Overview of Multivariate Optimization Topics

- Problem definition
- Algorithms
  - Cyclic coordinate method
  - Steepest descent
  - Conjugate gradient algorithms
  - PARTAN
  - Newton’s method
  - Levenberg-Marquardt
- Concise, subjective summary

Multivariate Optimization Overview

- The “unconstrained optimization” problem is a generalization of the line search problem
- Find a vector $\mathbf{a}$ such that
  $$\mathbf{a}^* = \arg\min_{\mathbf{a}} f(\mathbf{a})$$
- Note that there are no constraints on $\mathbf{a}$
- Example: Find the vector of coefficients ($\mathbf{w} \in \mathbb{R}^{p \times 1}$) that minimize the average absolute error of a linear model
- Akin to a blind person trying to find their way to the bottom of a valley in a multidimensional landscape
- We want to reach the bottom with the minimum number of “cane taps”
- Also vaguely similar to taking core samples for oil prospecting

Example 1: Optimization Problem

![Graph of an optimization problem](image)
**Example 1: Optimization Problem**

```matlab
function [] = OptimizationProblem();
%==============================================================================
% User-Specified Parameters
%==============================================================================
x = -5:0.05:5;
y = -5:0.05:5;
%==============================================================================
% Evaluate the Function%==============================================================================
[X,Y] = meshgrid(x,y);
[Z,G] = OptFn(X,Y);
functionName = 'OptimizationProblem';
fileIdentifier = fopen([functionName'.tex'],'w');
%==============================================================================
% Contour Map
%==============================================================================
figure;
FigureSet(2,'Slides');
contour(x,y,Z,50);
xlabel('a_1');
ylabel('a_2');
zoom on;
AxisSet(8);
fileName = sprintf('%s-%s',functionName,'Contour');
fclose(fileIdentifier);
```

**Example 1: Optimization Problem**

```matlab
function [] = OptimizationProblem();
%==============================================================================
% User-Specified Parameters
%==============================================================================
x = -5:0.05:5;
y = -5:0.05:5;
%==============================================================================
% Evaluate the Function%==============================================================================
[X,Y] = meshgrid(x,y);
[Z,G] = OptFn(X,Y);
functionName = 'OptimizationProblem';
fileIdentifier = fopen([functionName'.tex'],'w');
%==============================================================================
% Contour Map
%==============================================================================
figure;
FigureSet(2,'Slides');
contour(x,y,Z,50);
xlabel('a_1');
ylabel('a_2');
zoom on;
AxisSet(8);
fileName = sprintf('%s-%s',functionName,'Contour');
fclose(fileIdentifier);
```
Global Optimization?

- In general, all optimization algorithms find a local minimum in as few steps as possible.
- There are also “global” optimization algorithms based on ideas such as:
  - Evolutionary computing
  - Genetic algorithms
  - Simulated annealing
- None of these guarantee convergence in a finite number of iterations.
- All require a lot of computation.
Cyclic Coordinate Method

1. For \( i = 1 \) to \( p \),
   
   \[ a_i := \arg\min_{\alpha} f([a_1, a_2, \ldots, a_{i-1}, \alpha, a_{i+1}, \ldots, a_p]) \]

2. Loop to 1 until convergence
   + Simple to implement
   + Each line search can be performed semi-globally to avoid shallow local minima
   + Can be used with nominal variables
   + \( f(a) \) can be discontinuous
   + No gradient required
   - Very slow compared to gradient-based optimization algorithms
   - Usually only practical when the number of parameters, \( p \), is small
   - There are modified versions with faster convergence

Optimization Comments

- Ideally, when we construct models we should favor those which can be optimized with few shallow local minima and reasonable computation
- Graphically you can think of the function to be minimized as the elevation in a complicated high-dimensional landscape
- The problem is to find the lowest point
- The most common approach is to go downhill
- The gradient points in the most "uphill" direction
- The steepest downhill direction is the opposite of the gradient
- Most optimization algorithms use a line search algorithm
- The methods mostly differ only in the way that the "direction of descent" is generated

