Assignment 0

Timothy Lee 1003906231

January 20, 2020

The primary goal of this assignment is to allow you to practice and assess your prerequisite knowledge which will be relied on throughout this course. The secondary objective is to familiarize you with the tools and best practices, including:

- Mathematical typesetting (LaTeX)
- Version control (git and github)
- Unit testing
- Setting random seeds for reproducibility
- Automatic differentiation

The starter code and examples below are in the Julia programming language. You may also submit solutions using Python, if that is more familiar for you.

You are expected to submit a typeset (LaTeX) **write-up** (pdf) that contains everything that will be assessed. In particular, this means your writeup must include

- Important source code. If a question asks you to implement a piece of code, include it in the writeup. Make this clear for the marker, don't just append your entire source code into the pdf.
- Outputs from the code. If a question asks you to report some values, those must be included in the writeup.
- Plots must be included in the writeup, and be clearly labelled (title, axes, legend, caption).
- Unit tests. For some questions where you implement a piece of code, you will be expected to test the correctness of that code. Include your unit tests in your writeup.

You will also be expected to include **all source code** along with your writeup. However, graders will not be expected to run your source code.

Questions where you are asked to run unit tests may require you to produce the unit test. For example, in question 2.1 you will manually write the derivates for various functions. In question 2.2 you will use Automatic Differentiation to compute derivatives of those same functions. You will test the correctness of these answers by producing unit tests for each

question. This is a very useful practice because it's possible that either your code or your math may be incorrect, but it's much less likely (still possible) that both are incorrect for the same reasons!

If you are using the Julia starter code I have included all the packages you will need in the repo. You can activate those packages in the command-line by starting the julia session with `julia --project` or if you are already in the REPL (like in Atom) by opening the package manager (by typing `[` into the REPL) and activating the project `[ activate .` (the period is part of the command). If you've done this correctly, when you open the Package manager (type `]`) you should see `(assignment_0) pkg>`.

This document is as example of literate programming, which weaves together text (markdown), math (LaTeX), and code (julia) from a single document. The source for this write-up can be found in `A0.jmd` and can be produced using `make_pdf.jl`. You may use this to produce your own writeups, but this is not required. Feel free to use LaTeX as normal, and include the relevant source code, outputs, and plots.

```
# We will use unit testing to make sure our solutions are what we expect
# This shows how to import the Test package, which provides convenient functions like
@test
using Test
# Setting a Random Seed is good practice so our code is consistent between runs
using Random # Import Random Package
Random.seed!(414); #Set Random Seed
# ; suppresses output, makes the writeup slightly cleaner.
# ! is a julia convention to indicate the function mutates a global state.
```

1 Probability

1.1 Variance and Covariance

Let X and Y be two continuous, independent random variables.

1. [3pts] Starting from the definition of independence, show that the independence of X and Y implies that their covariance is 0.

Answer: By definition of the expectation of continuous, independent random variables,

$$
\begin{aligned}
E(XY) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyf_{xy}(x,y)dxdy && (1)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyf_{x}(x)f_{y}(y)dxdy && (2)\\
&= (\int_{-\infty}^{\infty} xf_{x}(x)dx)(\int_{-\infty}^{\infty} yf_{y}(y)dy) && (3)\\
&= E(X)E(Y) && (4)\\
& && (5)
\end{aligned}
$$

Hence, covariance could be written as follows,

$$Cov(X,Y) = E(XY) - E(X)E(Y) \tag{6}$$
$$= E(X)E(Y) - E(X)E(Y) \text{ since X, Y are independent} \tag{7}$$
$$= 0 \tag{8}$$

2. [3pts] For a scalar constant a, show the following two properties starting from the definition of expectation:

$$\mathbb{E}(X + aY) = \mathbb{E}(X) + a\mathbb{E}(Y) \tag{9}$$
$$\text{var}(X + aY) = \text{var}(X) + a^2\text{var}(Y) \tag{10}$$

