Conclusiones.

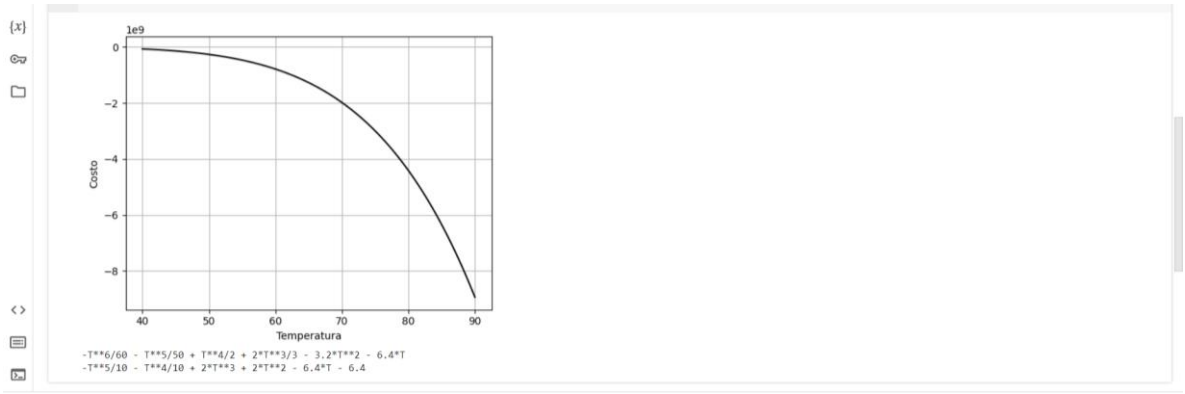
Antes de dar la conclusión mostraremos por cada método, la prueba de las funciones.

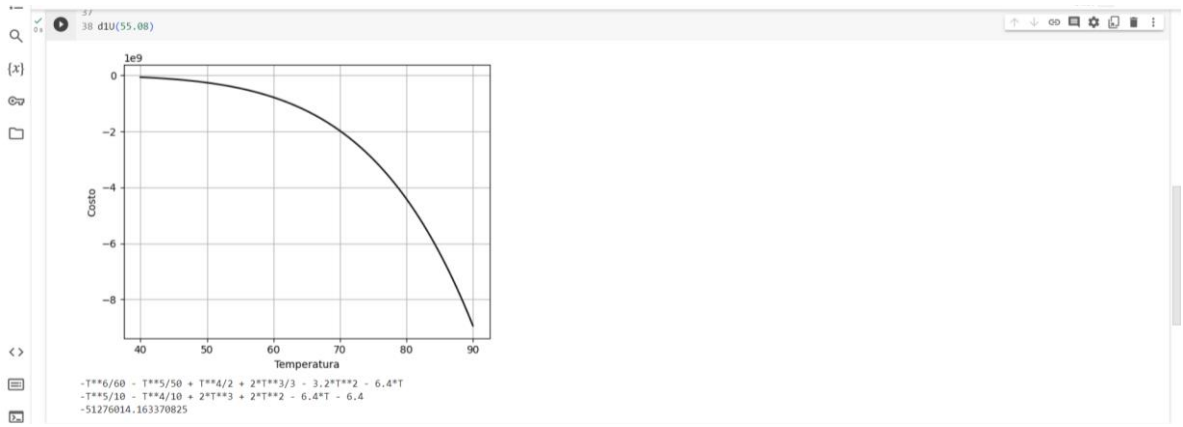
A continuación, mostraremos los métodos de cada función.

```
(-T**6/60) - (T**5/50)+(T**4/2)+(2*T**3/3)-3.2*T**2-6.4*T
```

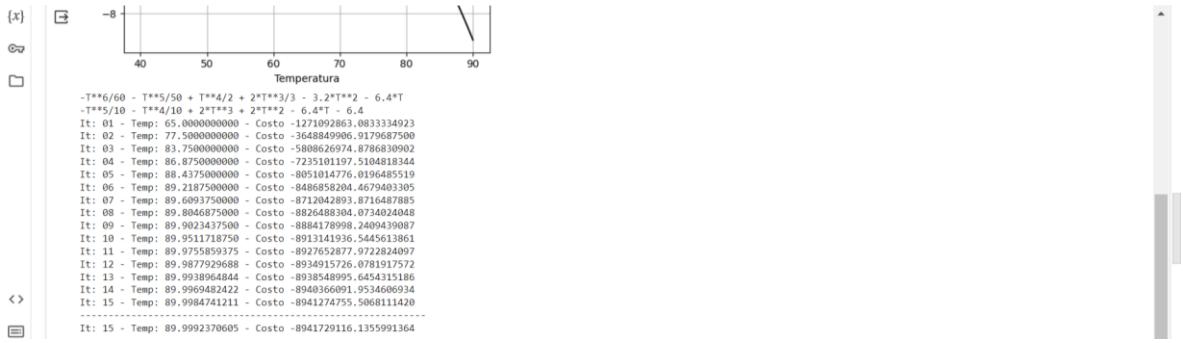
Gradiente.

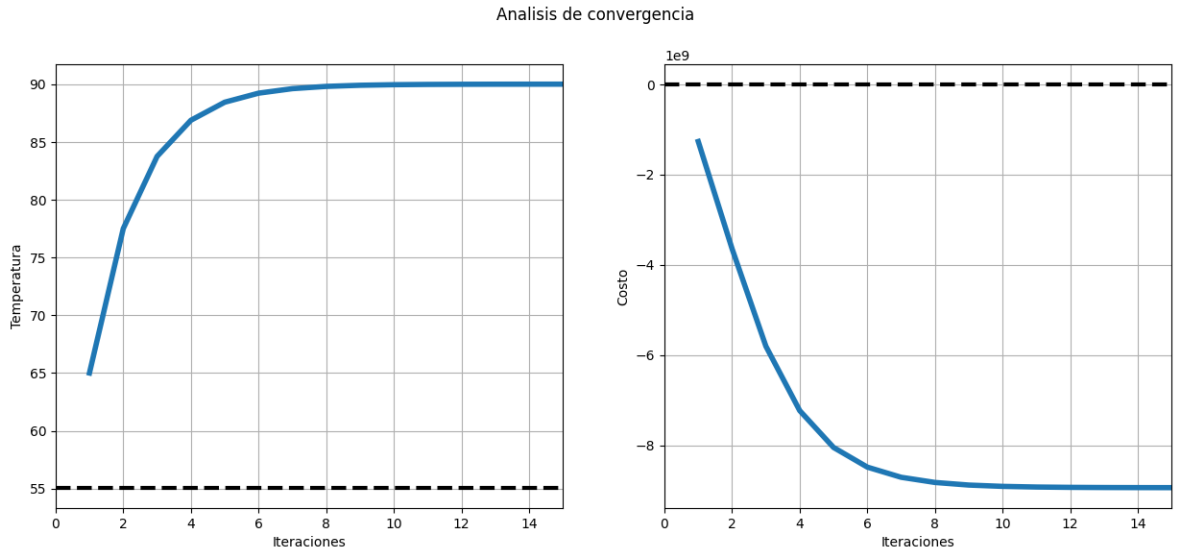
```
import matplotlib.pyplot as plt
import numpy as np
import sympy

a = 40
b = 90
T = np.linspace(a, b, 100)
U = (-T**6/60) - (T**5/50)+(T**4/2)+(2*T**3/3)-3.2*T**2-6.4*T

plt.figure()
plt.plot(T, U, 'k')
plt.xlabel('Temperatura')
plt.ylabel('Costo')
plt.grid()
plt.show()

def calcularGradiente():
    T = sympy.Symbol('T')
    fU = (-T**6/60) - (T**5/50)+(T**4/2)+(2*T**3/3)-3.2*T**2-6.4*T

    d1fU = sympy.diff(fU)

    print(fU)
    print(d1fU)

    return None

calcularGradiente()
```





Método de la bisección con la evaluación del método.



