5. Systems of equations

- system of equations
- Cramer rule for linear equations
- Gauss-Seidel and Jacobi methods for linear equations
- fixed point iteration for nonlinear equations
- Newton-Raphson for nonlinear equations

System of nonlinear equations

n nonlinear equations in n variables:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

- $f_i : \mathbb{R}^n \to \mathbb{R}$ is a function that takes n variables and outputs a scalar
- $f_i(x)$ is ith *residual*
- the *roots* or *solutions* is the set of x values that make all equations zero
- may have one solution, multiple solutions, or no solution
- in vector notation: $f(x) = 0$ with

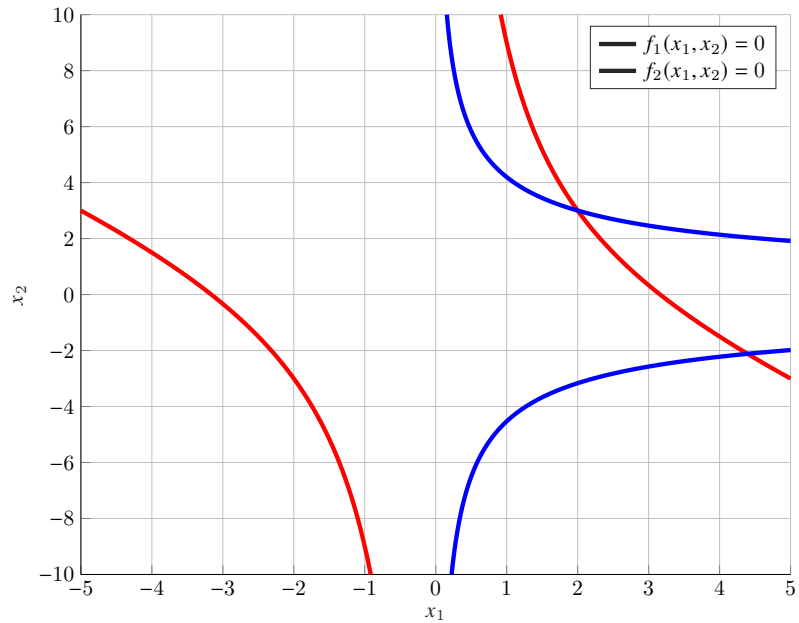
$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad f(x) = \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{bmatrix}$$

Example: nonlinear system

two nonlinear equations with two unknowns:

$$f_1(x_1, x_2) = x_1^2 + x_1 x_2 - 10 = 0$$
$$f_2(x_1, x_2) = x_2 + 3x_1 x_2^2 - 57 = 0$$

solution: values (x_1, x_2) such that both $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$



one solution is $x_1 = 2, x_2 = 3$

Linear equations

an equation in the variables $x_1, \ldots, x_n$ is called *linear* if each side consists of a sum of multiples of x_i, and a constant, *e.g.*,

$$1 + x_2 = x_3 - 2x_1$$

is a linear equation in x_1, x_2, x_3

System of linear equations: m linear equations in n variables $x_1, \ldots, x_n$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

- a_{ij} are the *coefficients*
- b_i are called the *right-hand sides*
- may have no solution, a unique solution, infinitely many solutions

Graphical approach

- we can use graphical approach for small systems $n \leq 3$

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$

- both equations can be solved for x_2:

$$x_2 = -\left(\frac{a_{11}}{a_{12}}\right)x_1 + \frac{b_1}{a_{12}}$$

$$x_2 = -\left(\frac{a_{21}}{a_{22}}\right)x_1 + \frac{b_2}{a_{22}}$$

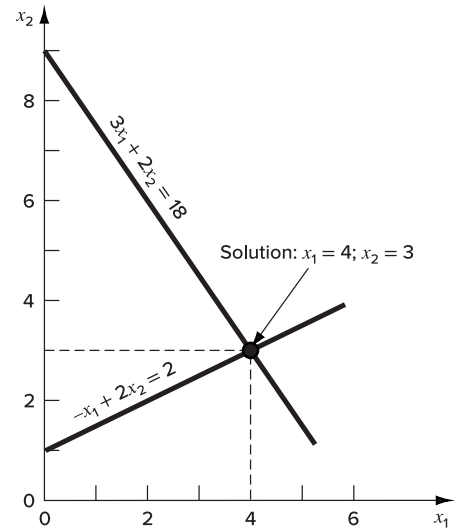
- thus, each equation is now in the form of a straight line:

$$x_2 = (\text{slope})\, x_1 + \text{intercept}$$

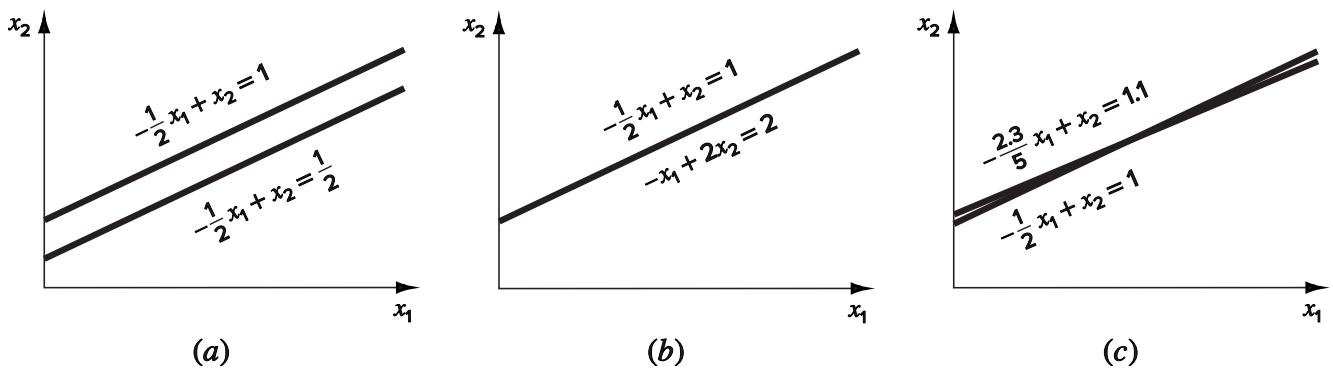
  these lines can be graphed on Cartesian coordinates

