STA414 Assignment 0

Hongbo Du (1003568089)

January 23, 2020

```julia
# 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:

$$\begin{aligned}\mathrm{Cov}(X,Y) &= \mathbb{E}[(X-\mathbb{E}(X))(Y-\mathbb{E}(Y))] \qquad \text{(by definition)}\\ &= \mathbb{E}[(XY) - Y\mathbb{E}(X) - X\mathbb{E}(Y) + \mathbb{E}(X)\mathbb{E}(Y)]\\ &= \mathbb{E}(X)\mathbb{E}(Y) - \mathbb{E}(Y)\mathbb{E}(X) - \mathbb{E}(X)\mathbb{E}(Y) + \mathbb{E}(X)\mathbb{E}(Y) \qquad \text{(by independence)}\\ &= 0\end{aligned}$$

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

$$\mathbb{E}(X+aY) = \mathbb{E}(X) + a\mathbb{E}(Y) \tag{1}$$
$$\mathrm{var}(X+aY) = \mathrm{var}(X) + a^2\mathrm{var}(Y) \tag{2}$$

   Answer:

$$\begin{aligned}\mathbb{E}(X+aY) &= \mathbb{E}(X) + \mathbb{E}(aY) \qquad \text{(by independence)}\\ &= \mathbb{E}(X) + a\mathbb{E}(Y)\end{aligned}$$

$$
\begin{aligned}
\text{var}(X + aY) &= \mathbb{E}[(X + aY)^2] - [\mathbb{E}(X + aY)]^2 \qquad \text{(by definition)} \\
&= \mathbb{E}[X^2 + 2aXY + a^2Y^2] - [\mathbb{E}(X)]^2 - 2a\mathbb{E}(X)\mathbb{E}(Y) - a^2[\mathbb{E}(Y)]^2 \\
&= \mathbb{E}(X^2) - [\mathbb{E}(X)]^2 + a^2\mathbb{E}(Y^2) - a^2[\mathbb{E}(Y)]^2 \\
&= \text{var}(X) + a^2\text{var}(Y)
\end{aligned}
$$

1.2 1D Gaussian Densities

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

   Answer: Yes, as long as the input value is finitely non-negative.

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:
   $$f_X(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 \right\}$$

   [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)
  sd = sqrt(variance)
  pdf = (1)/(sd * sqrt(2 * pi)) * exp(-0.5 * ((x - mean)/(sd))^2)
  return pdf
end
```

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

   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;
```

```
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 at $x = 0$ is a finite and non-negative number, and the probability of $x = 0$ is 0. These two are not the same.

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: Consider the transformation $x = \frac{z-\mu}{\sigma}$ which takes z to x. Thus, the transformation $z = x\sigma + \mu$ takes x to z.

[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; meam=0., variance=0.01)
  # n samples from standard gaussian
  x = rand(Normal(0, 1), n)

  # transform x to sample z from N(mean,variance)
  mu = transpose(zeros(true, length(x)))
  z = x * sqrt(variance) + mu
  return z
end;
```

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

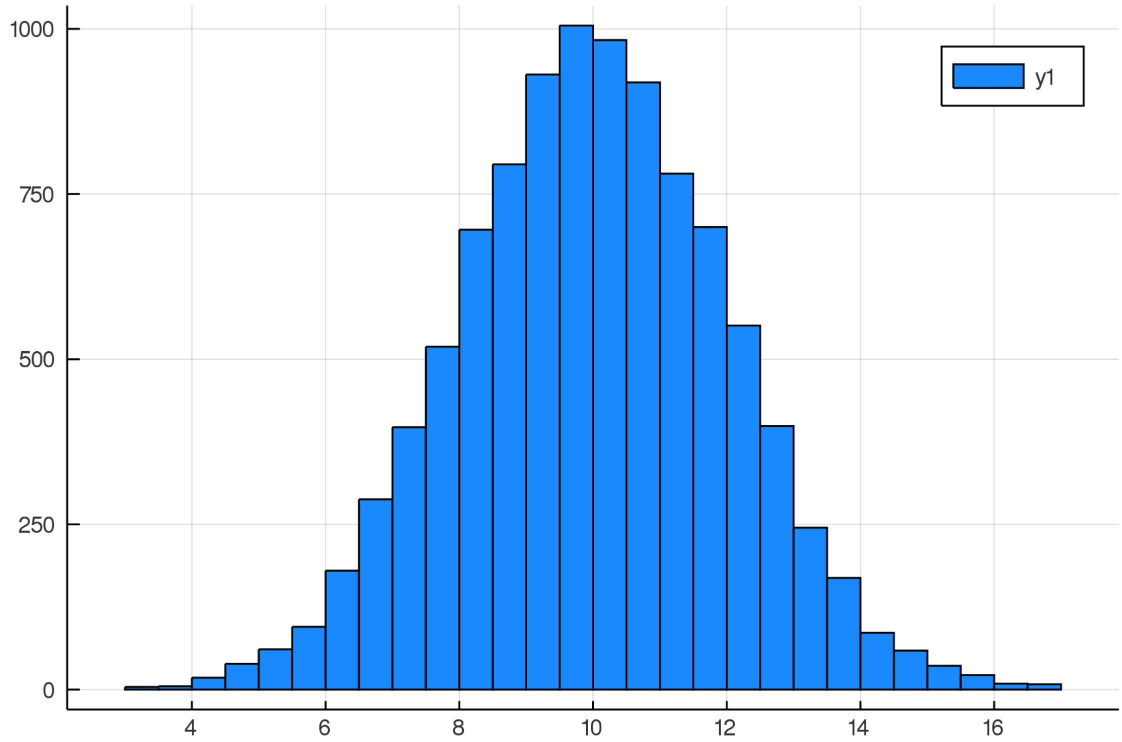
```
using Statistics: mean, var
@testset "Numerically testing Gaussian Sample Statistics" begin
  #TODO: Sample 100000 samples with your function and use mean and var to
  # compute statistics.
  # tests should compare statistics against the true mean and variance from
arguments.
  # hint: use isapprox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

