Advanced Shenanigans

Yufeng Huang

Associate Professor of Marketing, Simon Business School

August 18, 2022

Dictionary

Search for a word

she·nan·i·gans

/SHəˈnanəgənz/

noun INFORMAL

plural noun: **shenanigans**; noun: **shenanigan**

secret or dishonest activity or maneuvering.
"widespread financial shenanigans had ruined the fortunes of many"

- silly or high-spirited behavior; mischief.

Origin

mid 19th century: of unknown origin.

Translate shenanigans to Choose language

Use over time for: shenanigans

Show less

From Oxford

Feedback

What I mean is: let's spend a bit of time on algorithms

- But these are advanced applications
  - optional for the course; but I think they complement what we know nicely
- Specifically, we'll start with manually solving for a nonlinear equation (using a loop)
  - variation of this example: optimization of a (differentiable) function
  - application: profit maximization
  - goal: a) loops/functions, b) understanding how the optimizer works, c) understanding that different algorithms will lead to drastic differences in efficiency
  - why optimization per se? You'll find it useful across many applications of statistical models or profit functions of a company

Solving a nonlinear equation

Solving a non-linear equation

- As the most basic problem, consider solving for a nonlinear equation of the form
$$f(x) = 0$$
where f is a differentiable function (i.e. its derivative exists)
- For this class, I will introduce the standard Newton-Ralphson algorithm, i.e. the following recurve formula will arrive at the solution when $t \to \infty$:
$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$
where $f^{'}$ is the derivative of the function f

But how to find the derivative of f?

- A simple way to take numerical derivatives is to find the slope of $f(x)$ at a very small change of x, i.e. to calculate

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- For example, let's take the derivative for

$$f(x) = x^2 - 4$$

which some of us know (from some math classes) is $f'(x) = 2x$

```r
# let the function f be
f <- function(x) x^2 - 4

# then the derivative at x = 1 is
(f(1 + 0.001) - f(1)) / 0.001

## [1] 2.001

# which we can confirm if we know how to do the math
```

Example 1

- Now, let's find **one** solution for

$$x^2 - 4 = 0$$

```
# let the function f be
f <- function(x) x^2 - 4

# then the solution should be to continue the iteration
x_new <- 0 # starting point
for (i in 1:100) {      # run the iteration 100 times
    # replace previous x_old
    x_old <- x_new

    # take derivative (gradient) of f
    fx <- f(x_old)
    dfx <- (fx - f(x_old - 0.001)) / 0.001

    # evaluate the next x
    x_new <- x_old - fx / dfx
}

# print results
x_new

## [1] -2
```

Comments

- Good starting point, but:
  - the 100 iterations seem unecessary or not enough, depending on how difficult it is to find the solution
  - better is to decide whether to continue the loop using conditions about whether we "*converged*" at a solution
- Natural choice is a while loop:
  - with an explicit condition to continue/stop the iteration
  - a typical stopping criteria is to stop the iteration if changes in x is smaller than some tolerance
  - more sophisticated is to require both convergence in x and convergence in $f(x)$, but we'll keep it simple here

Example 1 modified

```r
# a better choice is a while loop
x_new <- 0 # starting point
tol_X <- 1E-6   # stopping criteria
x_old <- 1      # placeholder, you'll see why we need this

while (abs(x_new - x_old) > tol_X) {    # continue if diff in x is large
    # replace previous x_old
    x_old <- x_new

    # take derivative (gradient) of f
    fx <- f(x_old)
    dfx <- (fx - f(x_old - 0.001)) / 0.001

    # evaluate the next x
    x_new <- x_old - fx / dfx
}

# print results
x_new

## [1] -2
```

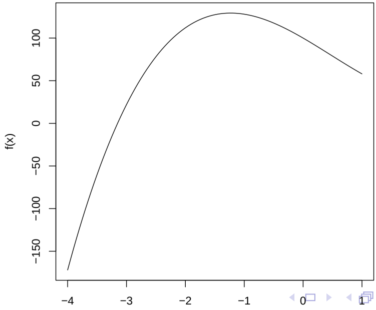
Example 2

- Now, let's find **one** solution for

$$5x^3 - 7x^2 - 40x + 100 = 0$$

but this time do it using a function

```
# let the function f be
f <- function(x) {
    5 * x^3 - 7 * x^2 - 40 * x + 100
}
# show the function in a figure
range_x <- seq(-4, 1, by = 0.05)
plot(range_x, f(range_x), type = 'l', xlab = 'x', ylab = 'f(x)')
```