- the values of x_1 and x_2 at the intersection of the lines represent the solution

Example: graphical approach



(a) no solution (singular), (b) infinite solutions (also called singular), (c) ill-conditioned system where slopes are so close that the point of intersection is difficult to detect



Linear equation in matrix form

can express linear equations compactly as

$$Ax = b$$

- $A \in \mathbb{R}^{m \times n}$ is the *coefficient matrix* with entries a_{ij}
- $b \in \mathbb{R}^m$ is called the *right-hand side* with entries b_i

Classification of linear equations

- *under-determined* if $m < n$ (more unknowns than equations, A wide)
- *square* if $m = n$ (no. equations equal no. unknowns, A square)
- *over-determined* if $m > n$ (more equations than unknowns, A tall)

Example

two equations in three variables x_1, x_2, x_3 (underdetermined system):

$$1 + x_2 = x_3 - 2x_1, \quad x_3 = x_2 - 2$$

- step 1: rewrite with variables on the l.h.s. side, and constants on the r.h.s. side:

$$\begin{array}{rrrcr} 2x_1 & +x_2 & -x_3 & = & -1 \\ 0x_1 & -x_2 & +x_3 & = & -2 \end{array}$$

  (each row is one equation)

- step 2: rewrite equations as a single matrix equation:

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

  - ith row of A gives the coefficients of the ith equation
  - jth column of A gives the coefficients of x_j in the equations
  - ith entry of b gives the constant in the ith equation

Solving square linear equations

suppose we have n linear equations in n variables $x_1, \ldots, x_n$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

- compact form: $Ax = b$, where A is an $n \times n$ matrix, and b is an n-vector
- suppose A is invertible, *i.e.*, its inverse A^{-1} exists
- multiply both sides of $Ax = b$ on the left by A^{-1}:

$$A^{-1}(Ax) = A^{-1}b$$

- lefthand side simplifies to $A^{-1}Ax = Ix = x$, so the solution is

$$x = A^{-1}b$$

Linear equations with non-inveretible matrix

when A isn't invertible, *i.e.*, inverse doesn't exist

- one or more of the equations is redundant (*i.e.*, can be obtained from the others)
- the equations are inconsistent or contradictory

in practice: A isn't invertible means

- you've set up the wrong equations
- or don't have enough of them

Solving linear equations in practice

- to solve $Ax = b$ (*i.e.*, compute $x = A^{-1}b$) by computer, we don't compute A^{-1}, then multiply it by b (but that would work!)
- practical methods compute $x = A^{-1}b$ directly, via specialized methods
  - Gaussian elimination or LU factorization
  - QR factorization
  - Jacobi and Gauss-Seidel methods
  - $\cdots$
- standard methods, that work for any (invertible) A, require about n^3 arithmetic operations to compute $x = A^{-1}b$
- but modern computers are very fast, so solving say a set of 1000 equations in 1000 variables takes only a second or so, even on a small computer
- in MATLAB, `x = A\b` solves $Ax = b$ if a solution exists
  - if no solution exists, it still returns a vector, which is *not* a solution

Outline

- system of equations
- **Cramer rule for linear equations**
- Gauss-Seidel and Jacobi methods for linear equations
- fixed point iteration for nonlinear equations
- Newton-Raphson for nonlinear equations

Matrix determinant

if A is an $n \times n$ matrix, then the ij**th submatrix** of A, denoted by A_{ij}, is the $(m-1) \times (m-1)$ obtained by deleting row i and column j of A; for example,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \qquad A_{12} = \begin{bmatrix} 4 & 6 \\ 7 & 9 \end{bmatrix}, \quad A_{32} = \begin{bmatrix} 1 & 3 \\ 4 & 6 \end{bmatrix}$$

Determinant: pick any value of $i = 1, 2, \ldots, n$ and compute

$$\det(A) = |A| = \sum_{j=1}^{n} (-1)^{i+j} \det(A_{ij}) a_{ij}$$

- $\det(A_{ij})$ is called the *minor* of element a_{ij}
- $(-1)^{i+j} \det(A_{ij})$ is called the *cofactor* of element a_{ij}

Examples

a) for a scalar matrix $A = [a_{11}]$, we have $\det(A) = a_{11}$

b) for a 2×2 matrix, the determinant is

$$\det(A) = \det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11}a_{22} - a_{21}a_{12}$$

c) for the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

- we have for $i = 1$

$$A_{11} = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}, \quad A_{12} = \begin{bmatrix} 4 & 6 \\ 7 & 9 \end{bmatrix}, \quad A_{13} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$$

- thus, the determinant is

$$\begin{aligned} \det(A) &= (-1)^2 a_{11} \det(A_{11}) + (-1)^3 a_{12} \det(A_{12}) + (-1)^4 a_{13} \det(A_{13}) \\ &= a_{11} \det(A_{11}) - a_{12} \det(A_{12}) + a_{13} \det(A_{13}) \\ &= 1(-3) - 2(-6) + 3(-3) = 0 \end{aligned}$$

