Linear Equations

Econ 5170

Computational Methods in Economics

2020-2021 Spring

Outline

1. Elementary Concepts of Numerical Analysis
   - Computer arithmetic
   - Error analysis
2. Linear Equations
   - Direct methods
   - Iterative methods
   - Acceleration and stabilization methods

Computer Arithmetic

- Unlike pure mathematics, computer arithmetic has finite precision and is limited by time and space.
- Real numbers are represented as floating point numbers of the form

$$\pm d_0.d_1 d_2 \ldots d_{p-1} \times \beta^e$$

where $d_0.d_1 d_2 \ldots d_{p-1}$ is the significand, β is the base, e is the exponent, and p is the precision.

Computer Arithmetic

- Machine epsilon: Smallest quantity ϵ such that $1 - \epsilon$ and $1 + \epsilon$ are both different from one.
  Matlab: $eps = 2.2204e - 016$.
- Machine infinity: Largest quantity that can be represented. Overflow occurs if an operation produces a larger quantity.
  Matlab: $realmax = 1.7977e + 308$.
- Machine zero: Any quantity that cannot be distinguished from zero. Underflow occurs if an operation on nonzero quantities produces a smaller quantity.
  Matlab: $realmin = 2.2251e - 308$.

Computer Arithmetic

- A computer can only execute the basic arithmetic operations of addition, subtraction, multiplication, and division. Everything else is approximated.
- Relative speeds:

operation	speed relative to addition
subtraction	1.03
multiplication	1.03
division	1.06
exponentiation	5.09
sine function	4.20

Computer Arithmetic: Efficient Polynomial Evaluation

Computing $\sum_{k=0}^{n} a_k x^k$.

- Direct method 1: compute the various powers of x, x^2, x^3, etc, then multiply each a_k by x^k, and finally add the terms.
- Direct method 2: replace the expensive exponentiations with multiplications; compute x^2 by computing xx, then compute $x^3 = (xx)x$.

Computer Arithmetic: Efficient Polynomial Evaluation

Computing $\sum_{k=0}^{n} a_k x^k$.

- Direct method 1: compute the various powers of x, x^2, x^3, etc, then multiply each a_k by x^k, and finally add the terms.
- Direct method 2: replace the expensive exponentiations with multiplications; compute x^2 by computing xx, then compute $x^3 = (xx)x$.
- Horner's method:

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 = a_0 + x(a_1 + x(a_2 + x \cdot a_3))$$

	Additions	Multiplications	Exponentiations
Direct method 1	n	n	n-1
Direct method 2	n	2n-1	0
Horner's method	n	n	0

Computer Arithmetic: Efficient Polynomial Evaluation

- Define a one-dimensional array $A(\cdot)$ that stores the a_k coefficients: let $A(k+1) = a_k$ for $k = 0, 1, ..., N$.
- Write a program to implement Horner's method

Analytic Derivatives

The derivatives of $f(x, y, z) = (x^\alpha + y^\alpha + z^\alpha)^\gamma$

- Direct approach: Calculate $\gamma\alpha x^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}$, $\gamma\alpha y^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}$, $\gamma\alpha z^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}$.
- Efficient approach: store the values of $x^\alpha, y^\alpha, z^\alpha, x^\alpha + y^\alpha + z^\alpha$.

$$f_x = (x^\alpha + y^\alpha + z^\alpha)^{\gamma-1} \cdot \gamma\alpha \cdot x^\alpha / x$$

```
XALP = X ^ ALPHA; YALP = Y ^ ALPHA; ZALP = Z ^ ALPHA
SUM = XALP + YALP + ZALP
F=SUM ^ (GAMMA-1)
COM=GAMMA*ALPHA*F
FX=COM*XALP/X; FY=COM*YALP/Y; FZ=COM*ZALP/Z
```

Computer Arithmetic: Finite Differences

When the analytic derivatives are absent or too time-consuming, we turn to finite differences.

