JDEP 384H: Numerical Methods in Business

Instructor: Thomas Shores
Department of Mathematics

Lecture 19, February 27, 2007
110 Kaufmann Center

Outline

Outline

Outline

Outline

Outline

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Iterative Methods for Linear Systems

Splitting:

A general procedure for developing iterations to solve $A\mathbf{x} = \mathbf{b}$:

- First write $A = B - C$, where solving $B\mathbf{y} = \mathbf{d}$ for $\mathbf{y}$ is easy.
- Rewrite the system as $(B - C)\mathbf{x} = \mathbf{b}$, i.e., $B\mathbf{x} = C\mathbf{x} + \mathbf{b}$.
- Or $\mathbf{x} = B^{-1}(C\mathbf{x} + \mathbf{b}) = B^{-1}C\mathbf{x} + B^{-1}\mathbf{b} = G\mathbf{x} + \mathbf{d}$.
- Now iterate on $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$.
- Notation: **spectral radius** of matrix G is $\rho(G)$, the maximum absolute value of any eigenvalue of G.
- **Key Theorem:** If $\rho(G) < 1$, or $\rho(G) = 1$ with exactly one eigenvalue equal 1 and the others smaller than 1, then the iterative method $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$ is guaranteed to converge; however, if $\rho(G) > 1$, method is guaranteed to diverge for nearly all initial $\mathbf{x}^{(0)}$.

Examples

Some Classical Splittings:

- Write $A = L(ower) + D(iagonal) + U(pper)$
- Jacobi: $D\mathbf{x} = -(L+U)\,\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -D^{-1}(L+U)\,\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$.
- Gauss-Seidel: $(L+D)\,\mathbf{x} = -U\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -(L+D)^{-1}\,U\mathbf{x}^{(k)} + (L+D)^{-1}\,\mathbf{b}$.
- SOR: Given any iteration scheme $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$, speed it up by $\mathbf{x}^{(k+1)} = \omega\left(G\mathbf{x}^{(k)} + \mathbf{d}\right) + (1-\omega)\,\mathbf{d}$, with $0 < \omega < 2$. (What does $\omega = 1$ give?)
- GS-SOR: Apply SOR to Gauss-Seidel. This is the most famous (and perhaps useful) example of SOR.

Let's try these on the example system given earlier, then check spectral radius of each iteration matrix. Smaller spectral radius means faster convergence.

Examples

Some Classical Splittings:

- Write $A = L(ower) + D(iagonal) + U(pper)$
- Jacobi: $D\mathbf{x} = -(L+U)\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$.
- Gauss-Seidel: $(L+D)\mathbf{x} = -U\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -(L+D)^{-1}U\mathbf{x}^{(k)} + (L+D)^{-1}\mathbf{b}$.
- SOR: Given any iteration scheme $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$, speed it up by $\mathbf{x}^{(k+1)} = \omega\left(G\mathbf{x}^{(k)} + \mathbf{d}\right) + (1-\omega)\mathbf{d}$, with $0 < \omega < 2$. (What does $\omega = 1$ give?)
- GS-SOR: Apply SOR to Gauss-Seidel. This is the most famous (and perhaps useful) example of SOR.

Let's try these on the example system given earlier, then check spectral radius of each iteration matrix. Smaller spectral radius means faster convergence.

Examples

Some Classical Splittings:

- Write $A = L(ower) + D(iagonal) + U(pper)$
- Jacobi: $D\mathbf{x} = -(L+U)\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$.
- Gauss-Seidel: $(L+D)\mathbf{x} = -U\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -(L+D)^{-1}U\mathbf{x}^{(k)} + (L+D)^{-1}\mathbf{b}$.
- SOR: Given any iteration scheme $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$, speed it up by $\mathbf{x}^{(k+1)} = \omega\left(G\mathbf{x}^{(k)} + \mathbf{d}\right) + (1-\omega)\mathbf{d}$, with $0 < \omega < 2$. (What does $\omega = 1$ give?)
- GS-SOR: Apply SOR to Gauss-Seidel. This is the most famous (and perhaps useful) example of SOR.

