COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

AMATH 460: Mathematical Methods for Quantitative Finance

8. Numerical Methods

Kjell Konis
Acting Assistant Professor, Applied Mathematics
University of Washington

Outline

Outline

Implied Volatility

- The Black-Scholes formula for the price of a European call option

$$C = Se^{-q(T-t)}\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2)$$

where

$$d_1 = \frac{\log\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}} \qquad d_2 = d_1 - \sigma\sqrt{T-t}$$

- The maturity T and strike price K are given
- When option is traded, spot price S and option price C are known
- Assume risk-free rate r is constant and q known
- Only value that is not known is the volatility σ

Implied Volatility

- The implied volatility problem is to find σ so that the Black-Scholes price and the market price are equal
- If S, K, q, r, T, and t are known, consider

$$f(\sigma) = Se^{-q(T-t)}\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2) - C$$

- Finding the implied volatility boils down to solving the nonlinear problem

$$f(\sigma) = 0$$

- Problem can be solved numerically
- For example, plot $f(\sigma)$ and see where it is equal to zero

Implied Volatility

- R function to compute Black-Scholes call price

```
bsc <- function(S, T, t, K, r, s, q) {
  d1 <- (log(S/K)+(r-q+0.5*s^2)*(T-t))/(s*sqrt(T-t))
  d2 <- d1-s*sqrt(T-t)
  S*exp(-q*(T-t))*pnorm(d1)-K*exp(-r*(T-t))*pnorm(d2)
}
```

- Since R treats all variables as vectors

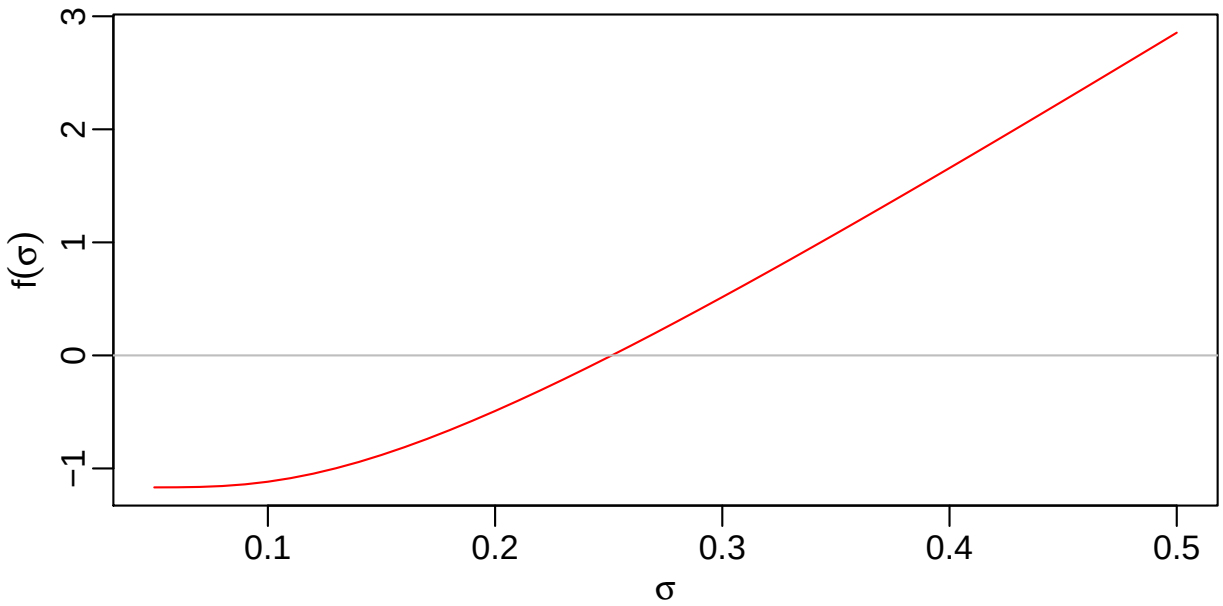
```
> bsc(50, 0.5, 0.0, 45, 0.06, 0.2, 0.02)
[1] 6.508365

> bsc(50, 0.5, 0.0, 45, 0.06, c(0.15, 0.2, 0.25), 0.02)
[1] 6.119266 6.508365 6.986157
```

Implied Volatility

- Suppose the option sold for \$7, find σ
- Plot $f(\sigma)$ over a range of values and see where it crosses the x axis

```
> sigmas <- seq(0.05, 0.5, by = 0.01)
> fsig <- bsc(50, 0.5, 0.0, 45, 0.06, sigmas, 0.02) - 7
> plot(sigmas, fsig, type = "l")
```



- The implied volatility is $\sigma_{\text{implied}} = 0.25$

Implied Volatility

- To compute σ_{implied}, had to evaluate Black-Scholes formula 46 times
- Computed answer still not very precise

```
> bsc(50, 0.5, 0.0, 45, 0.06, 0.25, 0.02) - 7
[1] -0.013843
```

