Chapter 16
DESIGN OF RECURSIVE FILTERS USING OPTIMIZATION METHODS
16.4 Quasi-Newton Algorithms
16.5 Minimax Algorithms
16.6 Improved Minimax Algorithms

Copyright © 2005 Andreas Antoniou
Victoria, BC, Canada
Email: aantoniou@ieee.org

July 14, 2018

Introduction

- The Newton algorithm described in the previous presentation is the basis of many other optimization algorithms such as the *Gauss-Newton algorithm* and the family of *conjugate-directions algorithms*.

Introduction

- The Newton algorithm described in the previous presentation is the basis of many other optimization algorithms such as the *Gauss-Newton algorithm* and the family of *conjugate-directions algorithms*.
- These algorithms possess certain advantages for certain applications depending, for example, on the characteristics and size of the problem to be solved.

Introduction

- The Newton algorithm described in the previous presentation is the basis of many other optimization algorithms such as the *Gauss-Newton algorithm* and the family of *conjugate-directions algorithms*.
- These algorithms possess certain advantages for certain applications depending, for example, on the characteristics and size of the problem to be solved.
- A specific family of algorithms derived from the basic Newton algorithm, the family of *quasi-Newton algorithms*, have certain unique features that make them very suitable for the design of recursive digital filters.

Introduction *Cont'd*

- Quasi-Newton algorithms offer a number of important advantages as follows:
  - They do not require the second derivatives of the function.
  - There is no need to invert an $n \times n$ matrix.
  - There is no need to check an $n \times n$ matrix for positive definiteness.
  - Can be used with *inexact line searches* which lead to improved efficiency.
  - They offer fast convergence.
  - Are robust.

Introduction *Cont'd*

- Quasi-Newton algorithms offer a number of important advantages as follows:
  - They do not require the second derivatives of the function.
  - There is no need to invert an $n \times n$ matrix.
  - There is no need to check an $n \times n$ matrix for positive definiteness.
  - Can be used with *inexact line searches* which lead to improved efficiency.
  - They offer fast convergence.
  - Are robust.
- The underlying principle in these algorithms is to generate a matrix $\mathbf{S}$ on the basis of *successive gradient evaluations*, which becomes the inverse Hessian under certain ideal conditions.

Introduction *Cont'd*

- Consider a function of n independent variables, $f(\mathbf{x})$, and let the gradients of $f(\mathbf{x})$ at points $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ be $\mathbf{g}_k$ and $\mathbf{g}_{k+1}$, respectively.

Introduction *Cont'd*

- Consider a function of n independent variables, $f(\mathbf{x})$, and let the gradients of $f(\mathbf{x})$ at points $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ be $\mathbf{g}_k$ and $\mathbf{g}_{k+1}$, respectively.
- The gradient of a function is obviously another function.

  So if we let

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$$

  then the Taylor series gives the elements of $\mathbf{g}_{k+1}$ as

$$g_{(k+1)m} = g_{km} + \sum_{i=1}^{n} \frac{\partial g_{km}}{\partial x_{k\,i}} \delta_{k\,i} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 g_{km}}{\partial x_{k\,i} \partial x_{kj}} \delta_{k\,i} \delta_{kj} + o\left(||\boldsymbol{\delta}||_2^2\right)$$

  for $m = 1, 2, \ldots, n$.

Introduction *Cont'd*

. . .

$$g_{(k+1)m} = g_{km} + \sum_{i=1}^{n} \frac{\partial g_{km}}{\partial x_{k\,i}} \delta_{k\,i} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 g_{km}}{\partial x_{k\,i} \partial x_{kj}} \delta_{k\,i} \delta_{kj} + o\left(||\boldsymbol{\delta}||_2^2\right)$$

- If $f(\mathbf{x})$ is a quadratic function, the second and higher derivatives of $f(\mathbf{x})$ are constant and zero, respectively, and as a result the second and higher derivatives of g_{km} are zero.

Introduction *Cont'd*

...

$$g_{(k+1)m} = g_{km} + \sum_{i=1}^{n} \frac{\partial g_{km}}{\partial x_{k\,i}} \delta_{k\,i} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 g_{km}}{\partial x_{k\,i} \partial x_{kj}} \delta_{k\,i} \delta_{kj} + o\left(||\boldsymbol{\delta}||_2^2\right)$$

- If $f(\mathbf{x})$ is a quadratic function, the second and higher derivatives of $f(\mathbf{x})$ are constant and zero, respectively, and as a result the second and higher derivatives of g_{km} are zero.
- Thus we get

$$g_{(k+1)m} = g_{km} + \sum_{i=1}^{n} \frac{\partial^2 f_k}{\partial x_{k\,i} \partial x_{km}} \delta_{k\,i} \quad \text{for} \quad m = 1, 2, \ldots, n$$

or

$$\mathbf{g}_{(k+1)m} = \mathbf{g}_k + \mathbf{H} \boldsymbol{\delta}_k$$

Introduction *Cont'd*

. . .

$$\mathbf{g}_{(k+1)m} = \mathbf{g}_k + \mathbf{H}\boldsymbol{\delta}_k$$

- The above relation can be expressed as

$$\boldsymbol{\gamma}_k = \mathbf{H}\boldsymbol{\delta}_k$$

where $$\boldsymbol{\delta}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$$

and $$\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$$

Introduction *Cont'd*

. . .

$$\mathbf{g}_{(k+1)m} = \mathbf{g}_k + \mathbf{H}\boldsymbol{\delta}_k$$

- The above relation can be expressed as

$$\boldsymbol{\gamma}_k = \mathbf{H}\boldsymbol{\delta}_k$$

where $\boldsymbol{\delta}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$

and $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$

- If $\boldsymbol{\gamma}_k$ and $\boldsymbol{\delta}_k$ are known, a *certain amount* of information is available about the Hessian is available.

Introduction *Cont'd*

...

$$\gamma_k = \mathbf{H}\boldsymbol{\delta}_k$$

- If we let $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$ be a series of points such that the changes

$$\boldsymbol{\delta}_0 = \mathbf{x}_1 - \mathbf{x}_0,\ \boldsymbol{\delta}_1 = \mathbf{x}_2 - \mathbf{x}_1,\ \ldots,\ \boldsymbol{\delta}_{n-1} = \mathbf{x}_n - \mathbf{x}_{n-1}$$

are *linearly independent*, then

$$\gamma_0 = \mathbf{H}\boldsymbol{\delta}_0,\ \gamma_1 = \mathbf{H}\boldsymbol{\delta}_1,\ \ldots,\ \gamma_{n-1} = \mathbf{H}\boldsymbol{\delta}_{n-1}$$

or

$$\begin{bmatrix}\gamma_0 & \gamma_1 & \ldots & \gamma_{n-1}\end{bmatrix} = \mathbf{H}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}$$

Introduction *Cont'd*

...

$$\begin{bmatrix}\gamma_0 & \gamma_1 & \dots & \gamma_{n-1}\end{bmatrix} = \mathbf{H}\begin{bmatrix}\delta_0 & \delta_1 & \cdots & \delta_{n-1}\end{bmatrix}$$

- Solving for **H**, we get

$$\mathbf{H} = \begin{bmatrix}\gamma_0 & \gamma_1 & \cdots & \gamma_{n-1}\end{bmatrix}\begin{bmatrix}\delta_0 & \delta_1 & \cdots & \delta_{n-1}\end{bmatrix}^{-1}$$

Introduction *Cont'd*

...
$$\begin{bmatrix}\gamma_0 & \gamma_1 & \ldots & \gamma_{n-1}\end{bmatrix} = \mathbf{H}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}$$

- Solving for **H**, we get

$$\mathbf{H} = \begin{bmatrix}\gamma_0 & \gamma_1 & \cdots & \gamma_{n-1}\end{bmatrix}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}^{-1}$$

- The solution exists because the changes in **x** have been assumed to be linearly independent and, therefore, matrix $[\boldsymbol{\delta}_0\ \boldsymbol{\delta}_1\ \cdots\ \boldsymbol{\delta}_{n-1}]$ is nonsingular.

Introduction *Cont'd*

$\cdots$ $$\begin{bmatrix}\boldsymbol{\gamma}_0 & \boldsymbol{\gamma}_1 & \ldots & \boldsymbol{\gamma}_{n-1}\end{bmatrix} = \mathbf{H}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}$$

- Solving for **H**, we get

$$\mathbf{H} = \begin{bmatrix}\boldsymbol{\gamma}_0 & \boldsymbol{\gamma}_1 & \cdots & \boldsymbol{\gamma}_{n-1}\end{bmatrix}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}^{-1}$$

- The solution exists because the changes in **x** have been assumed to be linearly independent and, therefore, matrix $[\boldsymbol{\delta}_0 \; \boldsymbol{\delta}_1 \; \cdots \; \boldsymbol{\delta}_{n-1}]$ is nonsingular.