Example 2

```r
# write the routine into a function:
solve_f <- function(par, fn, tolX = 1E-6, gr = NULL) {

    # initialize
    x_new <- par
    x_old <- par + 1

    # loop
    while (abs(x_new - x_old) > tolX) {

        # replace previous x_old
        x_old <- x_new
        fx <- fn(x_old)

        # take derivative (gradient) of f (address this later)
        if (is.null(gr)) {
            # if gradient does not exist
            dfx <- (fx - fn(x_old - 0.001)) / 0.001
        } else {
            # otherwise, gradient is the gradient function evaluated at x_old
            dfx <- gr(x_old)
        }

        # evaluate the next x
        x_new <- x_old - fx / dfx

    }

    return(list(par = x_new, f = fn(x_new)))

}
```

Example 2

```
# call the solve_f function with standard arguments
solve_f(par = 0, fn = f)

## $par
## [1] -3.151719
##
## $f
## [1] -2.960164e-09

# however, now that we can specify the gradient argument gr, let's see what it does
#    recall that we can numerically take the gradient as something like (fx - f(x_old - 0.001)) / 0.001
#    however, we can explicitly take the derivative as use it as the gradient
#    for function f here, the derivative is 5 * 3 * x^2 - 7 * 2 * x - 40
g_f <- function(x) {
    5 * 3 * x^2 - 7 * 2 * x - 40
}

# now, call solve_f with the gradient argument
solve_f(par = 0, fn = f, gr = g_f)

## $par
## [1] -3.151719
##
## $f
## [1] -4.263256e-14

# why am I covering this?
#    optim and other solvers also use the gr argument, this is the way to provide it
```

Optimization

Example 3: optimization

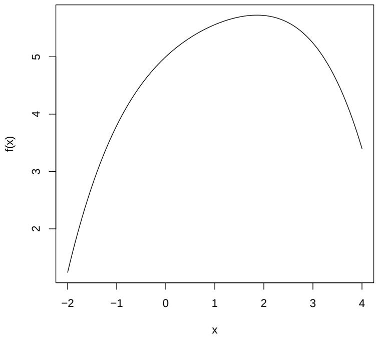
- At this point one asks: why are we learning how to solve nonlinear equations?
- Answer: this is the fundamental to many applications
- Example of such an application: we will code up our own optimizer to replace optim
  - well, we'll stick to a one-dimensional problem for simplicity
- To be specific: we want to maximize the function

$$\text{objfun}\left(x\right) = -0.02x^4 + 0.08x^3 - 0.3x^2 + 0.8x + 5$$

how do we do this by hand?

Example 3

```
# let function objfun be
objfun <- function(x) {
    -0.02 * x^4 + 0.08 * x^3 - 0.3 * x^2 + 0.8 * x + 5
}
# examine the function
range_x <- seq(-2, 4, by = 0.05)
plot(range_x, objfun(range_x), type = 'l', xlab = 'x', ylab = 'f(x)')
```



Example 3

- How do we optimize this? Note that for differentiable functions, a necessary condition for the maximum (or minimum) is where the derivative is zero
- So we can use the previously-defined solve_f() function to find the point where derivative is zero

```
# take numerical derivative
numd_objfun <- function(x) {
        (objfun(x + 0.0001) - objfun(x)) / 0.0001
}

# find the point where the derivative is zero
x1 <- solve_f(par = 0, fn = numd_objfun)
x1

## $par
## [1] 1.859048
##
## $f
## [1] 8.881784e-12

# what is the function value?
objfun(x1$par)

## [1] 5.725532

# compare against standard package
optim(0, function(x) -objfun(x))$par

## [1] 1.859375
```

Example 3

- What we've seen is essentially the optim() function in the simplest form (or the Excel solver, etc.)
  - but there are many ways to improve it
  - for example, using analytical derivatives instead of numerical ones

```
# take analytical derivative (i.e. ones we derive using math)
d_objfun <- function(x) {
    -0.02 * 4 * x^3 + 0.08 * 3 * x^2 - 0.3 * 2 * x + 0.8
}

# find the point where the analytical derivative is zero
x2 <- solve_f(par = 0, fn = d_objfun)
x2$par  # slightly different

## [1] 1.859098

# can even supply the second derivative
#    which will be useful in solving for df(x) = 0
d2_objfun <- function(x) {
    -0.02 * 4 * 3 * x^2 + 0.08 * 3 * 2 * x - 0.3 * 2
}
x3 <- solve_f(par = 0, fn = d_objfun, gr = d2_objfun)
x3$par

## [1] 1.859098
```

Summary

- Two points to learn from this section
  - how do we implement an algorithm using a while/for loop?
  - how does an optimizer work (in a simple way)?
- Next: real application
  - how do we maximize profit by finding the right quantity?
- Other applications?
  - how do we estimate a linear regression (or any linear and nonlinear models)?

