PSY8219
Scientific Computing for Psychological and Brain Sciences

Mondays 1:10-3:45
Wilson Hall 316

Thomas Palmeri
507 Wilson Hall
thomas.j.palmeri@vanderbilt.edu

Office Hours: Mon 11:00-12:00 or by appointment
Course Goals

• For students to develop proficiency designing, writing, and debugging computer programs to control experiments, perform data analyses, and simulate neural or psychological mechanisms

programming methods
algorithms and data structures
computational and numeric methods
high performance computing

how to design programs
how to debug programs
best practices
how to think computationally
Introductions
- name
- undergrad / grad / postdoc / other
- major / program
- lab / mentor (if relevant)
- programming experience (how much, what languages, what courses)
- where/when learned (middle school, high school, undergrad, other)
- Matlab experience or not
- why you're taking (sitting in) this course / what you hope to learn

you can also email or chat if you have detailed questions or recommendations regarding topics we might cover in class
Why do we program in psychology and neuroscience?

• controlling experiments
  present stimuli, record behavior, record physiology
  limitations of packages like e-prime

• performing data analyses and data visualization
  organizing data, transforming data, description and
  inferential statistics, custom analyses, graphs
  limitations of statistical scripting (SPSS or SAS)

• simulating mechanisms
  formalize theories of mind and brain in computations
  limitations of non-programming simulation tools
Scientific Computing

• We focus on aspects of computer programming and numerical methods critical to psychology and neuroscience. There are lots of algorithms and data structures from computer science that we will not discuss.

• Attempt to focus on “computational thinking” and not just mere “programming”. 
Best Practice

• Throughout the course I will try to emphasize “best practices” for programming specifically and also for any use of computers in science generally.
Throughout the course I will try to emphasize “best practices” for programming specifically and also for any use of computers in science generally.

Our first **Best Practice**
- Back up your computer every day.
- Ideally have two backups in two different places.
- Use external hard drives and/or servers.
- Talk to your advisor about backing up.
WD 2TB Elements Portable External Hard Drive - USB 3.0 - WDBU6Y0020BBK-WESN

Price: $64.99 prime | FREE One-Day

Get $60 off instantly: Pay $4.99 upon approval for the Amazon Prime Store Card.

Eligible for AmazonSmile donation.

Capacity: 2TB

Style: Portable

Service: Get professional installation Details

Without expert installation Include installation +$67.29

What's included with service

- USB 3.0 and USB 2.0 compatibility
- Fast data transfers
- Improve PC performance
- High Capacity
- WD quality and reliability
Course Information

• Textbook (recommended)
  – For Today : Chapter 1
  – For Next Week : Chapters 2, 7, and 8

• Course Website
  – http://catlab psy.vanderbilt.edu/palmeri/psy319
Syllabus

Course Schedule

Announcements

Course Description

The goal of this course is for students to develop proficiency designing, writing, and debugging computer programs to control experiments, perform data analyses, and simulate neural and psychological mechanisms. We discuss programming methods, algorithms and data structures, computational and numerical methods, and high performance computing techniques.

Contact

Dr. Thomas Palmeri
Department of Psychology
507 Wilson Hall
615-343-7900
thomas.j.palmeri@vanderbilt.edu
catlab.psy.vanderbilt.edu

Office Hours: Mon 12:00-1:00 or by appointment

http://catlab.psy.vanderbilt.edu/palmeri/psy319/
Course Requirements

• Weekly Homework Assignments (90%)
  – Homework 1 is on the web. Due next Monday.
  – Turn in assignments via Brightspace
  – I will read and run your code.
  – The code must work, it must be commented, it must be general, it must use good programming style.
  – I encourage students to help each other out, but you need to do the assignments on your own.

• Attendance and participation (10%)
Auditors

- Graduate students who audit should register formally – see Angel Gaither in WH301.

- Auditors and those sitting in can participate commensurate with their effort on class work.
• Hardware
  – chipsets, monitors, CPU vs. GPU

• O/S
  – Mac OS, Windows, Linux

• Programming Languages
  – MATLAB, R, Python, C++, Java, Javascript
Why will we focus on Matlab programming? What are some differences between programming languages?

- Your statistics course requires R.
- Another requires SAS.
- Your lab uses Python.
- Or E-Prime.
- Or a custom system.
- CS1101 teaches Java.
- CS2204 and SC3250/5250 use Python.
How do languages differ?

