Optimization

2D Optimization

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

References

- S. Chapra and R. Canale, Numerical Methods for Engineers
- A. Greenbaum and T. Chartier, Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms
- Carnahan and Wilkes, Applied Numerical Methods, University of Michigan, Class Notes, 1996.
- Burden and Faires, Numerical Analysis, 1993
- Pal, Numerical Analysis for Scientists and Engineers, 2007.
- M. Heath, Scientific Computing: An Introductory Survey
- wikipedia.org

Contents

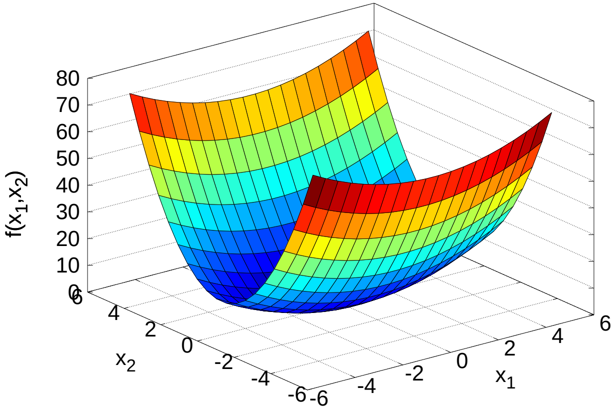
- Basics and Background
- Optimization: Single variable
  - Golden Section
  - Quadratic Interpolation
  - Newton's Method
- Multidimensional Optimization*
  - Steepest Descent
  - Newton's Method
  - BFGS Method: Quasi-Newton
- Miscellaneous

*In this class "multi" = 2, although we will sometimes use more general language

Multidimensional functions

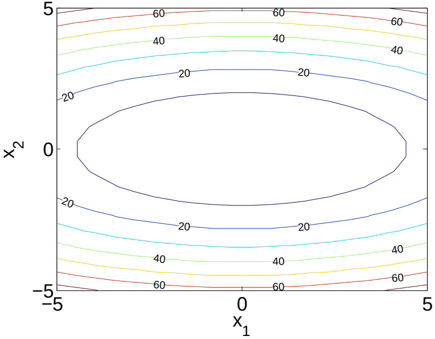
- Instead of just $f(x)$, we will now consider finding the *minima* of functions $f(\mathbf{x}) = f(x_1, x_2, ..., x_n)$
- Example: Consider the 2D function

$$f(x_1, x_2) = 0.5x_1^2 + 2.5x_2^2$$



2D Function

- We can also construct a contour plot



- An important concept in multidimensional optimization is the gradient of $f(\mathbf{x})$. For a 2D function such as the one here:

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial x_1}\mathbf{e}_1 + \frac{\partial f}{\partial x_2}\mathbf{e}_2$$

Gradient

- The gradient generalizes the concept of derivative to multiple dimensions
- Note that it is a vector, and has a "direction" in addition to a magnitude. This direction is important in optimization.
- We can also write it as a vector

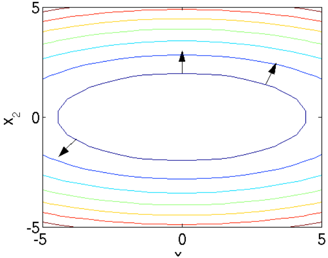
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}, \quad \text{assuming } \mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- We can evaluate the gradient of this particular function:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix} = x_1\mathbf{e}_1 + 5x_2\mathbf{e}_2$$

Gradient

- I plotted the direction of the gradient vector at a few different points on the contour map



- The three points and the corresponding normalized gradients are $(\mathbf{x}, \nabla f(\mathbf{x})/||\nabla f(\mathbf{x})||)$

$$\left(\begin{bmatrix}0.0\\2.0\end{bmatrix},\begin{bmatrix}0.0\\1.0\end{bmatrix}\right),\quad\left(\begin{bmatrix}3.0\\1.5\end{bmatrix},\begin{bmatrix}0.37\\0.93\end{bmatrix}\right),\quad\left(\begin{bmatrix}-3.8\\1.0\end{bmatrix},\begin{bmatrix}-0.61\\-0.80\end{bmatrix}\right)$$

