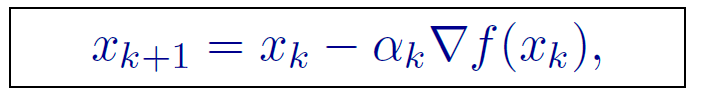
|  |  |
| --- | --- |
| Algoritmos en la Ciencia de Datos  Sección U | Eduardo Andrés Santizo Olivet  Carné: 21007361 |

**Laboratorio No. 3**

Gradient Descent Methods

Como se discutió en clase, la idea central del algoritmo de Gradient Descent (GD) es intentar resolver un problema de minimización mediante la evaluación del valor de la función en la dirección en donde se encuentra el mínimo. Una pregunta natural en este momento es: ¿Cuál es esta dirección? Sabemos que el vector gradiente nos provee la dirección de máximo crecimiento de la función minimizada. Por tanto, el negativo del gradiente nos devuelve la dirección de menor crecimiento de la función. Esta observación da origen al método GD:



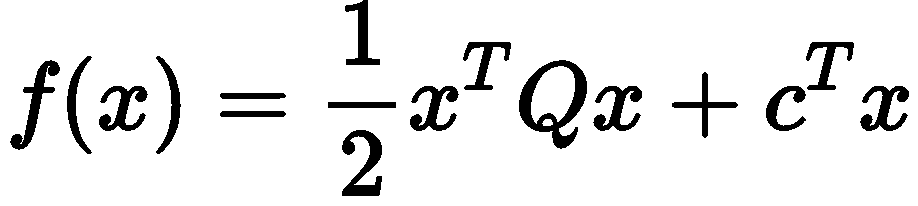
En donde Alpha sub “k” es el llamado “step size” o mejor conocido como “learning rate” en el ámbito de machine learning. En este laboratorio nos enfocaremos en dos puntos importantes:

1. Implementar el algoritmo de GD presentado en clase.
2. Investigar la convergencia de este algoritmo para distintas formas de elegir Alpha sub “k”.

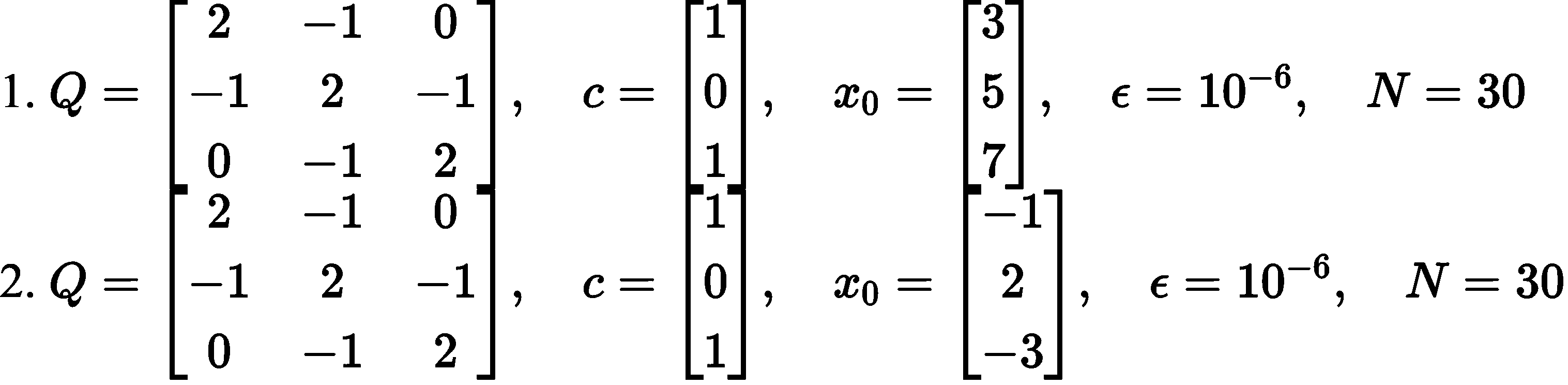
Instrucciones: Implemente el algoritmo GD y utilícelo para resolver cada uno de los problemas presentados a continuación.

