



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

OPTIMIZATION IN CHEMICAL ENGINEERING

Unconstrained Multivariable Optimization: Gradient Based Methods

(WEEK – 7: LECTURE – 31)

Dr. Debasis Sarkar

**Department of Chemical Engineering
Indian Institute of Technology Kharagpur**

Unconstrained Multivariable Optimization: Gradient Based Methods

Week 7:

- Cauchy's method
- Newton's method
- Marquardt method
- Conjugate gradient method

➤ Today's Topic:

- General concept of gradient based methods
- Cauchy's steepest descent method

Gradient of a Function

The gradient of a function is an n -component vector given by:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}_{n \times 1}$$

Important property of gradient:

If we move along the gradient direction from any point in n dimensional space, the function value increases at the fastest rate. Hence the gradient direction is called the direction of the steepest ascent. However, the direction of steepest ascent is a local property not a global one.

Gradient-based methods use derivative information of the function to determine the search direction. When derivative information are available, these methods are much faster compared to the Direct Search methods.

Gradient Based Methods

First order methods:

First order methods use only gradient of the objective function to determine the search direction. For example: Cauchy's Steepest Descent Method.

Second order methods:

Second order methods use both gradient and Hessian of the objective function to determine the search direction. For example: Newton's method

Numerical Computation of Derivatives

The first and second derivatives can be computed using the central difference formula as given below:

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i + \Delta x_i) - f(x_i - \Delta x_i)}{2\Delta x_i}$$

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{f(x_i + \Delta x_i) - 2f(x_i) + f(x_i - \Delta x_i)}{\Delta x_i^2}$$

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \left[f(x_i + \Delta x_i, x_j + \Delta x_j) - f(x_i + \Delta x_i, x_j - \Delta x_j) - f(x_i - \Delta x_i, x_j + \Delta x_j) + f(x_i - \Delta x_i, x_j - \Delta x_j) \right] / (4\Delta x_i \Delta x_j)$$

Numerical computation of gradient is often required.

The computation of first derivative requires two function evaluations with respect to each variable. So for an n variable problem, $2n$ function evaluations are required for computing the gradient vector.

General Steps of Gradient Based Methods

Step – 1: Estimate reasonable starting point $x^{(0)}$.

Set iteration counter $k = 0$

Step – 2: Compute search direction $s^{(k)}$

Step – 3: Check for convergence.

Step – 4: Compute step size $\alpha^{(k)}$ in the direction $s^{(k)}$

Step – 5: Update: $x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}$

Step – 6: Go to *Step – 2*

Different methods
determine the step length
 α^k and search direction s^k
differently.

Descent Direction

Consider the following unconstrained optimization problem: $\min_{x \in \mathcal{R}} f(x)$

At any k -th iteration the next point is given by: $x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}$

Here s^k is a search direction and α^k is the step length (positive).

If s^k is a descent direction: $f(x^{(k+1)}) < f(x^{(k)}) \Rightarrow f(x^{(k)} + \alpha^{(k)} s^{(k)}) < f(x^{(k)})$

By Taylor series expansion:

$$f(x^{(k)}) + \alpha^{(k)} (\nabla f(x^{(k)}) \cdot s^{(k)}) < f(x^{(k)})$$

$$\Rightarrow \nabla f(x^{(k)}) \cdot s^{(k)} < 0$$

Dot product is less than zero.
This is the condition for the
descent direction.

Descent Direction

Condition for descent direction:

$$\nabla f(x^{(k)}) \cdot s^{(k)} < 0$$

Since the gradient vector represents the direction of steepest ascent, the negative of the gradient vector denotes the direction of the steepest descent

The magnitude of the vector $\nabla f(x^{(k)}) \cdot s^{(k)}$ for a descent direction $s^{(k)}$ specifies how descent the search direction is.

If $s^{(k)} = -\nabla f(x^{(k)})$ is used, the quantity $\nabla f(x^{(k)}) \cdot s^{(k)}$ is maximally negative.

Thus, the search direction $s^{(k)} = -\nabla f(x^{(k)})$ is called steepest descent direction.

Cauchy's Steepest Descent Method

In Cauchy's method, the search direction at any point is the negative of the gradient of the objective function:

$$s^{(k)} = -\nabla f(x^{(k)})$$

At each iteration, we update: $x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}$

We determine $\alpha^{(k)}$ such that $f(x^{(k+1)})$ is minimum along $\nabla f(x^{(k)})$ using an appropriate unidirectional search method. This minimum point becomes the current point and the search is continued from this point.

Algorithm stops when the magnitude of the gradient vector becomes very small. Note that improvement at every iteration is guaranteed due to descent property.

Cauchy's Steepest Descent Method: Algorithm

Step – 1: Estimate reasonable starting point $x^{(0)}$; Maximum number of iterations, $K_{\max}$;

Termination parameter ε ; Set iteration counter $k = 0$

Step – 2: Compute $\nabla f(x^{(k)})$

Step – 3: Check for convergence: if $\|\nabla f(x^{(k)})\| \leq \varepsilon$, *Stop*

Else if $k \geq K_{\max}$, *Stop*; Else go to *Step – 4*

Step – 4: Compute $\alpha^{(k)}$ such that $f(x^{(k+1)}) = f(x^{(k)} - \alpha^{(k)}\nabla f(x^{(k)}))$ is minimum

Step – 5: If $\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)}\|} \leq \varepsilon$, *Stop*; Else set $k = k + 1$ and go to *Step – 2*

Cauchy's Steepest Descent Method: Features

Improvement at every iteration is guaranteed.

Cauchy's method works very well when the starting point $x^{(0)}$ is far away from the optimum point x^* .

As the current point becomes close to the optimal point, the magnitude of the gradient vector becomes very small. Thus the new point created becomes very close to the current point. This slows down the rate of convergence.

Cauchy's Steepest Descent Method: Example

Minimize $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$

starting from the point $\mathbf{X}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Solution:

Iteration - 1: The gradient of f is given by:

$$\nabla f = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \end{bmatrix} = \begin{bmatrix} 1 + 4x_1 + 2x_2 \\ -1 + 2x_1 + 2x_2 \end{bmatrix}$$

$$\nabla f_0 = \nabla f(\mathbf{X}^{(0)}) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\text{Therefore, } \mathbf{S}^{(0)} = -\nabla f_0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Cauchy's Steepest Descent Method: Example

To find $\mathbf{X}^{(1)}$, we need to find the optimal step length λ_0^* . For this, we minimize

$$f(\mathbf{X}^{(0)} + \lambda_0^* \mathbf{S}^{(0)}) = f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \lambda_0^* \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = f(-\lambda_0, \lambda_0) = \lambda_0^2 - 2\lambda_0 \text{ with respect to } \lambda_0.$$

Since $\frac{df}{d\lambda_0} = 0$ at $\lambda_0^* = 1$, we obtain: $\mathbf{X}^{(1)} = \mathbf{X}^{(0)} + \lambda_0^* \mathbf{S}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

Now, $\nabla f_1 = \nabla f(\mathbf{X}^{(1)}) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \neq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, Thus $\mathbf{X}^{(1)}$ is not optimum.

We proceed to the next iteration.

Cauchy's Steepest Descent Method: Example

Iteration - 2:

$$\mathbf{S}^{(1)} = -\nabla f_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

To minimize $f(\mathbf{X}^{(1)} + \lambda_1 \mathbf{S}^{(1)}) = f(-1 + \lambda_1, 1 + \lambda_1) = 5\lambda_1^2 - 2\lambda_1 - 1$

we set $\frac{df}{d\lambda_1} = 0$. This gives $\lambda_1^* = 0.2$, and hence

$$\mathbf{X}^{(2)} = \mathbf{X}^{(1)} + \lambda_1^* \mathbf{S}^{(1)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.8 \\ 1.2 \end{bmatrix}$$

$$\text{Now, } \nabla f_2 = \nabla f(\mathbf{x}^{(2)}) = \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}$$

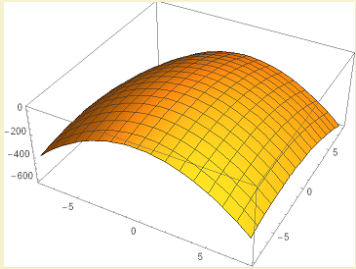
Since the components of the gradient at $\mathbf{X}^{(2)}$ are not zero, we must proceed to the next iteration.

