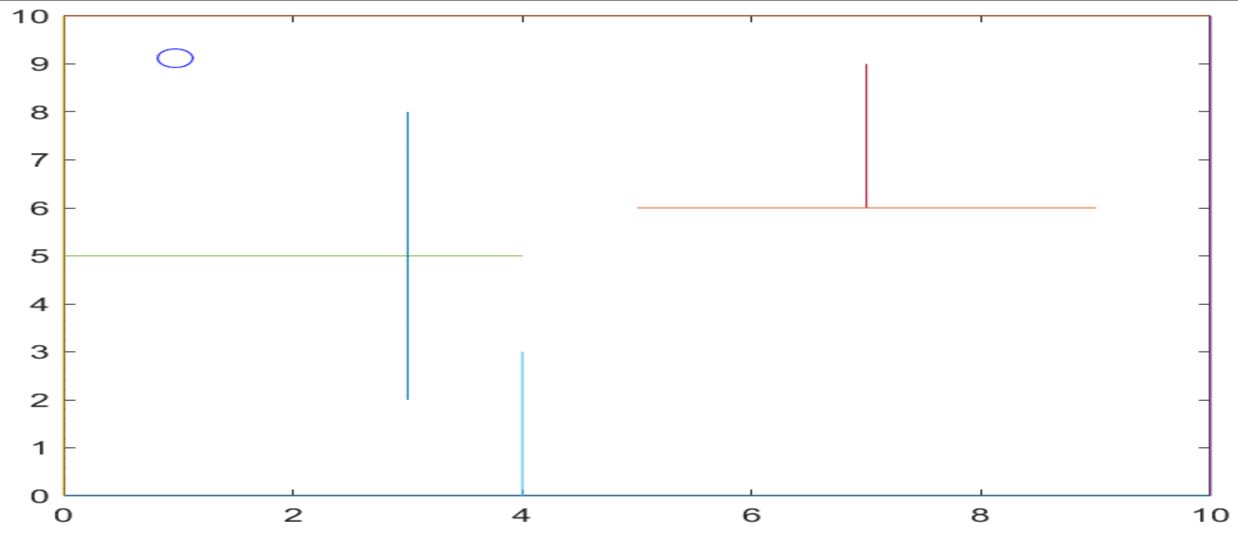
Introduction to Scientific Computing, Homework #4

(Note: Long codes will NOT be placed in a $\boxed{\text{box}}$ for better typography and copying)

Problem 1 Solution (Only key functions, full testing script in attachments)

```
%% My work
function [t, collisionState] = findCollision (ballState, wall,
coefficient_of_restitution)
x = ballState(1);
y = ballState(2);
vx = ballState(3);
vy = ballState(4);
if wall(1)==wall(3) % Wall is vertical
    if vx*(wall(1)-x)<=0 % No speed or wrong direction
        t = inf;
        collisionState = [];
        return ;
    end
    if wall(2)>wall(4) % swap if start>end
        tmp = wall(2);
        wall(2)=wall(4);
        wall(4)=tmp;
    end
    xt = wall(1); % target X
    new_y = vy*(xt-x)/vx+y; % Y of collision position.
    if new_y>=wall(2)&&new_y<=wall(4) % Fall inside the range of wall
        t = (xt-x)/vx; % How many time left before collison
        collisionState = [xt, new_y, ...
            -vx*coefficient_of_restitution, vy*coefficient_of_restitution];
        return ;
    else
        t = inf;
        collisionState = [];
        return ;
    end
else % Wall is horizontal, the same
    if vy*(wall(2)-y)<=0
        t = inf;
        collisionState = [];
        return ;
    end
    if wall(1)>wall(3)
        tmp = wall(1);
        wall(1)=wall(3);
        wall(3)=tmp;
    end
    yt = wall(2);
    new_x = vx*(yt-y)/vy+x; % x of collision position.
    if new_x>=wall(1)&&new_x<=wall(3)
        t = (yt-y)/vy;
        collisionState = [new_x, yt, ...
            vx*coefficient_of_restitution, -vy*coefficient_of_restitution];
        return ;
    else
        t = inf;
        collisionState = [];
        return ;
    end
end
end
function newBallState = updateBallState (ballState, dt, walls,
coefficient_of_restitution)
[wallcnt, ~] = size(walls);
minT_id = 1;
t = inf;
for i=1:wallcnt % Find the next wall to hit (minimum time before collision)
    wall = walls(i,:);
    [tmpt, ~] = findCollision(ballState, wall, coefficient_of_restitution);
    if tmpt<t % Log the wall with the minimum time to hit
        minT_id = i;
        t = tmpt;
    end
end
[t, collisionState] = findCollision(ballState, walls(minT_id,:),
coefficient_of_restitution);
if t<=dt % Collision is coming
    newBallState = updateBallState(collisionState, dt-t, walls,
coefficient_of_restitution);
    % Recursively update the state
else
    newBallState = UpdateBallStateNoCollisions(ballState, dt);
end
    function next_state = UpdateBallStateNoCollisions (current_state, dt)
        % Nested function, for normal movement without collisions
        next_state = current_state;
        x = current_state(1);
        y = current_state(2);
        vx = current_state(3);
        vy = current_state(4);
        next_state(1) = (x + vx*dt);
        next_state(2) = (y + vy*dt);
    end
end
```