Popularity

- Popularity means lots of code/tools that are shared.
- Popularity means many other users who can help.
  - MATLAB, Python, R
    - most popular languages in psychology and neuroscience
    - Matlab perhaps the most popular in Wilson Hall
  - C / C++
    - still used for certain critical applications where speed is critical
  - Fortran, Pascal, LISP, BASIC
    - widespread usage in the 70s and 80s
    - Fortran still used extensively in numeric computing tools

*thousands of programming languages have been developed...*
How do languages differ?
Cost

- **Free**
  - Python, R, C++, Octave (Matlab clone)

- **Expensive**
  - Matlab (commercial product)
  - “free” with Vanderbilt site license
  - without site license, $2000+ per user in year one, $500-1000 per year per user every year after (10x that much in industry)
How do languages differ?

Programming Environment

• IDE (Integrated Development Environment)
  – write and edit computer programs
  – debug computer programs

• MATLAB
  – outstanding, you get what you pay for

• Python, R, C++
  – mixed bag, idiosyncratic, paid and free solutions
  – a lot of variety in IDEs (IDE and language are separate)
How do languages differ?

Programming Environments

- **IDE (Integrated Development Environment)**
  - write and edit computer programs
  - debug computer programs

- **MATLAB**
  - outstanding, you get what you pay for

- **Python, R, C++**
  - mixed bag, idiosyncratic, paid and free solutions

**In The Old Days**

- use an editor program to write code
- use a compile to create object files
- use a linker to connect object files with libraries and create an executable program
- debug using print statements
- debug using a standalone debugger program
How do languages differ?
Toolboxes, Libraries, and Extensions

• What functions and tools are available above and beyond the core programming language?

• Matlab
  – 100+ toolboxes and 1000s of routines by Mathworks
  – 1000s of free routines created by users
    www.mathworks.com/matlabcentral/
  – free Psychophysics Toolbox psychtoolbox.org/

• Python and R
  – 1000s of free routines created, shared, and supported by scientists and statisticians
How do languages differ?

Functionality

• What can the language and toolboxes do for you?

• MATLAB
  – designed for numeric operations (Matlab = Matrix Laboratory)
  – outstanding graphing capabilities
  – difficult to implement non-numeric data structures
  – object-oriented capabilities are recent and tacked on
  – lean but quirky syntax
  – many computer scientists hate Matlab
How do languages differ?

Functionality

- R
  - designed from the ground up for statistical analyses
  - great for analyses and simulation
  - excellent graphing capabilities
  - more general than scripting languages like SAS or SPSS
  - not used as a general purpose programming language
How do languages differ?

Functionality

- Python
  - complete programming language
  - computer scientists love it
  - mathematics and graphics comparable to Matlab and R
  - object-oriented
  - designed to interface well with other languages
How do languages differ?

Functionality

- **C++**
  - full object-oriented language
  - “assembly language” of the 21st century
  - very few standard or centralized libraries
  - but there are some specialized, powerful libraries
  - Python and Matlab are "written in" C/C++
How do languages differ?
Interpreted vs. Compiled Language

• Matlab, Python, R
  – interpreted language
  – lines of a program are interpreted one at a time, as if you were typing them in by hand (only a bit faster)
  – programs run inside Matlab or Python or R

• C++, Fortran
  – programs compiled and linked to libraries
  – executable programs (.exe) files (or equivalent)
  – most programs you run are compiled
  – the operating system on your computer is compiled
How do languages differ?

Speed

• Matlab, Python, R
  – slow, because it is interpreted
  – these languages became popular for real applications only in the past 5-10 years as computer speeds have increased
  – make gpu/parallel programmer easy(er)

• C++, Fortran
  – blazingly fast
  – a C++ program can sometimes run 100x faster than an equivalent Matlab program
  – so folks doing simulations sometimes need to write some of their code in C++ sometime
  – gpu/parallel programming hard
Matlab

Start today by covering the basics.

Some of you already have experience with Matlab, but maybe we’ll discuss some new aspects.

I’ll go through things are a pretty brisk pace, so ask questions if you want me to slow down.
Installing Matlab

• Obtain Matlab
  – free to download with Vunet ID (see syllabus)

• Install Matlab
  – easiest to simply install everything
  – if disk space is limited
    • you don’t need Simulink
    • you don’t need the Sim toolboxes
    • you should install these toolboxes at a minimum:
      Global Optimization Toolbox
      Image Processing Toolbox
      Optimization Toolbox
      Signal Processing Toolbox
      Statistics and Machine Learning Toolbox
bring laptops to class to run Matlab

(but please, please, please, no email, no facebook, no shopping, no twitter, no gchat)
Best Practices

Be careful about upgrading your operating system.

