Econ 561a (Part I)
Yale University
Fall 2005
Prof. Tony Smith

Computational Methods in Economics: General Points

- Three modes of science: theoretical, computational, empirical. Computational work shares elements of both theoretical and empirical work.
- Two meanings of computational economics: computation as a tool for doing standard economic theory vs. computation as a model for how economic actors behave.
- Fundamental shortcoming of the first kind of computational economics: lack of error bounds. How far apart are the computational model and the theoretical model?
- Computation produces "data" which can be used to generate theoretical conjectures. These conjectures could be exact (theorems) or approximate (systematic patterns).
- Use computation to measure quantitative magnitudes (e.g., the relative sizes of opposing effects).
- Hamming's motto: The Goal of Computing is Insight, Not Numbers.

Econ 561a (Part I)
Yale University
Fall 2005
Prof. Tony Smith

Rules of Thumb for Doing Good Computational Work in Economics

1. Start with the simplest possible model, preferably one with an analytical solution.
2. Add features incrementally.
3. Never add another feature until you are confident of your current results.
4. Use the simplest possible methods.
5. Accuracy is more important than speed or elegance.
6. Use methods that are as transparent as possible (i.e., methods for which the computer code reflects as closely as possible the economic structure of the problem).
7. When you learn (or develop) a new method, test it on the simplest possible problem, preferably one with an analytical solution.
8. Dan Bernhardt's rule: If you have n errors in a piece of code and you remove one, you still have n errors. Scrutinize your results, even if they look right. Look for anomalies. Assume your code is wrong until proven otherwise.
9. Graph, graph, graph. Two-dimensional graphs are more informative than three-dimensional graphs.

10. Be able to replicate all of your intermediate and final results instantly. Save exact copies of the code used for each run, together with inputs and outputs.

11. Watch the computations as they proceed.

12. Exploit homotopy.

13. Learn a fast language, such as C or Fortran.

14. Look for hidden structure. Always compute (and print out) a few more numbers than you need.

15. Get good initial conditions.

16. Use one-dimensional algorithms as much as possible.

17. Use algorithms that can be "tightened".

18. Avoid black boxes. Understanding how the algorithm works is critical to interpreting the results.

19. Remember that programming is a creative activity analogous to writing a sonnet or composing a sonata (a static, visual representation of a process). Craft your programs. Strive for efficiency and elegance in your computer code. Develop a style. Practice structured programming, i.e., write code that reflects the structure of the algorithm.

20. Don't program when you are tired. Don't program too quickly.