Answer:

$$\mathbb{E}(X + aY) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(X + aY)f_{X,Y}(x,y)dxdy \tag{11}$$
$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}Xf_{X,Y}(x,y)dxdy + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}aYf_{X,Y}(x,y)dxdy \tag{12}$$
$$= \int_{-\infty}^{\infty}X(\int_{-\infty}^{\infty}f_{X,Y}(x,y)dy)dx + \int_{-\infty}^{\infty}aY(\int_{-\infty}^{\infty}f_{X,Y}(x,y)dx)dy \tag{13}$$
$$= \int_{-\infty}^{\infty}Xf_X(x)dx + \int_{-\infty}^{\infty}aYf_Y(y)dy \tag{14}$$
$$= \mathbb{E}(X) + a\mathbb{E}(Y) \tag{15}$$
$$\tag{16}$$
$$\tag{17}$$
$$\text{var}(X + aY) = \mathbb{E}[((X + aY) - \mathbb{E}[X + aY])^2] \tag{18}$$
$$= \mathbb{E}[(X + aY - (\mu_x + a\mu_y))^2] \text{ proven previously} \tag{19}$$
$$= \mathbb{E}[((X - \mu_x) + (aY - a\mu_y))^2] \tag{20}$$
$$= \mathbb{E}[((X - \mu_x) + a(Y - \mu_y))^2] \tag{21}$$
$$= \mathbb{E}[(X - \mu_x)^2 + 2(X - \mu_x)a(Y - \mu_y) + a^2(Y - \mu_y)^2] \tag{22}$$
$$= \mathbb{E}[(X - \mu_x)^2] + \mathbb{E}[2a(X - \mu_x)(Y - \mu_y)] + \mathbb{E}[a^2(Y - \mu_y)^2] \tag{23}$$
$$= \text{var}[X] + 2aCov(X,Y) + a^2\text{var}[Y] \tag{24}$$
$$= \text{var}[X] + a^2\text{var}[Y] \text{ since X, Y are independent, covariance is 0} \tag{25}$$

1.2 1D Gaussian Densities

1. [1pts] Can a probability density function (pdf) ever take values greater than 1?

Answer: Yes, since a pdf is not a probability, it only has to satisfy the conditions that it is non-negative and its area/integral is equal to one.

2. Let X be a univariate random variable distributed according to a Gaussian distribution with mean μ and variance σ^2.

[1pts] Write the expression for the pdf:

Answer:

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2) \tag{26}$$

[2pts] Write the code for the function that computes the pdf at x with default values $\mu = 0$ and $\sigma = \sqrt{0.01}$:

Answer:

```
function gaussian_pdf(x; mean=0., variance=0.01)
  return 1/(sqrt(2*pi)*sqrt(variance))* exp(-0.5*((x-mean)/sqrt(variance))^2)
end
```

```
gaussian_pdf (generic function with 1 method)
```

Test your implementation against a standard implementation from a library:

```
# Test answers
using Test
using Distributions: pdf, Normal # Note Normal uses N(mean, stddev) for parameters
@testset "Implementation of Gaussian pdf" begin
  x = randn()
  @test gaussian_pdf(x) ≈ pdf.(Normal(0.,sqrt(0.01)),x)
  # ≈ is syntax sugar for isapprox, typed with `\approx <TAB>`
  # or use the full function, like below
  @test isapprox(gaussian_pdf(x,mean=10., variance=1) , pdf.(Normal(10., sqrt(1)),x))
end;
```

```
Test Summary:                   | Pass  Total
Implementation of Gaussian pdf |    2      2
```

3. [1pts] What is the value of the pdf at $x = 0$? What is probability that $x = 0$ (hint: is this the same as the pdf? Briefly explain your answer.)

Answer: The value of the pdf is:

```
using Distributions: pdf, Normal
pdf.(Normal(10., sqrt(1)), 0)
```

```
7.69459862670642e-23
```

However, the probability that X = 0 is 0. Since we are working with a continuous random variable, the probability of X being a single point on the pdf is exactly 0. We can only interpret the pdf value (we get by plugging in X into the pdf function) as the likelihood and not a probability.

4. A Gaussian with mean μ and variance σ^2 can be written as a simple transformation of the standard Gaussian with mean 0. and variance 1..

[1pts] Write the transformation that takes $x \sim \mathcal{N}(0., 1.)$ to $z \sim \mathcal{N}(\mu, \sigma^2)$:

Answer:

$$z = x\sigma + \mu$$

[2pts] Write a code implementation to produce n independent samples from $\mathcal{N}(\mu, \sigma^2)$ by transforming n samples from $\mathcal{N}(0., 1.)$.