- In effect, we can *eliminate* the need for the second derivatives by calculating the Hessian using the values of n successive gradients and n successive linearly independent changes in **x**.

Introduction *Cont'd*

. . .
$$\begin{bmatrix}\gamma_0 & \gamma_1 & \dots & \gamma_{n-1}\end{bmatrix} = \mathbf{H}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}$$

- Solving for **H**, we get

$$\mathbf{H} = \begin{bmatrix}\gamma_0 & \gamma_1 & \cdots & \gamma_{n-1}\end{bmatrix}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}^{-1}$$

- The solution exists because the changes in **x** have been assumed to be linearly independent and, therefore, matrix $[\boldsymbol{\delta}_0\ \boldsymbol{\delta}_1\ \cdots\ \boldsymbol{\delta}_{n-1}]$ is nonsingular.
- In effect, we can *eliminate* the need for the second derivatives by calculating the Hessian using the values of n successive gradients and n successive linearly independent changes in **x**.
- This idea is implemented in the next algorithm.

Alternative Implementation of Newton Algorithm

1. Input $\mathbf{x}_{00}$ and ε.

   Input a set of n linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.

   Set $k = 0$.

Alternative Implementation of Newton Algorithm

1. Input $\mathbf{x}_{00}$ and ε.

   Input a set of n linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.

   Set $k = 0$.

2. Compute $\mathbf{g}_{k0}$.

Alternative Implementation of Newton Algorithm

1. Input $\mathbf{x}_{00}$ and ε.
   Input a set of n linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.
   Set $k = 0$.
2. Compute $\mathbf{g}_{k0}$.
3. For $i = 0$ to $n - 1$ do:

Alternative Implementation of Newton Algorithm

1. Input $\mathbf{x}_{00}$ and ε.

   Input a set of n linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.

   Set $k = 0$.
2. Compute $\mathbf{g}_{k0}$.
3. For $i = 0$ to $n - 1$ do:
   a. Set $\mathbf{x}_{k(i+1)} = \mathbf{x}_{k\,i} + \boldsymbol{\delta}_i$.

Alternative Implementation of Newton Algorithm

1. Input $\mathbf{x}_{00}$ and ε.

   Input a set of n linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.

   Set $k = 0$.
2. Compute $\mathbf{g}_{k0}$.
3. For $i = 0$ to $n-1$ do:
   a. Set $\mathbf{x}_{k(i+1)} = \mathbf{x}_{k\,i} + \boldsymbol{\delta}_i$.
   b. Compute $\mathbf{g}_{k(i+1)}$.

Alternative Implementation of Newton Algorithm

```
1. Input $\mathbf{x}_{00}$ and $\varepsilon$.
   Input a set of $n$ linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.
   Set $k = 0$.
2. Compute $\mathbf{g}_{k0}$.
3. For $i = 0$ to $n - 1$ do:
   a. Set $\mathbf{x}_{k(i+1)} = \mathbf{x}_{k\,i} + \boldsymbol{\delta}_i$.
   b. Compute $\mathbf{g}_{k(i+1)}$.
   c. Set $\boldsymbol{\gamma}_{k\,i} = \mathbf{g}_{k(i+1)} - \mathbf{g}_{k\,i}$.
```

Alternative Implementation of Newton Algorithm

1. Input $\mathbf{x}_{00}$ and ε.

   Input a set of n linearly independent vectors $\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{n-1}$.

   Set $k = 0$.

2. Compute $\mathbf{g}_{k0}$.

3. For $i = 0$ to $n-1$ do:

   a. Set $\mathbf{x}_{k(i+1)} = \mathbf{x}_{k\,i} + \boldsymbol{\delta}_i$.

   b. Compute $\mathbf{g}_{k(i+1)}$.

   c. Set $\boldsymbol{\gamma}_{k\,i} = \mathbf{g}_{k(i+1)} - \mathbf{g}_{k\,i}$.

4. Compute

$$\mathbf{H_k} = \begin{bmatrix}\boldsymbol{\gamma}_{k0} & \boldsymbol{\gamma}_{k1} & \cdots & \boldsymbol{\gamma}_{k(n-1)}\end{bmatrix}\begin{bmatrix}\boldsymbol{\delta}_0 & \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_{n-1}\end{bmatrix}^{-1}$$

   If $\mathbf{H}_k$ is not positive definite, force it to become positive definite.

Alternative Implementation of Newton Algorithm *Cont'd*

5. Determine $\mathbf{S}_k = \mathbf{H}_k^{-1}$.

Alternative Implementation of Newton Algorithm *Cont'd*

5. Determine $\mathbf{S}_k = \mathbf{H}_k^{-1}$.
6. Set $\mathbf{d}_k = -\mathbf{S}_k \mathbf{g}_{k0}$ and find α_k, the value of α that minimizes $f(\mathbf{x}_{k0} + \alpha\,\mathbf{d}_k)$, using a line search.

Alternative Implementation of Newton Algorithm *Cont'd*

5. Determine $\mathbf{S}_k = \mathbf{H}_k^{-1}$.
6. Set $\mathbf{d}_k = -\mathbf{S}_k \mathbf{g}_{k0}$ and find α_k, the value of α that minimizes $f(\mathbf{x}_{k0} + \alpha\, \mathbf{d}_k)$, using a line search.
7. Set $\mathbf{x}_{(k+1)0} = \mathbf{x}_{k0} + \alpha_k \mathbf{d}_k$ and compute $f_{(k+1)0} = f(\mathbf{x}_{(k+1)0})$.

Alternative Implementation of Newton Algorithm *Cont'd*

5. Determine $\mathbf{S}_k = \mathbf{H}_k^{-1}$.
6. Set $\mathbf{d}_k = -\mathbf{S}_k \mathbf{g}_{k0}$ and find α_k, the value of α that minimizes $f(\mathbf{x}_{k0} + \alpha\, \mathbf{d}_k)$, using a line search.
7. Set $\mathbf{x}_{(k+1)0} = \mathbf{x}_{k0} + \alpha_k \mathbf{d}_k$ and compute $f_{(k+1)0} = f(\mathbf{x}_{(k+1)0})$.
8. If $\|\alpha_k \mathbf{d}_k\|_2 < \varepsilon$, then output $\breve{\mathbf{x}} = \mathbf{x}_{(k+1)0}$, $f(\breve{\mathbf{x}}) = f_{(k+1)0}$, and stop.

   Otherwise, set $k = k + 1$ and repeat from step 2.

Alternative Implementation of Newton Algorithm *Cont'd*

- The algorithm described is essentially the Newton algorithm, except that the Hessian is calculated by using *n* values of γ and *n* linearly independent changes in **x** instead of using the second derivatives.

Alternative Implementation of Newton Algorithm *Cont'd*

- The algorithm described is essentially the Newton algorithm, except that the Hessian is calculated by using n values of γ and n linearly independent changes in $\mathbf{x}$ instead of using the second derivatives.
- In quasi-Newton algorithms, a matrix **S**, which serves the same purpose as the inverse Hessian, is initially assumed to be the identity matrix, **I**, and in each iteration an appropriate correction is made to **S** which assures that the corrected matrix is a better approximation of the inverse Hessian *while remaining positive definite*.

