Previous Lecture:
- Examples on vectors and simulation

Today’s Lecture:
- Finite vs. Infinite; Discrete vs. Continuous
- Vectors and vectorized code
- Color computation with linear interpolation
- plot and fill

Announcements:
- Project 3 due tonight at 11pm
- Prelim 1 on Tues 3/15 at 7:30pm

Loop patterns for working with a vector

```matlab
% Given a vector v
for k = 1:length(v)
    % Work with v(k)
    % E.g., disp(v(k))
end
```

```matlab
% Given a vector v
k = 1;
while k <= length(v)
    % Work with v(k)
    % E.g., disp(v(k))
    k = k+1;
end
```

Discrete vs. continuous

A plot is made from discrete values, but it can look continuous if there are many points.

Generating tables and plots

```matlab
x = linspace(0,2*pi,9);
y = sin(x);
plot(x,y)
```

Note: x, y are shown in columns due to space limitation; they should be rows.

How did we get all the sine values?

Built-in functions accept arrays

<table>
<thead>
<tr>
<th>x</th>
<th>sin(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1.57</td>
<td>1.00</td>
</tr>
<tr>
<td>3.14</td>
<td>0.00</td>
</tr>
<tr>
<td>4.71</td>
<td>-1.00</td>
</tr>
<tr>
<td>6.28</td>
<td>0.00</td>
</tr>
</tbody>
</table>

and return arrays

<table>
<thead>
<tr>
<th>x</th>
<th>sin(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
<td>-1.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Screen Granularity

After how many halvings will the disks disappear?
Xeno’s Paradox

- A wall is two feet away
- Take steps that repeatedly halve the remaining distance
- You never reach the wall because the distance traveled after $n$ steps =
  \[ 1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n} = 2 - \frac{1}{2^n} \]

Example: “Xeno” disks

What do you need to keep track of?
- Diameter ($d$)
- Position
  - Left tangent point ($x$)

<table>
<thead>
<tr>
<th>Disk</th>
<th>$x$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0+1</td>
<td>1/2</td>
</tr>
<tr>
<td>3</td>
<td>0+1/2</td>
<td>1/4</td>
</tr>
</tbody>
</table>

% Xeno Disks

```
% Xeno disks
DrawRect(0,-1,2,2,'k')
% Draw 20 Xeno disks
d= 1;
x= 0;  % Left tangent point
for k= 1:20
  % Draw kth disk
  % Update x, d for next disk
end
```

Here’s the output… Shouldn’t there be 20 disks?

The “screen” is an array of dots called pixels.
Disks smaller than the dots don’t show up.
The 20th disk has radius<.000001

Fading Xeno disks

- First disk is yellow
- Last disk is black (invisible)
- Interpolate the color in between
Color is a 3-vector, sometimes called the RGB values

- Any color is a mix of red, green, and blue
- Example: \( \text{color} = [0.4 \ 0.6 \ 0] \)
- Each component is a real value in \([0,1]\)
- \([0 \ 0 \ 0]\) is black
- \([1 \ 1 \ 1]\) is white

Example: 3 disks fading from yellow to black

\[
\begin{align*}
\text{r} &= 1; \quad \% \text{radius of disk} \\
\text{yellow} &= [1 \ 1 \ 0]; \\
\text{black} &= [0 \ 0 \ 0]; \\
\% \text{Left disk yellow, at x=1} \\
\text{DrawDisk}(1,0,r,\text{yellow}) \\
\% \text{Right disk black, at x=5} \\
\text{DrawDisk}(5,0,r,\text{black}) \\
\% \text{Middle disk with average color, at x=3} \\
\text{colr} &= 0.5\times\text{yellow} + 0.5\times\text{black}; \\
\text{DrawDisk}(3,0,r,\text{colr})
\end{align*}
\]

Vectorized code allows an operation on multiple values at the same time

\[
\begin{align*}
\text{yellow} &= [1 \ 1 \ 0]; \\
\text{black} &= [0 \ 0 \ 0]; \\
\% \text{Average color via vectorized op} \\
\text{colr} &= 0.5\times\text{yellow} + 0.5\times\text{black}; \\
\% \text{Average color via scalar op} \\
\text{for } k = 1: \text{length(black)} \\
\text{colr}(k) &= 0.5\times\text{yellow}(k) + 0.5\times\text{black}(k); \\
\text{Operation performed on vectors} \\
\end{align*}
\]

\[
\begin{align*}
\% \text{Draw } n \text{ fading Xeno disks} \\
d = 1; \\
x = 0; \quad \% \text{Left tangent point} \\
yellow = [1 \ 1 \ 0]; \\
black = [0 \ 0 \ 0]; \\
\text{for } k = 1:n \\
\% \text{Compute color of } k\text{th disk} \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\% \text{Draw } k\text{th disk} \\
\text{DrawDisk}(x+d/2, 0, d/2, \ldots) \\
x = x+d; \\
d = d/2;
\end{align*}
\]
Linear interpolation

<table>
<thead>
<tr>
<th>x</th>
<th>g(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>110</td>
</tr>
<tr>
<td>10</td>
<td>118</td>
</tr>
<tr>
<td>10.25</td>
<td>?</td>
</tr>
<tr>
<td>10.50</td>
<td>?</td>
</tr>
<tr>
<td>10.75</td>
<td>?</td>
</tr>
<tr>
<td>11</td>
<td>126</td>
</tr>
<tr>
<td>12</td>
<td>134</td>
</tr>
</tbody>
</table>

\[ g(10.5) = \frac{1}{2} g(11) + \frac{1}{2} g(10) \]

\[ g(10) = \frac{0}{4} g(11) + \frac{4}{4} g(10) \]

\[ g(10.5) = \frac{2}{4} g(11) + \frac{2}{4} g(10) \]

\[ g(10.75) = \frac{3}{4} g(11) + \frac{1}{4} g(10) \]

\[ g(11) = \frac{4}{4} g(11) + \frac{0}{4} g(10) \]

\[ f \cdot g(11) + (1-f) \cdot g(10) \]