Answer

```
using Distributions: pdf, Normal
function sample_gaussian(n; mean=0., variance=0.01)
  # n samples from standard gaussian
  x = rand(Normal(0., 1.), n)

  # TODO: transform x to sample z from N(mean,variance)
  z =  (x .* sqrt(variance)) .+ mean
  return z
end;
```

[2pts] Test your implementation by computing statistics on the samples:

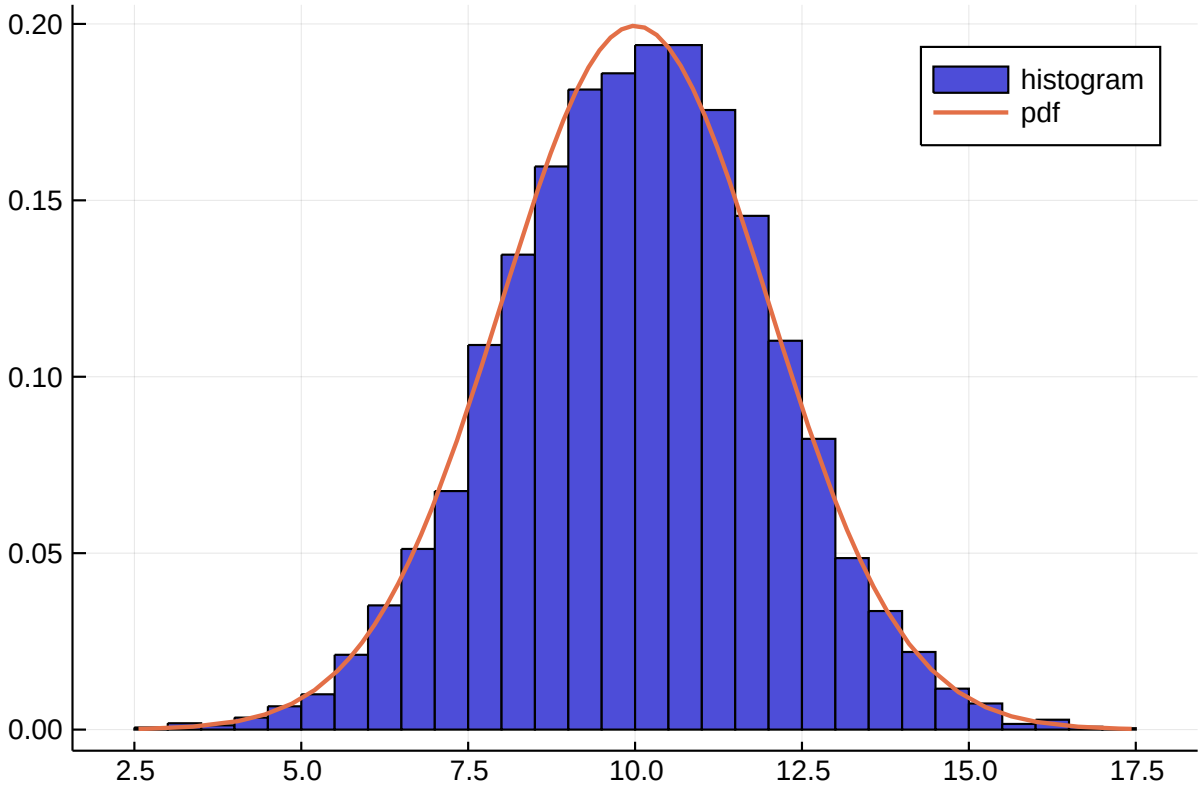
```
using Statistics: mean, var
using Test
@testset "Numerically testing Gaussian Sample Statistics" begin
  #TODO: Sample 100000 samples with your function and use mean and var to
  # compute statistics.
  # tests should compare statistics against the true mean and variance from arguments.
  # hint: use isapprox with keyword argument atol=1e-2
  @test isapprox(mean(sample_gaussian(100000; mean=0., variance=0.01)) , 0., atol=1e-2)
  @test isapprox(var(sample_gaussian(100000; mean=0., variance=0.01))  , 0.01, atol=1e-2)
end
```

```
Test Summary:                                  | Pass  Total
Numerically testing Gaussian Sample Statistics |    2      2
Test.DefaultTestSet("Numerically testing Gaussian Sample Statistics", Any[]
, 2, false)
```

5. [3pts] Sample 10000 samples from a Gaussian with mean 10. an variance 2. Plot the **normalized** `histogram` of these samples. On the same axes `plot!` the pdf of this distribution.

Confirm that the histogram approximates the pdf. (Note: with `Plots.jl` the function `plot!` will add to the existing axes.)

```
using Plots
using Distributions
using StatsPlots
x = rand(Normal(10., 2.), 10000)
histogram(x, normalize=true, label="histogram", color = :lighttest)
plot!(Normal(10.,2), lw=2, label="pdf")
```



2 Calculus

2.1 Manual Differentiation

Let $x, y \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and square matrix $B \in \mathbb{R}^{m \times m}$. And where x' is the transpose of x. Answer the following questions in vector notation.

