Computational Methods

Systems of Linear Equations

Systems of Equations

- Often a system model contains multiple variables (parameters) and contains multiple equations
  - Multiple equations arise because problems have multiple outputs and multiple parameters
  - Multiple equations can also arise from multiple measurements
    - This might lead to equations that are not solvable
- Many iterative solutions to equation solving do not easily extend to solving systems of equations or equations in multiple variables

Systems of Linear Equations

- Linear equations are a special type which is easier to solve and has analytic solution methods
  - Single linear equation with *n* variables corresponds to a hyperplane in *n+1* dimensional space

$$a_1 x_1 + \ldots + a_n x_n = b$$

$$\vec{a}^T \vec{x} = b$$

    - Finding one analytic solution requires only one division
    - Has usually an infinite number of solutions if *n* is larger than 1
- Systems of linear equations consist of multiple LEs
  - Solution to a sytem of linear equations corresponds to the intersection of multiple hyperplanes

Systems of Linear Equations

- A system of linear equations can be written as a matrix multiplication

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,n}x_n &= b_1 \\ &\vdots \\ a_{m,1}x_1 + \ldots + a_{m,n}x_n &= b_m \end{aligned} \quad \Rightarrow \quad A\vec{x} = \vec{b}$$

- Systems of linear equations do not always have a unique solution
  - If there are too many equations there might be no solution
  - If there are too few equations then the system might have multiple solutions

Solving Linear Systems

- To solve a linear system analytically it is typically transformed into a system for which a solution can be easily computed
  - Diagonal system

$$\begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & 0 & a_{n,n} \end{pmatrix} \vec{x} = \vec{b} \quad \Rightarrow \quad \begin{array}{l} x_1 = \dfrac{b_1}{a_{1,1}} \\ \quad\vdots \\ x_n = \dfrac{b_n}{a_{n,n}} \end{array}$$

  - Triangular systems

$$\begin{pmatrix} a_{1,1} & \cdots & & a_{1,n} \\ \vdots & & & \vdots \\ 0 & \cdots & 0 & a_{n,n} \end{pmatrix} \vec{x} = \vec{b} \quad \Rightarrow \quad \begin{array}{l} x_n = \dfrac{b_n}{a_{n,n}} \\ x_{n-1} = \dfrac{b_{n-1} - a_{n-1,n} x_n}{a_{n-1,n-1}} \\ \quad\vdots \end{array}$$

Solving Linear Systems

- To transform a linear system into a different linear system a number of legal operations can be applied
  - Transformations correspond to premultiplying both sides of the linear system by a nonsingular matrix

$$A\vec{x} = \vec{b} \quad \Leftrightarrow \quad MA\vec{x} = M\vec{b} \quad \text{if } M \text{ is not singular}$$

  - Useful transformations:
    - Permutation: Swaps 2 rows (equations)
    - Row scaling: Scales each row by a scalar
    - Row addition: Subtracts a row from another row

Solving Linear Systems

- The most important transformation matrix for transforming a system into triangular form is the elimination matrix which combines row scaling and row subtraction
    - Elementary elimination matrix with pivot a_k

$$M_k = I - m_k e_k^T = I - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \frac{a_{k+1}}{a_k} \\ \vdots \\ \frac{a_n}{a_k} \end{pmatrix} \begin{pmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{pmatrix}$$

Solving Linear Systems

- Elementary elimination matrices can be combined into one elimination matrix

$$M_k M_l = I - m_k e_k^T - m_l e_l^T$$

- Elimination matrices are lower triangular and nonsingular
- The inverse of an elimination matrix simply swaps the sign for the off-diagonal terms
  - Inverse is lower triangular

$$L_k = M_k^{-1} = I + m_k e_k^T$$

Naïve Gaussian Elimination

- For systems of *n* linear equations in *n* variables, a number of analytic solution methods exist
  - Matrix Inversion

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

  - Naïve Gaussian elimination
    - Reduce system of equations to upper diagonal form and back substitute to compute the values

      $$MA\vec{x} = M\vec{b} \quad , \quad MA = U$$

    - Transformation of the system occurs through elimination
      - Add or subtract one equation from another
      - Multiply equations with a non-zero constant

Naïve Gaussian Elimination

- Elimination step
  - For every variable x_i, starting with i=1
    - Subtract $a_{j,i}/a_{i,i}$ times equation i from every equation *j, j>i*
- Back substitution step
  - Once the elimination is complete back substitution computes the values
  - For every variable x_i, starting with $i=n$
    - Compute x_i by solving the i^{th} equation using the previously computed values for x_j , $j>I$
- Naïve Gaussian elimination fails if any $a_{i,i}$ is 0