Alternative Implementation of Newton Algorithm *Cont'd*

- The algorithm described is essentially the Newton algorithm, except that the Hessian is calculated by using *n* values of γ and *n* linearly independent changes in $\mathbf{x}$ instead of using the second derivatives.
- In quasi-Newton algorithms, a matrix **S**, which serves the same purpose as the inverse Hessian, is initially assumed to be the identity matrix, **I**, and in each iteration an appropriate correction is made to **S** which assures that the corrected matrix is a better approximation of the inverse Hessian *while remaining positive definite*.
- A generic quasi-Newton algorithm is described next.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k\mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k\mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.
3. Set $\boldsymbol{\delta}_k = \alpha_k\mathbf{d}_k$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$, and compute $f_{k+1} = f(\mathbf{x}_{k+1})$.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k\mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.
3. Set $\boldsymbol{\delta}_k = \alpha\,_k\mathbf{d}_k$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$, and compute $f_{k+1} = f(\mathbf{x}_{k+1})$.
4. If $\|\boldsymbol{\delta}_k\|_2 < \varepsilon$, then output $\breve{\mathbf{x}} = \mathbf{x}_{k+1}$, $f(\breve{\mathbf{x}}) = f_{k+1}$ and stop.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k\mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.
3. Set $\boldsymbol{\delta}_k = \alpha_k\mathbf{d}_k$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$, and compute $f_{k+1} = f(\mathbf{x}_{k+1})$.
4. If $\|\boldsymbol{\delta}_k\|_2 < \varepsilon$, then output $\breve{\mathbf{x}} = \mathbf{x}_{k+1}$, $f(\breve{\mathbf{x}}) = f_{k+1}$ and stop.
5. Compute $\mathbf{g}_{k+1}$ and set $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k\mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.
3. Set $\boldsymbol{\delta}_k = \alpha_k\mathbf{d}_k$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$, and compute $f_{k+1} = f(\mathbf{x}_{k+1})$.
4. If $\|\boldsymbol{\delta}_k\|_2 < \varepsilon$, then output $\breve{\mathbf{x}} = \mathbf{x}_{k+1}$, $f(\breve{\mathbf{x}}) = f_{k+1}$ and stop.
5. Compute $\mathbf{g}_{k+1}$ and set $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.
6. Compute $\mathbf{S}_{k+1} = \mathbf{S}_k + \mathbf{C}_k$.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k\mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.
3. Set $\boldsymbol{\delta}_k = \alpha_k\mathbf{d}_k$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$, and compute $f_{k+1} = f(\mathbf{x}_{k+1})$.
4. If $\|\boldsymbol{\delta}_k\|_2 < \varepsilon$, then output $\breve{\mathbf{x}} = \mathbf{x}_{k+1}$, $f(\breve{\mathbf{x}}) = f_{k+1}$ and stop.
5. Compute $\mathbf{g}_{k+1}$ and set $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.
6. Compute $\mathbf{S}_{k+1} = \mathbf{S}_k + \mathbf{C}_k$.
7. Check $\mathbf{S}_{k+1}$ for positive definiteness and if it is found to be nonpositive definite force it to become positive definite.

Quasi-Newton Algorithms

1. Input $\mathbf{x}_0$ and ε. Set $\mathbf{S}_0 = \mathbf{I}_n$ and $k = 0$. Compute $\mathbf{g}_0$.
2. Set $\mathbf{d}_k = -\mathbf{S}_k \mathbf{g}_k$ and find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$, using a line search.
3. Set $\boldsymbol{\delta}_k = \alpha_k \mathbf{d}_k$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k$, and compute $f_{k+1} = f(\mathbf{x}_{k+1})$.
4. If $\|\boldsymbol{\delta}_k\|_2 < \varepsilon$, then output $\breve{\mathbf{x}} = \mathbf{x}_{k+1}$, $f(\breve{\mathbf{x}}) = f_{k+1}$ and stop.
5. Compute $\mathbf{g}_{k+1}$ and set $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.
6. Compute $\mathbf{S}_{k+1} = \mathbf{S}_k + \mathbf{C}_k$.
7. Check $\mathbf{S}_{k+1}$ for positive definiteness and if it is found to be nonpositive definite force it to become positive definite.
8. Set $k = k + 1$ and go to step 2.

Quasi-Newton Algorithms *Cont'd*

- In Step 2 a direction vector is generated which is analogous to the Newton direction.

Quasi-Newton Algorithms *Cont'd*

- In Step 2 a direction vector is generated which is analogous to the Newton direction.
- In Step 3 the function is minimized with respect to the line defined by the direction vector as usual.

Quasi-Newton Algorithms *Cont'd*

- In Step 2 a direction vector is generated which is analogous to the Newton direction.
- In Step 3 the function is minimized with respect to the line defined by the direction vector as usual.
- In Step 5 the next gradient and the difference between the next and the present gradient, i.e., γ_k, are calculated.

Quasi-Newton Algorithms *Cont'd*

- In Step 2 a direction vector is generated which is analogous to the Newton direction.
- In Step 3 the function is minimized with respect to the line defined by the direction vector as usual.
- In Step 5 the next gradient and the difference between the next and the present gradient, i.e., γ_k, are calculated.
- In Step 6, a correction is applied to the current **S** matrix to obtain the next **S** matrix.

Quasi-Newton Algorithms *Cont'd*

- Several quasi-Newton algorithms are available, which differ from one another in the formula used to update matrix $\mathbf{S}_{k+1}$ in Step 6 of the generic quasi-Newton algorithm presented.

Quasi-Newton Algorithms *Cont'd*

- Several quasi-Newton algorithms are available, which differ from one another in the formula used to update matrix $\mathbf{S}_{k+1}$ in Step 6 of the generic quasi-Newton algorithm presented.

- The two most important updating formulas are the *Davidon-Fletcher-Powell* (DFP) formula in which

$$\mathbf{S}_{k+1} = \mathbf{S}_k + \frac{\boldsymbol{\delta}_k \boldsymbol{\delta}_k^T}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k} - \frac{\mathbf{S}_k \boldsymbol{\gamma}_k \boldsymbol{\gamma}_k^T \mathbf{S}_k}{\boldsymbol{\gamma}_k^T \mathbf{S}_k \boldsymbol{\gamma}_k}$$

  and the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) formula in which

$$\mathbf{S}_{k+1} = \mathbf{S}_k + \left(1 + \frac{\boldsymbol{\gamma}_k^T \mathbf{S}_k \boldsymbol{\gamma}_k}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k}\right) \frac{\boldsymbol{\delta}_k \boldsymbol{\delta}_k^T}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k} - \frac{(\boldsymbol{\delta}_k \boldsymbol{\gamma}_k^T \mathbf{S}_k + \mathbf{S}_k \boldsymbol{\gamma}_k \boldsymbol{\delta}_k^T)}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k}$$

Quasi-Newton Algorithms *Cont'd*

The DFP and BFGS algorithms have the following important advantages relative to other quasi-Newton algorithms:

- For a *quadratic problem*, matrix $\mathbf{S}$ becomes the inverse Hessian in n iterations.

  Also matrix $\mathbf{S}_{k+1}$ is positive definite, if $\mathbf{S}_k$ is positive definite.

  Thus if we start with $\mathbf{S}_0 = \mathbf{I}$, the positive definiteness of $\mathbf{S}$ is assured throughout the optimization.

Quasi-Newton Algorithms *Cont'd*

The DFP and BFGS algorithms have the following important advantages relative to other quasi-Newton algorithms:

- For a *quadratic problem*, matrix $\mathbf{S}$ becomes the inverse Hessian in n iterations.

  Also matrix $\mathbf{S}_{k+1}$ is positive definite, if $\mathbf{S}_k$ is positive definite.

  Thus if we start with $\mathbf{S}_0 = \mathbf{I}$, the positive definiteness of $\mathbf{S}$ is assured throughout the optimization.

- For a *general nonquadratic problem*, matrix $\mathbf{S}_{k+1}$ is positive definite, if $\mathbf{S}_k$ is positive definite *provided that an exact line search is used*.

Quasi-Newton Algorithms *Cont'd*

- The positive definiteness of $\mathbf{S}_{k+1}$ can also be assured if an inexact line search is used except that *a scalar parameter associated with these algorithms has to be checked*.

  If it becomes negative, $\mathbf{S}_{k+1}$ has to be forced to become positive definite, e.g., replaced by the identity matrix.

Quasi-Newton Algorithms *Cont'd*

- The positive definiteness of $\mathbf{S}_{k+1}$ can also be assured if an inexact line search is used except that *a scalar parameter associated with these algorithms has to be checked*.

  If it becomes negative, $\mathbf{S}_{k+1}$ has to be forced to become positive definite, e.g., replaced by the identity matrix.

- The DFP and BFGS formulas have very similar mathematical properties and, in fact, they are related through a principle known as *mathematical duality*.

  However, experiments carried out by Fletcher have shown that the use of the BFGS formula tends to lead to reduced computation.

  The reason has not as yet been identified.

Quasi-Newton Algorithms *Cont'd*

- An important advantage of quasi-Newton algorithms relates to the fact that they can be used with inexact line searches.

Quasi-Newton Algorithms *Cont'd*

- An important advantage of quasi-Newton algorithms relates to the fact that they can be used with inexact line searches.
- These are low precision line searches that have the unusual property that their *inexactness does not affect the convergence properties* of the quasi-Newton algorithm.

  Thus they lead to *reduced computational effort*.

