Problem Set #2: Monte Carlo / Convergence / Seeds /Scripts / Plotting

Anna M. Valentine

Winter 2024

Due: 01/24/2024

Tasks

Task 1 : These readings and example scripts have been reviewed.

Task 2: The scripts I reviewed can be found below:

#1 This is using Bayesian Data assimilation for estimating snow water equivalent (SWE):

https://github.com/fmidev/GlobSnow3.0

#2 and another bayesian approach to snow water equivalent geospatial analysis!

https://github.com/ealonsogzl/MuSA

Task 3: These readings and Labs 0-3 in Applegate & Keller (2016) have been reviewed.

Task 4: Use a Monte Carlo Simulation Method to determine the mean and the 95 percentile from a known univariate normal distribution with a mean of zero and a standard deviation of one with estimated uncertainties. Then find the value of pi and estimated uncertainties.

Monte Carlo Task 4a: Univariate Normal Distribution

Summary: I use R coding script to determine the mean and the 95 percentile from a known univariate normal distribution with a mean of zero and a standard deviation of one, but the twist is that I must use a Monte Carlo approach.

Approach: For this problem, R makes it quite easy to obtain a semi-random gaussian (univariate normal distribution), using the function “rnorm” which gives you a vector of random numbers in a normal distribution (gaussian). I wanted to visually check, so I plotted one of my random gaussian distribution, which you can find on the left of Figure 1. Next, I test to make sure I can get the mean and the 95th quantile using R functions I found through googling around for syntax. Now I know I can generate a univariate normal distribution and find the mean and 95th percentile. Next: Using many seeds to find the mean and 95 percentile, and estimating uncertainty.

To do this, I create a sample called compute_sample, and create a matrix for my sample mean and sample p95 to populate as the computations occur. In this function it loops through different seeds (to understand seed uncertainty) and different samples sizes (to understand sample size uncertainty). This function then returns my matrices with populated values of the mean and 95 percentile for different seeds and samples sizes, as shown on the right of Figure 1. Now I can finally use this to loop through and find our results.

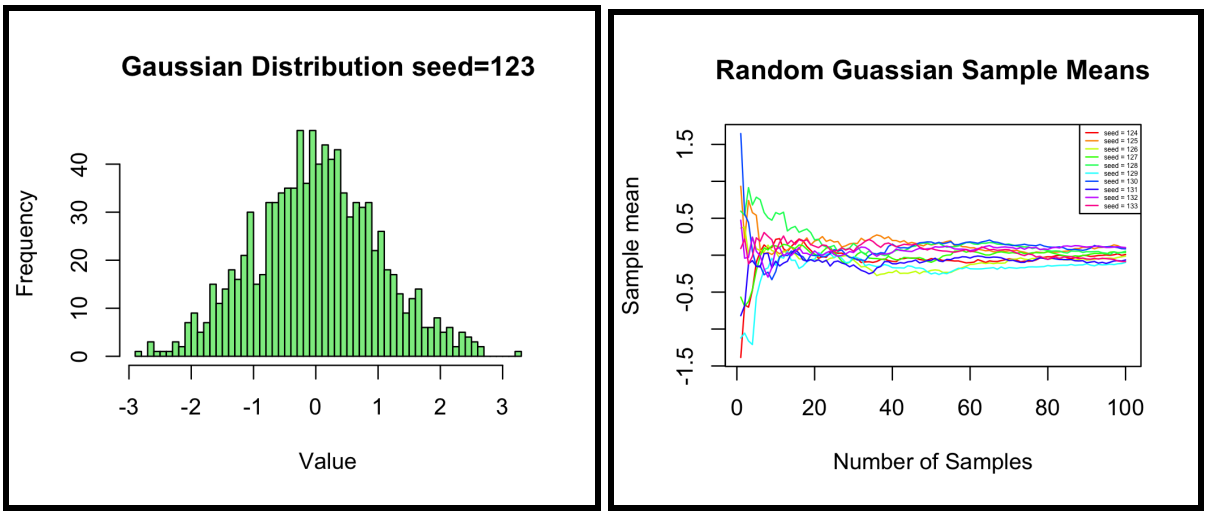

Figure 1: (left) We start by generating a univariate normal distribution, and we can see the "randomness" of the distribution, this will vary with seeds. (right) Next, we can calculate the mean of each seed as a function of increasing sample size.