Repeat this process until the optimum point $\mathbf{x}^*$ is found.

$$\mathbf{x}^* = \begin{bmatrix} -1.0 \\ 1.5 \end{bmatrix}$$

Optimization in Chemical Engineering

Thank You





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

OPTIMIZATION IN CHEMICAL ENGINEERING

Unconstrained Multivariable Optimization: Gradient Based Methods

(WEEK – 7: LECTURE – 32)

Dr. Debasis Sarkar

**Department of Chemical Engineering
Indian Institute of Technology Kharagpur**

Unconstrained Multivariable Optimization: Gradient Based Methods

Week 7:

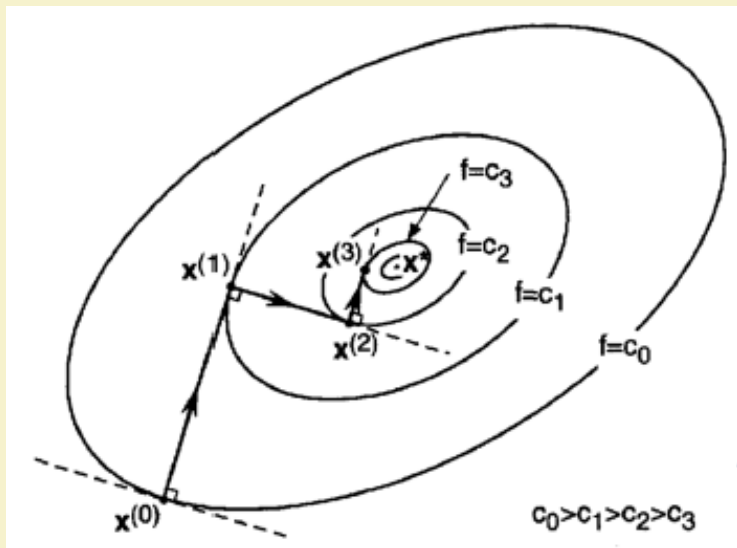
- Cauchy's method
- Newton's method
- Marquardt method
- Conjugate gradient method

➤ Today's Topic:

- Cauchy's steepest descent method (cont'd)
- Conjugate gradient method
(Fletcher-Reeves method)

Cauchy's Steepest Descent Method

The steepest descent method zigzags its way towards the optimum point.



Each step is orthogonal to previous step.

Update rule: $x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}$

$\alpha^{(k)}$ is obtained by minimizing $f(x^{(k)} + \alpha^{(k)} s^{(k)})$. We set $\frac{df}{d\alpha} = 0$

Upon differentiation : $\nabla f(x^{(k)} + \alpha^{(k)} s^{(k)})^T s^{(k)} = 0$

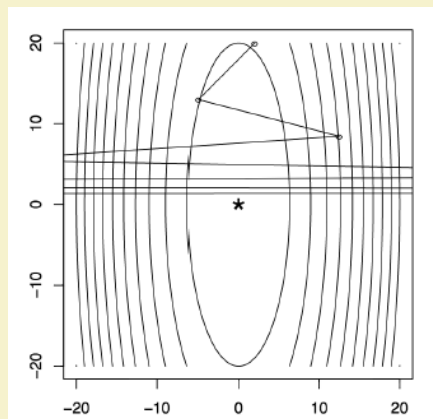
$$\Rightarrow \nabla f(x^{(k+1)})^T s^{(k)} = 0$$

Since $s^{(k+1)} = -\nabla f(x^{(k+1)})$, we get $(s^{(k+1)})^T s^{(k)} = 0$.

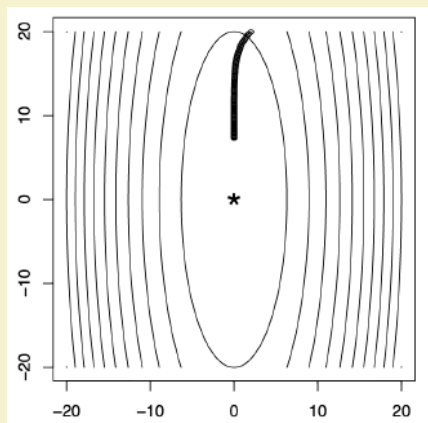
This result is true only if the line search is performed exactly.

Gradient Based Methods: Role of Step Length

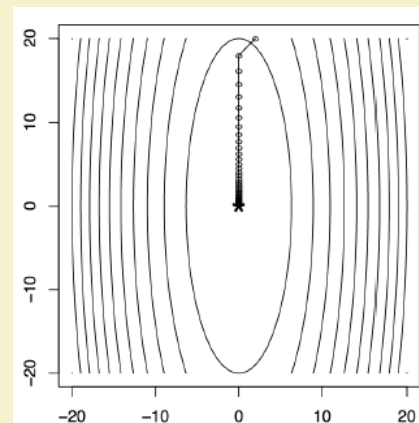
The update rule of a Gradient Based Method requires a step length controlling the amount of gradient updated to the current point at each iteration. A too large step length can lead to divergence. A too small step length takes longer time to converge. **Backtracking line search** and **exact line search** are two methods to find step length.



Too large step size



Too small step size

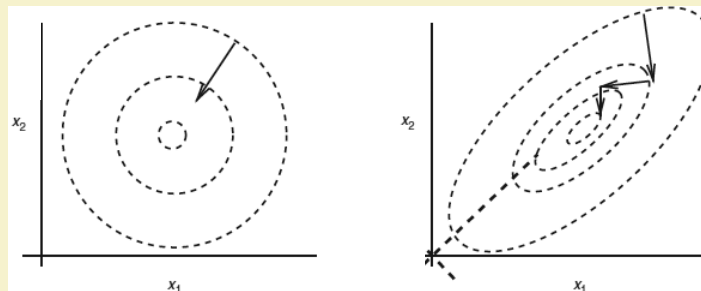


Appropriate step size

Cauchy's Steepest Descent Method: Scaling

Consider the two quadratic functions:

Larger is the value of the parameter a , more elongated are the contours of the function f , and slower is the convergence rate.



$$f = x_1^2 + x_2^2$$

$$f = x_1^2 + ax_2^2$$

Speed of convergence is related to the condition number of Hessian matrix. The condition number of a symmetric positive definite matrix is the ratio of largest to smallest eigenvalues. For a well conditioned Hessian matrix, the condition number is close to unity. Then the contours become more circular and the steepest descent method works best.

Cauchy's Steepest Descent Method: Scaling

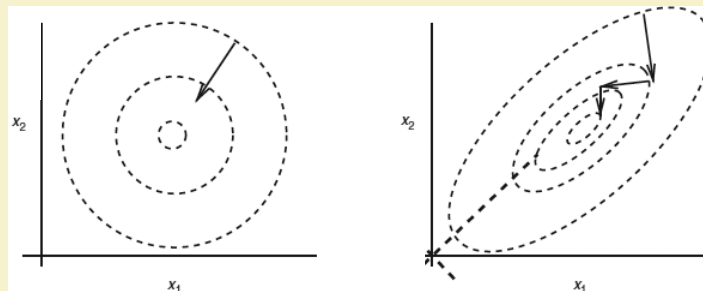
For the function $f = x_1^2 + ax_2^2$ we have:

$$H = \nabla^2 f = \begin{bmatrix} 2 & 0 \\ 0 & 2a \end{bmatrix}$$

Eigenvalues are 2 and $2a$.

Thus, the condition number = a

Convergence is fastest when $a = 1$



$$f = x_1^2 + x_2^2$$

$$f = x_1^2 + ax_2^2$$

Cauchy's Steepest Descent Method: Scaling

For the function $f = x_1^2 + ax_2^2$ we can define new variables y_1 and y_2 as follows:

$$y_1 = x_1, \quad y_2 = \sqrt{a}x_2$$

Then redefine f as: $g = y_1^2 + y_2^2$ The condition number of the Hessian of this new matrix will be 1. Then the contours become circular and the steepest descent method works best.

