Numerical Methods in Economics
MIT Press, 1998

Notes for Chapter 2: Elementary Concepts

Kenneth L. Judd
Hoover Institution

February 20, 2020

The Economics of Computation

- Economics: the study of allocation of scarce resources
- Computation as an economic problem:
  - Scarce resources:
    * Hardware
      - CPU - central processing unit to manipulate data, do arithmetic
      - Memory - cache (very fast), RAM (fast), hard drive (slow)
    * Software
      - algorithms to do mathematics
      - interfaces that make it easy to describe problems to the computer, and easy for you to process output
    * Human time: opportunity cost varies greatly across economists
    * Human ability: I quote Chari: "Abstracting from irrelevant detail [in macro models] is essential given scarce computational resources not to mention the limits of the human mind in absorbing detail!"
  - Preferences
    * Reduce resource use
    * Increase accuracy of results
    * Increase reliability; i.e., the likelihood of the algorithm working

Computer Arithmetic

- Finite representation of real numbers: $\pm m 2^{\pm n}$
  - m: mantissa (an integer)
  - n: exponent (an integer)
  - Typical double precision:
    - Uses 64 bits ("single precision" used 32; common until mid-80's)
    - $m = 52$, $n = 10$, plus sign bits, one for each.
- Machine epsilon
  - Smallest relative quantity
  - Definition: $\varepsilon_M = \sup \{x | 1 + x \text{ ``} = \text{''} \ 1\}$ ("=" means computer equality, that is, up to computer error)
  - Double precision: ε_M is $2^{-52} \doteq 10^{-16}$ if $m = 52$; typical choice for desktops

- Machine zero
  - Smallest absolute quantity
  - Definition: $0_M = \sup \{x | x \text{ ``} = \text{''}\ 0\}$
  - Double precision: 0_M is about 10^{-308} if $n = 10$
- Extended precision:
  - Desirable to use in many cases; occasionally necessary.
  - Specialized hardware can reduce ε_M and/or 0_M
  - Software packages can produce arbitrary precision arithmetic.
    - Implemented in Mathematica, Maple, and some other programs.
    - Can be added to C and Fortran programs via operator overloading.

- Arithmetic operations take time
  - Integer addition is fastest
  - Real addition and multiplication are a bit slower
  - Division is slower than multiplication and addition
  - Power, trigonometric and logarithmic operations are slower
  - The computer does only addition and multiplication; everything else is a sequence of those operations

Errors: The Central Problem of Numerical Mathematics

- Rounding
  - $1/3 = .33333...$ needs to be truncated.
  - $1/10$ has a finite decimal expression but an infinite binary expression which must be cut

- Truncation
  - Exponential function is defined an infinite sum

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{2.7.1}$$

    but must be approximated with a finite expression, such as

$$\sum_{n=0}^{N} \frac{x^n}{n!}$$

  - Infinite series: If a quantity is defined by

$$x^* = \lim_{n \to \infty} x_n$$

    we must take x_n for some finite n.

- Error Propagation
  - Initial errors are magnified by many mathematical operations
  - Example: $x^2 - 26\,x + 1 = 0$
    - True solution $x^* = 13 - \sqrt{168} = .0385186\cdots$
    - Five-digit machine says

$$x^* = 13 - \sqrt{168} \doteq 13.000 - 12.961 = 0.039 \equiv \hat{x}_1$$

    - A better approach (even in five-digit machine)

$$13 - \sqrt{168} = \frac{1}{13 + \sqrt{168}} \;\doteq\; \frac{1}{25.961} \;\doteq\; 0.038519 \equiv \hat{x}_2,$$

- Numerical methods must formulate algorithms which minimize the creation and propagation of errors.