- One-sided finite difference

$$f'(x) \doteq \frac{f(x+h) - f(x)}{h}$$

  where $h = \min\{\epsilon|x|, |x|\}$ is the step size and ϵ is chosen appropriately, usually on the order of 10^{-6}.
  - We want h to be small relative to x
  - We want h to stay away from zero to keep the division and differencing well-behaved.
- If $f : R^n \to R$, then

$$\frac{\partial f}{\partial x_i} \doteq \frac{f(x_1, ..., x_i + h_i, ..., x_n) - f(x_1, ..., x_i, ..., x_n)}{h_i}$$

Computer Arithmetic: Finite Differences

- Cross partials are approximated by

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \doteq \frac{1}{h_j}\Big(\frac{f(\dots, x_i + h_i, \dots, x_j + h_j, \dots) - f(\dots, x_i, \dots, x_j + h_j, \dots)}{h_i} - \frac{f(\dots, x_i + h_i, \dots, x_j, \dots) - f(\dots, x_i, \dots, x_j, \dots)}{h_i}\Big)$$

- The second partials are approximated by

$$\frac{\partial^2 f}{\partial x_i^2} \doteq \frac{f(\dots, x_i + h_i, \dots) - 2f(\dots, x_i, \dots) + f(\dots, x_i - h_i, \dots)}{h_i^2}$$

Direct versus Iterative Methods

- Direct methods: algorithms which, in the absence of round-off error, give the exact answer in a predetermined finite number of steps.
    - Pros: take a fixed amount of time and produce answers of fixed precision
    - Cons: may not exist, or require too much space or time
- Iterative methods:

$$x^{k+1} = g^{k+1}(x^k, x^{k-1}, ...)$$

- Whether the sequence x^k converges to x^*, and if convergent how fast it converges
- Must terminate the sequence at some finite point
- We can control the quality of the result

Sources of Error

- Rounding: arise from the fact that the only thing computers can do correctly is integer arithmetic
  - Example: Consider the decimal number 0.1. If $\beta = 10$ and $p = 3$, then 1.00×10^{-1} is exact. If $\beta = 2$ and $p = 24$, then
$$1.10011001100110011001101 \times 2^{-4}$$
is not exact.
  - Increasing the number of bits used to present a number is the only way to reduce rounding errors.

Sources of Error

- Mathematical truncation
  - Many mathematical objects and procedures are defined as the limit of an infinite process, such as an iterative algorithm.
  - For example, the exponential function is defined as

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

On some computers it becomes $\sum_{n=0}^{N} \frac{x^n}{n!}$ for some finite N.

Error Analysis

Error Propagation

- Once errors arise in a calculation, they can interact to reduce the accuracy of the final result even further.
- For example, solve the quadratic equation $x^2 - 26x + 1 = 0$. The solution is $x^* = 13 - \sqrt{168} = 0.0385186\ldots$.
  - Compute this number with a five-digit computer. $\sqrt{168} = 12.961$. The result is

    $$\hat{x_1} = 13 - 12.961 = 0.039$$

    The relative error is more than 1%.
  - A better approach is

    $$\hat{x_2} = 13 - \sqrt{168} = \frac{1}{13 + \sqrt{168}} \doteq \frac{1}{25.961} \doteq 0.038519$$

    The relative error is 10^{-5}.

Error Analysis

Reduce the propagation of errors

- Avoid unnecessary subtractions of numbers of similar magnitude.
- When adding a long list of numbers, first add the small numbers and then add the result to the larger numbers.

Error Analysis

Rates of Convergence

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

$$\lim_{k\to\infty} \frac{||x^{k+1} - x^*||}{||x^k - x^*||^q} < \infty$$

- If the above is true for $q = 2$, we say that x^k converges quadratically.
- If

$$\lim_{k\to\infty} \frac{||x^{k+1} - x^*||}{||x^k - x^*||} \le \beta < 1$$

  we say that x^k converges linearly at rate β.
  If $\beta = 0$, x^k is said to converge superlinearly.