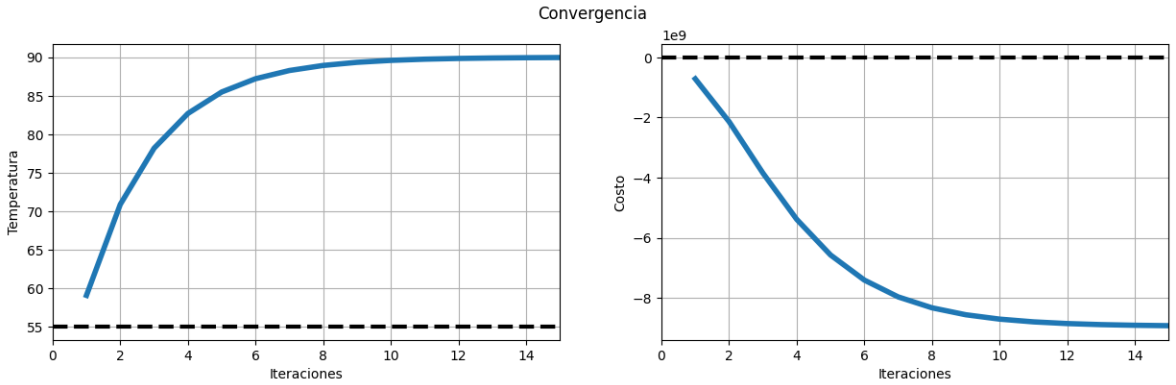


Método de Newton – Raphson.

Método de la secante.

Sección dorada.

```
It: 44 - Temp: 89.9999999681 - Costo: -8942183477.0131931305
It: 45 - Temp: 89.9999999803 - Costo: -8942183484.2655105591
It: 46 - Temp: 89.9999999878 - Costo: -8942183488.7476882935
It: 47 - Temp: 89.9999999925 - Costo: -8942183491.5178318024
It: 48 - Temp: 89.9999999953 - Costo: -8942183493.2298698425
It: 49 - Temp: 89.9999999971 - Costo: -8942183494.2879676819
It: 50 - Temp: 89.9999999982 - Costo: -8942183494.9419059753
It: 51 - Temp: 89.9999999989 - Costo: -8942183495.3460617065
It: 52 - Temp: 89.9999999993 - Costo: -8942183495.5958442688
It: 53 - Temp: 89.9999999996 - Costo: -8942183495.7502212524
It: 54 - Temp: 89.9999999997 - Costo: -8942183495.8456306458
It: 55 - Temp: 89.9999999998 - Costo: -8942183495.9045963287
It: 56 - Temp: 89.9999999999 - Costo: -8942183495.9410419464
It: 57 - Temp: 89.9999999999 - Costo: -8942183495.9635658264
It: 58 - Temp: 90.0000000000 - Costo: -8942183495.9774875641
It: 59 - Temp: 90.0000000000 - Costo: -8942183495.9860858917
It: 60 - Temp: 90.0000000000 - Costo: -8942183495.9913997650
It: 61 - Temp: 90.0000000000 - Costo: -8942183495.9946842194
It: 62 - Temp: 90.0000000000 - Costo: -8942183495.9967155457
It: 63 - Temp: 90.0000000000 - Costo: -8942183495.9979686737
It: 64 - Temp: 90.0000000000 - Costo: -8942183495.9987487793
It: 65 - Temp: 90.0000000000 - Costo: -8942183495.9992198944
It: 66 - Temp: 90.0000000000 - Costo: -8942183495.9995269775
It: 67 - Temp: 90.0000000000 - Costo: -8942183495.9997119904
It: 68 - Temp: 90.0000000000 - Costo: -8942183495.9998226166
It: 69 - Temp: 90.0000000000 - Costo: -8942183495.9998893738
It: 70 - Temp: 90.0000000000 - Costo: -8942183495.9999332428
It: 71 - Temp: 90.0000000000 - Costo: -8942183495.9999580383
It: 72 - Temp: 90.0000000000 - Costo: -8942183495.9999752045
It: 73 - Temp: 90.0000000000 - Costo: -8942183495.9999828339
--------------------------------------------------------------
It: 73 - Temp: 90.0000000000 - Costo: -8942183495.9999828339
```

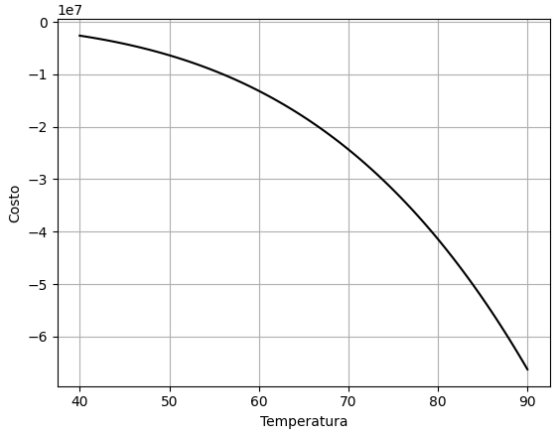


Método de Fibonacci.

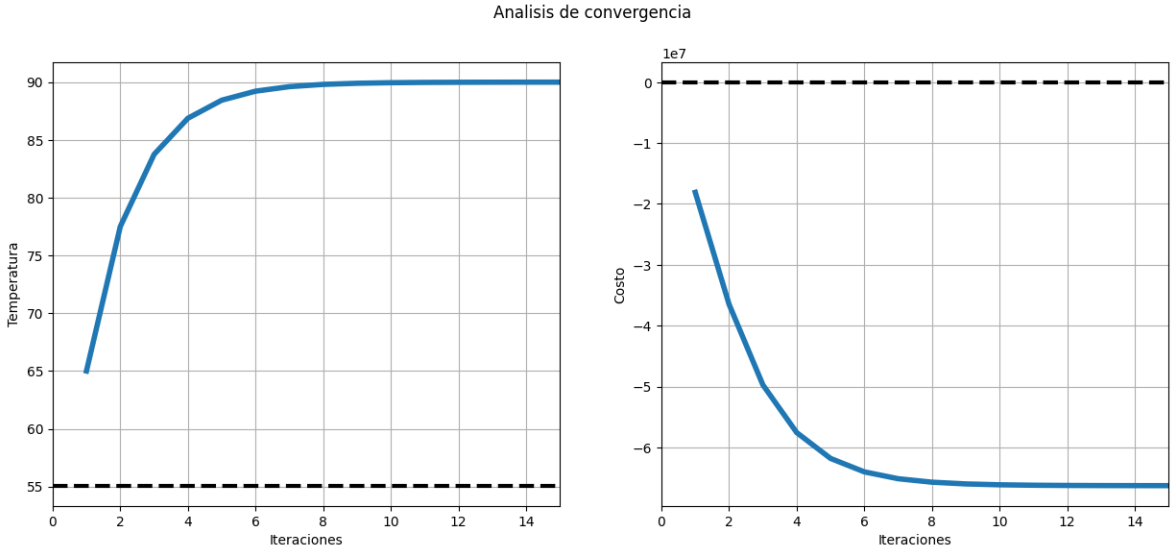
```
It: 31 - Temp: 139.9999770464 - Costo: -126373944744.5605163574
It: 32 - Temp: 120.9016764839 - Costo: -52461313406.8552474976
It: 33 - Temp: 109.0982776089 - Costo: -28340675740.7671775818
It: 34 - Temp: 101.8033759214 - Costo: -18717649257.7776374817
It: 35 - Temp: 97.2948787339 - Costo: -14267006261.7457046509
It: 36 - Temp: 94.5084742339 - Costo: -11986455339.8435935974
It: 37 - Temp: 92.7863815464 - Costo: -10735391116.4061050415
It: 38 - Temp: 91.7220697339 - Costo: -10018077410.2232456207
It: 39 - Temp: 91.0642888589 - Costo: -9595063882.0000858307
It: 40 - Temp: 90.6577579214 - Costo: -9341148137.8633136749
It: 41 - Temp: 90.4065079839 - Costo: -9187037473.8083686829
It: 42 - Temp: 90.2512269839 - Costo: -9092855390.4844398499
It: 43 - Temp: 90.1552580465 - Costo: -9035050900.6912918091
It: 44 - Temp: 90.0959459838 - Costo: -8999479082.1806716919
It: 45 - Temp: 90.0592891091 - Costo: -8977552883.9023704529
It: 46 - Temp: 90.0366339211 - Costo: -8964024018.5475654602
It: 47 - Temp: 90.0226322345 - Costo: -8955671209.5881271362
It: 48 - Temp: 90.0139787330 - Costo: -8950512142.7411308289
It: 49 - Temp: 90.0086305478 - Costo: -8947324886.0382270813
It: 50 - Temp: 90.0053252316 - Costo: -8945355552.1395130157
It: 51 - Temp: 90.0032823626 - Costo: -8944138574.9620780945
It: 52 - Temp: 90.0020199154 - Costo: -8943386579.3754730225
It: 53 - Temp: 90.0012394936 - Costo: -8942921735.7229461670
It: 54 - Temp: 90.0007574683 - Costo: -8942634636.4640197754
It: 55 - Temp: 90.0004590717 - Costo: -8942456912.1994724274
It: 56 - Temp: 90.0002754430 - Costo: -8942347544.8839111328
It: 57 - Temp: 90.0001606751 - Costo: -8942279190.8775730133
It: 58 - Temp: 90.0000918143 - Costo: -8942238178.6827106476
It: 59 - Temp: 90.0000459072 - Costo: -8942210837.3065280914
It: 60 - Temp: 90.0000229536 - Costo: -8942197166.6445579529
It: 61 - Temp: 90.0000000000 - Costo: -8942183495.9999923706
It: 62 - Temp: 90.0000000000 - Costo: -8942183495.9999923706
```

```
-(T**6+T**5-10*T-12)/(T**2+6)
```