Problem 1 Runtime Screenshot

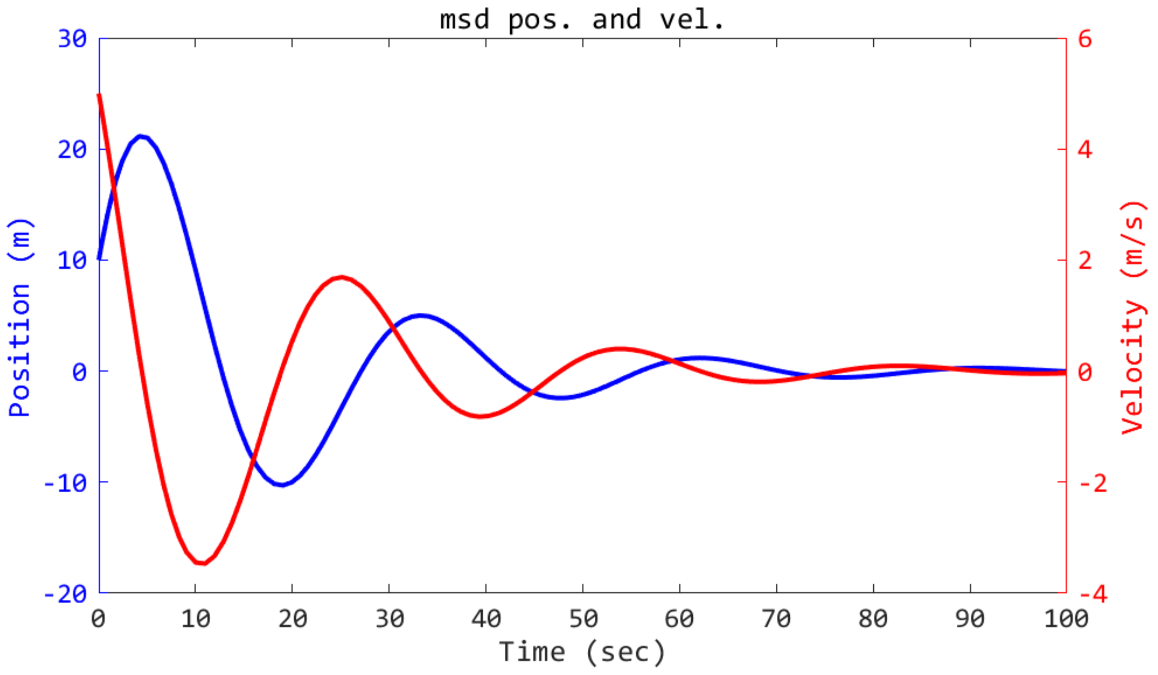


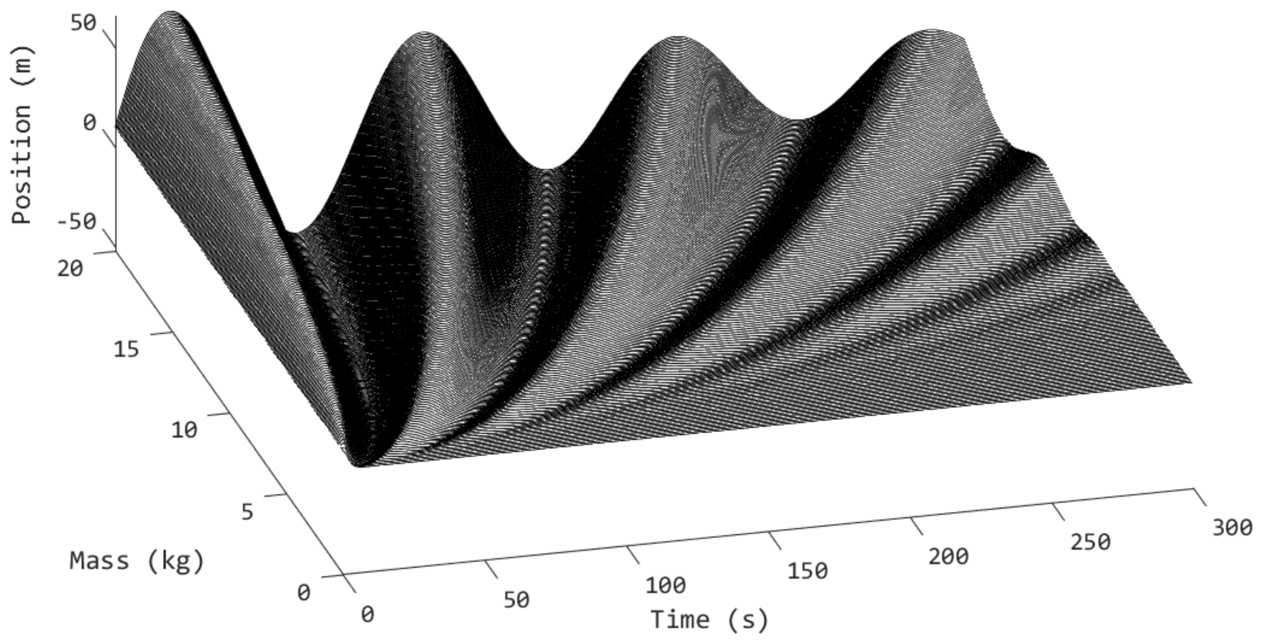
Problem 2 Solution

```
clc;clear;
msd_posVel(0.2, 0.1, 2, 10, 5, 0, 100);
msd_water(0.2, 0.1, 0:0.1:20, 10, 5, 0, 300);

function dy = msd(t, y, c, k, m)
% y1'=y2
% y2'=(-c*y2-k*y1)/m
dy=[y(2);(-c.*y(2)-k.*y(1))./m]; % Equations(given)
end

function msd_posVel(c, k, m, yi, vi, t_beg, t_end)
figure;
tspan = [t_beg, t_end];
[t, x]=ode45(@msd, tspan, [yi vi], [], c, k, m);
% Extra arguments is acceptable
% [yi vi] specifics initial conditions
% [] stands for no extra options

[Ax, C1, C2] = plotyy(t, x(:,1), t, x(:,2));
% plotyy is deprecated, but stick to it since it is required to use :)
title('msd pos. and vel.');

%% Style Settings
set(gcf, 'Color', 'w');
set(Ax, 'FontName', 'Consolas', 'FontSize', 12);
xlabel('Time (sec)');
set(get(Ax(1), 'Ylabel'), 'string', 'Position (m)');