Cramer's rule

if $\det(A) \neq 0$, then the square linear system $Ax = b$ has a unique solution

$$x = A^{-1}b$$

we can find the solution using *Cramer's formula:*

$$x_k = \frac{|D_k|}{|A|}, \quad k = 1, 2, \ldots, n$$

- D_k is the matrix obtained replacing the kth column of A by b
- from Cramer's formula (with some algebra), we have

$$A^{-1} = \frac{1}{\det A} \underbrace{\begin{bmatrix} \det A_{11} & \det A_{21} & \cdots & \det A_{n1} \\ \det A_{12} & \det A_{22} & \cdots & \det A_{n1} \\ \vdots & \vdots & \cdots & \vdots \\ \det A_{1n} & \det A_{2n} & \cdots & \det A_{nn} \end{bmatrix}}_{\text{adj } A}$$

- rarely used (*e.g.*, for small systems)

Example: Cramer's rule

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$
$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

the determinant can be written as

$$|A| = \begin{vmatrix} 0.3 & 0.52 & 1 \\ 0.5 & 1 & 1.9 \\ 0.1 & 0.3 & 0.5 \end{vmatrix}$$

the minors are:

$$A_{11} = \begin{vmatrix} 1 & 1.9 \\ 0.3 & 0.5 \end{vmatrix} = 1(0.5) - 1.9(0.3) = -0.07$$

$$A_{12} = \begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} = 0.5(0.5) - 1.9(0.1) = 0.06$$

$$A_{13} = \begin{vmatrix} 0.5 & 1 \\ 0.1 & 0.3 \end{vmatrix} = 0.5(0.3) - 1(0.1) = 0.05$$

Example: Cramer's rule

$$|A| = 0.3(-0.07) - 0.52(0.06) + 1(0.05) = -0.0022$$

Solution using Cramer's rule

$$x_1 = \frac{\begin{vmatrix} -0.01 & 0.52 & 1 \\ 0.67 & 1 & 1.9 \\ -0.44 & 0.3 & 0.5 \end{vmatrix}}{-0.0022} = \frac{0.03278}{-0.0022} = -14.9$$

$$x_2 = \frac{\begin{vmatrix} 0.3 & -0.01 & 1 \\ 0.5 & 0.67 & 1.9 \\ 0.1 & -0.44 & 0.5 \end{vmatrix}}{-0.0022} = \frac{0.0649}{-0.0022} = -29.5$$

$$x_3 = \frac{\begin{vmatrix} 0.3 & 0.52 & -0.01 \\ 0.5 & 1 & 0.67 \\ 0.1 & 0.3 & -0.44 \end{vmatrix}}{-0.0022} = \frac{-0.04356}{-0.0022} = 19.8$$

Outline

- system of equations
- Cramer rule for linear equations
- **Gauss-Seidel and Jacobi methods for linear equations**
- fixed point iteration for nonlinear equations
- Newton-Raphson for nonlinear equations

Gauss-Seidel method

a common iterative method for solving

$$Ax = b$$

Gauss-Seidel

- **step 1:** start with an initial guess $x_{2,0}, \ldots, x_{n,0}$
- **step 2:** update

$$x_{1,i} = \frac{b_1 - a_{12}x_{2,i-1} - a_{13}x_{3,i-1} - \cdots - a_{1n}x_{n,i-1}}{a_{11}}$$

$$x_{2,i} = \frac{b_2 - a_{21}x_{1,i} - a_{23}x_{3,i-1} - \cdots - a_{2n}x_{n,i-1}}{a_{22}}$$

$$\vdots$$

$$x_{n,i} = \frac{b_n - a_{n1}x_{1,i} - a_{n2}x_{2,i} - \cdots - a_{n,n-1}x_{n-1,i}}{a_{nn}}$$

- repeat until: $|\varepsilon_{a,j}| = \left|\frac{x_{j,i} - x_{j,i-1}}{x_{j,i}}\right| \times 100\% < \varepsilon_s$ for all entries $j = 1, \ldots, n$

Gauss-Seidel for three equations

for a 3×3 system:

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$
$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}}$$
$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}$$

- start with guesses for x_1, x_2, x_3 (often zero)
- compute a new x_1
- use the new x_1 to compute x_2
- use x_1, x_2 to compute x_3
- repeat until convergence: all

  $|\varepsilon_{a,1}| = \left|\frac{x_{1,i} - x_{1,i-1}}{x_{1,i}}\right| 100\%,\ |\varepsilon_{a,2}| = \left|\frac{x_{2,i} - x_{2,i-1}}{x_{2,i}}\right| 100\%,\ |\varepsilon_{a,3}| = \left|\frac{x_{3,i} - x_{3,i-1}}{x_{3,i}}\right| 100\%$

  less than ε_s

Example: Gauss-Seidel

implement Gauss-Seidel to solve

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

starting with $x_2 = 0, x_3 = 0$ (true solution: $x_1 = 3, x_2 = -2.5, x_3 = 7$)

solve each equation for the diagonal variable:

$$\begin{aligned} x_1 &= \frac{7.85 + 0.1x_2 + 0.2x_3}{3} \\ x_2 &= \frac{-19.3 - 0.1x_1 + 0.3x_3}{7} \\ x_3 &= \frac{71.4 - 0.3x_1 + 0.2x_2}{10} \end{aligned}$$

