Computational Statistics Summary

Maximilian Pfundstein

2019-03-11

Contents

1 Handeling Computational Errors **1**

2 Difference Quotient **1**

3 Variance Estimators **2**

4 Optimization **2**
4.1 `optimize()` . . . 2
4.2 `optim()` . . . 3

5 Sampling Based on Size **4**

6 Inverse CDF Method **5**

7 Acceptance / Rejection Method **6**

8 Metropolis-Hastings Algorithm **8**

9 Gelman-Rubin Factor **10**

10 Integral Estimation **12**

11 Gibbs Sampling **12**

12 General Plots **14**
12.1 Histrogram . . . 14
12.2 Simple X/Y Plot . . . 14

1 Handeling Computational Errors

```
x1 = 1/3
x2 = 1/4

if (all.equal(x1-x2, 1/12)) {
  print("Substraction is correct.")
} else {
  print("Substraction is wrong.")
}
```

2 Difference Quotient

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

```r
f_prime = function(x, epsilon = 10^(-5))  {
  return( (f(x + epsilon) - f(x)) / epsilon)
}
```

3 Variance Estimators

Krzysztof:

$$\mathrm{Var}(\vec{x}) = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right)$$

```r
myvar = function(x) return(1/(length(x)-1) * (sum(x^2) - (sum(x)^2)/length(x)))
```

My:

$$s = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$$

```r
custom_variance = function(x) {
  diff_mean = x - mean(x)
  return(sum(diff_mean^2 / (length(x) - 1)))
}
```

4 Optimization

4.1 `optimize()`

```r
myMSE = function(lambda, pars) {
  model = loess(Y ~ X, data=pars, enp.target = lambda)
  prediction = predict(model, newdata = pars$Xtest)
  mse = sum((prediction - pars$Ytest)^2)/length(pars$Ytest)
  return(mse)
}

# parameters for the myMSE function ----------------------
pars = list(X = train$Day, Y = train$LMR, Xtest = test$Day, Ytest = test$LMR)
lambdas = seq(from = 0.1, to = 40, by = 0.1)
# applying the myMSE function to all lambdas ------------
mses = sapply(X = lambdas, FUN = myMSE, pars = pars)

o = optimize(myMSE, tol = 0.01, interval = c(0.1, 40), pars = pars)
o$minimum
o$objective
```

Plotting a function with a minimum:

```r
lambdas[which.min(mses)]
length(lambdas)
```

```r
df = data.frame(lambdas, mses)

ggplot(df) +
  geom_line(aes(x = lambdas, y = mses), color = "#C70039") +
  geom_point(aes(x = seq(0.1, 40, by = 0.1)[which.min(mses)],
  y = mses[which.min(mses)], color = "min MSE"),
  colour = "blue") +
  labs(title = "Lambdas VS MSEs", y = "MSE", x = "Lambda") +
  theme_minimal()
```

4.2 optim()

General usage:

```r
optim(35, myMSE, method = "BFGS", pars = pars, control = list(fnscale = 1))
```

For optimizing likelihood:

```r
# c(mu, sigma)
neg_llik_norm = function(par) {
  n = nrow(as.matrix(data))
  p1 = (n/2)*log(2*pi)
  p2 = (n/2)*log(par[2]^2)
  sum = sum((data - par[1])^2)
  p3 = 1/(2*par[2]^2) * sum
  return(p1+p2+p3)
}

# c(mu, sigma)
neg_llik_norm_prime = function(par) {
  n = nrow(as.matrix(data))
  mu_prime = -1/(n*par[2]^2) * sum(data-par[1])
  sigma_prime = 1/(2*par[2]^2) * (n - (1/(par[2]^2)) * sum((data-par[1])^2))

  return(c(mu_prime, sigma_prime))
}

optim(c(0, 1), neg_llik_norm, method = "CG")
optim(c(0, 1), neg_llik_norm, method = "CG", gr = neg_llik_norm_prime)
optim(c(0, 1), neg_llik_norm, method = "BFGS")
optim(c(0, 1), neg_llik_norm, method = "BFGS", gr = neg_llik_norm_prime)
```

