Problemas de APA

Problema grupal número 4

Xenia Calisalvo
Javier Castaño
Ignasi Fibla
Mark Smithson

Departamento de Computer Science

9 de octubre de 2022

Índice

1. Enunciado: Lo nuclear ya es verde 1

2. Apartado a) 2

3. Apartado b) 3

4. Apartado c) 4

5. Apartado d) 5

6. Apartado e) 6

7. Apartado f) 8

1. Enunciado: Lo nuclear ya es verde

La distribución exponencial es una distribución continua definida sobre valores reales positivos. Es una distribución versátil donde la densidad de probabilidad cae de manera monótona. Tiene como función de probabilidad:

$$f(x, \lambda) = \frac{1}{\lambda} * e^{\frac{-x}{\lambda}} \ , \ x > 0$$

donde x es el valor y $\lambda > 0$ el parámetro. Consideramos un experimento aleatorio en el que medimos una determinada variable aleatoria X, que sigue una distribución exponencial, cosa que escribimos $X \sim Exp(\lambda)$. Tomamos n medidas independientes de X y obtenemos una muestra aleatoria simple $\{x_1, \ldots, x_n\}$, donde cada x_i es una realización de X, para i = 1, ... , n. Resolved los siguientes apartados, ilustrando los resultados de la manera que os parezca más adecuada.

2. Apartado a)

Construid la función de log verosimilitud (negativa) de una muestra.

$$L(\Theta) = -\sum_{i=1}^{N} log(\frac{1}{\lambda} * e^{\frac{-x_i}{\lambda}})$$

$$= -\sum_{i=1}^{N} log(\frac{1}{\lambda}) - \sum_{i=1}^{N} log(e^{\frac{-x_i}{\lambda}})$$

$$= -N * log(\frac{1}{\lambda}) - \sum_{i=1}^{N} \frac{-x_i}{\lambda}$$

$$= N * log(\lambda) - \frac{1}{\lambda} \sum_{i=1}^{N} -x_i$$

3. Apartado b)

Encontrad el estimador de máxima verosimilitud para λ a partir de la muestra

Para el estimador de λ, tomando derivadas parciales sobre este parámetro:

$$\frac{\partial L}{\partial \lambda}(N * log(\lambda) - \frac{1}{\lambda} \sum_{i=1}^{N} -x_i)$$

$$= \frac{N}{\lambda} + \frac{1}{\lambda^2} \sum_{i=1}^{N} -x_i$$

Igualando a 0:

$$\frac{N}{\lambda} + \frac{1}{\lambda^2} \sum_{i=1}^{N} -x_i = 0$$

$$\frac{N}{\lambda} = -\frac{1}{\lambda^2} \sum_{i=1}^{N} -x_i$$

$$\frac{\lambda^2 * N}{\lambda} = -\sum_{i=1}^{N} -x_i$$

$$\lambda = \frac{\sum_{i=1}^{N} x_i}{N}$$

El estimador $\hat{\lambda}$ corresponde a la media muestral.

4. Apartado c)

Demostrad que realmente es un mínimo (y no un extremo cualquiera).

Para demostrar que es un mínimo debemos hacer la segunda derivada, igualar a 0 y ver que cuando aislamos λ nos sale positiva

$$\frac{\partial^2 L}{\partial \lambda^2}(N * log(\lambda) - \frac{1}{\lambda}\sum_{i=1}^{N} -x_i)$$

$$= -\frac{N}{\lambda^2} - \frac{2}{\lambda^3}\sum_{i=1}^{N} -x_i$$

Igualando a 0:

$$-\frac{N}{\lambda^2} - \frac{2}{\lambda^3}\sum_{i=1}^{N} -x_i = 0$$

$$-\frac{2}{\lambda^3}\sum_{i=1}^{N} -x_i = \frac{N}{\lambda^2}$$

$$2\sum_{i=1}^{N} x_i = \frac{\lambda^3 * N}{\lambda^2}$$

$$\lambda = \frac{2*\sum_{i=1}^{N} x_i}{N}$$

Como es positivo podemos afirmar que es un mínimo

5. Apartado d)

Implementad la función de log verosimilitud usando JAX. Generad tres conjuntos de muestras independientes de tamaño 50, 500 y 2000 que sigan la distribución exponencial utilizando la función jax.random.exponential escogiendo un valor específico para el parámetro λ (el que queráis). Mirad como se trata la generación de muestras en el capítulo 3 para que sean independientes.

Para obtener números pseudoaleatorios la variable aleatoria `X` con distribución exponencial y parámetro λ , se utiliza un algoritmo basado en el método de la transformada inversa.