set(get(Ax(2), 'Ylabel'), 'string', 'Velocity (m/s)');
set(Ax(1), 'ylim', [-20, 30]);set(Ax(2), 'ylim', [-4, 6]);
set(Ax(1), 'ycolor', 'b');set(Ax(2), 'ycolor', 'r'); % Match the curves
set(Ax, 'Xlim', [0, 100]);
set(C1, 'Linestyle', '-', 'color', 'b', 'Linewidth', 2);
set(C2, 'Linestyle', '-', 'color', 'r', 'Linewidth', 2);
grid off; % No grid
hold off;
end

function msd_water(c, k, m, yi, vi, t_beg, t_end)
figure;set(gcf, 'Color', 'w');
N = length(m);
x = linspace(t_beg, t_end, N);
y = m;
Z = zeros(length(x), N, 2); % Pre-allocating for speeding
for i=1:N % Compute data points for different m
    [~, Z(:,i,:)]=ode45(@msd, x, [yi vi], [], c, k, m(i));
end
[X, Y] = meshgrid(x, y);
plot3(X', Y', Z(:,:,1), 'k');
set(gca, 'FontName', 'Consolas', 'FontSize', 10);
xlabel('Time (s)');ylabel('Mass (kg)');zlabel('Position (m)');
view(-15, 55);
box off;grid off;hold off;axis tight;
end
```