In general, if for a function $f(\mathbf{x})$, we scale the variables as: $\mathbf{x} = \mathbf{T}\mathbf{y}$, then the new function is

$$g(\mathbf{y}) = f(\mathbf{T}, \mathbf{y}); \quad \text{Gradient: } \nabla g = \mathbf{T}^T \nabla f; \quad \text{Hessian: } \nabla^2 g = \mathbf{T}^T \nabla^2 f \mathbf{T}$$

Usually $\mathbf{T}$ is chosen as a diagonal matrix such that Hessian of g has condition number close to unity.

Cauchy's Steepest Descent Method: Comments

Cauchy's steepest descent method works well when we start far from optimal point. It has linear convergence.

If an exact search direction is made at each iteration, two consecutive search directions will be orthogonal to each other.

In most implementations, the search directions are scaled such that the length of the search vector s is 1. For minimizing a function, we can achieve this by defining:

$$s^{(k)} = -\frac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|}$$

If the objective function is not convex, we can find local minima. We may also get stuck at a saddle point.

The Conjugate Gradient Method

The Conjugate Gradient Method

The steepest descent method works well when we start far from the optimal point. But the convergence becomes very slow as the optimal point is approached. The convergence characteristics of the steepest descent method can be improved greatly by modifying it into a conjugate gradient method. This was first proposed by Fletcher and Reeves (1964).

The conjugate gradient method can be considered as a conjugate directions method that uses the gradient of the objective function.

Any minimization method that uses the conjugate directions is quadratically convergent. Such methods will minimize a quadratic function of n variables in n steps or less.

$$f(x) = \frac{1}{2} x^T A x + b^T x$$

Here A is positive definite matrix.

The Conjugate Gradient Method

Powell's conjugate direction method requires n single-variable search per iteration. We then find new conjugate direction at the end of each iteration.

Therefore, to find the minimum of a quadratic function, in general, Powell's method requires n^2 single-variable search.

But if we use the gradients of the objective function, we can set up a new conjugate direction after every single-variable search, and hence we can achieve faster convergence.

The Conjugate Gradient Method

The conjugate gradient method combines current information about the gradient vector with that of gradient vectors from previous iterations to obtain the new search direction.

The proposed search direction for this method is a linear combination of the current gradient and the previous search direction.

$$S^{(k+1)} = -\nabla f_{k+1} + \frac{\|\nabla f_{k+1}\|^2}{\|\nabla f_k\|^2} S^{(k)} = -\nabla f_{k+1} + \frac{\nabla^T f_{k+1} \nabla f_{k+1}}{\nabla^T f_k \nabla f_k} S^{(k)}$$

Note that conjugate gradient method is a first-order method. This method represents a major improvement over steepest descent method with only a marginal increase in computational effort.

The Conjugate Gradient Method: Algorithm

Step-1: Start with an arbitrary initial point $\mathbf{X}^0$.

Step-2: Set the first search direction $\mathbf{S}^0 = -\nabla f(\mathbf{X}^0) = -\nabla f_0$

Step-3: Find the point $\mathbf{X}^1$ according to the relation $\mathbf{X}^1 = \mathbf{X}^0 + \lambda_0^* \mathbf{S}^0$

where λ_0^* is the optimal step length in the direction $\mathbf{S}^0$. Set $k=2$ and go to next step.

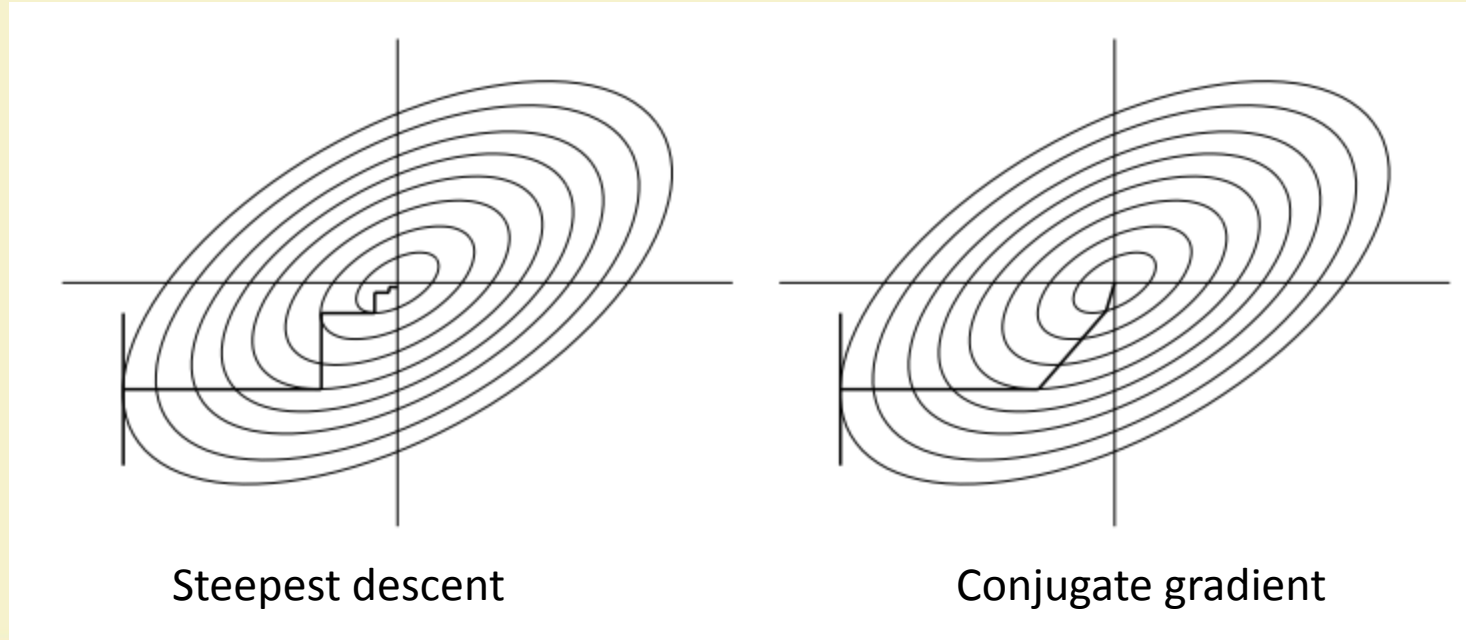
Step-4: Find $\nabla f_k = \nabla f(\mathbf{X}^k)$, and set $\mathbf{S}^{(k+1)} = -\nabla f_{k+1} + \frac{\|\nabla f_{k+1}\|^2}{\|\nabla f_k\|^2} \mathbf{S}^{(k)}$

Step-5: Compute the optimum step length λ_k^* in the direction $\mathbf{S}^k$ and find the new point $\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \lambda_k^* \mathbf{S}^{(k)}$

Step-6: If convergence (small value of gradient) is achieved, stop.

Otherwise set the value of $k=k+1$ and go to step 4.

The Conjugate Gradient Method Vs Steepest Descent Method



The Conjugate Gradient Method: Example

Minimize $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$

starting from the point $\mathbf{X}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Solution:

Iteration - 1: The gradient of f is given by:

$$\nabla f = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \end{bmatrix} = \begin{bmatrix} 1 + 4x_1 + 2x_2 \\ -1 + 2x_1 + 2x_2 \end{bmatrix}$$

$$\nabla f_0 = \nabla f(\mathbf{X}^{(0)}) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\text{Therefore, } \mathbf{S}^{(0)} = -\nabla f_0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

The Conjugate Gradient Method: Example

To find $\mathbf{X}^{(1)}$, we need to find the optimal step length λ_0^* . For this, we minimize $f(\mathbf{X}^{(0)} + \lambda_0^* \mathbf{S}^{(0)})$ with respect to λ_0 .

$$\text{Now, } f(\mathbf{X}^{(0)} + \lambda_0^* \mathbf{S}^{(0)}) = f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \lambda_0^* \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = f(-\lambda_0, \lambda_0)$$

$$\text{Evaluate } f(-\lambda_0, \lambda_0) = -\lambda_0 - \lambda_0 + 2\lambda_0^2 - 2\lambda_0^2 + \lambda_0^2 = \lambda_0^2 - 2\lambda_0$$

$$\text{Set } \frac{df}{d\lambda_0} = 0 \text{ and we get } \lambda_0^* = 1. \text{ Thus, } \mathbf{X}^{(1)} = \mathbf{X}^{(0)} + \lambda_0^* \mathbf{S}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\text{Now, } \nabla f_1 = \nabla f(\mathbf{X}^{(1)}) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

We proceed to the next iteration.