**Problema 1**

Aplique el método GD para minimizar la función cuadrática (QP).



Utilizar los parámetros dados a continuación para dicha ecuación:

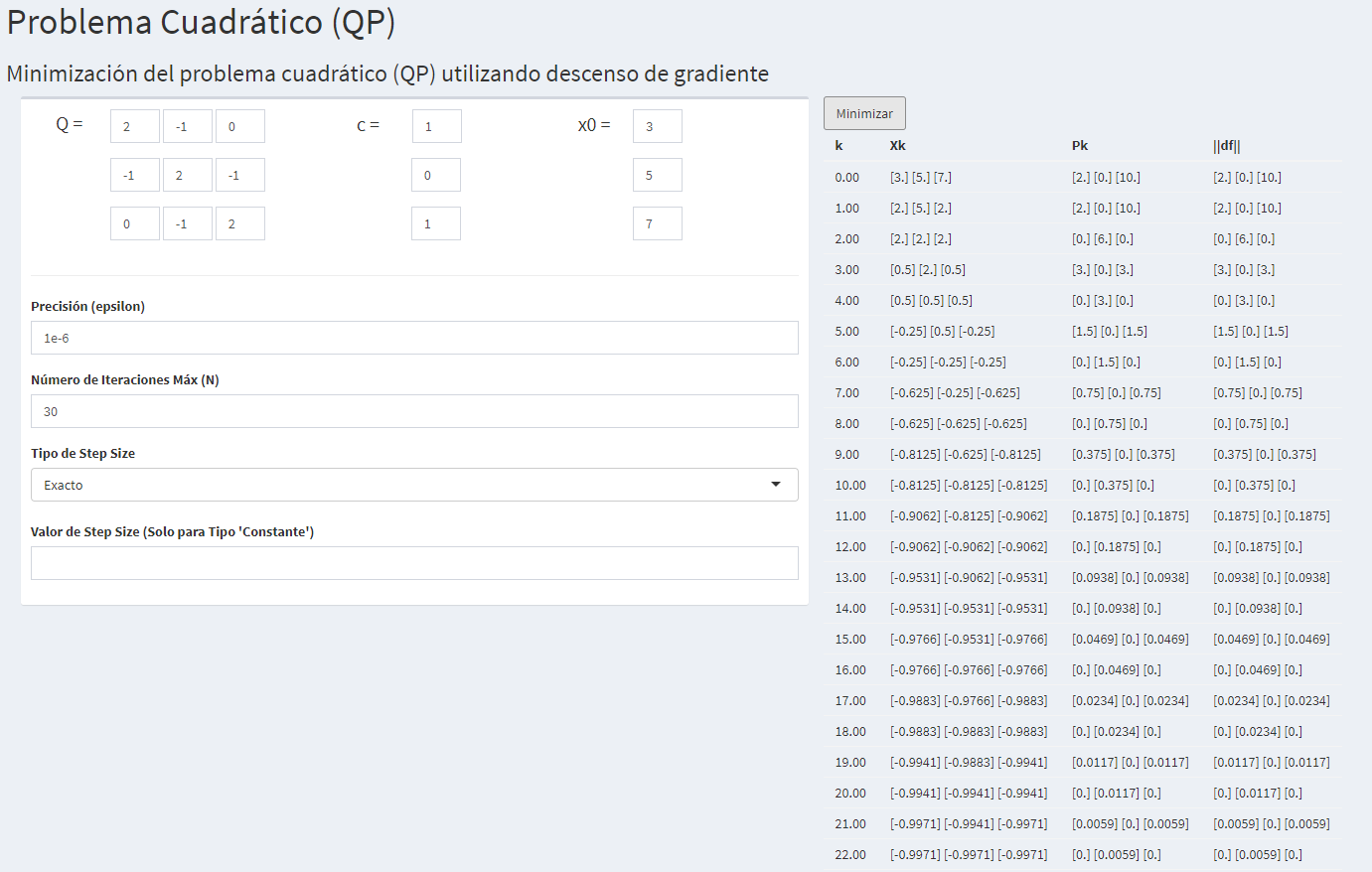


Para la elección de Alpha aplique

* Step size exacto: Solucionar un problema de minimización para encontrar el Alpha óptimo.
* Step size constante: Elegir un valor constante de step size para todas las iteraciones.
* Step size variable: Utilizar la sucesión Alpha = 1/k para todas las iteraciones.