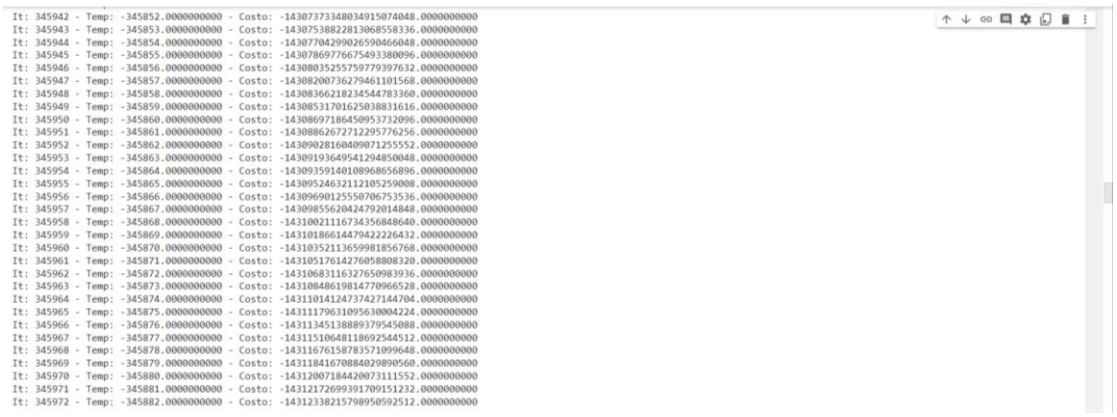
Método del gradiente y método de la bisección.



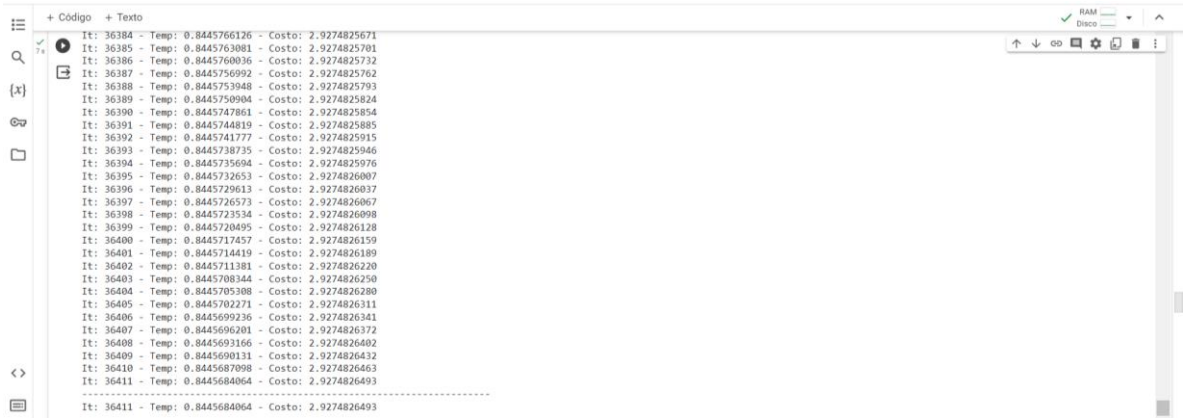

```
(-T**6 - T**5 + 10*T + 12)/(T**2 + 6)
-2*T*(-T**6 - T**5 + 10*T + 12)/(T**2 + 6)**2 + (-6*T**5 - 5*T**4 + 10)/(T**2 + 6)
It: 01 - Temp: 65.0000000000 - Costo -18099546.3455447890
It: 02 - Temp: 77.5000000000 - Costo -36504057.2347264960
It: 03 - Temp: 83.7500000000 - Costo -49741949.8909853324
It: 04 - Temp: 86.8750000000 - Costo -57571118.5405857340
It: 05 - Temp: 88.4375000000 - Costo -61815300.1193809438
It: 06 - Temp: 89.2187500000 - Costo -64023287.7781422958
It: 07 - Temp: 89.6093750000 - Costo -65149200.3535024896
It: 08 - Temp: 89.8046875000 - Costo -65717692.3521447256
It: 09 - Temp: 89.9023437500 - Costo -66003329.3109137192
It: 10 - Temp: 89.9511718750 - Costo -66146496.4109141901
It: 11 - Temp: 89.9755859375 - Costo -66218167.2262745649
It: 12 - Temp: 89.9877929688 - Costo -66254024.4640775546
It: 13 - Temp: 89.9938964844 - Costo -66271958.5422330275
It: 14 - Temp: 89.9969482422 - Costo -66280926.9463397041
It: 15 - Temp: 89.9984741211 - Costo -66285411.4896772057
------------------------------------------------------------
It: 15 - Temp: 89.9992370605 - Costo -66287653.8466703668
```

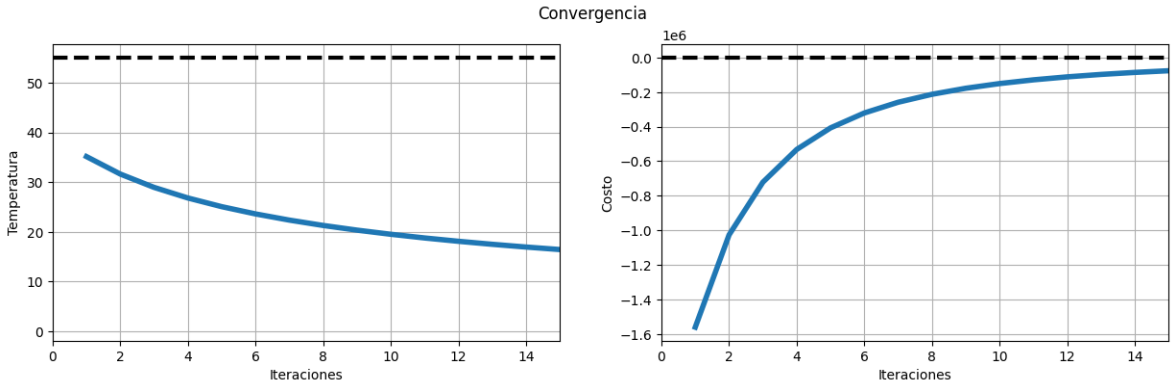


Método de Newton – Raphson.