Results:

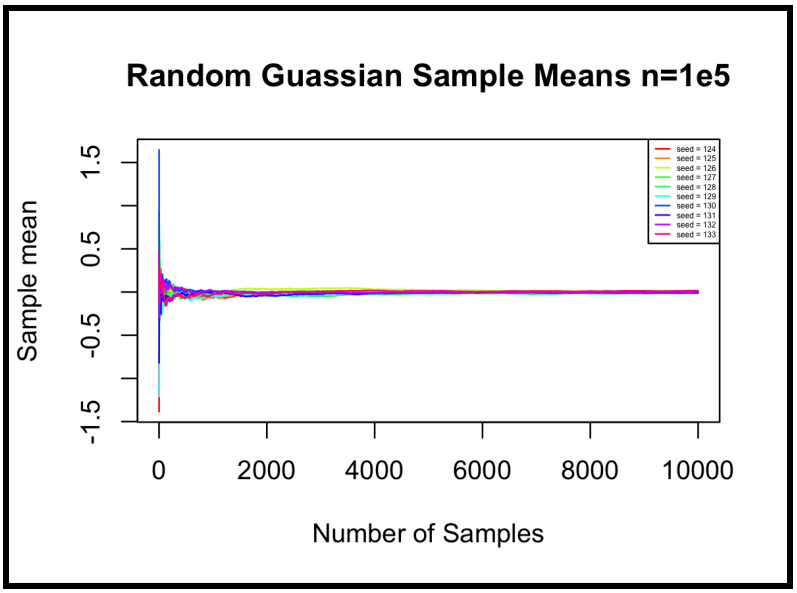

Figure 2: results of sample mean converging using 10 seeds and a sample size of n=1e5.

Then, I experimented, knowing I wanted to converge, and using visual inspection as well as an uncertainty range until my sample mean and 95 percentile within my uncertainty range of ±0.05. This convergence happened at n=10000. This brought me to my choices:

Results Choices:
1. 10 different "random" seeds
2. Sample size of 10,000
3. Seed start at 123

Results Summary: Using a Monte Carlo simulation, I have found that the mean is -0.001, and where the uncertainty is ± 0.015 and the 95 percentile is 1.649 with an uncertainty of ± 0.023. This is using the above assumptions, and I decided upon these by deciding that uncertainty under 0.05 was acceptable, and increasing my sample size until this was achieved. The seed start is trivial, but important for reproducibility. Note that this result run is different from Figure 1, as using n=1 to n=100 for sample size is important to visualize and see that convergence happens with higher sample sizes.

How did you determine convergence?
I determined convergence both visually and by using an uncertainty of under 0.05 as the goal. Once this was achieved through using a sample size of n=10,000 I stopped running my experiment.

Are these analyses reproducible?
These analyses are indeed reproducible, as the code is included in the appendices of this HW submission, as well as in GitHub. I have written the seed # I have started with, so even though this is a Monte Carlo simulation that uses "randomness", you can reproduce this randomness by using the same seed.

Monte Carlo Task 4b: Estimating Pi !

Summary: In this task, I must find pi and the estimated uncertainties using a Monte Carlo approach. Below is my approach, results, assumptions, and citations.

Approach: To find the value of Pi using Monte Carlo, I can use the area of a circle vs. the area of a square using a field of randomly generated points. If we know that the area of a square is 4r^2, and the are of a circle is pi*r^2, then solve for pi: we find the ratio is equal to pi /4. From here, all we need to do is generate a random field of points in a specified 1x1 square, compute the points inside a circle radius, and the points outside a circle radius, multiply by 4 and there we have an estimated value for Pi using monte carlo. See Figure 3 below for a visualization of this technique.