Profit maximization in a competitive market (aka Samsung vs HTC redux)

Example 4: profit maximization and competition

- Recall that in the Samsung vs HTC case, we understood how to solve for the equilibrium in the following loop:
  - for each month, observe the market condition, define the game
  - for each combination of quantities q_H and q_S, determine the market price and profits
    - holding Samsung's quantity, HTC chooses q_H to maximize its profit
    - holding HTC's quantity, Samsung chooses q_S to maximize its profit
    - and so on and so forth...
- Looks cumbersome, right?
- Let's try to crack this problem in a simpler way

Example 4: the old method

```
# take a look at the old method
#    first part of market.simulator.R

# ===== Step 1: construct profit matrix =====
pos.qty <- 1:nq

# initialize profit
profit_S <- array(NA, c(nq, nq))
profit_H <- array(NA, c(nq, nq))

# double for-loop to get the profit matrix
#    note: don't have /100 any more
for (q_S in pos.qty) {
    for (q_H in pos.qty) {
        price <- b1*income + b2*log(mktsize) - b3*(q_S + q_H)  # more people: price decrease not as fast w
        profit_S[q_S, q_H] <- q_S * (price - mc_S)
        profit_H[q_S, q_H] <- q_H * (price - mc_H)
    }
}
```

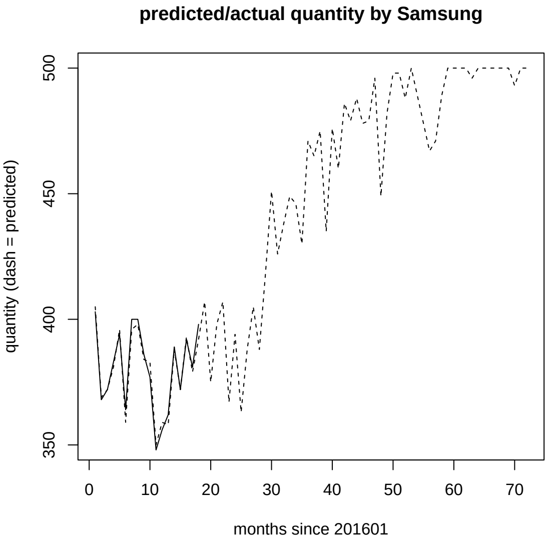
Example 4: the old method

```r
#    second part of market.simulator.R

# ===== Step 2: while loop to find best response / equilibrium quantity =====
#   NOTE: This is what your market research team tells you

# initialize
i <- 300
j <- 100
change <- 1000

# keep iterating until no one wants to choose a different quantity
while (change > 0) {

    # 1. store previous values
    i_old <- i
    j_old <- j

    # 2. Samsung's response
    i <- which.max(profit_S[, j])

    # 3. HTC's response
    j <- which.max(profit_H[i, ])

    # 4. calculate changes from prev value
    change <- abs(i - i_old) + abs(j - j_old)

}

# quantities
q_S <- pos.qty[i]
q_H <- pos.qty[j]

# price
price <- b1*income + b2*log(mktsize) - b3*(q_S + q_H)
```

Example 4: application of the old method

```r
# reads data
df <- read.table('mobile_phone.csv', header = T, stringsAsFactors = F)

# parameters (say you know these)
nq <- 500
b1 <- 250
b2 <- 150
b3 <- 0.3

# generate 3 columns as the predicted quantity/price
df$quantity_S_pred <- NA
df$quantity_H_pred <- NA
df$price_pred <- NA

# run through all 72 periods (past and future)
for (t in 1:72) {
        # assign some values
        income <- df$income[t]
        mktsize <- df$mktsize[t]
        mc_S <- df$mc_S[t]
        mc_H <- df$mc_H[t]

        # run the market simulator every time period
        source('market.simulator.R')

        # collect results
        df$quantity_S_pred[t] <- q_S # recall that reacted.quantity is HTC - Samsung
        df$quantity_H_pred[t] <- q_H
        df$price_pred[t] <- price
}
```

```
# can compare predictions
df$t <- (df$year - min(df$year))*12 + df$month
plot(df$t, df$quantity_S_pred,
        type = 'l', lty = 2, xlab = "months since 201601", ylab = "quantity (dash = predicted)",
        main = "predicted/actual quantity by Samsung")
points(df$t, df$quantity_S, type = 'l')
```



