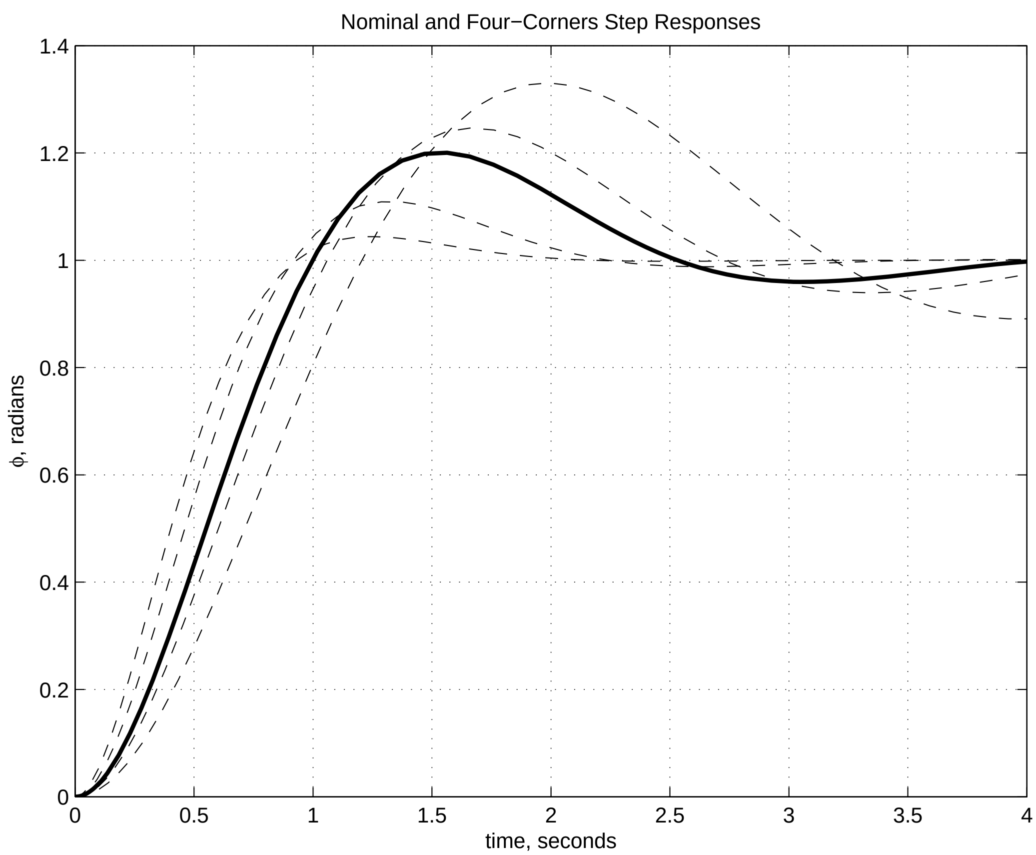
   We see the grid-based calculation is much cleaner, evidently reaching very stable values *of $\bar{z}$ and $var(z)$ in only a hundred or so trials!*

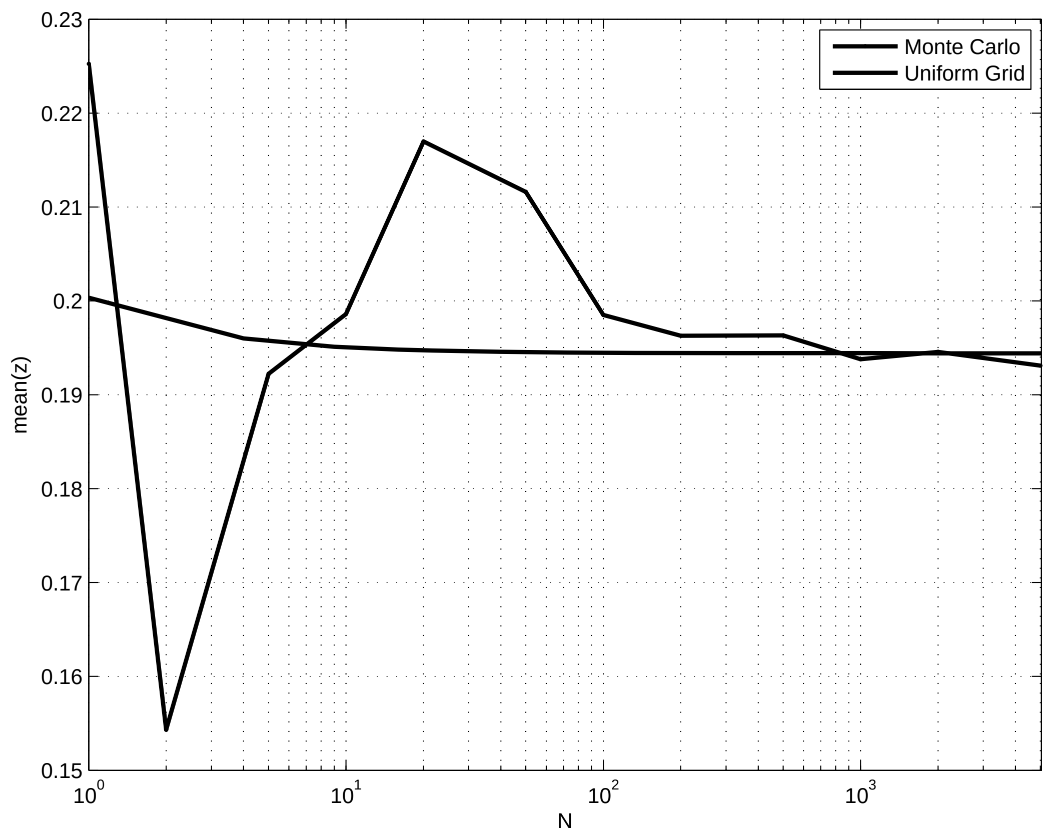
4. Comparing the curves you obtained, which is the superior technique for this problem, and how can you tell?

