**Introduction to Scientific Computation, Homework #2**

**Due by 6 pm on Wednesday 5/15/2024**

**Problem 1 [20]:** If a regular fixed payment P is made n times a year to repay a loan of amount A over a period of k years, where the nominal annual interest rate is r, P is given by

You need to generate a table of repayments for a loan of $1000 over 15, 20, and 25 years, at interest rates that vary from 10% to 20% per annum in steps of 1%. (Tip: The table has 11 rows and 3 columns, with rows representing different interest rates and columns representing different years, n = 12). You need to use two different methods, the first is to use nested for loops, and the second is to vectorize the outer loop. You might use repmat(). The generated table is as follows:

10.7461 9.6502 9.0870

11.3660 10.3219 9.8011

12.0017 11.0109 10.5322

12.6524 11.7158 11.2784

13.3174 12.4352 12.0376

13.9959 13.1679 12.8083

14.6870 13.9126 13.5889

15.3900 14.6680 14.3780

16.1042 15.4331 15.1743

16.8288 16.2068 15.9768

17.5630 16.9882 16.7845

**Problem 1 Solution**

**method 1**

years=15:5:25;

rates=0.1:0.01:0.2;

result = zeros(11,3);

for i = 1:length(years)

for j = 1:length(rates)

r = rates(j);

k = years(i);

n = 12;

A = 1000;

result(j,i)=r\*A\*(1+r/n)^(n\*k)/(n\*((1+r/n)^(n\*k)-1));

end

end

disp(result);

**method 2**

years=15:5:25;

rates=0.1:0.01:0.2;

result = zeros(11,3);

r = repmat(rates',1,3);

k = repmat(years,11,1);

A = 1000;

n = 12;

result=r\*A.\*(1+r/n).^(n\*k)./(n\*((1+r/n).^(n\*k)-1));

disp(result);

**Problem 2 [20]:** Around 300 B.C. Euclid developed a wonderfully simple algorithm for determining the greatest common divisor of two positive integers. Please search online for “Euclidean Algorithm” to find out the details. Your job is to implement the following two functions:

1. function out=my\_gcd(a,b)

which computes the greatest common divisor of two positive integers *a* and *b*. You may find the MATLAB function rem() useful.

1. function out=my\_lcm(a,b)

which computes the least common multiple of two positive integers a and b. This can be computed in part using my\_gcd.

Make sure you account for the case where a and/or b is zero in your my\_gcd function. You can check this behavior by entering gcd(0,2), gcd(2,0), and gcd(0,0) into the MATLAB command line. Note that the least common multiple of a=0 or b=0 is not defined (only least common multiples of a and b larger than 0 should be returned).

You can compare your implementations to MATLAB’s built-in gcd and lcm functions to determine if everything is working correctly.

**Problem 2 Solution**

function out=my\_gcd(a,b)

pair = [a,b];

x = max(pair);

y = min(pair);

while(y>0)

r = rem(x,y);

x = y;

y = r;

end

out = x;

end

function out=my\_lcm(a,b)

if a==0 || b==0

out = 'undefined';

else

g = my\_gcd(a,b);

x = a/g;

y = b/g;

out = x\*y\*g;

end

end

**Problem 3 [20]:** Narcissistic number is a three-digit number, the sum of the third power of the digits in each bit of which is equal to itself (for example, 153 = 13 + 53 + 33). You may find more information by searching " Narcissistic number ". You need to write two functions, one for finding all the narcissistic numbers and one for determining whether a three-digit number is a narcissistic number. The second function should be nested within the first function.

**Problem 3 Solution**

function out = narcissistic(x)

out = 'False';

function arr = find()

arr=[];

for i = 100:999

a = rem(i,10);

b = (rem(i,100)-rem(i,10))/10;

c = (i-rem(i,100))/100;

if a^3+b^3+c^3==i

arr(end+1) = i;

end

end

end

for nar = find()

if nar == x

out = 'True';

end

end

end

**Problem 4 [20]:** You are to write a MATLAB function that simulates the flight of a basketball in 2D. At any point in time the state of the basketball is characterized by its 2D position, (*x*,*y*), and velocity, (*v*x, *v*y). As it flies through the air the basketball will experience forces due to gravity and air resistance.

The accelerations that the basketball experiences in the *x* and *y* directions are given by the following expressions:





where *g*, the gravitational constant, is 9.8 m/s2 and *c*, the damping coefficient, is 0.2.

Your function declaration should be as follows.

function [x,y] = basketball(g,c,x0,y0,vx0,vy0,tstep,tmax)

where tstep is the time step of the simulation and tmax is the simulation time. At the end of the simulation your program should have produced two arrays, x and y, indicating the x and y positions of the ball at every instant in time.

In this simulation, you can use a simple first order approximation to compute the next state of the ball, as shown below:



You should run the simulation with the initial position of the ball as (*x*,*y*) = (0,0), the initial velocity as (*v*x,*v*y) = (2.5,5.4), a time step of 0.01 s, and a total simulation time is 1.5 s. Note: these values should not be hard-coded into the function but should instead be called from an external script. You should output a visually appealing plot (external to the function) that shows the x and y positions of the ball and includes x and y labels.