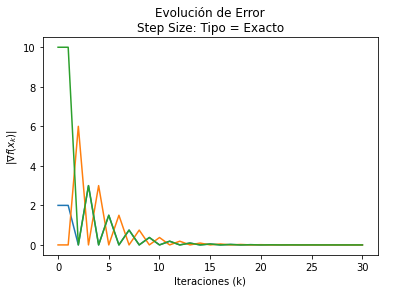
Para cada caso, su output debe ser mostrado en una tabla con cuatro columnas: El número de iteración, el valor de la solución, la dirección del gradiente y el valor del error. Finalmente, realizar una gráfica del error versus el número de iteración en donde se observe el comportamiento del algoritmo para cada una de las formas de elegir el step size. ¿Qué observa? ¿Con cuál elección se obtiene el mejor comportamiento?

Interfaz Utilizada



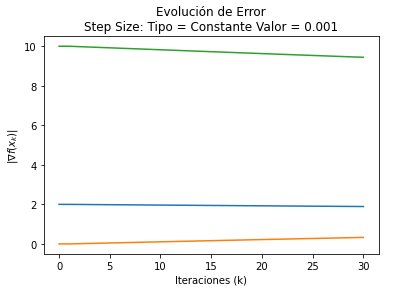
Set de Parámetros 1 – Step Size Exacto

|  |  |
| --- | --- |
|  |  |



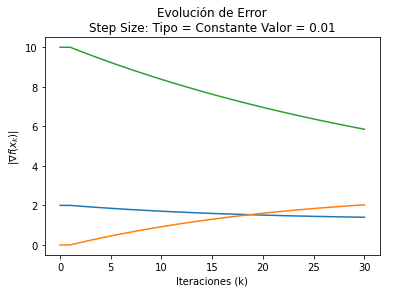
Set de Parámetros 1 – Step Size Constante (0.001)

|  |  |
| --- | --- |
|  |  |



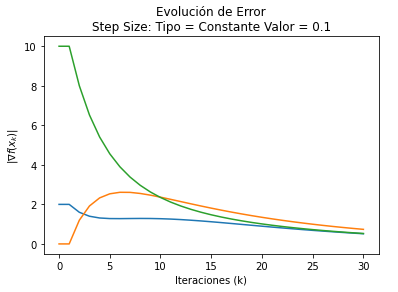
Set de Parámetros 1 – Step Size Constante (0.01)

|  |  |
| --- | --- |
|  |  |



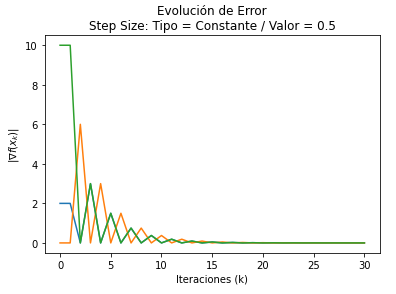
Set de Parámetros 1 – Step Size Constante (0.1)