Let's try these on the example system given earlier, then check spectral radius of each iteration matrix. Smaller spectral radius means faster convergence.

Examples

Some Classical Splittings:

- Write $A = L(ower) + D(iagonal) + U(pper)$
- Jacobi: $D\mathbf{x} = -(L+U)\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$.
- Gauss-Seidel: $(L+D)\mathbf{x} = -U\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -(L+D)^{-1}U\mathbf{x}^{(k)} + (L+D)^{-1}\mathbf{b}$.
- SOR: Given any iteration scheme $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$, speed it up by $\mathbf{x}^{(k+1)} = \omega\left(G\mathbf{x}^{(k)} + \mathbf{d}\right) + (1-\omega)\mathbf{d}$, with $0 < \omega < 2$. (What does $\omega = 1$ give?)
- GS-SOR: Apply SOR to Gauss-Seidel. This is the most famous (and perhaps useful) example of SOR.

Let's try these on the example system given earlier, then check spectral radius of each iteration matrix. Smaller spectral radius means faster convergence.

Examples

Some Classical Splittings:

- Write $A = L(ower) + D(iagonal) + U(pper)$
- Jacobi: $D\mathbf{x} = -(L+U)\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$.
- Gauss-Seidel: $(L+D)\mathbf{x} = -U\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -(L+D)^{-1}U\mathbf{x}^{(k)} + (L+D)^{-1}\mathbf{b}$.
- SOR: Given any iteration scheme $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$, speed it up by $\mathbf{x}^{(k+1)} = \omega\left(G\mathbf{x}^{(k)} + \mathbf{d}\right) + (1-\omega)\mathbf{d}$, with $0 < \omega < 2$. (What does $\omega = 1$ give?)
- GS-SOR: Apply SOR to Gauss-Seidel. This is the most famous (and perhaps useful) example of SOR.

Let's try these on the example system given earlier, then check spectral radius of each iteration matrix. Smaller spectral radius means faster convergence.

Examples

Some Classical Splittings:

- Write $A = L(ower) + D(iagonal) + U(pper)$
- Jacobi: $D\mathbf{x} = -(L+U)\,\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -D^{-1}(L+U)\,\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$.
- Gauss-Seidel: $(L+D)\,\mathbf{x} = -U\mathbf{x} + \mathbf{b}$, so $\mathbf{x}^{(k+1)} = -(L+D)^{-1}\,U\mathbf{x}^{(k)} + (L+D)^{-1}\,\mathbf{b}$.
- SOR: Given any iteration scheme $\mathbf{x}^{(k+1)} = G\mathbf{x}^{(k)} + \mathbf{d}$, speed it up by $\mathbf{x}^{(k+1)} = \omega\left(G\mathbf{x}^{(k)} + \mathbf{d}\right) + (1-\omega)\,\mathbf{d}$, with $0 < \omega < 2$. (What does $\omega = 1$ give?)
- GS-SOR: Apply SOR to Gauss-Seidel. This is the most famous (and perhaps useful) example of SOR.

Let's try these on the example system given earlier, then check spectral radius of each iteration matrix. Smaller spectral radius means faster convergence.

Iterative Methods for Discrete Dynamical Systems

Definition

A **linear discrete dynamical system** consists of an initial state $\mathbf{x}^{(0)} \in \mathbb{R}^n$, an $n \times n$ transition matrix A and a transition rule from one state to the next given as

$$\mathbf{x}^{(k+1)} = A\mathbf{x}^{(k)},\ k = 0, 1, 2, \ldots$$

Definition

A **discrete Markov chain** is a linear discrete dynamical system such that each column of the transition matrix and the initial state are probability distribution vectors, that is, their entries are non-negative and sum to one.