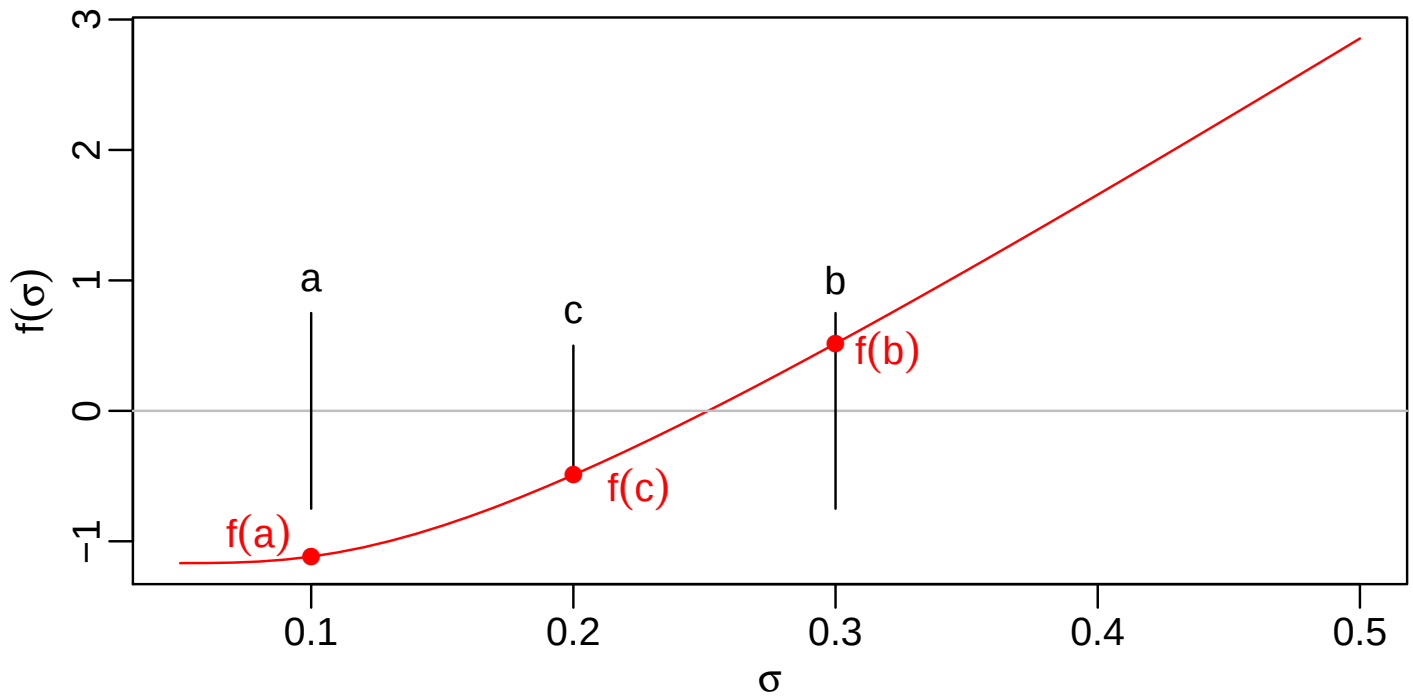
- Goal: compute σ_{implied} to within a pre-specified tolerance with minimum number of function evaluations
- Methods are called nonlinear solvers

Outline

Bisection Method

- Let f be a continuous function defined on the interval $[a, b]$
- If $f(a)$ and $f(b)$ have different signs, intermediate value theorem says there is $x \in (a, b)$ such that $f(x) = 0$
- Bisection method
  1. Compute $f(c)$ where $c = \frac{1}{2}(a + b)$ is the midpoint of $[a, b]$
  2. If $\text{sign}(f(c)) = \text{sign}(f(a))$, let $a := c$, otherwise let $b := c$
  3. Goto 1
- Repeat steps 1–3 until $|b - a|$ is smaller than a pre-specified tolerance

Example: Bisection Method



- Let $a = 0.1$, $b = 0.3$; $f(0.1) < 0$ and $f(0.3) > 0$
- Let $c = \frac{1}{2}(a + b) = 0.2$; $f(0.2) < 0$
- Since $\text{sign}\big(f(0.2)\big) = \text{sign}\big(f(0.1)\big)$, let $a := 0.2$
- Each step preserves $\text{sign}\big(f(a)\big) = -\text{sign}\big(f(b)\big)$, cuts interval in half

Bisection Method: R Implementation

- Implementation of the bisection method for a function of one variable
- No error checking

```
bisection <- function(f, a, b, tol = 0.001) {
  while(b-a > tol) {
    c <- (a+b)/2
    if(sign(f(c)) == sign(f(a)))
      a <- c
    else
      b <- c
  }
  (a+b)/2
}
```

Example: Bisection Method

- Write $f(\sigma)$ as a function of one variable

```
fsig <- function(sigma)
  bsc(50, 0.5, 0.0, 45, 0.06, sigma, 0.02) - 7
```

- Use `bisection` to solve $f(\sigma) = 0$

```
> bisection(fsig, 0.1, 0.3)
[1] 0.2511719
```

- Check computed solution

```
> bsc(50, 0.5, 0.0, 45, 0.06, 0.2511719, 0.02)
[1] 6.998077
```