Stopping Rules

- Stop and accept x_{k+1} if

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

  This allows us to stop the sequence if it appears that the changes are small or if the limit appears to be close to zero.

- If we know a sequence is linearly convergent at rate $\beta < 1$. We have

$$||x^{k+1} - x^*|| \leq \beta ||x^k - x^*||$$
$$||x^k - x^*|| \leq ||x^k - x^{k+1}||/(1 - \beta)$$

  Stop and accept x_{k+1} if

$$||x^k - x^{k+1}|| \leq \epsilon(1 - \beta)$$

  the error is bounded by $||x^k - x^*|| \leq \epsilon$.

Error Analysis

Stopping Rule

- For example, consider the scalar sequence

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

The limit of x_k is infinite, but any particular x_k is finite.

  - First stopping rule: If $\epsilon = 0.001$, it will end at $k = 9330$ where $x_k = 9.71827$.
  - Second stopping rule: never conclude that the sequence converges.

Error Analysis

Compute and Verify

- Consider the problem of solving $f(x) = 0$. The exact solution is x^*, our approximate solution is $\hat{x}$.
- Forward error analysis: How far is $\hat{x}$ from x^*?
- Backward error analysis: Construct a similar problem $\hat{f}$ such that $\hat{f}(\hat{x}) = 0$. How far is $\hat{f}$ from f?
- Compute and verify: how far is $f(\hat{x})$ from its target value of 0?

Direct Methods: Backsubstitution

- Consider the system of linear equations $Ax = b$, where A is an $n \times n$ matrix, and b is an $n \times 1$ vector.
- Backsubstitution: Suppose A is lower triangular

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

  Then

$$x_1 = \frac{b_1}{a_{11}}$$

$$x_k = \frac{b_k - \sum_{j=1}^{k-1} a_{kj} x_j}{a_{kk}}, \quad k = 2, 3, ..., n$$

- If A is upper triangular, we can similarly solve $Ax = b$ beginning with $x_n = b_n / a_{nn}$.

Direct Methods: LU Decomposition

- Factor A into the product of two triangular matrices, $A = LU$
  - L is lower triangular and U is upper triangular.

$$\begin{aligned} Ax &= b \\ \Rightarrow LUx &= b \\ \Rightarrow Lz &= b \text{ and } Ux = z \end{aligned}$$

- Solve for z in $Lz = b$ by backsubstitution
- Solve for x in $Ux = z$ by backsubstitution
- Gaussian elimination produces such an LU decomposition for any nonsingular A.
  Matlab: $[L, U] = lu(A)$.

Direct Methods: QR Factorization

- If A is nonsingular, then decompose $A = QR$, where Q is orthogonal ($Q'Q$ is a diagonal matrix) and R is upper triangular.
  Matlab: $[Q, R] = qr(A)$.

$$\begin{aligned} Ax &= b \\ \Rightarrow Q'Ax &= Q'b \\ \Rightarrow Q'QRx &= Q'b \\ \Rightarrow DRx &= Q'b \end{aligned}$$

- D is a diagonal matrix. DR is upper triangular.
- Compute x by applying backsubstitution.

Direct Methods: Cholesky Factorization

- The LU and QR decomposition can be applied to any nonsingular matrix
- Cholesky factorization can be used for symmetric positive definite matrices.
- Cholesky decomposition: $A = LL'$, where L is a lower triangular matrix. L is the "square root" of A.
  Matlab: $C = chol(A)$
- A special case of LU decomposition, but only has half the cost of LU decomposition.