Iterative Methods for Discrete Dynamical Systems

Definition

A **linear discrete dynamical system** consists of an initial state $\mathbf{x}^{(0)} \in \mathbb{R}^n$, an $n \times n$ transition matrix A and a transition rule from one state to the next given as

$$\mathbf{x}^{(k+1)} = A\mathbf{x}^{(k)},\ k = 0, 1, 2, \ldots$$

Definition

A **discrete Markov chain** is a linear discrete dynamical system such that each column of the transition matrix and the initial state are probability distribution vectors, that is, their entries are non-negative and sum to one.

Example

Example

Recall this example from Lecture 2: Two toothpaste companies compete for customers in a fixed market in which each customer uses either Brand A or Brand B. Market analysis shows that buying habits of customers fit the following pattern in the quarters that were analyzed: each quarter (three month period) 30% of A users will switch to B, and rest stay with A. Also, 40% of B users will switch to A, and rest will stay with B. *Assume* that this pattern does not vary from quarter to quarter, and we have a Markov chain model.

Solution. We expressed the problem in matrix form as

$$\mathbf{x}^{(k)} = \begin{bmatrix} a_k \\ b_k \end{bmatrix}, \quad A = \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix}, \quad \mathbf{x}^{(k+1)} = A\mathbf{x}^{(k)},$$

treated this formula as fixed point iteration and found experimentally a long-term state $(4/7, 3/7)$, which was also an eigenvector corresponding to eigenvalue $\lambda = 1$.

Finite State Discrete Stochastic Process:

Sequence of r.v.'s $X_0, X_1, X_2, \ldots$, where at the k-th stage the r.v. X_k assumes integer values between 1 and n that corresponds to one of n mutually exclusive states.

- Thus each stage is described by a probability distribution vector $\mathbf{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right]$, where $x_j^{(k)}$ is probability that $X^{(k)} = j$, that is, $P\left[X^{(k)} = j\right] = x_j^{(k)}$.
- FSD stochastic process $\{X_k\}_{k=0}^{\infty}$ is a **Markov chain** if there is a matrix of probabilities $P = [p_{i,j}]_{n,n}$ such that for all i, j, k, $P\left[X^{(k+1)} = i \mid X^{(k)} = j\right] = p_{i,j}$.
- Thus, the columns of P are probability distribution vectors (non-negative entries summing to 1).
- By law of total probability: for all $k \geq 0$, $\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)}$.
- Reinterpret the toothpaste example from this perspective.

Finite State Discrete Stochastic Process:

Sequence of r.v.'s $X_0, X_1, X_2, \ldots$, where at the k-th stage the r.v. X_k assumes integer values between 1 and n that corresponds to one of n mutually exclusive states.

- Thus each stage is described by a probability distribution vector $\mathbf{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right]$, where $x_j^{(k)}$ is probability that $X^{(k)} = j$, that is, $P\left[X^{(k)} = j\right] = x_j^{(k)}$.
- FSD stochastic process $\{X_k\}_{k=0}^{\infty}$ is a **Markov chain** if there is a matrix of probabilities $P = [p_{i,j}]_{n,n}$ such that for all i, j, k, $P\left[X^{(k+1)} = i \mid X^{(k)} = j\right] = p_{i,j}$.
- Thus, the columns of P are probability distribution vectors (non-negative entries summing to 1).
- By law of total probability: for all $k \geq 0$, $\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)}$.
- Reinterpret the toothpaste example from this perspective.

Probabilistic Interpretation

Finite State Discrete Stochastic Process:

Sequence of r.v.'s $X_0, X_1, X_2, \ldots$, where at the k-th stage the r.v. X_k assumes integer values between 1 and n that corresponds to one of n mutually exclusive states.

- Thus each stage is described by a probability distribution vector $\mathbf{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right]$, where $x_j^{(k)}$ is probability that $X^{(k)} = j$, that is, $P\left[X^{(k)} = j\right] = x_j^{(k)}$.
- FSD stochastic process $\{X_k\}_{k=0}^{\infty}$ is a **Markov chain** if there is a matrix of probabilities $P = [p_{i,j}]_{n,n}$ such that for all i, j, k, $P\left[X^{(k+1)} = i \mid X^{(k)} = j\right] = p_{i,j}$.
- Thus, the columns of P are probability distribution vectors (non-negative entries summing to 1).
- By law of total probability: for all $k \geq 0$, $\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)}$.
- Reinterpret the toothpaste example from this perspective.