Example: Gauss-Seidel

First iteration

$$x_1 = \frac{7.85 + 0 + 0}{3} = 2.616667$$

$$x_2 = \frac{-19.3 - 0.1(2.616667) + 0}{7} = -2.794524$$

$$x_3 = \frac{71.4 - 0.3(2.616667) + 0.2(-2.794524)}{10} = 7.005610$$

Second iteration

$$x_1 = \frac{7.85 + 0.1(-2.794524) + 0.2(7.005610)}{3} = 2.990557$$

$$x_2 = \frac{-19.3 - 0.1(2.990557) + 0.3(7.005610)}{7} = -2.499625$$

$$x_3 = \frac{71.4 - 0.3(2.990557) + 0.2(-2.499625)}{10} = 7.000291$$

converging toward:

$$x_1 \to 3, \quad x_2 \to -2.5, \quad x_3 \to 7$$

Example: Gauss-Seidel

approximate relative error:

$$|\varepsilon_{a,1}| = \left|\frac{2.990557 - 2.616667}{2.990557}\right| 100\% = 12.5\%$$

$$|\varepsilon_{a,2}| = 11.8\%, \quad |\varepsilon_{a,3}| = 0.076\%$$

- conservative measure of convergence
- ensures accuracy within specified tolerance ε_s

MATLAB implementation

```
function x = GaussSeidel(A,b,es,maxit)
if nargin<2,error('at least 2 input arguments required'),end
if nargin<4 || isempty(maxit),maxit = 50;end
if nargin<3 || isempty(es),es = 0.00001;end
[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
C = A;
for i = 1:n
C(i,i) = 0;x(i) = 0;
end
x = x';
for i = 1:n
C(i,1:n) = C(i,1:n)/A(i,i);
end
for i = 1:n
d(i) = b(i)/A(i,i);
end
iter = 0;
while (1)
xold = x;
for i = 1:n
x(i) = d(i) - C(i,:)*x;
if x(i) ~= 0
ea(i) = abs((x(i) - xold(i))/x(i)) * 100;
end
end
iter = iter+1;
if max(ea)<=es || iter >= maxit, break, end
end
```

Sufficient conditions for convergence

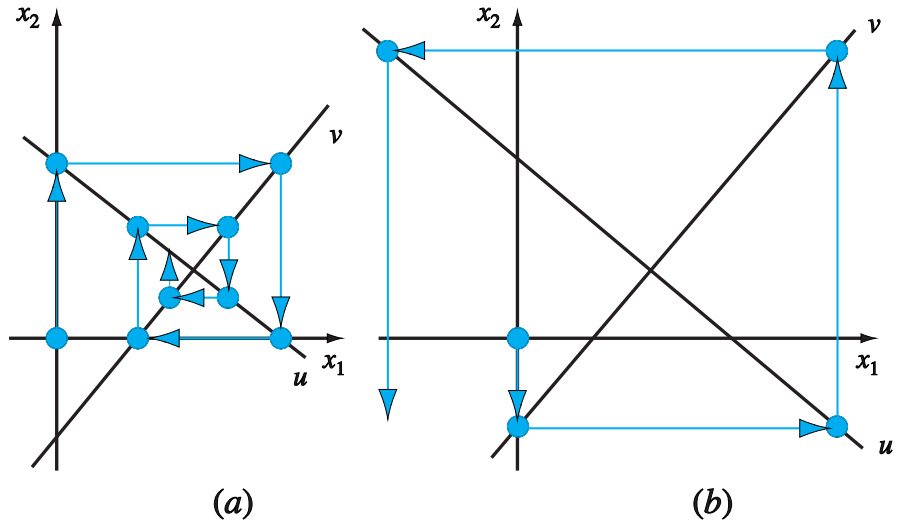
system is **diagonally dominant** if

$$|a_{kk}| > \sum_{\substack{\ell=1 \\ \ell \neq k}}^{n} |a_{k\ell}|, \quad k = 1, 2, \ldots, n$$

- each diagonal element is greater than sum of off-diagonal terms in its row
- diagonal dominance $\Longrightarrow$ guaranteed convergence
- if not satisfied, convergence is not guaranteed (but still possible in some cases)

Graphical illustration

$$u : 11x_1 + 13x_2 = 286, \quad v : 11x_1 - 9x_2 = 99$$



- iteration cobwebs show convergence vs divergence
- same functions plotted; difference arises from the order of implementation
  - (a) update x_2 in equation u first (diagonally dominant)
  - (b) update x_2 in equation v first (not diagonally dominant)
- if diagonal dominance does not hold, divergence can occur

Relaxation to improve convergence

Gauss-Seidel with relaxation

$$x_j^{\text{new}} = \lambda x_j^{\text{new}} + (1 - \lambda) x_j^{\text{old}}, \quad j = 1, \ldots, n$$

- *underrelaxation:* $0 < \lambda < 1$
  - dampens oscillations
  - helps nonconvergent systems converge
- *overrelaxation:* $1 < \lambda < 2$
  - speeds up convergence if system already convergent
  - common in large-scale engineering systems
  - also called *successive* or *simultaneous overrelaxation* (SOR)
- $\lambda = 1$: standard Gauss-Seidel

Example: Gauss-Seidel with relaxation

solve the following system

$$-3x_1 + 12x_2 = 9$$
$$10x_1 - 2x_2 = 8$$

with Gauss-Seidel using overrelaxation ($\lambda = 1.2$) and stopping criterion $\varepsilon_s = 10\%$

Rearrangement (diagonally dominant)

$$x_1 = \frac{8 + 2x_2}{10} = 0.8 + 0.2x_2$$
$$x_2 = \frac{9 + 3x_1}{12} = 0.75 + 0.25x_1$$