|  |  |
| --- | --- |
|  |  |

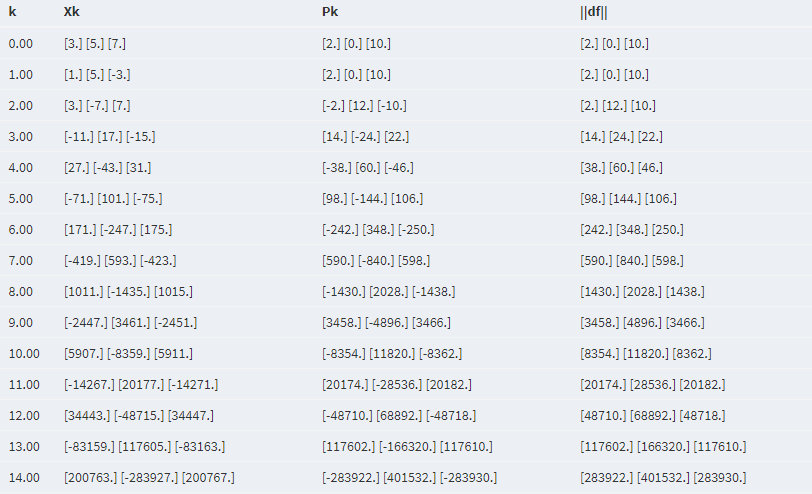


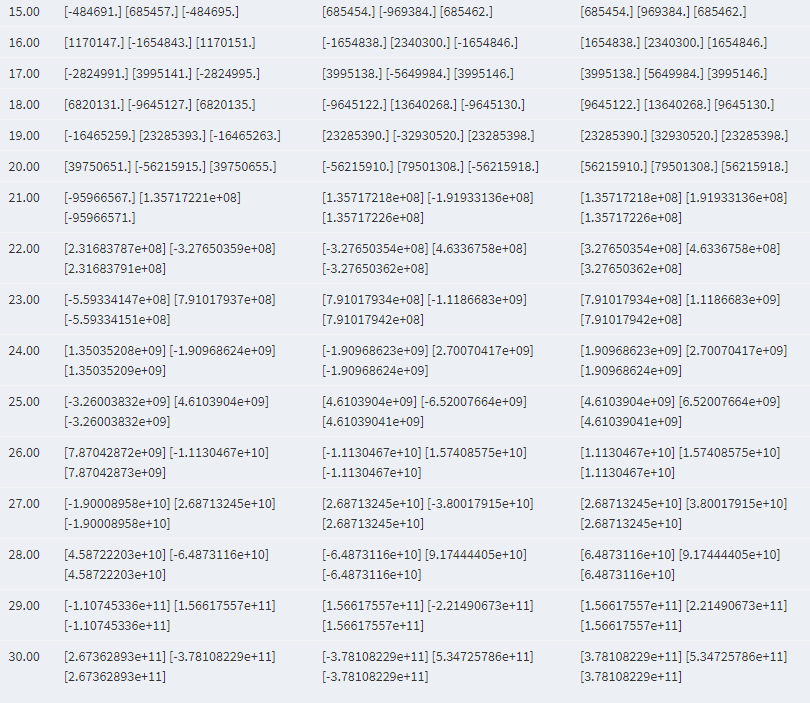
Set de Parámetros 1 – Step Size Constante (0.5)

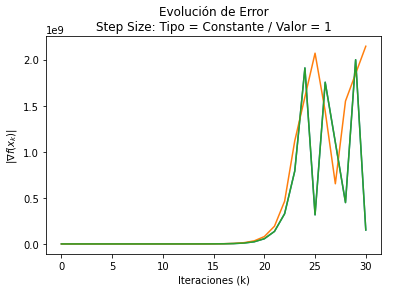
|  |  |
| --- | --- |
|  |  |



Set de Parámetros 1 – Step Size Constante (1)

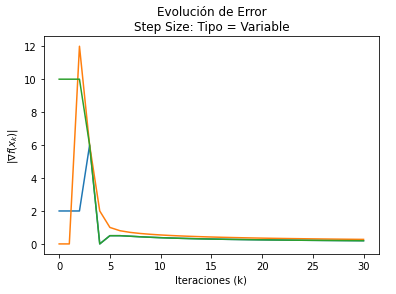






Set de Parámetros 1 – Step Size Variable

|  |  |
| --- | --- |
|  |  |



Set de Parámetros 1 – Conclusiones

Al finalizar todas las pruebas realizadas, se llegó a determinar que los “step sizes” para los que se obtuvieron los mejores resultados fueron el “step size” exacto y el “step size” constante con un valor de 0.5. Ambos métodos, fueron capaces de llegar a un valor de error casi nulo en aproximadamente 15 iteraciones. De hecho, si se observan las gráficas generadas, se puede comprobar que el comportamiento fue virtualmente idéntico entre ambos métodos. Esto se debe a que el valor empleado al utilizar un “step size” exacto consistía de 0.5, lo que implica que ambos métodos (Exacto y constante con valor de 0.5) son básicamente iguales.