Para generar un valor de $X \sim Exp(\lambda)$ a partir de una variable aleatoria $U \sim U(0,1)$ sabiendo que si $U \sim U(0,1)$ entonces $1 - U \sim U(0,1)$ por lo que una versión eficiente del algoritmo es $\rightarrow X = -\frac{1}{\lambda}\ln(U)$ [1]

Primero de todo fijaremos la `SEED` y λ para realizar el ejercicio.

```
SEED = 1234
lmbda = 5
```

Por lo tanto con el siguiente trozo de código generamos tres muestras independientes de tamaño 50, 500 y 2000 con la `SEED` y λ previamente definidas.

```
key1, key2, key3 = random.split(jax.random.PRNGKey(SEED),3)

data_50 = -lmbda*np.log(jax.random.uniform(key1,[50],minval=0.0,maxval=1.0))
data_500 = -lmbda*np.log(jax.random.uniform(key2,[500],minval=0.0,maxval=1.0))
data_2000 = -lmbda*np.log(jax.random.uniform(key3,[2000],minval=0.0,maxval=1.0))

print(len(data_50), len(data_500), len(data_2000))

50 500 2000
```

6. Apartado e)

Implementad un algoritmo de descenso de gradiente usando `JAX` para optimizar la log verosimilitud y estimad el parámetro de la distribución explorando la tasa de aprendizaje y el número máximo de iteraciones. Podéis utilizar un valor de ϵ para acabar la optimización de `1e-10` e inicializad el valor del parámetro a un valor razonable que no este ni demasiado lejos, ni demasiado cerca del valor real. Comparad el valor estimado con el valor del estimador de máxima verosimilitud y con el valor que hayáis escogido. Comentad los resultados.

Primero de todo implementaremos un algoritmo del descenso del gradiente para optimizar la log verisimilitud. Para el algoritmo deberemos definir primero la función a optimizar para posteriormente poder hacer la derivada y así optimizarla. Si implementamos la función que previamente hemos obtenido en el apartado `a)`:

```
def log_function(lmbd, x):
  return len(x)*jnp.log(lmbd[0]) - 1/lmbd[0] * sum([-xs for xs in x])
```

Una vez tenemos la función de log verosimilitud ya podemos hacer el algoritmo del descenso del gradiente. En este algoritmo usaremos un lr $= 0.01$ y el valor inicial de 2.2 ya que los datos han sido generados con $\lambda = 5$ y no queremos un valor ni muy lejano ni muy cercano. Este algoritmo se basa en hacer la derivada de la función a optimizar, en este caso la log verosimilitud e ir actualizando la λ para que vaya descendiendo hasta el mínimo. Los errores los calculamos haciendo la diferencia de la λ que de momento tenemos predecida con la λ original.

```
def gradLoss(data):
  paramMix = jnp.array([2.2])
  grad_mix = jit(grad(log_function))

  ploss = jnp.abs(paramMix[0]-lmbda)
  lr = 0.01

  for i in range(1000):
    part = grad_mix(paramMix,data)

    paramMix -= (lr * part)

    loss = jnp.abs(paramMix[0]-lmbda)

    if jnp.abs(ploss-loss) < 1e-10:
      print("Finished in iteration: ")
      print(i)
      break
    ploss = loss
  return paramMix
```

Si ejecutamos el algoritmo con los tres conjuntos de datos y observamos la salida vemos lo siguiente:

```
l1 = gradLoss(data_50) # Finished in iteration 615
l2 = gradLoss(data_500) # Finished in iteration 59
l3 = gradLoss(data_2000) # Finished in iteration 17

print(l1, l2, l3)

[5.2542224] [5.277811] [4.823823]
```

Imprimimos las medias de los tres conjuntos:

```
print(data_50.mean(), data_500.mean(), data_2000.mean())

5.2542353 5.277811 4.8238254
```

Ponemos todos los resultados obtenidos en una tabla para poder ver mejor la relación que hay entre ellos y así poder comentarlos mejor.

	$\lambda_{original}$	$\lambda_{predicted}$	media muestral	iterations
data_{50}	5.0	$\sim 5{,}25$	$\sim 5{,}25$	615
data_{500}	5.0	$\sim 5{,}28$	$\sim 5{,}28$	59
data_{2000}	5.0	$\sim 4{,}82$	$\sim 4{,}82$	17

Como podemos observar todas las $\lambda_{predicted}$ son muy parecidas a la original la cual es $\lambda_{original} = 5.0$. También podemos observar como la $\lambda_{predicted}$ es casi el mismo valor que la media muestral, esto tiene todo el sentido ya que como hemos visto en apartados anteriores, la media muestral es el estimador de máxima verosimilitud por tanto al optimizar está función obtenemos la media muestral. En la última columna de la tabla tenemos el número de iteraciones que cada una de las tres ejecuciones ha necesitado para acercarse lo suficiente a la $\lambda_{original}$, podemos observar que cuantos más datos tenga el cojunto más rápido encuentra la $\lambda_{predicted}$. Esto se debe a que cuantos más datos tengas, más preciso el descenso del gradiente y por tanto baja más rápido hacia el mínimo.