Probabilistic Interpretation

Finite State Discrete Stochastic Process:

Sequence of r.v.'s $X_0, X_1, X_2, \ldots$, where at the k-th stage the r.v. X_k assumes integer values between 1 and n that corresponds to one of n mutually exclusive states.

- Thus each stage is described by a probability distribution vector $\mathbf{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right]$, where $x_j^{(k)}$ is probability that $X^{(k)} = j$, that is, $P\left[X^{(k)} = j\right] = x_j^{(k)}$.
- FSD stochastic process $\{X_k\}_{k=0}^{\infty}$ is a **Markov chain** if there is a matrix of probabilities $P = [p_{i,j}]_{n,n}$ such that for all i, j, k, $P\left[X^{(k+1)} = i \mid X^{(k)} = j\right] = p_{i,j}$.
- Thus, the columns of P are probability distribution vectors (non-negative entries summing to 1).
- By law of total probability: for all $k \geq 0$, $\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)}$.
- Reinterpret the toothpaste example from this perspective.

Probabilistic Interpretation

Finite State Discrete Stochastic Process:

Sequence of r.v.'s $X_0, X_1, X_2, \ldots$, where at the k-th stage the r.v. X_k assumes integer values between 1 and n that corresponds to one of n mutually exclusive states.

- Thus each stage is described by a probability distribution vector $\mathbf{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right]$, where $x_j^{(k)}$ is probability that $X^{(k)} = j$, that is, $P\left[X^{(k)} = j\right] = x_j^{(k)}$.
- FSD stochastic process $\{X_k\}_{k=0}^{\infty}$ is a **Markov chain** if there is a matrix of probabilities $P = [p_{i,j}]_{n,n}$ such that for all i, j, k, $P\left[X^{(k+1)} = i \mid X^{(k)} = j\right] = p_{i,j}$.
- Thus, the columns of P are probability distribution vectors (non-negative entries summing to 1).
- By law of total probability: for all $k \geq 0$, $\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)}$.
- Reinterpret the toothpaste example from this perspective.

Probabilistic Interpretation

Finite State Discrete Stochastic Process:

Sequence of r.v.'s $X_0, X_1, X_2, \ldots$, where at the k-th stage the r.v. X_k assumes integer values between 1 and n that corresponds to one of n mutually exclusive states.

- Thus each stage is described by a probability distribution vector $\mathbf{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right]$, where $x_j^{(k)}$ is probability that $X^{(k)} = j$, that is, $P\left[X^{(k)} = j\right] = x_j^{(k)}$.
- FSD stochastic process $\{X_k\}_{k=0}^{\infty}$ is a **Markov chain** if there is a matrix of probabilities $P = [p_{i,j}]_{n,n}$ such that for all i, j, k, $P\left[X^{(k+1)} = i \mid X^{(k)} = j\right] = p_{i,j}$.
- Thus, the columns of P are probability distribution vectors (non-negative entries summing to 1).
- By law of total probability: for all $k \geq 0$, $\mathbf{x}^{(k+1)} = P\mathbf{x}^{(k)}$.
- Reinterpret the toothpaste example from this perspective.

Outline

Polynomial Data Fitting

The Idea:

Since we love polynomials (they're easy functions), try to fit them to a given set of data points:

- Taylor polynomials: $f(x) \approx f(0) + f'(0)\,x + \frac{f''(0)}{2!}x^2 + \cdots$. Good locally, rotten globally.
- **Fact:** given n data points $P_i = (x_i, y_i)$, $i = 1, \ldots, n$ with distinct abscissas x_i, there is a unique polynomial $p(x)$ that interpolates these points, i.e., $p(x_i) = y_i$, $i = 1, \ldots, n$.
- Try a third degree Taylor polynomial and third degree fit with polyfit on $f(x) = e^x$, $-2 \le x \le 2$.