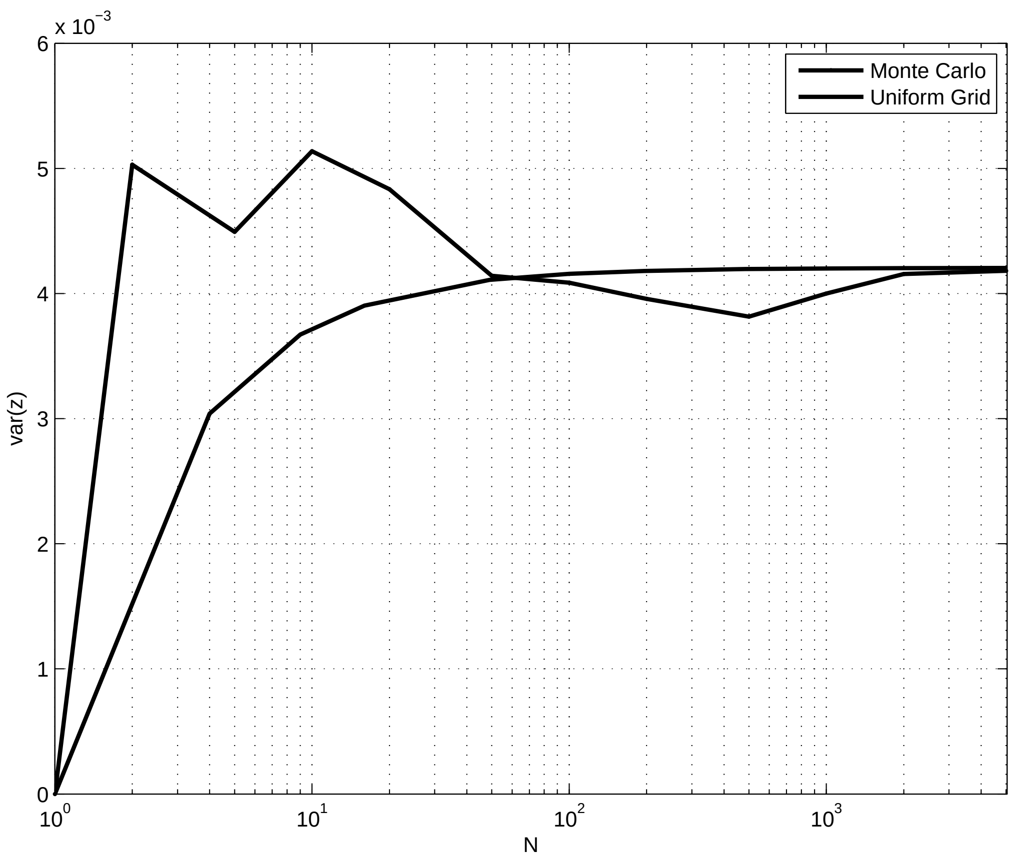
   The grid!

