Métodos de Optimización II Tarea I

Iván Vega Gutiérrez

Centro de Investigación en Matemáticas A.C.
Unidad Aguascalientes
E-mail: ivan.vega@cimat.mx

I. Introducción

Consideremos la siguiente función

$$f(x_1, x_2) = \frac{1}{2}[(x_1^4 - 16x_1^2 + 5x_1) + (x_2^4 - 16x_2^2 + 5x_2)] \tag{1}$$

Supongamos que el espacio de búsqueda es

$$\Omega = [-8, 8] \times [-8, 8]$$

y la solución óptima global es $x^* = [2.9035, 2.9035]$ con $f(x^*) = -78.33$

II. Metodología

Para hallar la solución del problema dado se implementaron los siguientes tres algoritmos:

- Búsqueda aleatoria simple (BAS).
- Búsqueda aleatoria localizada (BAL).
- Búsqueda aleatoria localizada mejorada (BALM).

Cada algoritmo tiene como parámetro 500 iteraciones y solución inicial $x_0 = [4, 6.4]$. Para los algoritmos de Búsqueda aleatoria localizada y Búsqueda aleatoria localizada mejorada se define el parámetro de la distribución normal $\sigma = \sqrt{3}$.

Para realizar una comparación entre los tres algoritmos se hicieron 40 ejecuciones independientes para cada algoritmo y se halló la diferencia relativa, la cual está definida como la diferencia de la mejor solución encontrada por el algoritmo en esa ejecución y la solución óptima, es decir, $f(x^{mejor}) - f(x^*)$.

Asimismo, se procedió a hallar los siguientes intervalos de confianza para cada algoritmo

$$[\text{media de la muestra} - 2.023\sqrt{s^2/40},\ \text{media de la muestra} + 2.023\sqrt{s^2/40}]$$

2.023 es el valor-t para la probabilidad de 0.025 con 39 grados de libertad y s^2 es la varianza de la muestra.

III. Experimentación

A continuación se presentan los resultados de las 40 ejecuciones de cada algoritmo.

III.1. Búsqueda aleatoria simple

Ejec	x1	x2	f(x)
1	-2.4194714449714585	-2.9921965174952945	-74.7718397342805
2	-2.7269093407030898	-2.500705904948358	-75.38493091088797
3	-3.044677767076152	-2.9362350933715664	-77.9526359787271
4	-3.1439065609880927	-2.514845664291503	-74.96813679159241
5	-2.866317834899787	-2.6885763748215474	-77.56630530527977
6	-2.983145636359353	-2.933050850152492	-78.20457250450644
7	-2.805414327764705	-3.000470556572795	-78.00348004419835
8	-3.1075362125295634	-2.93716091563417	-77.54276992733935
9	-2.4927042508834063	-2.680658645428096	-75.00639482283975
10	-3.302209934361249	-2.874752709966655	-75.18918211747123
11	-2.704759676557954	-2.9017482084601713	-77.69389358213081
12	-2.775387167869562	-2.960742803742736	-78.00277683151853
13	-2.9674472044123323	-3.5440608827354296	-69.55568334619193
14	-3.0117467383174805	-2.6808813786806045	-77.32807548640864
15	-2.7315011673083767	-2.460614337557123	-74.94283542204437
16	-2.8873574538189093	-2.791120688525906	-78.11749152230226
17	-2.8293824987976635	-2.5979845185201516	-76.78655726911776
18	-2.2211919495826127	-2.745970990733557	-71.61115412730621
19	-2.6463082726057703	-3.0971607468504807	-76.59373932721849
20	-2.2524014852937704	-2.807886015235521	-72.36126669084007
21	-2.969806019506553	-3.1581240944582856	-77.03599188133678
22	-2.7182574542956672	-2.8454742422448884	-77.71794431707946
23	-2.8786372415200088	-2.9893794319148235	-78.19057301076992
24	-2.9301889500460483	-3.3843384972748467	-73.65043195893819
25	-2.452789946264419	-2.701750700947528	-74.67319743430218
26	-2.9055765876918436	-2.829523501228735	-78.23988302608691
27	-2.8451954245510525	-2.720143177045891	-77.7283279843161
28	-2.802830589169311	-3.111446863170773	-77.36225409675654
29	-3.0737646855211835	-2.9950929920520935	-77.65273496686035
30	-2.869267504874978	-3.3901656906677076	-73.52020437455093
31	-2.843084990043044	-3.181523329914068	-76.80643084657567
32	-2.8566014460437117	-2.416364472190855	-74.8342318768718
33	-2.9892253443836267	-2.1956785820977167	-71.47170307914021
34	-3.2745788548119066	-2.8774479264337085	-75.63394760197497
35	-2.8273811879692357	-3.033593763160944	-77.92918691562787
36	-3.1949858598878755	-2.8089452450012597	-76.56630912198823
37	-3.115256795039681	-2.5541926348774044	-75.6309740253069
38	-2.974830341950403	-2.5902011980363753	-76.71849867705876
39	-3.028975192793421	-2.762580524019647	-77.72117336264458
40	-2.783580696223426	-2.6358981020281256	-76.96363062128115