For example, Matlab sometimes does not work right after a new OSX upgrade on Macs.

More generally, specialized applications and programs often break when an OS changes.

Also, be careful about upgrading Matlab. Some applications may not work in an upgraded version.
Getting Oriented to the Matlab IDE
Getting Oriented to the Matlab IDE

generating starting with Matlab
Getting Oriented to the Matlab IDE

Command Window
Getting Oriented to the Matlab IDE
Getting Oriented to the Matlab IDE

Current Folder
Getting Oriented to the Matlab IDE
Getting Oriented to the Matlab IDE

we will not talk about how to use many built-in tools
Getting help from Matlab

• Within Matlab
  >> help clear
  >> doc clear

• Online documentation
  http://www.mathworks.com/help/techdoc/

• PDF files

• Video Tutorials
  http://www.mathworks.com/videos/
Scripts in Matlab
Scripts in Matlab

- The Command Window is useful for trying out commands and debugging, but we will use Matlab scripts (.m files) for most things.

- "Script" is just a series of commands. Rather than typing them, you run the script. Later we will talk about functions.

- Being an interpreted language, a complex program in Matlab is really just a more elaborate script.
Creating a script

• Change Current Folder
• Create Script
  – New → Script
• Save / Save As
  – Save ...
  – Save As ...
• “Dock” / “Undock” script window (useful if you have multiple monitors)
Best Practices

• Use folders/directories, subfolders/subdirectories
• Make a copy of a working program before you make new edits to it.
• Keep copies of experiment programs and analysis programs with any data you collect in an experiment.
Comments in Scripts

% this is a comment

>> x = 2  % this is a comment too

Use comments to:
- briefly describe what a section of code does
- reference particular formulas or calculations
- describe variables the first time you assign them
- write comments for yourself
- write comments for others in your lab
Running a Script

Green Arrow
  Save and Run

Debugging
  setting breakpoints in code
Defining and Executing Sections

%% a comment and a start of a section

sections should be self-contained units of code

single-click (left click) to select a section
two-finger click (right click) to select
Evaluate Current Section
Variables
Variables

• A variable is the name of a place where you store data in computer memory.

• Every language has rules for variable names. And not all the rules are the same.*

• Matlab variables
  – must start with a character (not a number)
  – can include lowercase, uppercase, number, and _
  – can be any length, but only the first N characters may matter (\texttt{nameLengthMax} = 63 on my installation)

* e.g., \texttt{my.var} is an acceptable variable name in R, but indicates a structure in Matlab
<table>
<thead>
<tr>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1x</td>
</tr>
<tr>
<td>x_fact</td>
<td>x!</td>
</tr>
<tr>
<td>sin</td>
<td>end</td>
</tr>
</tbody>
</table>

Some keywords (like `end`) are reserved.

But Matlab lets you overwrite others (like `sin`)*

* Leading to some of the most difficult bugs to locate in Matlab.
Best Practice

- use meaningful variable names
  - not xxxy, xxxx, xyxyxy
- but also use short variable names
  - idx vs. index_counter
- you can use letters from equations, if meaningful
  - x, N, a
- by convention, i, j, and k are often used to index
  - but using i and j can means no more complex numbers
Matlab Workspace

• The Workspace is all currently defined variables
• The Workspace Window shows names and values of currently defined variables

```matlab
>> clear x           % clear a particular variable
>> clear all         % clear all variables
```
Variable Types (called “Classes” in Matlab)

• Every programming language defined different Types for different kinds of data

  Numeric
  Boolean
  Strings and Characters
  Arrays, Vectors, Matrices
  Cell Arrays
  Structures
  Tables
  Categorical Arrays
How you can check the type of a variable

- You can check the type of a variable

```plaintext
>> i = 1
>> whos i
>> i = pi
>> whos i
>> i = 'a'
>> whos i
```
Numeric Types
Does Matlab check the type?

• Some programming languages require you to declare the type of a variable explicitly. If you deviate from that type you get an error.

• Matlab let’s you change types on the fly.