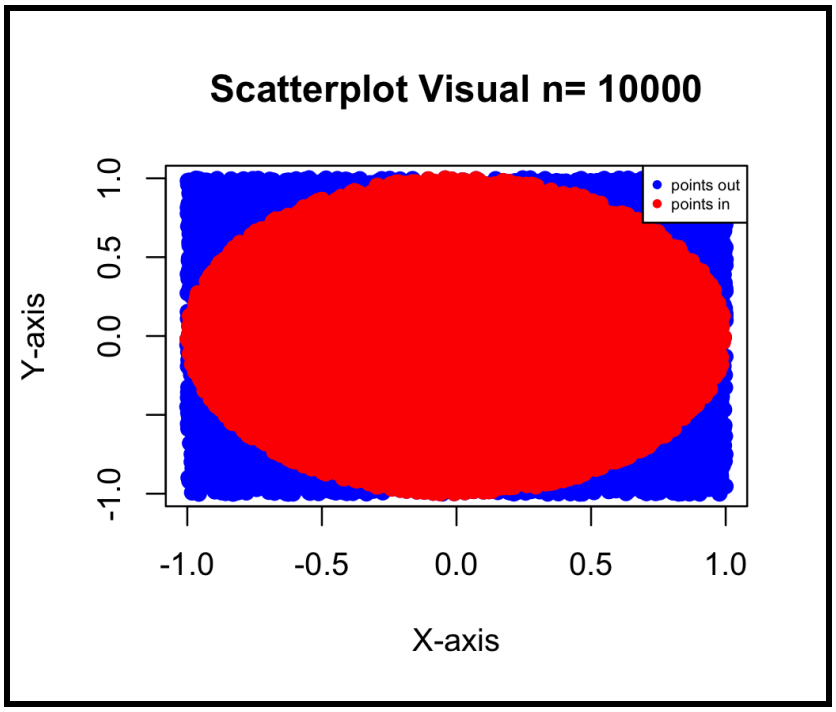

Figure 3: Visualization of 'points in' and 'points out' of the circle, used to estimate Pi using a geometric relationship, using n=10000 points.

Results: I found that Pi = 3.14, with an uncertainty of ±0.01. This converged at n=10,000 samples with 10 seeds. These choices I made by wanting to get within 0.01 uncertainty (which I just barely made) and to see visual convergence. Similar to the first part of Task 4, I again created a function which looped over both seeds and sample size to determine these model runs. As seen in Figure 4 below.

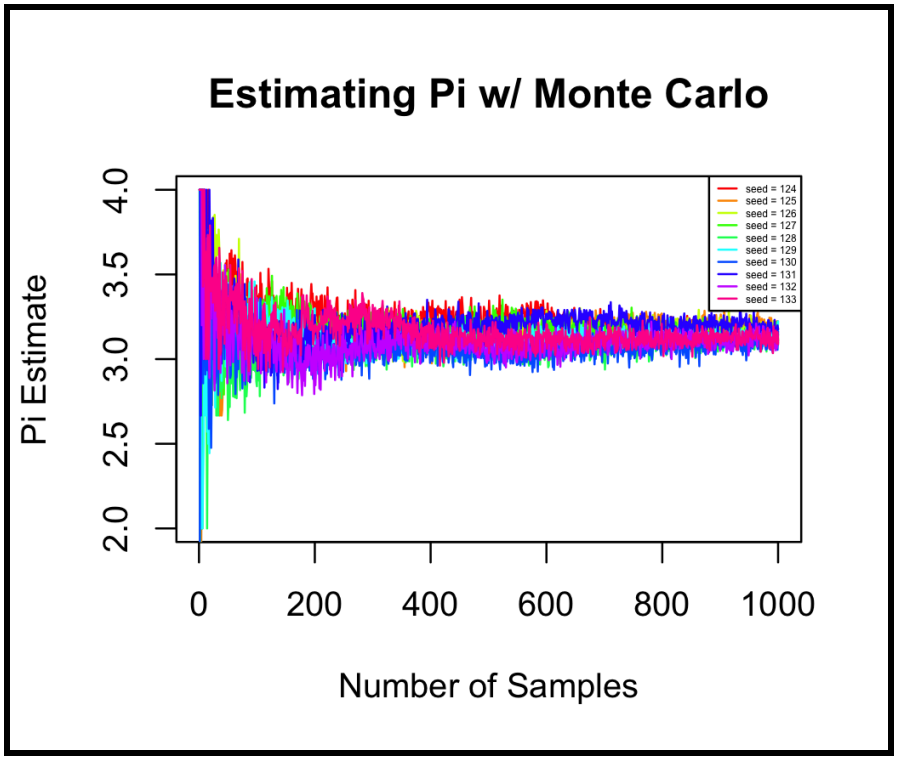

Figure 4: Visualization of convergence for estimating pi with a Monte Carlo methods given different seeds and sample size of random points.

How did you determine convergence?
I determined convergence both visually and by using the uncertainty of ±0.01 for my estimation of Pi. This was found by finding the minimum and maximum estimated value of Pi for the sample size of n=10000 from my 10 different seeds.

Are these analyses reproducible?
Yes, just download and run the code from the GitHub repository, or the files attached to this homework, or the code in the appendix of this .pdf and run to find the reproducible plots and results.

Citations & Sources:
I worked with Maggie O'Shea and Alexis Hudes to talk over the problem set: "what is uncertainty?".