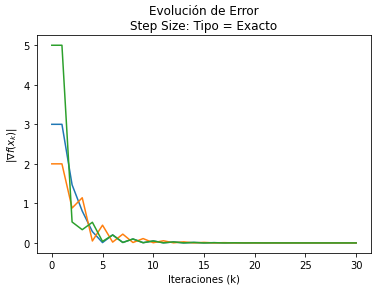
Curiosamente, si el valor constante del “step size” se reducía, la convergencia se alentaba severamente, mientras que, si se incrementaba, el algoritmo divergía.

Otra observación interesante fue el método de “step size” variable. Este, a primera vista, podría parecer el método con la convergencia más rápida de todos, al estabilizarse luego de apenas 5 iteraciones. Sin embargo, esto es engañoso. Al observar la tabla de iteraciones, se puede observar que el valor de error más pequeño alcanzado luego de 30 iteraciones fue de 0.1941, lo que implica que este método realiza ajustes sumamente bruscos al inicio del proceso (como es de esperar en un “step size” inicial igual a 1), para luego lentamente refinar sus resultados a medida que continúa el algoritmo (a medida que el “step size” disminuye). Esto significa que el algoritmo llega rápidamente a valores bajos de error, pero conforme avanza, el progreso se estanca, tomando probablemente incluso más iteraciones que todos los demás métodos con “step sizes” constantes y bajos.

Al fin de al cabo, existiría un trade-off entre utilizar el método exacto y variable. El exacto cuenta con una tasa de convergencia alta, y una precisión similar. La desventaja de dicho método es que este es mucho más computacionalmente demandante en caso se generalice a cualquier función. El método variable no es preciso, porque “le cuesta” llegar a la respuesta real, pero converge de manera sumamente rápida a un valor “suficientemente bueno” y su complejidad computacional es virtualmente nula.

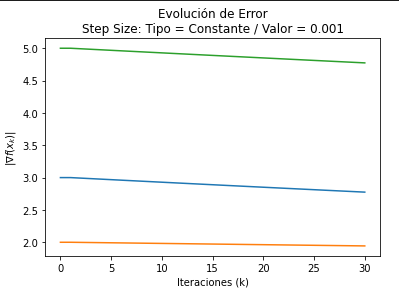
Set de Parámetros 2 – Step Size Exacto

|  |  |
| --- | --- |
|  |  |



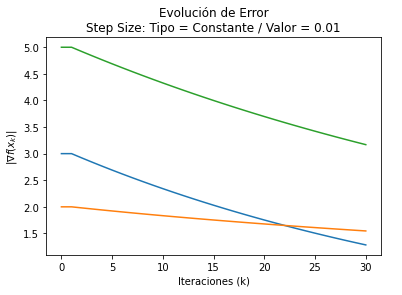
Set de Parámetros 2 – Step Size Constante (0.001)

|  |  |
| --- | --- |
|  |  |



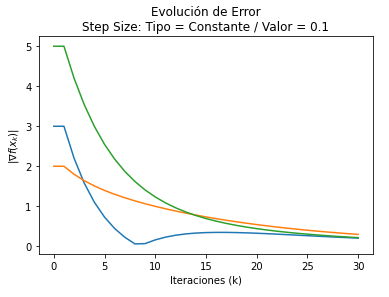
Set de Parámetros 2 – Step Size Constante (0.01)

|  |  |
| --- | --- |
|  |  |



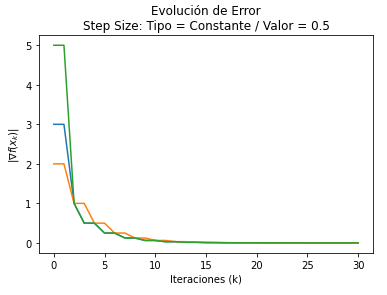
Set de Parámetros 2 – Step Size Constante (0.1)