Rows of Xeno disks

Code to draw one row of Xeno disks at some y-coordinate

for y = __ : __ : __

Does this script print anything?

\[ k = 0; \]

\[ \text{while } 1 + 1/2^k > 1 \]

\[ k = k+1; \]

\[ \text{end} \]

\[ \text{disp}(k) \]

Floating point arithmetic

Suppose you have a calculator with a window like this:

\[ \begin{array}{cccc}
+ & 2 & 4 & 1 \\
- & 3
\end{array} \]

representing \( 2.41 \times 10^{-3} \)

Floating point addition

\[ \begin{array}{cccc}
+ & 2 & 4 & 1 \\
+ & 1 & 0 & 0 \\
\end{array} \]

Result:

\[ \begin{array}{cccc}
+ & 3 & 4 & 1 \\
- & 3
\end{array} \]
Floating point addition

\[
\begin{array}{c}
+ & 2 & 4 & 1 & - & 3 \\
+ & 1 & 0 & 0 & - & 4 \\
\end{array}
\]

Result: \[
\begin{array}{c}
+ & 2 & 5 & 1 & - & 3 \\
\end{array}
\]

Floating point addition

\[
\begin{array}{c}
+ & 2 & 4 & 1 & - & 3 \\
+ & 1 & 0 & 0 & - & 6 \\
\end{array}
\]

Result: \[
\begin{array}{c}
+ & 2 & 4 & 1 & - & 3 \\
\end{array}
\]

The loop DOES terminate given the limitations of floating point arithmetic!

\[
k = 0; \\
while 1 + 1/2^k > 1 \\
k = k+1; \\
end \\
disp(k)
\]

\[1 + 1/2^{53}\] is calculated to be just 1, so "53" is printed.

Computer arithmetic is inexact

- There is error in computer arithmetic—floating point arithmetic—due to limitation in "hardware." Computer memory is finite.

- What is \[1 + 10^{-16}\]?
  - \[1.0000000000000001\] in real arithmetic
  - \[1\] in floating point arithmetic (IEEE)

- Read Sec 4.3

Vectorized code

- A Matlab-specific feature

- Code that performs element-by-element arithmetic-relational/logical operations on array operands in one step

- Scalar operation: \[x + y\]
  where \(x, y\) are scalar variables

- Vectorized code: \[x + y\]
  where \(x\) and/or \(y\) are vectors. If \(x\) and \(y\) are both vectors, they must be of the same shape and length

Vectorized addition

\[
\begin{array}{c}
x & 2 & 1 & .5 & 8 \\
+ & y & 1 & 2 & 0 & 1 \\
\end{array}
\]

\[= z & 3 & 3 & .5 & 9 \]

Matlab code: \(z = x + y\)
### Vectorized multiplication

Vectorized multiplication:

\[
\begin{align*}
\text{a} & = \begin{bmatrix} 2 & 1.5 & 8 \end{bmatrix} \\
\text{x} & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
\text{b} & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
\text{c} & = \begin{bmatrix} 2 & 2 & 0 & 8 \end{bmatrix}
\end{align*}
\]

Matlab code: `c = a .* b`

### Vectorized element-by-element arithmetic operations on arrays

Vectorized element-by-element arithmetic operations:

\[
\begin{align*}
\text{a} + \text{b} & \rightarrow \text{c} \\
\text{a} - \text{b} & \rightarrow \text{d} \\
\text{a} \times \text{b} & \rightarrow \text{e} \\
\text{a} \div \text{b} & \rightarrow \text{f}
\end{align*}
\]

A dot (.) is necessary in front of these math operators.

### Shift

Shift:

\[
\begin{align*}
\text{x} & = \begin{bmatrix} 2 & 1.5 & 8 \end{bmatrix} \\
\text{y} & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
\text{z} & = \begin{bmatrix} 5 & 4 & 3.5 & 11 \end{bmatrix}
\end{align*}
\]

Matlab code: `z = x + y`

### Reciprocate

Reciprocate:

\[
\begin{align*}
\text{x} & = \begin{bmatrix} 2 & 1.5 & 8 \end{bmatrix} \\
\text{y} & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
\text{z} & = \begin{bmatrix} 5 & 1 & 2 & 125 \end{bmatrix}
\end{align*}
\]

Matlab code: `z = x ./ y`

### Vectorized element-by-element arithmetic operations between an array and a scalar

Vectorized element-by-element arithmetic operations between an array and a scalar:

\[
\begin{align*}
\text{x} & = \begin{bmatrix} 2 & 1.5 & 8 \end{bmatrix} \\
\text{y} & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
\text{x} / \text{y} & \rightarrow \text{z}
\end{align*}
\]

A dot (.) is necessary in front of these math operators.

The dot in `.* .* ./` not necessary but OK

### Plot using vectorized computation

Plot using vectorized computation:

\[
f(x) = \frac{\sin(5x) \exp(-x/2)}{1 + x^2} \quad \text{for} \quad -2 \leq x \leq 3
\]

\[
x = \text{linspace}(-2,3,200); \\
y = \sin(5\times x) \times \exp(-x/2) ./ (1 + x \times \times 2);
\]

Matlab code: `plot(x,y)`

vectorized arithmetic operations on arrays