I heavily used this website for R documentation to find functions:
https://www.rdocumentation.org/

Code Appendix:
Find all code in this Github Repository:

```
##############################################
##  file:
##   R Problem Set #2 : Practicing with Monte Carlo
##   - Estimate the mean and p95 of gaussian using monte Carlo methods
##
############################################################
##  authors: Anna Valentine 01/24/2025
##  copyright by the author
##  distributed under the GNU general public license
##  https://www.gnu.org/licenses/gpl.html
##  no warranty (see license details at the link above)

## Ran in R Studio: Version 2024.12.0+467 (2024.12.0+467)
############################################################

### most function

# okay first things first, let's set the mean and the STD
num_samples <- 10^3   # number of random draws (for our initial look)
mu <- 0              # mean of normal distribution to draw from
sigma <- 1           # standard deviation of normal distribution
set.seed(123)       # setting a seed

# and now, let's generate a random guassian:
# reference:
samples <- rnorm(num_samples, mean = mu, sd = sigma)

## we can look at this using a historgram:
hist(samples, breaks = 50, main = "Gaussian Distribution seed=123", xlab = "Value", col = "lightgreen")

## woah! looks pretty terrible, so we must need more samples (this is also a function of binning)

####### finding mean and 95th percentile ###########
test_mean = mean(samples)  # find the mean
#print(test_mean)

test_p95 <- quantile(samples, probs = 0.95)

#########################################################
# now I'm ready to actually do the problem, so I'll make a function I can call with different
# seeds (i), and samples sizes (j), and I can put this all in a nested for loop. I'm sure there
# might be a slicker way to do this, but then I can put it all in a matrix at the end to make
# plotting easier.
#

## turning this into a function:
compute_sample= function(i_val, j_val, seed, mu, sigma){

  sample_mean <- matrix(NA, nrow = length(i_val), ncol = length(j_val))
  sample_p95 <- matrix(NA, nrow = length(i_val), ncol = length(j_val))
```

```
  for (i in i_val){
    for (j in j_val){
      new_seed = seed+i
      set.seed(new_seed)
      samples <- rnorm(j, mean = mu, sd = sigma)
      sample_mean[i, j] = mean(samples)
      sample_p95[i, j] = quantile(samples, probs = 0.95)
    }
  }
  return(list(sample_mean = sample_mean, sample_p95 = sample_p95))
}

## calling my function, we set all the other parameters at the top:
i_val <- 1:10
j_val <- 1:10000
seed = 123

results_test <- compute_sample(i_val, j_val, seed, mu, sigma)
sample_mean_test = results_test$sample_mean

# Plot the sample_mean matrix
matplot(t(sample_mean_test), type = "l", lty = 1, col = rainbow(length(i_val)),
        xlab = "Number of Samples", ylab = "Sample mean",
        main = "Random Guassian Sample Means n=1e5")
legend("topright", legend = paste("seed =", 123 +i_values), col = rainbow(length(i_val)), lty = 1, cex=0.3)

## to quntify our uncertainty, let's look at the largest sample size:
# Find the column corresponding to j = 100000
n = 100
i_val <- 1:10
j_val <- 1:n
seed = 123

results_bigJ <- compute_sample(i_val, j_val, seed, mu, sigma)
sample_mean_bigJ = results_bigJ$sample_mean
sample_p95_bigJ = results_bigJ$sample_p95

# Find the last column in the sample_mean matrix
last_column_mean <- sample_mean_bigJ[, n-1]
last_column_p95 <- sample_p95_bigJ[, n-1]

# Calculate the minimum and maximum for uncertainty in n_samples = 10000
min_value_mean <- min(last_column_mean)
max_value_mean <- max(last_column_mean)
uncertainty_mean <- (max_value_mean - min_value_mean) /2.  ### our uncertainty
avg_value_mean <- mean(last_column_mean) ### our estiamted mean

min_value_p95 <- min(last_column_p95)
max_value_p95 <- max(last_column_p95)
avg_value_p95 <- mean(last_column_p95) #### our estiamted p95 value
uncertainty_p95 <- (max_value_p95 - min_value_p95) / 2 ### our uncertainty
```

```
##############################################
##  file: anna.m.valentine.th.Task4b.ps#2
##   R Problem Set #2 : Practicing with Monte Carlo
##   - Estimate Pi using Monte Carlo Methods
##
############################################################