|  |  |
| --- | --- |
|  |  |

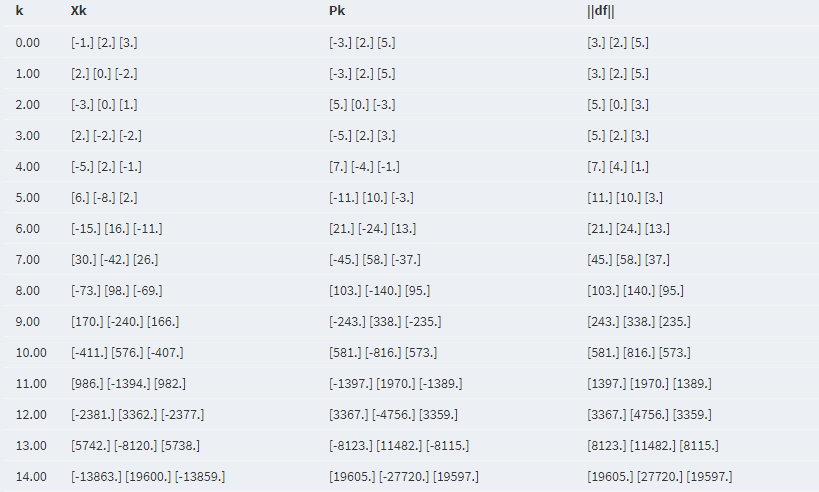


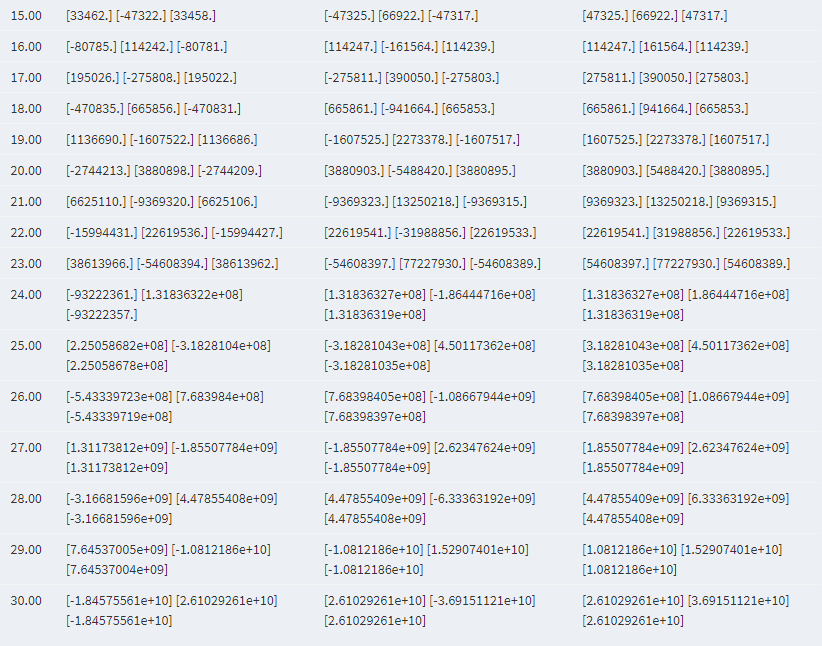
Set de Parámetros 2 – Step Size Constante (0.5)

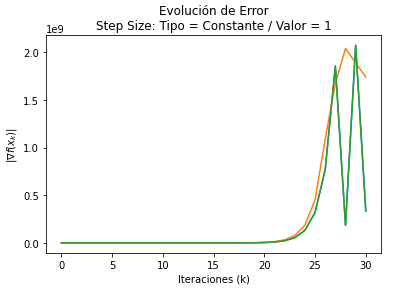
|  |  |
| --- | --- |
|  |  |



Set de Parámetros 2 – Step Size Constante (1)

