

# Computational Optimization

Last of unconstrained 2/26



# Half-way there

- **Minimize  $f(x)$**  (objective function)  
**subject to  $x \in \mathcal{S}$**  (constraints)
- Can characterize problem by type of objective functions and constraints
- NEOS Optimization Guide

<http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/>





# Optimization Recipes

Optimization algorithms are like recipes using common ingredients:


- Step-size
- Trust regions
- Newton's method
- Quasi-Newton
- Conjugate directions ....

Just stir up the right combination





# Some other ingredients

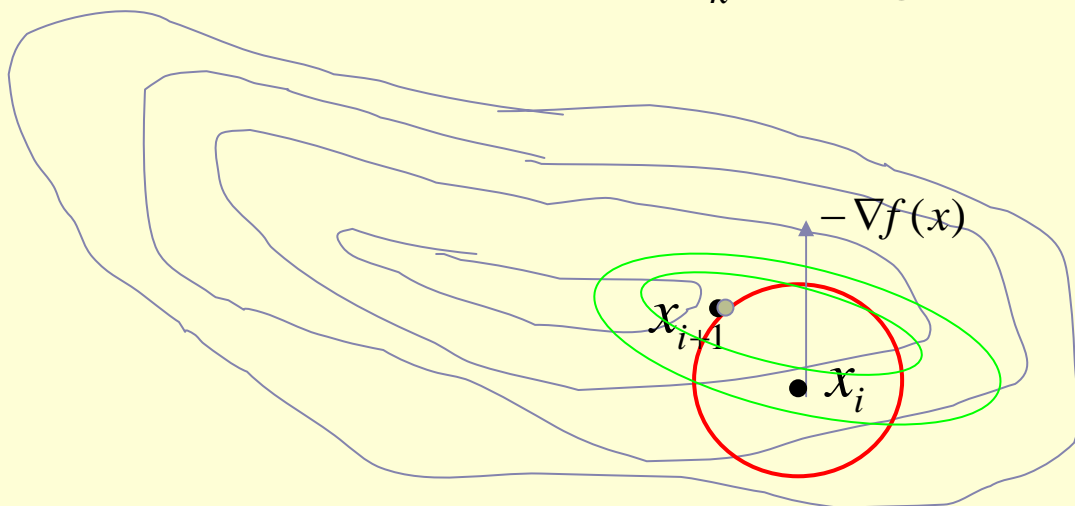
- Trust Region Methods
  - Limited Memory Quasi Newton
  - Linear Least Square
  - Nonlinear Least Squares
  - Finite difference methods
  - Automatic Differentiation
- 

# Trust Region Methods

- Alternative to line search methods
- Optimize **quadratic model** of objective within the **“trust region”**


$$p_k \in \arg \min f(x_i) + \nabla f(x_i)' p + \frac{1}{2} p' B_k p$$


$$s.t. \quad \|p\| \leq \Delta_k$$





# Options

- How to pick  $B_k$  --
    - Newton or Quasi-Newton
  - How to pick trust region radius  $\Delta_k$  --
    - Shrink if fail to get a decrease
    - Increase if you get a good decrease
    - Otherwise keep the same
  - Trust region problem need not be solved exactly.
  - Many variations
- 



# Use ratio to determine trust region radius

$$p_k \in \arg \min_p m_k(p) := f(x_k) + \nabla f(x_k)' p + \frac{1}{2} p' B_k p$$
$$s.t. \quad \|p\| \leq \Delta_k$$

Look at ratio of actual versus predicted decrease.

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$$

If ratio is near one and  $\|p\| = \Delta_k$  then increase radius.

If ratio is near zero then decrease radius.






# Trust Region Methods

Pros:

- Pick direction and stepsize simultaneously
- Global convergence
- Superlinear convergence in many cases
- Some types very effective in practice.