Example: Gauss-Seidel with relaxation

First Iteration: initial guesses: $x_1 = x_2 = 0$

- first value

$$x_1 = 0.8 + 0.2(0) = 0.8$$

apply relaxation:

$$x_{1,r} = 1.2(0.8) - 0.2(0) = 0.96$$

- now compute x_2 using relaxed value $x_{1,r}$:

$$x_2 = 0.75 + 0.25(0.96) = 0.99$$

apply relaxation:

$$x_{2,r} = 1.2(0.99) - 0.2(0) = 1.188$$

errors are initially 100% since we started from zero

Example: Gauss-Seidel with relaxation

Second Iteration

- using updated values from iteration 1:

$$x_1 = 0.8 + 0.2(1.188) = 1.0376$$

relaxed value:

$$x_{1,r} = 1.2(1.0376) - 0.2(0.96) = 1.05312$$

approximate error:

$$\varepsilon_{a,1} = \left|\frac{1.05312 - 0.96}{1.05312}\right| \times 100\% = 8.84\%$$

- next variable:

$$x_2 = 0.75 + 0.25(1.05312) = 1.01328$$

relaxed value:

$$x_{2,r} = 1.2(1.01328) - 0.2(1.188) = 0.978336$$

approximate error:

$$\varepsilon_{a,2} = \left|\frac{0.978336 - 1.188}{0.978336}\right| \times 100\% = 21.43\%$$

Example: Gauss-Seidel with relaxation

Stopping criterion

- at the end of iteration 2:

$$\varepsilon_{a,1} = 8.84\% < 10\% \quad \Rightarrow \quad x_1 \text{ satisfies criterion}$$

and

$$\varepsilon_{a,2} = 21.43\% > 10\% \quad \Rightarrow \quad x_2 \text{ does not satisfy criterion}$$

- thus, further iterations are required until stopping criteria satisfied
- overrelaxation $(\lambda = 1.2)$ accelerates convergence when system is convergent

Jacobi iteration

- computes new values using only the previous iteration's estimates
- all updates occur simultaneously after each iteration

Jacobi iteration

- **step 1:** start with an initial guess $x_{1,0}, x_{2,0}, \ldots, x_{n,0}$
- **step 2:** update

$$x_{1,i} = \frac{b_1 - a_{12}x_{2,i-1} - a_{13}x_{3,i-1} - \cdots - a_{1n}x_{n,i-1}}{a_{11}}$$

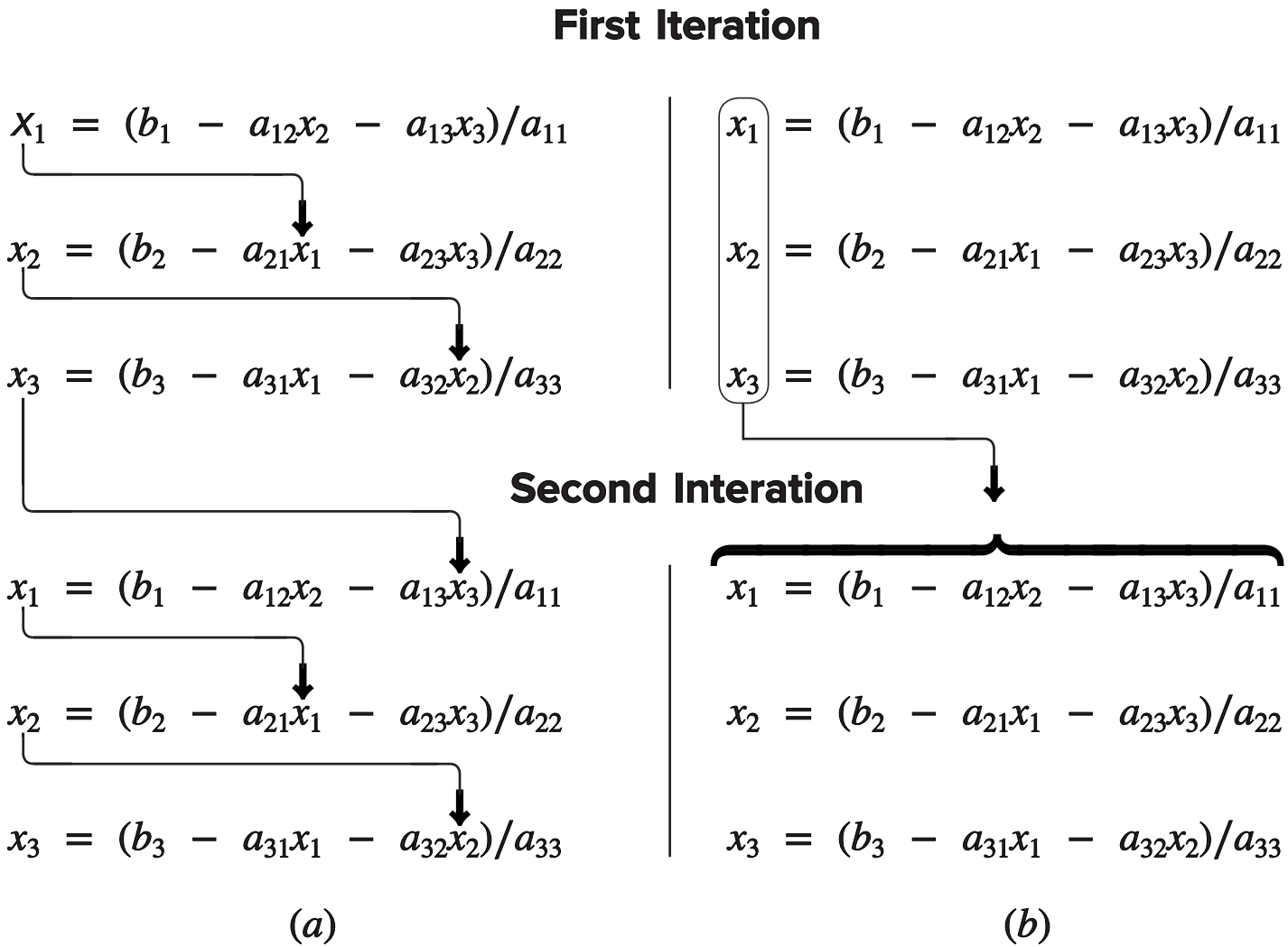
$$x_{2,i} = \frac{b_2 - a_{21}x_{1,i-1} - a_{23}x_{3,i-1} - \cdots - a_{2n}x_{n,i-1}}{a_{22}}$$