Test Summary:                                  | Pass  Total
Numerically testing Gaussian Sample Statistics |    2      2
```

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.)

   Answer:

```
using Plots
hist = histogram(rand(Normal(10, 2), 10000))
plot!(hist)
```



According to the bell-shaped plot, we can conclude that the histogram approximates the pdf of normal distribution with mean of 10 and variance of 2.

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:

$$\nabla_{\boldsymbol{x}} \boldsymbol{x}'\boldsymbol{y} = \frac{\partial}{\partial x}\left[\sum_{i=1}^{m} x_i y_i\right] = \boldsymbol{y}$$

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

Answer:

$$\nabla_{\boldsymbol{x}} \boldsymbol{x}'\boldsymbol{x} = \frac{\partial}{\partial x}\left[\sum_{i=1}^{m} x_i x_i\right] = \frac{\partial}{\partial x}\left[\sum_{i=1}^{m} x_i^2\right] = 2\boldsymbol{x}$$

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

Answer:

$$
\begin{aligned}
\boldsymbol{J_x x'}A &= \frac{\partial}{\partial x}\left(\sum_{i=1}^{m} x_i a_{i1} \cdots \sum_{i=1}^{m} x_i a_{in}\right) \\
&= \begin{pmatrix} \frac{\partial}{\partial x_1}[\sum_{i=1}^m x_i a_{i1}] & \dots & \frac{\partial}{\partial x_m}[\sum_{i=1}^m x_i a_{i1}] \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1}[\sum_{i=1}^m x_i a_{in}] & \dots & \frac{\partial}{\partial x_m}[\sum_{i=1}^m x_i a_{in}] \end{pmatrix} \\
&= \begin{pmatrix} a_{11} & \dots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \dots & a_{mn} \end{pmatrix} \\
&= A'
\end{aligned}
$$

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

Answer:

$$
\begin{aligned}
\nabla_{\boldsymbol{x}} \boldsymbol{x}' B \boldsymbol{x} &= \frac{\partial}{\partial x_k}\left[x_1 \sum_{i=1}^{m} b_{i1} x_i + \cdots + x_m \sum_{i=1}^{m} b_{im} x_i\right] \\
&= \frac{\partial}{\partial x_k}\left[x_1 b_{k1} x_k + \cdots + x_m b_{km} x_k\right] + \frac{\partial}{\partial x_k}\left[x_k \sum_{i=1}^{m} b_{ik} x_i\right] \\
&= \sum_{j=1}^{m} x_j b_{kj} + \sum_{i=1}^{m} b_{ik} x_i \\
&= B\boldsymbol{x} + B'\boldsymbol{x} \\
&= (B + B')\boldsymbol{x}
\end{aligned}
$$

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

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

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

```
3×3 Array{Int64,2}:
  2082611162394249223  -1584805259920528499   3312359338613220565
 -1719917214949896604  -7684552650385197880  -6714780201888608491
  6844828970218836293    -19336610032898351   4478452655413003417
```

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

```julia
# 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.
  @test isapprox(size(x, 1), m)
  @test isapprox(size(y, 1), m)
  @test isapprox(size(A, 1), m)
  @test isapprox(size(A, 2), n)
  @test isapprox(size(B, 1), m)
  @test isapprox(size(B, 1), m)
end;
```

```
Test Summary:      | Pass  Total
Sizes of Toy Data |    6      6
```

2.2.2 Automatic Differentiation

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

```julia
# 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(x -> f1(x), x)[1]
```

```
3-element Array{Int64,1}:
  6835238219692564448
  -381083529860314721
 -7858831951335573680
```

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

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

```
3-element Array{Int64,1}:
 -8270442776141894610
  -605331187883253270
 -1523442933334844300
```

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.

```julia
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
```

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

[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 dimensionality of the output of the `jacobian` function is n. The `jacobian` calls $m * n$ times of computing the partial derivatives to compute the whole Jacobian; while the `gradient` calls m times partial derivatives to calculate the gradient. Thus the output of the `jacobian` with respect to one vector is a matrix of $n \times m$, and the output of `gradient` with respect to one vector is a vector or matrix of $m \times 1$.

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) #using jacobian

2×3 Array{Int64,2}:
  4408026600522538309  -6232865938377085614  1349691602347355064
 -2873661868152111795   7436537824285171979  3974329157710246768
```

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

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

3-element Array{Int64,1}:
 -6604381366815181109
  -603149473546851437
  -897390866673629496
```

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 df4dx == (B + transpose(B)) * x
end

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