Polynomial Data Fitting

The Idea:

Since we love polynomials (they're easy functions), try to fit them to a given set of data points:

- Taylor polynomials: $f(x) \approx f(0) + f'(0)\,x + \frac{f''(0)}{2!}x^2 + \cdots$. Good locally, rotten globally.
- **Fact:** given n data points $P_i = (x_i, y_i)$, $i = 1, \ldots, n$ with distinct abscissas x_i, there is a unique polynomial $p(x)$ that interpolates these points, i.e., $p(x_i) = y_i$, $i = 1, \ldots, n$.
- Try a third degree Taylor polynomial and third degree fit with polyfit on $f(x) = e^x$, $-2 \leq x \leq 2$.

Polynomial Data Fitting

The Idea:

Since we love polynomials (they're easy functions), try to fit them to a given set of data points:

- Taylor polynomials: $f(x) \approx f(0) + f'(0)\,x + \frac{f''(0)}{2!}x^2 + \cdots$. Good locally, rotten globally.
- **Fact:** given n data points $P_i = (x_i, y_i)$, $i = 1, \ldots, n$ with distinct abscissas x_i, there is a unique polynomial $p(x)$ that interpolates these points, i.e., $p(x_i) = y_i$, $i = 1, \ldots, n$.
- Try a third degree Taylor polynomial and third degree fit with polyfit on $f(x) = e^x$, $-2 \leq x \leq 2$.

Polynomial Data Fitting

The Idea:

Since we love polynomials (they're easy functions), try to fit them to a given set of data points:

- Taylor polynomials: $f(x) \approx f(0) + f'(0)\,x + \frac{f''(0)}{2!}x^2 + \cdots$. Good locally, rotten globally.
- **Fact:** given n data points $P_i = (x_i, y_i)$, $i = 1, \ldots, n$ with distinct abscissas x_i, there is a unique polynomial $p(x)$ that interpolates these points, i.e., $p(x_i) = y_i$, $i = 1, \ldots, n$.
- Try a third degree Taylor polynomial and third degree fit with polyfit on $f(x) = e^x$, $-2 \le x \le 2$.

Example

Let's do some calculation with polynomial objects in Matlab:

```
> x = [-2 -1 1 2]
> y = exp(x)
> plot(x,y,'o'),grid,hold on
> plot(x,y)
> poly = polyfit(x,y,4)
> xx = -2:.1:2;
> plot(xx,polyval(poly,xx))
> plot(xx,exp(xx))
```

Outline

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Other Methods

Other Models of Curve Fitting:

- Rational functions $p(x)/q(x)$, where $p(x), q(x)$ are polynomials. These fittings get very complicated.
- Splines: functions $P(x)$ that are polynomials in between "knots" $x_1, x_2, \ldots, x_n$ and fitted as smoothly as possible at the knots. Most useful:
- Linear splines: we've all used them; they are "dot-to-dots".
- Cubic splines 1: (**Natural cubic splines**) minimize "wiggle" in a curve, but not the most accurate cubic spline. Second derivatives at the endpoints are zero, which may be incorrect.
- Cubic splines 2: (**Clamped cubic splines**) Match derivatives at endpoints and interpolate all points.
- Cubic splines 3: (**Not-a-knot cubic splines**) Fake clamping by using two next to endpoints not as knots. Matlab default.

Examples

Let's do some calculations for a given data set:

```
> x=1:10
> y=[4, 2.5, -2, -1, 2, 5, 4, 6, 4.5, 3]
> plot(x,y,'o'),grid,hold on
> plot(x,y)
> poly = polyfit(x,y,9)
> xx = 1:.01:10;
> plot(xx,polyval(poly,xx))
> spln = spline(x,y)
> plot(xx,ppval(spln,xx))
```

Outline

Root Finding for Univariate Functions

Basic Problem:

Given an equation $f(x) = 0$ that we cannot solve explicitly for real number x with algebra or calculus. How do we find a solution (given that there is one)?

