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 system 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

\[ a_{1,1}x_1 + ... + a_{1,n}x_n = b_1 \]
\[ \vdots \]
\[ a_{m,1}x_1 + ... + a_{m,n}x_n = b_m \]

\[ \Rightarrow A\vec{x} = \vec{b} \]

- 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**
  \[
  \begin{pmatrix}
  a_{1,1} & 0 & \cdots & 0 \\
  \vdots & \ddots & \ddots & \vdots \\
  0 & \cdots & 0 & a_{n,n}
  \end{pmatrix}
  \begin{pmatrix}
  x_1 \\
  \vdots \\
  x_n
  \end{pmatrix}
  =
  \begin{pmatrix}
  b_1 \\
  \vdots \\
  b_n
  \end{pmatrix}
  \Rightarrow
  x_1 = \frac{b_1}{a_{1,1}},
  \quad
  x_n = \frac{b_n}{a_{n,n}},
  \quad
  x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}
  \]

- **Triangular systems**
  \[
  \begin{pmatrix}
  a_{1,1} & \cdots & a_{1,n} \\
  \vdots & \ddots & \vdots \\
  0 & \cdots & 0
  \end{pmatrix}
  \begin{pmatrix}
  x_1 \\
  \vdots \\
  x_n
  \end{pmatrix}
  =
  \begin{pmatrix}
  b_1 \\
  \vdots \\
  b_n
  \end{pmatrix}
  \Rightarrow
  x_1 = \frac{b_1}{a_{1,1}},
  \quad
  x_n = \frac{b_n}{a_{n,n}},
  \quad
  x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}
  \]

© Manfred Huber 2011
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
  \[ A\tilde{x} = \tilde{b} \iff MA\tilde{x} = M\tilde{b} \quad \text{if } M \text{ is not singular} \]

- 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 $a_k$

$$ M_k = I - m_k e_k^T = I - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \frac{a_{k+1}}{a_k} \\ \frac{a_{k}}{a_k} \\ \vdots \\ \frac{a_{n}}{a_k} \end{pmatrix} (0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0) $$
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
    \[ \tilde{x} = A^{-1}\tilde{b} \]
  - Naïve Gaussian elimination
    - Reduce system of equations to upper diagonal form and back substitute to compute the values
      \[ MA\tilde{x} = Mb, \quad MA = U \]
    - 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, \( \text{rank}(A)=n, \text{det}(A)\neq0 \)
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: $\|x + y\| \leq \|x\| + \|y\|$
    $\|x - y\| \geq |\|x\| - \|y\||$
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: \[ \|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \]
  
  - \(\infty\)-norm: \[ \|x\|_\infty = \max_i |x_i| \]

- P-norms are related
  
  \[ \|x\|_1 \geq \|x\|_2 \geq \|x\|_\infty \text{ for all } x \in \mathbb{R}^n \]
Matrix Norms

- Matrix norms can be defined in terms of a number of properties similar to the vector norm
  - Positive: \( \|A\| > 0 \) if \( A \neq 0 \)
  - Scalar multiplication: \( \|\alpha A\| = |\alpha| \|A\| \) for any scalar \( \alpha \)
  - 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: \( \|A\|_1 = \max_j \sum_{i=1}^n |a_{i,j}| \)
    - \( \infty \)-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: \( \| x - \hat{x} \| = \| \Delta x \| \)
- Backward error (residual): \( \| A\hat{x} - b \| = \| A(x + \Delta x) - Ax \| = \| A\Delta x \| \)

- (Relative) Condition number:

\[
\frac{\| \Delta x \|}{\| x \|} = \frac{\| \Delta x \|}{\| b \|} = \frac{\| Ax \| A^{-1} \text{res} \|}{\| x \| \| x \| A^{-1} \text{res} \|} \leq \frac{\| A \| x \| A^{-1} \| \text{res} \|}{\| x \| \| res \|} = \| A \| \| A^{-1} \|
\]

\[ cond(A) = \| A \| \| A^{-1} \| \]
Sensitivity and Conditioning

- Properties of condition number
  - $\text{cond}(A) = \infty$ for singular $A$
  - Condition measures the ratio of maximum stretching to maximum shrinking
    \[
    \|A\|\|A^{-1}\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} / \min_{x \neq 0} \frac{\|Ax\|}{\|x\|}
    \]
  - $\text{cond}(A) \geq 1$
  - $\text{cond}(\alpha A) = \text{cond}(A)$
  - 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:
  \[
  \frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta A\|}{\|A\|}
  \]

- 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 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
    \[ A\tilde{x} = \tilde{b} \iff PA\tilde{x} = Pb \]
  - 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
  $$PA\tilde{x} = LU\tilde{x} = P\tilde{b}$$
  - Permute the elements in $b$ using $P$
    $$\tilde{z} = Pb$$
  - Forward substitution using $L$
    $$L\tilde{y} = \tilde{z}$$
  - Back substitution using $U$
    $$U\tilde{x} = \tilde{y}$$
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:
    \[ A\tilde{x} = \tilde{b} \iff AP\tilde{x} = b \]
  - 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 $x_i$

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

- 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

\[ A\tilde{x} = \tilde{b} \]

\[ (D + L + U)\tilde{x} = \tilde{b} \]

\[ D\tilde{x} = (\tilde{b} - (L + U)\tilde{x}) \]

\[ \tilde{x} = D^{-1}(\tilde{b} - (L + U)\tilde{x}) \]

- 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 a variation of the Jacobi method in which for each equation the most recent estimate (from the current iteration) is used rather than the result from 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 the 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 $\omega > 1$.

\[
x_i^t = (1 - \omega)x_i^{t-1} + \omega \left( b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^t - \sum_{j=i+1}^{n} a_{i,j} x_j^{t-1} \right) / a_{i,i}
\]

- 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).