Optimization Algorithm Outline

- The basic steps of these algorithms is as follows
  1. Pick a starting vector \( a \)
  2. Find the direction of descent, \( d \)
  3. Move in that direction until a minimum is found:
     \[ \alpha^* := \arg\min_{\alpha} f(a + \alpha d) \]
     \[ a := a + \alpha^* d \]
  4. Loop to 2 until convergence
- Most of the theory of these algorithms is based on quadratic surfaces
- Near local minima, this is a good approximation
- Note that the functions should (must) have continuous gradients (almost) everywhere

Example 2: Cyclic Coordinate Method
Example 2: Cyclic Coordinate Method

Example 2: Relevant MATLAB Code

```matlab
function [] = CyclicCoordinate();
%clear all;
%close all;
ns = 26;
x = -3;
y = 1;
b0 = -1;
ls = 30;
a = zeros(ns,2);
f = zeros(ns,1);
[z,dzx,dzy] = OptFn(x,y);
a(1,:) = [x y];
f(1) = z;
for cnt = 2:ns,
    ifrem(cnt,2) == 1,
        d = [1 0]'; % Along x direction
    else
        d = [0 1]'; % Along y direction
    end;
    [b,fmin] = LineSearch([x y]',d,b0,ls);
x = x + b*d(1);
y = y + b*d(2);
end;
```

J. McNames Portland State University ECE 4/557 Multivariate Optimization Ver. 1.14 19
\[
\begin{align*}
\text{s(cnt,:) = [x y];} \\
\text{f(cnt) = fmin;} \\
\text{end;} \\
\end{align*}
\]

\[
\begin{align*}
[x,y] &= \text{meshgrid}(0:-0.01:0.01,3*(-0.01:0.001:0.002)); \\
[x,dx,dy] &= \text{OptFn}(x,y); \\
[xopt,idd] &= \text{min}(x); \\
[xopt,idd2] &= \text{min}(xopt); \\
id1 &= \text{id}(idd); \\
xopt2 &= x(id1,idd2); \\
yopt2 &= y(id1,idd2); \\
\end{align*}
\]

\[
\begin{align*}
[x,y] &= \text{meshgrid}(1.883*(-0.02:0.001:0.02),-2.963*(-0.02:0.001:0.002)); \\
[x,dx,dy] &= \text{OptFn}(x,y); \\
xopt2 &= x(id1,idd2); \\
yopt2 &= y(id1,idd2); \\
\end{align*}
\]

\[
\begin{align*}
\text{figure;} \\
\text{FigureSet(1,4.5,2.75);} \\
\text{x = Optim(x,y);} \\
\text{z = Optim(x,y,z);} \\
\text{contour(x,y,z,50);} \\
\end{align*}
\]
Steepest Descent

The gradient of the function $f(a)$ is defined as the vector of partial derivatives:

$$\nabla_a f(a) \equiv \left[ \frac{\partial f(a)}{\partial a_1} \ \frac{\partial f(a)}{\partial a_2} \ \ldots \ \frac{\partial f(a)}{\partial a_p} \right]^T$$

- It can be shown that the gradient, $\nabla_a f(a)$, "points" in the direction of maximum ascent
- The negative of the gradient, $-\nabla_a f(a)$, "points" in the direction of maximum descent
- A vector $d$ is a direction of descent if there exists a $\epsilon$ such that $f(a + \lambda d) < f(a)$ for all $0 < \lambda < \epsilon$
- It can also be shown that $d$ is a direction of descent iff $(\nabla_a f(a))^T d < 0$
- The algorithm of steepest descent uses $d = -\nabla_a f(a)$
- The most fundamental of all algorithms for minimizing a continuously differentiable function