III.2. Búsqueda aleatoria localizada

```
Ejec      x1                x2                f(x)
1 -2.85683277804952 -2.8339133098948723 -78.21334263490162
2 -2.881437236701908 -3.0113936705233733 -78.11543210581188
3 2.599844938596279 -3.059371622080846 -63.45455012554689
4 -2.981050618488437 -2.892605013932511 -78.22364901371363
5 2.7719272011850813 -2.871412103255557 -64.16863724691251
6 2.8088341338515757 -2.8699294485092564 -64.11867364425862
7 -2.8089245615978906 2.821869889416278 -63.96090660664777
8 -2.9362586932090564 2.603446961110099 -63.89210654249338
9 -2.908780745255348 2.689717739230798 -64.14846178733379
10 -2.942408839449113 -2.8095742065872784 -78.15797799706446
11 -2.978715347209527 -2.7587910111010574 -77.88723530209222
12 2.863870264270162 -2.879330541567815 -63.97609056404374
13 2.7381667853468907 -2.897377339482142 -64.19387035162131
14 -2.919014070238757 -2.7586196946538046 -77.98249299298885
15 2.70694084701214 2.654693339634412 -49.91607924443318
16 2.732375333351469 2.743404368383868 -50.05569474042463
17 -2.8052862754498147 2.683269124099338 -63.97649140331163
18 -2.595537163040577 2.687421926880439 -62.670009656975594
19 -2.8525005891512203 -2.6670635794343953 -77.3963766571188
20 -2.8755454420382742 2.547689323211391 -63.64456218158982
21 2.8539182439937494 -2.926658714974171 -64.01156080927848
22 -2.869783220560065 -2.8109990434575667 -78.16935893834625
23 -2.785701530349604 -2.9429201360076873 -78.07447168580224
24 -3.0660100497379514 2.669538495558457 -63.62903689924052
25 -2.310423121606441 -2.607901842278116 -72.03421857964653
26 2.682445330544249 2.875067942183053 -49.74723494354623
27 -2.8216822567592654 -2.8977523247063903 -78.21906828747993
28 -2.948899767516408 -2.9622554202298805 -78.2353938101769
29 -2.866127835591053 -2.9251230807842337 -78.30032175645984
30 -2.6534338142101332 -3.1062506060901556 -76.57983446956284
31 -2.8446262409505074 -2.8854717634945453 -78.26790152797963
32 -2.9405117733528643 -2.987084354180047 -78.18427572688353
33 2.6264353548767385 -2.8952284138130735 -63.991865068702054
34 2.6620051838601206 2.7645237854522957 -49.952357456337424
35 2.4381461762471726 -3.1541493551817306 -61.778950223210295
36 -2.9260193410499977 -2.8960907791133956 -78.32256725915511
37 -2.8660818581870764 -2.7578167276229584 -77.95896320029001
38 2.9651477230398604 2.8076359155268356 -49.24746324857187
39 2.7679958600254104 2.822393927187984 -49.96625512764776
40 2.6585552326023376 2.718957186685513 -49.93743906031335
```