1. [1pts] What is the gradient of $x'y$ with respect to x?

Answer:

$$x'y = x_1 \cdot y_1 + x_2 \cdot y_2 + ... + x_m \cdot y_m \tag{27}$$

$$= \sum_{i=1}^{m} x_i \cdot y_i \tag{28}$$

$$\frac{\partial \sum_{i=1}^{m} x_i \cdot y_i}{\partial x_i} = y_i \tag{29}$$

$$\frac{\partial x'y}{\partial x} = y \tag{30}$$

$$\tag{31}$$

2. [1pts] What is the gradient of $x'x$ with respect to x?

Answer:

$$x'x = {x_1}^2 + {x_2}^2 + ...{x_m}^2 \tag{32}$$
$$= \sum_{i=1}^{m} {x_i}^2 \tag{33}$$
$$\frac{\partial \sum_{i=1}^{m} {x_i}^2}{\partial x_i} = 2x_i \tag{34}$$
$$\frac{\partial x'x}{\partial x} = 2x \tag{35}$$
$$\tag{36}$$

3. [2pts] What is the Jacobian of $x'A$ with respect to x?

Answer:

$$x'A = [(x_1 \cdot a_{11}) + (x_2 \cdot a_{21}) + ... + (x_m \cdot a_{m1}) \tag{37}$$
$$(x_1 \cdot a_{12}) + (x_2 \cdot a_{22}) + ... + (x_m \cdot a_{m2}) \tag{38}$$
$$... \tag{39}$$
$$(x_1 \cdot a_{1n}) + (x_2 \cdot a_{2n}) + ... + (x_m \cdot a_{mn})] \tag{40}$$
$$= J_{n \times m} \tag{41}$$
$$= \begin{bmatrix} \frac{\partial \sum_{i=1}^{m} x_i \cdot a_i 1}{\partial x_1} & ... & \frac{\partial \sum_{i=1}^{m} x_i \cdot a_i 1}{\partial x_m} \\ & \vdots & \\ \frac{\partial \sum_{i=1}^{m} x_i \cdot a_i n}{\partial x_1} & ... & \frac{\partial \sum_{i=1}^{m} x_i \cdot a_i n}{\partial x_m} \end{bmatrix} \tag{42}$$
$$= \begin{bmatrix} a_{11} & ... & a_{m1} \\ & \vdots & \\ a_{1n} & ... & a_{mn} \end{bmatrix} \tag{43}$$
$$= {A'}_{n \times m} \tag{44}$$

4. [2pts] What is the gradient of $x'Bx$ with respect to x?

Answer:

$$
\begin{aligned}
x'B &= [x_1 \cdot b_{11} + x_2 \cdot b_{21} + x_m \cdot b_{m1} && (45)\\
&\quad x_1 \cdot b_{12} + x_2 \cdot b_{22} + x_m \cdot b_{m2} && (46)\\
&\quad \ldots && (47)\\
&\quad x_1 \cdot b_{1m} + x_2 \cdot b_{2m} + x_m \cdot b_{mm}] && (48)\\
x'Bx &= (x_1 \cdot b_{11} + x_2 \cdot b_{21} + x_m \cdot b_{m1}) \cdot x_1 && (49)\\
&\quad + (x_1 \cdot b_{12} + x_2 \cdot b_{22} + x_m \cdot b_{m2}) \cdot x_2 && (50)\\
&\quad + \ldots && (51)\\
&\quad + (x_1 \cdot b_{1m} + x_2 \cdot b_{2m} + x_m \cdot b_{mm}) \cdot x_m && (52)\\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} x_i b_{ij} x_j && (53)\\
&= \sum_{i=1}^{m}(b_{ii}{x_i}^2 + \sum_{j\neq i} x_i b_{ij} x_j) && (54)\\
\frac{\partial \sum_{i=1}^{m}\sum_{j=1}^{m} x_i b_{ij} x_j}{\partial x_k} &= \sum_{j=1}^{m} x_j b_{jk} + \sum_{j=1}^{m} b_{kj} x_j && (55)\\
\frac{\partial \sum_{i=1}^{m}\sum_{j=1}^{m} x_i b_{ij} x_j}{\partial x} &= \left(\begin{bmatrix} \sum_{j=1}^{m} x_j b_{j1} \\ \vdots \\ \sum_{j=1}^{m} x_j b_{jm} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{m} b_{1j} x_j \\ \vdots \\ \sum_{j=1}^{m} b_{mj} x_j \end{bmatrix}\right) && (56)\\
&= B^T x + Bx && (57)\\
&= (B^T + B)x && (58)
\end{aligned}
$$

2.2 Automatic Differentiation (AD)

Use one of the accepted AD library (Zygote.jl (julia), JAX (python), PyTorch (python)) to implement and test your answers above.

