**Introduction to Scientific Computing, Homework #1**

**Problem 1 Solution**

clear;clc;

d = floor(6\*rand(1,20)+1);

% generate a vector d filled with random integers in range of [1,6]

disp('The vector d is:');

disp(d); % display the content of vector d

a = (d == 6); % Convert d to a logical vector

% ("sixes" replaced by 1, otherwise 0)

fprintf('The ''six(es)'' count of d: %d\n', sum(a));

% Using function sum() to sum the whole logical vector

% Because every "1" in logical vector a represents a "6" in d,

% by summing vector a we can yield the count of 6 in vector d.

**Example Output:**

>>The vector d is:

5 4 4 2 4 1 1 1 6 6 1 3 4 2 5 4 3 2 5 6

The 'six(es)' count of d: 3

(Next Page)

(Next Page)

**Problem 2 Solution**

clear;clc;

x\_in\_rad = 0:pi/6:2\*pi; % First column from 0 to 2π in steps of π/6

x\_in\_deg = rad2deg(x\_in\_rad); % Second column convert rad. to deg.

sinx = sin(x\_in\_rad); % Calculate sin(x), which constitutes column 3

cosx = cos(x\_in\_rad); % Calculate cos(x), which constitutes column 4

A = table(x\_in\_rad', x\_in\_deg', sinx', cosx',...

'VariableNames', {'x(rad)', 'x(deg)', 'sin(x)', 'cos(x)'});

% Assemble the final array A and display it.

% Named argument for naming the variables.

% Note that here the ' notation is used for

% transpose the vectors, to make them filled column-by-column,

% instead of row-by-row.

disp('The whole array looks like:');

disp(A);

% Extra work to visualize is followed here.

plot(x\_in\_rad, sinx, 'g');

hold on; % !Essential for add more curves in one plot figure

plot(x\_in\_rad, cosx, 'b');

legend('sin(x)','cos(x)')

title('A simple figure')

**Output:**

>> The whole array looks like:

x(rad) x(deg) sin(x) cos(x)

\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

0 0 0 1

0.5236 30 0.5 0.86603

1.0472 60 0.86603 0.5

***(more lines omitted)***

**Figure for sin(x) & cos(x):**



(Next Page)

**Problem 3 Solution**

clear;clc;

A = [

-23,-18,2,-19,-15;

-1,-16,10,1,3;

-15,-23,1,8,7;

-19,7,2,19,-4;

-15,-11,-3,-11,-15];% Matrix (5 x 5) of coefficients

B = [22;-21;-20;-18;-17];% Vector (5 x 1) of constants

xSol1 = A\B;% Method1: using operator \

disp('Method1 x1~x5: ');disp(xSol1');

xSol2 = inv(A)\*B;

disp('Method2 x1~x5: ');disp(xSol2');

% Method2: multiplied by the inverse matrix of A

% Note: Less efficient & less accurate than method1

xSol3 = linsolve(A, B);

disp('Method3 x1~x5: ');disp(xSol3');

% Method3: using built-in MATLAB function linsolve()

% which is designated for solving linear equations.

**Output:**

>>Method1 x1~x5:

-2.7527 2.7366 1.0375 -3.8683 4.5085

Method2 x1~x5:

-2.7527 2.7366 1.0375 -3.8683 4.5085

Method3 x1~x5:

-2.7527 2.7366 1.0375 -3.8683 4.5085

(Next Page)

**Problem 4 Solution**

clear;clc;

gcd(44,28);

gcd(14,24);

function gcd(M, N)

% Calculate GCD(M,N)

% If M/N is not designated then got it from user input

switch nargin

case 0

M = round(input('Input value of M:'));

N = round(input('Input value of N:'));

end

while M ~= N % 更相减损法/Euclidean algorithm to calculate GCD

while M > N

M = M - N;

end

% fprintf("M=%d N=%d\n", M, N);

**% Deprecated due to typographical problems**

**% (avoid too many lines in output)**

while M < N

N = N - M;

end

fprintf("M=%d N=%d\n", M, N); % Sketch M and N in loop

end

disp(['Ultimately, value of M is:']);disp(M);

end

**Output:**

>>M=16 N=12

M=4 N=4

Ultimately, value of M is:

4

M=14 N=10

M=4 N=2

M=2 N=2

Ultimately, value of M is:

2

(Next Page)

We can dispatch different option arguments(by editing the “plot” code line) for different visual effects. Here is what I used for 3 figures above:

‘.b’ for figure1: Blue and dot line;

‘-r’ for figure2: Red and straight line;

‘-c’ for figure3: Cyan and straight line.

There’re also “named arguments” for it, for example I modified ‘LineWidth’ to make a **bold line.** The reason not to modify the marker type (diamond, star…) is that there are too many data points, which will lead to overlapped markers.

**Problem 5 Solution**

clear;clc;

figure(1);% Seperated window for different curves

spirograph(5,1,0.4);

figure(2);

spirograph(12,-1,1.5);

figure(3);

spirograph(7,-1,1);

function spirograph(R,r,d)

theta = 0:0.0005:10\*pi; % Generate theta vector in steps=0.0005 rad

x = cos(theta)\*(R+r)+cos(theta\*(R+r)/r)\*d; % Generate x vector without loop

y = sin(theta)\*(R+r)-sin(theta\*(R+r)/r)\*d; % The same to y vector

plot(x,y,'.b',...

'LineWidth',2); % Try to change the style/color/width!

grid on; % Display grid on background

title('Beautiful Curve for You');

legend('roulette')

end

**Plots in different plot() style args:**



(End)

**Problem 6 Solution**

clear;clc;

% For test only

output\_str=join(string(fib(20)),', '); % More pretty output

fprintf("The Fibonacci sequence for N = 20:\n%s\n", output\_str);

function nums = fib(N)

% fib(N) returns a list of the first N Fibonacci Numbers.

% N must be an integer.

if N <= 0

error('N must be positive.'); % Error reporting for illegal N;

end

if N <= 1 % Special judge

nums = 0;

return

end

if N == 2

nums = [0, 1];

return

end

nums = [0, 1]; % Initial value for the sequence

for now\_index = 3:N+1

% In MATLAB array index starts from 1, but the sequence index starts from 0

% So here is the reason why we offset 1

nums(now\_index) = nums(now\_index-1) + nums(now\_index-2);

end

end

% Of course we can use the built-in function fib()

% But that's too boring!

% Recursive function? Yes, but the time complexity will be Θ(n^2)

% When it comes to time complexity, the method that applies binary exponentiation

% to matrix can yield F\_n in Θ(logn).

% However it’s not so effective when generating a whole sequence

% Just keep it simple.

**Output that required:**

>> The Fibonacci sequence for N = 20:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765