The Conjugate Gradient Method: Example

Iteration - 2:

$$\mathbf{S}^{(1)} = -\nabla f_1 + \frac{\|\nabla f_1\|^2}{\|\nabla f_0\|^2} \mathbf{S}^{(0)}; \quad \text{Here, } \|\nabla f_0\|^2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}^T \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 2 \quad \text{and} \quad \|\nabla f_1\|^2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}^T \begin{bmatrix} -1 \\ -1 \end{bmatrix} = 2$$

$$\text{Therefore, } \mathbf{S}^{(1)} = -\nabla f_1 + \frac{\|\nabla f_1\|^2}{\|\nabla f_0\|^2} \mathbf{S}^{(0)} = -\begin{bmatrix} -1 \\ -1 \end{bmatrix} + \left(\frac{2}{2}\right) \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

$$\text{To minimize } f(\mathbf{X}^{(1)} + \lambda_1 \mathbf{S}^{(1)}) = f(-1, 1 + 2\lambda_1) = 4\lambda_1^2 - 2\lambda_1 - 1$$

we set $\frac{df}{d\lambda_1} = 0$. This gives $\lambda_1^* = 0.25$, and hence

$$\mathbf{X}^{(2)} = \mathbf{X}^{(1)} + \lambda_1^* \mathbf{S}^{(1)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.25 \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix} \quad \text{This is the optimum.}$$

$$\text{Note, } \nabla f_2 = \nabla f(x^{(2)}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix};$$

$$\|\nabla f_1\|^2 = 2; \quad \|\nabla f_2\|^2 = 0$$

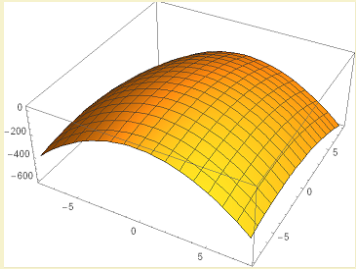
$$\text{Thus, } \mathbf{S}^{(2)} = 0$$

No search direction to reduce f further.

$$\mathbf{x}^* = \begin{bmatrix} -1.0 \\ 1.5 \end{bmatrix}$$

Optimization in Chemical Engineering

Thank You





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

OPTIMIZATION IN CHEMICAL ENGINEERING

Unconstrained Multivariable Optimization: Gradient Based Methods

(WEEK – 7: LECTURE – 33)

Dr. Debasis Sarkar

**Department of Chemical Engineering
Indian Institute of Technology Kharagpur**

Unconstrained Multivariable Optimization: Gradient Based Methods

Week 7:

- ✔ ➤ Cauchy's method
 - Newton's method
 - Marquardt method
- ✔ ➤ Conjugate gradient method

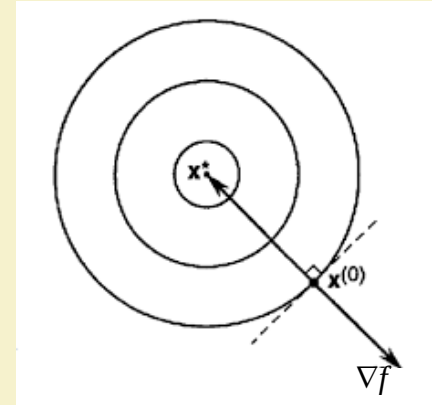
➤ Today's Topic:

- Newton's method

Newton's Method: Second-Order Method

As we have seen in Cauchy's method, negative of gradient points directly toward the minimum only when the contours of the function f is circular.

Thus, in general, negative gradient may not be a good global direction for any arbitrary nonlinear function.



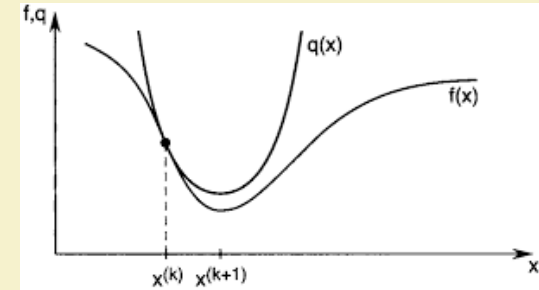
Newton's method uses information about second derivative to create search direction and achieve faster convergence. In other words, information about the Hessian of the objective function is used. Thus, it is possible to take into account the curvature of $f(x)$ (objective function surface) and identify better search directions than can be obtained using only gradient information.

Newton's Method: Basic Idea

The basic idea of the classical Newton's method is to use second-order (quadratic) approximation of the function $f(x)$ about the current search point x^k .

$$f(x) \approx f(x^k) + \nabla^T f(x^k) \Delta x^k + \frac{1}{2} (\Delta x^k)^T H(x^k) \Delta x^k$$

Here $H(x^k)$ is the Hessian matrix.



The idea is to determine the optimal solution to the approximate function and use this point to determine a search direction. If the approximation is of high quality, the search direction will also be of high quality.

Newton's Method

Consider the second-order Taylor expansion of a multivariable function $f(x)$ about the current point x^k

$$f(x) \approx f(x^k) + \nabla^T f(x^k) \Delta x^k + \frac{1}{2} (\Delta x^k)^T H(x^k) \Delta x^k$$

The minimum of this quadratic approximation of $f(x)$ can be obtained by differentiating the equation with respect to each of the components of Δx and equating the resulting expressions to zero. This is First Order Necessary Condition (FONC).

$$\nabla f(x) = \nabla f(x^k) + H(x^k) \Delta x^k = 0$$

$$\Rightarrow x^{k+1} - x^k = \Delta x^k = -[H(x^k)]^{-1} \nabla f(x^k)$$

where $[H(x^k)]^{-1}$ is the inverse of the Hessian matrix $H(x^k)$. Here, H is assumed to be non-singular.

If $H(x^k)$ is positive semi-definite, the approximate function will have a minimum at:

$$x^{k+1} = x^k - [H(x^k)]^{-1} \nabla f(x^k) \quad \text{This is recursive formula for Newton's method.}$$

Newton's Method: Compare With 1D Search

$$\nabla f(x) = \nabla f(x^k) + H(x^k)\Delta x^k = 0$$

$$\Rightarrow x^{k+1} - x^k = \Delta x^k = -[H(x^k)]^{-1} \nabla f(x^k)$$

where $[H(x^k)]^{-1}$ is the inverse of the Hessian matrix $H(x^k)$.

Note that we have solved for a vector, Δx , which has both a step length and direction. Also, the search direction is valid both for minimization and maximization.

Compare with one-dimension search (single variable unconstrained search):

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}$$

Newton's Method: Modified Update Rule

Classical Newton's Method: $x^{k+1} - x^k = \Delta x^k = -[H(x^k)]^{-1} \nabla f(x^k)$

If $f(x)$ is actually quadratic, only one step is required to reach the minimum of $f(x)$.