2.2.1 [1pts] Create Toy Data

```
using Zygote
# Choose dimensions of toy data
m = 2 #TODO
n = 3 #TODO

# Make random toy data with correct dimensions
x = rand(m)#m
y = rand(m)#m
A = rand(m,n)#mxn
B = rand(m,m)#mxm
```

```
2×2 Array{Float64,2}:
 0.990607  0.000834152
 0.966344  0.692019
```

[1pts] Test to confirm that the sizes of your data is what you expect:

```
# Make sure your toy data is the size you expect!
using Test
```

```
@testset "Sizes of Toy Data" begin
  #TODO: confirm sizes for toy data x,y,A,B
  #hint: use `size` function, which returns tuple of integers.
  @test size(x) == (2,)
  @test size(y) == (2,)
  @test size(A) == (2,3)
  @test size(B) == (2,2)
end;
```

```
Test Summary:     | Pass  Total
Sizes of Toy Data |    4      4
```

2.2.2 Automatic Differentiation

1. [1pts] Compute the gradient of $f_1(x) = x'y$ with respect to x?

```
# Use AD Tool
using Zygote: gradient
# note: `Zygote.gradient` returns a tuple of gradients, one for each argument.
# if you want just the first element you will need to index into the tuple with [1]

f1(x) = (transpose(x)* y)
df1dx = gradient(f1, x)[1]
```

```
2-element Array{Float64,1}:
 0.3847709557615704
 0.5958271795600607
```

2. [1pts] Compute the gradient of $f_2(x) = x'x$ with respect to x?

```
f2(x) = (transpose(x)* x)
df2dx = gradient(f2, x)[1]
```

```
2-element Array{Float64,1}:
 1.4749728062951584
 1.3123269338085213
```

3. [1pts] Compute the Jacobian of $f_3(x) = x'A$ with respect to x?

If you try the usual `gradient` fucntion to compute the whole Jacobian it would give an error. You can use the following code to compute the Jacobian instead.

```
function jacobian(f, x)
    y = f(x)
    n = length(y)
    m = length(x)
    T = eltype(y)
    j = Array{T, 2}(undef, n, m)
    for i in 1:n
        j[i, :] .= gradient(x -> f(x)[i], x)[1]
    end
    return j
end
f3(x) = (transpose(x)*A)
jacobian(f3, x)
```

```
3×2 Array{Float64,2}:
 0.923724  0.0490162
 0.166053  0.767333
 0.495712  0.15528
```

[2pts] Briefly, explain why `gradient` of f_3 is not well defined (hint: what is the dimensionality of the output?) and what the `jacobian` function is doing in terms of calls to `gradient`. Specifically, how many calls of `gradient` is required to compute a whole `jacobian` for $f : \mathbb{R}^m \to \mathbb{R}^n$?

Answer: The gradient of f_3 is not well defined because the resulting dimensions of the output should be $n \times m$.

The very important takeaway here is that, with AD, `gradient`s are cheap but full `jacobian`s are expensive.

```
f3(x) = transpose(x)*A
df3dx = jacobian(f3, x)# use jacobian
```

```
3×2 Array{Float64,2}:
 0.923724  0.0490162
 0.166053  0.767333
 0.495712  0.15528
```

4. [1pts] Compute the gradient of $f_4(x) = x'Bx$ with respect to x?

```
f4(x) = transpose(x) * B * x
df4dx = gradient(f4, x)[1]
```

```
2-element Array{Float64,1}:
 2.095744836162754
 1.62143546360265
```

5. [2pts] Test all your implementations against the manually derived derivatives in previous question

```
# Test to confirm that AD matches hand-derived gradients
@testset "AD matches hand-derived gradients" begin
  @test df1dx == y
  @test df2dx == 2 .* x
  @test df3dx == transpose(A)
  @test isapprox(df4dx, (transpose(B) + B) * x)   #used is approx since there is a very
small decimal place mismatched
end
```

```
Test Summary:                      | Pass  Total
AD matches hand-derived gradients |    4      4
Test.DefaultTestSet("AD matches hand-derived gradients", Any[], 4, false)
```