(Next Page)

Problem 2 Plots



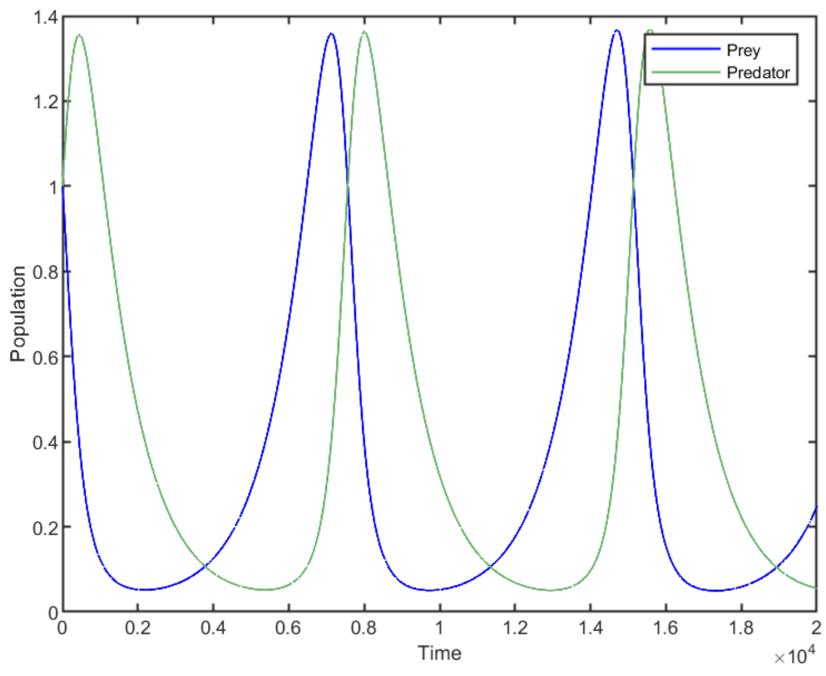


(Next Page)

Problem 3 Solution

```
clc;clear;

ntimesteps = 20000;
xt = lotkaVolterra([1, 1], 1, 2.5, 1, 2.5, 0.001, ntimesteps);
figure;
plot(1:ntimesteps, xt(1,:), '-', 'LineWidth', 1, 'color', 'blue');
% prey, blue
hold on;
plot(1:ntimesteps, xt(2,:), '-', 'LineWidth', 1, 'color', [0.4 0.7 0.4]); %
% predator, deep green.
xlabel('Time');ylabel('Population');
legend('Prey', 'Predator');
set(gca, 'FontSize', 8, 'LineWidth', 1);
axis auto;

function state = lotkaVolterra (initial_state, alpha, beta, gamma, delta, dt,
ntimesteps)
LotkaVolterra = @(t, x) [
    alpha*x(1)-beta*x(1)*x(2);
    -gamma*x(2)+delta*x(1)*x(2)]; % Derived from given equations
t = 0:dt:ntimesteps*dt-dt; % Total counts: ntimesteps
assert(ntimesteps==length(t)); % Required
[~, state] = ode45(LotkaVolterra, t, initial_state);
state = state'; % Just to meet requirements in problem

end
```

Problem 3 Plot



(Next Page)

