

Lecture 3

Numerical Methods

L. Olson

Department of Computer Science
University of Illinois at Urbana-Champaign

2005.24.01



On a sheet of paper...

Name 1 or 2 "tricks" that you found helpful...

Luke's:

- 1 » [rand(3) zeros(3,1)]
- 2 » figure;



On a sheet of paper...

Name 1 or 2 "tricks" that you found helpful...

Luke's:

- 1 » [rand(3) zeros(3,1)]
- 2 » figure;

The top 3...



Today: Programming Basics

Three things to keep in mind about Matlab when programming:

- 1 everything is linear
- 2 script versus function
- 3 vectorize vectorize vectorize



Interpreter vs. Scripts vs. Function

```
Current Directory: C:\Program Files\MATLAB\71\work
What's New
Command Window
To get started, select MATLAB Help or Demo
>>
>>
>>
>>
>> whos
>>
```

```
ditor - Untitled*
Edit Text Cell Tools Debug Desktop Window
n = 10;
x = linspace(0,1,n);
y = 3*x.^3 + 2*x.^2 + 5*x + 4;
plot(x,y,'r-');
y = 3*x.^3 + 2*x.^2 + 5*x + 4;
plot(x,y,'r-');
```

One copy of x

```
ditor - Untitled
Edit Text Cell Tools Debug Desktop Window Help
n = 10;
x = linspace(0,1,n);
y = mypoly(3,2,5,4);
plot(x,y,'r-');
y = mypoly(1,1,1,1);
plot(x,y,'r-');
% subfunction
function y = mypoly(a,b,c,d,x)
y = a*x.^3 + b*x.^2 + c*x + d;
```

Copy of x for each function call

Interpreter vs. Scripts vs. Function

Interpreter:

- quick access
- good for developing ideas
- good for accessing data in the workspace
- poor performance and editing/debugging capabilities

Script

- good test bed: all info in the workspace
- fast, linear
- poor profiling/optimization

Functions

- can be optimized and cached
- doesn't fill up the workspace with memory that you have to manage, but...
- pass by value (ouch!)
- reusable code



Functions

- save in *functionname.m*
- can have 0, n, or variable number of inputs
- can have 0, n, or variable number of outputs
- use `return;` to leave a function early
- use `inline(...)` to define without another file
- variables not "passed in" are *local*: only available within the function scope
- use `global varname;` to declare a variable that is visible through all functions



More on functions:

- use `eval` to call a function from a string
- `varargin`: variable number of argument inputs
- `nargin`: number of argument inputs
- `varargout`: variable number of argument outputs
- `nargout`: number of argument outputs



break versus return

break

Use to skip out of current code block and continue with code

```
function z = test()  
n = 10;  
for i=1:10  
    if(i>7)  
        z=0;  
        break;  
    end  
    disp(i);  
end  
disp(done!)  
z=1;
```

break

Use to immediate return to the calling function

```
function z = test()  
n = 10;  
for i=1:10  
    if(i>7)  
        z=0;  
        return;  
    end  
    disp(i);  
end  
disp(done!)  
z=1;
```

Vectorize!

The single most effective aspect in Matlab in order to build efficient code:

```
function test1(n)
t=cputime;
for j=1:n
    x(j) = sin(j);
end
disp(cputime-t);
```

```
>> test1(10000)
    0.1250
>> test1(100000)
    82.6094
```

```
function test2(n)
t=cputime;
j=1:n;
x=sin(j);
disp(cputime-t);
```

```
>> test2(100000)
    0.0156
>> test2(1000000)
    0.2656
```



More vectorization: Memory

Preallocate memory:

- set size of the matrix beforehand: `A=zeros(n)`
- this allocates a chunk of memory at once instead of multiple times on the fly

```
function test3(n)
t=cputime;
for j=1:n
    x(j) = sin(j);
end
disp(cputime-t);
```

```
>> test3(100000)
82.0469
```

```
function test4(n)
t=cputime;
x=zeros(1,n);
for j=1:n
    x(j) = sin(j);
end
disp(cputime-t);
```

```
>> test4(100000)
0.0313
```

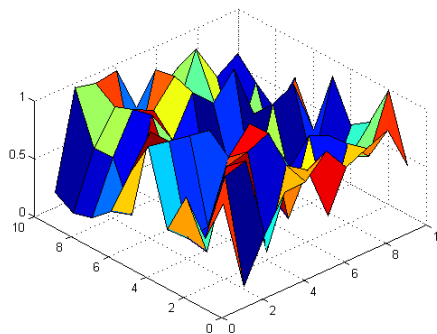
Many Vectorized Functions

- `all(X)`: true if all elements are nonzero
- `any(X)`: true if any elements are nonzero
- `is*`: checks for *: `isnan`, `isfinite`, `isinf`, `isempty`, `isnumeric`
- `find`: finds subset satisfying conditions

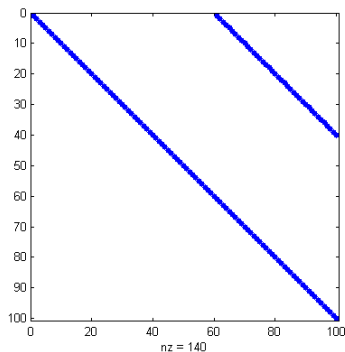


Visualizing Data

```
» A=rand(10); » surf(A);
```



```
» A=diag(ones(100,1)) +  
diag(ones(40,1),60);  
» spy(A);
```



Advice on programming

- use `disp` `pause` `spy` `surf` `whos` `profile` and semicolons to help debug and visualize your code.
- Matlab's in-house debugger often not enough
- vectorize when you can
- avoid loops when you can
- modularize your code...this will take practice



An exercise...

newsy(A)

- plots red for negative values in the matrix
- plots blue for positive values in the matrix
- plots green for zeros values in the matrix

An exercise...

newspy(A)

- plots red for negative values in the matrix
 - plots blue for positive values in the matrix
 - plots green for zeros values in the matrix
-
- read §3.4 - §3.6
 - glance at chapter 4
 - start playing around with the wiki