For a general nonlinear objective function, however, the minimum of $f(x)$ cannot be reached in one step. The Newton step change may be large when we are far from optimum and there is a chance of divergence. Thus, we can modify the above equation (search direction) by introducing the step length parameter (α):

$$x^{k+1} - x^k = \Delta x^k = -\alpha^k [H(x^k)]^{-1} \nabla f(x^k)$$

By performing one-dimensional line search, we can choose α^k such that $f(x^{k+1})$ is minimized. This will ensure descent: $f(x^{k+1}) < f(x^k)$

Newton's Method: Algorithm

Step – 1: Estimate reasonable starting point $x^{(0)}$; Maximum number of iterations, $K_{\max}$;
Termination parameter ε ; Set iteration counter $k = 0$

Step – 2: Compute $\nabla f(x^{(k)})$

Step – 3: Check for convergence: if $\|\nabla f(x^{(k)})\| \leq \varepsilon$, Stop

Else if $k \geq K_{\max}$, Stop; Else go to Step – 4

Step – 4: Compute $\alpha^{(k)}$ such that $f(x^{(k+1)}) = f\left(x^{(k)} - \alpha^{(k)} [\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})\right)$ is minimum

Step – 5: If $\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)}\|} \leq \varepsilon$, Stop; Else set $k = k + 1$ and go to Step – 2

Newton's Method: Example - 1

Minimize $f(x_1, x_2) = 4x_1^2 + x_2^2 - 2x_1x_2$ starting from the point $\mathbf{X}^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Now,

$$\nabla f(\mathbf{X}^{(0)}) = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \end{bmatrix} = \begin{bmatrix} 8x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \end{bmatrix}$$

$$H(\mathbf{X}^{(0)}) = \begin{bmatrix} 8 & -2 \\ -2 & 2 \end{bmatrix}, H^{-1}(\mathbf{X}^{(0)}) = \begin{bmatrix} 1/6 & 1/6 \\ 1/6 & 2/3 \end{bmatrix}$$

With $\alpha = 1$,

$$\Delta \mathbf{X}^0 = -[H(\mathbf{X}^{(0)})]^{-1} \nabla f(\mathbf{X}^{(0)}) = -\begin{bmatrix} 1/6 & 1/6 \\ 1/6 & 2/3 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\mathbf{x}^1 = \mathbf{x}^0 + \Delta \mathbf{x}^0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{x}^*$$

To check this is optimal, evaluate

$$\begin{aligned} \nabla f(\mathbf{X}^{(1)}) &= \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \end{bmatrix}_{(0,0)} \\ &= \begin{bmatrix} 8x_1 - 2x_2 \\ 2x_2 - 2x_1 \end{bmatrix}_{(0,0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

Newton's Method: Quadratic Convergence

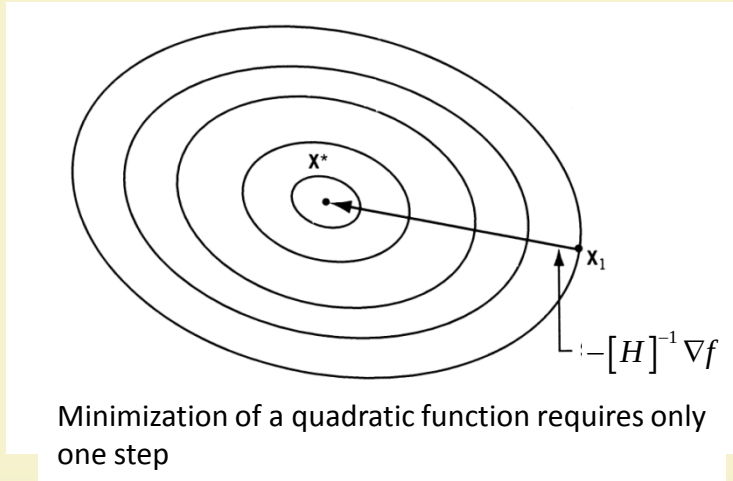
Minimize $f(x_1, x_2) = 4x_1^2 + x_2^2 - 2x_1x_2$ starting from the point $\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

We have seen:

$$\mathbf{x}^1 = \mathbf{x}^0 + \Delta \mathbf{x}^0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{x}^*$$

Newton's method shows quadratic convergence.

For a quadratic function, Newton's method will converge in one iteration starting from any arbitrary point.



Newton's Method: Is Inverse Always Required?

Newton's method involves inverse of Hessian matrix. This may be computationally expensive.

$$\Delta \mathbf{x}^0 = -\left[H(\mathbf{X}^{(0)})\right]^{-1} \nabla f(\mathbf{X}^{(0)}) = -\begin{bmatrix} 1/6 & 1/6 \\ 1/6 & 2/3 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

The matrix inversion is not necessarily required. Instead, we can solve the following set of linear equations:

$$\nabla f(x) = \nabla f(x^k) + H(x^k)\Delta x^k = 0$$

$$\Rightarrow H(x^k)\Delta x^k = -\nabla f(x^k)$$

$$\text{Note, } \nabla f(\mathbf{X}^{(0)}) = \begin{bmatrix} 6 \\ 0 \end{bmatrix}; \quad H(\mathbf{X}^{(0)}) = \begin{bmatrix} 8 & -2 \\ -2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 8 & -2 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} \Delta x_1^0 \\ \Delta x_2^0 \end{bmatrix} = -\begin{bmatrix} 6 \\ 0 \end{bmatrix}$$

$$\Rightarrow \Delta x_1^0 = -1, \quad \Delta x_2^0 = -1$$

We obtained the same results using matrix inversion

Newton's Method: Example - 2

Minimize: $f(\vec{X}) = 0.5x_1^2 + 2.5x_2^2$

$$\nabla f(\vec{X}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

$$\vec{X}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}, \vec{X}_1 = \vec{X}_0 - H^{-1} \nabla f(\vec{X}_0) = \begin{bmatrix} 5 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{5} \end{bmatrix} \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Quadratic function converges just in one iteration.

Newton's Method Converges in One Iteration for Quadratic Functions: Proof

Let the quadratic function be given by $f(x) = \frac{1}{2}x^T Hx + B^T x + C$

The minimum of $f(x)$ is given by: $\nabla f = Hx + B = 0$
 $\Rightarrow x^* = -H^{-1}B$

Now we can write from the recursive formula: $x^{k+1} - x^k = -[H_k]^{-1} \nabla f_k$

$$x^{k+1} = x^k - H^{-1}(Hx^k + B)$$

$$\Rightarrow x^{k+1} = -H^{-1}B = x^*$$

where x^k is the starting point for the k -th iteration. Thus, the above equation gives the exact solution

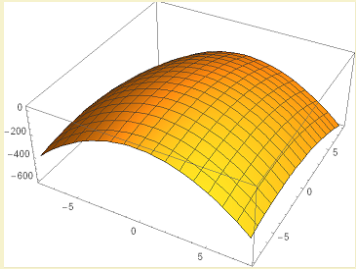
Newton's Method: Disadvantage

Newton's method, when it converges, converges at a faster rate than first-order methods. However, it has the following disadvantages:

1. The method does not necessarily find the global solution if multiple local solutions exist, but this is a general characteristic of gradient based methods.
2. It requires matrix inversion or solution of a set of n symmetric linear equations. Thus the method may be computationally expensive.
3. It requires both first and second partial derivatives, which may not be available always.
4. Using a step size of unity, the method may not always converge.
5. The method is generally sensitive to initial starting point.

Optimization in Chemical Engineering

Thank You





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

OPTIMIZATION IN CHEMICAL ENGINEERING

Unconstrained Multivariable Optimization: Gradient Based Methods

(WEEK – 7: LECTURE – 34)

Dr. Debasis Sarkar

**Department of Chemical Engineering
Indian Institute of Technology Kharagpur**

Unconstrained Multivariable Optimization: Gradient Based Methods

Week 7:

- ✓ ➤ Cauchy's method
- ✓ ➤ Newton's method
 - Marquardt method
- ✓ ➤ Conjugate gradient method

➤ Today's Topic:

- Marquardt method

Newton's Method: Not a Descent Algorithm

Far from the minimum, Newton's method can either go uphill or down. In other words, Newton's method is not a descent algorithm because $f(x^{k+1}) < f(x^k)$ is not guaranteed.

This can be remedied by modifying the increment step in Newton's algorithm so that it is a partial step in a descent direction.

To generate a descent direction we can define the search direction

$$\Delta x^k = -[H(x^k)]^{-1} \nabla f(x^k)$$

using a symmetric positive definite matrix F which is a good approximation of H . We then take a sufficiently short step in the direction:

$$\Delta x^k = -[F(x^k)]^{-1} \nabla f(x^k)$$

Newton's Method: Not a Descent Algorithm: Improvement

1. Introduce the step-size parameter: $x^{k+1} - x^k = \Delta x^k = -\alpha^k [H(x^k)]^{-1} \nabla f(x^k)$
2. Ensure that the search direction is a descent direction to the function f at x^k . This means, we must ensure

$$-\nabla f^T(x^k) [H(x^k)]^{-1} \nabla f(x^k) < 0$$

If the search direction is a descent direction, then the line search will result in a new point with $f(x^{k+1}) < f(x^k)$. If the Hessian matrix H is positive definite, its inverse will also be positive definite, and the above condition for descent direction will be satisfied. One scheme is to replace the Hessian with a symmetric positive definite matrix F^k such as:

$$F^k = H(x^k) + \lambda^k I \quad \text{where } \lambda^k \text{ is chosen such that all the eigenvalues of } F^k \text{ are greater than some scalar } \delta > 0.$$

Marquardt's Method

The steepest descent method works very well (rapidly reduces the function value for minimization problem) when the initial vector (initial guess) $\mathbf{X}^k$ is away from the optimum point $\mathbf{X}^*$.