Cons:

- Must solve one or more constrained trust region problems at each iteration
- 

# BFGS in NW

$$x_{x+1} = x_k - \alpha H_k \nabla f_k$$

$$\text{where } H_{k+1} = V_k' H_k V_k + \rho_k s_k s_k'$$

$$\text{with } \rho_k = \frac{1}{y_k' s_k} \quad V_k = (I - \rho_k s_k y_k')$$

$$s_k = x_{k+1} - x_k \quad y_k = \nabla f_{k+1} - \nabla f_k$$

$H_k$  has nice low rank structure and all we really need to do is multiply it times the gradient. So we can do these without explicitly storing it.

# H<sub>k</sub> grows at each iteration

$$\begin{aligned} H_k &= (V'_{k-1} \cdots V'_{k-m}) H_k^0 (V_{k-m} \cdots V_{k-1}) \\ &\quad + \rho_{k-m} (V'_{k-1} \cdots V'_{k-m+1}) s_{k-m} s'_{k-m} (V_{k-m+1} \cdots V_{k-1}) \\ &\quad + \rho_{k-m+1} (V'_{k-1} \cdots V'_{k-m+2}) s_{k-m+1} s'_{k-m+1} (V_{k-m+2} \cdots V_{k-1}) \\ &\quad \dots \\ &\quad + \rho_{k-1} s_{k-1} s'_{k-1} \end{aligned}$$


So can define a recursive procedure (see Algorithm 9 page 225)  
that only requires inner products (assuming H<sub>0</sub> diagonal)  
Uses only 4mn multiplications. Requires only storage of s<sub>k</sub>, y<sub>k</sub>



# More improvements


- $H_0$  can be changed at each iteration. A good choice in practice is:

$$H_k^0 = \gamma_k I \quad \gamma_k = \frac{s_{k-1} y_{k-1}}{y_{k-1} y_{k-1}}$$

- Limit memory: only store  $s_k, y_k$  for last  $m$  iterates and base approximation on that.
- 



# Limited Memory BFGS –pros and cons

- Usually best algorithm for large problems with non-sparse Hessians
  - May not be best if problem has special structure, e.g. sparsity, separable structure, nonlinear least squares.
  - Needs Wolfe stepsize.
  - Relatively cheap iterates
  - Robust
  - May converge slowly on highly ill conditioned problems.
- 