Gradient

- Note that the gradient is perpendicular to contour lines
- The direction of ∇f tells us which way to travel in to gain elevation as quickly as possible
- The magnitude of ∇f tell us how much we gain by travelling in that direction
- This is similar to derivative of a single variable $f(x)$ where df/dx measures the "rate" at which $f(x)$ changes with x.
- Next we are going to consider a method called "steepest descent" to find the *minima* of a function $f(\mathbf{x})$ by travelling in the direction of $-\nabla f(\mathbf{x})$
- There is completely analogous method called "steepest ascent" to find the maxima by travelling in the direction of $\nabla f(\mathbf{x})$

Steepest Descent: Algorithm

1. $k = 0$; $\mathbf{x}_0 =$ initial guess
2. Compute the -ve gradient $\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$
3. Choose α_k to minimize $f(\mathbf{x}_k + \alpha \mathbf{s}_k)$. Note that this is a 1D problem, for which we have devised methods before. This is called a *line* search.
4. Update the solution: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$
5. Set $k = k + 1$, and go back to step 2, and repeat until convergence.

Example

Problem: Use steepest descent to minimize the function[†]

$$f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2,$$

where $\mathbf{x} = [x_1,\ x_2]^T$, and whose gradient is:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$$

Start with an initial guess of $\mathbf{x}_0 = [5,\ 1]^T$.

Solution:

$$\mathbf{x}_0 = [5,\ 1]^T, \quad \mathbf{s}_0 = -\nabla f(\mathbf{x}_0) = -[5,\ 5]^T$$

Therefore,

$$f(\mathbf{x}_0 + \alpha \mathbf{s}_0) = f\left(\begin{bmatrix} 5 \\ 1 \end{bmatrix} - \alpha \begin{bmatrix} 5 \\ 5 \end{bmatrix}\right) = f\left(\begin{bmatrix} 5 - 5\alpha \\ 1 - 5\alpha \end{bmatrix}\right)$$

[†]Problem from Heath, chapter 6.

Example

Thus,

$$\begin{aligned} f(\mathbf{x}_0 + \alpha \mathbf{s}_0) &= 0.5(5 - 5\alpha)^2 + 2.5(1 - 5\alpha)^2 \\ &= 75\alpha^2 - 50\alpha + 15 \end{aligned}$$

One can easily find the minima of this function by taking the derivative with respect to α which gives us $150\alpha_0 - 50 = 0$, or $\alpha_0 = 1/3$.

Thus,

$$\mathbf{x}_1 = \mathbf{x}_0 + (1/3)\mathbf{s}_0 = \begin{bmatrix} 3.333 \\ -0.667 \end{bmatrix}$$

We can now repeat the process until we are happy!

Or we can write the following matlab code.

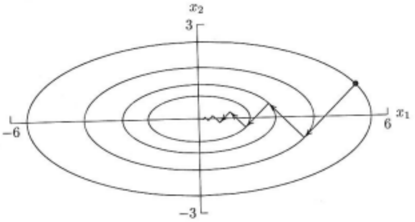
Steepest Descent: Matlab Program

```
% Steepest Descent Demo: use [x f n] = steepDescent([5;1], 1e-3)

function [xopt fopt nopt] = steepDescent(x0, tol)

  x = x0;
  k = 0;

  while(norm(gradf(x)) > tol)                      % Need to make gradf = 0

    s      = -gradf(x);
    falpha = @(alpha) f(x + alpha*s);
    alpha  = fminsearch(falpha,0.1);
    x      = x + alpha * s;
    k      = k + 1;

  end

  xopt = x;   fopt = f(x); nopt = k;

end

function Z = f(x)
  Z = 0.5*x(1)^2 + 2.5*x(2)^2;
end

function Z = gradf(x)
  Z = [x(1);5*x(2)];
end
```

Example

From Heath pg 278:

k	x_k^T		$f(x_k)$	$\nabla f(x_k)^T$	
0	5.000	1.000	15.000	5.000	5.000
1	3.333	−0.667	6.667	3.333	−3.333
2	2.222	0.444	2.963	2.222	2.222
3	1.481	−0.296	1.317	1.481	−1.481
4	0.988	0.198	0.585	0.988	0.988
5	0.658	−0.132	0.260	0.658	−0.658
6	0.439	0.088	0.116	0.439	0.439
7	0.293	−0.059	0.051	0.293	−0.293
8	0.195	0.039	0.023	0.195	0.195
9	0.130	−0.026	0.010	0.130	−0.130



What is the geometrical interpretation?