Efficient Evaluations of Expressions

- Consider cost of evaluating

$$\sum_{k=0}^{n} a_k\, x^k \tag{2.4.1}$$

  - Obvious method involves n additions, n multiplications, and $m-1$ exponentiations
  - Alternative: replace x^i with $x \cdot x \cdot ... \cdot x$, $i-1$ multiplications
  - Better method: compute $x^1 = x$, $x^{i+1} = x * x^i$, $i = 1, n$, to replace $n-1$ exponentiations with $n-1$ multiplications.
  - Best method is *Horner's method*:

$$\begin{aligned} & a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n \\ = & a_0 + x(a_1 + ... + x(a_{n-1} + x \cdot a_n)) \end{aligned} \tag{2.4.2}$$

Table 2.1: Polynomial Evaluation Costs

	additions	multiplications	exponentiations
Direct Method 1:	n	n	$n-1$
Alternative:	n	$n+(n-1)\,n/2$	0
Better Method	n	$2n-1$	0
Horner's Method:	n	n	0

- Lesson: Mathematically irrelevant changes to a mathematical expression can have large impact on computational time

Direct versus Iterative Methods

- Direct methods:
  - Aim to compute high accuracy answer
  - Uses fixed number of steps (depending on size of input)
  - Example: quadratic formula

$$
\begin{aligned}
0 &= ax^2 + bx + c \\
x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\end{aligned}
$$

- Iterative methods:
  - Compute sequence

$$x_{k+1} = g(x_k\,,\ x_{k-1},\ \cdots)$$

    and stop when stopping criterion is satisfied
  - Uses unknown number of steps
  - Accuracy is adjusted by adjusting stopping criterion
  - User faces a tradeoff between time and accuracy.
  - Example: By varying N, we can determine quality of approximation to e^x

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \doteq \sum_{i=0}^{N} \frac{x^i}{i!}$$

Rates of Convergence

- Suppose sequence $x_k \in \mathbb{R}^n$ satisfies $\lim_{k\to\infty} x_k = x^*$.
- x_k *converges at rate* q *to* x^* if

$$\lim_{k\to\infty} \frac{\| x_{k+1} - x^* \|}{\| x_k - x^* \|^q} < \infty, \tag{2.8.1}$$

  – If (2.8.1) is true for $q = 2$, we say that x_k *converges quadratically*. Example: $x_k = 10^{-2^k}$

  – If $q = 1$ and

$$\lim_{k\to\infty} \frac{\| x_{k+1} - x^* \|}{\| x_k - x^* \|} \leq \beta < 1 \tag{2.8.2}$$

  we say x_k converges *linearly at rate* β.

  – If $\beta = 0$, x_k is said to converge *superlinearly*.

  – Convergence at rate $q > 1$ implies superlinear (and linear) convergence.

Stopping Rules

- Iterative algorithms need to know when to stop
- Problem: Suppose you know that

$$x_{k+1} = g(x_k\,,\; x_{k-1}, \cdots)$$

  converges to some *unknown* solution x^*.
- We want to
  - Stop the sequence only when we are close to x^*
  - Stop sequence for small k

- Consider the sequence

$$x_k = \sum_{j=1}^{k} \frac{1}{j} \tag{2.8.3}$$

  - The limit of x_k is infinite
  - But x_k goes to infinity slowly; e.g., $x_{1000} = 7.485$
  - Hard to tell x_k diverges from examining numerical sequence.

- We rely on heuristic methods, *stopping rules*, to end a sequence.
  - Stop when the sequence is not "changing much."
    - "Stop when $|x_{k+1} - x_k|$ is small"

      $$|x_{k+1} - x_k| \leq \varepsilon$$

      for some small ε.
    - This rule is never good.
      - Depends on units.
      - Can fail spectacularly: for example, if $\varepsilon = 0.001$ it would end (2.8.3) at $k = 1000$, $x_k =$??.
    - This simple rule is not reliable

- Stop when the sequence is not "changing much" relative to zero
  * "Stop when $|x_{k+1} - x_k|$ is small relative to $|x_k|$"

$$\frac{|x_{k+1} - x_k|}{|x_k|} \leq \varepsilon$$

    for some small ε.
  * This may never stop if x_k converges to zero.
  * Solution is hybrid rule for any $\delta > 0$: "stop if changes are small relative to $\delta + |x_k|$"

$$\text{Stop and accept } x_{k+1} \text{ if} \frac{|x_{k+1} - x_k|}{\delta + |x_k|} \leq \varepsilon \tag{2.8.4}$$

  * (2.8.4) can fail spectacularly: for example, if $\varepsilon = 0.001$ and $\delta = 1$, it would end (2.8.3) at $k = 9330$, $x_k = 7.48547$.
  * This simple rule is not reliable
  * Economists love this rule; they know that "convergence" is helped by increasing ε.