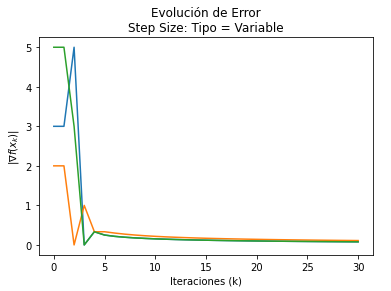






Set de Parámetros 2 – Step Size Variable

|  |  |
| --- | --- |
|  |  |



Set de Parámetros 2 – Conclusión

Los resultados fueron casi idénticos a los de el primer set de parámetros: Si para un valor de “step size” constante, el valor se decrementa mucho, la convergencia se ralentiza, mientras que si se incrementa, el algoritmo diverge. Existe un valor ideal, que puede ser obtenido utilizando el método “Exacto”. A su vez, el método de “step size” variable converge rápido a un valor “suficientemente bueno” y seguido de esto ralentiza severamente la velocidad de convergencia.

La única diferencia significativa entre ambos sets es que, debido a que el punto inicial del segundo set de parámetros está mucho más cercano a la solución real (-1, -1, -1), todos los casos probados convergen mucho más rápido que los casos correspondientes a los del primer set de parámetros.

**Problema 2**

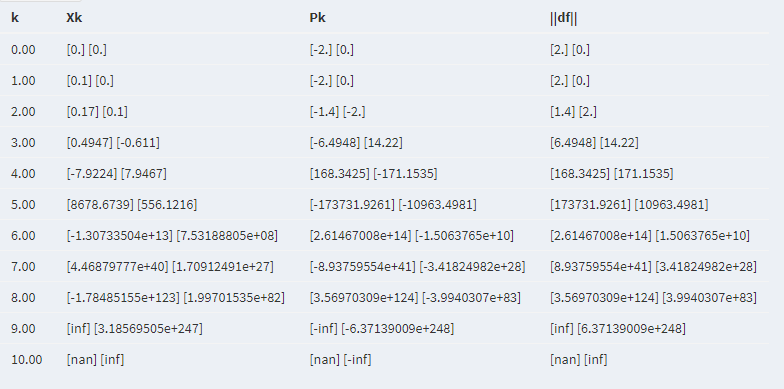
Considere la función de Rosenbrock. Esta es utilizada como benchmark en la evaluación de algoritmos. Algunos autores le llaman “banana function” debido a la forma de sus curvas de nivel. Aplique el método de GD para resolver el problema de minimización. Utilice un punto inicial de (0,0) y un “step size” constante de 0.05.

Detenga la ejecución del algoritmo cuando la precisión alcance un nivel deseado o bien cuando el número de iteraciones exceda 1000. Su output debe ser mostrado en una tabla similar a la del problema anterior. Finalmente, varíe el punto inicial ¿qué observa?

Interfaz Utilizada

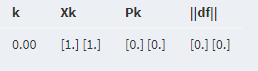


Punto Inicial (0,0)



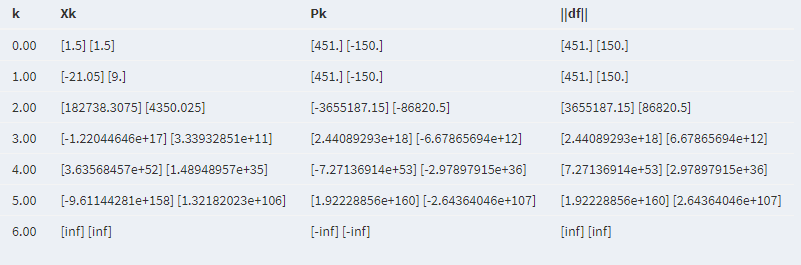
Resultado: Diverge

Punto Inicial (1,1)