The Newton's method, on the other hand, converges very fast when the initial vector $\mathbf{X}^k$ is close to the optimum point $\mathbf{X}^*$.

The Marquardt's method (1963) combines both the steepest descent method and Newton's method and take advantages of both the methods.

In Marquardt's method, initially Cauchy's steepest descent method is followed and then Newton's method is adopted.

Marquardt's Method

The Marquardt's method uses the search direction as: $s(x^k) = -[H(x^k) + \lambda^k I]^{-1} \nabla f(x^k)$

Here we take the step length $\alpha^k = 1$ because in the above equation λ^k controls both the direction of the search and length of the step. Here I is the identity matrix.

To begin the search, we set λ to a large value, say 10^4 . Then, initially the Hessian matrix will have little effect on the determination of search direction and the method will be similar to Cauchy's steepest descent method.

$$[H(x^0) + \lambda^0 I]^{-1} \nabla f(x^0) \cong [\lambda^0 I]^{-1} \nabla f(x^0) = \left(\frac{1}{\lambda^0} I \right) \nabla f(x^0)$$

Marquardt's Method

The Marquardt's method uses the search direction as: $s(x^k) = -[H(x^k) + \lambda^k I]^{-1} \nabla f(x^k)$

The search direction $s(x^k)$ becomes a steepest descent direction for large values of λ^0 . In the Marquardt method, the value of λ is taken large at the beginning and then it is gradually reduced to zero as the number of iterations increases.

The large value of λ makes all the eigenvalues of the matrix $[H(x^k) + \lambda^k I]^{-1}$ positive and thus makes the matrix $[H(x^k) + \lambda^k I]^{-1}$ positive definite. Then the search direction will always point in a descent direction.

$$-\nabla f^T(x^k) [H(x^k) + \lambda^k I]^{-1} \nabla f(x^k) < 0$$

Marquardt's Method

The Marquardt's method uses the search direction as: $s(x^k) = -[H(x^k) + \lambda^k I]^{-1} \nabla f(x^k)$

As the value of λ decreases from a large value to zero, the characteristic of the search direction $s(x^k)$ changes from those of the steepest descent method to those of the Newton method. The transition of from Cauchy's steepest descent method to Newton's method depends on the intermediate solutions.

After the first step, we test for descent. Thus we check: is $f(x^1) < f(x^0)$?. If the step is success, we set $\lambda^1 < \lambda^0$ and iterate. For this, we define γ ($0 < \gamma < 1$) and set $\lambda^1 = \gamma \lambda^0$.

Otherwise, we set $\lambda^0 > \beta \lambda^0$, where $\beta > 1$, and compute $s(x^k)$ again.

Marquardt's Method: Algorithm

Step – 1: Define starting point $x^{(0)}$; Maximum number of iterations, $K_{\max}$; $0 < \gamma < 1$; $\beta > 1$

Termination parameter ε ; Set iteration counter $k = 0$; $\lambda^{(0)} = 10^4$ (a large number)

Step – 2: Compute $\nabla f(x^{(k)})$

Step – 3: Check for convergence: if $\|\nabla f(x^{(k)})\| \leq \varepsilon$, *Stop*

Else if $k \geq K_{\max}$, *Stop*; Else go to *Step – 4*

Step – 4: Compute $s(x^{(k)}) = -[H(x^k) + \lambda^{(k)} I]^{-1} \nabla f(x^{(k)})$

Set $x^{(k+1)} = x^{(k)} + s(x^{(k)})$

Step – 5: If $f(x^{(k+1)}) < f(x^{(k)})$ go to *Step – 6*; Else go to *Step – 7*

Step – 6: Set $\lambda^{(k+1)} = \gamma \lambda^{(k)}$; $k = k + 1$; and go to *Step – 2*

Step – 7: Set $\lambda^{(k)} = \beta \lambda^{(k)}$ and go to *Step – 4*.

Marquardt's Method: Advantage/Disadvantage

Major advantages:

1. Simplicity
2. The descent property
3. Excellent convergence near optimal point x^*
4. Absence of line search
5. The method has been used extensively with problems where $f(x)$ is a sum of squares

Disadvantage:

The need to compute Hessian H and $s(x^k) = -[H(x^k) + \lambda^0 I]^{-1} \nabla f(x^k)$

Marquardt's Method: Example

Minimize $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$ starting from the point $\mathbf{X}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
Consider: $\lambda^0 = 10^4$, $\gamma = 0.5$, $\beta = 2$, termination criterion $\varepsilon = 10^{-4}$.

Solution:

Iteration - 1: The function value $f_0 = f(\mathbf{x}^0) = 0$

The gradient of f is given by:

$$\nabla f = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \end{bmatrix} = \begin{bmatrix} 1 + 4x_1 + 2x_2 \\ -1 + 2x_1 + 2x_2 \end{bmatrix}$$

$$\nabla f_0 = \nabla f(\mathbf{X}^{(0)}) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\|\nabla f_0\| = \sqrt{(1)^2 + (-1)^2} = \sqrt{2} = 1.414 > \varepsilon$$

$$\text{The Hessian, } H(\mathbf{X}^0) = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}$$

$$s(\mathbf{X}^0) = -[H(\mathbf{X}^0) + \lambda^0 \mathbf{I}]^{-1} \nabla f(\mathbf{X}^0)$$

$$\mathbf{X}^1 = \mathbf{X}^0 + s(\mathbf{X}^0)$$

Marquardt's Method: Example

Iteration – 1 (Cont'd)

$$H(\mathbf{X}^0) = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}; \nabla f(\mathbf{X}^0) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\begin{aligned} s(\mathbf{X}^0) &= -[H(\mathbf{X}^0) + \lambda^0 \mathbf{I}]^{-1} \nabla f(\mathbf{X}^0) \\ &= -\left[\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix} + 10^4 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right]^{-1} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\ &= -\begin{bmatrix} 4+10^4 & 2 \\ 2 & 2+10^4 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \\ &= 10^{-4} \begin{bmatrix} -0.998 \\ 1.000 \end{bmatrix} \end{aligned}$$

$$\text{Now, } \mathbf{X}^1 = \mathbf{X}^0 + s(\mathbf{X}^0)$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 10^{-4} \begin{bmatrix} -0.998 \\ 1.000 \end{bmatrix} = 10^{-4} \begin{bmatrix} -0.998 \\ 1.000 \end{bmatrix}$$

Test for descent:

$$f_1 = f(\mathbf{X}^1) = -1.9997 \times 10^{-4} < f_0$$

$$\text{Convergence: } \|\nabla f_1\| = 1.414 > \varepsilon$$

Since we are moving in descent direction and convergence has not been achieved, set $\lambda^1 = 0.5\lambda^0 = 5000$ and proceed to the next iteration.

Marquardt's Method: Example

Iteration - 2:

$$H(\mathbf{X}^1) = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}; \nabla f(\mathbf{X}^{(1)}) = \begin{bmatrix} 0.9998 \\ -1.0000 \end{bmatrix}$$

$$\begin{aligned} s(\mathbf{X}^1) &= -[H(\mathbf{X}^1) + \lambda^1 \mathbf{I}]^{-1} \nabla f(\mathbf{X}^1) \\ &= -\left[\begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix} + 5000 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right]^{-1} \begin{bmatrix} 0.9998 \\ -1.0000 \end{bmatrix} \\ &= -\begin{bmatrix} 4 + 5000 & 2 \\ 2 & 2 + 5000 \end{bmatrix}^{-1} \begin{bmatrix} 0.9998 \\ -1.0000 \end{bmatrix} \\ &= 10^{-3} \begin{bmatrix} -0.1999 \\ 0.2000 \end{bmatrix} \end{aligned}$$

Now, $\mathbf{X}^2 = \mathbf{X}^1 + s(\mathbf{X}^1)$

$$\begin{aligned} &= 10^{-4} \begin{bmatrix} -0.998 \\ 1.000 \end{bmatrix} + 10^{-3} \begin{bmatrix} -0.1999 \\ 0.2000 \end{bmatrix} \\ &= 10^{-4} \begin{bmatrix} -2.9986 \\ 3.0000 \end{bmatrix} \end{aligned}$$

Test for descent:

$$f_2 = f(\mathbf{X}^2) = -5.9977 \times 10^{-4} < f_1$$

Thus, we are moving in a descent direction.