Outline

Newton's Method

- Commonly used method for solving nonlinear equations
- Again, want to find x^* so that $f(x^*) = 0$
- Assumptions
  - $f(x)$ is differentiable
  - Starting point x_0
- Idea: approximate $f(x)$ with a first order Taylor polynomial around x_k

$$f(x) \approx f(x_k) + (x - x_k)f'(x_k)$$

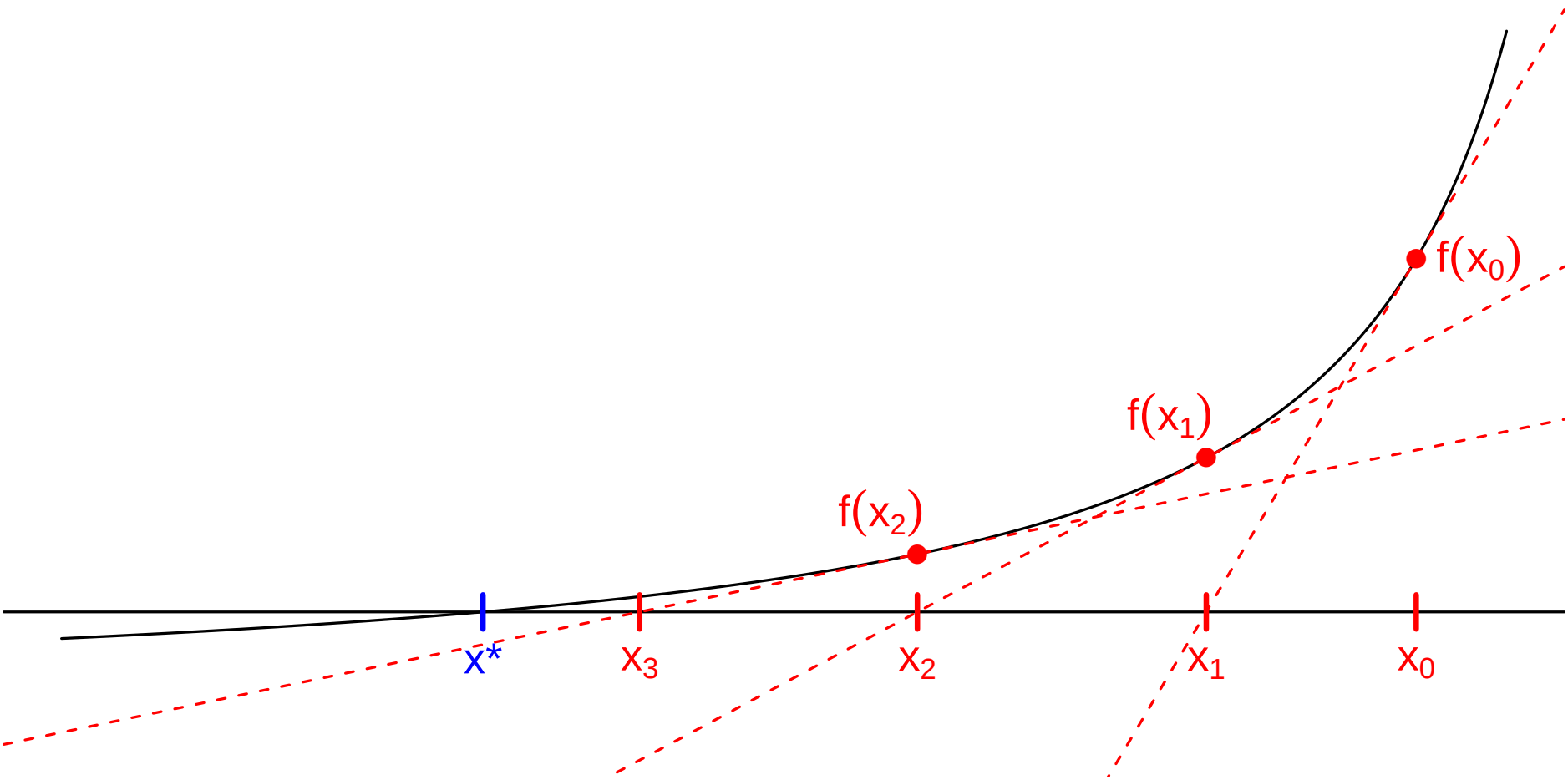
- Want to choose x_{k+1} so that $f(x_{k+1}) = 0$

$$f(x_k) + (x_{k+1} - x_k)f'(x_k) = f(x_{k+1}) \approx 0$$

- Leads to the recursion

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Illustration: Newton's Method



- Newton's method produces a sequence $\{x_k\}$ that "*converges*" to x^*

Analysis: Newton's Method

- Consider an order 1 Taylor polynomial around x_k evaluated at x^*

$$0 = f(x^*) = f(x_k) + (x^* - x_k) f'(x_k) + \frac{(x^* - x_k)^2}{2} f''(\xi_k) \qquad \xi_k \in [x^*, x_k]$$

$$\begin{aligned} \frac{f(x_k)}{f'(x_k)} + (x^* - x_k) &= -\frac{f''(\xi_k)}{2f'(x_k)}(x^* - x_k)^2 \\ x^* - x_{k+1} &= -\frac{f''(\xi_k)}{2f'(x_k)}(x^* - x_k)^2 \\ \epsilon_{k+1} &= -\frac{f''(\xi_k)}{2f'(x_k)}\epsilon_k^2 \end{aligned}$$

- If f' and f'' are bounded on the interval where the solver is active

$$|\epsilon_{k+1}| \leq M |\epsilon_k|^2 \qquad M = \max \left| \frac{f''(\xi_k)}{2f'(x_k)} \right|$$

