Topics: Vectorized code, simulation
Reading: CFile Chapter 9 Sec 9.3

Local minimum in a "neighborhood"

```
function minVal = minInNeighborhood(M, loc)
    % minVal is the lowest value in the neighborhood of position
    % (loc(1),loc(2)), i.e., M(loc(1)-1:loc(1)+1, loc(2)-1:loc(2)+1).
    % M is a matrix of numbers.  loc is a vector of length 2.

    [nr, nc]= size(M);  % dimensions of M

    %Build a border around M
    bigM= [ ones(nr,1)*realmax  M  ones(nr,1)*realmax ];
    bigM= [ ones(1,nc+2)*realmax;  bigM;  ones(1,nc+2)*realmax ];

    nr= nr+2;  nc= nc+2;        % dimensions of bigM
    r= loc(1)+1;  c= loc(2)+1;  % center of neighborhood in bigM

    minVal= minInMatrix( bigM(r-1:r+1, c-1:c+1) );
```

Vectorized Code in 1-D

MATLAB can operate (e.g., perform arithmetic, relational, or logical operations) on entire vectors in one step (in one statement). Code that operate on vectors, instead of on scalars, in one statement is said to be *vectorized*.

```
x= [10 20 30]
y= [2  1  2]
z= [2; 3; 2]  % column

% Vectorized addition, subtraction
  x+y           % [12 21 32]
  x-y           % [8 19 28]
  x+5           % [15 25 35]

% Vectorized multiplication, division, power
% Need DOT OPERATOR (.)
  x.*y          % [20 20 60]
  x./y          % [5 10 15]
  x.^y          % [100 20 900]
  x.*5          % [50 100 150]

% Shape is important!
  x+z           % ERROR!  x is a row while z is column
  x+z'          % [12 23 32]
```

Note: MATLAB is designed to support a field of mathematics called *linear algebra*. After you learn linear algebra (not in CS100!), you can really harness the power of MATLAB's matrix computation capabilities. In CS100, we will *not* use "matrix multiplication" as defined in linear algebra and coded in MATLAB as `m*n` where `m` and `n` are vectors (or matrices). Rather, we multiply the vectors "cell-by-cell" using the MATLAB code `m.*n`, which means to perform the operation `m(i)*n(i)` for all index i for vectors `m` and `n` (assuming that they have the same shape and length). The result of "dot multiply" is a vector the same shape and length as `m` and `n`, as illustrated in the example above.

Example: Pair-sums

Given a vector **x**, calculate the pair sums and store them in a vector. For example, the vector [2 1 1 8] has pair sums [3 2 9].

Example: How many 'M's

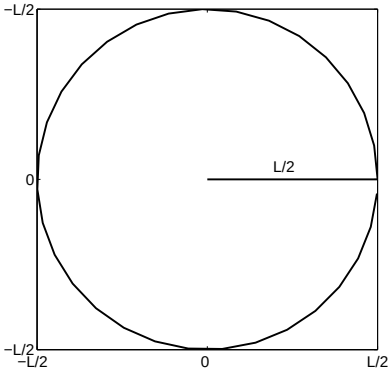
Write a code fragment to determine how many times the character '**M**' appears in a string **s**.

Simulation of systems

Simulation is the application of mathematical and computer models that imitate the behavior of a system. Simulation is a useful tool for design, training, and games!

Estimate Pi

The value π can be approximated in many ways. One method is to simulate dart throwing! Let N be the number of darts thrown randomly over a square domain of area $L \times L$. The largest circle that can fit inside this domain has a diameter of L and an area of $\pi L^2/4$.



Let the number of darts N be the area of the square domain:

$$N = L \times L. \tag{1}$$

Then the number of darts that fall inside the circle, N_{in}, is the area of the circle:

$$N_{in} = \frac{\pi L^2}{4}. \tag{2}$$

Substitute equation (1) into (2) to get π:

$$\pi = \frac{4N_{in}}{N} \tag{3}$$

The following function performs a simulation of dart throwing. The function parameter is the number of darts to be thrown.

```
function myPi = approxPi(nDarts)
% Approximate Pi using Monte Carlo simulations
% Usage:  myPi = approxPi(nDarts)
%   NDARTS = number of "darts" thrown
%   myPi = Monte Carlo approximation of Pi

L = 10;  % length of square

% Throw darts in L-by-L area, centered at 0,0
  throws =

  x = throws(:,1);  % x-coordinates of darts
  y = throws(:,2);  % y-coordinates of darts

% Location of darts relative to center
  dist =                                % distance from center

  nIn =                                 % #darts inside circle


myPi = 4*nIn/nDarts;

% Plot darts in domain
% YOU ARE NOT RESPONSIBLE FOR LEARNING AXIS FORMATS
  % Circle data
    theta = 0:0.2:2*pi;
    xcircle = cos(theta)*L/2;
    ycircle = sin(theta)*L/2;
plot(x,y,'*',xcircle,ycircle,'r','linewidth',2)
axis([-L/2 L/2 -L/2 L/2]);  axis('square');
title(['Pi = ' num2str(myPi)]);
```

Estimate Pi

```
function myPi = approxPi(nDarts)
% Approximate Pi using Monte Carlo simulations
% Usage:  myPi = approxPi(nDarts)
%   NDARTS = number of "darts" thrown
%   myPi = Monte Carlo approximation of Pi

L = 10;  % length of square

% Throw darts in L-by-L area, centered at 0,0
throws = L*rand(nDarts,2) - L/2;
x = throws(:,1);  % x-coordinates of darts
y = throws(:,2);  % y-coordinates of darts

% Location of darts relative to center
  dist = sqrt(x.^2+y.^2);  % distance from center
  nIn = sum(dist <= L/2);  % #darts inside circle

myPi = 4*nIn/nDarts;

% Plot darts in domain
% YOU ARE NOT RESPONSIBLE FOR LEARNING AXIS FORMATS
  % Circle data
    theta = 0:0.2:2*pi;
    xcircle = cos(theta)*L/2;
    ycircle = sin(theta)*L/2;
plot(xcircle,ycircle,'r',x,y,'*','linewidth',2)
axis([-L/2 L/2 -L/2 L/2]);  axis('square');
title(['Pi = ' num2str(myPi)]);
```