If convergence has not been achieved,

set $\lambda^2 = 0.5\lambda^1 = 2500$ and proceed to the next iteration.

Marquardt's Method: Example

Iteration - 3:

Find the gradient of f at X^2 . Then compute X^3 as follows:

$$s(X^2) = -[H(X^2) + \lambda^2 I]^{-1} \nabla f(X^2)$$

$$X^3 = X^2 + s(X^2)$$

If $f(X^3) < f(X^2)$, then set $\lambda^3 = 0.5\lambda^2$ and proceed to next iteration.

If $f(X^3) > f(X^2)$, then set $\lambda^2 = 2\lambda^2$ and set :

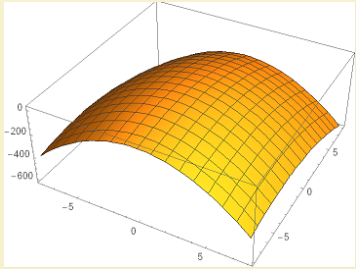
$$s(X^2) = -[H(X^2) + \lambda^2 I]^{-1} \nabla f(X^2)$$

$$X^3 = X^2 + s(X^2) \quad \text{again check for descent}$$

Iterations continue until
convergence is achieved.

Optimization in Chemical Engineering

Thank You





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

OPTIMIZATION IN CHEMICAL ENGINEERING

Unconstrained Multivariable Optimization: Gradient Based Methods

(WEEK – 7: LECTURE – 35)

Dr. Debasis Sarkar

**Department of Chemical Engineering
Indian Institute of Technology Kharagpur**

Unconstrained Multivariable Optimization: Gradient Based Methods

Week 7:

- ✔➤ Cauchy's method
- ✔➤ Newton's method
- ✔➤ Marquardt method
- ✔➤ Conjugate gradient method

➤ Today's Topic:

- Quasi Newton method
- Trust region method
- MATLAB for unconstrained optimization

Quasi-Newton Method

The basic iterative process used in the Newton's method is given by

$$x^{k+1} - x^k = \Delta x^k = -[H(x^k)]^{-1} \nabla f(x^k)$$

where the Hessian matrix $[H]$ is composed of the second partial derivatives of the function f and varies with the current estimate for a general nonlinear objective function f . Note that for a quadratic function, $[H]$ is a constant matrix.

The basic idea behind the Quasi-Newton or variable metric methods is to approximate either $[H]$ or $[H]^{-1}$ by another matrix, using only the first partial derivatives of the objective function f .

Quasi-Newton Method

If $[H]^{-1}$ is approximated by $[B]$, we can write $x^{k+1} - x^k = \Delta x^k = -[H(x^k)]^{-1} \nabla f(x^k)$ as:

$$x^{k+1} - x^k = \Delta x^k = -\alpha^k [B^k] \nabla f(x^k)$$

Thus the search direction s^k is: $s^k = -[B^k] \nabla f(x^k)$

Note that Steepest Descent method is a special case of above equation with $[B] = [I]$.

Two popular methods:

- Davidon-Fletcher-Powell (DFP) method
- Broyden-Fletcher-Goldfarb-Shanno (BFGS) method

Quasi Newton Method

These methods approximate the Hessian matrix using only the first partial derivatives of $f(\mathbf{x})$. Define:

$$\Delta \mathbf{g}^k = \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k)$$

$$\hat{\mathbf{H}}(\mathbf{x}^k) = \hat{\mathbf{H}}^k \quad (\text{approximation of } \mathbf{H}(\mathbf{x}^k))$$

$$\Delta \hat{\mathbf{H}}^k = \hat{\mathbf{H}}^{k+1} - \hat{\mathbf{H}}^k$$

Basically, the methods update an estimate of the Hessian matrix, or it's inverse. Two particular updating mechanisms are:

- Davidon-Fletcher-Powell (DFP) method
- Broyden-Fletcher-Goldfarb-Shanno (BFGS) method

Quasi Newton Method

Two Updating Mechanisms: (Ref: Edgar and Himmelblau)

Davidon-Fletcher-Powell (DFP) Method:

$$[\Delta \hat{\mathbf{H}}^k]^{-1} = \frac{\Delta \mathbf{x}^k (\Delta \mathbf{x}^k)^T}{(\Delta \mathbf{x}^k)^T \Delta \mathbf{g}^k} - \frac{[\hat{\mathbf{H}}^k]^{-1} \Delta \mathbf{g}^k (\Delta \mathbf{g}^k)^T [\hat{\mathbf{H}}^k]^{-T}}{(\Delta \mathbf{g}^k)^T [\hat{\mathbf{H}}^k]^{-1} \Delta \mathbf{g}^k}$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) Method:

$$\Delta \hat{\mathbf{H}}^k = \frac{\Delta \mathbf{g}^k (\Delta \mathbf{g}^k)^T}{(\Delta \mathbf{g}^k)^T \Delta \mathbf{x}^k} - \frac{\hat{\mathbf{H}}^k \Delta \mathbf{x}^k (\Delta \mathbf{x}^k)^T \hat{\mathbf{H}}^k}{(\Delta \mathbf{x}^k)^T \hat{\mathbf{H}}^k \Delta \mathbf{x}^k}$$

NOTE: In both DFP and BFGS methods, the Hessian matrix method updates remain positive definite. have this property.

Trust Region Method

All gradient based methods use: $x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}$

1. Choose a search direction s^k that will produce descent
2. Perform a line search along this direction

Here emphasis is on finding the descent direction

Trust region methods use the following strategy:

1. Form a “trustworthy” or “safe” approximation of the function $f(x)$ in a “trust region” (neighborhood of the current estimate) in which the current approximation of $f(x)$ produces descent
2. Find the minimum of the approximation

Here emphasis is on finding a trustworthy approximation to $f(x)$

Trust Region Method

Newton or Quasi Newton Method: $\min m(x^k + s) = f(x^k) + \nabla f(x^k)s + \frac{1}{2}s^T H^k s$

Trust Region Method: The model $m(x^k+s)$ is an approximation of $f(x^k+s)$: $m(x^k+s) = f(x^k+s)$
Alternate problem formulation may be:

$$\begin{aligned} \min m(x^k + s) &= f(x^k) + \nabla f(x^k)s + \frac{1}{2}s^T H^k s \\ \text{subject to } \|s\| &\leq \delta_k \end{aligned}$$

Here δ_k is the trust region of the model $m(x)$, i.e. the region where we trust the model is valid.

MATLAB Optimization Toolbox

We will discuss two MATLAB functions from the Optimization Toolbox to solve unconstrained optimization problems:

`fminsearch`: Gradient free, uses only function value (implements simplex method)

`fminunc`: Uses gradient, implements various methods such as Quasi-Newton method, Trust region method, etc.

MATLAB Optimization Toolbox: `fminunc`

A Basic call of `fminunc` :

Without any extra options the syntax is:

```
[x,fval] = fminunc(@objfun,x0)
```

where

`objfun`: name of a **function file** supplied by you in which the objective function is coded. You can give any other name.

`x0`: a column vector of starting values (initial guess)

`x` (1st output): optimal solution vector (column vector)

`fval` (2nd output): optimal function value

`fminunc` seeks a minimum.