- Newton's method converges quadratically

Caveats

- Quadratic convergence not guaranteed
  - Need good starting point + well-behaved function
- It gets worse: convergence not guaranteed

- In particular, algorithm may cycle
- For example: $sin(x) = 0$ between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$
- There is an x_k such that

$$x_{k+1} = x_k - \frac{sin(x_k)}{cos(x_k)} = -x_k$$

- What happens in the next iteration?

$$\begin{aligned} x_{k+2} &= x_{k+1} - \frac{sin(x_{k+1})}{cos(x_{k+1})} = -x_k - \frac{sin(-x_k)}{cos(-x_k)} = -x_k + \frac{sin(x_k)}{cos(x_k)} \\ &= x_k \end{aligned}$$

Outline

Newton's Method for n Dimensional Nonlinear Problems

- Let $F : \mathbb{R}^n \to \mathbb{R}^n$ have continuous partial derivatives
- Want to solve n-dimensional nonlinear problem

$$F(x) = 0$$

- Recall that the gradient of F is the $n \times n$ matrix

$$DF(x) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x) & \frac{\partial F_1}{\partial x_2}(x) & \cdots & \frac{\partial F_1}{\partial x_n}(x) \\ \frac{\partial F_2}{\partial x_1}(x) & \frac{\partial F_2}{\partial x_2}(x) & \cdots & \frac{\partial F_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}(x) & \frac{\partial F_n}{\partial x_2}(x) & \cdots & \frac{\partial F_n}{\partial x_n}(x) \end{bmatrix}$$

- Taylor polynomial for $F(x)$ around the point x_k

$$F(x) \approx F(x_k) + DF(x_k)(x - x_k)$$

Newton's Method for n Dimensional Nonlinear Problems

- To get Newton's method, approximate $F(x_{k+1})$ by 0

$$0 \approx F(x_k) + DF(x_k)(x_{k+1} - x_k)$$

- Solving for x_{k+1} gives the Newton's method recursion

$$x_{k+1} = x_k - \left[DF(x_k)\right]^{-1} F(x_k)$$

- Need stopping condition, e.g., for $\tau > 0$

$$\left\| \left[DF(x_k)\right]^{-1} F(x_k) \right\| < \tau$$

- Similar to univariate version, have quadratic convergence for x_k sufficiently close to x^*

Algorithm

- Need starting point x_0
- If $D\,F(x_k)$ is nonsingular, can compute x_{k+1} using the recursion

$$x_{k+1} = x_k - \left[D\,F(x_k)\right]^{-1} F(x_k)$$

- To avoid computing inverse of $\left[D\,F(x_k)\right]$, let

$$\begin{aligned} u &= \left[D\,F(x_k)\right]^{-1} F(x_k) \\ \left[D\,F(x_k)\right] u &= \left[D\,F(x_k)\right] \left[D\,F(x_k)\right]^{-1} F(x_k) = F(x_k) \end{aligned}$$

- Algoroithm: (starting from x_0, tolerance $\tau > 0$)
  1. Compute $F(x_k)$ and $D\,F(x_k)$
  2. Solve the linear system $\left[D\,F(x_k)\right] u = F(x_k)$
  3. Update $x_{k+1} = x_k - u$
  4. If $\|u\| < \tau$ return x_{k+1}, otherwise goto 1

Example

- Let $g(x, y) = 1 - (x-1)^4 - (y-1)^4$
- Local maximum at $(1, 1) \implies$ critical point at $(1, 1)$
- Gradient of $g(x, y)$

$$F(x, y) = \left[D\, g(x, y)\right]^{\mathsf{T}} = \begin{bmatrix} 4(x-1)^3 & 4(y-1)^3 \end{bmatrix}^{\mathsf{T}}$$

- Gradient of $F(x, y)$

$$D\, F(x, y) = \begin{bmatrix} 12(x-1)^2 & 0 \\ 0 & 12(y-1)^2 \end{bmatrix}$$

- Use R to compute solution

Example: R Implementation

- First, need functions for $F(x, y)$ and its gradient

```
F <- function(x, y)
  c(4*(x - 1)^3, 4*(y - 1)^3)

DF <- function(x, y)
  diag(c(12*(x - 1)^2, 12*(y - 1)^2))
```

- Need starting point: `x <- c(0, 0)`
- Do 25 Newton iterations

```
for(i in 1:25)
  x <- x - solve(DF(x[1], x[2]), F(x[1], x[2]))
```

- Result

```
[1] 0.9999604 0.9999604
```

Outline

Lagrange's Method

- Lagrange's method for solving constrained optimization problems
- Need to find the critical points of the Lagrangian
  - Easy in certain cases: minimum variance portfolio (solve linear system)
  - Difficult when system is nonlinear: maximum expected return portfolio

- Revisit the example from the Lagrange's method slides

$$\begin{aligned} \text{max/min:} \quad & 4x_2 - 2x_3 \\ \text{subject to:} \quad & 2x_1 - x_2 - x_3 = 0 \\ & x_1^2 + x_2^2 - 13 = 0 \end{aligned}$$