However, how much time do we waste?

```
# make the loop into a function so that we can evaluate this
test1 <- function() {
        for (t in 1:72) {
                # assign some values
                income <- df$income[t]
                mktsize <- df$mktsize[t]
                mc_S <- df$mc_S[t]
                mc_H <- df$mc_H[t]

                # run the market simulator every time period
                source('market.simulator.R')

                # collect results
                df$quantity_S_pred[t] <- q_S # recall that reacted.quantity is HTC - Samsung
                df$quantity_H_pred[t] <- q_H
                df$price_pred[t] <- price
        }
}

library(rbenchmark)
benchmark(test1(), replications = 1)

##      test replications elapsed relative user.self sys.self user.child sys.child
## 1 test1()            1    5.28        1      5.17      0.1         NA        NA
```

Thoughts

- In the above algorithm
  - have to calculate profits in every combinations of q_H and q_S
  - and loop over 72 months
  - what would happen if q_H and q_S do not only have 500 possible values, but 50,000?
- We observe that maybe we do not have to exhaust all possible quantity combinations
  - e.g. some quantities result in low profits and will never be chosen
  - which leads to the next algorithm (plus the code will be more elegant)

Improved algorithm (1)

```
# re-define the price and profit functions
find_price <- function(q1, q2, income, mktsize) {
        price <- b1*income + b2*log(mktsize) - b3*(q1 + q2)     # q1 and q2 interchangeable
        return(price)
}
find_profit <- function(q, q_opp, mc, income, mktsize) {
        profit <- q * (find_price(q, q_opp, income, mktsize) - mc)
        return(profit)
}
```

Improved algorithm (2)

```
# first, setup the variables

# focus on month 1
income <- df$income[1]
mktsize <- df$mktsize[1]
mc_S <- df$mc_S[1]
mc_H <- df$mc_H[1]

# setup the initial variables
q_S_old <- 0
q_H_old <- 0
q_S <- 300
q_H <- 100
```

Improved algorithm (3)

```
# for a given time period, simply iteratively optimize profit
#    until neither firm wants to deviate from a given quantity

while (abs(q_S - q_S_old) > 1 | abs(q_H - q_H_old) > 1) {

        # update the _old variables
        q_S_old <- q_S
        q_H_old <- q_H

        # Samsung maximizes profit
        q_S <- optim(par = q_S_old, fn = function(q) {
                -find_profit(q, q_H, mc_S, income, mktsize)
        })$par

        # HTC maximizes profit
        q_H <- optim(par = q_H_old, fn = function(q) {
                -find_profit(q, q_S, mc_H, income, mktsize)
        })$par

}

# examine results
q_S

## [1] 404.5878

q_H

## [1] 304.3925
```

```r
# can now write the above into a function (that can go across many time periods)

solve_equilibrium <- function(t) {

        # initialize
        income <- df$income[t]
        mktsize <- df$mktsize[t]
        mc_S <- df$mc_S[t]
        mc_H <- df$mc_H[t]
        q_S_old <- 0
        q_H_old <- 0
        q_S <- 300
        q_H <- 100

        # loop
        while (abs(q_S - q_S_old) > 1 | abs(q_H - q_H_old) > 1) {
                # update the _old variables
                q_S_old <- q_S
                q_H_old <- q_H
                # Samsung maximizes profit
                q_S <- optim(par = q_S_old, fn = function(q) {
                        -find_profit(q, q_H, mc_S, income, mktsize)
                })$par
                # HTC maximizes profit
                q_H <- optim(par = q_H_old, fn = function(q) {
                        -find_profit(q, q_S, mc_H, income, mktsize)
                })$par
        }

        # return
        return(c(q_S = q_S, q_H = q_H))

}

solve_equilibrium(1)

##      q_S      q_H
## 404.5878 304.3925
```

```
# then we loop over this algorithm over t = 1:72, see the speed differences

test2 <- function() {
        for (t in 1:72) {

                # solve for equilibrium
                res <- solve_equilibrium(t)

                # collect results
                df$quantity_S_pred[t] <- res["q_S"]
                df$quantity_H_pred[t] <- res["q_H"]

        }
}

# benchmark
benchmark(test1(), test2(), replications = 1)

##      test replications elapsed relative user.self sys.self user.child sys.child
## 1 test1()            1    5.18   43.167      5.00     0.17         NA        NA
## 2 test2()            1    0.12    1.000      0.12     0.00         NA        NA
```

- HUGE speed gain, because we have taken away much of the wasteful calculations!
- But, can we do even better??

Pushing it even further...