Fletcher's Inexact Line Search

- In each iteration of a typical optimization algorithm, a change is applied to the most recent value of $\mathbf{x}_k$ by applying a change $\alpha\,\mathbf{d}_k$ such that
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\,\mathbf{d}_k$$
where $\mathbf{d}_k$ is a descent direction.

Fletcher's Inexact Line Search

- In each iteration of a typical optimization algorithm, a change is applied to the most recent value of $\mathbf{x}_k$ by applying a change $\alpha\,\mathbf{d}_k$ such that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\,\mathbf{d}_k$$

  where $\mathbf{d}_k$ is a descent direction.

- Then the function

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$$

  is minimized with respect to parameter α.

Fletcher's Inexact Line Search

- In each iteration of a typical optimization algorithm, a change is applied to the most recent value of $\mathbf{x}_k$ by applying a change $\alpha\,\mathbf{d}_k$ such that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\,\mathbf{d}_k$$

  where $\mathbf{d}_k$ is a descent direction.

- Then the function

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$$

  is minimized with respect to parameter α.

- In an exact line search, the function is continuously reduced until the *precise value* of α that minimizes the function, $\breve{\alpha}$, is obtained.

Fletcher's Inexact Line Search

- In each iteration of a typical optimization algorithm, a change is applied to the most recent value of $\mathbf{x}_k$ by applying a change $\alpha\,\mathbf{d}_k$ such that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\,\mathbf{d}_k$$

  where $\mathbf{d}_k$ is a descent direction.

- Then the function

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$$

  is minimized with respect to parameter α.

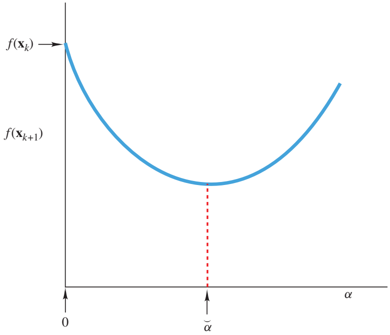
- In an exact line search, the function is continuously reduced until the *precise value* of α that minimizes the function, $\breve{\alpha}$, is obtained.

- In an inexact line search, the function is continuously reduced until a value of α that falls in a specified *parametric interval* is obtained.

Fletcher's Inexact Line Search *Cont'd*

- Let us assume that function $f(\mathbf{x}_{k+1})$ has a unique minimum.

  Since $\mathbf{d}_k$ is a descent direction, the function will decrease as we increase α and it follows that the minimum will be located over some positive range of α as shown in the figure.

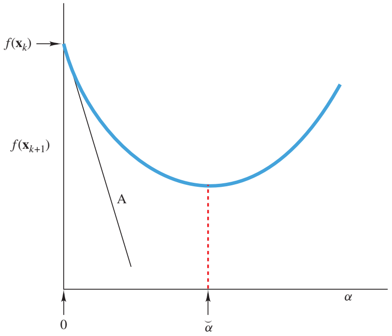


Fletcher's Inexact Line Search *Cont'd*

- The linear approximation of the Taylor series for $f(\mathbf{x}_{k+1})$ is of the form

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \alpha\, \mathbf{g}_k^T \mathbf{d}_k \quad \text{where} \quad \mathbf{g}_k^T \mathbf{d}_k = \left. \frac{df(\mathbf{x}_k + \alpha\, \mathbf{d}_k)}{d\alpha} \right|_{\alpha=0}$$

See line A in the figure.



Fletcher's Inexact Line Search *Cont'd*

...

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \alpha\,\mathbf{g}_k^T\mathbf{d}_k$$

- Similarly, the equation

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \rho\,\alpha\,\mathbf{g}_k^T\mathbf{d}_k$$

  where $0 \le \rho \le 0.5$ represents a line whose slope ranges from 0 to $0.5\mathbf{g}_k^T\mathbf{d}_k$ depending on the value of ρ.

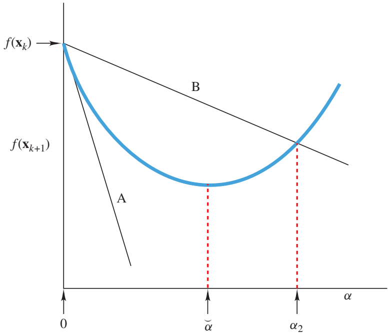
  See line B in the next slide.

Fletcher's Inexact Line Search *Cont'd*

. . .

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \rho\,\alpha\,\mathbf{g}_k^T\mathbf{d}_k$$

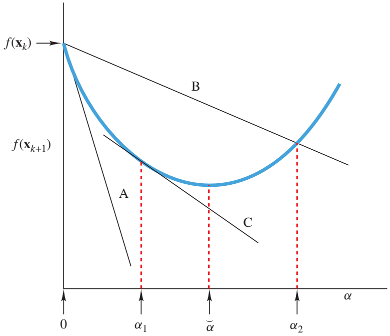
- Let us assume that line B intersects the curve at $\alpha = \alpha_2$ as shown.



Fletcher's Inexact Line Search *Cont'd*

- Now consider the equation

$$\mathbf{g}_{k+1}^T \mathbf{d}_k = \sigma\, \mathbf{g}_k^T \mathbf{d}_k \quad \text{where} \quad 0 < \sigma < 1 \text{ and } \sigma \geq \rho$$

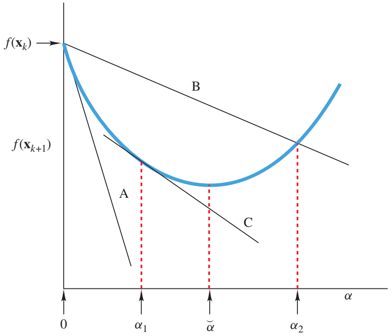


Fletcher's Inexact Line Search *Cont'd*

- Now consider the equation

$$\mathbf{g}_{k+1}^T \mathbf{d}_k = \sigma\, \mathbf{g}_k^T \mathbf{d}_k \quad \text{where} \;\; 0 < \sigma < 1 \text{ and } \sigma \geq \rho$$

- This equation defines some point $\alpha = \alpha_1$ at which the first derivative of $f(\mathbf{x}_{k+1})$ is equal to a fraction of the derivative of the function at $\alpha = 0$ as shown by line C in the figure.

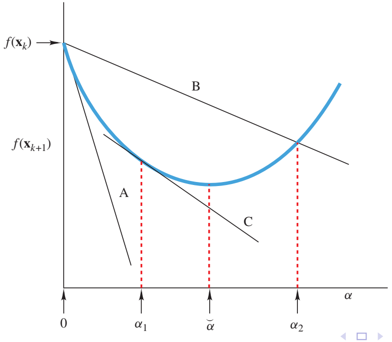


Fletcher's Inexact Line Search *Cont'd*

- Thus the equations

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \rho\,\alpha\,\mathbf{g}_k^T\mathbf{d}_k$$
$$\mathbf{g}_{k+1}^T\mathbf{d}_k = \sigma\,\mathbf{g}_k^T\mathbf{d}_k$$

  define an interval $[\alpha_1, \alpha_2]$ that would *bracket* the minimum point.



Fletcher's Inexact Line Search *Cont'd*

- By choosing parameters, ρ and σ, the width of interval $[\alpha_1, \alpha_2]$ can be controlled.

Fletcher's Inexact Line Search *Cont'd*

- By choosing parameters, ρ and σ, the width of interval $[\alpha_1, \alpha_2]$ can be controlled.
- If a value of α in the range $[\alpha_1, \alpha_2]$, say, α_0, is somehow obtained, the reduction in the function may be deemed to be sufficient and the line search can be terminated.

Fletcher's Inexact Line Search *Cont'd*

- By choosing parameters, ρ and σ, the width of interval $[\alpha_1, \alpha_2]$ can be controlled.
- If a value of α in the range $[\alpha_1, \alpha_2]$, say, α_0, is somehow obtained, the reduction in the function may be deemed to be sufficient and the line search can be terminated.
- Value α_0 can be checked by checking whether the inequalities

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$

and

$$\mathbf{g}_{k+1}^T \mathbf{d}_k \geq \sigma\, \mathbf{g}_k^T \mathbf{d}_k$$

are satisfied at point $\alpha = \alpha_0$.

If they are satisfied, then $\alpha_1 \leq \alpha \leq \alpha_2$.

Fletcher's Inexact Line Search *Cont'd*

- By choosing parameters, ρ and σ, the width of interval $[\alpha_1, \alpha_2]$ can be controlled.

- If a value of α in the range $[\alpha_1, \alpha_2]$, say, α_0, is somehow obtained, the reduction in the function may be deemed to be sufficient and the line search can be terminated.