Newton's Method

- Lagrangian

$$G(x,\lambda) = 4x_2 - 2x_3 + \lambda_1(2x_1 - x_2 - x_3) + \lambda_2(x_1^2 + x_2^2 - 13)$$

- Set $F(x,\lambda) = \left[D\,G(x,\lambda)\right]^{\mathsf{T}} = 0$ and solve for x and λ

$$\begin{aligned} -4 + 2\lambda_2 x_1 &\overset{\text{set}}{=} 0 \\ 6 + 2\lambda_2 x_2 &\overset{\text{set}}{=} 0 \\ -2 - \lambda_1 &\overset{\text{set}}{=} 0 \\ 2x_1 - x_2 - x_3 &\overset{\text{set}}{=} 0 \\ x_1^2 + x_2^2 - 13 &\overset{\text{set}}{=} 0 \end{aligned}$$

- Nonlinear equation to solve

$$F(x,\lambda_2) = \begin{bmatrix} -4 + 2\lambda_2 x_1 \\ 6 + 2\lambda_2 x_2 \\ 2x_1 - x_2 - x_3 \\ x_1^2 + x_2^2 - 13 \end{bmatrix} \qquad DF(x,\lambda_2) = \begin{bmatrix} 2\lambda_2 & 0 & 0 & 2x_1 \\ 0 & 2\lambda_2 & 0 & 2x_2 \\ 2 & -1 & -1 & 0 \\ 2x_1 & 2x_2 & 0 & 0 \end{bmatrix}$$

R Implementation of Newton's Method

- R function for evaluating the function F

```
F <- function(x)
  c(-4+2*x[4]*x[1],
     6+2*x[4]*x[2],
     2*x[1]-x[2]-x[3],
     x[1]^2+x[2]^2-13)
```

- R function for evaluating the gradient of F

```
DF <- function(x)
  matrix(c(2*x[4],      0,    0, 2*x[1],
                0, 2*x[4],    0, 2*x[2],
                2,      -1,  -1,      0,
           2*x[1], 2*x[2],    0,      0),
           4, 4, byrow = TRUE)
```

R Implementation of Newton's Method

- Starting point

```
x <- rep(1, 4)
```

- 15 Newton iterations

```
for(i in 1:15)
  x <- x - solve(DF(x), F(x))
```

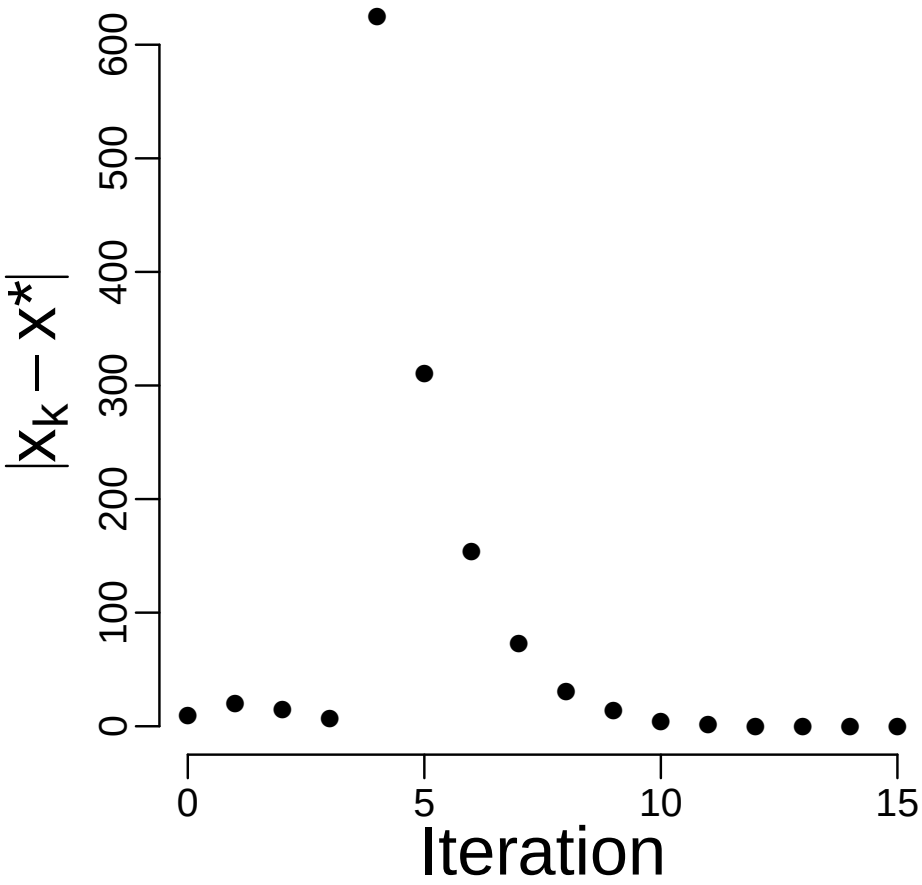
- Code for Newton iterations very simple!

