Previous Lecture:
- Nested loops
- Developing algorithms and code

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

Announcements:
- Project 2 due tonight at 11pm

Rational approximation of $\pi$

- $\pi \approx 3.141592653589793…$
- Can be closely approximated by fractions, e.g., $\pi \approx \frac{22}{7}$
- 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

% Check all possible numerators
for p = 1:M

% At this q, check all possible numerators
for p = 1:M

end

end

myPi = pBest/qBest;
```

Analyze the program for efficiency

- See Eg3_1 and FasterEg3_1 in the book

```matlab
for a = 1:n
    disp('alpha')
    for b = 1:m
        disp('beta')
    end
end
```

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

A: n, m
B: m, n
C: n, n+m
D: n, n^2
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
% 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

end

end

myPi = pBest/qBest;
```
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

- What are the main tasks?
- Accommodate variable number of rings—loop
  - For each ring
    - Need many dots
    - For each dot
      - Generate random position
      - Choose color
      - Draw it

Convert from polar to Cartesian coordinates

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

% Convert from polar to Cartesian
x = r*cos(rads);
y = r*sin(rads);

A common task! Create a function polar2xy to do this. polar2xy likely will be useful in other problems as well.
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);

% 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
Returning a value ≠ printing a value

You have this function:

```matlab
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:

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

Comments in functions

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

Given this function:

```matlab
function m = convertLength(ft,in)
% Convert length from feet (ft) and inches (in)
% to meters (m).
...
```

How many proper calls to `convertLength` are shown below?

```matlab
% Given f and n
d= convertLength(f,n);
d= convertLength(f*12+n);
d= convertLength(f+n/12);
x= min(convertLength(f,n), 1);
y= convertLength(pi*(f+n/12)^2);
```

A: 1    B: 2    C: 3    D: 4    E: 5 or 0

Accessing your functions

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

```
MyDirectory
```

*The `path` function gives greater flexibility

Why write user-defined function?

- Easy code re-use—great for "common" tasks
- A function can be tested independently easily
- Keep a driver program clean by keeping detail code in *functions*—separate, non-interacting files
- Facilitate top-down design
- Software management

```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
```
Facilitates top-down design

1. Focus on how to draw the figure given just a specification of what the function \texttt{DrawStar} does.
2. Figure out how to implement \texttt{DrawStar}.

To specify a function...
... you describe how to use it, e.g.,

\begin{verbatim}
function DrawStar(xc,yc,r,c)
    \% Adds a 5-pointed star to the figure window. Star has radius \(r\), \% center(xc,yc) and color \(c\) where \(c\) \% is one of 'r', 'g', 'y', etc.
end
\end{verbatim}

Given the specification, the user of the function doesn't need to know the detail of the function—they can just use it!

To implement a function...
... you write the code so that the function "lives up to" the specification. E.g.,

\begin{verbatim}
r2 = r/(2*(1+sin(pi/10)));
tau = pi/5;
for k=1:11
    theta = (2*k-1)*pi/10;
    if 2*floor(k/2)~=k
        x(k) = xc + r*cos(theta);
y(k) = yc + r*sin(theta);
    else
        x(k) = xc + r2*cos(theta);
y(k) = yc + r2*sin(theta);
    end
end
fill(x,y,c)
\end{verbatim}

Software Management

Today:
I write a function \texttt{EPerimeter(a,b)}
that computes the perimeter of the ellipse
\[
\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1
\]

Software Management

During this year:
You write software that makes extensive use of \texttt{EPerimeter(a,b)}
Imagine hundreds of programs each with several lines that reference \texttt{EPerimeter}

Next year:
I discover a more efficient way to approximate ellipse perimeters. I change the implementation of \texttt{EPerimeter(a,b)}
You do not have to change your software at all.