– Use additional information

  * If x_k converges quadratically, (2.8.4) works well enough if $\varepsilon << 1$ since, for some $M > 0$

$$\| x_{k+1} - x^* \| < M \| x_{k+1} - x^* \|^2 \tag{2.8.1}$$

  * If x_k satisfies

$$\| x_{k+1} - x_k \| \leq \beta \| x_k - x_{k-1} \| \tag{2.8.5}$$

for some $\beta < 1$, then we know that

$$\| x_k - x^* \| \leq \frac{\| x_k - x_{k-1} \|}{1 - \beta}.$$

Hence, the rule

$$\text{Stop and accept } x_{k+1} \text{ if } \| x_{k+1} - x_k \| \leq \varepsilon(1 - \beta) \tag{2.8.6}$$

will stop only when $\| x_k - x^* \| \leq \varepsilon$.

* If x_k converges linearly at unknown rate $\beta < 1$, then at iteration k choose a large $L << k$, estimate β

$$\hat{\beta}_{k,L} = \max_{1<j<L} \frac{\| x_{k-j} - x_{k-j+1} \|}{\| x_{k-j-1} - x_{k-j} \|},$$

  estimate the error

$$\| x_{k+1} - x^* \| \leq \frac{\| x_{k+1} - x_k \|}{1 - \hat{\beta}_{k,L}}$$

  and stop only if

$$\| x_{k+1} - x_k \| \leq \varepsilon (1 - \hat{\beta}_{k,L}).$$

* A less stringent alternative would be a p-norm

$$\hat{\beta}_{k,L} = \left(\frac{1}{L} \sum_{j=1}^{L} \left(\frac{\| x_{k-j} - x_{k-j+1} \|}{\| x_{k-j-1} - x_{k-j} \|} \right)^p \right)^{1/p}$$

* $p = \infty$ in the p-norm definition is the same as the max definition.

– Conclusion:

* There is no fool-proof, general method
* Heuristic rules generally do well when carefully implemented using a consistent theory of the rate of convergence

Evaluating the Errors in the Final Result

- When we have completed a computation, we
  - Hope that error is small – difficult to verify
  - Hope that error is small in terms of *economic significance* – more feasible
  - Need to choose ε to accomplish this.
- Error Bounds
  - Sometimes, we can put a bound on the actual error, $\| x^* - \hat{x} \|$; called *forward error analysis*.
  - Usually difficult to determine $\| x^* - \hat{x} \|$ with useful precision
    - Error bounds tend to be very conservative, producing, at best, information about the order of magnitude of the error.
    - Error bounds often need information about the true solution, which is not available, and must also be approximated.
  - Forward error analysis is rarely available (dynamic programming is unusual).

- Error Evaluation: Compute and Verify
  - Use numerical solution to generate information about its quality
  - Consider solving $f(x) = 0$ for some function f.
    * A numerical solution, $\hat{x}$, will generally not satisfy $f(x) = 0$ exactly.
    * Use $f(\hat{x})$, or some related $g(\hat{x})$, to measure importance of error if we accept $\hat{x}$.
  - *compute and verify*
    * first, *compute* an approximation
    * second, *verify* that it is an *acceptable* approximation according to some economically meaningful criteria.

- Consider $f(x) = x^2 - 2 = 0$.
  * A three-digit machine would produce $\hat{x} = 1.41$.
  * We compute (on the three-digit machine) $f(1.41) = -.01$.
  * $f(1.41) = -.01$ may tell us that $\hat{x} = 1.41$ is an acceptable approximation
  * The value $f(\hat{x})$ can be a useful index of acceptability in our economic problems, *but only if it is formulated correctly*
- Let $E(p) = D(p) - S(p)$ be an excess demand function
  * Suppose numerical solution $\hat{p}$ to $E(p) = 0$ implies $E(\hat{p}) = 10.0$.
  * $\hat{p}$ is acceptable depending on $D(\hat{p})$ and $S(\hat{p})$.
    · If $D(\hat{p}) = 10^5$, then $E(\hat{p})$ is 10^{-4} of $D(\hat{p})$ – looks good
    · If $D(\hat{p}) = 10$, then $E(\hat{p})$ equals $D(\hat{p})$ – looks bad!

