MATH 6390
General Advice on Scientific Computing Projects

1. Write your code so you can read it like a technical paper several months after you wrote it, but which time you will have forgotten why you wrote the code, what it does, and how it does it.

2. Choose variable names and function names so as to improve readability and aid understanding. When you code up a numerical analysis algorithm like an Runga-Kutta method for solving an ODE or an optimization method like Newton's method, use variable names that closely correspond to variable names in pseudo-code for the algorithm. For other variables and functions use names that are as descriptive and specific as possible. For example, `GlobalError` is more descriptive than `GE` and `LongJumpODEFunction` is clearer than `JumpBody`.

3. Always put code comments on the line (or lines) prior to the code being commented, not to the right of the code on the same line.

4. Often, using well-chosen variable names means you don't even need to add code comment explaining what that variable means.

5. The layout of code and comments on the page can either help or hinder readability and hence understanding. **White space imposes structure.** Make it clear where functions begin and end using comment symbols, use indentation to make clear where loops being and end, put comments by `end` statements so it is clear what block is being ended, (e.g. `end %for loop over tolerance values`), use lots of blank lines to improve readability, don't make any line of code or comment too long.

6. Use spacing on either side of (in)equality or other symbols to improve readability.

7. Develop code gradually and debug it as you write it.

8. Any time you use a numerical algorithm to solve an applications problem you should make sure that the results you obtain do not depend on the numerical discretization parameters of the algorithm. So for example, decrease the tolerance of an ODE solver by a factor of 2 or 10 to make sure your results and conclusions about the application don't depend on the tolerance.

9. Make good use of functions (and other high-level programming constructs like classes) to write code once and re-use it multiple times within your project.

10. All figures and plots must have axes labels (including units if appropriate) and titles. Think about whether it makes more sense to use markers, lines, or both. Choose ranges of values on horizontal and vertical axes to show pertinent information.

11. For plots, think about whether it makes sense to use a linear or log scale for any given variable. For example, for positive variables that vary over many orders of magnitude a log scale often makes more sense.

12. The discussion of your results should be in complete, grammatically correct, English sentences. You should make **quantitative** statements wherever possible, rather than qualitative ones. For example, "Higher air densities result in shorter jumps" is qualitative whereas "Increasing the density by a factor of two only decreases the length of the jump by 10%" is quantitative.

13. You should provide an explanation for the significant features in every figure or table. The explanation can be based on a qualitative understanding of the physics governing the problem, or on your understanding of the properties of the equation being solved and the numerical algorithm being used.

14. If you make a statement explaining a feature in a plot or something about the behavior of an algorithm always back that up with hard evidence. For example, when using an ODE solver, if a calculation shows that for a certain tolerance the relative global error is 1, then you might show a plot of the solution as a function of time that shows what goes wrong when you use such a large tolerance.