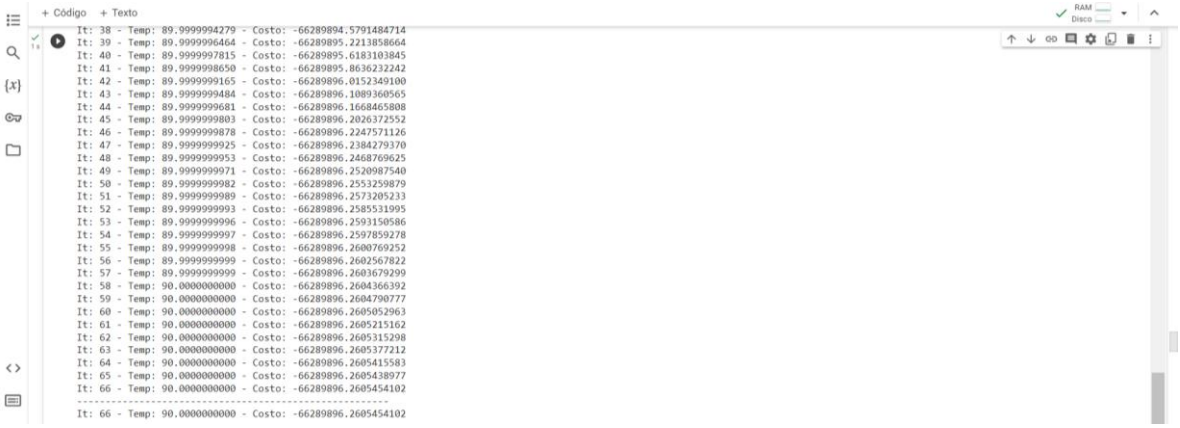


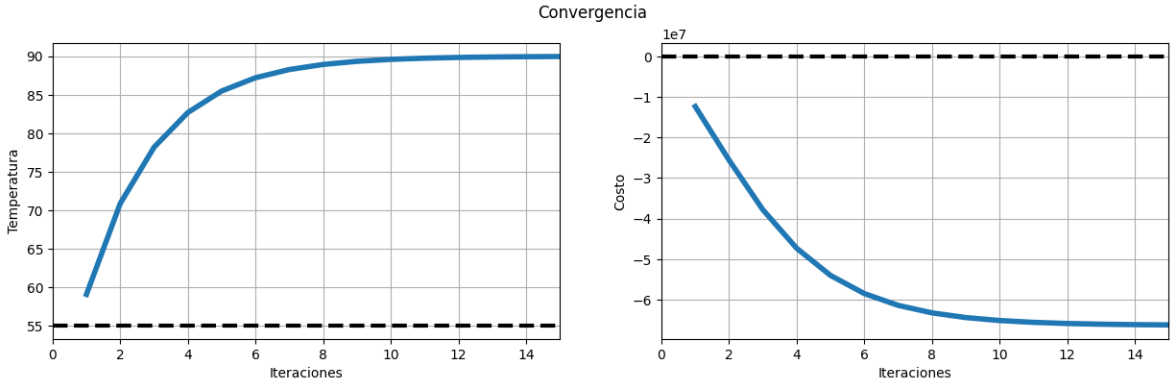
Método de la secante.



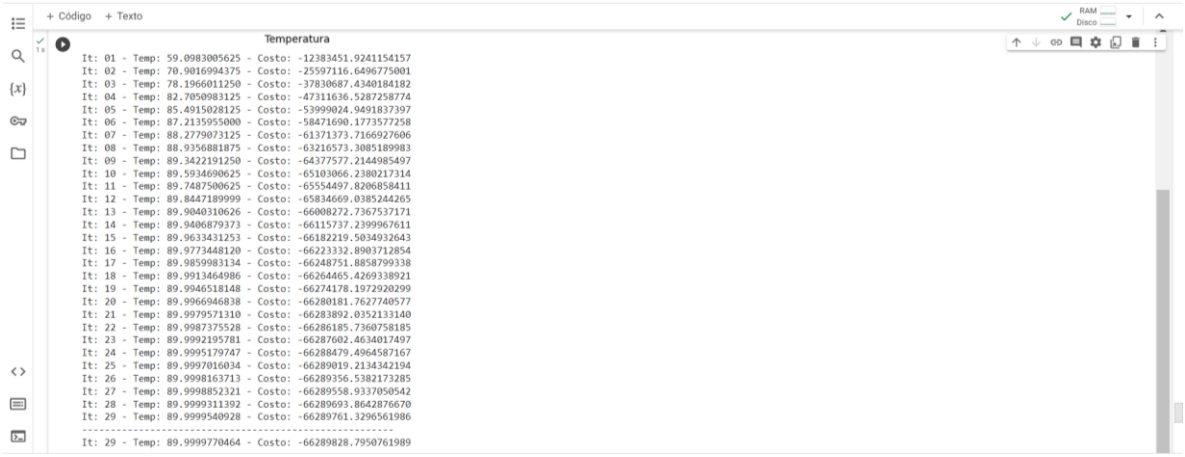


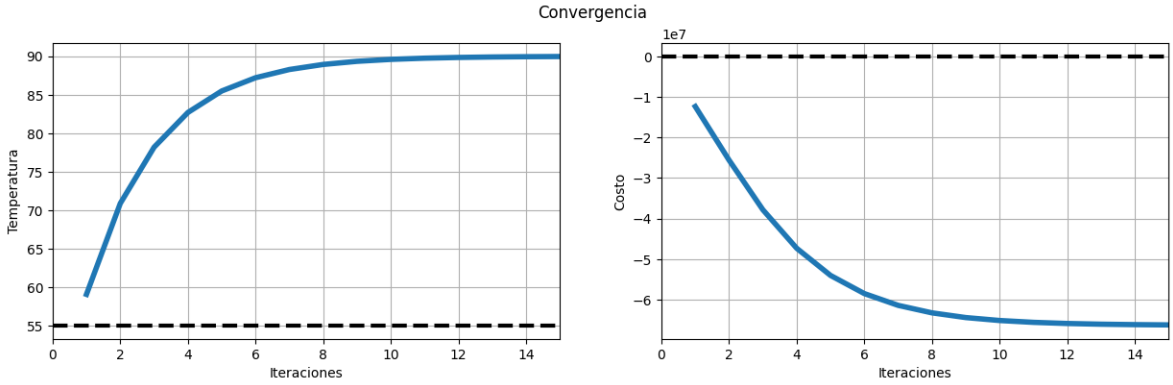
Método de la sección dorada.





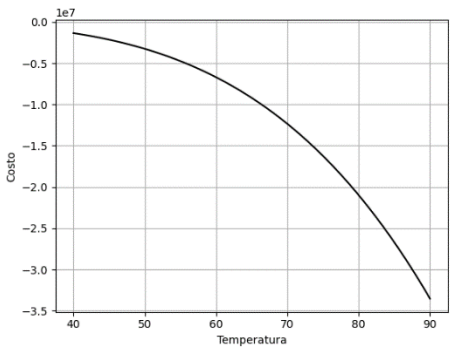
Método de Fibonacci.