- Value α_0 can be checked by checking whether the inequalities

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$

  and

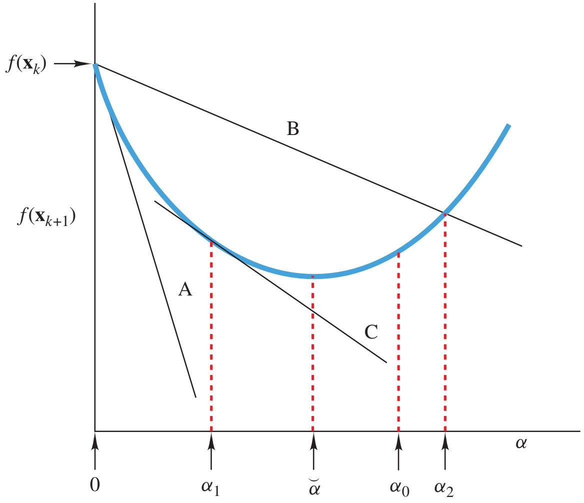
$$\mathbf{g}_{k+1}^T \mathbf{d}_k \geq \sigma\, \mathbf{g}_k^T \mathbf{d}_k$$

  are satisfied at point $\alpha = \alpha_0$.

  If they are satisfied, then $\alpha_1 \leq \alpha \leq \alpha_2$.

- These inequalities are known as the *Goldstein inequalities*.

Fletcher's Inexact Line Search *Cont'd*

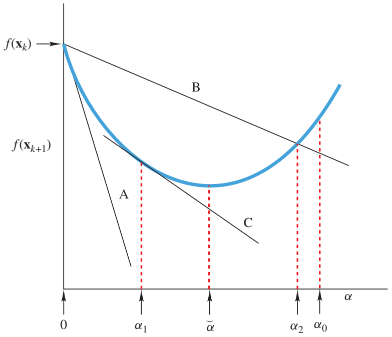


Fletcher's Inexact Line Search *Cont'd*

...

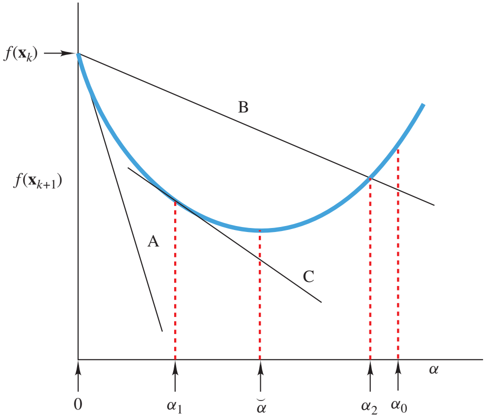
$$f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$

- If the *first* Goldstein inequality is not satisfied at $\alpha = \alpha_0$, then obviously $\alpha_0 > \alpha_2$ as shown.



Fletcher's Inexact Line Search *Cont'd*

- In such a case, a better value of α can be deduced by applying *interpolation*.

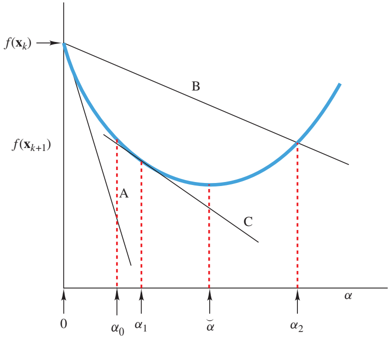


Fletcher's Inexact Line Search *Cont'd*

. . .

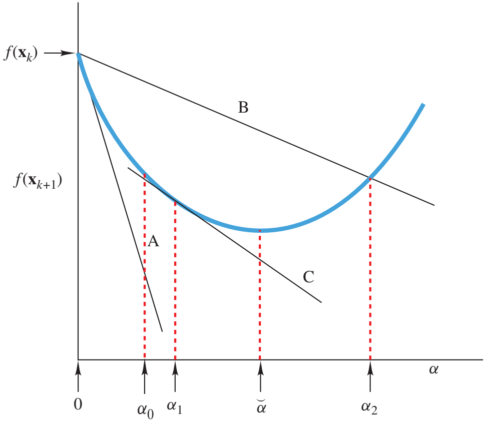
$$\mathbf{g}_{k+1}^T \mathbf{d}_k \geq \sigma\, \mathbf{g}_k^T \mathbf{d}_k$$

- If the *second* Goldstein inequality is not satisfied at $\alpha = \alpha_0$, then obviously $\alpha_0 < \alpha_1$ as shown.



Fletcher's Inexact Line Search *Cont'd*

- In such a case, a better value of α can be deduced by applying *extrapolation*.



Fletcher's Inexact Line Search *Cont'd*

- *Interpolation and extrapolation formulas* for use in Fletcher's inexact line search can be readily deduced by assuming a *quadratic representation* for $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$ and then finding the location of the minimum point.

Fletcher's Inexact Line Search *Cont'd*

- *Interpolation and extrapolation formulas* for use in Fletcher's inexact line search can be readily deduced by assuming a *quadratic representation* for $f(\mathbf{x}_k + \alpha\,\mathbf{d}_k)$ and then finding the location of the minimum point.

- To achieve efficiency, the interpolation information available for the implementation of Goldstein's inequalities should as far as possible be used.

Fletcher's Inexact Line Search *Cont'd*

- The interpolation and extrapolation formulas used by Fletcher are as follows:

  For $\alpha_0 > \alpha_2$

$$\breve{\alpha}_0 = \alpha_L + \frac{(\alpha_0 - \alpha_L)^2 f'_L}{2[f_L - f_0 + (\alpha_0 - \alpha_L) f'_L]}$$

  For $\alpha_0 < \alpha_1$

$$\breve{\alpha}_0 = \alpha_0 + \frac{(\alpha_0 - \alpha_L) f'_0}{(f'_L - f'_0)}$$

  where

$$\begin{aligned} f_L &= f(\mathbf{x}_k + \alpha_L \mathbf{d}_k) \quad \text{and} \quad f'_L = f'(\mathbf{x}_k + \alpha_L \mathbf{d}_k) = \mathbf{g}(\mathbf{x}_k + \alpha_L \mathbf{d}_k)^T \mathbf{d}_k \\ f_0 &= f(\mathbf{x}_k + \alpha_0 \mathbf{d}_k) \quad \text{and} \quad f'_0 = f'(\mathbf{x}_k + \alpha_0 \mathbf{d}_k) = \mathbf{g}(\mathbf{x}_k + \alpha_0 \mathbf{d}_k)^T \mathbf{d}_k \end{aligned}$$

Fletcher's Inexact Line Search *Cont'd*

A *basic* inexact line search is as follows:

1. Obtain an estimate of the minimum point α_0 in some way.

Fletcher's Inexact Line Search *Cont'd*

A *basic* inexact line search is as follows:

1. Obtain an estimate of the minimum point α_0 in some way.

2. Check the *first Goldstein* condition

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$

Fletcher's Inexact Line Search *Cont'd*

A *basic* inexact line search is as follows:

1. Obtain an estimate of the minimum point α_0 in some way.
2. Check the *first Goldstein* condition

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$

3. If the condition in Step 2 is not satisfied, carry out an interpolation and repeat from Step 2.

Fletcher's Inexact Line Search *Cont'd*

A *basic* inexact line search is as follows:

1. Obtain an estimate of the minimum point α_0 in some way.
2. Check the *first Goldstein* condition
$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$
3. If the condition in Step 2 is not satisfied, carry out an interpolation and repeat from Step 2.
4. Check the *second Goldstein* condition
$$\mathbf{g}_{k+1}^T \mathbf{d}_k \geq \sigma\, \mathbf{g}_k^T \mathbf{d}_k$$

Fletcher's Inexact Line Search *Cont'd*

A *basic* inexact line search is as follows:

1. Obtain an estimate of the minimum point α_0 in some way.
2. Check the *first Goldstein* condition

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$

3. If the condition in Step 2 is not satisfied, carry out an interpolation and repeat from Step 2.
4. Check the *second Goldstein* condition

$$\mathbf{g}_{k+1}^T \mathbf{d}_k \geq \sigma\, \mathbf{g}_k^T \mathbf{d}_k$$

5. If the condition in Step 4 is not satisfied, carry out an extrapolation and repeat from Step 2.

Fletcher's Inexact Line Search *Cont'd*

A *basic* inexact line search is as follows:

1. Obtain an estimate of the minimum point α_0 in some way.
2. Check the *first Goldstein* condition
$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \rho\,\alpha \mathbf{g}_k^T \mathbf{d}_k$$
3. If the condition in Step 2 is not satisfied, carry out an interpolation and repeat from Step 2.
4. Check the *second Goldstein* condition
$$\mathbf{g}_{k+1}^T \mathbf{d}_k \geq \sigma\, \mathbf{g}_k^T \mathbf{d}_k$$
5. If the condition in Step 4 is not satisfied, carry out an extrapolation and repeat from Step 2.
6. Output the most recent $\breve{\alpha}_0$ and the function value $f(\breve{\alpha}_0)$, and stop.

