Incremental Testing

- Test as you go!
- Once low-level functions are tested, you know later bugs are in higher-level code
- NEVER EVER neglect to test even the simplest code (you WILL get burned!)
- Then, put components together in a minimal end-to-end system and test it
- Make sure algorithms perform as expected (e.g. convergence rate)
- One test isn’t enough!
Localizing Bugs

- Hardest part of debugging: finding bugs!
- Never simply guess what part of your code the bug is in, you will be way off
- Once you know what function it's in, through testing of individual components, check values of arguments and local variables
- Debuggers can pinpoint crash, but if it involves memory issues, bug can still be instigated elsewhere

The Mighty `print` Statement

- Primitive debugging technique: print out values along the way to see what your program really is doing
- Many ridicule this and advocate debuggers, but debuggers are flaky, and let's face it: printing works!
- When printing values, use `<< endl` to force output buffer to be flushed, so output is up-to-date
Get the Computer to Help

• Write output routines (by overloading <<) for new data structures to check values in human-readable way while debugging
• Also add an output function that prints out absolutely everything in the structure, including pointers
• Also overload >> for interactive testing
• Of course, test these thoroughly too!

Using Debuggers

• Popular debuggers:
  – `gdb`: GNU debugger, for use with by g++
  – `ddd`: popular on Linux systems
• Need to be compiled in debug mode
  – default on MDS
  – when using g++, use `-g` option
• If program crashes on UNIX, core file is dumped containing execution info
  – File is named `core`
  – Type `unlimit` at prompt to make sure it’s dumped, then use `gdb exec-name core`
Debugging Functional Objects

- *Functional objects* are functions that implicitly represent objects, such as a function that computes a matrix-vector product.
- Black-box testing: comparing actual outputs with expected outputs.
- White-box testing: testing how output values are computed.
- Test scale invariance: how should output change if input is scaled?

Error and Exception Handling

- When writing an interface, determine what are invalid input arguments, and add code to throw exceptions.
- C functions use return values to indicate failure. If return type is
  - A pointer, `NULL` is returned
  - A nonnegative integer, -1 is returned
- Thoroughly document when your code throws exceptions, so users of your code can catch them!
Compare and Contrast

- Often, scientific computing software is prototyped in MATLAB, then implemented in a compiled language for efficiency on large-scale problems.
- Older version can be used to test newer version.
- By prototyping in MATLAB first, can take advantage of interactive environment for easy testing.

Tracking Bugs

- If coding by yourself, no bug-tracking required: find 'em and fix 'em.
- For larger projects, need to document the evolution of each bug, from the time it is detected to the time it is resolved.
- Various software tools provide software configuration management, which includes version and bug tracking, as well as task management.
Stress & Performance Testing

• Stress testing
  – Large, extreme or degenerate inputs,
  – Examples: large or near-singular matrices, or arithmetic over/underflow

• Performance testing
  – memory usage, speed, accuracy
  – For direct methods like Gaussian elimination, these criteria are in sync
  – For iterative methods, they're a tradeoff

• Number of iterations or operations may be more reliable measures than time

Next Time

• Syntax of mathematical expressions
• Recognizing expressions from tokens
• Project 4 preview