# R bootcamp in Beg-Meil

## Basic optimization with R

State of the R

https://github.com/finistR2018

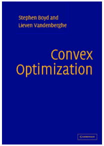
Beg-Meil, August 2018

# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`
8. Use external library by embedding C++ code by yourself

# References

See Chapter 9 in Convex Optimization (Boyd & Vandenberghe, 2004),
http://:web.stanford.edu/~boyd/cvxbook/




Online courses

All slides stolen (extracted/re-arranged) from Lieve Vandenberghe:

- Convex Optimization:
  http://www.seas.ucla.edu/∼vandenbe/ee236b/ee236b.html
- Optimization Methods for Large-Scale Systems
  http://www.seas.ucla.edu/∼vandenbe/ee236c/ee236c.html

# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# Unconstrained minimization

$$\text{minimize} \quad f(x)$$

- $f$ convex, twice continuously differentiable (hence $\mathbf{dom}\, f$ open)
- we assume optimal value $p^\star = \inf_x f(x)$ is attained (and finite)

**unconstrained minimization methods**

- produce sequence of points $x^{(k)} \in \mathbf{dom}\, f$, $k = 0, 1, \ldots$ with

$$f(x^{(k)}) \to p^\star$$

- can be interpreted as iterative methods for solving optimality condition

$$\nabla f(x^\star) = 0$$

# Descent methods

$$x^{(k+1)} = x^{(k)} + t^{(k)}\Delta x^{(k)} \quad \text{with } f(x^{(k+1)}) < f(x^{(k)})$$

- other notations: $x^+ = x + t\Delta x$, $x := x + t\Delta x$
- $\Delta x$ is the *step*, or *search direction*; $t$ is the *step size*, or *step length*
- from convexity, $f(x^+) < f(x)$ implies $\nabla f(x)^T \Delta x < 0$
  ($i.e.$, $\Delta x$ is a *descent direction*)

---

*General descent method.*

**given** a starting point $x \in \mathbf{dom}\, f$.
**repeat**
1. Determine a descent direction $\Delta x$.
2. *Line search.* Choose a step size $t > 0$.
3. *Update.* $x := x + t\Delta x$.

**until** stopping criterion is satisfied.

---

# Line search types

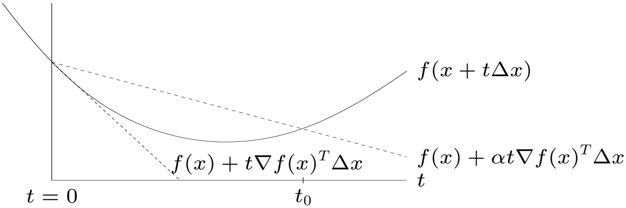
**exact line search:** $t = \operatorname{argmin}_{t>0} f(x + t\Delta x)$

**backtracking line search** (with parameters $\alpha \in (0, 1/2)$, $\beta \in (0, 1)$)

- starting at $t = 1$, repeat $t := \beta t$ until

$$f(x + t\Delta x) < f(x) + \alpha t \nabla f(x)^T \Delta x$$

- graphical interpretation: backtrack until $t \leq t_0$



# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# Gradient descent method

general descent method with $\Delta x = -\nabla f(x)$

---

**given** a starting point $x \in \mathbf{dom}\, f$.
**repeat**
1. $\Delta x := -\nabla f(x)$.
2. *Line search.* Choose step size $t$ via exact or backtracking line search.
3. *Update.* $x := x + t\Delta x$.

**until** stopping criterion is satisfied.

---

- stopping criterion usually of the form $\|\nabla f(x)\|_2 \leq \epsilon$
- convergence result: for strongly convex $f$,

$$f(x^{(k)}) - p^\star \leq c^k (f(x^{(0)}) - p^\star)$$

  $c \in (0, 1)$ depends on $m$, $x^{(0)}$, line search type
- very simple, but often very slow; rarely used in practice

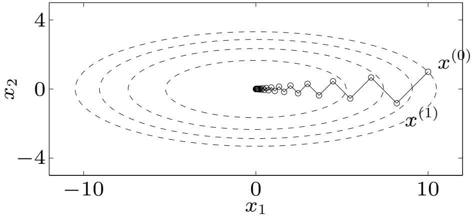
**quadratic problem in $\mathbf{R}^2$**

$$f(x) = (1/2)(x_1^2 + \gamma x_2^2) \qquad\qquad (\gamma > 0)$$

with exact line search, starting at $x^{(0)} = (\gamma, 1)$:

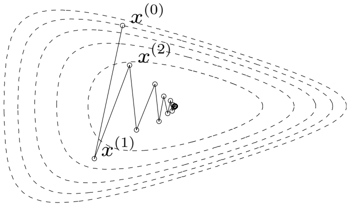
$$x_1^{(k)} = \gamma \left(\frac{\gamma - 1}{\gamma + 1}\right)^k, \qquad x_2^{(k)} = \left(-\frac{\gamma - 1}{\gamma + 1}\right)^k$$