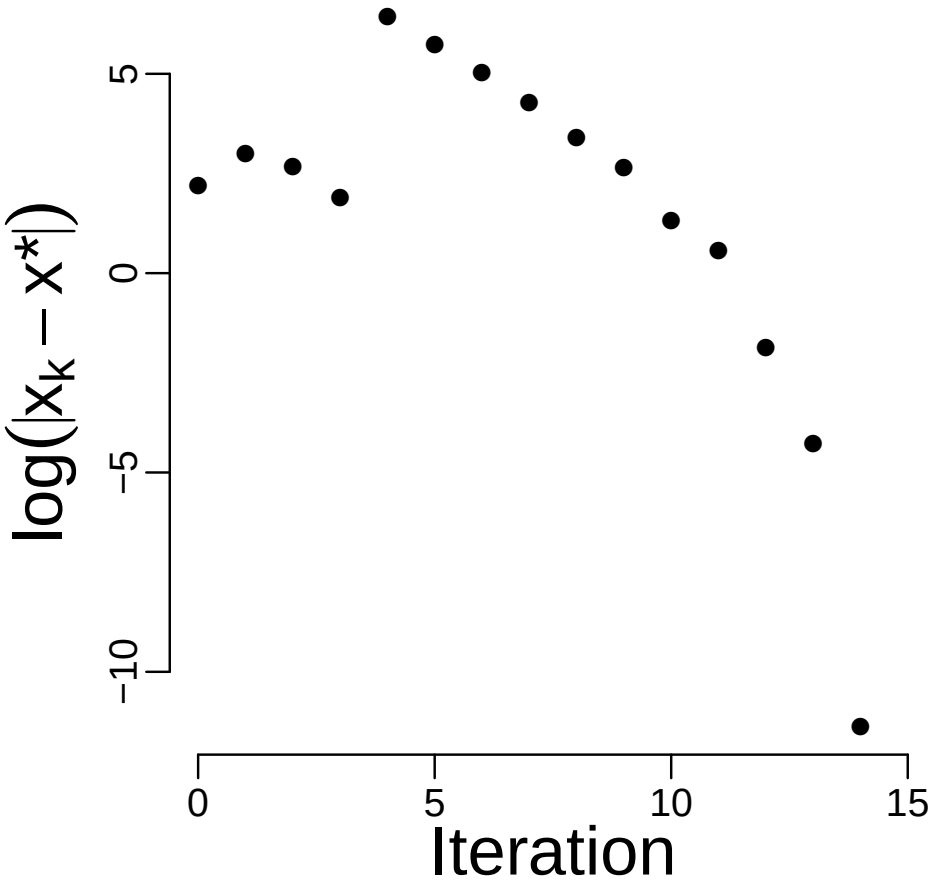
Trace of Newton's Method

- Recall solutions: $(-2, 3, -7, -2, -1)$ and $(2, -3, 7, -2, 1)$

	x1	x2	x3	lambda2
0	1.000000	1.000000	1.000000	1.00000000
1	6.250000	1.250000	11.250000	-3.25000000
2	3.923817	1.830917	6.016716	-0.88961538
3	1.628100	5.180972	-1.924771	-0.01078146
4	-247.240485	81.795257	-576.276226	-0.41960973
5	-121.982204	45.928353	-289.892761	-0.22067419
6	-57.679989	31.899766	-147.259743	-0.13272296
7	-22.882945	26.924965	-72.690855	-0.11474286
8	-8.779338	15.966384	-33.525061	-0.15812161
9	-6.263144	7.360141	-19.886430	-0.27312590
10	-2.393401	5.191356	-9.978158	-0.48808187
11	-2.715121	3.147713	-8.577955	-0.77002329
12	-1.941247	3.135378	-7.017871	-0.95609045
13	-2.006288	2.999580	-7.012156	-0.99823214
14	-1.999995	3.000010	-7.000000	-0.99999694
15	-2.000000	3.000000	-7.000000	-1.00000000

Convergence Rate





Observations

- From a given starting point, Newton's method converges (if it converges) to a single value x^*
- Starting at $(1, 1, 1, 1)$, computed solution $(-2, 3, -7, -1)$
- For this particular problem, know there are 2 critical points
- Try another starting point

```
x <- -rep(1, 4)
for(i in 1:15)
  x <- x - solve(DF(x), F(x))
```

  Solution: $(2, -3, 7, 1)$
- In general, will not know the number of critical points
- Need additional information about the problem
- Multiple starting points

Outline

Example: Lagrange's Method

- Homework 7 asked why is it difficult to find the critical points of the Lagrangian for the following optimization problem

$$\begin{aligned} \text{minimize:} \quad & 3x_1 - 4x_2 + x_3 - 2x_4 \\ \text{subject to:} \quad & -x_2^2 + x_3^2 + x_4^2 = 1 \\ & 3x_1^2 + x_3^2 + 2x_4^2 = 6 \end{aligned}$$

- Lagrangian

$$\begin{aligned} F(x, \lambda) \quad = \quad & 3x_1 - 4x_2 + x_3 - 2x_4 \\ & + \lambda_1(-x_2^2 + x_3^2 + x_4^2 - 1) \\ & + \lambda_2(3x_1^2 + x_3^2 + 2x_4^2 - 6) \end{aligned}$$

Example: Lagrange's Method

- Gradient of Lagrangian

$$G(x,\lambda) = \left[D\,F(x,\lambda)\right]^{\mathsf{T}} = \begin{bmatrix} 3 + 6\lambda_2 x_1 \\ -4 - 2\lambda_1 x_2 \\ 1 + 2\lambda_1 x_3 + 2\lambda_2 x_3 \\ -2 + 2\lambda_1 x_4 + 4\lambda_2 x_4 \\ -x_2^2 + x_3^2 + x_4^2 - 1 \\ 3x_1^2 + x_3^2 + 2x_4^2 - 6 \end{bmatrix}$$