7. Apartado f)

Los procesos de desintegración radiactiva siguen una distribución exponencial. Los elementos radiactivos en muestras no suelen aparecer solos, es posible deducir cuáles son los elementos estimando la distribución conjunta de las desintegraciones en la muestra. Supondremos quetenemos una muestra en la que sabemos que hay exactamente dos elementos radiactivos que están en la misma proporción (50 % 50 %). Esto definiría una distribución conjunta con esta expresión:

$$f(x;\lambda_1,\lambda_2) = 0{,}5 * \frac{1}{\lambda_1} * e^{\frac{-x}{\lambda_1}} + 0{,}5 * \frac{1}{\lambda_2} * e^{\frac{-x}{\lambda_2}}$$

Elegid dos valores para los parámetros λ_1 y λ_2 que sean diferentes. Generad una muestra que combine dos muestras independientes del mismo tamaño (2500) generadas con cada distribución exponencial. Estimad los dos parámetros usando descenso de gradiente estudiando el efecto de la tasa de aprendizaje. Fijaos que es una función de dos parámetros por lo que tenéis que calcular derivadas parciales respecto a cada parámetro. La función grad tiene un parámetro argnums donde se puede indicar el número del parámetro sobre el que calcular el gradiente. Representad como evolucionan los parámetros y la función de log verosimilitud con las iteraciones. Comentad los resultados.

Primero de todo fijaremos los dos valores de las diferentes λ's. Decidiremos que $\lambda_1 = 3.0$ y que $\lambda_2 = 11.0$ y también fijaremos la `SEED`.

```
SEED2 = 2345

lmbda1 = 3
lmbda2 = 11
```

Generamos los datos de la misma manera que en el apartado interior:

```
key_1, key_2 = random.split(jax.random.PRNGKey(SEED2),2)

data_2500_1 =
↪  -lmbda1*np.log(jax.random.uniform(key1,[2500],minval=0.0,maxval=1.0))
data_2500_2 =
↪  -lmbda2*np.log(jax.random.uniform(key2,[2500],minval=0.0,maxval=1.0))

data_final = data_2500_1 + data_2500_2

print(len(data_final), len(data_2500_1), len(data_2500_2))
2500 2500 2500

print(data_final.mean(), data_2500_1.mean(), data_2500_2.mean())
14.088757 3.0402856 11.0484705
```

Antes del algoritmo del descenso del gradiente hay que definir la función a optimizar de la cual usaremos sus derivadas parciales.

```
@jax.jit
def func_log(lmbd1,lmbd2,xs):
  return jnp.log(0.5*1/lmbd1[0]*jnp.exp(-xs/lmbd1[0]) +
  ↪  0.5*1/lmbd2[0]*jnp.exp(-xs/lmbd2[0]))
```

Una vez definida la función definimos el algoritmo del descenso del gradiente con las derivadas parciales de los dos parámetros.

```
def gradLoss_2(data):
  param1 = jnp.array([0.5])
  gradFunc1 = jit(grad(func_log,argnums=0)) # first param

  param2 = jnp.array([15.3])
  gradFunc2 = jit(grad(func_log,argnums=1)) # second param

  ploss1 = jnp.abs(param1[0]-lmbda1)
  ploss2 = jnp.abs(param2[0]-lmbda2)

  lr = 0.001

  stop1 = False
  stop2 = False

  paramProg1 = []
  paramProg2 = []

  logProg1 = []
  logProg2 = []

  part1 = -sum(vmap(gradFunc1, in_axes=(None,None, 0))(param1,param2, data))
  part2 = -sum(vmap(gradFunc2, in_axes=(None,None, 0))(param1,param2, data))

  paramProg1.append(param1)
  paramProg2.append(param2)

  logProg1.append(part1)
  logProg2.append(part2)

  for i in range(1000):
      if not stop1:
        part1 = -sum(vmap(gradFunc1, in_axes=(None,None, 0))(param1,param2, data))
        param1 -= (lr * part1)

        paramProg1.append(param1)
        logProg1.append(part1)

        loss1 = jnp.abs(param1[0]-lmbda1)

        if jnp.abs(ploss1-loss1) < 1e-10:
          stop1 = True
```

```
            ploss1 = loss1

        if not stop2:
            part2 = -sum(vmap(gradFunc2, in_axes=(None,None, 0))(param1,param2, data))
            param2 -= (lr * part2)

            paramProg2.append(param2)
            logProg2.append(part2)

            loss2 = jnp.abs(param2[0]-lmbda2)

            if jnp.abs(ploss2-loss2) < 1e-10:
                stop2 = True

            ploss2 = loss2

        if stop1 and stop2:
            break


    return param1, param2, paramProg1, logProg1, paramProg2, logProg2
```

Ejecutamos el algoritmo y observamos el resultado.