Answer: The negative log-likelihood function for the normal distribution is defined by:

$$\mathcal{L}(\mu, \sigma^2, x_1, ..., x_{100}) = \frac{n}{2} ln(2\pi) + \frac{n}{2} ln(\sigma^2) + \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2$$

The estimators are:

$$\hat{\mu}_n = \frac{1}{n} \sum_{j=1}^{n} x_j$$

and

$$\hat{\sigma}_n^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \hat{\mu})^2$$

Answer: The partial derivates for the negative log-likelihood are given by:

$$\frac{\partial \mathcal{L}(\mu, \sigma^2, x_1, ..., x_{100})}{\partial \mu} = -\frac{1}{n\sigma^2}\sum_{j=1}^{n}(x_j - \mu)$$

$$\frac{\partial \mathcal{L}(\mu, \sigma^2, x_1, ..., x_{100})}{\partial \sigma^2} = \frac{1}{2\sigma^2}\left(n - \frac{1}{\sigma^2}\sum_{j=1}^{n}(x_j - \mu)^2\right)$$

5 Sampling Based on Size

Task: Use a uniform random number generator to create a function that selects 1 city from the whole list by the probability scheme offered above (do not use standard sampling functions present in R).

```
get_city_by_urn_wo = function(city_pool) {

  # We take the cumulative sum and then runif from 1 to max(cumulative sum).
  # This way we respect the proportions. As we need every intermediate result,
  # we use a loop
  cumulative_pop_sum = 0

  for (i in 1:nrow(city_pool)) {
    cumulative_pop_sum = cumulative_pop_sum + city_pool$Population[i]
    city_pool$CumSum[i] = cumulative_pop_sum
  }

  # Now we get a random value between 1 to max(cumulative sum). As larger muni-
  # cipalities have larger ranges, this works as expected
  selection =
    floor(runif(n = 1, min = 1, max = city_pool$CumSum[nrow(city_pool)]))

  # Return the first city which has a greater CumSum than the selection
  return(city_pool[city_pool$CumSum > selection,][1, c(1, 2)])
}
```

Task: Use the function you have created in step 2 as follows:

a. Apply it to the list of all cities and select one city
b. Remove this city from the list
c. Apply this function again to the updated list of the cities
d. Remove this city from the list
e. ... and so on until you get exactly 20 cities.

Answer: We will combine all of these steps in one function. We're lazy.

```
get_n_cities = function(data, n) {

  # Create a copy to not touch the original data.
  city_pool = data
  selected_cities = data.frame()
```

```
  # As long as we don't have n samples, get one and remove it from the pool,
  # as we sample without replacement
  while(nrow(selected_cities) < n) {
    selected_city = get_city_by_urn_wo(city_pool)
    selected_cities = rbind(selected_cities, selected_city)
    city_pool = city_pool[!rownames(city_pool) %in% rownames(selected_cities),]
  }

  return(selected_cities)
}

sample = get_n_cities(data, 20)
```

6 Inverse CDF Method

The double exponential (Laplace) distribution is given by formula

$$DE(\mu, \alpha) = \frac{\alpha}{2} e^{-\alpha|x-\mu|}$$

Task: Write a code generating double exponential distribution $DE(0,1)$ from $Unif(0,1)$ by using the inverse CDF method. Explain how you obtained that code step by step. Generate `10000` random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

1. Derive the CDF from the PDF (`dfunc()`)by taking the integral $\int_{-\infty}^{x} \text{CDF} dx$. This function is the `pfunc()` (CDF!, cumulative).
2. Swap `x` and `y` to receive the quantile function `qfunc()`.
3. Combine both functions to create the `rfunc()`.