Steepest Descent

+ Very stable algorithm
– Can converge very slowly once near the local minima where the surface is approximately quadratic
Example 3: Relevant MATLAB Code

```matlab
function [] = SteepestDescent();
  %clear all;
  close all;
  ns = 26;
  x = -3;
  y = 1;
  b0 = 0.01;
  ls = 30;
  a = zeros(ns,2);
  f = zeros(ns,1);
  [z,g] = OptFn(x, y);
  a(1,:) = [x y];
  f(1) = z;
  d = -g/norm(g);
  for cnt = 2:ns,
    [b,fmin] = LineSearch([x y]',d,b0,ls);
    x = x + b*d(1);
    y = y + b*d(2);
    [z,g] = OptFn(x, y);
    d = -g;\
    d = d/norm(d);
    a(cnt,:) = [x y];
    f(cnt) = z;
  end;
  [x,y] = meshgrid(0+(-0.01:0.001:0.01),3+(-0.01:0.001:0.01));
  [z,dzx,dzy] = OptFn(x,y);
  [zopt,id1] = min(z);
  [zopt2,id2] = min(zopt);
  id1 = id1(id2);
  xopt = x(id1,id2);yopt = y(id1,id2);
  [x,y] = meshgrid(1.883+(-0.02:0.001:0.02),-2.963+(-0.02:0.001:0.02));
  [z,dzx,dzy] = OptFn(x,y);
  [zopt2 ,id1] = min(z);
  [zopt2 ,id2] = min(zopt2);
  id1 = id1(id2);
  xopt2 = x(id1,id2);yopt2 = y(id1,id2);
  [zopt zopt2]
  figure;
  FigureSet(1,4.5,2.75);
  [x,y] = meshgrid(5:0.1:5.5:0.1:5.5);
  z = OptFn(x,y);
  contour(x,y,z,50);
  h = get(gca,'Children');
  set(h,'LineWidth',0.2);
  axis('square');
  hold on;
```
Conjugate Gradient Algorithms

1. Take a steepest descent step

2. For $i = 2$ to $p$
   - $\alpha := \arg\min_{\alpha} f(a + \alpha d)$
   - $a : = a + \alpha d$
   - $g_i : = \nabla f(a)$
   - $\beta : = \frac{g_i^T g_i}{g_{i-1}^T g_{i-1}}$
   - $d : = -g_i + \beta d_i$

3. Loop to 1 until convergence
   - Based on quadratic approximations of $f$
   - Called the Fletcher-Reeves method
Example 4: Relevant MATLAB Code

```matlab
function [] = FletcherReeves();
%clear all;
close all;
ns = 26;
x = -3;
y = 1;
b0 = 0.01;
ls = 30;
a = zeros(ns,2);
f = zeros(ns,1);
[a(:,1), a(:,2)] = OptFn(x, y);
f(1) = z;
d = -g/|g|; % First direction
for cnt = 2:ns,
    [b,fmin] = LineSearch([x y]',d,b0,ls);
    x = x + b*d(1);
    y = y + b*d(2);
    go = g;
    [z,g] = OptFn(x, y);
    beta = (g'*g)/(go'*go);
d = -g + beta*d;
a(cnt,:) = [x y];
f(cnt) = z;
end;
[x,y] = meshgrid(-0.01:0.01:0.01,-0.01:0.01:0.01);
z = OptFn(x,y);
[zopt1,id1] = min(z);
[zopt2,id2] = min(zopt1);
[xopt1,yopt1] = x(id1,id2);
yopt = y(id1,id2);
[x,y] = meshgrid(-0.02:0.001:0.02,-0.02:0.001:0.02);
z = OptFn(x,y);
[zopt2,id1] = min(z);
[zopt2,id2] = min(zopt2);
[xopt2,yopt2] = x(id1,id2);
yopt2 = y(id1,id2);
figure;
FigureSet(1,4,5,2.75);
[x,y] = meshgrid(-5:0.1:5,-5:0.1:5);
z = OptFn(x,y);
contour(x,y,z,50);
h = get(gca,'Children');
set(h,'LineWidth',0.2);
axis('square');
hold on;
h = plot(a(:,1),a(:,2),'k',a(:,1),a(:,2),'r');
set(h(1),'LineWidth',1.2);
set(h(2),'LineWidth',0.6);
h = plot(xopt,yopt,'kx',xopt,yopt,'rx');
set(h(1),'LineWidth',1.5);
set(h(2),'LineWidth',0.5);
set(h(1),'MarkerSize',5);
set(h(2),'MarkerSize',4);
hold off;
label('x');
ylabel('y');
zoom on;
FigureSet(8);
AxsSet(8);
print -depsc FletcherReevesContourA;
figure;
FigureSet(2,4,4,2.75);
%xerr = (sum((x-ones(ns,1)*[xopt; yopt])).^2).

h = plot(xopt2,yopt2,'kx',xopt2,yopt2,'rx');
set(h(1),'LineWidth',1.5);
set(h(2),'LineWidth',0.5);
set(h(1),'MarkerSize',5);
set(h(2),'MarkerSize',4);
hold off;
label('x');
ylabel('y');
zoom on;
FigureSet(2,4,4,2.75);
[x,y] = meshgrid(-5:0.01:5,-5:0.01:5);
[z,dzx,dzy] = OptFn(x,y);
[zopt,id1] = min(z);
[zopt2,id2] = min(zopt);
[xopt1,yopt1] = x(id1,id2);
yopt = y(id1,id2);
[x,y] = meshgrid(-0.01:0.001:0.01,-0.01:0.001:0.01);
[z,dzx,dzy] = OptFn(x,y);
[zopt1,id1] = min(z);
[zopt2,id2] = min(zopt1);
[xopt2,yopt2] = x(id1,id2);
yopt2 = y(id1,id2);
figure;
FigureSet(1,4,5,2.75);
[x,y] = meshgrid(-0.01:0.01:0.01,-0.01:0.01:0.01);
z = OptFn(x,y);
contour(x,y,z,50);
h = get(gca,'Children');
set(h,'LineWidth',0.2);
axis('square');
hold on;
h = plot(a(:,1),a(:,2),'k',a(:,1),a(:,2),'r');
set(h(1),'LineWidth',1.2);
set(h(2),'LineWidth',0.6);
h = plot(xopt,yopt,'kx',xopt,yopt,'rx');
set(h(1),'LineWidth',1.5);
set(h(2),'LineWidth',0.5);
set(h(1),'MarkerSize',5);
set(h(2),'MarkerSize',4);
hold off;
label('x');
ylabel('y');
zoom on;
FigureSet(8);
AxsSet(8);
print -depsc FletcherReevesContourB;
```
Conjugate Gradient Algorithms Continued