Gaussian Elimination

- Complexity of naïve Gaussian elimination (in terms of multiplications and additions)
  - Elimination step: $$\sum_{i=1}^{n-1}(n-i)(1+(n-i+1)) = \frac{n^3}{3}+\frac{n^2}{2}-5\frac{n}{6}$$
  - Back substitution step: $$\sum_{i=n}^{1}((n-i)+1) = \frac{n^2}{2}+\frac{n}{2}$$
- Computation has to be repeated for every *b*
- To address the problem with a 0 on the diagonal we have to use additional operations
  - Swap equations (rows) and variables (columns)

LU Factorization

- Gaussian elimination has to be recomputed every time *A* or *b* change
  - Often in practical problems we have to solve the same linear system for different result values
- LU factorization resolves this by explicitly decomposing *A* into the upper triangular matrix and the inverse of the elimination matrix
  - Only A is transformed through elimination $A = LU$
  - Solving for *b*, forward and backward substitution are used
    - Forward substitution with *L* $L\vec{y} = \vec{b}$
    - *Backward substitution with U* $U\vec{x} = \vec{y}$

LU Factorization

- Complexity of LU factorization is approximately the same as for Gaussian Elimination
  - Elimination step: $\approx \frac{n^3}{3}$
  - Forward and back substitution step: $\approx \frac{n^2}{2}$
- Only the forward and back substitution step has to be repeated for a new *b*
- Both methods (Gaussian Elimination and LU Factorization) are approximately 3 times faster than matrix inversion $\approx n^3$

Existence and Uniqueness

- Existence and uniqueness of solution depends on the equations and the target result value
  - In systems with *n* equations and *n* variables:
    - There exists a unique solution iff *A* is not singular
    - There are infinitely many solutions iff *A* is singular and *b* is in the span of
    - There is no solution iff *A* is singular and *b* is not in the span of *A*
- *A* is not singular if the following equivalent conditions apply:
  - There are *n* equations that are not linearly dependent
  - *A* is invertible, $rank(A)=n$, $det(A)\neq 0$

Error Measures and Norms

- To measure errors in multi-dimensional spaces, error vectors have to be reduced to scalars
  - Vector and matrix norms allow to do this
- Properties of vector norms
  - Positive $\|x\| > 0 \;\; if \;\; x \neq 0$
  - Scalar multiplication $\|\alpha x\| = |\alpha| \; \|x\| \;\; for \;\; any \;\; scalar \;\; \alpha$
  - Triangle inequality $\|x + y\| \leq \|x\| + \|y\|$

$$\|x - y\| \geq \big| \; \|x\| - \|y\| \; \big|$$

Vector Norms

- Common vector norms: p-norms

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}$$

  - 1-norm: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
  - 2-norm: $\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$
  - ∞-norm: $\|x\|_\infty = \max_i |x_i|$

- P-norms are related

$$\|x\|_1 \geq \|x\|_2 \geq \|x\|_\infty \quad for \quad all \quad x \in R^n$$

Matrix Norms

- Matrix norms can be defined in terms of a number of properties similar to the vector norm
  - Positive $\|A\| > 0 \quad if \quad A \neq 0$
  - Scalar multiplication $\|\alpha A\| = |\alpha| \, \|A\| \quad for \quad any \quad scalar \quad \alpha$
  - Triangle inequality $\|A + B\| \leq \|A\| + \|B\|$
- For each vector norm there is a special matrix norm that can be derived from it and has all the properties
  - Induced Matrix norm $\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$

Induced Matrix Norms

- Induced vector norms (operator norms) correspond to the maximum scaling the matrix applies to the vector in terms of the specific vector norm
  - For vector p-norms:
    - 1-norm: $\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{i,j}|$
    - ∞-norm: $\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{i,j}|$
  - Induced matrix norms have additional properties

$$\|AB\| \leq \|A\| \; \|B\|$$
$$\|Ax\| \leq \|A\| \; \|x\|$$

Sensitivity and Conditioning

- To estimate the sensitivity of solving a system of linear equations we have to calculate the forward and backward errors
  - Forward error: $\left\|x-\hat{x}\right\| = \left\|\Delta x\right\|$
  - Backward error (residual): $\left\|A\hat{x}-b\right\| = \left\|A(x+\Delta x)-Ax\right\|$

$$= \left\|A\Delta x\right\|$$

- (Relative) Condition number:

$$\frac{\|\Delta x\|/\|x\|}{\|res\|/\|b\|} = \frac{\|\Delta x\|}{\|x\|}\frac{\|b\|}{\|res\|} = \frac{\|Ax\|\left\|A^{-1}res\right\|}{\|res\|\|x\|} \le \frac{\|A\|\|x\|\left\|A^{-1}\right\|\|res\|}{\|x\|\|res\|} = \|A\|\left\|A^{-1}\right\|$$

$$cond(A) = \|A\|\left\|A^{-1}\right\|$$

Sensitivity and Conditioning

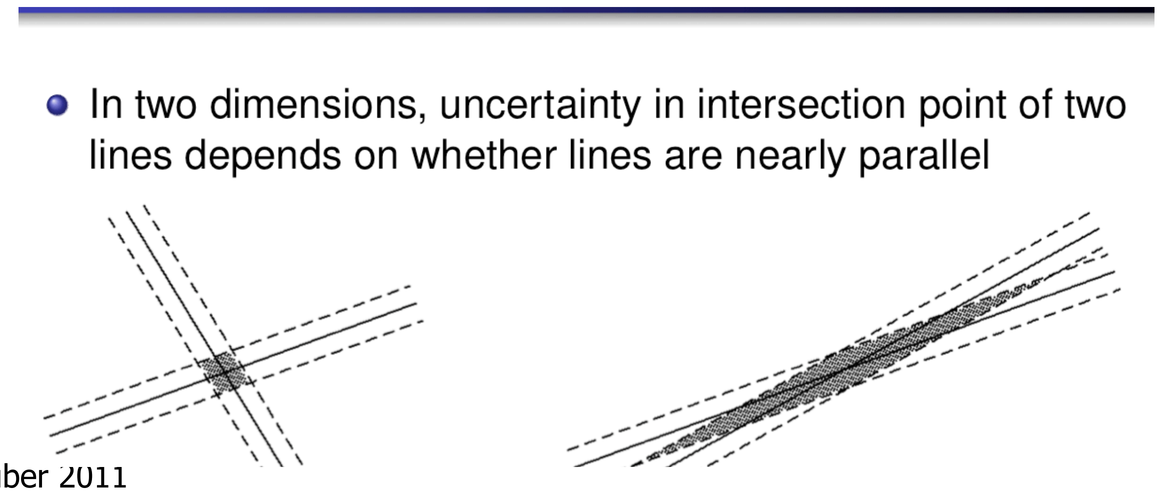
- Properties of condition number
  - *cond(A)* = ∞ for singular *A*
  - Condition measures the ratio of maximum stretching to maximum shrinking
$$\|A\|\|A^{-1}\| = \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} \Big/ \min_{x\neq 0} \frac{\|Ax\|}{\|x\|}$$
  - *cond(A) ≥ 1*
  - *cond(αA) = cond(A)*
- To compute condition number the norm of the inverse is often approximated as the maximum ratio of a set of solutions $\|A^{-1}\| \geq \frac{\|x\|}{\|Ax\|}$

Sensitivity and Conditioning

- Parameter sensitivity:

$$\frac{\|\Delta x\|}{\|x\|} \leq cond(A)\frac{\|\Delta A\|}{\|A\|}$$

- Conditioning depends on the relation of the hyperplanes


- In two dimensions, uncertainty in intersection point of two lines depends on whether lines are nearly parallel

Stability

- Naïve Gaussian Elimination and LU Factorization fail when a pivot is *0*
- While Gaussian elimination and LU Factorization have no truncation error, they introduce rounding error during the elimination and substitution steps
  - Using small pivots (and thus large multiplication factors) can lead to swamping where the the equation being subtracted overwhelms the equation it is subtracted from
    - Non-singular matrix becomes close to singular
  - Algorithms are more stable if they use larger pivots

Partial Pivoting

- To improve stability, partial pivoting uses row swaps (permutation matrices) to ensure that always the largest remaining entry in a column is used as the pivot
  - For Gaussian Elimination this is directly implemented by applying permutations between elimination steps

$$A\vec{x} = \vec{b} \quad \Leftrightarrow \quad PA\vec{x} = P\vec{b}$$

  - For LU-Factorization this causes problems
    - P^{-1} is not lower triangular
    - Row permutations have to be handled separately

$$PA = LU$$

PA=LU Factorization

- Permutations are tracked separately from eliminations leading to a factorization of *PA*
  $PA = LU$
- The solution step is extended to address row permutations $PA\vec{x} = LU\vec{x} = P\vec{b}$
  - Permute the elements in *b* using *P*
    $\vec{z} = P\vec{b}$
  - Forward substitution using *L*
    $L\vec{y} = \vec{z}$
  - *Back substitution using U*
    $U\vec{x} = \vec{y}$