This can look like this:

```
# double exponential (Laplace) distribution

# PDF
ddel = function(x = 1, mu = 0, b = 1) {
return(1/(2*b) * exp(-abs(x-mu)/(b)))
}

# CDF
pdel = function(x = 1, mu = 0, b = 1) {
  return(1/2 + 1/2 * sgn(x-mu) * (1 - exp(-abs(x-mu)/b)))
}

# Quantile
qdel = function(p, mu = 0, b = 1) {
  if (p < 0 | p > 1) stop("p must be in range (0, 1)")
  if (p <= 0.5) return(mu + b * log(2 * p))
  return (mu - b * log(2 - 2 * p))
}

# Random
rdel = function(n = 1, mu = 0, b = 1) {
  quantiles = runif(n = n, min = 0, max = 1)
```

```
  rdels = sapply(X = quantiles, FUN = qdel, mu = mu, b = b)
  return(rdels)
}
```

To look how the distribution looks like with different parameters, use the following code and plots:

```
sample_rdel_0_1 = rdel(10000, mu = 0, b = 1)
sample_rdel_0_2 = rdel(10000, mu = 0, b = 2)
sample_rdel_0_4 = rdel(10000, mu = 0, b = 4)
sample_rdel_m5_4 = rdel(10000, mu = -5, b = 4)

df = data.frame(sample_rdel_0_1, sample_rdel_0_2,
sample_rdel_0_4, sample_rdel_m5_4)

p1 = ggplot(df) +
  geom_histogram(aes(x = sample_rdel_0_1),
  color = "#FFC300", fill = "#FFC300", binwidth = 0.01) +
  xlim(-10, 10) +
  ylim(0, 60) +
  ggtitle("RD(0, 1)") +
  theme_minimal()

# p2, p3, p4

grid.arrange(p1, p2, p3, p4, nrow = 2)
```

7 Acceptance / Rejection Method

For plotting the PDF and CDF to compare for instance two of them:

```
sequence = seq(from = -10, to = 10, by = 0.01)

dnorm_samples = sapply(X = sequence, FUN = dnorm)
ddel_samples = sapply(X = sequence, FUN = ddel)
pnorm_samples = sapply(X = sequence, FUN = pnorm)
pdel_samples = sapply(X = sequence, FUN = pdel)

df = data.frame(dnorm_samples, ddel_samples, pnorm_samples, pdel_samples)

ggplot(df) +
  geom_line(aes(x = sequence, y = dnorm_samples,
                colour = "Normal Distribution (PDF)")) +
  geom_line(aes(x = sequence, y = ddel_samples,
                colour = "Double Exponential Distribution (PDF)")) +
  labs(title = "dnorm() and ddel()", y = "Density",
  x = "X", color = "Legend") +
  scale_color_manual(values = c("#17202A", "#C70039")) +
  theme_minimal()

ggplot(df) +
  geom_line(aes(x = sequence, y = pnorm_samples,
                colour = "Normal Distribution (CDF)")) +
  geom_line(aes(x = sequence, y = pdel_samples,
```

```
                colour = "Double Exponential Distribution (CDF)")) +
  labs(title = "pnorm() and pdel()", y = "Cumulative Density",
  x = "X", color = "Legend") +
  scale_color_manual(values = c("#17202A", "#C70039")) +
  theme_minimal()
```

For calculating the c value, use this code:

```
c = max(dnorm_samples / ddel_samples)
```

For creating nice plots, use this:

```
df$scaled_envelop = c * df$ddel_samples

ggplot(df) +
  geom_line(aes(x = sequence, y = dnorm_samples,
                colour = "Normal Distribution (PDF)")) +
  geom_line(aes(x = sequence, y = ddel_samples,
                colour = "Double Exponential Distribution (PDF)")) +
  geom_line(aes(x = sequence, y = scaled_envelop,
                colour = "Scaled Double Exponential Distribution (PDF)")) +
  labs(title = "Envelope", y = "Density",
  x = "X", color = "Legend") +
  scale_color_manual(values = c("#17202A", "#C70039", "#581845")) +
  theme_minimal()

ggplot(df) +
  geom_ribbon(aes(x = sequence, ymin = df$dnorm_samples, ymax = df$scaled_envelop),
              alpha = 0.8, fill = "#C70039", color = "#C70039") +
  geom_ribbon(aes(x = sequence, ymin = 0, ymax = df$dnorm_samples),
              alpha = 0.8, fill = "#DAF7A6", color = "#DAF7A6") +
  labs(title = "Acceptance and Rejection Regions", y = "Density",
  x = "X", color = "Legend") +
  scale_color_manual(values = c("#17202A", "#C70039", "#581845")) +
  theme_minimal()
```