Fletcher's Inexact Line Search *Cont'd*

- A *practical* inexact line-search algorithm would need to initialize σ and ρ and some other parameters that are required.

Fletcher's Inexact Line Search *Cont'd*

- A *practical* inexact line-search algorithm would need to initialize σ and ρ and some other parameters that are required.
- It should generate the information required by the Goldstein conditions and the interpolation and extrapolation formulas.

Fletcher's Inexact Line Search *Cont'd*

- A *practical* inexact line-search algorithm would need to initialize σ and ρ and some other parameters that are required.
- It should generate the information required by the Goldstein conditions and the interpolation and extrapolation formulas.
- In addition, it should deal with *unusual circumstances* that may arise in general nonquadratic optimization problems.

Fletcher's Inexact Line Search *Cont'd*

- A *practical* inexact line-search algorithm would need to initialize σ and ρ and some other parameters that are required.
- It should generate the information required by the Goldstein conditions and the interpolation and extrapolation formulas.
- In addition, it should deal with *unusual circumstances* that may arise in general nonquadratic optimization problems.

  For example, an extrapolation could in principle locate *a maximum point at some negative value of α* or it may generate *a positive value larger than the most recent value of $\breve{\alpha}_0$ generated by the previous interpolation*.

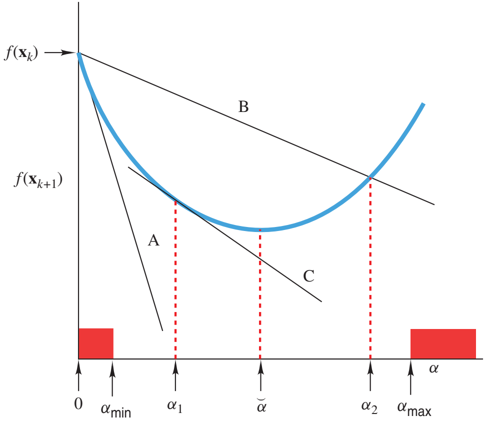
Fletcher's Inexact Line Search *Cont'd*

- A *practical* inexact line-search algorithm would need to initialize σ and ρ and some other parameters that are required.
- It should generate the information required by the Goldstein conditions and the interpolation and extrapolation formulas.
- In addition, it should deal with *unusual circumstances* that may arise in general nonquadratic optimization problems.

  For example, an extrapolation could in principle locate *a maximum point at some negative value of* α or it may generate *a positive value larger than the most recent value of* $\breve{\alpha}_0$ *generated by the previous interpolation*.

  These are all *unreasonable values* for $\breve{\alpha}_0$ and should be ignored. This can be achieved by ensuring that $\alpha_{\text{min}} < \breve{\alpha}_0 < \alpha_{\text{max}}$ where α_{min} and α_{max} are the values of $\breve{\alpha}_0$ predicted by the most recent extrapolation and interpolation, respectively.

Fletcher’s Inexact Line Search *Cont’d*



Fletcher's Inexact Line Search *Cont'd*

- If during an interpolation or extrapolation a value of $\breve{\alpha}_0$ is obtained in the red regions shown in the previous figure, it is replaced by a more reasonable value.

Fletcher's Inexact Line Search *Cont'd*

- If during an interpolation or extrapolation a value of $\breve{\alpha}_0$ is obtained in the red regions shown in the previous figure, it is replaced by a more reasonable value.
- A *practical implementation* of Fletcher's inexact line search as well as relevant details can be found in the textbook.

Choice between L_2 and L_∞ Solutions

- L_2 solutions are easier to obtain and filters based on these solutions will reject *more signal power in stopbands*.

  However, the passband error tends to have *large peaks* near passband edges, which are undesirable in practice.

Choice between L_2 and L_∞ Solutions

- L_2 solutions are easier to obtain and filters based on these solutions will reject *more signal power in stopbands*.

  However, the passband error tends to have *large peaks* near passband edges, which are undesirable in practice.

- L_∞ solutions are more difficult to obtain because they require the application of *minimax algorithms* which entail much more computation.

  However, they offer the advantage that the approximation error tends to be uniformly distributed in passbands, i.e., they tend to yield *equiripple solutions* such as those obtained with the elliptic approximation.

Minimax Algorithms

- Minimax algorithms are essentially sequential algorithms that involve a *series* of unconstrained optimizations.

Minimax Algorithms

- Minimax algorithms are essentially sequential algorithms that involve a *series* of unconstrained optimizations.
- A representative algorithm of this class is the so-called *least-pth algorithm*.

Least-pth Minimax Algorithm

- Since

$$\lim_{p\to\infty} L_p(\mathbf{x}) = L_\infty(\mathbf{x}) = \widehat{E}(\mathbf{x}) \lim_{p\to\infty} \left\{ \sum_{i=1}^{K} \left[\frac{|e_i(\mathbf{x})|}{\widehat{E}(\mathbf{x})} \right]^p \right\}^{1/p}$$

where

$$\widehat{E}(\mathbf{x}) = \underset{1 \le i \le K}{max} \ |e_i(\mathbf{x})|$$

it should be possible, in theory, to solve the minimax problem by minimizing the L_p norm for some large value of p, say, 128.

Least-pth Minimax Algorithm

- Since

$$\lim_{p\to\infty} L_p(\mathbf{x}) = L_\infty(\mathbf{x}) = \widehat{E}(\mathbf{x}) \lim_{p\to\infty} \left\{ \sum_{i=1}^{K} \left[\frac{|e_i(\mathbf{x})|}{\widehat{E}(\mathbf{x})} \right]^p \right\}^{1/p}$$

where

$$\widehat{E}(\mathbf{x}) = \max_{1 \le i \le K} |e_i(\mathbf{x})|$$

it should be possible, in theory, to solve the minimax problem by minimizing the L_p norm for some large value of p, say, 128.

- Unfortunately, *that does not work out well* in practice because the L_∞ norm tends to be badly ill-conditioned.

Least-pth Minimax Algorithm *Cont'd*

- It turns out that the objective function has many valleys and ridges as well as multiple minima and locating a *good minimum point* would not be easy.

  To visualize the situation, think of a problem that involves only two independent variables and think of the Rocky Mountains.

Least-pth Minimax Algorithm *Cont'd*

- It turns out that the objective function has many valleys and ridges as well as multiple minima and locating a *good minimum point* would not be easy.

  To visualize the situation, think of a problem that involves only two independent variables and think of the Rocky Mountains.

- What the least-pth algorithm does is to obtain a solution for the L_2 norm, which is easy to obtain since the *problem is well behaved*.

Least-pth Minimax Algorithm *Cont'd*

- Then the L_4 norm is minimized using the solution obtained for the L_2 norm as the initialization.

Least-pth Minimax Algorithm *Cont'd*

- Then the L_4 norm is minimized using the solution obtained for the L_2 norm as the initialization.
- Since the L_4 problem is *similar* to the L_2 problem, the solution of the L_4 problem is most likely to be located in the same locale of the parameter space as the solution of the L_2 problem and, thus, it would be relatively easy to obtain.

Least-*p*th Minimax Algorithm *Cont'd*

- Then the L_4 norm is minimized using the solution obtained for the L_2 norm as the initialization.

- Since the L_4 problem is *similar* to the L_2 problem, the solution of the L_4 problem is most likely to be located in the same locale of the parameter space as the solution of the L_2 problem and, thus, it would be relatively easy to obtain.

- Similarly, the solution of the L_8 problem can more easily be obtained by using the solution of the L_4 problem as the initialization, and so on.

Least-pth Minimax Algorithm *Cont'd*

- Then the L_4 norm is minimized using the solution obtained for the L_2 norm as the initialization.

- Since the L_4 problem is *similar* to the L_2 problem, the solution of the L_4 problem is most likely to be located in the same locale of the parameter space as the solution of the L_2 problem and, thus, it would be relatively easy to obtain.

- Similarly, the solution of the L_8 problem can more easily be obtained by using the solution of the L_4 problem as the initialization, and so on.