• There is also a variant called Polak-Ribiere where

\[ \beta := \frac{(g_i - g_{i-1})^T g_i}{g_{i-1}^T g_{i-1}} \]

+ Only requires the gradient
+ Converges in a finite No. steps when \( f(a) \) is quadratic and perfect line searches are used
  – Less stable numerically than steepest descent
  – Sensitive to inexact line searches
Example 5: MATLAB Code

```matlab
function [] = PolakRibiere();
%clear all;
close all;
ns = 26;
x = -3;
y = 1;
b0 = 0.01;
ls = 30;
a = zeros(ns,2);
f = zeros(ns,1);
[z,g] = OptFn(x, y);
a(1,:) = [x y];
f(1) = z;
d = -g/norm(g); % First direction
for cnt = 2:ns,
    [b,fmin] = LineSearch([x y]',d,b0,ls);
    x = x + b*d(1);
y = y + b*d(2);
go = g;
    % Old gradient
    [z,g] = OptFn(x, y);
    beta = ((g-go)'*g)/(go'*go);
    d = -g + beta*d;
    a(cnt,:) = [x y];
f(cnt) = z;
end;
[x,y] = meshgrid(0+(-0.01:0.001:0.01),3+(-0.01:0.001:0.01));
[z,dzx,dzy] = OptFn(x,y);
[zopt,id1] = min(z);
[zopt,id2] = min(zopt);
id1 = id1(id2);

xopt = x(id1,id2);
yopt = y(id1,id2);
[x,y] = meshgrid(1.883+(-0.02:0.001:0.02),-2.963+(-0.02:0.001:0.02));
[z,dzx,dzy] = OptFn(x,y);
[zopt2 ,id1] = min(z);
[zopt2 ,id2] = min(zopt2);
id1 = id1(id2);

xopt2 = x(id1,id2);
yopt2 = y(id1,id2);
figure;
FigureSet(1,4.5,2.75);
[x,y] = meshgrid(-5:0.1:5,-5:0.1:5);
z = OptFn(x,y);
contour(x,y,z,50);
h = get(gca,'Children');
set(h,'LineWidth',0.2);
axis('square');
hold on;
```
Parallel Tangents (PARTAN)