And for actually using it, use:

```
rs = c()
rs_rejected = c()

while (length(rs) < 2000) {
  # Take a random sample from our proposal (x-axis)
  z = rdel(n = 1, mu = 0, b = 1)

  # Take a unform, thus a random y value
  u = runif(n = 1, min = 0, max = c * ddel(z))

  # Check in which region this on lies
  if (u <= dnorm(z)) {
    rs = c(rs, z)
  }
  else {
    rs_rejected = c(rs_rejected, z)
  }
}
```

Plot the drawn samples:

```
df2 = as.data.frame(rs)

ggplot(df2) +
  geom_histogram(aes(x = rs),
                 color = "#C70039", fill = "#C70039", binwidth = 0.01) +
  xlim(-5, 5) +
  ylim(0, 30) +
  ggtitle("N(0, 1) sampled from DE(0, 1)") +
  theme_minimal()
```

Expected rejection rate:

```
1 - 1/c
```

Observed rejection rate:

```
length(rs_rejected) / (length(rs)+length(rs_rejected))
```

8 Metropolis-Hastings Algorithm

We have given a target function (probably only a proportional one).

```
# Target function with original scaling
f = function(x) {
  return(120 * x^5 * exp(-x))
}

# Target function
df = function(x) {
  x = ifelse(x <= 0, 0.000001, x)
  return(x^5 * exp(-x))
}

sequence = seq(from = 0.01, to = 20, by = 0.01)

real_f = f(sequence)
plotdf = data.frame(sequence, real_f)

ggplot(plotdf) +
  geom_line(aes(x = sequence, y = real_f), color = "#6091ec") +
  labs(title = "Target Density Function", y = "Density",
  x = "X", color = "Legend") +
  theme_minimal()
```

The Metropolis-Hastings algorithm is implemented by:

```
#' Metropolis Hasting Algorithm
#'
#' @param n Number of samples from the target distribution.
#' @param x_0 Initial state from Pi.
#' @param b Burn-in steps to remove from samples.
#' @param proposal Selects the proposal function.
#' @param keep_burnin Decides if to keep the samples during the burn-in period. #'
#' @return Returns a list containing samples.
```

```r
#' @export
#'
#' @examples
metropolis_hastings = function(n = 1, x_0 = 1, b = 50, proposal = "lnorm",
                               keep_burnin = FALSE) {

  # Vectors to store the samples in
  samples = c()

  # Samples from proposel from the random walk (MC)
  xt = x_0
  xt_1 = x_0

  while (length(samples) < n)  {

    # Generate proposal state
    if (proposal == "lnorm") {
      x_star = rlnorm(n = 1, meanlog = log(xt), sdlog = 1)

      # Calculate correction factor C
      c = dlnorm(xt_1, meanlog = log(xt), sdlog = 1) /
      dlnorm(x_star, meanlog = log(xt), sdlog = 1)
    }
    else if (proposal == "chisquared") {
      x_star = rchisq(n = 1, df = floor(xt + 1))

      # Calculate correction factor C
      c = dchisq(x = xt_1, df = floor(xt + 1)) /
      dchisq(x_star, df = floor(xt + 1)) }
    else {
      stop("Invalid proposal.")
    }

    # Calculate acceptance probability alpha
    if (df(xt_1) <= 0) {
      # We need this to avoid troublesome areas where the density is so low that
      # it's 0 from a computiationally point of view.
      alpha = 0
    }
    else {
      alpha = min(1, df(x_star)/df(xt_1) * c)
    }

    # Generate u from uniform
    u = runif(n = 1, min = 0, max = 1)

    # Decide if to accept or reject the proposal
    if (u <= alpha) { # Accept
      xt_1 = xt
      xt = x_star
      samples = c(samples, x_star)
    }
    else {
```

```
      # Reject
      xt = xt_1
      }
  }

  # Return samples
  if (keep_burnin) return(samples) {
    return(samples[b+1:length(samples)])
  }
}
```