Complete Pivoting

- Stability can be further improved by ordering the pivots from largest to smallest through column swaps
  - Always use the largest element in the remaining sub-matrix below row *k* as the pivot
    - Column swaps correspond to reordering the variables *x*
  - For Gaussian Elimination this is directly implemented by applying permutations between elimination steps

$$A\vec{x} = \vec{b} \quad \Leftrightarrow \quad AP\vec{x} = \vec{b}$$

  - For LU-Factorization row and column swaps have to be tracked separately $PAQ = LU$

Scaling and Iterative Refinement

- Scaling of rows and columns can be used to reduce rounding error and thus to improve stability
  - Large differences in coefficients can decrease stability
  - Sometimes scaling can improve the stability
- Iterative refinement allows to iteratively break down the residual to improve precision

$$Ax = b \quad \Rightarrow \quad x_0 \quad \Rightarrow \quad r_0 = b - Ax_0$$

$$Az = r_0 \quad \Rightarrow \quad z_0, x_1 = x_0 + z_0 \quad \Rightarrow \quad r_1 = r_0 - Az_0$$

  - Can sometimes lead to improved precision
    - However, residual calculation is sensitive to cancellation

Direct Solution Methods

- LU Factorization and Gaussian Elimination return a result without truncation error in $O(n^3)$ multiplications.
  - Fixed calculation complexity
  - Guaranteed solution for nonsingular matrix A
- For very large n or for very sparse matrices the complexity of $O(n^3)$ can be very high
  - Iterative methods can be used
    - Lower cost per iteration
    - Convergence has to be analyzed

Iterative Solution Methods

- Fixed point methods allowed for iterative solutions in single equations
- Iteration in systems of equations is more difficult
  - Convergence properties are more complex
- Complexity of iterative methods consists of evaluating the system of equations in each iteration, thus $O(n^2)$
  - If approximate solution is known the number of iterations required can be low
  - If A is sparse the complexity per iteration can drop further

Jacobi Method

- The Jacobi Method defines a fixed point system by rewriting each equation *$f_i(x)$* to compute *x_i*

$$f_i(\vec{x}) = \sum_{j=1}^{n} a_{i,j} x_j = b_i \quad \Rightarrow \quad x_i = \frac{b_i - \sum_{j \in [1..n],\, j \neq i} a_{i,j} x_j}{a_{i,i}}$$

- Starting with an initial vector $x^{(0)}$, the next value is calculated by evaluating the equations using the previous iteration's value $x^{(t-1)}$
  - Is ensured to converge if *A* is strictly diagonally dominant (i.e. the coefficients on the diagonal are strictly larger than all other coefficients in the corresponding row

Jacobi Method

- Jacobi Method can be rewritten as strict fixed point iteration

$$A\vec{x} = \vec{b}$$

$$(D + L + U)\vec{x} = \vec{b}$$

$$D\vec{x} = (\vec{b} - (L + U)\vec{x})$$

$$\vec{x} = D^{-1}(\vec{b} - (L + U)\vec{x})$$

  - Number of iterations required depends on the starting point
  - For sparse matrices the function form is more efficient than the matrix form

Gauss-Seidel Method

- The Gauss-Seidel method is variation of the Jacobi method in which for each equation the most recent estimate (from the current iteration) is used rather than the result form the previous iteration

$$A\vec{x} = \vec{b}$$

$$(D+L)\vec{x}^{(t)} = \vec{b} - U\vec{x}^{t-1}$$

$$\vec{x}^{(t)} = D^{-1}(\vec{b} - U\vec{x}^{(t-1)} - L\vec{x}^{(t)})$$

  - Has the same convergence conditions as the Jacobi method
  - Usually converges faster than Jacobi method

Successive Over-Relaxation

- Successive Over-Relaxation (SOR) further increases convergence speed by anticipating future changes and "overshooting" the iteration point of the Gauss-Seidel method by a relaxation parameter $\omega > 1$

$$x_i^t = (1-\omega)x_i^{t-1} + \omega \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^t - \sum_{j=i+1}^{n} a_{i,j} x_j^{t-1}}{a_{i,i}}$$

  - For $\omega = 1$ SOR is equal to Gauss-Seidel
  - Usually converges faster than Gauss-Seidel for appropriate relaxation parameters

Systems of Linear Equations

- Systems of linear equations can be solved using either direct or iterative methods
- Direct methods have fixed computation costs ($O(n^3)$) and incur no truncation error
- Iterative methods are variants of fixed point methods and have a smaller per iteration complexity ($O(n^2)$)
  - Less complex for large systems in which a good starting point is known
  - Less complex for sparse matrices (i.e. systems of equations where each equation only depends on a subset of the variables