Hessian

- The Hessian of a multidimensional scalar function $f(\mathbf{x})$ is given by the symmetric square matrix

$$\mathbf{H}_f(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- Just as the gradient generalizes df/dx, the Hessian generalizes d^2f/dx^2.

Hessian

- In 2D, the Hessian is:

$$\mathbf{H}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

- Recall how $d^2 f(x^*)/dx^2$ told us whether x^* (obtained by solving $df/dx = 0$) was a maxima, minima or a saddle point
- The Hessian does the same job. If $\mathbf{x}^*$ is a solution to $\nabla f(\mathbf{x}) = \mathbf{0}$, then if $\mathbf{H}_f(\mathbf{x}^*)$ is

$$\begin{array}{lll} +\text{ve definite} & \implies & x^* \text{ is a minima} \\ \text{-ve definite} & \implies & x^* \text{ is a maxima} \\ \text{indefinite} & \implies & x^* \text{ is a saddle point} \end{array}$$

Newton's Method

- Recall 1D Newton's Method for optimization:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

- We could rewrite this expression as:

$$(x_{i+1} - x_i)f''(x_i) = -f'(x_i)$$

- We are going to generalize this method for multiple dimensions
- It is useful to visualize Newton's method as a quadratic approximation to a Taylor's series

Newton's Method

- That is consider

$$f(x+s) \approx f(x) + \frac{df}{dx}s + \frac{1}{2}\frac{d^2f}{dx^2}s^2.$$

- We can think of the RHS as a quadratic function in s which can be minimized

$$\frac{df(x+s)}{ds} = 0 \implies 0 + \frac{df}{dx} + \frac{d^2f}{dx^2}s = 0$$

- Leading to

$$s\frac{d^2f}{dx^2} = -\frac{df}{dx}$$

which is the same as Newton's method for optimization

Newton: Multidimensional case

- We can repeat the Taylor series expansion for $f(\mathbf{x})$.

$$f(\mathbf{x}+\mathbf{s}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{s} + \frac{1}{2}\mathbf{s}^T \mathbf{H}_f(\mathbf{x})\mathbf{s}$$

  Note that for 1D this collapses to the previous expression.

- Taking the derivative, leads us to:

$$\mathbf{H}_f(\mathbf{x})\mathbf{s} = -\nabla f(\mathbf{x})$$

  Compare with the 1D case:

$$s\frac{d^2 f}{dx^2} = -\frac{df}{dx}$$

- This allows us to write an algorithm for Newton's method

Newton's Method: Algorithm

1. $k = 0$; $\mathbf{x}_0 =$ initial guess
2. Compute the gradient $\nabla f(\mathbf{x}_k)$ and the Hessian $\mathbf{H}_f(\mathbf{x}_k)$
3. Solve $\mathbf{H}_f(\mathbf{x}_k)\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$ for $\mathbf{s}_k$
4. Update the solution: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
5. Set $k = k + 1$, and go back to step 2, and repeat until convergence.

Example

Problem: Solve the previous example again, this time using Newton's method

$$f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2,$$

Solution:
The gradient and Hessian are given by:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}, \quad \mathbf{H}_f(\mathbf{x}_k) = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

We solve the system (with $\mathbf{x}_0 = [5,\ 1]^T$)

$$\begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} \mathbf{s}_0 = -\begin{bmatrix} 5 \\ 5 \end{bmatrix} \implies \mathbf{s}_0 = \begin{bmatrix} -5 \\ -1 \end{bmatrix}$$

Example

- This implies
$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{s}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
which is the true solution
- It is not surprising that Newton's method converged in 1 iteration since the $f(\mathbf{x})$ was quadratic
- In general the convergence rate is quadratic, but the method can veer off unless we start close enough to the solution
- Note: No line search required, but we had to determine a Hessian matrix and solve a linear system at each iteration
- In damped Newton methods, a line search is added to make the method more robust.

Quasi-Newton Methods

- Newton's method converges rapidly once you are close to the solution. But it doesn't come cheap.
- For a n-dimensional problem, each iteration requires $\mathcal{O}(n^2)$ function evaluations to form the gradient and the Hessian, and $\mathcal{O}(n^3)$ operations to solve the linear system.
- To reduce overhead, quasi-Newton methods have been developed which seek to replace the step:

  $$\mathbf{H}_f(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\nabla f(\mathbf{x}_k)$$

  with

  $$\mathbf{B}_k(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\alpha_k \nabla f(\mathbf{x}_i)$$

  where $\mathbf{B}$ is an approximation to the Hessian matrix, and may be obtained by secant updating and α_k is a line-search parameter.