- There are many classical numerical methods. One of them is **Newton's method**: start with an initial guess x_0 and iterate $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- Another classical method is bisection: Find an interval where $f(x)$ changes sign and bisect it iteratively, preserving the sign change.
- Matlab has a built-in function `fzero` that uses a bisection type method and no derivative information.
- Solve $f(x) = 0$ numerically, where $f(x) = x - 2\sin(x)$, on the interval $[0, 3]$.

Root Finding for Univariate Functions

Basic Problem:

Given an equation $f(x) = 0$ that we cannot solve explicitly for real number x with algebra or calculus. How do we find a solution (given that there is one)?

- There are many classical numerical methods. One of them is **Newton's method**: start with an initial guess x_0 and iterate $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- Another classical method is bisection: Find an interval where $f(x)$ changes sign and bisect it iteratively, preserving the sign change.
- Matlab has a built-in function `fzero` that uses a bisection type method and no derivative information.
- Solve $f(x) = 0$ numerically, where $f(x) = x - 2\sin(x)$, on the interval $[0, 3]$.

Root Finding for Univariate Functions

Basic Problem:

Given an equation $f(x) = 0$ that we cannot solve explicitly for real number x with algebra or calculus. How do we find a solution (given that there is one)?

- There are many classical numerical methods. One of them is **Newton's method**: start with an initial guess x_0 and iterate $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- Another classical method is bisection: Find an interval where $f(x)$ changes sign and bisect it iteratively, preserving the sign change.
- Matlab has a built-in function `fzero` that uses a bisection type method and no derivative information.
- Solve $f(x) = 0$ numerically, where $f(x) = x - 2\sin(x)$, on the interval $[0, 3]$.

Root Finding for Univariate Functions

Basic Problem:

Given an equation $f(x) = 0$ that we cannot solve explicitly for real number x with algebra or calculus. How do we find a solution (given that there is one)?

- There are many classical numerical methods. One of them is **Newton's method**: start with an initial guess x_0 and iterate $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- Another classical method is bisection: Find an interval where $f(x)$ changes sign and bisect it iteratively, preserving the sign change.
- Matlab has a built-in function `fzero` that uses a bisection type method and no derivative information.
- Solve $f(x) = 0$ numerically, where $f(x) = x - 2\sin(x)$, on the interval $[0, 3]$.

Root Finding for Univariate Functions

Basic Problem:

Given an equation $f(x) = 0$ that we cannot solve explicitly for real number x with algebra or calculus. How do we find a solution (given that there is one)?

- There are many classical numerical methods. One of them is **Newton's method**: start with an initial guess x_0 and iterate $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- Another classical method is bisection: Find an interval where $f(x)$ changes sign and bisect it iteratively, preserving the sign change.
- Matlab has a built-in function `fzero` that uses a bisection type method and no derivative information.
- Solve $f(x) = 0$ numerically, where $f(x) = x - 2\sin(x)$, on the interval $[0, 3]$.

Matlab Calculations

```
> help fzero
> myfcn = @(x) x-2*sin(x) % an ‘‘anonymous’’ function
> x = 0:.01:3;
> plot(x,myfcn(x))
> grid
> fzero(myfcn,0.5)
> fzero(myfcn,3)
> [x,y,exitflag,output] = fzero(myfcn,3)
> % now give Newton a spin
> x = 3;
> x = x - myfcn(x)/(1-2*cos(x)) % iterate this line
```

Example

Example

A home buyer can afford monthly payments of at most \$900. What is the maximum interest rate that the buyer can afford to pay on a \$200000 house (after the down) with a 25 year mortgage. The **ordinary annuity equation** is helpful:

$$A = \frac{P}{i}\left(1 - (1+i)^{-n}\right)$$

where A is the mortgage amount, P the monthly payment and i is the interest rate per period over the n payment periods. The unknown is i.

```
> P = 900
> A = 100000
> n = 12*15
> % let's make an anonymous function:
> fcn = @(i) i*A - P*(1 - 1/(1+i)^n)
> r = fzero(fcn,0.01)*12
```