1. First gradient step
   - $d := \nabla f(a)$
   - $\alpha := \arg\min_{\alpha} f(a + \alpha d)$
   - $s_p := \alpha d$
   - $a := a + s_p$

2. Gradient Step
   - $d_g := \nabla f(a)$
   - $\alpha := \arg\min_{\alpha} f(a + \alpha d)$
   - $s_g := \alpha d$
   - $a := a + s_g$

3. Conjugate Step
   - $d_p := s_p + s_g$
   - $\alpha := \arg\min_{\alpha} f(a + \alpha d)$
   - $s_p := \alpha d$
   - $a := a + s_p$

4. Loop to 2 until convergence
First two steps are steepest descent

Thereafter, each iteration consists of two steps

1. Search along the direction
   \[ d_i = a_i - a_{i-2} \]
   where \( a_i \) is the current point and \( a_{i-2} \) is the point from two steps ago

2. Search in the direction of the negative gradient
   \[ d_i = -\nabla f(a_i) \]
cnt = 2;
while cnt < ns,
  % Gradient step
  [z,g] = OptFn(x,y);
  d = -g/norm(g); % Direction
  [bg,fmin] = LineSearch([xy]',d,b0,ls);
  xg = x + bg*d(1);
  yg = y + bg*d(2);
  cnt = cnt + 1;
  a(cnt,:) = [xg yg];
  f(cnt) = OptFn(xg,yg);
  fprintf('G : %d %5.3f
',cnt,f(cnt));
  if cnt == ns,
    break;
  end;

  % Conjugate
  d = -g/norm(g); % First direction
  [bp,fmin] = LineSearch([xy]',d,b0,ls);
  x = x + bp*d(1);
  y = y + bp*d(2);
  a(2,:) = [x y];
  f(2) = fmin;
end;

% Update anchor point
xa = xg; ya = yg;

example 6: MATLAB Code

function [] = Partan()
%clear all;
%close all;
ns = 26;
x = -3;
y = 1;
b0 = 0.01;
ls = 30;
a = zeros(ns,2);
f = zeros(ns,1);
[z,g] = OptFn(x,y);
a(1,:) = [x y];
f(1) = z;
xa = x;
ya = y;

% First step - substitute for a Conjugate step
b = -g/norm(g); % First direction
[bp,fmin] = LineSearch([xy]',b,b0,100);
x = x + bp*b(1);
y = y + bp*b(2);
[a(2,:),f(2)] = [x y];
 end;

% Could not move - do another gradient update
if cnt == ns,
  break;
else
  % Line search in conjugate direction was successful
  fprintf('G2: %d %5.3f
',cnt,f(cnt));
  [z,g] = OptFn(x,y);
  d = -g/norm(g); % Direction
  [bp,fmin] = LineSearch([xy]',d,b0,ls);
  x = x + bp*d(1);
  y = y + bp*d(2);
  fprintf('A1: %d %5.3f
',cnt,f(cnt));
end;

% Update anchor point
xa = xg; ya = yg;

% Two steps...

[x,y] = meshgrid(0+(-0.01:0.001:0.01),3+(-0.01:0.001:0.01));
[z,dzx,dzy] = OptFn(x,y);
[zopt,id1] = min(z);
id2 = id1(id2);
xopt = x(id1,id2);
yopt = y(id1,id2);

example 6: PARTAN
\( \text{xlim}([-0.02,0.001,0.02]; \) 
\( \text{ylim}([-0.02,0.001,0.02]); \) 
\( \text{grid on; set(gca,'Box','Off'); AxisSet(8); print -depscPartanPositionError; \) 
\( \text{figure; FigureSet(2,4.5,2.75); k = 1:ns; h = plot(k-1,f,'b',k-1,xerr,'r'); set(h(1),'LineWidth',1.5); set(h(2),'LineWidth',0.5); set(h(1),'MarkerSize',5); set(h(2),'MarkerSize',4); hold off; xlabel('Iteration'); ylabel('Euclidean PositionError'); zoom on; \) 