If you want to maximize a function, code `-f` in the function file (`objfun`)

MATLAB Optimization Toolbox: `fminunc`

Syntax

```
x = fminunc(fun,x0)
```

```
x = fminunc(fun,x0,options)
```

```
[x,fval] = fminunc(____)
```

```
[x,fval,exitflag,output] = fminunc(____)
```

```
[x,fval,exitflag,output,grad,hessian] = fminunc(____)
```

MATLAB Optimization Toolbox: fminunc

Example: The Rosenbrock Function

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Gradient:

$$\begin{aligned}\nabla f(x_1, x_2) &= \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right]^T \\ &= \left[-400(x_2 - x_1^2)x_1 - 2(1 - x_1), \quad 200(x_2 - x_1^2) \right]^T\end{aligned}$$

MATLAB Optimization Toolbox: `fminunc`

Step-1

You first have to code the objective function. Open a new M-file in the editor and type in:

```
function [f, gradf] = objfun(x)
f = 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;
gradf(1) = -400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
gradf(2) = 200*(x(2)-x(1)^2);
```

The first argument `f` is the value of the function at the input vector `x`. The 2nd output argument, `gradf`, is the gradient vector of `f` (column vector), evaluated at `x`

Save the file under any name - here we choose `objfun.m`. Note all MATLAB functions will have extension `.m`.

MATLAB Optimization Toolbox: `fminunc`

Step-2

We have to tell MATLAB to use gradient information. This is done by specifying one of the optimization options, and the program has to be informed that it has to use this option. The general syntax is

```
>> options=optimset('GradObj','on');
```

```
>> [x,fval] = fminunc(@objfun, x0, options)
```

MATLAB Optimization Toolbox: `fminunc`

Step-2

We have to tell MATLAB to use gradient information. This is done by specifying one of the optimization options, and the program has to be informed that it has to use this option. The general syntax is

```
>> options=optimset('GradObj','on');
```

```
>> [x,fval] = fminunc(@objfun, x0, options)
```

MATLAB Optimization Toolbox: fminunc

More Options:

```
opt = optimoptions(@fminunc,...  
'Display','iter',... % shows output at each iteration step  
'gradobj','on',... % use analytic gradients  
'Algorithm','quasi-newton',...  
'HessUpdate','bfgs',... % set the Hessian-update to BFGS  
'TolFun',1e-08,... % main tolerance (gradients)  
'TolX',1e-08, ... % x-change tolerance  
'MaxIter',300,... % max iterations  
'MaxFunEvals',500,... % max function evaluations  
'DerivativeCheck','on' ... % check our analytic gradients  
);
```

MATLAB Optimization Toolbox: fminunc

```
x0 = [-1.5; -4];  
[xhat, fhat, flag, out, ghat, hhat] =  
fminunc(@rosenbrock,x0,opt);
```

MATLAB Optimization Toolbox: fminunc

Results:

Algorithm: Quasi Newton Method:

CheckGradients Information

Objective function derivatives:

Maximum relative difference between supplied
and finite-difference derivatives = 9.18616e-09.

CheckGradients successfully passed.

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	1	3912.5		3.76e+03
1	2	1536.63	0.000266312	786
2	3	1135.01	1	673
3	4	650.897	1	510
4	5	309.937	1	377
5	6	60.3901	1	244
6	7	2.43146	1	55.4
7	8	0.0148755	1	3.02
8	9	0.00703328	1	0.18
9	11	0.00693671	10	0.515
10	13	0.00530386	10	0.625
11	15	0.000918213	0.5	0.549
12	16	0.000408093	1	0.61
13	17	2.15428e-05	1	0.0491
14	18	4.86863e-07	1	0.0251
15	19	5.33201e-09	1	0.00221
16	20	2.83783e-13	1	1.76e-05

[Local minimum found.](#)

Optimization completed because the [size of the gradient](#) is less than
the selected value of the [optimality tolerance](#).

[<stopping criteria details>](#)

MATLAB Optimization Toolbox: fminunc

Results:

```
xhat =  
    1.0000  
    1.0000  
fhat =  
    2.8378e-13  
flag =  
     1  
out =  
    struct with fields:  
        iterations: 16  
        funcCount: 20  
        stepsize: 1.0617e-04  
        lssteplength: 1  
        firstorderopt: 1.7588e-05  
        algorithm: 'quasi-newton'  
        message: 'Local minimum found. Optimization completed because the size of  
the gradient is less than the selected value of the optimality tolerance. Stopping  
criteria details: Optimization completed: The first-order optimality measure,
```

MATLAB Optimization Toolbox: fminunc

Now use Trust Region Algorithm:

```
opt = optimoptions(@fminunc,...  
'Display','iter',... % shows output at each iteration step  
'gradobj','on',... % use analytic gradients  
'Algorithm','trust-region',...  
'HessUpdate','bfgs',... % set the Hessian-update to BFGS  
'TolFun',1e-08,... % main tolerance (gradients)  
'TolX',1e-08, ... % x-change tolerance  
'MaxIter',300,... % max iterations  
'MaxFunEvals',500,... % max function evaluations  
'DerivativeCheck','on' ... % check our analytic gradients  
);
```

MATLAB Optimization Toolbox: fminunc

Iteration	f(x)	Norm of step	First-order optimality	CG-iterations
0	3912.5		3.76e+03	
1	6.24001	6.24401	5	1
2	6.24001	7.88371	5	1
3	6.24001	1.97093	5	0
4	5.50743	0.492732	14.3	0
5	5.41817	0.985464	51.8	1
6	3.65108	0.0864031	6.09	1
7	3.19595	0.246366	6.28	1
8	2.76465	0.492732	17.1	1
9	2.00507	0.109746	4.57	1
10	2.00507	0.578825	4.57	1
11	1.71896	0.123183	3.33	0
12	1.37804	0.246366	8.39	1
13	0.978233	0.118359	2.73	1
14	0.978233	0.264033	2.73	1
15	0.848752	0.0615915	1.67	0
16	0.663295	0.123183	3.18	1
17	0.502908	0.212605	7.32	1
18	0.288515	0.118991	1.07	1
19	0.232956	0.246366	7.67	1
20	0.0959621	0.104984	0.451	1
21	0.0718443	0.246366	6.71	1
22	0.0179744	0.0804931	0.201	1
23	0.015862	0.223585	4.8	1
24	0.000254066	0.0257771	0.0156	1
25	6.25444e-06	0.0349246	0.0996	1
26	3.86852e-11	0.000513953	6.24e-06	1
27	1.52262e-19	1.39171e-05	1.53e-08	1

Local minimum possible.

Trust Region Algorithm:

fminunc stopped because the final change in function value relative to its initial value is less than the selected value of the [function tolerance](#).
<[stopping criteria details](#)>

```
xhat =  
1.0000  
1.0000
```

```
fhat =  
1.5226e-19  
flag =  
3  
out =  
struct with fields:  
iterations: 27  
funcCount: 28  
stepsize: 1.3917e-05  
cgiterations: 23  
firstorderopt: 1.5332e-08  
algorithm: 'trust-region'  
message: 'Local minimum possible.'  
fminunc stopped because the final  
change in function value relative to its initial value is less than the selected  
value of the function tolerance.  
Stopping criteria details:  
Optimization stopped
```

MATLAB Optimization Toolbox: fminsearch

```
function f = rosenbrock(x)
% % Returns the Rosenbrock function value
% % Note: x should have two elements.
  f = 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;
end

x0 = [-1.5; -4];
opt = optimset('Display','iter','TolX',1e-08,...
              'TolFun',1e-08,'MaxIter',500,'MaxFunEvals',1000);

[xhat, fval] = fminsearch(@rosenbrock,x0,opt)
```

MATLAB Optimization Toolbox: fminsearch

Results:

Iteration	Func-count	min f(x)	Procedure
0	1	3912.5	
1	3	3912.5	initial simplex
2	5	3754.02	expand
3	7	3226.45	expand
4	9	2923.97	expand
5	11	2160.55	expand
6	12	2160.55	reflect
7	14	2004.75	reflect
8	16	1390.74	expand
9	17	1390.74	reflect
10	19	513.088	expand

99	192	9.32149e-18	contract inside
100	194	6.88373e-18	contract outside

Optimization terminated:

the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-08

and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-08

xhat =

1.0000

1.0000

fval =

6.8837e-18

Optimization in Chemical Engineering

Thank You