# Partially Separable Structure

## Examples

$$f(x) = f_1(x_1, x_3) + f_2(x_1, x_4) + f_3(x_4, x_5)$$


$$f(x) = \sum_{i=1}^m f_i(x)$$



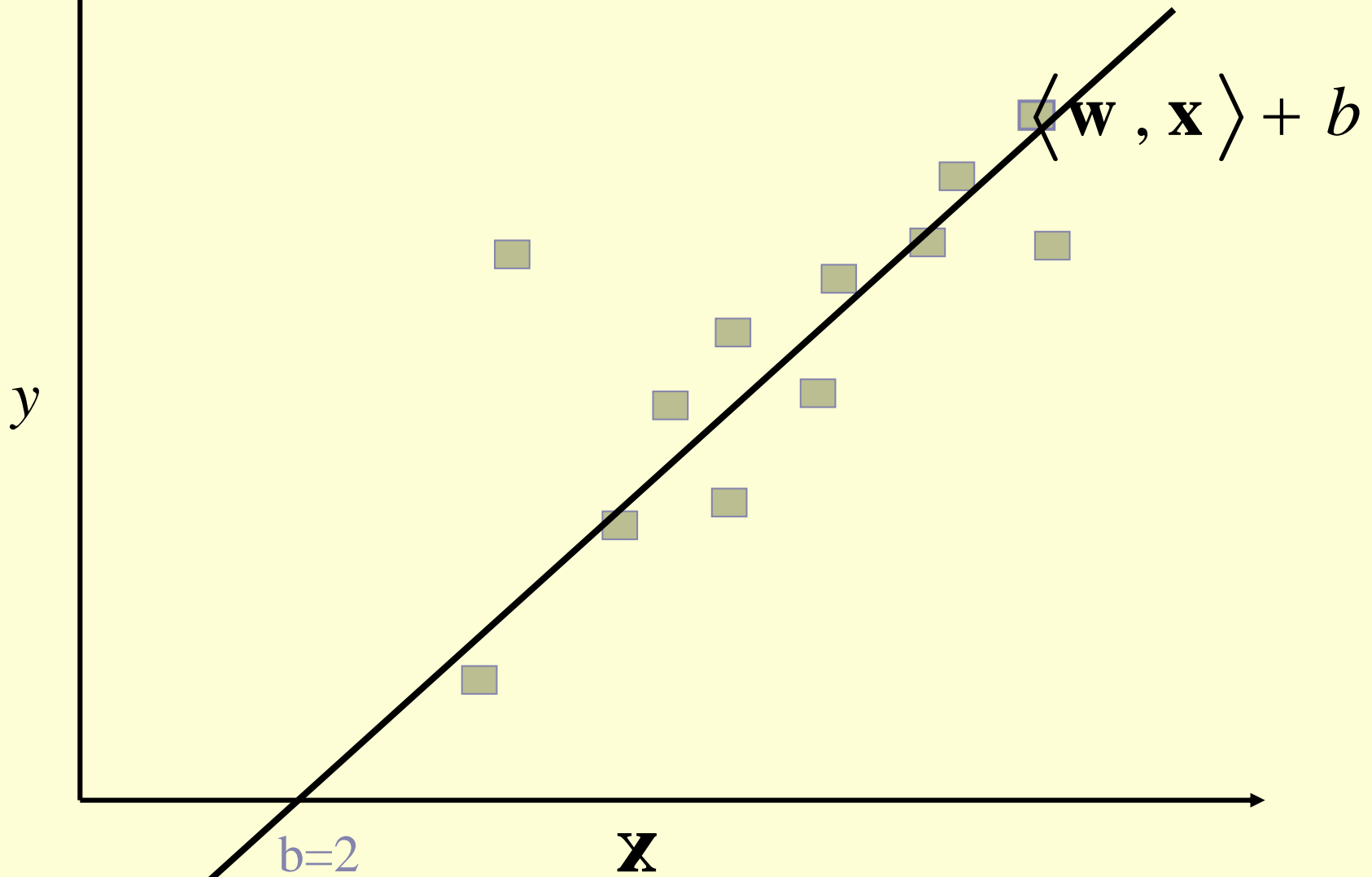


# Predict Drug Bioavailability

- Aqua solubility = Aquasol  $y \in R$
- 525 descriptors generated
  - Electronic TAE
  - Traditional  $\mathbf{x}_i \in R^{525}$
- 197 molecules with tested solubility

$$\ell = 197$$


# 1- d Regression with bias





# Linear Regression

Given training data:


$$S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_i, y_i), \dots, (\mathbf{x}_\ell, y_\ell))$$

points  $\mathbf{x}_i \in R^n$  and labels  $y_i \in R$

● Construct linear function:

$$g(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b = \mathbf{x}' \mathbf{w} + b = \sum_{i=1}^n w_i (x)_i + b$$

● Goal for future data  $(x, y)$  with  $y$  unknown

$$g(\mathbf{x}) \approx y$$





# Least Squares Approximation

• Want  $g(x) \approx y$

• Define error  $f(\mathbf{x}, y) = y - g(\mathbf{x}) = \xi$

• Minimize loss

$$L(g, s) = \sum_{i=1}^{\ell} (y_i - g(\mathbf{x}_i))^2$$




# Linear Least Squares Loss

$$\begin{aligned}L(\mathbf{w}, b) &= \sum_{i=1}^{\ell} (\mathbf{x}_i' \mathbf{w} + b - y_i)^2 \\ &= \|\mathbf{X}\mathbf{w} + \mathbf{e}b - \mathbf{y}\|^2 \quad 2\text{-norm} \\ &= (\mathbf{X}\mathbf{w} + \mathbf{e}b - \mathbf{y})'(\mathbf{X}\mathbf{w} + \mathbf{e}b - \mathbf{y})\end{aligned}$$