- Gradient of G

$$D\,G(x,\lambda) = \begin{bmatrix} 6\lambda_2 & 0 & 0 & 0 & 0 & 6x_1 \\ 0 & -2\lambda_1 & 0 & 0 & -2x_2 & 0 \\ 0 & 0 & 2\lambda_1 + 2\lambda_2 & 0 & 2x_3 & 2x_3 \\ 0 & 0 & 0 & 2\lambda_1 + 4\lambda_2 & 2x_4 & 2x_4 \\ 0 & -2x_2 & 2x_3 & 2x_4 & 0 & 0 \\ 6x_1 & 0 & 2x_3 & 4x_4 & 0 & 0 \end{bmatrix}$$

R Implementation

- Function to compute $G(x, \lambda)$

```
G <- function(x)
  c(3 + 6*x[6]*x[1], -4 - 2*x[5]*x[2],
    1 + 2*x[5]*x[3] + 2*x[6]*x[3],
    -2 + 2*x[5]*x[4] + 4*x[6]*x[4],
    -x[2]^2 + x[3]^2 + x[4]^2 - 1,
    3*x[1]^2 + x[3]^2 + 2*x[4]^2 - 6)
```

- Function to compute $D\,G(x, \lambda)$

```
DG <- function(x) {
  grad <- matrix(0, 6, 6)
  grad[1,] <- c(6*x[6], 0, 0, 0, 0, 6*x[1])
  grad[2,] <- c(0, -2*x[5], 0, 0, -2*x[2], 0)
  grad[3,] <- c(0, 0, 2*x[5] + 2*x[6], 0, 2*x[3], 2*x[3])
  grad[4,] <- c(0, 0, 0, 2*x[5] + 4*x[6], 2*x[4], 2*x[4])
  grad[5,] <- c(0, -2*x[2], 2*x[3], 2*x[4], 0, 0)
  grad[6,] <- c(6*x[1], 0, 2*x[3], 4*x[4], 0, 0)
  grad
}
```

Newton's Method

- Starting point

```
x <- c(1, -1, 1, -1, 1, -1)
```

- Newton iterations

```
for(i in 1:25)
  x <- x - solve(DG(x), G(x))
```

- Numeric solution

```
> x
[1]  0.4067205 -2.0091498  2.1376693 -0.6834125
[5]  0.9954460 -1.2293456
```

- That is

$$x_c = \begin{bmatrix} 0.4067205 \\ -2.0091498 \\ 2.1376693 \\ -0.6834125 \end{bmatrix} \qquad \lambda_c = \begin{bmatrix} 0.9954460 \\ -1.2293456 \end{bmatrix}$$

Analysis

- Does the point (x_c, λ_c) correspond to a minimum or a maximum?
- Value of objective at (x_c, λ_c)

```
f <- function(x) 3*x[1] - 4*x[2] + x[3] - 2*x[4]
```

```
> f(x)
[1] 12.76125
```

- Rewrite Lagrangian with fixed multipliers (for feasible x)

$$\begin{aligned} f(x) = F(x, \lambda_c) &= 3x_1 - 4x_2 + x_3 - 2x_4 \\ &\quad + 0.9954460\,(-x_2^2 + x_3^2 + x_4^2 - 1) \\ &\quad - 1.2293456\,(3x_1^2 + x_3^2 + 2x_4^2 - 6) \end{aligned}$$

- Constrained min/max $f(x) \sim$ min/max $F(x, \lambda_c)$

Analysis

- Already know x_c is a critical point of $F(x, \lambda_c)$
  - $x_c \sim$ minimum if $D^2F(x_c, \lambda_c)$ positive definite
  - $x_c \sim$ maximum if $D^2F(x_c, \lambda_c)$ negative definite
- Already have $D^2F(x_c, \lambda_c)$, upper-left 4×4 block of $D\,G(x_c, \lambda_c)$

```
> round(DG(x), digits = 3)
      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,] -7.376  0.000  0.000  0.000  0.000  2.440
[2,]  0.000 -1.991  0.000  0.000  4.018  0.000
[3,]  0.000  0.000 -0.468  0.000  4.275  4.275
[4,]  0.000  0.000  0.000 -2.926 -1.367 -1.367
[5,]  0.000  4.018  4.275 -1.367  0.000  0.000
[6,]  2.440  0.000  4.275 -2.734  0.000  0.000
```

- Follows that x_c corresponds to a maximum

Outline

Maximum Expected Returns Optimization

- Recall:
  - Portfolio weights: $w = (w_1, \ldots, w_n)$
  - Asset expected returns: $\mu = (\mu_1, \ldots, \mu_n)$
  - Asset returns covariance matrix: Σ (symmetric, positive definite)
- Want to maximize expected return subject to constraint on risk

$$\begin{aligned} \text{maximize:} \quad & \mu^T w \\ \text{subject to:} \quad & e^T w = 1 \\ & w^T \Sigma w = \sigma_P^2 \end{aligned}$$