Problem 4 Solution

```
% O(n^3) for special constructed map.
% BFS can reach O(n^2) in worst cases.
clc;clear;
%% Generate test data
map = randi(10, 11); % Random maps for test
map = map>2; % 1s:0s=8:2
map(5, 5) = 1;
disp(grassfire(map, 5, 5));
%% Grassfire function
function distance = grassfire (occupancy, dest_row, dest_col)
[rows, cols] = size(occupancy);
distance = inf(rows+2, cols+2); % border padding
distance(dest_row+1, dest_col+1) = 0;
updated = 1; %flag
while updated
    updated = 0;
    for i=2:rows+1
        for j=2:cols+1
            if ~occupancy(i-1, j-1) % Keep dis. of obstacles be inf.
                continue;
            end
            new_dis = 1 + min([distance(i-1, j), distance(i+1, j), ...
                distance(i, j-1), distance(i, j+1)]);
            %NOTE: this kind of query(crossing row/column) is very common
            %in scientific computation needs, like particle physics. But it
            %breaks "space locality" and leads to high performance loss,
            %since cache misses will occur high frequently, and
            %pre-fetching no longer make a difference.
            %The way to solve it is by dividing the full matrix into small
            %blocks in special orders.
            %
            %See also: morton code
            if new_dis<distance(i, j)
                distance(i, j) = new_dis;
                updated = 1;
            end
        end
    end
    if ~updated
        break;
    end
end
distance = distance(2:rows+1, 2:cols+1); % de-padding
end
```

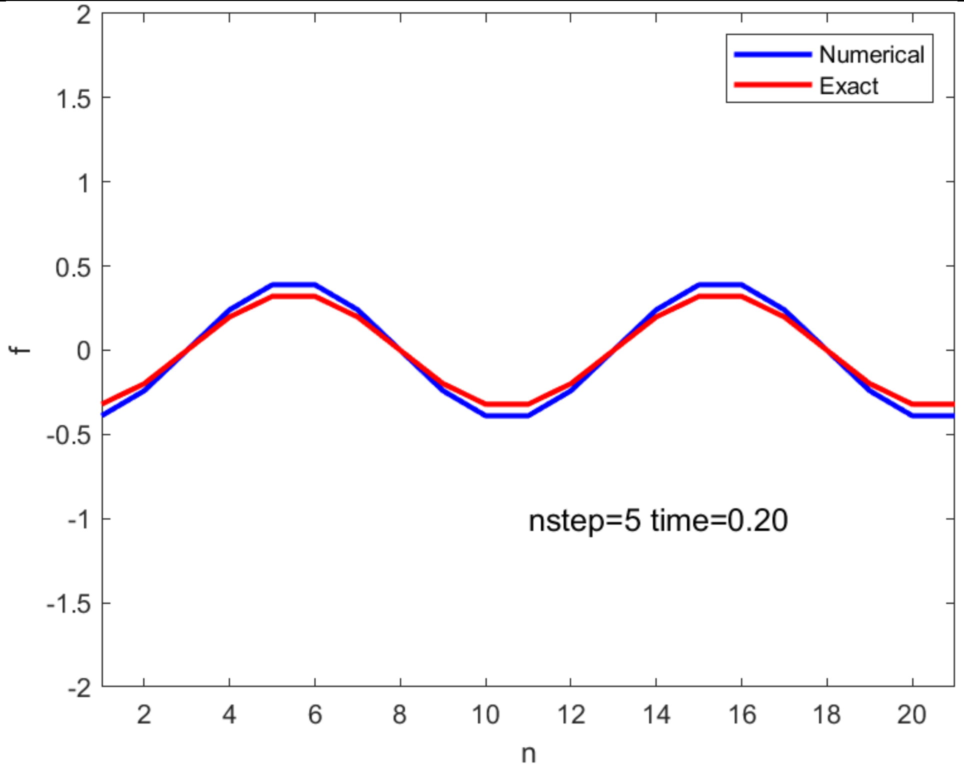
(Next Page)

