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

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

Announcements:
- Project 2 due Thursday at 11pm
- Final exam will be on Dec 7 at 2pm ONLY for both Lecl and Lecl2. The second exam date posted on the University exam calendar is wrong.

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

% Check all possible denominators for q = 1:M
For current q find best numerator p-
Check all possible numerators

end

% 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;

% Complicated version in the book
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  % new best numerator found
    err_pq = abs(p/q - pi);
    pBest= p;
    qBest= q;
  end
end
myPi = pBest/qBest;
% Rational approximation of pi
M = input('Enter M: ');
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
end
myPi = pBest/qBest;

Algorithm: Finding the best in a set

Init bestSoFar
Loop over set
  if current is better than bestSoFar
    bestSoFar = current
  end
end

Analyze the program for efficiency

■ See Eg3_1 and FasterEg3_1 in the book

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*m
E:  m*n,m

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

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

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

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**

Polar coordinates

\[ r \quad \theta \]

Cartesian coordinates

\[ x \quad y \]

---

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  

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

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

---

The function `polar2xy` takes two arguments, `r` and `theta`, and returns two values, `x` and `y`, representing the Cartesian coordinates corresponding to the polar coordinates `(r, theta)`. The function calculates the `x` and `y` values using the conversion formulas:

\[
x = r \cos(\theta) \\
y = r \sin(\theta)
\]

where `r` is the radius and `\theta` is the angle in radians. The function is defined as follows:

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

---

Think of `polar2xy` as a factory that takes in polar coordinates and outputs their Cartesian equivalents. This function can be very useful in various applications where polar to Cartesian coordinate conversion is needed.
Function header is the “contract” for how the function will be used (called).

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*')
...
```

You have this function:

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*')
...
```

General form of a user-defined function

```matlab
function [out1, out2, ...]= functionName(in1, in2, ...)
% 1-line comment to describe the function
% Additional description of function

Executable code that at some point assigns values to output parameters out1, out2, ...
```

- `in1`, `in2`, ... are defined when the function begins execution. Variables `in1`, `in2`, ... are called function parameters and they hold the function arguments used when the function is invoked (called).
- `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:

```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*')
...
```

Accessing your functions

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

```matlab
dotsInCircles.m    polar2xy.m
randDouble.m       drawColorDot.m
```

Any script/function that calls `polar2xy.m`

*The `path` function gives greater flexibility.