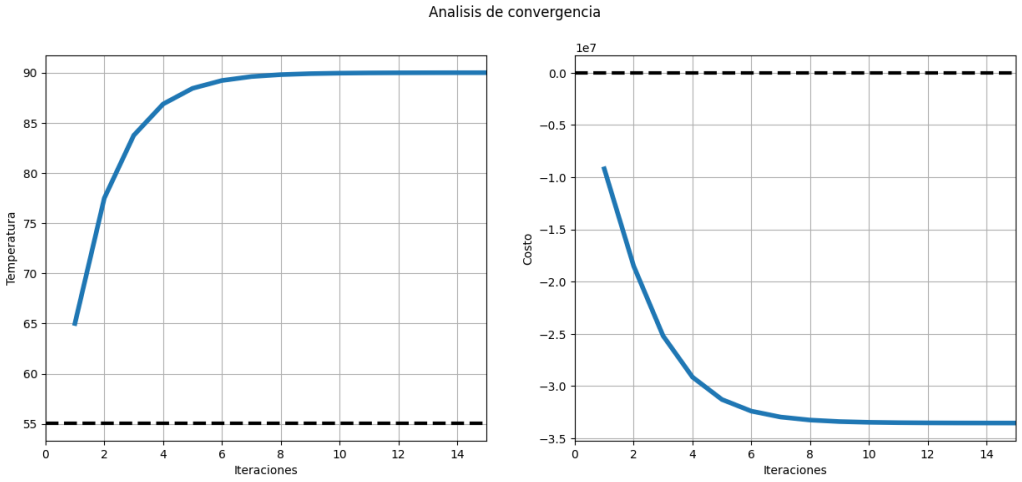



```
5-(T**4+2*T**3+13)**1/2
```

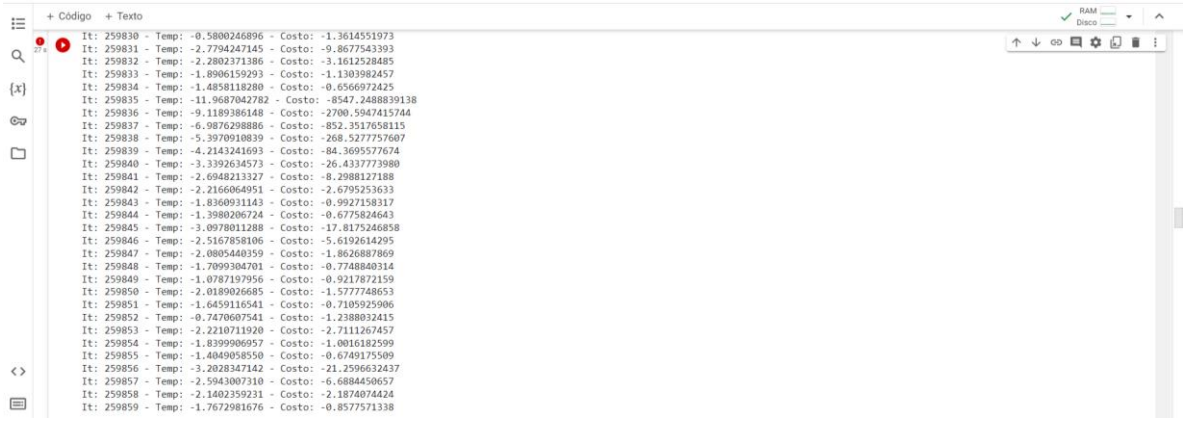
Método del gradiente y bisección.



```
-T**4/2 - T**3 - 3/2
-2*T**3 - 3*T**2
It: 01 - Temp: 65.0000000000 - Costo -9199939.0000000000
It: 02 - Temp: 77.5000000000 - Costo -18503005.4062500000
It: 03 - Temp: 83.7500000000 - Costo -25185965.6113281250
It: 04 - Temp: 86.8750000000 - Costo -29136279.4083251953
It: 05 - Temp: 88.4375000000 - Costo -31277285.6215896606
It: 06 - Temp: 89.2187500000 - Costo -32390864.8416295052
It: 07 - Temp: 89.6093750000 - Costo -32958716.3701734841
It: 08 - Temp: 89.8046875000 - Costo -33245425.7045406699
It: 09 - Temp: 89.9023437500 - Costo -33389479.7898806296
It: 10 - Temp: 89.9511718750 - Costo -33461682.1286890544
It: 11 - Temp: 89.9755859375 - Costo -33497827.1773837544
It: 12 - Temp: 89.9877929688 - Costo -33515910.6784642041
It: 13 - Temp: 89.9938964844 - Costo -33524955.1740517356
It: 14 - Temp: 89.9969482422 - Costo -33529478.1082153432
It: 15 - Temp: 89.9984741211 - Costo -33531739.7469031103
------------------------------------------------------------
It: 15 - Temp: 89.9992370605 - Costo -33532870.6091501750
```



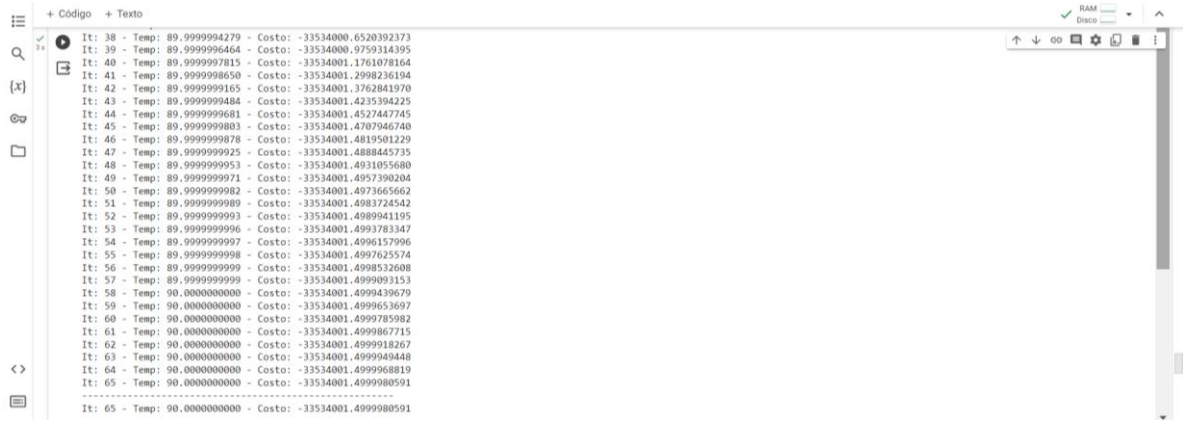
Método de Newton – Raphson.

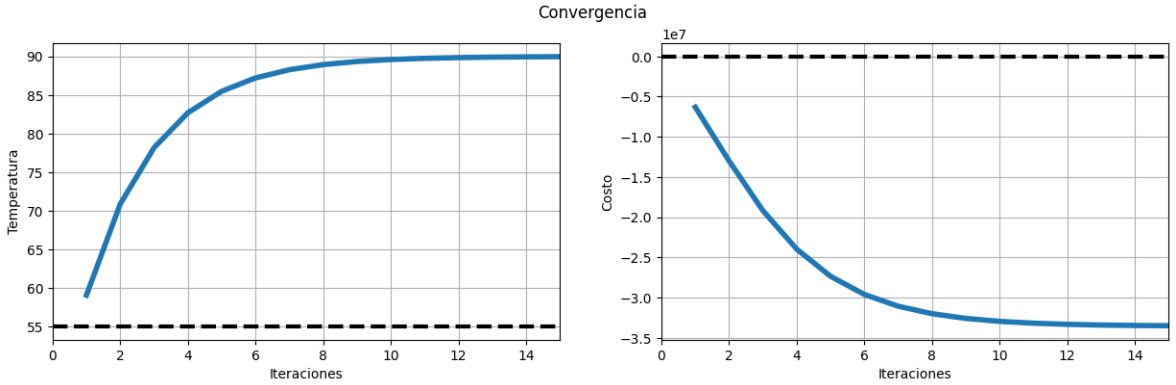


```
It: 259830 - Temp: -0.5800246896 - Costo: -1.3614551973
It: 259831 - Temp: -2.7794247145 - Costo: -9.8677543393
It: 259832 - Temp: -2.2802371386 - Costo: -3.1612528485
It: 259833 - Temp: -1.8906159293 - Costo: -1.1303982457
It: 259834 - Temp: -1.4858118280 - Costo: -0.6566972425
It: 259835 - Temp: -11.96870427782 - Costo: -8547.2488839138
It: 259836 - Temp: -9.1189386148 - Costo: -2700.5947415744
It: 259837 - Temp: -6.9876298886 - Costo: -852.3517658115
It: 259838 - Temp: -5.3970910839 - Costo: -268.5277757607
It: 259839 - Temp: -4.2143241693 - Costo: -84.3695577674
It: 259840 - Temp: -3.3392634573 - Costo: -26.4337773980
It: 259841 - Temp: -2.6948213327 - Costo: -8.2988127188
It: 259842 - Temp: -2.2166064951 - Costo: -2.6795253633
It: 259843 - Temp: -1.8360931143 - Costo: -0.9927158317
It: 259844 - Temp: -1.3980206724 - Costo: -0.6775824643
It: 259845 - Temp: -3.0978011288 - Costo: -17.8175246858
It: 259846 - Temp: -2.5167858106 - Costo: -5.6192614295
It: 259847 - Temp: -2.0805440359 - Costo: -1.8626887869
It: 259848 - Temp: -1.7099304701 - Costo: -0.7748840314
It: 259849 - Temp: -1.0787197956 - Costo: -0.9217872159
It: 259850 - Temp: -2.0189026685 - Costo: -1.5777748653
It: 259851 - Temp: -1.6459116541 - Costo: -0.7105925906
It: 259852 - Temp: -0.7470607541 - Costo: -1.2388032415
It: 259853 - Temp: -2.2210711920 - Costo: -2.7111267457
It: 259854 - Temp: -1.8399906957 - Costo: -1.0016182599
It: 259855 - Temp: -1.4049058550 - Costo: -0.6749175509
It: 259856 - Temp: -3.2028347142 - Costo: -21.2596632437
It: 259857 - Temp: -2.5943007310 - Costo: -6.6884450657
It: 259858 - Temp: -2.1402359231 - Costo: -2.1874074424
It: 259859 - Temp: -1.7672981676 - Costo: -0.8577571338
```

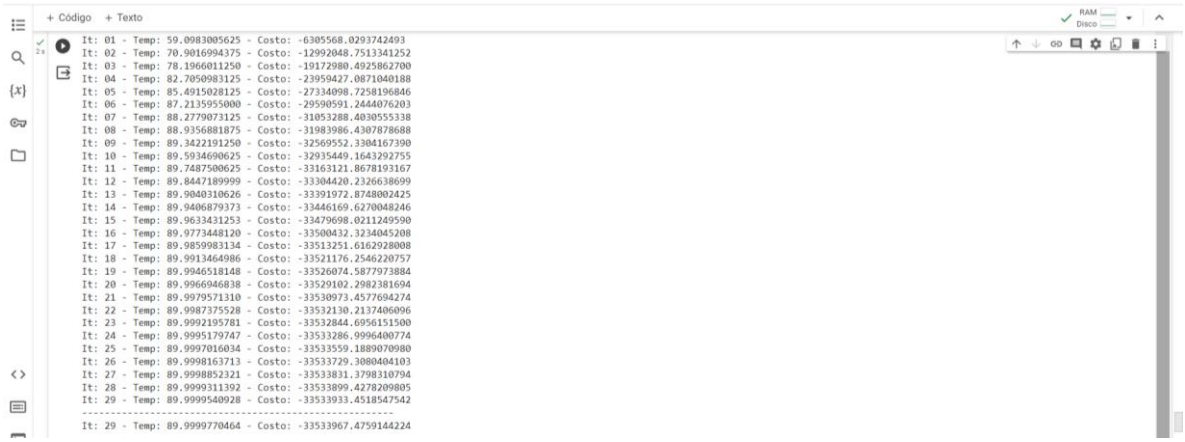
Método de la secante.

