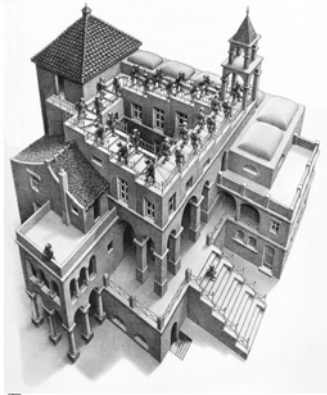


Announcements

- Project 2
 - Due Thursday, Feb 16
 - Online since Friday
- For this week, section will be in the classroom instead of the lab

Iteration



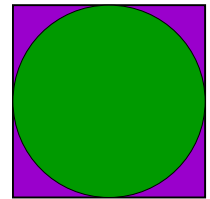
Lecture 5 (Feb 7)
CS100M - Spring 2006

Topics

- Reading: CFile Chapter 2
 - Be sure you understand Section 2.2 on floating point numbers
- Recall
 - If-else-end construct
 - Logical operators & Boolean expressions
- Plans for today
 - Constructs for iteration
 - For loop
 - While loop

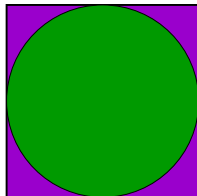
Goal

- 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"



Code Development for One Throw

- Compute position of dart

```
px = 2*rand - 1;
py = 2*rand - 1;
```
- If within unit circle

```
if (px^2 + py^2 <= 1)
```

 - Draw a "hit"

```
plot(px, py, 'og');
```
- Otherwise

```
else
```

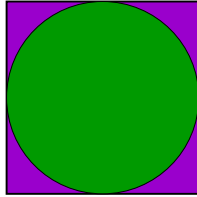
 - Draw a "miss"

```
plot(px, py, 'or');
```

Final Code for One Throw

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

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



Throwing Many Darts

- One (not very efficient) method:
- We can get an approximation of pi if we do many trials
- How do we create lots of trials?

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

Using a For-Loop

```
count = 2000;
for n = 1:1:count
    px = 2*rand - 1;
    py = 2*rand - 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
```

For-Loop Examples

```
for i = 1:1:4
    disp(i)
end
for i = 5:2:11
    disp(i)
end
for i = 10:2:17
    disp(i)
end
for i = 3:6
    disp(i)
end
```

- i takes on the values 1, 2, 3, 4
- i takes on the values 5, 7, 9, 11
- i takes on the values 10, 12, 14, 16
- i takes on the values 3, 4, 5, 6 (because if no increment is specified then 1 is assumed)

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 Algorithm Example

- Goal: Determine the sum of 10 numbers

```
sum = 0;
for i = 1:10
    n = input('Enter a number');
    sum = sum + n;
end
fprintf('Sum is %f\n', sum);
```

- Algorithm
 - Initialize sum
 - Loop 10 times
 - Get number and add to sum
 - Report sum

Another Kind of Loop

- We don't always know exactly which values we'll need
 - Example: The sum $1 + 1/2 + 1/3 + 1/4 + \dots$ 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 $\text{sum} < \text{bound}$:
 - $\text{sum} = \text{sum} + \text{next term}$
 - Report number of terms used
- Matlab (and most other languages) provide a while-loop for this kind of problem

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
```

Problems

- It takes forever to get to any value much greater than 20
- When n gets large enough, the sum quits changing
 - This happens because numbers in the computer have finite precision
 - In other words, each number is represented
 - Using a certain fixed number of digits (typically 53 binary digits) for the *mantissa*
 - Using a certain fixed number of digits (typically 11 binary digits) for the *exponent*

6.02×10^{23}

mantissa exponent

Floating Point Numbers

- Matlab notation for 6.02×10^{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)

Discovering Mantissa Length

- Idea
 - $10^n + 1$ will be equal to 10^n when n is greater than the mantissa length
- Algorithm
 - Test $10 + 1 == 10$
 - Test $10^2 + 1 == 10^2$
 - Test $10^3 + 1 == 10^3$
 - ...
 - until equality is found
- Code

```
n = 1;
while 10^n + 1 ~= 10^n
    fprintf('Mantissa is at least %d decimal digits\n', n);
    n = n + 1;
end
```