$$\vdots$$

$$x_{n,i} = \frac{b_n - a_{n1}x_{1,i-1} - a_{n2}x_{2,i-1} - \cdots - a_{n,n-1}x_{n-1,i-1}}{a_{nn}}$$

- repeat until : $|\varepsilon_{a,j}| = \left|\frac{x_{j,i} - x_{j,i-1}}{x_{j,i}}\right| 100\% < \varepsilon_s$ for $j = 1, \ldots, n$

Gauss-Seidel vs Jacobi iteration



Gauss-Seidel [left (a)] generally converges faster and is preferred

Outline

- system of equations
- Cramer rule for linear equations
- Gauss-Seidel and Jacobi methods for linear equations
- **fixed point iteration for nonlinear equations**
- Newton-Raphson for nonlinear equations

Fixed-point iteration for nonlinear systems

consider

$$f_1(x_1, x_2) = 0$$
$$f_2(x_1, x_2) = 0$$

Fixed point iteration

$$x_{1,i+1} = g_1(x_{1,i}, x_{2,i}), \qquad x_{2,i+1} = g_2(x_{1,i}, x_{2,i})$$

- extends single equation idea
- also called *successive substitution*

Example

$$f_1(x_1, x_2) = x_1^2 + x_1 x_2 - 10 = 0$$
$$f_2(x_1, x_2) = x_2 + 3x_1 x_2^2 - 57 = 0$$

with $x_{1,0} = 1.5$ and $x_{2,0} = 3.5$

Example: fixed-point iteration setup

formulate as

$$x_{1,i+1} = \frac{10 - x_{1,i}^2}{x_{2,i}}$$

and

$$x_{2,i+1} = 57 - 3x_{1,i}x_{2,i}^2$$

on the basis of the initial guesses, we have

$$x_1 = \frac{10 - (1.5)^2}{3.5} = 2.21429$$

and

$$x_2 = 57 - 3(2.21429)(3.5)^2 = -24.37516$$

repeating the computation:

$$x_1 = \frac{10 - (2.21429)^2}{-24.37516} = -0.20910$$

$$x_2 = 57 - 3(-0.20910)(-24.37516)^2 = 429.709$$

Observation: the approach is diverging rapidly

Alternative formulation

rearrange the equations as

$$x_1 = \sqrt{10 - x_1 x_2}, \qquad x_2 = \sqrt{\frac{57 - x_2}{3x_1}}$$

using initial guesses $x_1 = 1.5, x_2 = 3.5$:

$$x_1 = \sqrt{10 - 1.5(3.5)} = 2.17945$$

and

$$x_2 = \sqrt{\frac{57 - 3.5}{3(2.17945)}} = 2.86051$$

next iteration:

$$x_1 = \sqrt{10 - 2.17945(2.86051)} = 1.94053$$

$$x_2 = \sqrt{\frac{57 - 2.86051}{3(1.94053)}} = 3.04955$$

Conclusion: the reformulated system converges to the true solution

$$x_1 = 2, \quad x_2 = 3$$

Remarks

- convergence depends heavily on formulation
- poor initial guesses can cause divergence
- sufficient (but restrictive) conditions:

$$\left|\frac{\partial f_1}{\partial x_1}\right| + \left|\frac{\partial f_1}{\partial x_2}\right| < 1, \qquad \left|\frac{\partial f_2}{\partial x_1}\right| + \left|\frac{\partial f_2}{\partial x_2}\right| < 1$$

- limited utility for nonlinear systems, but useful for linear systems

Outline

- system of equations
- Cramer rule for linear equations
- Gauss-Seidel and Jacobi methods for linear equations
- fixed point iteration for nonlinear equations
- **Newton-Raphson for nonlinear equations**

First-order Taylor (affine) approximation

first-order *Taylor approximation* of $f : \mathbb{R}^n \to \mathbb{R}$, near point z:

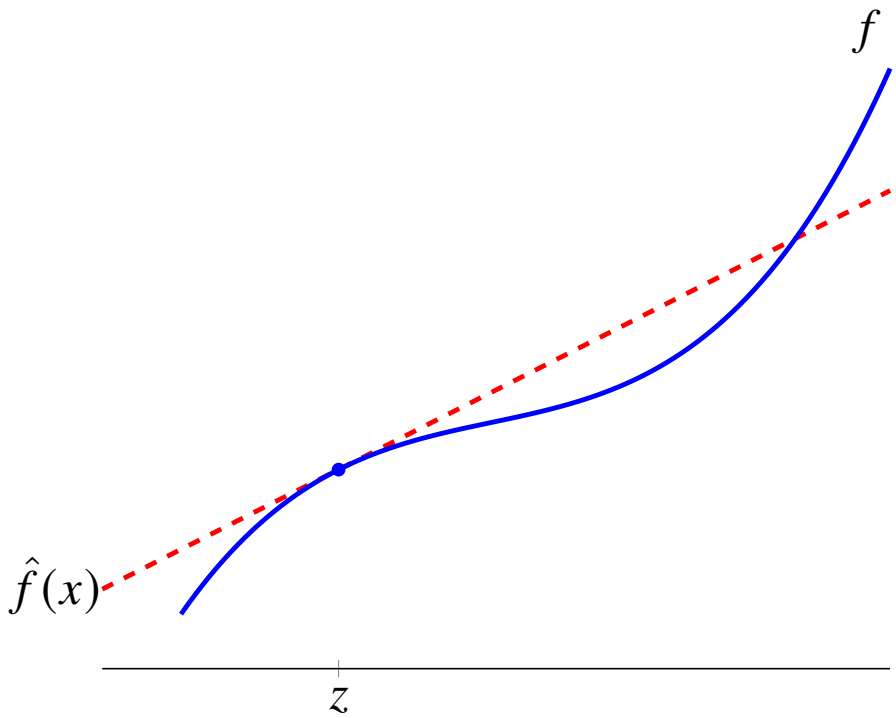
$$\begin{aligned}\hat{f}(x) &= f(z) + \frac{\partial f}{\partial x_1}(z)\,(x_1 - z_1) + \cdots + \frac{\partial f}{\partial x_n}(z)\,(x_n - z_n) \\ &= f(z) + \nabla f(z)^T (x - z)\end{aligned}$$

- n-vector $\nabla f(z)$ is the **gradient** of f at z,

$$\nabla f(z) = \left(\frac{\partial f}{\partial x_1}(z), \ldots, \frac{\partial f}{\partial x_n}(z)\right)$$

- $\hat{f}(x)$ is very close to $f(x)$ when x_k are all near z_k
- sometimes written $\hat{f}(x; z)$, to indicate that z where the approximation appear
- $\hat{f}$ is an affine function of x (linear plus constant term)
- often called *linear approximation* of f near z, even though it is in general affine

Example with one variable



$$\hat{f}(x) = f(z) + f'(z)(x - z)$$

Example with two variables

$$f(x_1, x_2) = x_1 - 3x_2 + e^{2x_1+x_2-1}$$

- gradient:

$$\nabla f(x) = \begin{bmatrix} 1 + 2e^{2x_1+x_2-1} \\ -3 + e^{2x_1+x_2-1} \end{bmatrix}$$

- Taylor approximation around $z = 0$:

$$\begin{aligned} \hat{f}(x) &= f(0) + \nabla f(0)^T (x - 0) \\ &= e^{-1} + (1 + 2e^{-1})x_1 + (-3 + e^{-1})x_2 \end{aligned}$$

Taylor approximation for vector-valued functions

first-order Taylor approximation of differentiable $f : \mathbb{R}^n \to \mathbb{R}^m$ around z:

$$\hat{f}_k(x) = f_k(z) + \frac{\partial f_k}{\partial x_1}(z)(x_1 - z_1) + \cdots + \frac{\partial f_k}{\partial x_n}(z)(x_n - z_n), \quad k = 1, \ldots, m$$

in matrix-vector notation: $\hat{f}(x) = f(z) + J(z)(x - z)$ where

$$J(z) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(z) & \frac{\partial f_1}{\partial x_2}(z) & \cdots & \frac{\partial f_1}{\partial x_n}(z) \\ \frac{\partial f_2}{\partial x_1}(z) & \frac{\partial f_2}{\partial x_2}(z) & \cdots & \frac{\partial f_2}{\partial x_n}(z) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(z) & \frac{\partial f_m}{\partial x_2}(z) & \cdots & \frac{\partial f_m}{\partial x_n}(z) \end{bmatrix} = \begin{bmatrix} \nabla f_1(z)^T \\ \nabla f_2(z)^T \\ \vdots \\ \nabla f_m(z)^T \end{bmatrix}$$

- $J(z)$ is the *derivative* or *Jacobian* matrix of f at z (sometimes written as $Df(z)$)
- $\hat{f}$ is a local affine approximation of f around z

Example

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} e^{2x_1+x_2} - x_1 \\ x_1^2 - x_2 \end{bmatrix}$$

- derivative matrix:

$$J(x) = \begin{bmatrix} 2e^{2x_1+x_2} - 1 & e^{2x_1+x_2} \\ 2x_1 & -1 \end{bmatrix}$$

- first order approximation of f around $z = 0$:

$$\hat{f}(x) = \begin{bmatrix} \hat{f}_1(x) \\ \hat{f}_2(x) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Newton-Raphson method for nonlinear equations

- linearize f (*i.e.*, make affine approximation) around current iterate x_i

$$\hat{f}(x; x_i) = f(x_i) + J(x_i)(x - x_i)$$

- take solution x of linearized equation $\hat{f}(x; x_i) = 0$ as the next iterate:

$$x_{i+1} = x_i - J(x_i)^{-1} f(x_i)$$

---

given a starting point x_0 and solution tolerance ε_s

repeat for $i \geq 0$

1. evaluate $J(x_i)$
2. set

$$x_{i+1} = x_i - J(x_i)^{-1} f(x_i)$$

   if $\sum_{k=1}^{n} (f_k(x_{i+1}))^2 < \varepsilon_s$ or $\frac{|x_{j,i+1} - x_{j,i}|}{|x_{j,i+1}|} 100\% < \varepsilon_s$, stop and output x_{i+1}

- $J(x_i)$ is assumed to be nonsingular
- each iteration requires one evaluation of $f(x)$ and $J(x)$
- also called (just) *Newton method*