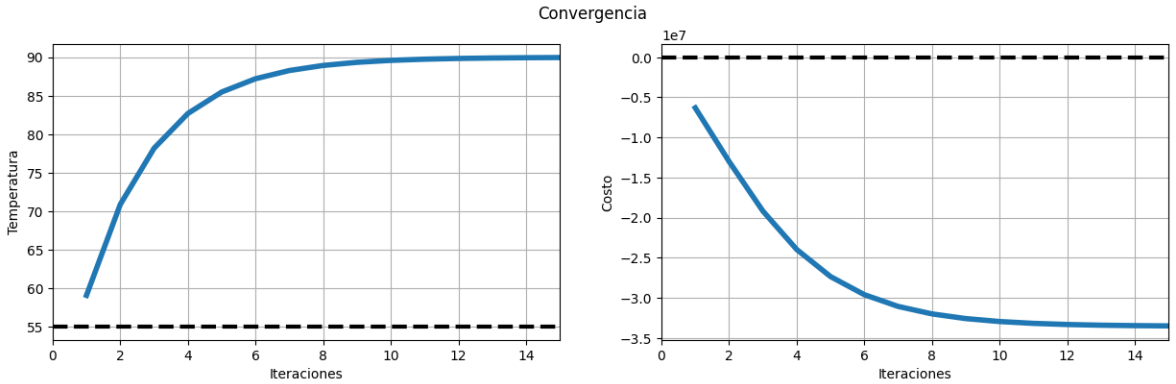
Método de la sección dorada.





Método de Fibonacci.






```
-  sin(3*T+45)**2+0.9*sin(9*T)**3-sin(15*T+50)*cos(2*T-30)
-  sin(2*pi*T)/T - cos(3*pi*T)/T
```

Para estas funciones Python no me dejo compilar los métodos, me da errores de tamaños y que el objeto U no es iterable, se ingreso de las dos formas, pero no resultó.

+ Código + Texto

RAM Disco

```python
import matplotlib.pyplot as plt
import numpy as np
import sympy
import math
from math import sqrt, exp, sin, cos, tan, log

a = 40
b = 90
T = np.linspace(a, b, 100)
U =  sin**2(3*T+45)+0.9*sin**3(9*T)-sin(15*T+50)*cos(2*T-30)

plt.figure()
plt.plot(T, U, 'k')
plt.xlabel('Temperatura')
plt.ylabel('Costo')
plt.grid()
plt.show()

def calcularGradiente():
    T = sympy.Symbol('T')
    fU = (sin(3*T+45)**3)+(0.9*sin(9*T)**3)-(sin(15*T+50)*cos(2*T-30))

    d1fU = sympy.diff(fU)

    print(fU)
    print(d1fU)

    return None
```

```
<>:10: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<>:10: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<>:10: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<>:10: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<ipython-input-39-a3c24affb2d7>:10: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
  U =  sin**2(3*T+45)+0.9*sin**3(9*T)-sin(15*T+50)*cos(2*T-30)
<ipython-input-39-a3c24affb2d7>:10: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
  U =  sin**2(3*T+45)+0.9*sin**3(9*T)-sin(15*T+50)*cos(2*T-30)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-a3c24affb2d7> in <cell line: 10>()
      8 b = 90
      9 T = np.linspace(a, b, 100)
---> 10 U =  sin**2(3*T+45)+0.9*sin**3(9*T)-sin(15*T+50)*cos(2*T-30)
     11
     12 plt.figure()

TypeError: 'int' object is not callable

BUSCAR EN STACK OVERFLOW
```

+ Código + Texto

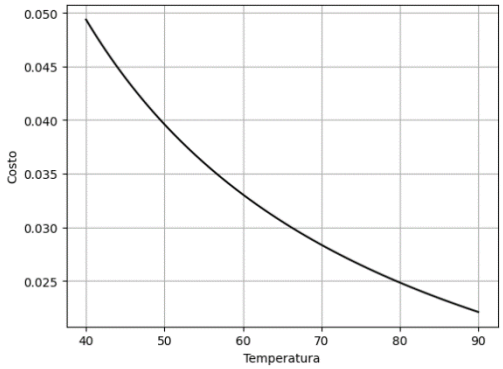
RAM Disco

```python
import numpy as np
import sympy
import math
from math import sqrt, exp, sin, cos, tan, log, pi

a = 40
b = 90
T = np.linspace(a, b, 100)
U = sin(2*pi*T)/T - cos(3*pi*T)/T

plt.figure()
plt.plot(T, U, 'k')
plt.xlabel('Temperatura')
plt.ylabel('Costo')
plt.grid()
plt.show()

def calcularGradiente():
    T = sympy.Symbol('T')
    fU = sin(2*pi*T)/(T) - cos(3*pi*T)/T

    d1fU = sympy.diff(fU)

    print(fU)
    print(d1fU)

    return None

calcularGradiente()
```

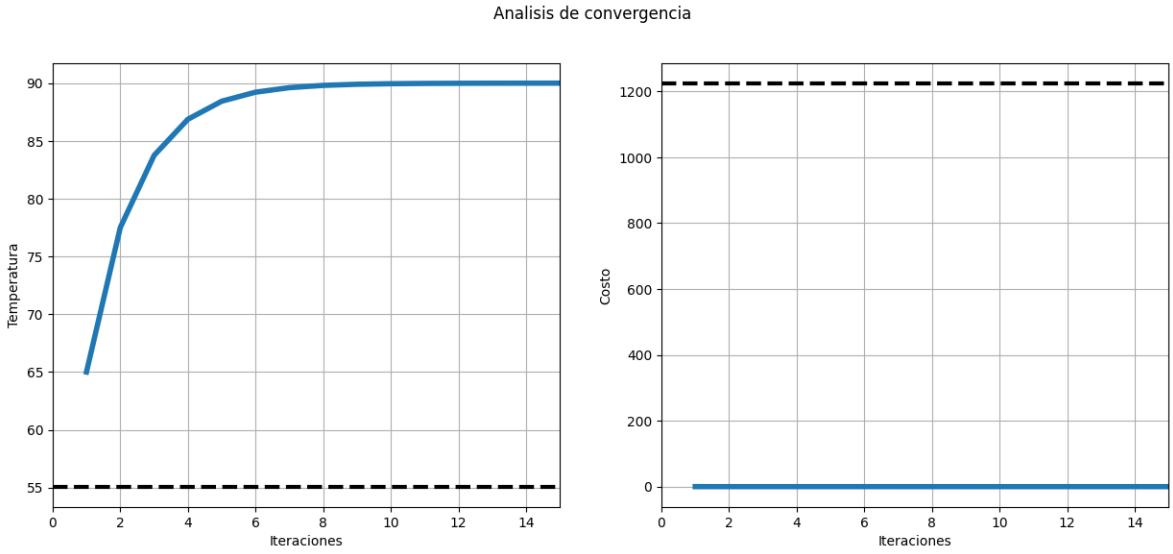
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-fa0beb4bbe17> in <cell line: 10>()
      8 b = 90
      9 T = np.linspace(a, b, 100)
---> 10 U = sin(2*pi*T)/T - cos(3*pi*T)/T
     11
     12 plt.figure()