- very slow if $\gamma \gg 1$ or $\gamma \ll 1$
- example for $\gamma = 10$:



## nonquadratic example

$$f(x_1, x_2) = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1}$$




backtracking line search

exact line search

# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# Newton method for unconstrained minimization

$$\text{minimize} \quad f(x)$$

$f$ convex, twice continously differentiable

**Newton method**

$$x^+ = x - t\nabla^2 f(x)^{-1} \nabla f(x)$$

- advantages: fast convergence, affine invariance
- disadvantages: requires second derivatives, solution of linear equation

can be too expensive for large scale applications

# Newton step

$$\Delta x_{\mathrm{nt}} = -\nabla^2 f(x)^{-1} \nabla f(x)$$

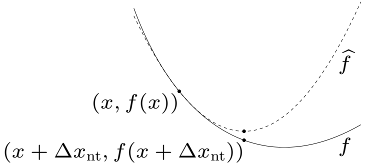
**interpretations**

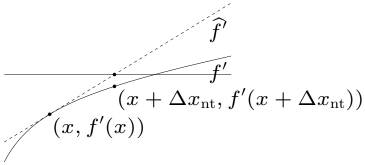
- $x + \Delta x_{\mathrm{nt}}$ minimizes second order approximation

$$\widehat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v$$

- $x + \Delta x_{\mathrm{nt}}$ solves linearized optimality condition

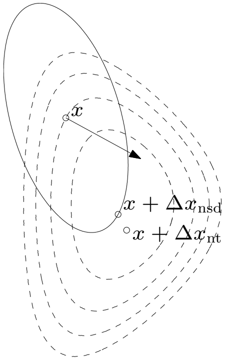
$$\nabla f(x+v) \approx \nabla \widehat{f}(x+v) = \nabla f(x) + \nabla^2 f(x) v = 0$$



- $\Delta x_{\rm nt}$ is steepest descent direction at $x$ in local Hessian norm

$$\|u\|_{\nabla^2 f(x)} = \left(u^T \nabla^2 f(x) u\right)^{1/2}$$




dashed lines are contour lines of $f$; ellipse is $\{x + v \mid v^T \nabla^2 f(x) v = 1\}$

arrow shows $-\nabla f(x)$

# Newton's method

---

**given** a starting point $x \in \mathbf{dom}\, f$, tolerance $\epsilon > 0$.
**repeat**

1. *Compute the Newton step and decrement.*
   $\Delta x_{\mathrm{nt}} := -\nabla^2 f(x)^{-1}\nabla f(x)$; $\quad \lambda^2 := \nabla f(x)^T \nabla^2 f(x)^{-1}\nabla f(x)$.
2. *Stopping criterion.* **quit** if $\lambda^2/2 \leq \epsilon$.
3. *Line search.* Choose step size $t$ by backtracking line search.
4. *Update.* $x := x + t\Delta x_{\mathrm{nt}}$.

---

affine invariant, $i.e.$, independent of linear changes of coordinates:

Newton iterates for $\tilde{f}(y) = f(Ty)$ with starting point $y^{(0)} = T^{-1}x^{(0)}$ are

$$y^{(k)} = T^{-1}x^{(k)}$$

# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# Variable metric methods

$$x^+ = x - tH^{-1}\nabla f(x)$$

$H \succ 0$ is approximation of the Hessian at $x$, chosen to:

- avoid calculation of second derivatives
- simplify computation of search direction

**'Variable metric' interpretation** (EE236B, lecture 10, page 11)

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

is steepest descent direction at $x$ for quadratic norm

$$\|z\|_H = \left(z^T H z\right)^{1/2}$$

# Quasi-Newton methods

**given** starting point $x^{(0)} \in \operatorname{dom} f$, $H_0 \succ 0$

1. compute quasi-Newton direction $\Delta x = -H_{k-1}^{-1} \nabla f(x^{(k-1)})$
2. determine step size $t$ (*e.g.*, by backtracking line search)
3. compute $x^{(k)} = x^{(k-1)} + t\Delta x$
4. compute $H_k$

- different methods use different rules for updating $H$ in step 4
- can also propagate $H_k^{-1}$ to simplify calculation of $\Delta x$

# Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

**BFGS update**

$$H_k = H_{k-1} + \frac{yy^T}{y^T s} - \frac{H_{k-1} s s^T H_{k-1}}{s^T H_{k-1} s}$$

where

$$s = x^{(k)} - x^{(k-1)}, \qquad y = \nabla f(x^{(k)}) - \nabla f(x^{(k-1)})$$

**Inverse update**

$$H_k^{-1} = \left(I - \frac{sy^T}{y^T s}\right) H_{k-1}^{-1} \left(I - \frac{ys^T}{y^T s}\right) + \frac{ss^T}{y^T s}$$

- note that $y^T s > 0$ for strictly convex $f$; see page 1-9
- cost of update or inverse update is $O(n^2)$ operations