Iterative Methods

- Decomposition methods for linear equations are direct methods of solution
  - Can be very costly for large systems, since the time requirement are order n^3 and the space requirements are order n^2
- Iterative methods can economize on space and provide good answers in reasonable time
  - Gauss-Jacobi Algorithm
  - Gauss-Seidel Algorithm

Iterative Methods: Fixed-Point Iteration

- Rewrite the problem as a fixed-point problem and repeatedly iterate the fixed-point mapping
- For the problem $Ax = b$, define $G(x) = Ax - b + x$

$$x^{k+1} = G(x^k) = (A + I)x^k - b$$

- It will converge only if all the eigenvalues of $A + I$ have modulus less than 1.

Iterative Methods: Gauss-Jacobi Algorithm

- Consider the equation from the first row of $Ax = b$:

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$

We can solve for x_1 in terms of $(x_2, .., x_n)$ if $a_{11} \neq 0$

$$x_1 = a_{11}^{-1}(b_1 - a_{12}x_2 - ... - a_{1n}x_n)$$

- In general, if $a_{ii} \neq 0$, we have

$$x_i = \frac{1}{a_{ii}}\{b_i - \sum_{j \neq i} a_{ij}x_j\}$$

- The GJ Iteration

$$x_i^{k+1} = \frac{1}{a_{ii}}\{b_i - \sum_{j \neq i} a_{ij}x_j^k\}$$

Iterative Methods: Gauss-Seidel Algorithm

- In the GJ method, we use a new guess for x_i, x_i^{k+1}, only after we have computed the entire vector of new values
- If x_i^{k+1} is a better estimate of x_i^* than x_i^k, using x_i^{k+1} to compute x_{i+1}^{k+1} would seem to be better than using x_i^k
- The idea of GS method is to use a new approximation of x_i^* as soon as it is available.

$$x_1^{k+1} = a_{11}^{-1}(b_1 - a_{12}x_2^k - ... - a_{1n}x_n^k)$$
$$x_2^{k+1} = a_{22}^{-1}(b_2 - a_{21}x_1^{k+1} - a_{23}x_3^k - ... - a_{2n}x_n^k)$$

- The GS iteration

$$x_i^{k+1} = \frac{1}{a_{ii}}\{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k\}$$

Iterative Methods: Example

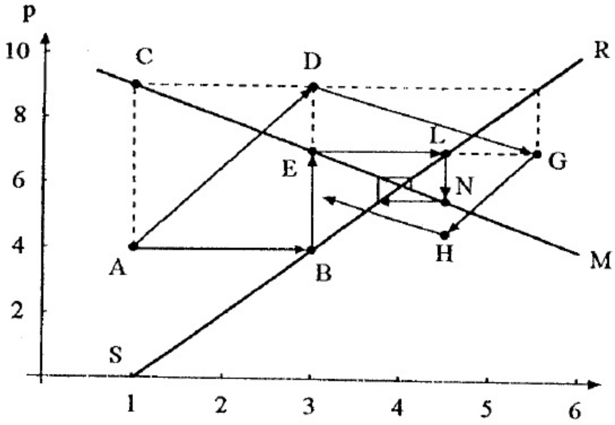
- Inverse demand equation $p = 10 - q$ and the supply curve $q = p/2 + 1$.
- Initial guess is $p = 4$ and $q = 1$.
- Gauss-Jacobi iteration:

$$q_{n+1} = 1 + \frac{1}{2}p_n$$
$$p_{n+1} = 10 - q_n$$

- Gauss-Seidel iteration:

$$q_{n+1} = 1 + \frac{1}{2}p_n$$
$$p_{n+1} = 10 - q_{n+1}$$

Iterative Methods: Example



Iterative Methods: Example

Table 3.2
Gaussian methods for (3.6.6)

Iteration	Gauss-Jacobi		Gauss-Seidel	
n	p_n	q_n	p_n	q_n
0	4	1	4	1
1	9	3	7	3
2	7	5.5	5.5	4.5
3	4.5	4.5	6.25	3.75
4	5.5	3.25	5.875	4.125
5	6.75	3.75	6.0625	3.9375
7	5.625	4.125	6.0156	3.9844
10	6.0625	4.0938	5.9980	4.0020
15	5.9766	4.0078	6.0001	3.9999
20	5.9980	3.9971	6.0000	4.0000