\( \text{J. McNames Portland State University ECE 4/557 Multivariate Optimization Ver. 1.14} \) 

\( yopt = y(id1,id2); [x,y] = meshgrid(yopt(1.483*(-0.02:0.001:0.02)),yopt(2*(-0.02:0.001:0.02))); [x,dx,dy] = OptPn(x,y); [zopt2,id1] = min(z); id1 = id1(id2); zopt2 = z(id1,id2); yopt2 = y(id1,id2); \) 

\( \text{figure; FigureSet(1,4.5,2.75); [x,y] = meshgrid([0:5:0.1:5,-0.1:0.5]; z = OptPn(x,y); h = get(gca,'Children'); set(h,'LineWidth',0.2); axis('square'); hold on; h = plot(a(:,1),a(:,2),'k',a(:,1),a(:,2),'r'); set(h(1),'LineWidth',1.2); set(h(2),'LineWidth',0.6); hold off; xlabel('X'); ylabel('Y'); zoom on; AxisSet(8); print -depscPartanContourA; \) 

\( \text{PARTAN Pros and Cons} \) 

\( + \) For quadratic functions, converges in a finite number of steps
\( + \) Easier to implement than 2nd order methods
\( + \) Can be used with large number of parameters
\( + \) Each (composite) step is at least as good as steepest descent
\( + \) Tolerant of inexact line searches
\( - \) Each (composite) step requires two line searches

\( \text{J. McNames Portland State University ECE 4/557 Multivariate Optimization Ver. 1.14} \)
Newton’s Method

\[ a_{k+1} = a_k - H(a_k)^{-1} \nabla f(a_k) \]

where \( \nabla f(a_k) \) is the gradient and \( H(a_k) \) is the hessian of \( f(a) \),

\[
H(a_k) \equiv
\begin{bmatrix}
\frac{\partial^2 f(a)}{\partial a^2_1} & \frac{\partial^2 f(a)}{\partial a_1 \partial a_2} & \cdots & \frac{\partial^2 f(a)}{\partial a_1 \partial a_p} \\
\frac{\partial^2 f(a)}{\partial a_2 \partial a_1} & \frac{\partial^2 f(a)}{\partial a_2^2} & \cdots & \frac{\partial^2 f(a)}{\partial a_2 \partial a_p} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f(a)}{\partial a_p \partial a_1} & \frac{\partial^2 f(a)}{\partial a_p \partial a_2} & \cdots & \frac{\partial^2 f(a)}{\partial a_p^2}
\end{bmatrix}
\]

- Based on a quadratic approximation of the function \( f(a) \)
- If \( f(a) \) is quadratic, converges in one step
- If \( H(a) \) is positive-definite, the problem is well defined near local minima where \( f(a) \) is nearly quadratic
Example 7: Relevant MATLAB Code

```matlab
function [] = Newtons();
%clear all;
close all;
ns = 100;
x = -3; % Starting x
y = 1; % Starting y
b0 = 1;
a = zeros(ns,2);
f = zeros(ns,1);
[z,g,H] = OptFn(x, y);
a(1,:) = [x y];
f(1) = z;
forcnt = 2:ns,
d = -inv(H)*g;
if d'*g<0, % Revert to steepest descent if is not direction of descent
  fprintf('(%2d of %2d) Min. Eig %5.3f Reverting...\n',cnt,ns,min(eig(H)));
  d = -g;
end;
d = d/norm(d);
[b,fmin] = LineSearch([x y],d,b0,100);
a(cnt,:) = (a(cnt-1,:) - inv(H)*g)'; % Pure Newton's Method
x = x + b*d(1);
```
Newton’s Method Pros and Cons

\[ a_{k+1} = a_k - H(a_k)^{-1} \nabla f(a_k) \]

+ Very fast convergence near local minima

− Not guaranteed to converge (may actually diverge)

− Requires \( p \times p \) Hessian

− Requires a \( p \times p \) matrix inverse that uses \( O(p^3) \) operations

Levenberg-Marquardt

1. Determine if \( \epsilon_k I + H(a_k) \) is positive definite. If not, \( \epsilon_k := 4\epsilon_k \) and repeat.

2. Solve the following equation for \( a_{k+1} \)

\[ [\epsilon_k I + H(a_k)] (a_{k+1} - a_k) = -\nabla f(a_k) \]