III.3. Búsqueda aleatoria localizada mejorada

```
Ejec         x1                x2                f(x)
1 -3.1149191620577854 -2.752920730572704 -77.13116912383649
```

```
2 -2.9885048141170047 2.6711126256964595 -63.98570143253355
3 -2.917891523687241 2.730258158147154 -64.18804966369707
4 -2.8376052237674343 -2.9277371730626687 -78.24861483793843
5 -2.893139058417764 -2.9194013733259236 -78.32609271643551
6 -2.9977068051175753 -2.749501824509727 -77.7847753953724
7 -2.8192529308718504 -2.9845642287202825 -78.09631027833822
8 2.808152905037974 -3.011048157869806 -63.932092265712754
9 -2.897811295900626 -2.927733991258232 -78.32155715111386
10 -2.986731655039455 -3.048455663252662 -77.82821699478352
11 -2.882337171161698 -2.8842371134355327 -78.31822020740407
12 -2.8405201189391227 -3.000574192723291 -78.09693449453664
13 -2.9140330931023164 2.657242388071731 -64.08022739849605
14 2.6948487310618163 2.750771250768661 -50.01992672782588
15 -2.911981454425335 -2.9174807238322558 -78.3277148446117
16 -2.7914655879156745 -2.895916114333218 -78.12225499409688
17 -2.9957400178128992 -3.0216176237393064 -77.92996408176933
18 -2.885461034817322 -2.874944563270424 -78.31271965265674
19 -2.890437358257228 -2.929889757709953 -78.3172608736792
20 -2.916943975221683 -3.0066821009534292 -78.13880470204009
21 -2.884614664973716 -2.8335411359827365 -78.2434492289963
22 -2.8917572174328536 2.7750095690290384 -64.18145624054964
23 2.566704022651244 -2.8965352621338325 -63.75164564042291
24 -2.9052910553363427 -2.9108526809115127 -78.331349535991 73
25 -2.928390941191632 -2.8324087607950474 -78.236160365792
26 2.7992253028051453 2.6532652256227327 -49.89429392476667
27 -2.8656300568567055 2.7691300397915533 -64.16372782468845
28 -2.9377062215043988 -3.001067985164932 -78.14198167854138
29 -2.9162306812516703 -2.6645255496360987 -77.4194069160942
30 2.6865834290609545 -2.91216610786085 -64.14244218418628
31 -2.9638495240810623 2.8634300080279167 -63.92355739389063
32 -2.968041225819566 -2.7683309196093835 -77.95690881836859
33 -2.9838631269040556 -2.9265522792319336 -78.20848965744108
34 -2.901229011263562 -3.0180027891686887 -78.09687117241144
35 -2.9100858935171523 -2.900073940719113 -78.33138072363425
36 -2.8848728421579883 -2.9308330466762773 -78.31334278911953
37 -2.784485801088583 -3.075042832389057 -77.55860068181576
38 -2.8873801426901613 2.6684866561708827 -64.10398349087615
39 2.7543316694630238 -2.7996893744702738 -64.01475836530686
40 -2.9140866951985247 -2.855577932245641 -78.29126998701024
```

IV.. Discusión de Resultados

En la Tabla (1) se puede observar que la mejor solución para cada algoritmo es muy similar, y aunque la diferencia es mínima, la mejor solución se obtiene con el algoritmo de búsqueda aleatoria localizada mejorada, seguido de la búsqueda aleatoria localizada y por último la búsqueda aleatoria simple, lo cual es bastante razonable, pues se supone que la intensificación es mayor en los algoritmos de búsqueda localizada y localizada mejorada que en la búsqueda simple.

Por otro lado, para la peor solución pasa lo contrario, los algoritmos de búsqueda localizada son

menos eficientes que la búsqueda simple, esto se debe a que la búsqueda aleatoria simple tiene un mayor rango de exploración, por lo tanto el algoritmo no se atora en mínimos locales, caso contrario para los algoritmos de búsqueda localizada, los cuales no salen de un mínimo local.