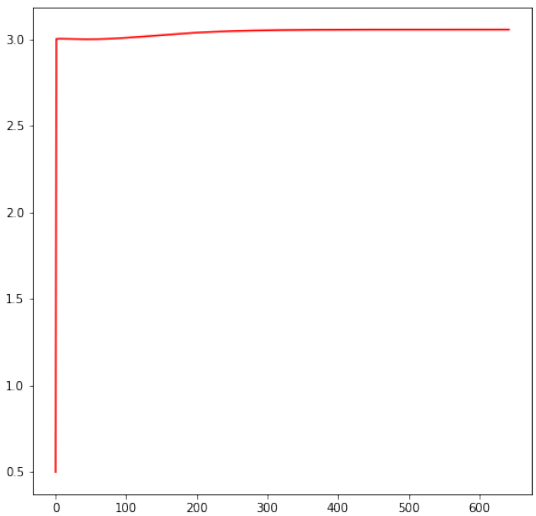
```
l_pred1, l_pred2, paramProg1, logProg1, paramProg2, logProg2 =
↪  gradLoss_2(data_final)

print(l_pred1, l_pred2)
```

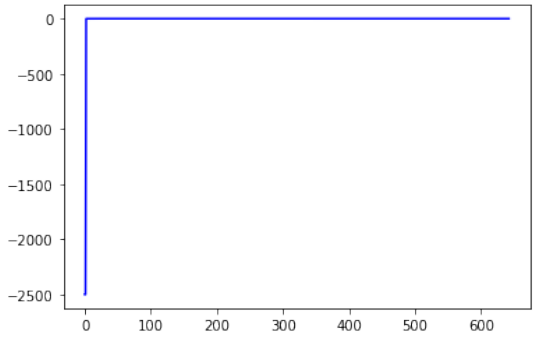
```
[3.0546207] [11.0590315]
```

Representamos la evolución del primer parámetro y de su función log verosimilitud.

```
it1 = np.arange(len(paramProg1))
it2 = np.arange(len(paramProg2))
fig = plt.figure(figsize=(7.5,7.5))
plt.plot(it1,paramProg1, 'red')
```

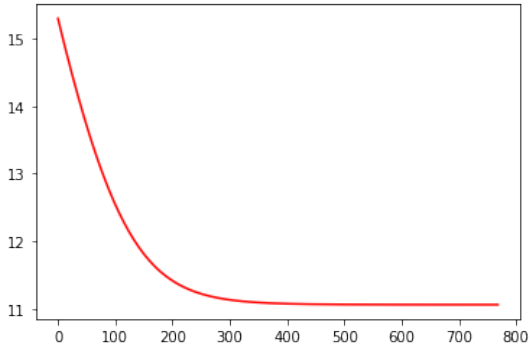


```
plt.plot(it1,logProg1, 'blue')
```

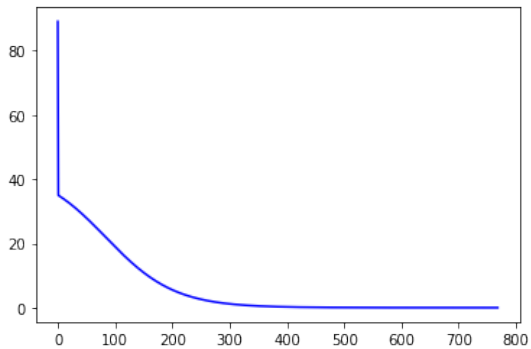


Hacemos lo mismo para el segundo parámetro.

```
plt.plot(it2,paramProg2, 'red')
```




```
plt.plot(it2,logProg2, 'blue')
```



Podemos observar que el primer parámetro ha necesitado menos iteraciones para ser ajustado correctamente, podemos ver como en las primeras iteraciones ya hace una gran corrección del parámetro y lo ajusta casi a la perfección. Por otra banda el segundo parámetro progresa de manera menos radical y con una curva bién marcada. Podemos volver a observar como los parámetros se ajustan a las medias muestrales de los datos generados previamente antes de juntarlos y hacer el dataset de 5000 datos.

Referencias

[1] Wikipedia. Distribución exponencial — Wikipedia, the free encyclopedia. http://es.wikipedia.org/w/index.php?title=Distribuci%C3%B3n%20exponencial&oldid=142929667, 2022. [Online; accessed 08-October-2022].