**Problem 4 Solution**

function [x,y] = basketball(g,c,x0,y0,vx0,vy0,tstep,tmax)

t = 0;

x=[x0];

y=[y0];

vx=[vx0];

vy=[vy0];

while t <tmax

if t + tstep <= tmax

ax = -c\*vx(end)\*sqrt(vx(end)^2 + vy(end)^2);

ay = -g-c\*vy(end)\*sqrt(vx(end)^2 + vy(end)^2);

vx(end+1) = vx(end) + ax\*tstep;

vy(end+1) = vy(end) + ay\*tstep;

x(end+1) = x(end) + vx(end)\*tstep;

y(end+1) = y(end) + vy(end)\*tstep;

t = t + tstep;

else

% if t + tstep < tmax, make the time of the last point tmax.

tstep = tmax - t;

ax = -c\*vx(end)\*sqrt(vx(end)^2 + vy(end)^2);

ay = -g-c\*vy(end)\*sqrt(vx(end)^2 + vy(end)^2);

vx(end+1) = vx(end) + ax\*tstep;

vy(end+1) = vy(end) + ay\*tstep;

x(end+1) = x(end) + vx(end)\*tstep;

y(end+1) = y(end) + vy(end)\*tstep;

t = t + tstep;

end

end

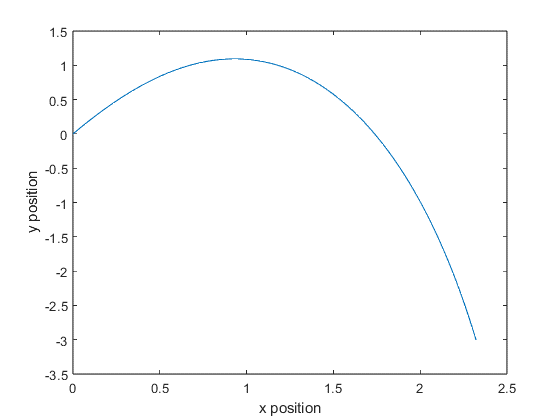
end

>> [x,y] = basketball(9.8,0.2,0,0,2.5,5.4,0.01,1.5);

>> plot(x,y);

>> xlabel('x position');

>> ylabel('y position');



**Problem 5 [20]:** You need to write a series of functions that perform operations on a numeric row vector.

i) Given a numeric row vector, replace all elements less than 0 or greater than 10 with NaN. For instance, if a numeric row vector is [5 17 -20 99 3.4 2 8 -6], the output is [5 NaN NaN NaN 3.4 2 8 NaN]. Your function declaration should be as follows.

function y = cleanup(x)

ii) Given a numeric row vector, return the largest number adjacent to ‘0’ (Tip: you might use the MATLAB function find ()). For instance, if a numeric row vector is [1 5 3 0 2 7 0 8 9 1 0], the output is 8. Your function declaration should be as follows.

function y = nearZero(x)

iii) Given a numeric row vector, find the indexes of the two nearest digits. For instance, if a numeric row vector is [2 5 3 10 0 -3.1], the index1 is 1 and the index2 is 3. Your function declaration should be as follows.

function [index1, index2] = nearestNumbers(A)

And notice, index2 should be greater than index1. You might use [a, b] = min(A)to find the value(a) and index(b) of the smallest element in an array A.

**Problem 5 Solution**

i)

function y = cleanup(x)

y=[];

for i = 1:length(x)

if x(i)<0 || x(i)>10

y(end + 1)=NaN;

else

y(end + 1)=x(i);

end

end

ii)

function y = nearZero(x)

if max(x) == 0 && min(x) == 0 % If all the elements are zeros, return zero.

y = 0;

else

nonZeroIndices = find(x);

y = x(nonZeroIndices(1));

if nonZeroIndices(1) ~=1 % when the first element is zero, consider the second one.

y = max(y,x(2));

end

for i = 1:length(nonZeroIndices)-1

if nonZeroIndices(i+1) - nonZeroIndices(i) >1

% when x(i+1) - x(i)>1, x(i+1) and x(i) are elements near zero.

if nonZeroIndices(i+1) - nonZeroIndices(i) >2

% it means there is zero near another zero

y = max([y,x(nonZeroIndices(i)),x(nonZeroIndices(i+1)),0]);

% maybe all the non-zero elements are less than zero.

else

y = max([y,x(nonZeroIndices(i)),x(nonZeroIndices(i+1))]);

end

end

end

if nonZeroIndices(end) ~= length(x) % when the last element is zero, consider the last second one.

y = max([y,x(end-1)]);

end

end

end

iii)

function [index1, index2] = nearestNumbers(A)

B = A';

P = repmat(A,length(A),1);

Q = repmat(B,1,length(A));

Diff = abs(P-Q);

min\_diff = Diff(1,2);

pair = [1,2];

for i = 1:length(A)

for j = 1:length(A)

if i>j

if Diff(i,j)<min\_diff

min\_diff=Diff(i,j);

pair = [i,j];

end

end

end

end

index1=pair(2);

index2=pair(1);

end