# Optimal Solution

• Want:  $\mathbf{y} \approx \mathbf{X}\mathbf{w} + b\mathbf{e}$   $\mathbf{e}$  is a vector of ones

• Mathematical Model:

$$\min_{\mathbf{w}} L(\mathbf{w}, b, S) = \|\mathbf{y} - (\mathbf{X}\mathbf{w} + b\mathbf{e})\|^2 + \lambda \|\mathbf{w}\|^2$$

• Optimality Conditions:

$$\frac{\partial L(\mathbf{w}, b, S)}{\partial \mathbf{w}} = 2\mathbf{X}'(\mathbf{y} - \mathbf{X}\mathbf{w} - b\mathbf{e}) + 2\lambda\mathbf{w} = 0$$

$$\frac{\partial L(\mathbf{w}, b, S)}{\partial b} = 2\mathbf{e}'(\mathbf{y} - \mathbf{X}\mathbf{w} - b\mathbf{e}) = 0$$



# Optimal Solution


Thus :

$$\mathbf{e}'\mathbf{e}b = \mathbf{e}'\mathbf{y} - \mathbf{e}'\mathbf{X}\mathbf{w}$$

$$\Rightarrow b = \frac{\mathbf{e}'\mathbf{y}}{\ell} - \frac{\mathbf{e}'\mathbf{X}\mathbf{w}}{\ell} = \text{mean}(\mathbf{y}) - \text{mean}(\mathbf{X})'\mathbf{w}$$

Assume data scaled such that  
 $\text{mean}(\mathbf{x}) = \mathbf{X}'\mathbf{e} = 0$

$$(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{w} = \mathbf{X}'\mathbf{y} - \mathbf{X}'\mathbf{e}b$$

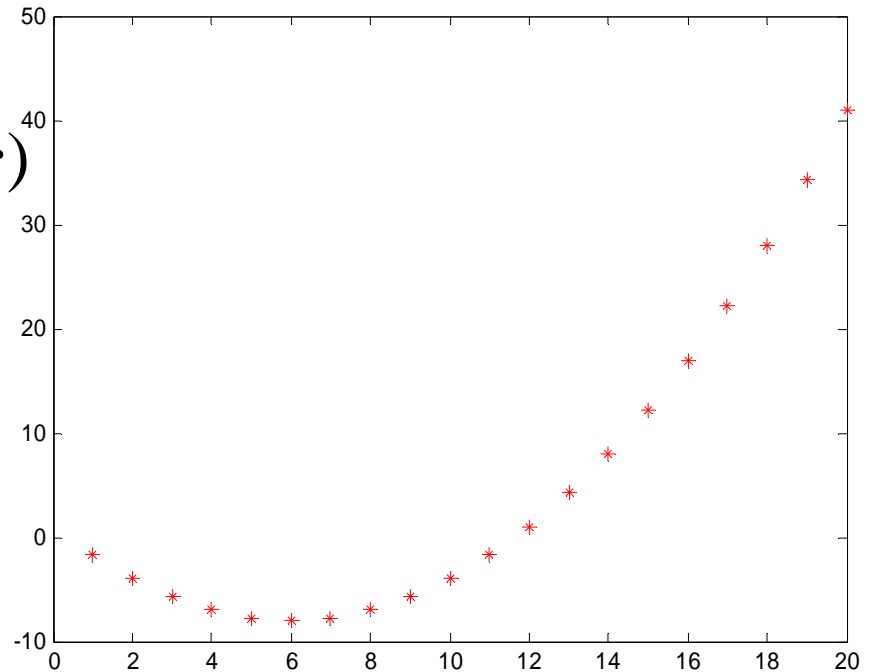
$$\mathbf{w} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}'\mathbf{y} \quad b = \text{mean}(\mathbf{y})$$


