Previous Lecture:
- Nested loops
- Developing algorithms and code

Today’s Lecture:
- Review nested loops
- User-defined functions

Announcements:
- Project 2 due Monday at 11pm
- Upson Hall elevator out-of-service starting next week.
  Access for the physically disabled can be pre-arranged by emailing TA Pooja <pn226>

Rational approximation of $\pi$
- $\pi \approx 3.141592653589793$...
- Can be closely approximated by fractions, e.g., $\pi \approx 22/7$
- Rational number: a quotient of two integers
- Approximate $\pi$ as $p/q$ where $p$ and $q$ are positive integers $\leq M$
- Start with a straight forward solution:
  - Get $M$ from user
  - Calculate quotient $p/q$ for all combinations of $p$ and $q$
  - Pick best quotient $\rightarrow$ smallest error

```matlab
% Rational approximation of pi
M = input('Enter M: ');

% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);

% Check all possible denominators
for q = 1:M

% At this q, check all possible numerators
for p = 1:M
  if abs(p/q - pi) < err_pq  % best p/q found
    err_pq = abs(p/q - pi);
    pBest= p;
    qBest= q;
  end
end

myPi = pBest/qBest;
end
```

Analyze the program for efficiency
- See Eg3_1 and FasterEg3_1 in the book

```matlab
% Rational approximation of pi
M = input('Enter M: ');

% Best q, p, and error so far
qBest=1;  pBest=1;
err_pq = abs(pBest/qBest - pi);

% Check all possible denominators
for q = 1:M

% At this q, check all possible numerators
for p = 1:M
  if abs(p/q - pi) < err_pq  % best p/q found
    err_pq = abs(p/q - pi);
    pBest= p;
    qBest= q;
  end
end

myPi = pBest/qBest;
end
```

How many times are “alpha” and “beta” displayed?

A: $n, m$
B: $m, n$
C: $n, n+m$
D: $n, n*m$
E: $m^n, m$
**Built-in functions**
- We’ve used many Matlab built-in functions, e.g., `rand`, `abs`, `floor`, `rem`.
- Example: `abs(x-.5)`

**Observations:**
- `abs` is set up to be able to work with any valid data.
- `abs` doesn’t prompt us for input; it expects that we provide data that it’ll then work on.
- `abs` returns a value that we can use in our program.

```matlab
yDistance= abs(y2-y1);
while abs(myPi-pi) > .0001
  ...
```

**User-defined functions**
- We can write our own functions to perform a specific task.
- Example: draw a disk with specified radius, color, and center coordinates.
- Example: generate a random floating point number in a specified interval.
- Example: convert polar coordinates to x-y (Cartesian) coordinates.

**Draw a bulls eye figure with randomly placed dots**
- Dots are randomly placed within concentric rings.
- User decides how many rings, how many dots.

**Convert from polar to Cartesian coordinates**

```matlab
c= input('How many concentric rings? ');
d= input('How many dots? ');
% Put dots btwn circles with radii rRing and (rRing-1)
for rRing= 1:c
  % Draw d dots
  for count= 1:d
    % Generate random dot location (polar coord.)
    theta= _______
    r= _______
    % Convert from polar to Cartesian
    x= _______
    y= _______
    % Use plot to draw dot
    end
  end
end
```

A common task! Create a function `polar2xy` to do this. `polar2xy` likely will be useful in other problems as well.
c = input('How many concentric rings? ');
if = input('How many dots? ');

% Put dots between circles with radii rRing and (rRing-1)
for rRing = 1:c
    % Draw d dots
    for count = 1:d
        % Generate random dot location (polar coord.)
        theta = _______
        r = _______
        % Convert from polar to Cartesian
        x = _______
        y = _______
        % Use plot to draw dot
    end
end

[x,y] = polar2xy(r,theta);

% Generate random dot location (polar)
theta = _______

% Convert from polar to Cartesian
rads = theta*pi/180;  % radian
x = r*cos(rads);
y = r*sin(rads);

% Convert polar (r1,t1) to Cartesian (x1,y1)
r1 = 1;   t1= 30;
[x1, y1]= polar2xy(r1, t1);
plot(x1, y1, 'b*')

function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r, theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.
rads = theta*pi/180;  % radian
x = r*cos(rads);
y = r*sin(rads);

r = input('Enter radius: ');
theta = input('Enter angle in degrees: ');
rads = theta*pi/180;  % radian
x = r*cos(rads);
y = r*sin(rads);

You have this function:

function [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r, theta) to
% Cartesian coordinates (x, y).  Theta in degrees.
...

Code to call the above function:

% Convert polar (r1,t1) to Cartesian (x1,y1)
r1 = 1;   t1= 30;
[x1, y1]= polar2xy(r1, t1);
plot(x1, y1, 'b*')
dotsInRings.m

(functions with multiple input parameters)
(functions with a single output parameter)
(functions with multiple output parameters)
(functions with no output parameter)

General form of a user-defined function

\[
\text{function } [\text{out1, out2, ...}]=\text{functionName}(\text{in1, in2, ...})
\]

% I-line comment to describe the function
% Additional description of function

Executable code that at some point assigns values to output parameters \text{out1, out2, ...}

- \text{in1, in2, ...} are defined when the function begins execution. Variables \text{in1, in2, ...} are called function \text{parameters} and they hold the function \text{arguments} used when the function is invoked (called).
- \text{out1, out2, ...} are not defined until the executable code in the function assigns values to them.

Returning a value ≠ printing a value

You have this function:

\[
\text{function } [x, y] = \text{polar2xy}(r, \theta)
\]
% Convert polar coordinates \((r,\theta)\) to \% Cartesian coordinates \((x,y)\). Theta in degrees.
...

Code to call the above function:
%% Convert polar \((r1,t1)\) to Cartesian \((x1,y1)\)
\r1= 1; \ t1= 30;
\[x1,y1]= \text{polar2xy}(r1,t1);
\text{plot}(x1,y1,'b*')
...%

Comments in functions

- Block of comments after the function header is printed whenever a user types \text{help <functionName>} at the Command Window
- 1st line of this comment block is searched whenever a user types \text{lookfor <someWord>} at the Command Window
- Every function should have a comment block after the function header that says \text{what the function does concisely}

Accessing your functions

For now*, put your related functions and scripts in the same directory.

\begin{verbatim}
MyDirectory

dotsInCircles.m  polar2xy.m
randDouble.m  drawColorDot.m
\end{verbatim}

*The path function gives greater flexibility