Assignment 0

Timothy Lee 1003906231

January 16, 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:

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:

[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 #TODO: implement pdf at x
end
```

Test your implementation against a standard implementation from a library:

```
# Test answers
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;
```

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

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

[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

```
function sample_gaussian(n; mean=0., variance=0.01)
  # n samples from standard gaussian
  x = #TODO

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

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

```julia
using Statistics: mean, var
@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
end;
```

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.)

```julia
using Plots

#histogram(#TODO)
#plot!(#TODO)
```

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:

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

Answer:

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

Answer:

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

Answer:

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

```
# Choose dimensions of toy data
m = #TODO
n = #TODO

# Make random toy data with correct dimensions
x = #TODO
y = #TODO
A = #TODO
B = #TODO
```

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

```
# Make sure your toy data is the size you expect!
@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.
end;
```

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) = #TODO
df1dx = #TODO
```

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

```
f2(x) = #TODO
df2dx = #TODO
```

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
```

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

```
f3(x) = #TODO
df3dx = #TODO: use jacobian
```

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

```
f4(x) = #TODO
df4dx = #TODO
```

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 == #TODO: rhs from 2.1
  @test df2dx == #TODO: rhs from 2.1
  @test df3dx == #TODO: rhs from 2.1
  @test df4dx == #TODO: rhs from 2.1
end
```