Exercise

- Write a function to implement the Gauss-Jacobi iteration, set $p_0 = 4, q_0 = 1$, and the number of iterations $N = 5$. Try different values of N.
- Write a function to implement the Gauss-Seidel iteration, set $p_0 = 4, q_0 = 1$, and the number of iterations $N = 5$. Try different values of N.

Iterative Methods: Operator Splitting Approach

- Write A as $A = N - P$.
- Define the iteration

$$Nx^{m+1} = b + Px^m$$

If N is invertible, can also be written

$$x^{m+1} = N^{-1}(b + Px^m)$$

- GJ iteration

$$N = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}, P = -\begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{pmatrix}$$

Iterative Methods: Operator Splitting Approach

- GS iteration

$$N = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, P = -\begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

- General iterative scheme for $Ax = b$
  1. Find an N with an easily computed N^{-1} and split the operator $A = N - P$.
  2. Construct the iterative scheme $x^{m+1} = N^{-1}(b + Px^m)$.
  3. Find acceleration scheme to ensure and/or speed up convergence.
  4. Find adaptive scheme to learn acceleration parameters.

Iterative Methods: Convergence

- Converge if $\rho(N^{-1}P) < 1$, where $\rho()$ is the spectral radius (largest eigenvalue in absolute value) of the matrix.
- If A is diagonally dominant, both Gauss-Jacobi and Gauss-Seidel iteration schemes are convergent for all initial guesses.

$$\sum_{j \neq i} |a_{ij}| < |a_{ii}|, \quad i = 1, ..., n$$

- GJ and GS methods are at best linearly convergent and the rate of convergence is given by $\rho(N^{-1}P)$.

Acceleration and Stabilization Methods

- Consider the problem $Ax = b$. Define $G = I + A$. The iteration

$$x^{k+1} = Gx^k - b$$

  It only converges if $\rho(G) < 1$. But if $\rho(G)$ is close to 1, converge will be slow.

- Consider next the iteration

$$\begin{aligned} x^{k+1} &= w(Gx^k - b) + (1 - w)x^k \\ &\equiv G_{|w|}x^k - wb \end{aligned}$$

Acceleration and Stabilization Methods

- When $w > 1$, it's called extrapolation. If it converges, then $Gx^k - b - x^k$ is a good direction to move, and perhaps it would be better to move to a point in this direction beyond $Gx^k - b$ and converge even faster.
- When $w < 1$, it's called dampening. If it is unstable, it could be that the direction $Gx^k - b - x^k$ is a good one, but that the point $Gx^k - b$ overshoots the solution.

Acceleration and Stabilization Methods

- Define m as the minimum element of $\sigma(G)$ and M the maximum. $\sigma()$ is the spectrum of the matrix (set of eigenvalues).
- The optimal w will be

$$w^* = \frac{2}{2 - m - M}$$

- The new spectral radius is

$$\rho(G_{|w^*|}) = \left| \frac{M - m}{2 - M - m} \right| \tag{1}$$

- If $M < 1$, then $\rho(G_{|w^*|}) < 1$, no matter what m is. So we can always find an w^* that produces a stable iteration.
- If $M > 1$ and $m < -1$, (1) fails.

Acceleration and Stabilization Methods: SOR

Successive Overrelaxation (SOR) Method

$$x_i^{k+1} = w(\frac{1}{a_{ii}})[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^{k}] + (1-w)x_i^k$$

- The ith component of the $k+1$ iterate is a linear combination, parameterized by w, of the Gauss-Seidel value and the kth iterate.
- Write $A = L + D + U$, where L, D, and U consists of the elements of A below, on, and above the diagonal, respectively. Then

$$x_i^{k+1} = N_w^{-1}(P_w x^k + wb)$$

where $N_w = D + wL$ and $P_w = (1-w)D - wU$.