- But, can we do even better??
- Note that instead of finding the maximum profit for each firm, holding the other firm's strategy fixed...
  - ... we can find such point where the derivative of profit is **zero** for each firm, holding the other firm's strategy fixed
  - of course, relies on a little bit of math, but the derivative of profit is still not very difficult: profit is

    $$\text{profit}(q) = q \cdot (p(q) - mc)$$

    then the derivative of it is

    $$\begin{aligned} \text{profit}^{'}(q) &= (p(q) - mc) + q \cdot p^{'}(q) \\ &= (p(q) - mc) - b_3 \cdot q \end{aligned}$$

    where the second line follows the linear inverse demand function

```r
# construct the derivative function
d_profit <- function(q, q_opp, mc, income, mktsize, b3) {
        (find_price(q, q_opp, income, mktsize) - mc) - b3 * q
}

# now, can solve for zero-derivatives
solve_equilibrium_2 <- function(t) {
        # initialize
        income <- df$income[t]
        mktsize <- df$mktsize[t]
        mc_S <- df$mc_S[t]
        mc_H <- df$mc_H[t]
        q_S_old <- 0; q_H_old <- 0; q_S <- 300; q_H <- 100

        # loop
        while (abs(q_S - q_S_old) > 1 | abs(q_H - q_H_old) > 1) {
                # update the _old variables
                q_S_old <- q_S
                q_H_old <- q_H
                # Samsung finds q_S where derivative of profit is zero
                q_S <- solve_f(par = q_S_old, fn = function(q) {
                        d_profit(q, q_H, mc_S, income, mktsize, b3)
                })$par
                # HTC finds q_H where derivative of profit is zero
                q_H <- solve_f(par = q_H_old, fn = function(q) {
                        d_profit(q, q_S, mc_H, income, mktsize, b3)
                })$par
        }

        # return
        return(c(q_S = q_S, q_H = q_H))

}

solve_equilibrium_2(1)

##      q_S      q_H
## 404.5658 304.4161
```

Putting it all together

- Now, you might ask: what's the point of defining a new d_profit function just to use our solve_f()?
  - well, if you think that's the only modification we're going to do, you're under-estimating the potential for the derivative function!
- The true purpose: we observe that the equilibrium is reached where for both firms, the derivative of profit is zero, i.e.

  $$\text{profit}_S^{'}\left(q_S\right)|q_H = 0$$

  for Samsung and for HTC,

  $$\text{profit}_H^{'}\left(q_H\right)|q_S = 0$$

- This gives us a system of equations that will completely get around the while loop, and instead replace it with a system of two equations (which we'll solve using a package)

```r
# MODIFY the derivative function such that
d_profit_vec <- function(q_S, q_H, mc_S, mc_H, income, mktsize, b3) {
        p <- find_price(q_S, q_H, income, mktsize)
        c((p - mc_S) - b3 * q_S, (p - mc_H) - b3 * q_H)
}

# load a nonlinear equation solver
library(nleqslv)

# define this procedure on one period
solve_equilibrium_3 <- function(t) {
        # initialize
        income <- df$income[t]
        mktsize <- df$mktsize[t]
        mc_S <- df$mc_S[t]
        mc_H <- df$mc_H[t]

        # solve the system using fsolve()
        res <- nleqslv(f = function(q) d_profit_vec(q[1], q[2], mc_S, mc_H, income, mktsize, b3),
                x = c(300, 100))
        # return
        return(c(q_S = res$x[1], q_H = res$x[2]))

}

# try it on period 1!
res <- solve_equilibrium_3(1)
res[1:2]

##     q_S     q_H
## 404.466 304.466
```

```r
# loop over periods 1-72
test3 <- function() {
        for (t in 1:72) {
                res <- solve_equilibrium_3(t)
        }
        return(res)
}

# benchmark
benchmark(test1(), test2(), test3(), replications = 10) #!!

##      test replications elapsed relative user.self sys.self user.child sys.child
## 1 test1()           10   52.50   1050.0     51.25      1.2         NA        NA
## 2 test2()           10    1.17     23.4      1.17      0.0         NA        NA
## 3 test3()           10    0.05      1.0      0.05      0.0         NA        NA

# find that the new algorithm does about 1000 times better than
#    the very first algorithm
```

Conclusion

- The central message I wanted to convey: **algorithm matters**
- It's easy to boast the computer's hardware if you're rich
  - but it's more impressive if you beat everyone else with speed using the same hardware
  - (and practically, the only way out of a tech interview when algorithms are involved)
- Of course, this lecture note is more advanced than the rest and definitely won't appear in the final