Newton-Raphson for two nonlinear equations

consider two equations in two variables x, y:

$$f_1(x_1, x_2) = 0, \qquad f_2(x_1, x_2) = 0$$

write

$$\begin{bmatrix} x_{1,i+1} \\ x_{2,i+1} \end{bmatrix} = \begin{bmatrix} x_{1,i} \\ x_{2,i} \end{bmatrix} - \begin{bmatrix} \frac{\partial f_{1,i}}{\partial x_1} & \frac{\partial f_{1,i}}{\partial x_2} \\ \frac{\partial f_{2,i}}{\partial x_1} & \frac{\partial f_{2,i}}{\partial x_2} \end{bmatrix}^{-1} \begin{bmatrix} f_{1,i} \\ f_{2,i} \end{bmatrix}$$

computing inverse of 2 by 2 matrix gives the update below

Newton-Raphson update

$$x_{1,i+1} = x_{1,i} - \frac{f_{1,i}\frac{\partial f_{2,i}}{\partial x_2} - f_{2,i}\frac{\partial f_{1,i}}{\partial x_2}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

$$x_{2,i+1} = x_{2,i} - \frac{f_{2,i}\frac{\partial f_{1,i}}{\partial x_1} - f_{1,i}\frac{\partial f_{2,i}}{\partial x_1}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

denominator = determinant of Jacobian

Example: Newton-Raphson

$$f_1(x_1, x_2) = x_1^2 + x_1 x_2 - 10, \quad f_2(x_1, x_2) = x_2 + 3x_1 x_2^2 - 57$$

initial guess: $x_1 = 1.5, x_2 = 3.5$

- compute derivatives:

$$\frac{\partial f_1}{\partial x_1} = 2x_1 + x_2, \quad \frac{\partial f_1}{\partial x_2} = x_1$$

$$\frac{\partial f_2}{\partial x_1} = 3x_2^2, \quad \frac{\partial f_2}{\partial x_2} = 1 + 6x_1 x_2$$

- substitute $x_0 = (x_{1,0}, x_{2,0}) = (1.5, 3.5)$:

$$\frac{\partial f_{1,0}}{\partial x_1} = 6.5, \ \frac{\partial f_{1,0}}{\partial x_2} = 1.5, \ \frac{\partial f_{2,0}}{\partial x_1} = 36.75, \ \frac{\partial f_{2,0}}{\partial x_2} = 32.5$$

Example: Newton-Raphson results

- evaluate functions:

$$f_{1,0} = -2.5, \quad f_{2,0} = 1.625$$

- Jacobian determinant:

$$6.5(32.5) - 1.5(36.75) = 156.125$$

- updates:

$$x_{1,1} = x_{1,0} - \frac{f_{1,0}\frac{\partial f_{2,0}}{\partial x_2} - f_{2,0}\frac{\partial f_{1,0}}{\partial x_2}}{\frac{\partial f_{1,0}}{\partial x_1}\frac{\partial f_{2,0}}{\partial x_2} - \frac{\partial f_{1,0}}{\partial x_2}\frac{\partial f_{2,0}}{\partial x_1}} = 1.5 - \frac{-2.5(32.5)-1.625(1.5)}{156.125} = 2.036$$

$$x_{2,1} = x_{2,0} - \frac{f_{2,0}\frac{\partial f_{1,0}}{\partial x_1} - f_{1,0}\frac{\partial f_{2,i}}{\partial x_1}}{\frac{\partial f_{1,0}}{\partial x_1}\frac{\partial f_{2,0}}{\partial x_2} - \frac{\partial f_{1,0}}{\partial x_2}\frac{\partial f_{2,0}}{\partial x_1}} = 3.5 - \frac{1.625(6.5)-(-2.5)(36.75)}{156.125} = 2.844$$

$\Longrightarrow$ converges toward $(2, 3)$

Example: Newton-Raphson results

MATLAB first iteration

```
>> x = [1.5;3.5];
>> J = [2*x(1)+x(2)  x(1); 3*x(2)^2  1+6*x(1)*x(2)]
J =
6.5000    1.5000
36.7500   32.5000
>> f = [x(1)^2 + x(1)*x(2) - 10; x(2) + 3*x(1)*x(2)^2 - 57]
f =
-2.5000
1.6250
>> x = x - J\f
x =
2.0360
2.8439
```

General code

```
function [x,f,ea,iter] = newtmult(func,x0,es,maxit,varargin)
% newtmult: Newton-Raphson root zeroes nonlinear systems
if nargin < 2, error('at least 2 input arguments required'), end
if nargin < 3 || isempty(es), es = 0.0001; end
if nargin < 4 || isempty(maxit), maxit = 50; end
iter = 0;
x = x0;
while (1)
[J,f] = func(x,varargin{:});
dx = J\f;
x = x - dx;
iter = iter + 1;
ea = 100*max(abs(dx./x));
if iter >= maxit || ea <= es, break, end
end
```

Remarks on nonlinear equations

- both fixed-point and Newton-Raphson can diverge if initial guesses are poor
- Newton-Raphson does not work if Jacobian is non-singular (or nearly singular)
- no simple graphical procedure for choosing initial guesses in multivariable case
- advanced methods exist, but often trial and error, system knowledge are needed

References and further readings

- S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers* (8th edition). McGraw Hill, 2021. (Ch.9.1, 9.6, 11.2)

- S. C. Chapra. *Applied Numerical Methods with MATLAB for Engineers and Scientists* (5th edition). McGraw Hill, 2023. (Ch.9.1, 12)