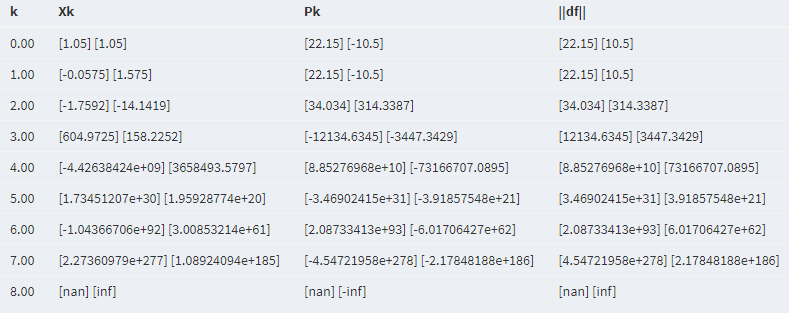


Resultado: Dado que el mínimo de la función está en (1,1), el número de iteraciones se reduce a 0, no hay resultado que optimizar.

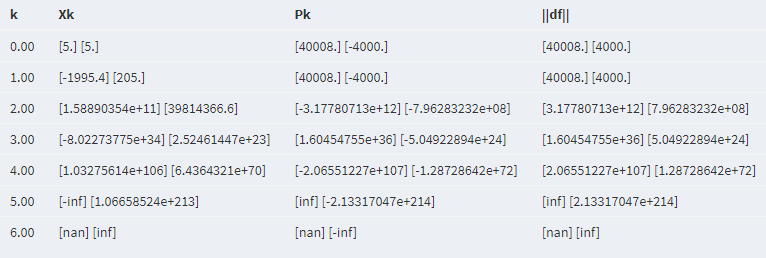
Punto Inicial (1.5,1.5)



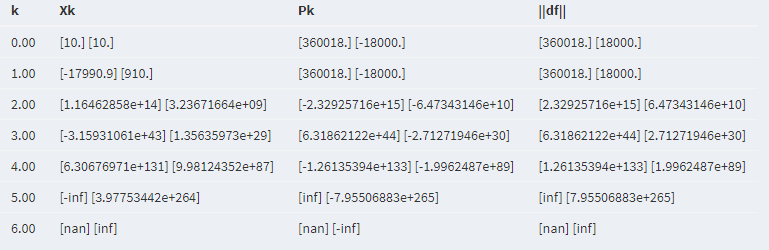
Punto Inicial (1.05,1.05)



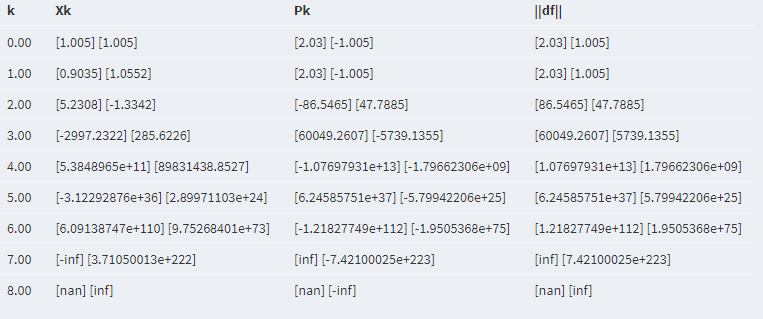
Punto Inicial (5, 5)



Punto Inicial (10, 10)



Punto Inicial (1.005, 1.005)



Conclusiones

Luego de variar múltiples veces el punto inicial del algoritmo, se llegó a observar que, mientras el “step size” permanezca inalterado en un valor de 0.05, no importando la cercanía del punto inicial a la solución real, el algoritmo siempre divergirá. Esto se debe a que la función cuenta con dos problemas fundamentales. En primer lugar, la región donde se encuentra el mínimo tiene una curva patológica muy cercana, lo que puede llegar a “atrapar” al algoritmo. A su vez, la misma cuenta con un problema significativo de escalamiento, ya que la función incrementa agresivamente su “altura” a medida que cualquier punto se aleja del origen. Esto implica que cualquier “step” donde se avance “de más”, causaría que el algoritmo casi automáticamente diverja (como se observó en las pruebas anteriores). Aquí es donde se aprecia la razón por la que se requieren métodos inmunes a estos problemas.