Optimización Tarea 5

Francisco Javier Peralta Ramírez

10 de marzo de 2018

1. Implementar busqueda de linea con máximo descenso usando los métodos de *backtracking*, *interpolación cuadrática* y *cúbica*. Aplica las implementaciones a las siguientes funciones y compara los resultados con respecto a : el número de interaciones, la norma del gradiente $||\nabla f(\boldsymbol{x}_k)||$ y el error $|f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)|$

   a) Función de Rosembrock, para n = 2 y n = 100

   $$\begin{aligned} f(\boldsymbol{x}) &= [100(x_2 - x_1^2)^2 + (1 - x_1)^2] \\ \boldsymbol{x^0} &= [-1.2, 1]^T \\ \boldsymbol{x^*} &= [1, 1]^T \\ f(\boldsymbol{x^*}) &= 0 \end{aligned}$$

   Recordamos de la tarea anterior que el gradiente para Rosembrock de n variables está dado por:

   $$\begin{pmatrix} -400(x_2 - x_1^2)x_1 - 2(1 - x_1) \\ 200(x_2 - x_1^2) - 400(x_3 - x_2^2)x_2 - 2(1 - x_2) \\ \vdots \\ 200(x_i - x_{i-1}^2) - 400(x_{i+1} - x_i^2)x_i - 2(1 - x_i) \\ \vdots \\ 200(x_n - x_{n-1}^2) \end{pmatrix}$$

   Para Rosembrock con $n = 2$ obtenemos los siguientes resultados:

Algoritmo	# iter	$\|f(x) - f(x^*)\|$	$\|\nabla f(x_k)\|$	α_0
Backtracking	34439	1.221E-23	8.092E-11	1
Interpolación	162067	1.212E-20	9.951E-11	0.01

   Para Rosembrock con $n = 100$ obtenemos los siguientes resultados:

Algoritmo	# iter	$\|f(x) - f(x^*)\|$	$\|\nabla f(x_k)\|$	α_0
Backtracking	19302	3.98662	4.266E-05	1
Interpolación	17048	3.98662	9.426E-06	0.01

   Podemos ver que ambos algoritmos convergen, aun que curisoamente para Rosembrock 100 ambos convergen más rapido y a un mínimo local. Para el punto dado, *Backgracking* parece tener mejor desempeño ya que en $n = 2$ converge en casi 5 veces menos iteraciones pero en cuanto a tiempos *Interpolación* gana. Para $n = 2$ los timepos son casi iguales, pero para $n = 100$ *Interpolación* toma 0.192s mientras que *Backtracking* toma 1.854s.

   b) Función de Wood

   $$\begin{aligned} f(\boldsymbol{x}) &= 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 \\ &\quad + 90(x_3^2 - x_4)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] \\ &\quad + 19.8(x_2 - 1)(x_4 - 1) \\ \boldsymbol{x^0} &= [-3, -1, -3, -1]^T \\ \boldsymbol{x^*} &= [1, 1, 1, 1]^T \\ f(\boldsymbol{x^*}) &= 0 \end{aligned}$$

   De igual manera en la tarea anterior vimos que el gradiente de está función es:

   $$\begin{pmatrix} 400(x_1^2 - x_2)x_1 + 2(x_1 - 1) \\ -200(x_1^2 - x_2) + 20.2(x_2 - 1) + 19.8(x_4 - 1) \\ 2(x_3 - 1) + 360(x_3^2 - x_4)x_3 \\ -180(x_3^2 - x_4) + 20.2(x_4 - 1) + 19.8(x_2 - 1) \end{pmatrix}$$

Algoritmo	# iter	$\|f(x) - f(x^*)\|$	$\|\nabla f(x_k)\|$	α_0
Backtracking	18660	4.990E-23	9.901E-11	1
Interpolación	34305708	6.078E-21	9.620E-11	0.05