# Limited memory quasi-Newton methods

main disadvantage of quasi-Newton method is need to store $H_k$ or $H_k^{-1}$

**Limited-memory BFGS** (L-BFGS): do not store $H_k^{-1}$ explicitly

- instead we store the $m$ (*e.g.*, $m = 30$) most recent values of

$$s_j = x^{(j)} - x^{(j-1)}, \qquad y_j = \nabla f(x^{(j)}) - \nabla f(x^{(j-1)})$$

- we evaluate $\Delta x = H_k^{-1} \nabla f(x^{(k)})$ recursively, using

$$H_j^{-1} = \left(I - \frac{s_j y_j^T}{y_j^T s_j}\right) H_{j-1}^{-1} \left(I - \frac{y_j s_j^T}{y_j^T s_j}\right) + \frac{s_j s_j^T}{y_j^T s_j}$$

  for $j = k, k-1, \ldots, k-m+1$, assuming, for example, $H_{k-m}^{-1} = I$

- cost per iteration is $O(nm)$; storage is $O(nm)$

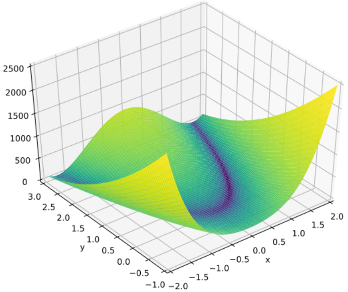
# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# Rosenbrock Banana function

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

$$\nabla f(x) = \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{pmatrix}$$



# Rosenbrock Banana function in R

## objective

```
objective <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}
```

## gradient

```
gradient <- function(x) {
  return( c( -400 * x[1] * (x[2] - x[1] * x[1]) - 2 * (1 - x[1]), 200 * (x[2] - x[1] * x[1]) )
}
```

# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# optim usage

### Definition

```
optim(par, fn, gr = NULL, ...,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN",
                 "Brent"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

- Nelder-Mead: approximation of the gradient
- BFGS: quasi-Newton
- CG: conjuguate gradient
- L-BFGS-B: BFGS with limited memory, box constrained
- SANN: simulated annealing

# Call to optim - BFGS

```r
x0 <- c(-1.2, 1)
res_bfgs <- optim(x0, objective, gradient, method = "BFGS", control= list(trace = 2))
```

```
## initial  value 24.200000
## iter  10 value 1.367383
## iter  20 value 0.134560
## iter  30 value 0.001978
## iter  40 value 0.000000
## final  value 0.000000
## converged
```

# Call to optim - CG

```r
x0 <- c(-1.2, 1)
res_cg <- optim(x0, objective, gradient, method = "CG", control= list(trace = 2))
```

```
##   Conjugate gradients function minimizer
## Method: Fletcher Reeves
## tolerance used in gradient test=3.63798e-12
## 0 1 24.200000
## parameters   -1.20000    1.00000
## **** i< 1 7 4.132161
## parameters   -1.02752    1.07040
## * i> 2 10 4.126910
## parameters   -1.02855    1.06882
## **** i> 3 16 4.121409
## parameters   -1.02924    1.06533
##  i> 4 18 4.106523
## parameters   -1.02586    1.05731
## **** i> 5 24 4.100955
## parameters   -1.02261    1.05573
##  i> 6 26 4.086136
## parameters   -1.01839    1.04818
## **** i> 7 32 4.080524
## parameters   -1.01914    1.04464
##  i> 8 34 4.065787
## parameters   -1.01579    1.03670
## **** i> 9 40 4.060127
## parameters   -1.01250    1.03514
##  i> 10 42 4.045415
## parameters   -1.00824    1.02768
## **** i> 11 48 4.039717
## parameters   -1.00900    1.02412
```

# Outline

1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# nloptr usage

Definition

```
nloptr(x0, eval_f, eval_grad_f, ..., opts = list())
```

- Many gradient free methods
- Most existing gradient-based methods
- global optimizer

# Call to nloptr - BFGS

```
library(nloptr)
opts <- list("algorithm"="NLOPT_LD_LBFGS", "xtol_rel"=1.0e-8)
res <- nloptr(x0=x0, eval_f=objective, eval_grad_f=gradient, opts=opts)
print(res)
```

```
##
## Call:
##
## nloptr(x0 = x0, eval_f = objective, eval_grad_f = gradient, opts = opts)
##
##
## Minimization using NLopt version 2.4.2
##
## NLopt solver status: 1 ( NLOPT_SUCCESS: Generic success return value. )
##
## Number of Iterations....: 56
## Termination conditions:  xtol_rel: 1e-08
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  7.35727226897802e-23
## Optimal value of controls: 1 1
```

# Outline


1. Minimization Problem
2. Gradient methods
3. Newton methods
4. Quasi-Newton methods
5. An example
6. Use `stats::optim`
7. Use external library via `nloptr`

# References

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press. Retrieved from http://web.stanford.edu/~boyd/cvxbook/