• It also let’s you do things like this:

  ```matlab
  >> x = 1
  >> y = 'a'
  >> x+y
  ```
Numeric Types

• By default, every number is “double-precision floating point”

  1 bit = 0 or 1

  8 bits in a byte
Numeric Types

- By default, every number is “double-precision floating point”

64 bits (8 bytes) per number in scientific notation

<table>
<thead>
<tr>
<th>Bits</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Sign (0 = positive, 1 = negative)</td>
</tr>
<tr>
<td>62 to 52</td>
<td>Exponent, biased by 1023</td>
</tr>
<tr>
<td>51 to 0</td>
<td>Fraction $f$ of the number $1.f$</td>
</tr>
</tbody>
</table>

What’s critical to remember is that there is limited “real estate” to store significant digits of a number.
Numeric Types

• By default, every number is “double-precision floating point”

64 bits (8 bytes) per number in scientific notation

<table>
<thead>
<tr>
<th>Bits</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Sign (0 = positive, 1 = negative)</td>
</tr>
<tr>
<td>62 to 52</td>
<td>Exponent, biased by 1023</td>
</tr>
<tr>
<td>51 to 0</td>
<td>Fraction ( f ) of the number ( 1.f )</td>
</tr>
</tbody>
</table>

\[(\text{Sign}) \ 1.f \times 10^{\text{Exponent}}\]
Significant Digits

• Change the number of significant digits you can see

```python
>> pi
>> format long
>> pi
>> format short
>> pi
```

`format` only changes how the number is displayed, not how a number is represented internally!
Significant Digits

\[ \begin{align*}
\text{>> } & 0.000000000000000000000000000000001 \\
\text{>> } & 1 - 0.0001 \\
\text{>> } & 1 - 0.000000000001 \\
\text{>> } & 1 - 0.000000000000000000000000000000001 \\
\text{this is beyond the numeric resolution of Matlab}
\end{align*} \]

“... squeezing infinitely many real numbers into a finite number of bits requires an approximate representation ...” - Goldberg (1991)
What is $1000 \times 0.01$?
What is $1000 \times .01$?

\[\text{>> } 1000 \times .01\]
What is $1000 \times .01$?

$$\text{>> format long}$$

$$\text{>> 1000 } \ast \text{ .01}$$

$$\text{>> sum = 0; for i=1:1000; sum = sum+.01; end}$$
Special Numbers in Matlab

Try these:

```matlab
>> 1/0
truly infinite
```

```matlab
>> exp(1000)
beyond the numeric resolution of Matlab
```

```matlab
>> 0/0
>> Inf/Inf
```
Simple Mathematical Operations in Matlab

+ addition
- subtraction
* multiplication
/ division
^ exponentiation

As we’ll see, these may do different things depending on whether a variable is scalar or a matrix

```matlab
>> x = 1
>> x = [ 2 3 ; 4 5 ]
```
What do you think these will give you?

(first without punching them into Matlab)

\[
\begin{align*}
\texttt{2} & \quad \texttt{^}\quad \texttt{2+1} \\
\texttt{2+3} & \quad \texttt{^}\quad \texttt{2} \\
\texttt{2} & \quad \texttt{*}\quad \texttt{3^2} \\
\texttt{4+2} & \quad \texttt{/}\quad \texttt{1+2} \\
\texttt{-2} & \quad \texttt{^}\quad \texttt{2}
\end{align*}
\]
What do you think these will give you?

How Matlab interprets these

```
>> (2 ^ 2)+1
>> 2+(3 ^ 2)
>> 2 * (3^2)
>> 4+(2 / 1)+2
>> -(2 ^ 2)
```
Always use parentheses.

\[
\begin{align*}
\text{>> } 2 & \ ^\wedge \ (2+1) \\
\text{>> } (2+3) & \ ^\wedge \ 2 \\
\text{>> } 2 & \ * \ (3^2) \\
\text{>> } (4+2) & \ / \ (1+2) \\
\text{>> } (-2) & \ ^\wedge \ 2
\end{align*}
\]
Order of Operations in Matlab

1) parentheses ( )
2) exponents ^
3) positive (+) or negative (-) (not add or subtract)
4) multiplication (*) or division (/)
5) addition (+) or subtraction (-)

Best Practices
Don’t rely on order of operations. Use parentheses.

>> (-2)^2
>> -(2^2)
Modulo Operator

- calculates the “remainder” after division; used a fair amount in programming

what should these give you?

```plaintext
>> mod(10,3)
>> mod(3,3)
>> mod(1,5)
```
A few other mathematical functions

Built-in constants
>>> pi

Functions
>>> sin(pi/3)
>>> cos(deg2rad(60))
>>> log(10)
>>> log10(10)
>>> exp(1)
>>> abs(-1)

NOTE: most of these can be overwritten by variable names
A few other mathematical functions

>> round(1.49)
>> round(1.51)
>> ceil(1.2)
>> floor(1.2)
the “equals sign” =

• What can = mean in mathematics (outside Matlab)?
  – assert equivalence
    2 + 1 = 3
  – ask for a solution
    2 + 1 = ?
  – pose an algebraic problem
    2 + x = 3
  – ask a question regarding equivalence
    does 2 + (3-1) = 3?
In Matlab, = is an assignment operator

```matlab
>> x = 2
```

What does this do?