TypeError: only size-1 arrays can be converted to Python scalars

BUSCAR EN STACK OVERFLOW
```

```
(2*T-1) / T**2
```

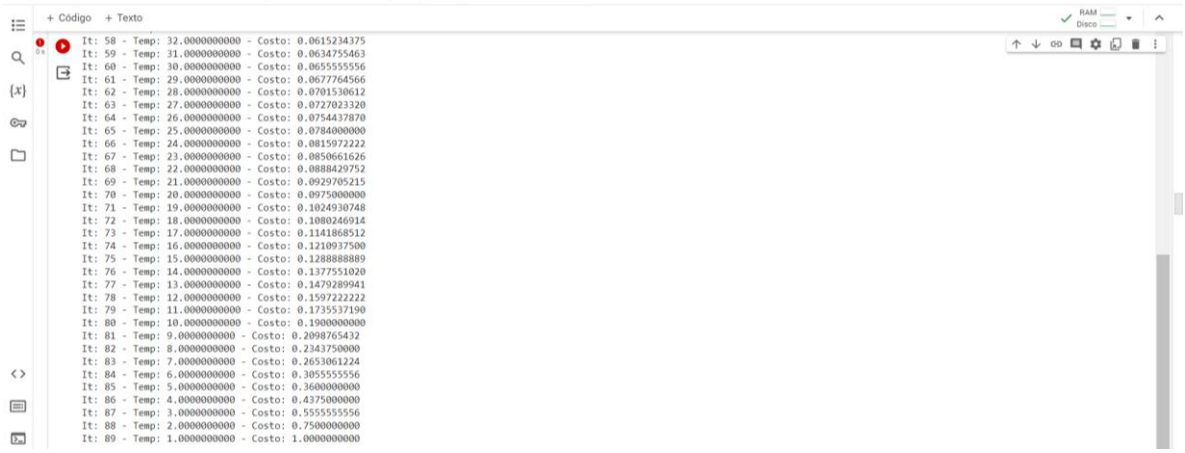
Método del gradiente y Método de la bisección.



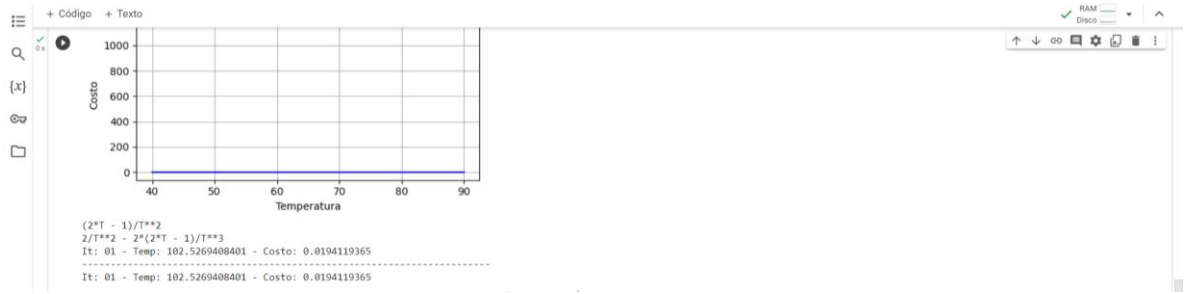