##  authors: Anna Valentine 01/24/2025
##  copyright by the author
##  distributed under the GNU general public license
##  https://www.gnu.org/licenses/gpl.html
##  no warranty (see license details at the link above)

## Ran in R Studio: Version 2024.12.0+467 (2024.12.0+467)
############################################################

# okay first things first, let's set the mean and the STD
num_samples <- 10^3   # number of random draws
mu <- 0               # mean of normal distribution to draw from
sigma <- 1            # standard deviation of normal distribution

# and now, let's generate a guassian:
# reference:
# https://numpy.org/doc/2.1/reference/random/generated/numpy.random.normal.html
# first set a seed:
set.seed(123)
n = 10000
#### create a grid from -1 to 1
x = runif(n, min=-1, max=1)
y = runif(n, min=-1, max=1)

### set up vectors to sort them out:
x_in = c()
y_in = c()
x_out = c()
y_out = c()
### calculate which are in a circle: (x^2 + y^2 = 1)
for (i in seq_along(x)){ #since x and y are same length, we can use this:
  distance = (x[i])^2 + (y[i])^2
  #print(distance)
  if (distance < 1){
    x_in = c(x_in, x[i])
    y_in = c(y_in, y[i])
  }else{
    x_out = c(x_out, x[i])
    y_out = c(y_out, y[i])
  }
}

#### estimating pi here:
pi_est = 4*length(x_in) / length(x)
print(pi_est)

#### plot them on a grid:
### add a legend !
title <- paste("Scatterplot Visual n=", n)
plot(x, y, main = title, xlab = "X-axis", ylab = "Y-axis", pch = 19, col = "blue")
```

```
points(x_in, y_in, col="red", pch=19)
legend("topright", legend = c("points out", "points in"), col = c("blue", "red"), pch = 19, cex=0.5)

#### okay so now I will make a function, that computes pi for diff seeds and sample sizes:
i_val <- 1:10  # number of seeds to iterate over
j_val <- 1:1000  # number of samples
seed = 123

compute_pi= function(i_val, j_val, seed){

  pi_est <- matrix(NA, nrow = length(i_val), ncol = length(j_val))

  for (i in i_val){
    for (j in j_val){
      new_seed = seed+i
      set.seed(new_seed)
      #### create a grid from -1 to 1
      x = runif(j, min=-1, max=1)
      y = runif(j, min=-1, max=1)

      ### set up vectors to sort them out:
      x_in = c()
      y_in = c()
      x_out = c()
      y_out = c()
      ### calculate which are in a circle: (x^2 + y^2 = 1)
      for (k in seq_along(x)){ #since x and y are same length, we can use this:
        distance = (x[k])^2 + (y[k])^2
        #print(distance)
        if (distance < 1){
          x_in = c(x_in, x[k])
          y_in = c(y_in, y[k])
        }else{
          x_out = c(x_out, x[k])
          y_out = c(y_out, y[k])
        }
      }

      #### estimating pi here:
      pi = 4*length(x_in) / length(x)
      pi_est[i, j] = pi
    }
  }
  return(pi_est)
}

pi = compute_pi(i_val, j_val, seed)

###### and plot them:
# Plot the sample_mean matrix
matplot(t(pi), type = "l", lty = 1, col = rainbow(length(i_val)),
        xlab = "Number of Samples", ylab = "Pi Estimate",
        main = "Estimating Pi w/ Monte Carlo", ylim=c(2, 4))
legend("topright", legend = paste("seed =", 123 +i_values), col = rainbow(length(i_val)), lty = 1, cex=0.3)
```

```
#### I need a function to test with just num_samples and 10 different seeds:
#### okay so now I will make a function, that computes pi for diff seeds and sample sizes:
i_val <- 1:10  # number of seeds to iterate over
seed = 123

compute_pi_n= function(i_val, n, seed){

  pi_est_n <- matrix(NA, nrow = length(i_val), ncol = 1)

  for (i in i_val){
      new_seed = seed+i
      set.seed(new_seed)
      #### create a grid from -1 to 1
      x = runif(n, min=-1, max=1)
      y = runif(n, min=-1, max=1)

      ### set up vectors to sort them out:
      x_in = c()
      y_in = c()
      x_out = c()
      y_out = c()
      ### calculate which are in a circle: (x^2 + y^2 = 1)
      for (k in seq_along(x)){ #since x and y are same length, we can use this:
        distance = (x[k])^2 + (y[k])^2
        #print(distance)
        if (distance < 1){
          x_in = c(x_in, x[k])
          y_in = c(y_in, y[k])
        }else{
          x_out = c(x_out, x[k])
          y_out = c(y_out, y[k])
        }
      }

      #### estimating pi here:
      pi = 4*length(x_in) / length(x)
      pi_est[i] = pi
    }
  return(pi_est)
}

#### now we can run and just look for the right number of samples:
pi_n = compute_pi_n(i_val, 10000, seed)

min_value <- min(pi_n)
max_value <- max(pi_n)
mean_value <- mean(pi_n)

range = max_value - min_value / 2
```