BFGS Method

- A popular secant updating method named after its co-inventors: Broyden, Fletcher, Goldfarb and Shanno.
- Initially set $\mathbf{B}_0 = \mathbf{I}$, which means the first step is in the negative gradient direction (like steepest descent).
- Unlike Newton's method, the second derivatives (Hessian) do not have to be pre-computed.
- It is built up over time.
- Convergence is superlinear.
- We consider a simple algorithm (better implementations update a factorization of the matrix $\mathbf{B}$ rather than the matrix itself)

BFGS Algorithm

1. $k = 0$; $\mathbf{x}_0 =$ initial guess
2. Set $\mathbf{B}_0 = \mathbf{I}$ as the initial Hessian approximation
3. Compute the gradient $\nabla f(\mathbf{x}_k)$
4. Solve $\mathbf{B}_k \mathbf{s}_k = -\nabla f(\mathbf{x}_k)$ for $\mathbf{s}_k$
5. Update the solution: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$
6. Set $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$
7. Update the Hessian

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k}$$

8. Set $k = k + 1$, and go back to step 4, and repeat until convergence.

Example

Problem: Solve the previous example again, this time using BFGS method

$$f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2,$$

Solution:

The gradient and approximate Hessian are given by:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}, \quad \mathbf{B}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We solve the system (with $\mathbf{x}_0 = [5,\ 1]^T$).

The first step is simply: $\mathbf{I}\mathbf{s}_0 = -\nabla f(\mathbf{x}_0) = -[5,\ 5]^T$

$\implies \mathbf{x}_1 = [5,\ 1]^T + [-5,\ -5]^T = [0,\ -4]^T$

We can update the approximate Hessian to

$$\mathbf{B}_1 = \begin{bmatrix} 0.667 & 0.333 \\ 0.333 & 0.667 \end{bmatrix}$$

Example

We can continue to get the sequence:

k	x_1	x_2	$f(\mathbf{x})$
0	5.0000	1.0000	15.0000
1	0.0000	-4.0000	40.0000
2	-2.2222	0.4444	2.9630
3	0.8163	0.0816	0.3499
4	-0.0092	-0.0153	0.0006
5	-0.0005	0.0009	0.0000

using the following Matlab code:

Matlab Code

```
% BFGS Demo: use as [x f n] = bfgs([5;1], 1e-3);

function [xopt fopt nopt] = bfgs(x0, tol)

  x = x0;
  n = length(x);
  B = eye(n);
  k = 0;

  while(norm(gradf(x)) > tol)                  % Need to make gradf ~ 0
    df     = gradf(x);
    s      = B\(-df);
    x      = x + s;
    y      = gradf(x) - df;
    B      = B + (y*y')/(y'*s) - (B*s*s'*B)/(s'*B*s);
    k      = k + 1;
  end

  xopt = x;
  fopt = f(x);
  nopt = k;

end

function Z = f(x)
  Z = 0.5*x(1)^2 + 2.5*x(2)^2;
end

function Z = gradf(x)
  Z = [x(1);5*x(2)];
end
```

Summary

- Methods for optimization in 1D have "counterparts" in methods for the solution of nonlinear equations:

$$\begin{array}{rcl} \text{Golden Search} & \rightarrow & \text{Bisection} \\ \text{Parabolic Interpolation} & \rightarrow & \text{Regula Falsi} \\ \text{Newton } (f(x)=0) & \rightarrow & \text{Newton } (f'(x)=0) \end{array}$$

  and resemble many of their properties (linear/quadratic convergence etc.).
- Multidimensional optimization requires knowledge of gradients and sometimes Hessians, which generalize first and second order derivatives.
- Steepest descent moves in the direction of negative gradient - results in zig-zag moves (a method called conjugate-gradient fixes this problem).

Summary

- Multidimensional Newton's method generalizes Newton's method for optimization in 1D. It is fast, but requires significant work (deriving the Hessian, and solving a linear system).
- BFGS is an extremely popular secant-updating method which works with an approximate Hessian.