3450:589 Topics: Research Tools
Dr. Kevin Kreider, CAS 273, (330) 972-7519, kreider@uakron.edu

1. Project Development. As your project takes definition, try to anticipate whether you will do extensive code development, minimal code development or no code development. What computing platform will you use? Do you need to learn new techniques, will you be using techniques you've seen before but haven't mastered, or should you be the one teaching this course?

   (a) Notation. Develop notation as you go that will be suitable for writing code. If you'll be using MATLAB, k and K can coexist, but in FORTRAN, you'll need different variable names. Often, groups of parameters appear together. As soon as you can, label groupings with variable names that you can use in the code.

   (b) Equations, Variables, Parameters. You should create a master list in LaTeX of the set of equations you will need to solve. You should list all independent variables, all dependent variables and all parameters, with the mathematical notation, the variable names and the units. For parameters, list baseline values as well as ranges. All of this will end up in your thesis.

   (c) Derivations. Develop a system for sorting out your derivations. If you are nondimensionalizing a differential equation, write it carefully on paper, and clip the pages together. Write some notes to remind your future self of important details. Much of this will end up in your thesis, but some of the simpler steps might be omitted. You'd better write it out now while it's fresh, because in 6 months it will take you 5 times as long to write.

   (d) Discretizations. Start with the mathematical equations, domain, initial and boundary conditions. Discretize the domain first, and apply standard notation to subscripts – ie, x is always discretized as x_i and i is used for nothing else, t is always discretized as t_n and n is used for nothing else. The formal statement of the problem (equations, domain and conditions) will appear in your thesis, with a description of the discretization.

   (e) Handout 1 – Corrosion Example, equations and parameters.

2. Code Development. This depends a lot on the complexity of the equations you are solving. I'm assuming in this outline that you have complicated equations and not much experience with numerics. Skip steps as conditions permit. You always start on paper, and move to the keyboard later.

   (a) Decide which computing platform you will use – MATLAB, FORTRAN, Maple, Python, for example. Do you need ODE solvers, PDE solvers, linear system solvers? Will you write your own or use existing routines? What kind of output do you need, and how will you generate graphs?

   (b) Write a high-level outline of the solution process, with just a few words for each step. Identify which steps are easy and which will be extensive or difficult. Also, which steps are you comfortable with, and in which steps will you need to learn something new?

   (c) Handout 2 – Corrosion Example, outline and discretization.

   (d) Do NOT try to solve your problem in one shot. Instead, build a set of codes that incorporate important algorithmic features in stages. Test and understand each stage before going to the next. One month of systematic development is much more palatable than two months of desperate bug hunting.

(e) Handout 3 – Corrosion Example, development stages.

(f) Folder organization. Develop a folder organization for housing your files. Use a folder called test for individual pieces of the algorithm, and other folders for the working code. Once your code is working, you'll want different folders for different analyses of the problem, which will probably appear in subsections in your results chapter.

(g) Handout 4 – Folder Example.

(h) Keeping track of codes. Code names should indicate the nature of the code without being too long. Keep distinct the names for source files and the names for output files so you don't accidently erase your source files. You should put a few comments at the beginning of each code, especially if you have different versions.

(i) Keeping track of output files. I'll present details later, but for now, realize that systematic names for your output files are vital for survival. After your code is working, we'll ask you to compare different cases, and you need a mechanism for doing that quickly and accurately. Put different scenarios in different folders, but you'll likely have several different versions of each scenario, in the same folder.

(j) Structure of a code. Begin with brief comments, followed by: initialization of parameters, main body and output.

(k) Handout 5 – Corrosion Example: fakephi0.f.

(l) Debugging. Write a segment of the code and print the intermediate results. Once the segment is working, comment or delete the write statements. For loops, run 2 time steps instead of 10000 so that you can see the intermediate results. Write results to temporary files if there is more output than you can see on the screen at once.

(m) How do you know if your results are right? First, you should have a rough feel for what solutions should be based on the physics of your problem and limiting cases. Second, run your code with a set of refined grids to see 'numerical convergence'.

3. Dealing with Output. A bit of forethought goes a long way here. Be clear at the start about what you think you'll want, but realize that the list will be change as preliminary results roll in. Output files names should have 2 or 3 components – a set of symbols to describe the data in the file, a set of symbols to describe which case was run, and possibly a set of symbols to indicate the time step.

   (a) Corrosion Example, output needs. At the least, we'll want the value of ϕ, the concentrations of metal, chloride and hydrogen ions, as well as the crevice height, at a variety of time steps. A possible file naming convention is Fphi[id][step], Fmet[id][step], Fchl[id][step], FHpl[id][step], Fcre[id][step]. [id] is a one character string identifying the case, and [step] is a 3 digit number identifying the time step. For example, FmetA053 is the metal concentration at time step 053 for case A.

   (b) Handout 6: FORTRAN example.

   (c) Handout 7: MATLAB example.

4. The Thesis. How much detail you write about numerics in your thesis depends on how involved your work was. If you used only standard routines, you can get by with less detail, but if you developed your own code and/or used routines in non-standard ways, you'll need to provide more detail. Your chapter on numerical work will likely have this format:

(a) A formal statement of the mathematical problem – governing equations, domain, boundary and initial conditions.

(b) A summary of the numerical method/s to be used.

(c) A summary of the numerical grid you are using.

(d) A list of the parameter values you are using.

(e) The discretization of each equation and BC/IC. If warranted, present the derivation. If it's not necessary, just present the final result. If it's very complicated, put the details in an appendix. As a general rule, you can skip simple algebraic work, but provide a brief description of the process. As an example, "Integrate (3.23) over the domain, using integration by parts on the second term on the left. The result is"

(f) A formal statement of the numerical problem – discrete versions of the governing equations and BC/ICs.

(g) When you describe your results, present the set of parameters you used for each case. Interpret the results in terms of the physics, not the mathematics. Don't write "As Y_0 is increased, H increases." Instead, write "As the chloride concentration is increased, the crevice depth increases."

(h) Include a copy of your code in an appendix. Because the thesis style uses double spacing, you'll need to change the baselineskip to single spacing. One way to do this is, in the appendix:

```
{
\baselineskip 12pt
\ begin{verbatim}
put your code here
\ end{verbatim}
}
```