Create a fancy plot of the burn-in period using:

```
burnin = 10

results = metropolis_hastings(100, x_0 = 40, b = burnin, proposal = "lnorm",
                              keep_burnin = TRUE)

plotdf = data.frame(index = 1:length(results), values = results)

ggplot(plotdf) +
  geom_line(aes(x = index, y = values), color = "#6091ec") +
  labs(title = "Traceplot For The Burn-In Period (burnin = 10)", y = "X*",
  x = "Iteration", color = "Legend") +
  geom_vline(xintercept = burnin, color = "#C70039") +
  theme_minimal()
```

And to plot the distribution of the samples, use:

```
results_lnorm = metropolis_hastings(10000, x_0 = 5, b = 100, proposal = "lnorm")

plotdf = data.frame(results_lnorm)

ggplot(plotdf) +
  geom_histogram(aes(x = results_lnorm),
                 color = "#000000", fill = "#C70039",
                 bins = length(results_lnorm)/100) +
  labs(title = "Samples From Target Function Using Normal Proposal",
       y = "Density",
  x = "X", color = "Legend") +
  theme_minimal()
```

9 Gelman-Rubin Factor

The custom implementation looks like this:

```
gelman_rubin_factor = function(sequence_matrix) {

  k = nrow(sequence_matrix)
  n = ncol(sequence_matrix)

  B = n/(k-1) * sum((rowMeans(sequence_matrix) - mean(sequence_matrix))^2)
  S_squared = rowSums((sequence_matrix - rowMeans(sequence_matrix))^2 / (n - 1))
  W = sum(S_squared / k)
```

```
  V = ((n - 1) / n * W) + (1/n * B)
  R = sqrt(V / W)
  return(R)
}
```

And calling it with the previously defined Metropolis-Hastings algorithm, use:

```
k = 10 # row
n = 1000 # col

sequence_matrix = matrix(NaN, nrow = k, ncol = n)

for (i in 1:k) {
  sequence_matrix[i,] = metropolis_hastings(n, x_0 = i, b = 0,
                                    proposal = "chisquared", keep_burnin = TRUE)
}

print(gelman_rubin_factor(sequence_matrix))

k = 10 # row
n = 5 # col

sequence_matrix = matrix(NaN, nrow = k, ncol = n)

for (i in 1:k) {
  sequence_matrix[i,] = metropolis_hastings(n, x_0 = i, b = 0,
                                    proposal = "chisquared", keep_burnin = TRUE)
}

print(gelman_rubin_factor(sequence_matrix))
```

There is also a built-in implementation which can be invoked with the following code:

```
library(coda)

results5 = list()
results10 = list()
results50 = list()
k = 10

for (i in 1:k) {
  results5[[i]] = mcmc(metropolis_hastings(5, x_0 = i, b = 0,
                                    proposal = "chisquared", keep_burnin = TRUE))
}

for (i in 1:k) {
  results10[[i]] = mcmc(metropolis_hastings(10, x_0 = i, b = 0,
                                    proposal = "chisquared", keep_burnin = TRUE))
}

for (i in 1:k) {
  results50[[i]] = mcmc(metropolis_hastings(50, x_0 = i, b = 0,
                                    proposal = "chisquared", keep_burnin = TRUE))
}
```

```
mcmc_list5 = mcmc.list(results5)
gelman.diag(mcmc_list5)

mcmc_list10 = mcmc.list(results10)
gelman.diag(mcmc_list10)

mcmc_list50 = mcmc.list(results50)
gelman.diag(mcmc_list50)
```

10 Integral Estimation

Let's say we want to integrate the following integral:

$$\int_0^\infty xf(x)dx$$

Answer: To estimate this integral we take our previous samples. We then calculate:

$$\int_0^\infty xf(x)dx = E[x] \sim \frac{1}{n}\sum_{i=1}^{n} x_i$$

where x_i are our samples. As the decomposed left-side function is just x, this is all we have to do. The limits are given by the proposing function.