- Is effect, starting with the L_2 norm, the problem is *repeatedly* modified a *little* by increasing the value of p and then solving the problem again using the previous solution as initialization.

  This *sequential technique* enables the algorithm to locate a *good* minimum point.

Least-pth Minimax Algorithm *Cont'd*

1. Input $\breve{\mathbf{x}}_0$ and ε_1 and et $k = 1, p = 2, \mu = 2, \widehat{E}_0 = 10^{99}$.

Least-pth Minimax Algorithm *Cont'd*

1. Input $\breve{\mathbf{x}}_0$ and ε_1 and et $k = 1, p = 2, \mu = 2, \widehat{E}_0 = 10^{99}$.
2. Initialize frequencies $\omega_1, \omega_2, \ldots, \omega_K$.

Least-pth Minimax Algorithm *Cont'd*

1. Input $\breve{\mathbf{x}}_0$ and ε_1 and et $k = 1, p = 2, \mu = 2, \widehat{E}_0 = 10^{99}$.
2. Initialize frequencies $\omega_1, \omega_2, \ldots, \omega_K$.
3. Using $\breve{\mathbf{x}}_{k-1}$ as initial value, minimize

$$\Psi_k(\mathbf{x}) = \widehat{E}(\mathbf{x}) \left\{ \sum_{i=1}^{K} \left[\frac{|e_i(\mathbf{x})|}{\widehat{E}(\mathbf{x})} \right]^p \right\}^{1/p}$$

where

$$\widehat{E}(\mathbf{x}) = \max_{1 \le i \le K} |e_i(\mathbf{x})|$$

with respect to $\mathbf{x}$, to obtain $\breve{\mathbf{x}}_k$. Set $\widehat{E}_k = \widehat{E}(\breve{\mathbf{x}})$.

Least-pth Minimax Algorithm *Cont'd*

1. Input $\breve{\mathbf{x}}_0$ and ε_1 and et $k = 1, p = 2, \mu = 2, \ \widehat{E}_0 = 10^{99}$.
2. Initialize frequencies $\omega_1, \omega_2, \ldots, \omega_K$.
3. Using $\breve{\mathbf{x}}_{k-1}$ as initial value, minimize

$$\Psi_k(\mathbf{x}) = \widehat{E}(\mathbf{x}) \left\{ \sum_{i=1}^{K} \left[\frac{|e_i(\mathbf{x})|}{\widehat{E}(\mathbf{x})} \right]^p \right\}^{1/p}$$

   where

$$\widehat{E}(\mathbf{x}) = \max_{1 \leq i \leq K} |e_i(\mathbf{x})|$$

   with respect to $\mathbf{x}$, to obtain $\breve{\mathbf{x}}_k$. Set $\widehat{E}_k = \widehat{E}(\breve{\mathbf{x}})$.

4. If $|\widehat{E}_{k-1} - \widehat{E}_k| < \varepsilon_1$, then output $\breve{\mathbf{x}}_k$ and $\widehat{E}_k$, and stop. Otherwise, set $p = \mu p, k = k + 1$ and go to step 3.

Charalambous Minimax Algorithm

- Another popular minimax algorithm is one due to Charalambous.

Charalambous Minimax Algorithm

- Another popular minimax algorithm is one due to Charalambous.
- Like the least-pth algorithm, the Charalambous algorithm is a sequential optimization algorithm.

Charalambous Minimax Algorithm

- Another popular minimax algorithm is one due to Charalambous.
- Like the least-*p*th algorithm, the Charalambous algorithm is a sequential optimization algorithm.
- It involves minimizing a function of the form

$$\Psi(\mathbf{x}, \boldsymbol{\lambda}, \xi) = \sum_{i \in I_1} \frac{1}{2} \lambda_i [\phi_i(\mathbf{x}, \xi)]^2 + \sum_{i \in I_2} \frac{1}{2} [\phi_i(\mathbf{x}, \xi)]^2$$

where ξ and λ_i for $i = 1, 2, \ldots, K$ are constants and

$$\begin{aligned} \phi_i(\mathbf{x}, \xi) &= |e_i(\mathbf{x})| - \xi \\ I_1 &= \{i : \phi_i(\mathbf{x}, \xi) > 0 \text{ and } \lambda_i > 0\} \\ I_2 &= \{i : \phi_i(\mathbf{x}, \xi) > 0 \text{ and } \lambda_i = 0\} \end{aligned}$$

Charalambous Minimax Algorithm *Cont'd*

...

$$\Psi(\mathbf{x}, \boldsymbol{\lambda}, \xi) = \sum_{i \in I_1} \frac{1}{2}\lambda_i[\phi_i(\mathbf{x}, \xi)]^2 + \sum_{i \in I_2} \frac{1}{2}[\phi_i(\mathbf{x}, \xi)]^2$$

- Since the objective function comprises sums of squares of function $\phi_i(\mathbf{x}, \xi)$, ill-conditioning is less likely to arise in this algorithm.

Charalambous Minimax Algorithm *Cont'd*

...

$$\Psi(\mathbf{x}, \boldsymbol{\lambda}, \xi) = \sum_{i \in I_1} \frac{1}{2} \lambda_i [\phi_i(\mathbf{x}, \xi)]^2 + \sum_{i \in I_2} \frac{1}{2} [\phi_i(\mathbf{x}, \xi)]^2$$

- Since the objective function comprises sums of squares of function $\phi_i(\mathbf{x}, \xi)$, ill-conditioning is less likely to arise in this algorithm.

- Note that *i*th term $\lambda_i[\phi_i(\mathbf{x}, \xi)]^2$ is *dropped* from the first summation if λ_i becomes negative.

  This tends to reduce the amount of computation significantly and renders the Charalambous algorithm *more efficient* than the least-*p*th algorithm.

Charalambous Minimax Algorithm *Cont'd*

. . .

$$\Psi(\mathbf{x}, \boldsymbol{\lambda}, \xi) = \sum_{i \in I_1} \frac{1}{2} \lambda_i [\phi_i(\mathbf{x}, \xi)]^2 + \sum_{i \in I_2} \frac{1}{2} [\phi_i(\mathbf{x}, \xi)]^2$$

- Since the objective function comprises sums of squares of function $\phi_i(\mathbf{x}, \xi)$, ill-conditioning is less likely to arise in this algorithm.

- Note that ith term $\lambda_i[\phi_i(\mathbf{x}, \xi)]^2$ is *dropped* from the first summation if λ_i becomes negative.

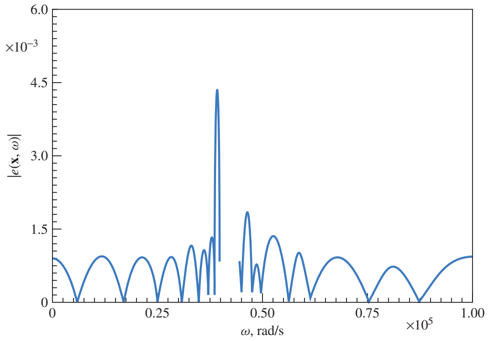
  This tends to reduce the amount of computation significantly and renders the Charalambous algorithm *more efficient* than the least-pth algorithm.

  (See Chap. 16 and references at the end of the chapter for details.)

Improved Minimax Algorithms

- To achieve good results in the minimax algorithms described, the sampling of $e(\mathbf{x}, \omega)$ with respect to ω *must be dense*.

  Otherwise, the error function may develop *spikes* in the intervals between sample points during the minimization, usually near band edges, as shown.



Improved Minimax Algorithms *Cont'd*

- This problem is usually overcome by using a fairly large number of sample points of the order of 5 to 10 times the number of variables.

  For example, if an eighth-order digital filter is to be designed,

Improved Minimax Algorithms *Cont'd*

- This problem is usually overcome by using a fairly large number of sample points of the order of 5 to 10 times the number of variables.

  For example, if an eighth-order digital filter is to be designed,

  - the transfer function would involve 17 independent variables (four per biquadratic transfer function plus one).

Improved Minimax Algorithms *Cont'd*

- This problem is usually overcome by using a fairly large number of sample points of the order of 5 to 10 times the number of variables.

  For example, if an eighth-order digital filter is to be designed,

  - the transfer function would involve 17 independent variables (four per biquadratic transfer function plus one).
  - In such a case 85 to 170 sample points may be required which means that 85 to 170 gain evaluations would be required.

Improved Minimax Algorithms *Cont'd*

- This problem is usually overcome by using a fairly large number of sample points of the order of 5 to 10 times the number of variables.