- Linear objective, linear and quadratic constraints
- Lagrangian

$$F(w, \lambda) = \mu^T w + \lambda_1 (e^T w - 1) + \lambda_2 (w^T \Sigma w - \sigma_P^2)$$

Maximum Expected Returns Optimization

- Gradient of the Lagrangian

$$D\,F(w,\lambda) = \begin{bmatrix} \mu^{\mathsf{T}} + \lambda_1 e^{\mathsf{T}} + 2\lambda_2 w^{\mathsf{T}}\Sigma & \qquad e^{\mathsf{T}}w - 1 & \qquad w^{\mathsf{T}}\Sigma w - \sigma_P^2 \end{bmatrix}$$

- Solve to find critical point

$$G(w,\lambda) = \left[D\,F(w,\lambda)\right]^{\mathsf{T}} = \begin{bmatrix} \mu + \lambda_1 e + 2\lambda_2 \Sigma w \\ e^{\mathsf{T}}w - 1 \\ w^{\mathsf{T}}\Sigma w - \sigma_P^2 \end{bmatrix} = 0$$

- Gradient of G

$$D\,G(w,\lambda) = \begin{bmatrix} 2\lambda_2\Sigma & e & 2\Sigma w \\ e^{\mathsf{T}} & 0 & 0 \\ 2(\Sigma w)^{\mathsf{T}} & 0 & 0 \end{bmatrix}$$

Example

- Vector of expected returns

$$\mu = (0.08,\ 0.10,\ 0.13,\ 0.15,\ 0.20)$$

- Asset returns covariance matrix

$$\Sigma = \begin{bmatrix} 0.019600 & -0.007560 & 0.012880 & 0.008750 & -0.009800 \\ -0.007560 & 0.032400 & -0.004140 & -0.009000 & 0.009450 \\ 0.012880 & -0.004140 & 0.052900 & 0.020125 & 0.020125 \\ 0.008750 & -0.009000 & 0.020125 & 0.062500 & -0.013125 \\ -0.009800 & 0.009450 & 0.020125 & -0.013125 & 0.122500 \end{bmatrix}$$

- Target risk: $\sigma_P^2 = 0.25^2 = 0.0625$

R Implementation

- Definition of $G(w, \lambda)$

$$G(w, \lambda) = \begin{bmatrix} \mu + \lambda_1 e + 2\lambda_2 \Sigma w \\ e^\top w - 1 \\ w^\top \Sigma w - \sigma_P^2 \end{bmatrix}$$

- Function to compute $G(w, \lambda)$

```
G <- function(x, mu, Sigma, sigmaP2)
{
  n <- length(mu)
  c(mu + rep(x[n+1], n) + 2*x[n+2]*(Sigma %*% x[1:n]),
    sum(x[1:n]) - 1,
    t(x[1:5]) %*% Sigma %*% x[1:5] - sigmaP2)
}
```

R Implementation

- Gradient of G

$$D\,G(w,\lambda) = \begin{bmatrix} 2\lambda_2\Sigma & e & 2\Sigma w \\ e^{\mathsf{T}} & 0 & 0 \\ 2(\Sigma w)^{\mathsf{T}} & 0 & 0 \end{bmatrix}$$

- Function to compute $D\,G(w,\lambda)$

```
DG <- function(x, mu, Sigma, sigmaP2)
{
  n <- length(mu)
  grad <- matrix(0.0, n+2, n + 2)
    grad[1:n, 1:n] <- 2*x[n+2]*Sigma
    grad[1:n, n+1] <- 1
    grad[1:n, n+2] <- 2*(Sigma %*% x[1:5])
    grad[n+1, 1:n] <- 1
    grad[n+2, 1:n] <- 2*t(x[1:5]) %*% Sigma
    grad
}
```

R Implementation

- From starting point

```
> x <- c(rep(0.5, 5), 1, 1)
> x
[1] 0.5 0.5 0.5 0.5 0.5 1.0 1.0
```

- Newton iterations

```
> for(i in 1:25)
  x <- x - solve(DG(x, mu, Sigma, 0.25^2),
                 G(x, mu, Sigma, 0.25^2))
```

- Numerical solution

```
> x
[1] -0.39550317  0.09606231  0.04583865  0.70988017
[5]  0.54372203 -0.09200813 -0.85714641
```

Analysis

- Recall: upper-left $n \times n$ block of $D\,G(w, \lambda_c) \sim$ Hessian of $F(w, \lambda_c)$

```
> DG(x, mu, Sigma, sigmaP2)[1:5, 1:5]
          [,1]     [,2]     [,3]     [,4]    [,5]
[1,] -0.03360  0.01296 -0.02208 -0.01500  0.0168
[2,]  0.01296 -0.05554  0.00710  0.01543 -0.0162
[3,] -0.02208  0.00710 -0.09069 -0.03450 -0.0345
[4,] -0.01500  0.01543 -0.03450 -0.10714  0.0225
[5,]  0.01680 -0.01620 -0.03450  0.02250 -0.2100
```

- Can check second order condition by computing eigenvalues

```
> eigen(DG(x, mu, Sigma, sigmaP2)[1:5, 1:5])$values
[1] -0.02024 -0.05059 -0.05806 -0.14445 -0.22364
```

- Computed w is a constrained maximum

```
> t(x[1:5]) %*% mu
[1] 0.1991514
```

COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

http://computational-finance.uw.edu