- if x does not exist
  - create it, reserve a place in memory for it, make it a double-precision floating point, assign its value to 2

- if x does exist
  - see if the type needs to be changed, if so change it, assign its value to 2

```matlab
>> 2 = x
```

Assignment in R:
```
x <- 2
```
```python
>>> x = 2

>>> whos x

>>> x = 'cat'

>>> whos x
```
```
>> a = 23
>> b = a
>> a = 44
>> b
```
Sometimes we want to ask about equivalence

Does $2 + (3-1) = 3$?

How do we do that in Matlab?

```matlab
>> 2 + (3-1) == 3
```
Logical Types
Logical Operators
Logical/Boolean Types

```>> true
>> false

>> a = (1 == 1)
>> whos a

>> a = 2 + (1==1)
>> whos a```
```python
>>> x = 2;
>>> y = 1;
>>> z = 3;
```

suppresses output
Logical and Relational Operators

```matlab
>> x = 2;
>> y = 1;
>> z = 3;

>> x > y
>> y > z
```
Logical and Relational Operators

>  greater-than

<  less-than

>=  great-than-or-equal-to

<=  less-than-or-equal-to

==  equivalent (equal)

?  not-equivalent (not equal)
Be careful using == and ~= with real numbers

Does \((\sqrt{3})^2 = 3\)?
Be careful using == and ~= with real numbers

Does \((\sqrt{3})^2 = 3\)?

```plaintext
>> (sqrt(3))^2 == 3
>> (sqrt(3))^2 - 3
```

What’s going on?
How else might you check equivalence?
How else might you check equivalence?

```
>> epsilon = 0.0000001
>> abs((sqrt(3))^2 - 3) < epsilon
```

why is abs() needed?

Will this work?
```
>> realmin
>> abs((sqrt(3))^2 - 3) < realmin
```

Why or why not?
Logical Operators

```python
>>> x = 2
>>> y = 1
>>> z = 3

>>> (x > y) & (y > z)
```

Logical **And**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Logical Operators

```
>> x = 2
>> y = 1
>> z = 3
>> (x > y) & (y > z)
```

<table>
<thead>
<tr>
<th>Logical And</th>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>
Logical Operators

```python
>> x = 2
>> y = 1
>> z = 3
>> (x > y) & (y > z)
```

Logical **And**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Logical Operators

>>> x = 2
>>> y = 1
>>> z = 3

>>> (x > y) | (y > z)

 Logical **Or**

|   | A | B | A|B |
|---|---|---|---|
| 0 | 0 | 0 | ? |
| 1 | 0 | 0 | ? |
| 0 | 1 | ? |
| 1 | 1 | ? |
Logical Operators

>>> x = 2
>>> y = 1
>>> z = 3

>>> (x > y) | (y > z)

Logical **Or**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Logical Operators

```python
>>> x = 2
>>> y = 1
>>> z = 3
```

```
>>> xor((x > y), (y > z))
```

<table>
<thead>
<tr>
<th>Logical Xor</th>
<th>A</th>
<th>B</th>
<th>xor(A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>
Logical Operators

```python
>>> x = 2
>>> y = 1
>>> z = 3

>>> xor((x > y), (y > z))
```

Logical Xor

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>xor(A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Logical Operators

```python
>> x = 2
>> y = 1
>> z = 3
>> ~(x > y)
```

<table>
<thead>
<tr>
<th>A</th>
<th>~A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Use parentheses

\[\gg (x > y) \mid (y > z) \& (x > z)\]
\[\gg (((x > y) \mid (y > z)) \& (x > z))\]
\[\gg (x > y) \mid ((y > z) \& (x > z))\]

even if you don’t need them, they add to readability

never assume you don’t need them
Order of Operations in Matlab

1) parentheses ( )
2) exponents ^
3) positive (+) or negative (-) or logical not (~)
4) multiplication (*) or division (/)
5) addition (+) or subtraction (-)
6) relational operators (<,>,<=,>=,==,~=)
7) and (&)
8) or (|)