En cuanto a la media de las diferencias relativas, se obtuvo que la más pequeña es para el algoritmo BAS, seguido de BALM y por último BAL. En consecuencia existe menor variabilidad en la búsqueda simple que en las búsquedas localizadas. En particular, la búsqueda aleatoria localizada mejorada da las peores aproximaciones, pues en la mayoría de las ejecuciones se alcanza un mínimo local.

Por último se muestran los respectivos intervalos de confianza para cada algoritmo:

- Búsqueda aleatoria simple: [1.5661415884974534 , 2.9122908654190702]
- Búsqueda aleatoria localizada: [7.440277191777801 , 14.081663864326442]
- Búsqueda aleatoria localizada mejorada: [2.9084398664103497 , 8.110975910750522]

	$f(x_1, x_2)$				Mejor		Peor	
Algoritmo	Mejor	Peor	Media	Varianza	x_1	x_2	x_1	x_2
BAS	-78.2399	-69.5557	2.2392	4.4279	-2.9056	-2.8295	-2.9674	-3.5441
BAL	-78.3226	-49.2475	10.7610	107.7769	-2.9260	-2.8961	2.9651	2.8076
BALM	-78.3314	-49.8943	5.5097	66.1361	-2.9101	-2.9001	2.7992	2.6533

Tabla 1. Comparación

V. Conclusiones

De las comparaciones realizadas podemos concluir que para los parámetros dados, las mejores aproximaciones a la solución del problema es mediante al algoritmo de búsqueda aleatoria simple. La búsqueda aleatoria localizada mejorada es menos efectiva, pero es superior a la búsqueda aleatoria localizada, ya que este último algoritmo es el que da la peores aproximaciones.

Cabe resaltar que este comportamiento está ligado a las condiciones iniciales. Por ejemplo, si en lugar de utilizar una desviación estándar $\sigma = \sqrt{3}$ utilizáramos $\sigma = 2$, el algoritmo de búsqueda aleatoria localizada mejorada sería superior. También, sería interesante variar el vector inicial, ya que al estar más lejano a un mínimo local el algoritmo de búsqueda localizada tendría un mejor rendimiento. Por último, un detalle interesante, es que el comportamiento de la media incrementa casi el doble entre cada algoritmo, es decir, la media del algoritmo BALM es casi el doble que BAS y a su vez BAL es el doble que BALM.

VI. Anexos

```
import random
import numpy as np
import matplotlib.pyplot as plt

#Valores iniciales
x0 = np.array([4, 6.4]) #Vector inicial
N = 500                 #Numero de iteraciones
n = 40                  #Numero de ejecuciones del algoritmo
opt = -78.33            #Solucion optima
```

```
# Definimos el espacio de busqueda
limX= [-8, 8] #Eje X
limY= [-8, 8] #Eje Y
# Definimos la funcion objetivo
def fobj(x):
    z = 0.5*((x[0]**4 - 16*x[0]**2 +5*x[0])+(x[1]**4 - 16*x[1]**2 +5*x[1]))
    return z
# Funcion para graficar
def graficar():
    xx = np.linspace(limX[0], limX[1])
    yy = np.linspace(limY[0], limY[1])
    X, Y =  np.meshgrid(xx, yy)
    Z = fobj([X,Y])
    plt.contour(X, Y, Z, levels=16, cmap=plt.get_cmap('jet'))
    plt.title('B squeda aleatoria simple')
    plt.xlabel("x1")
    plt.ylabel("x2")
    #plt.show()
    return
# Busqueda aleatoria simple
def BAS(x, N):
    iter = 0
    #graficar()
    #plt.scatter(x[0],x[1], c = "black")
    while iter < N:
        iter += 1
        xhat = np.random.uniform(limX[0], limX[1], 2)
        if fobj(xhat) < fobj(x):
            #plt.pause(1)
            #plt.arrow(x[0], x[1], xhat[0]-x[0], xhat[1]-x[1], head_width=0.25)
            x = xhat
    #plt.show()
    return x, fobj(x)
# Numero de ejecuciones del algoritmo
def ejecuciones(n):
    res = []
    dif = np.zeros(n)
    sol = np.zeros(n)
    print('Ejec', '     x1', '               x2', '               f(x)')
    for i in range(n):
        res.append(BAS(x0,N))
        dif[i] = res[0][1] - opt
        sol[i] = res[0][1]
        print(i+1, res[0][0][0], res[0][0][1], res[0][1])
        res.pop()
    return dif, sol
# Impresion de resultados
muestra = ejecuciones(n)
media = np.mean(muestra[0])
varianza = np.var(muestra[0])
print('La peor solucion es ')
print(np.max(muestra[1]))
print('La mejor solucion es')
print(np.min(muestra[1]))
```

```
print('La media es:')
print(media)
print('La varianza es:')
print(varianza)
print("El intervalo de confianza es")
k = 2.023*(varianza/40)**0.5
print("[ {} , {} ]".format(media - k, media + k))
```