3. \[ r_k = \frac{f(a_k) - f(a_{k+1})}{q(a_k) - q(a_{k+1})} \]

where \( q(a) \) is the quadratic approximation of \( f(a) \) based on the \( f(a), \nabla f(a), \) and \( H(a_k) \)

4. If \( r_k < 0.25 \), then \( \epsilon_{k+1} := 4\epsilon_k \)

   If \( r_k > 0.75 \), then \( \epsilon_{k+1} := \frac{1}{2} \epsilon_k \)

   If \( r_k \leq 0 \), then \( a_{k+1} := a_k \)

5. If not converged, \( k := k + 1 \) and loop to 1.
Levenberg-Marquardt Comments

- Similar to Newton’s method
- Has safety provisions for regions where quadratic approximation is inappropriate
- Compare

Newton’s: \[ a_{k+1} = a_k - H(a_k)^{-1} \nabla f(a_k) \]
LM: \[ [\epsilon I + H(a_k)](a_{k+1} - a_k) = -\nabla f(a_k) \]

- If \( \epsilon = 0 \), these are equivalent
- If \( \epsilon \to \infty \), \( a_{k+1} \to a_k \)
- \( \epsilon \) is chosen to ensure that the smallest eigenvalue of \( H(a_k) \) is positive and sufficiently large (\( \geq \delta \))
Example 8: Levenberg-Marquardt Conjugate Gradient

\[ y = a(cnt,2); \]
\[ z = a(cnt,:); \]
\[ t = z + \gamma \cdot x' \cdot d; \]
\[ x = a(cnt,:); \]
\[ y = a(cnt,2); \]
\[ a(cnt,:) = \{x y\}; \]
\[ f(cnt) = \text{OptFn}(x,y); \]
\[ \text{disp}\{\{cnt a(cnt,:) f(cnt) r eta\}\} \]

Example 8: Relevant MATLAB Code

```matlab
function [] = LevenbergMarquardt();
    clear all;
    close all;
    x = zeros(2,1);
    y = zeros(1,1);
    [x0,y0] = OptFn(x,y);
    x(1,:) = [x y];
    f(1) = x0;
    eta = 0.0001;
    ap = [x y];
    zn = x0;
    qo = zn;
    for cnt = 2:ns,
        [zn,g,H] = OptFn(x,y);
        a(cnt,:) = (ap - inv(eta*eye(2)+H)*g )';
        x = a(cnt ,1);
        y = a(cnt ,2);
        f(cnt) = OptFn(x,y);
        end;
    end;
end;
```

Example 8: Relevant MATLAB Code

```matlab
function [] = LevenbergMarquardt();
    clear all;
    close all;
    ns = 26;
    x = -3; % Starting x
    y = 1; % Starting y
    eta = 0.0001;
    x = zeros(ns,2);
    y = zeros(ns,1);
    [x0,y0] = OptFn(x,y);
    x(1,:) = [x y];
    f(1) = x0;
    while min(eig(eta*eye(2)+H))<0,
        eta = eta * 4;
    end;
    a(cnt,:) = (ap - inv(eta*eye(2)+H)*g )';
    x = a(cnt,1);
end;
```
Levenberg-Marquardt Pros and Cons

\[ \epsilon_k I + H(a_k) \left( a_{k+1} - a_k \right) = -\nabla f(a_k) \]

- Many equivalent formulations
  + No line search required
  + Can be used with approximations to the hessian
  + Extremely fast convergence (2nd order)
    - Requires gradient and hessian (or approximate hessian)
  - Requires \(O(p^3)\) operations for each solution to the key equation
## Optimization Algorithm Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Convergence</th>
<th>Stable</th>
<th>$\nabla f(a)$</th>
<th>$H(a)$</th>
<th>LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclic Coordinate</td>
<td>Slow</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Steepest Descent</td>
<td>Slow</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Conjugate Gradient</td>
<td>Fast</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>PARTAN</td>
<td>Fast</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Newton’s Method</td>
<td>Very Fast</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>Very Fast</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>