  For example, if an eighth-order digital filter is to be designed,

  - the transfer function would involve 17 independent variables (four per biquadratic transfer function plus one).
  - In such a case 85 to 170 sample points may be required which means that 85 to 170 gain evaluations would be required.
  - A single optimization may necessitate from 300 to 600 function evaluations, and a minimax algorithm may require 5 to 10 unconstrained optimizations to converge.

Improved Minimax Algorithms *Cont'd*

- This problem is usually overcome by using a fairly large number of sample points of the order of 5 to 10 times the number of variables.

  For example, if an eighth-order digital filter is to be designed,

  – the transfer function would involve 17 independent variables (four per biquadratic transfer function plus one).

  – In such a case 85 to 170 sample points may be required which means that 85 to 170 gain evaluations would be required.

  – A single optimization may necessitate from 300 to 600 function evaluations, and a minimax algorithm may require 5 to 10 unconstrained optimizations to converge.

- That means that we would require to calculate the gain of the filter some 10^5 to 10^6 times, *many more* if a very selective filter is required.

Improved Minimax Algorithms *Cont'd*

- The amount of computation required to carry out a design and/or the probability of spikes in the error function can be reduced by using a so called *nonuniform variable sampling*.

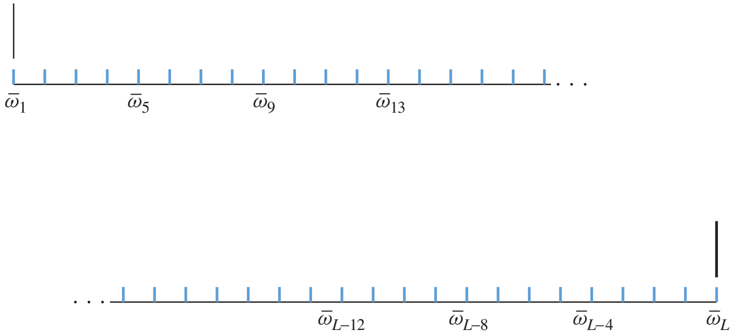
Improved Minimax Algorithms *Cont'd*

- The amount of computation required to carry out a design and/or the probability of spikes in the error function can be reduced by using a so called *nonuniform variable sampling*.
- This technique involves seven steps as detailed in the next few slides.

Improved Minimax Algorithms *Cont'd*

1. Define a dense grid of L uniformly spaced sample points $\bar{\omega}_1$ to $\bar{\omega}_L$ over the frequency band of interest, as shown, where L is of the order of 10 to 20 times the number of independent variables depending on the application.

   These points can be referred to as *virtual sample points*.



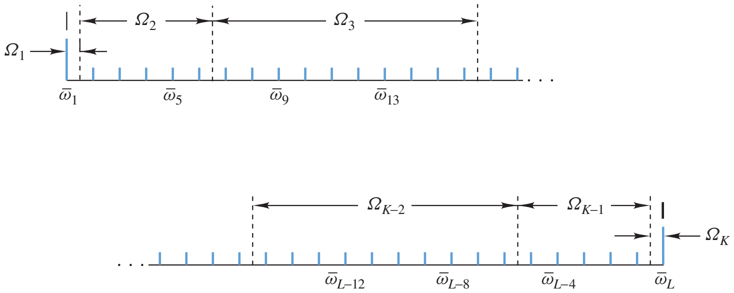
Improved Minimax Algorithms *Cont'd*

2. Divide the frequency band of interest into K intervals, where $K = L/10$, such that
   - intervals Ω_1 and Ω_K contain just one virtual sample point each, the left-hand band edge and right-hand band edge, respectively,
   - intervals Ω_2 and $\Omega_{(K-1)}$ contain 5 virtual sample points each,
   - intervals Ω_3 to $\Omega_{(K-2)}$ contain 10 virtual sample points each.

   As will be seen later on, each of intervals Ω_1 and Ω_K will carry just one *actual sample frequency* and, consequently, the above scheme will lead to a higher density of actual sample frequencies near band edges where the error tends to be larger in practice.

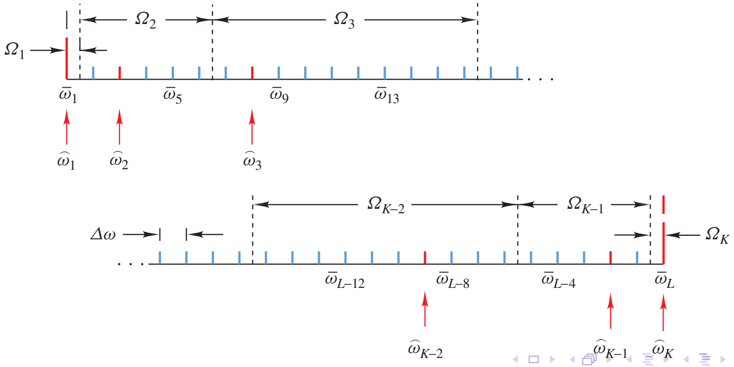
   Other choices of interval sizes could work out just as well or better, e.g., 1, 2, 5, 11, . . ., 11, 5, 2, 1.

Improved Minimax Algorithms *Cont'd*



Improved Minimax Algorithms *Cont'd*

3. Evaluate the error function with respect to the *virtual sample frequencies*, $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_L$.
4. For each of the K intervals, find the virtual sample frequency that yields the maximum error for the interval. Let these frequencies be $\widehat{\omega}_i$ for $i = 1, 2, \ldots, K$.



Improved Minimax Algorithms *Cont'd*

5. Using frequencies $\widehat{\omega}_i$ as the actual sample frequencies, i.e., set $\omega_i = \widehat{\omega}_i$ for $i = 1, 2, \ldots, K$, optimize the objective function.

Improved Minimax Algorithms *Cont'd*

5. Using frequencies $\widehat{\omega}_i$ as the actual sample frequencies, i.e., set $\omega_i = \widehat{\omega}_i$ for $i = 1, 2, \ldots, K$, optimize the objective function.
6. Repeat from Step 4 until convergence is achieved.

Improved Minimax Algorithms *Cont'd*

- By applying the nonuniform variable sampling technique before each optimization, *the frequencies at which spikes are beginning to form* are located and are used as the actual sample frequencies in the next optimization.

Improved Minimax Algorithms *Cont'd*

- By applying the nonuniform variable sampling technique before each optimization, *the frequencies at which spikes are beginning to form* are located and are used as the actual sample frequencies in the next optimization.
- In this way, *spikes are suppressed* and the maximum error is reduced.

Improved Minimax Algorithms *Cont'd*

- By applying the nonuniform variable sampling technique before each optimization, *the frequencies at which spikes are beginning to form* are located and are used as the actual sample frequencies in the next optimization.
- In this way, *spikes are suppressed* and the maximum error is reduced.
- *Note* that the error function is evaluated with respect to the dense set of frequencies $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_L$ *just once* before each optimization and only the actual sampling frequencies $\omega_i = \widehat{\omega}_i$ for $i = 1, 2, \ldots, K$ are used by the optimization.

Improved Minimax Algorithms *Cont'd*

- By applying the nonuniform variable sampling technique before each optimization, *the frequencies at which spikes are beginning to form* are located and are used as the actual sample frequencies in the next optimization.
- In this way, *spikes are suppressed* and the maximum error is reduced.
- *Note* that the error function is evaluated with respect to the dense set of frequencies $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_L$ *just once* before each optimization and only the actual sampling frequencies $\omega_i = \widehat{\omega}_i$ for $i = 1, 2, \ldots, K$ are used by the optimization.

  In this way, the amount of computation required can be *reduced by as much as 50%* relative to that required if *uniform fixed sampling* is used.

Improved Minimax Algorithms *Cont'd*

- By applying the nonuniform variable sampling technique before each optimization, *the frequencies at which spikes are beginning to form* are located and are used as the actual sample frequencies in the next optimization.
- In this way, *spikes are suppressed* and the maximum error is reduced.
- *Note* that the error function is evaluated with respect to the dense set of frequencies $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_L$ *just once* before each optimization and only the actual sampling frequencies $\omega_i = \widehat{\omega}_i$ for $i = 1, 2, \ldots, K$ are used by the optimization.

  In this way, the amount of computation required can be *reduced by as much as 50%* relative to that required if *uniform fixed sampling* is used.

  In other words, we perform a small amount of computation to *tune the initialization* in order to save a lot of computation in *performing the optimization*.

Improved Minimax Algorithms *Cont'd*

- The details of the technique, including a set of formulas that implement the segmentation scheme described, can be found in the textbook.

This slide concludes the presentation.
Thank you for your attention.