Código 1. Búsqueda aleatoria simple

```
import random
import numpy as np
import matplotlib.pyplot as plt

# Valores iniciales
x0 = np.array([4, 6.4])   #Vector inicial
N = 500                   #Numero de iteraciones
opt = -78.33              #Solucion optima
n = 40                    #Numero de ejecuciones del algoritmo
# Definimos la funcion objetivo
def fobj(x):
    z = 0.5*((x[0]**4 - 16*x[0]**2 +5*x[0])+(x[1]**4 - 16*x[1]**2 +5*x[1]))
    return z
# Definimos el espacio de busqueda
limX= [-8, 8] #Eje X
limY= [-8, 8] #Eje Y
# Funcion para graficar
def graficar():
    xx = np.linspace(limX[0], limX[1])
    yy = np.linspace(limY[0], limY[1])
    X, Y =  np.meshgrid(xx, yy)
    Z = fobj([X,Y])
    plt.contour(X, Y, Z, levels=16, cmap=plt.get_cmap('jet'))
    plt.title('B squeda aleatoria localizada')
    plt.xlabel("x1")
    plt.ylabel("x2")
    #plt.show()
    return
# Busqueda aleatoria localizada
def BAL(x, N):
    iter = 0
    mu = 0
    sigma = 3**0.5
    #graficar()
    #plt.scatter(x[0],x[1], c = "red")
    while iter < N:
        iter +=1
        band = True
        while band:
            d = np.random.normal(mu, sigma, 2)
            xhat = x + d
            if limX[0] <= xhat[0] <= limX[1] and limY[0] <= xhat[1] <= limY[1]:
                band =  False
        if fobj(xhat) < fobj(x):
```

```
            #plt.pause(1)
            #plt.arrow(x[0],x[1], xhat[0] - x[0], xhat[1] - x[1], head_width=0.2)
            x = xhat
    #plt.show()
    return x, fobj(x)
# Numero de ejecuciones del algoritmo
def ejecuciones(n):
    res = []
    dif = np.zeros(n)
    sol = np.zeros(n)
    print('Ejec', '     x1', '               x2', '               f(x)')
    for i in range(n):
        res.append(BAL(x0,N))
        dif[i] = res[0][1] - opt
        sol[i] = res[0][1]
        print(i+1, res[0][0][0], res[0][0][1], res[0][1])
        res.pop()
    return dif, sol
#Impresion de resultados
muestra = ejecuciones(n)
media = np.mean(muestra[0])
varianza = np.var(muestra[0])
print('La peor solucion es ')
print(np.max(muestra[1]))
print('La mejor solucion es')
print(np.min(muestra[1]))
print('La media es:')
print(media)
print('La varianza es:')
print(varianza)
print("El intervalo de confianza es")
k = 2.023*(varianza/40)**0.5
print("[ {} , {} ]".format(media - k, media + k))
```