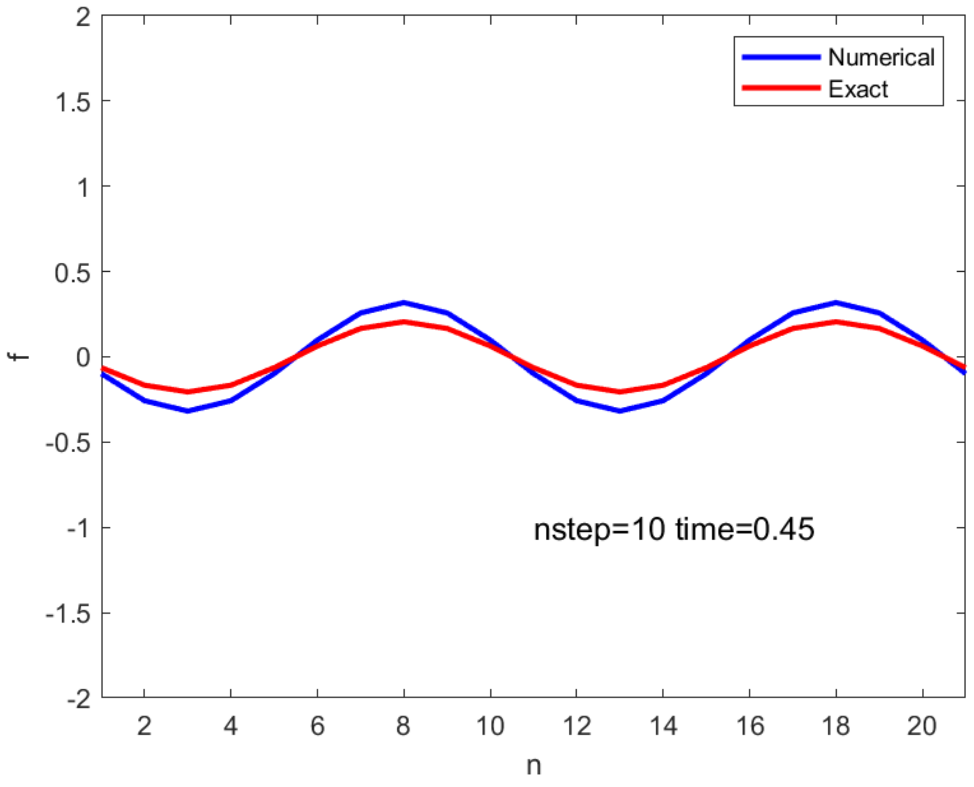
Problem 5 Solution

```
clc;clear;
close all;
%% Constants
FIGS = 10;
A = 0.5;
k = 2*pi;
D = 0.05;
U = 1;
n = 21;
h = 2/(n-1);
x = 0:h:2;
dt=0.05;
%%
y2 = A*sin(k*x); % Initial phase

for i=0:FIGS-1
    t=i*dt;
    y1 = exp(1)^(-D*k^2*t)*A*sin(k*(x-U*t));
    plot(1:n, y2, '-b', 'LineWidth', 2);
    hold on;
    plot(1:n, y1, '-r', 'LineWidth', 2);
    hold off;

    legend('Numerical', 'Exact');
    xlabel('n');ylabel('f');
    xlim([1 n]);ylim([-4*A 4*A]);
    str = sprintf("nstep=%d time=%.2f", i+1, t);
    text(11, -1, str, "Color", 'k', 'FontSize', 12); % Required

    %% Compute next numerical sequence
    newy=zeros(1, n);
    calc = @(a,b,c)(b-(c-a)*U*dt/2/h+(c-2*b+a)*D*dt/h/h); % shortcut
    for j=2:n-1
        newy(j) = calc(y2(j-1), y2(j), y2(j+1));
    end
    newy(1) = calc(y2(n-1), y2(1), y2(2)); % Special judge for boundaries
    % Regard node 1 and node n as the same node.
    % Then turn the sequence into a ring:
    % ...--(n-1)--(1|n)--2--3--...--(n-1)--(1|n)--2--3--...
    newy(n) = newy(1);
    y2=newy;
    %%

    pause(1);
end
```

(Next Page)

Problem 5 Plots(nstep=5, nstep=10)





(The End)