Iteration

Lecture 5 (Feb 6)
CS100M – Spring 2007

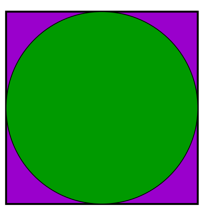
Announcements

- Project 2
  - Due Thursday, Feb 15, 6pm
  - Online since Friday
- For this week, section will be in the classroom instead of the lab

---

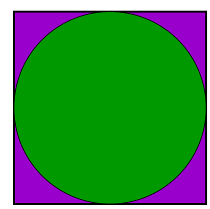
Goal

- Create a Matlab program to “throw darts” at a simple target
  - We use random numbers to determine where each dart lands
  - We can use this as a way to approximate π
- Strategy
  - First, we figure out a program to “throw” one dart
  - Then we modify it to throw many darts (say, 1000)



Algorithm Outline for One “Throw”

- Compute position of dart
- If within unit circle
  - Draw a “hit”
- Otherwise
  - Draw a “miss”



Final Code for One Throw

initTarget:

```
close all
axis('square');
axis([-1 1 -1 1]);
hold on

theta = 2*pi*(0:.01:1);
plot(cos(theta), sin(theta), '-r');
```

[This code draws a circle; we'll discuss exactly what it's doing later in the course]

oneThrow:

```
px = 2*rand(1) - 1;
py = 2*rand(1) - 1;
if (px^2 + py^2 <= 1)
    plot(px, py, 'og');
else
    plot(px, py, 'or');
end
```

Using a For-Loop

```
count = 1000;
for n = 1:1:count
    px = 2*rand(1) - 1;
    py = 2*rand(1) - 1;
    if (px^2 + py^2 <= 1)
        plot(px, py, 'og');
    else
        plot(px, py, 'or');
    end
end
```

For-loop syntax:

```
for <index variable> = <lower bound> : <increment> : <upper bound>
        Statements to execute (also called loop body)
end
```

More For-Loop Examples

- You can use negative increments
  - for i = 10:-1:5   % i takes on the values 10, 9, 8, 7, 6, 5
  - for i = 0:-2:-5   % i takes on the values 0, -2, -4
- You can use non-integers
  - for x = 0: 0.5: 2   % x takes on the values 0, 0.5, 1.0, 1.5, 2
  - for x = 0: pi/3: pi   % x takes on the values 0, pi/3, 2pi/3, pi
- Note that the upper bound is checked *every time*, even the *first time* through the loop
  - for x = 5:1:0   % The loop body will *not* be executed
  - for x = 10:1   % The loop body will *not* be executed
  - for x = 1:-1:10   % The loop body will *not* be executed

Another Kind of Loop

- We don't always know exactly which values we'll need
  - Example: The sum 1 + 1/2 + 1/3 + 1/4 + … can be made arbitrarily large by using enough terms
  - How many terms do we need to reach a given bound?
- Algorithm outline
  - Determine bound
  - Initialize sum
  - Loop as long as sum < bound:
    - sum = sum + next term
  - Report number of terms used
- Matlab (and most other languages) provide a while-loop for this kind of situation

Resulting Code

```
bound = input('Specify bound: ');
sum = 0;
n = 0;
while sum < bound
    n = n + 1;
    sum = sum + 1/n;
end
fprintf('Bound %d was exceeded at term %d\n', bound, n);
```

while-loop syntax:

while <boolean condition>
    Statements to execute (also called *loop body*)
end

Floating Point Numbers

- Matlab notation for 6.02 x 10^{23} is
  - 6.02e23 or
  - 6.02E23 or
  - 6.02e+23 or
  - 6.02E+23
- The 6.02 part is called the *mantissa*
- The 23 part is the *exponent*

Finite Precision

- Finite precision implies
  - There are just finitely many numbers that can be represented
  - There is a largest possible floating-point number
    - In Matlab, this is called realmax (typically, realmax = 1.7977e+308)
  - There is a smallest possible *positive* floating-point number
    - In Matlab, this is called realmin (typically, realmin = 2.2251e-308)
  - There is a largest possible integer
    - In Matlab, this is called intmax (typically, intmax = 2147483647)

Summary

- Matlab loops
  - For-loop
  - While-loop
- For-loop increment-control
  - <start value> : <increment> : <upper bound>
  - <start value> : <upper bound>
- Numbers in Matlab
  - Finite precision
  - Only finitely many numbers are represented