- In general,
  - *Compute* a candidate solution $\hat{x}$ to $f(x) = 0$.
  - Then *verify* that $\hat{x}$ is acceptable by computing $g(\hat{x})$ where
    - g is function(s) with same zeros as f.
    - g is unit-free
    - g expresses importance of error.
  - In excess demand example,
    - solve $E(p) = 0$
    - but compute $g(\hat{p}) \equiv S(\hat{p})/D(\hat{p}) - 1$ to check $\hat{p}$.
  - In economic, $g(\hat{x})$ expresses quantities like
    - measures of agents' optimization errors
    - "leakage" between demand and supply.
- Compute and verify is always possible

- Backward error analysis
  - Find a problem, $\hat{f}(x) = 0$, such that $\hat{x}$ is exact solution
  - If $\hat{f}(.) \doteq f(.)$, then accept $\hat{x}$ as an approximation to $f(x) = 0$.
  - For example, is $x = 1.41$ is an acceptable solution to $x^2 - 2 = 0$
    * $x = 1.41$ is solution to $x^2 - 1.9881 = 0$.
    * If $x^2 - 1.9881 = 0$ is "close enough" to $x^2 - 2 = 0$, then accept $x = 1.41$ as solution.
- Multiplicity:
  - There are many $\hat{x}$ that satisfy stopping rules and error analysis.
  - Existence of multiple acceptable equilibria makes it difficult to make precise statements (e.g., comparative statics) about equilibrium.
  - However, we could usually run some diagnostics to estimate the size of the set of acceptable solutions.
  - Two ideas:
    * For any guess $\hat{x}$, do random sampling of x near $\hat{x}$ to see how many nearby points satisfy acceptance criterion.
    * Restart algorithm from many initial guesses to see if you get values for $\hat{x}$ that are not close to each other.

- General Philosophy
  - Any economic model approximates reality
  - A good numerical approximation is as useful as exact solution.
  - But, we should always do some error analysis

Computational Complexity of an Algorithm

- Measured by relation between accuracy and computational effort.
  - Let ε denote the error
  - N: computational effort (flops, iterates, ..) to reduce error to ε
  - Examine $N(\varepsilon)$ for small ε, or its inverse, $\varepsilon(N)$ for large N.
  - If iterative method converges linearly at rate β and N is the number of iterations, then $\varepsilon(N) \sim \beta^N$ and $N(\varepsilon) \sim (\log \varepsilon)(\log \beta)^{-1}$.
  - If an algorithm obeys the convergence rule

$$\lim_{\varepsilon \to 0} \frac{N(\varepsilon)}{\varepsilon^{-p}} = \lim_{\varepsilon \to 0} \varepsilon^p N(\varepsilon) = a < \infty$$

    then we need $a\varepsilon^{-p}$ operations to bring error down to ε.
  - Asymptotic ranking depends on p, not a

- Asymptotic results are not necessarily relevant
  - Suppose algorithm A uses $a\varepsilon^{-p}$ operations and B uses $b\varepsilon^{-q}$ operations
    - Algorithm A is asymptotically more efficient if $q > p$.
    - Algorithm A is better with target ε only if $a\varepsilon^{-p} < b\varepsilon^{-q}$, i.e.

$$\varepsilon < \varepsilon^* \equiv (b/a)^{1/(q-p)}$$

    - E.g., if $q = 2, p = 1, b = 1$, and $a = 1000$, then $\varepsilon^* = 0.001$.
  - Asymptotic superiority may imply superiority only for very small ε.
- Know many algorithms since best choice depends on accuracy target.

Types of processes

- Serial processing
  - One action at a time
  - Each action potentially uses any previous computation
- Parallel processing: multiple simultaneous actions
  - Parallel or distributed processing uses many processors
  - Must manage communication among independent processes
  - Parallel processing is present in modern processors; e.g., pipelining
- This course will mainly focus on serial processes and algorithms, but will discuss parallel algorithms that can be implemented easily.