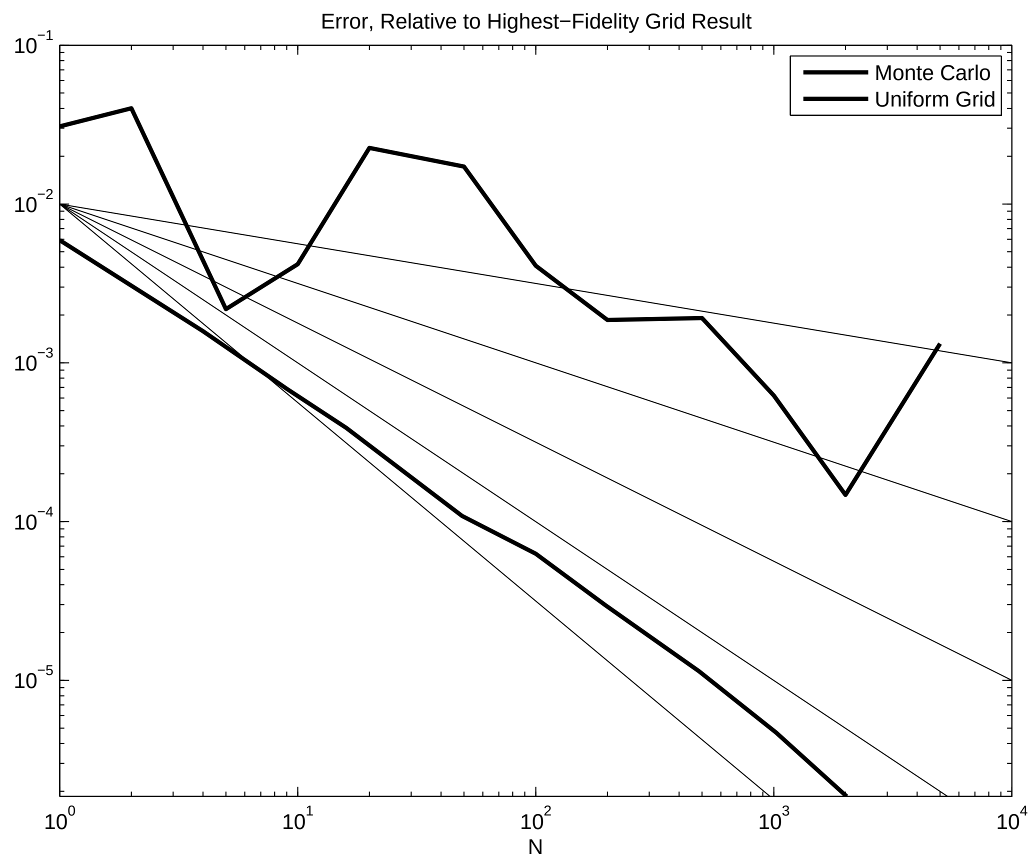
5. Taking your highest-fidelity result for $\bar{z}$ (probably the grid-based calculation with high N) as *truth*, you can calculate the apparent errors in $\bar{z}$ for each method, as a function of N. Making a log-log plot of the absolute values of these errors, can you argue that the error scaling laws $1/N^{1/2}$ (MC) and $1/N^{2/d} = 1/N$ (grid) hold?

   See the last plot. The thin lines indicate trends for $N^{-1/4}, N^{-1/2}, N^{-3/4}, N^{-1}, N^{-5/4}$. The MC points are scattered but generally fit the $N^{-1/2}$ line. The grid data fit the N^{-1} line, and since the dimension is two, it all works out.




0.23
0.22
0.21
0.2
0.19
0.18
0.17
0.16
0.15
mean(z)
Monte Carlo
Uniform Grid
10^0
10^1
10^2
10^3
N


x 10^−3
6
5
4
3
2
1
0
var(z)
10^0
10^1
10^2
10^3
N
Monte Carlo
Uniform Grid


Error, Relative to Highest−Fidelity Grid Result
Monte Carlo
Uniform Grid
10^−1
10^−2
10^−3
10^−4
10^−5
10^0
10^1
10^2
10^3
10^4
N

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Study MC vs. grid-based sensitivity
% FSH MIT 2.017 November 2009
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

global kp kd J kt ;

Jl = 5 ; Ju = 15 ;   % lower and upper values of the MMOI
ktl = 4 ; ktu = 6 ;   % lower and upper values of torque constant

zeta = .455 ; % set the CL damping ratio and natural frequency
wn = 2.3 ;

tfinal = 4 ;  % final time for all simulations

odeset('AbsTol',1e-4, 'RelTol',1e-2); % lower the accuracy a bit = faster

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% first, show that the gains achieve the desired step response with
% the nominal system

J = (Jl + Ju)/2 ;       % nominal values = midpoints
kt = (ktl + ktu)/2 ;

kp = J*wn^2/kt ;        % control gains - work these out for the nominal
kd = 2*zeta*wn*J/kt ;   % case and then leave them alone

[t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);

figure(1);clf;hold off;
plot(t,s(:,2),'LineWidth',2);
grid;
xlabel('time, seconds');
ylabel('\phi, radians');

% also run the four corners to make sure the time scale is about right

J4corners = [Jl Jl Ju Ju];
kt4corners = [ktu ktl ktl ktu] ;
figure(1);hold on;
for i = 1:4,
```

```
    J = J4corners(i);
    kt = kt4corners(i);
    [t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);
    plot(t,s(:,2),'--');
end;
title('Nominal and Four-Corners Step Responses');
pause ;