Using the different defined proposals we can use the follwing code for the estimation:

```
valid_samples_lnorm = results_lnorm[!is.na(results_lnorm)]
sum(valid_samples_lnorm) / length(valid_samples_lnorm)
```

```
valid_samples_chisquared = results_chisquared[!is.na(results_chisquared)]
sum(valid_samples_chisquared) / length(valid_samples_chisquared)
```

These, of course, do not include the proportional factor.

11 Gibbs Sampling

For Gibbs Sampling it is mandatory to find the Marginals. For doing that, do the following:

- First write down the prior and the likelihood. The is no general way for this, it depends on the problem.
- Then write down the posterior which is the product of both.
- For finding the marginals, drop every factor that is not dependend on the marginal variable.

If we have to combine two normal distributions, it can be done using the following rules:

The product of two normal distributions is given by (see https://www.johndcook.com/blog/2012/10/29/product-of-normal-pdfs/ as a recap if not present):

$$\sigma^2_{new} = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

$$\mu_{new} = \frac{\sigma_1^{-2}\mu_1 + \sigma_2^{-2}\mu_2}{\sigma_1^{-2} + \sigma_2^{-2}}$$

For implementing the Gibbs Sampler we write one function for getting the marginals and one main function for doing the actual sampling.

```
get_mu = function(i, sigma = 0.2, mu, Y) {

  if (i == 1) {
    # First Marginal
    return(rnorm(n = 1, mean = (mu[2] + Y[1])/2, sd = sqrt(1/2 * sigma)))
  }
  else if (i == 50) {
    # Last Marginal
    return(rnorm(n = 1, mean = (2 * mu[i-1] - Y[i-1] + 2 * Y[i])/3,
                 sd = sqrt(2/3 * sigma)))
  }

  # General Marginals
    return(rnorm(n = 1, mean = (2 * mu[i-1] + 2 * mu[i+1] + 2 * Y[i] - Y[i-1])/5,
                 sd = sqrt(2/5 * sigma)))
}

gibbs_sample = function(n, Y, include_matrix = FALSE) {

  # Vectors to store the samples in
  mu_matrix = matrix(0, ncol = length(Y), nrow = n)
  mu = rep(0, length(Y))

  for(j in 1:n) {
    for (i in 1:length(Y)) {
      mu[i] = get_mu(i, sigma = 0.2, mu = mu, Y = Y)
    }
    mu_matrix[j,] = mu
  }

  if (include_matrix) {
    return(list(means = colMeans(mu_matrix), matrix = mu_matrix))
  }

  return(colMeans(mu_matrix))
}
```

Then this function can be used like this. This also includes a plot of the random walk.

```
gibbs_sample_result = gibbs_sample(1000, Y, TRUE)

data$gibbs_samples = gibbs_sample_result$means

ggplot(data) +
  geom_line(aes(x = X, y = Y), color = "#C70039") +
  geom_line(aes(x = X, y = gibbs_samples), color = "#581845") +
  labs(title = "Dependence from Concentration on Day of Measurement", y =
         "Concentration",  x = "Day of Measurement", color = "Legend") +
  theme_minimal()
```

If a trace plot is needed, use this:

```r
df = data.frame(
  iteration = seq(from = 1, to = length(gibbs_sample_result$matrix[,50]), by = 1),
  mu_n = gibbs_sample_result$matrix[,50])

ggplot(df) +
  geom_line(aes(x = iteration, y = mu_n), color = "#C70039") +
  labs(title = "Trace of mu_n", y = "Value of mu_n",  x = "Iteration",
       color = "Legend") +
  theme_minimal()
```

12 General Plots

12.1 Histrogram

General rool for `n` datapoints: $\sqrt{n}$ bars.

```r
ggplot(sample)+
  geom_histogram(aes(x = Population), bins = nrow(sample), color = "black",
                 fill = "#C70039") +
  ggtitle("Histogram of selected cities")
```

12.2 Simple X/Y Plot

```r
ggplot(data) +
  geom_line(aes(x = X, y = Y), color = "#c70039") +
  labs(title = "Dependence from Concentration on Day of Measurement",
       y = "Concentration",  x = "Day of Measurement", color = "Legend") +
  theme_minimal()
```