# Nonlinear Least squares

Partially separable problem

$$f(a,b,c) = \frac{1}{2} \sum_{i=1}^{\ell} (f_i(a,b,c))^2$$

$$f_i(a,b,c) = y_i - (ax_i^2 + bx_i + c)$$



# Nonlinear Least squares

Partially separable problem

$$\nabla f(a, b, c) = \sum_{i=1}^{\ell} \nabla f_i(a, b, c)' (f_i(a, b, c))$$

$$f(a, b, c) = \frac{1}{2} \sum_{i=1}^{\ell} (f_i(a, b, c))^2 = \sum_{i=1}^{\ell} - \left( y_i - (ax_i^2 + bx_i + c) \right) \begin{bmatrix} x_i^2 \\ x_i \\ 1 \end{bmatrix}$$

$$\nabla f_i(a, b, c) = - \begin{bmatrix} x_i^2 \\ x_i \\ 1 \end{bmatrix}$$

$$f_i(a, b, c) = y_i - (ax_i^2 + bx_i + c)$$

# Nonlinear Least Squares

- Problems of type:

$$\min_x f(x) = \frac{1}{2} \sum_i f_i(x)^2$$

*Gradient is*

$$\sum_i \nabla f_i(x) f_i(x)$$

*Hessian is*

$$\sum_i \nabla f_i(x) \nabla f_i(x)' + \sum_i f_i(x) \nabla^2 f_i(x)$$


*Approximate Hessian by*

$$\sum_i \nabla f_i(x) \nabla f_i(x)'$$

- Newton = Gauss-Newton
- Newton + trust region = Levenberg-Marquardt



# Matlab

- Check out Matlab optimization
  - Type bandem
  - Help fminunc
  - Has all the basics:
- 

# What if gradient not available?

- Can use finite difference methods

- Recall

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

- Approximate using small  $h$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

$$\frac{\partial f(x)}{\partial x_1} \approx \frac{f(x + h e_1) - f(x)}{h} \text{ where } e_1 = [1, 0, \dots, 0]'$$



# Problems

Introduces error

$$\frac{f(x+h) - f(x)}{h}$$

$$\frac{f(x+h) - f(x-h)}{2h}$$

Best value of  $h$  is very small. Close to machine precision.


Have to do for each dimension





# Automatic Differentiation

```
function makegradient(fcn, name)
%Creates a new matlab function (defined by gradient(fcn))
%and saves it with the specified name
% Example: makegradient('x^2+y^2','gf') creates a file gf.m:
% function functout = gf(v)
% x = v(1); % y = v(2);
% functout = [2*x, 2*y];
%
```






# Automatic Differentiation

```
function makehessian(fcn, name)
%Creates a new matlab function (defined by gradient(fcn))
%and saves it with the specified name
%
% Example: makehessian('x^7+x*y^3','hf') creates a file hf.m:
%
% function functout = hf(v)
%     x = v(1);
%     y = v(2);
%     functout = [[42*x^5, 3*y^2]; [3*y^2, 6*x*y]];
%
```





# Fminunc

- First try using finite difference approximation for the gradient
  - For example on L
  - $X0=[1,1]'$ ;
  - `Options = optimset('display','iter');`
  - `X=fminunc(@L,X0,Options)`
- 

# To use real gradient-Option 1

- Combine f,h,g into on matlab file

```
function [f,g,H] = matL(x)
```

```
f = L(x);
```

```
if nargin > 1
```

```
    g = gradL(x);
```

```
end;
```

```
if nargin > 2
```

```
    H = hessL(x);
```


```
end;
```

- Options = optimset('gradobj','on','Display','iter');

- X=fminunc(@matL,X0,Options)




# To use real gradient

- Options =  
optimset('gradobj','on','Display','iter');
  - X=fminunc(@L,@gradL,X0,Options)
- 



# To use Hessian

- Same as gradient but add
  - Options =  
optimset(Options,'hessian','on');
  - X=fminunc(@matL,X0,Options)
  
  - Or  
X=fminunc({@L,@gradL,@hessL},X0,Options)
- 



# Check your gradients!

● Try

Options =

```
optimset(Options,'DerivativeCheck','on')
```

```
X=fminunc(@L,X0,Options)
```

Try on the our family of functions.

What happens?