%%%%%%%%%%%%%%%%
% do the MC runs
%%%%%%%%%%%%%%%%

% Nvec carries the sizes of the ensembles for which we will do statistics
Nvec = [1,2,5,10,20,50,100,200,500,1000,2000,5000] ;

% Note that as written, we do just the largest ensemble, and then
% use portions of it for the statistics

tic;
for i = 1:max(Nvec),
    J = (Ju-Jl)*rand + Jl ; % generate random J in the domain
    kt = (ktu-ktl)*rand + ktl ; % generate random kt in the domain
    [t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);
    z(i) = max(s(:,2)-1); % get the overshoot
    if rem(i,100) == 0,
        disp(sprintf('Done with %d/%d', i,max(Nvec)));
    end;
end;
toc ;

% calculate the mean and variance for subsets given by Nvec
for k = 1:length(Nvec);
    meanzMC(k) = mean(z(1:Nvec(k))) ;
    varzMC(k) = var(z(1:Nvec(k)),1) ;
end;

figure(2);clf;hold off;
semilogx(Nvec,meanzMC,'.-','LineWidth',2) ;
a=axis ; axis([min(Nvec) max(Nvec) a(3) a(4)]);
grid;

figure(3);clf;hold off;
semilogx(Nvec,varzMC,'.-','LineWidth',2) ;
a=axis ; axis([min(Nvec) max(Nvec) a(3) a(4)]);
```

```
grid;

pause(.1);

%%%%%%%%%%%%%%%%%%%
% do the grid runs
%%%%%%%%%%%%%%%%%%%

% N1vec is the set of (one-diminsion) ensemble sizes for which we will
% compute statistics.  Note we will use N1 = N2 so that the total number
% of evaluations is N = N1 * N2
N1vec = [1 2 3 4 7 10 14 22 32 45 71];

% Most of the grids don't overlap, so we just use the brute force - do
% all the ensembles and their statistics independently.  It's more
% expensive than what we did for MC

tic;
for k = 1:length(N1vec),
    clear z ;
    for i = 1:N1vec(k),
        for j = 1:N1vec(k),
            J = Jl + (Ju-Jl)/N1vec(k)/2 + (i-1)*(Ju-Jl)/N1vec(k) ;
            kt = ktl + (ktu-ktl)/N1vec(k)/2 + (j-1)*(ktu-ktl)/N1vec(k);
            [t,s] = ode45('MCvsGridDeriv',[0 tfinal],[0 0]);
            z(i,j) = max(s(:,2)-1);
        end;
    end;

    meanzGrid(k) = mean(mean(z)); % the mean is easy...
    % but the variance calculation takes a little more attention
    sumsq = 0 ;
    for i = 1:N1vec(k),
        for j = 1:N1vec(k),
            sumsq = sumsq + (z(i,j) - meanzGrid(k))^2 ;
        end;
    end;
    varzGrid(k) = sumsq / N1vec(k)^2 ;

    disp(sprintf('Done with %d/%d', sum(N1vec(1:k).^2),sum(N1vec.^2)))
end;
toc;

figure(2);hold on;
```

```
semilogx(N1vec.^2,meanzGrid,'r','LineWidth',2);
axis('auto');a=axis ; axis([min([Nvec,N1vec.^2]) max([Nvec,N1vec.^2]) a(3) a(4)]);
legend('Monte Carlo', 'Uniform Grid');
xlabel('N');ylabel('mean(z)');

figure(3);hold on;
semilogx(N1vec.^2,varzGrid,'r','LineWidth',2);
axis('auto');a=axis ; axis([min([Nvec,N1vec.^2]) max([Nvec,N1vec.^2]) a(3) a(4)]);
legend('Monte Carlo', 'Uniform Grid');
xlabel('N');ylabel('var(z)');

figure(4);clf;hold off;
surf(z);
title('Values of z Seen Over the Random Domain');

figure(5);clf;hold off;
loglog(Nvec,abs(meanzMC - meanzGrid(end)),'LineWidth',2);
hold on;
loglog(N1vec.^2,abs(meanzGrid - meanzGrid(end)),'r','LineWidth',2);
for i = 3:7,
    loglog( [1e0 1e4], [.01 10^(-i)]) ;
end;
title('Error, Relative to Highest-Fidelity Grid Result');
legend('Monte Carlo', 'Uniform Grid');
a = axis ; axis([a(1) a(2) abs(meanzGrid(end-1)-meanzGrid(end)), a(4)]);
xlabel('N');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sdot] = MCvsGridDeriv(t,s) ;
global kp kd J kt ;

phidot = s(1);
phi = s(2);

torque = kt*(-kp*(phi-1) - kd*phidot); % control action

phidotdot = torque/J ; % equation of motion

sdot(1,1) = phidotdot;
sdot(2,1) = phidot ;
```

%%