2. Podemos obtener una función para clasificar números en una imágen de 28×28 y encontrar los parámetros que la minimizan. Dicha función está dada por:

   $$\begin{aligned} h(\boldsymbol{\beta}, \beta_0) &= \sum_{i=1}^{n} y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i) \\ \pi_i := \pi_i(\boldsymbol{\beta}, \beta_0) &= \frac{1}{1 + exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)} \\ \boldsymbol{\beta} &= [\beta_1, \beta_2, \cdots, \beta_{784}] \end{aligned}$$

   Si calculamos su gradiente, podemos usar nuestro algoritmo de máximo descenso con alguno de nuestros tipos de paso, preferentemente uno que no utilize el Hessiano, como *backtracking*.

   Primero encontramos el gradiente de π_i con respecto a $\beta_0, \beta_1, \cdots, \beta_{784}$

   $$\begin{aligned} \frac{\delta \pi_i}{\delta \beta_0} &= \frac{exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)}{[1 + exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)]^2} \\ \frac{\delta \pi_i}{\delta \beta_j} &= \frac{x_{i,j} exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)}{[1 + exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)]^2} \end{aligned}$$

Luego encontramos el gradiente de la función h

$$\frac{\delta h}{\delta \beta_0} = \sum_{i=1}^{n} \frac{y_i exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0) - (1 - y_i)}{1 + exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)}$$
$$\frac{\delta h}{\delta \beta_j} = \sum_{i=1}^{n} \frac{y_i x_{i,j} exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0) - x_{i,j}(1 - y_i)}{[1 + exp(-\boldsymbol{x}_i^T \boldsymbol{\beta} - \beta_0)]^2}$$

Podemos simplificar la formula usando π_i

$$\begin{aligned}
\frac{\delta h}{\delta \beta_0} &= \sum_{i=1}^{n} y_i(1 - \pi_i) + (y_i - 1)\pi_i \\
&= \sum_{i=1}^{n} y_i - \pi_i \\
\frac{\delta h}{\delta \beta_j} &= \sum_{i=1}^{n} x_{i,j} y_i (1 - \pi_i) + x_{i,j}(y_i - 1)\pi_i \\
&= \sum_{i=1}^{n} x_{i,j}(y_i - \pi_i)
\end{aligned}$$

Para utilizar los métodos aprendidos en clase, podemos tomar dos caminos, cambiar la confición de Armijo para asegurar ascenso suficiente o cambiar el problema a uno de minimización. Ya que se tienen muchos datos, se optó por usar un subconjunto de ellos, en este caso 1000, es importante resaltar que son 1000 datos de los números seleccionados y no de todos los disponibles, así que para el archivo de 50000 datos, si estos están distribuidos uniformemente, tendríamos 5000 por digito, lo que signigica que sólo usamos el $10\,\%$. Para hacer esto multiplicamos la función por -1 y el gradiente también.

Se probó con diferentes números, como se esperaría, con números similares tarda más en converger y tiene mayor error al hacer las pruebas.

Números	# iter	$f(x)$	Error(test)
0, 1	167	-8.461E-12	0.0008
0, 8	1674	-1.778E-05	0.0054

Nuestro algoritmo de entrenamiento queda de la siguiente forma:

Algorithm 1 trainNumbers

```
1:  function TRAIN_NUMBER(ydat, xdat, n1, n2, maxIter)
2:      for y in ydat do
3:          if y is n1 or n2 then
4:              x.append(readLine(xdat))
5:          else
6:              readLine(xdat)                          ▷ Saltar linea
7:          end if
8:      end for
9:      beta := [1/748] * 748    ▷ Inicializa con valores iguales
10:     ev = numer(beta, y, x)
11:     for i to maxIter do
12:         dir := number_gradient(beta, y, x)
13:         step := number_step_backtrack(dir, beta, y, x)
14:         scaleVector(dir, step)
15:         nev = numer(beta+dir, y, x)
16:         if |ev - nev| < 1E-8 then
17:             break
18:         end if
19:     end for
20:     return beta
21: end function
```