Stabilization Example

- Inverse demand function is $p = 21 - q$ and supply function is $q = \frac{p}{2} - 3$.
- GS iteration:

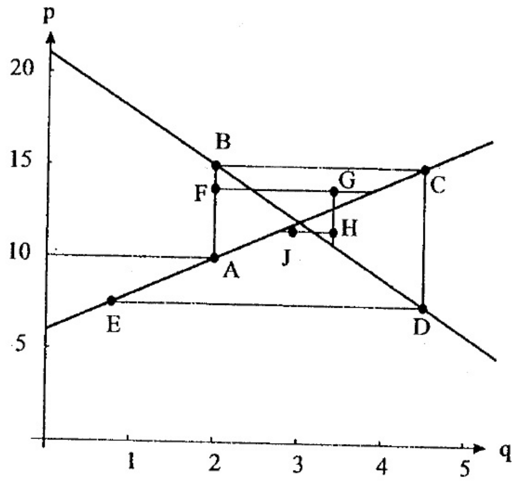
$$p^{k+1} = 21 - 3q^k$$
$$q^{k+1} = \frac{p^{k+1}}{2} - 3$$

- SOR:

$$p^{k+1} = w(21 - 3q^k) + (1 - w)p^k$$
$$q^{k+1} = w(\frac{p^{k+1}}{2} - 3) + (1 - w)q^k$$

where $w = 0.75$

Stabilization Example



Acceleration Example

- Reaction curves of two price-setting duopolists: Firm 1's reaction function is $p_1 = 1 + 0.75p_2$ and firm 2's reaction function is $p_2 = 2 + 0.8p_1$.
- GS iteration:

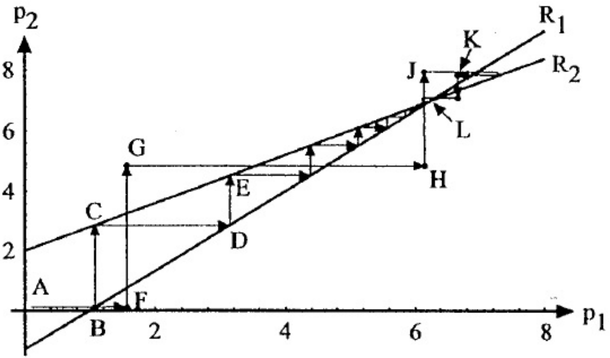
$$
\begin{aligned}
p_1^{k+1} &= 1 + 0.75p_2^k \\
p_2^{k+1} &= 2 + 0.8p_1^{k+1}
\end{aligned}
$$

- SOR:

$$
\begin{aligned}
p_1^{k+1} &= w(1 + 0.75p_2^k) + (1 - w)p_1^k \\
p_2^{k+1} &= w(2 + 0.8p_1^{k+1}) + (1 - w)p_2^k
\end{aligned}
$$

where $w = 1.5$

Acceleration Example



Exercise

- Write a function to implement the GS iteration and SOR algorithms for the supply-demand problem, set $p_0 = 10, q_0 = 0, w = 0.75$, and the number of iterations $N = 5$. Try different values of p_0, q_0, w.
- Write a function to implement the GS iteration and SOR algorithms for the duopolist problem, set $p_{1,0} = 0, p_{2,0} = 0, w = 1.5, N = 5$. Try different values of $p_{1,0}, p_{2,0}, w$.

Nonlinear Equations

- Many concepts and methods carry over from linear to nonlinear equations.
- Idea: $f(x) = 0$ is approximated by $f(x_0) + f_x(x_0)(x - x_0) = 0$
- Iterative methods: GJ, GS, convergence (local instead of global), acceleration and stabilization methods.