Código 2. Búsqueda aleatoria localizada

```
import random
import numpy as np
import matplotlib.pyplot as plt

#Valores iniciales
x0 = np.array([4, 6.4]) #Vector inicial
N = 500                 #Numero de iteraciones
n = 40                  #Numero de ejecuciones del algoritmo
opt = -78.33            #Solucion optima
# Definimos el espacio de busqueda
limX= [-8, 8] #Eje X
limY= [-8, 8] #Eje Y
# Definimos la funcion objetivo
def fobj(x):
    z = 0.5*((x[0]**4 - 16*x[0]**2 +5*x[0])+(x[1]**4 - 16*x[1]**2 +5*x[1]))
    return z
# Funcion para graficar
def graficar():
```

```
    xx = np.linspace(limX[0], limX[1])
    yy = np.linspace(limY[0], limY[1])
    X, Y =  np.meshgrid(xx, yy)
    Z = fobj([X,Y])
    plt.contour(X, Y, Z, levels=16, cmap=plt.get_cmap('jet'))
    plt.title('B squeda aleatoria localizada mejorada')
    plt.xlabel("x1")
    plt.ylabel("x2")
    #plt.show()
    return
# Busqueda aleatoria localizada mejorada
def BALM(x, N):
    # Paso 0 Inicializacion
    iter = 0
    mu = 0
    sigma = 3**0.5
    b = np.array([0, 0])
    #graficar()
    #plt.scatter(x[0],x[1], c = "red")
    while iter < N:
        iter += 1
        # Paso 1 Generar el vector independiente dk
        band = True
        while band:
            d = np.random.normal(mu, sigma, 2)
            xhat = x + b + d
            # Verificamos que el vector se encuentre en el espacio de busqueda
            if limX[0] <= xhat[0] <= limX[1] and limY[0] <= xhat[1] <= limY[1]:
                band = False
        # Paso 2
        if fobj(xhat) < fobj(x):
            #plt.pause(1)
            #plt.arrow(x[0],x[1], xhat[0] - x[0], xhat[1] - x[1], head_width=0.2)
            x = xhat
            b = 0.2*b + 0.4*d
        # Paso 3
        else :
            xhat = x + b - d
            if limX[0] <= xhat[0] <= limX[1] and limY[0] <= xhat[1] <= limY[1]:
                if fobj(xhat) < fobj(x):
                    #plt.pause(1)
                    #plt.arrow(x[0],x[1], xhat[0] - x[0], xhat[1] - x[1], head_width=0.2
                    x = xhat
                    b = b - 0.4*d
                # Paso 4
                else:
                    b = 0.5*b
            # Paso 3
            else:
                band = True
                while band:
                    d = np.random.normal(mu, sigma, 2)
                    xhat = x +b + d
                    if limX[0] <= xhat[0] <= limX[1] and limY[0] <= xhat[1] <= limY[1]:
```

```
                    band = False
                if fobj(xhat) < fobj(x):
                    #plt.pause(1)
                    #plt.arrow(x[0],x[1], xhat[0] - x[0], xhat[1] - x[1], head_width=0.2
                    x = xhat
                    b = b - 0.4*d
                else:
                    b = 0.5*b
    #plt.show()
    return x, fobj(x)
# Numero de ejecuciones del algoritmo
def ejecuciones(n):
    res = []
    dif = np.zeros(n)
    sol = np.zeros(n)
    print('Ejec', '     x1          ', '        x2        ', '      f(x)')
    for i in range(n):
        res.append(BALM(x0,N))
        dif[i] = res[0][1] - opt
        sol[i] = res[0][1]
        print(i+1, res[0][0][0], res[0][0][1], res[0][1])
        res.pop()
    return dif, sol

muestra = ejecuciones(n)
media = np.mean(muestra[0])
varianza = np.var(muestra[0])
print('La peor solucion es ')
print(np.max(muestra[1]))
print('La mejor solucion es')
print(np.min(muestra[1]))
print('La media es:')
print(media)
print('La varianza es:')
print(varianza)
print("El intervalo de confianza es")
k = 2.023*(varianza/40)**0.5
print("[ {} , {} ]".format(media - k, media + k))
```

Código 3. Búsqueda aleatoria localizada mejorada