```
(2*T - 1)/T**2
2/T**2 - 2*(2*T - 1)/T**3
It: 01 - Temp: 65.0000000000 - Costo 0.0153846154
It: 02 - Temp: 77.5000000000 - Costo -0.0000000000
It: 03 - Temp: 83.7500000000 - Costo -0.0034972325
It: 04 - Temp: 86.8750000000 - Costo -0.0037343695
It: 05 - Temp: 88.4375000000 - Costo 0.0106092287
It: 06 - Temp: 89.2187500000 - Costo 0.0057094338
It: 07 - Temp: 89.6093750000 - Costo 0.0024923210
It: 08 - Temp: 89.8046875000 - Costo -0.0075144330
It: 09 - Temp: 89.9023437500 - Costo -0.0131400271
It: 10 - Temp: 89.9511718750 - Costo -0.0133180277
It: 11 - Temp: 89.9755859375 - Costo -0.0125194121
It: 12 - Temp: 89.9877929688 - Costo -0.0118906469
It: 13 - Temp: 89.9938964844 - Costo -0.0115195152
It: 14 - Temp: 89.9969482422 - Costo -0.0113199392
It: 15 - Temp: 89.9984741211 - Costo -0.0112166771
------------------------------------------------------------
It: 15 - Temp: 89.9992370605 - Costo -0.0111641815
```

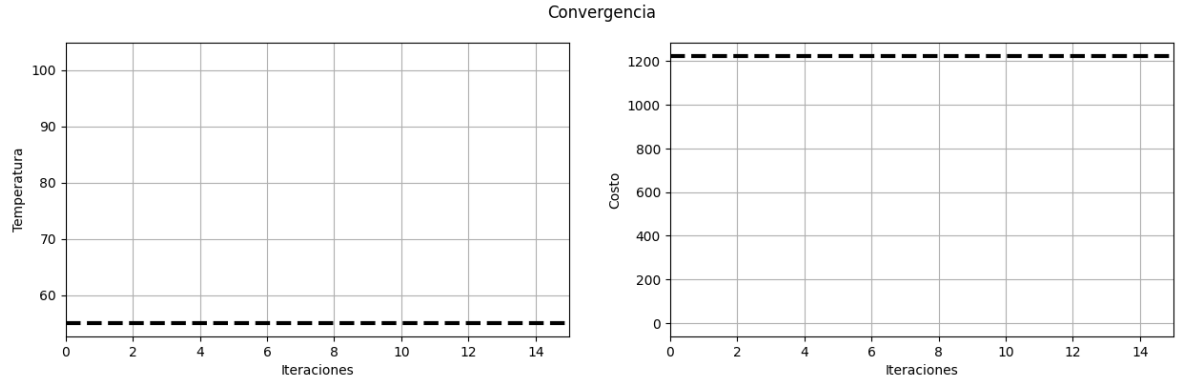


Método de Newton – Raphson.

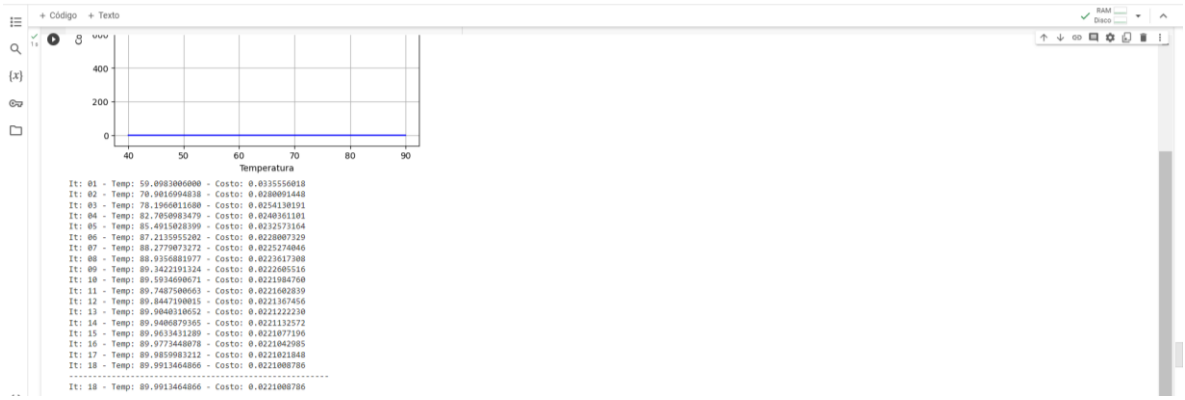


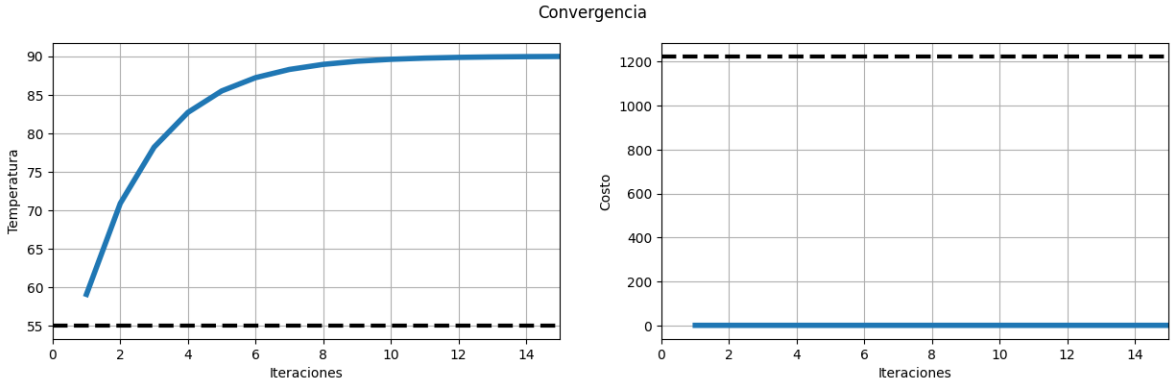
Método de la secante.



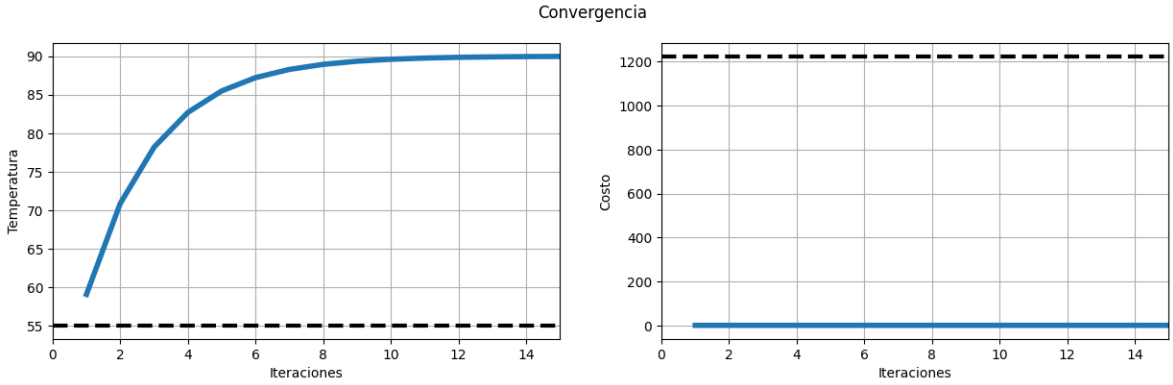


Método de la sección dorada.

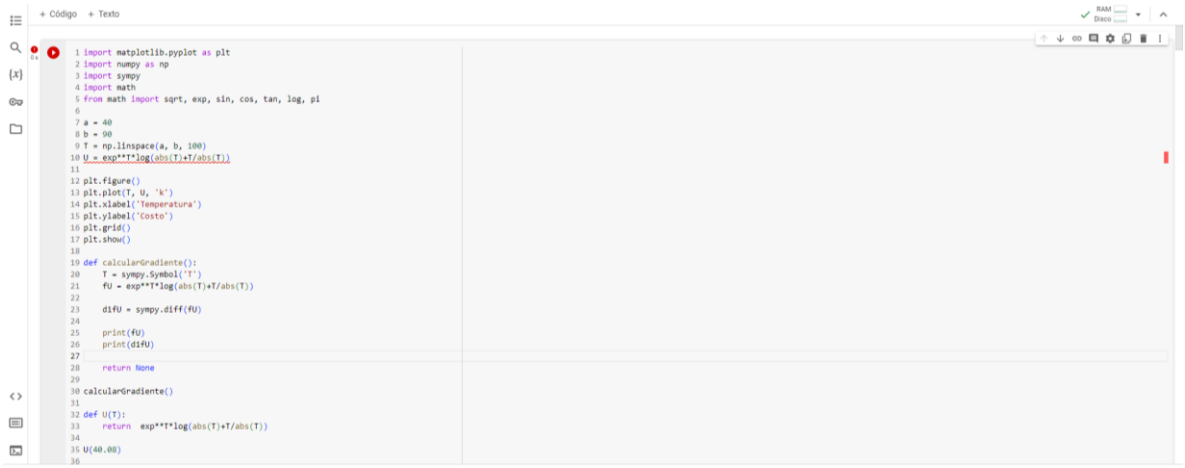




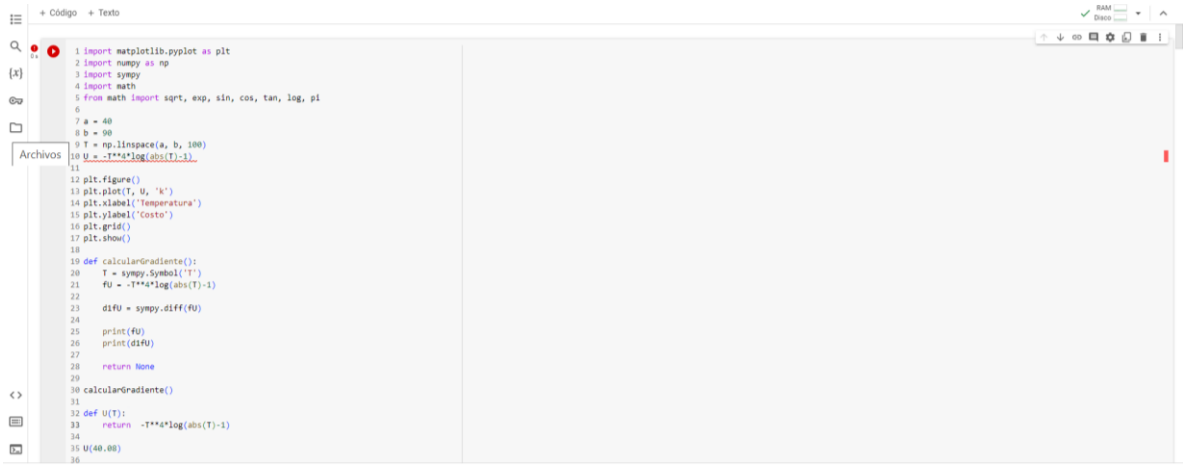
Método de Fibonacci.



Con las funciones `-T**4*log(abs(T)-1)` y `exp**T*log(abs(T)+T/abs(T)`, python no me permitió hacer las comprobaciones con los métodos.

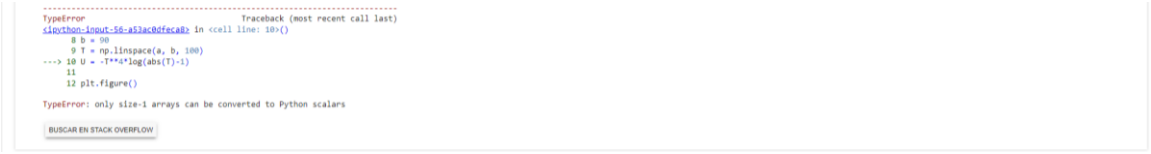


```
import matplotlib.pyplot as plt
import numpy as np
import sympy
import math
from math import sqrt, exp, sin, cos, tan, log, pi

a = 40
b = 90
T = np.linspace(a, b, 100)
U = exp**T*log(abs(T)+T/abs(T))

plt.figure()
plt.plot(T, U, 'k')
plt.xlabel('Temperatura')
plt.ylabel('Costo')
plt.grid()
plt.show()

def calcularGradiente():
    T = sympy.Symbol('T')
    fU = exp**T*log(abs(T)+T/abs(T))

    difU = sympy.diff(fU)

    print(fU)
    print(difU)

    return None

calcularGradiente()

def U(T):
    return  exp**T*log(abs(T)+T/abs(T))

U(40.00)
```



```
import matplotlib.pyplot as plt
import numpy as np
import sympy
import math
from math import sqrt, exp, sin, cos, tan, log, pi

a = 40
b = 90
T = np.linspace(a, b, 100)
U = -T**4*log(abs(T)-1)

plt.figure()
plt.plot(T, U, 'k')
plt.xlabel('Temperatura')
plt.ylabel('Costo')
plt.grid()
plt.show()

def calcularGradiente():
    T = sympy.Symbol('T')
    fU = -T**4*log(abs(T)-1)

    difU = sympy.diff(fU)

    print(fU)
    print(difU)

    return None

calcularGradiente()

def U(T):
    return  -T**4*log(abs(T)-1)

U(40.00)
```

Mismo error en ambas funciones.



```
TypeError                                 Traceback (most recent call last)
<ipython-input-56-a53ac0dfeca8> in <cell line: 10>()
      8 b = 90
      9 T = np.linspace(a, b, 100)
---> 10 U = -T**4*log(abs(T)-1)
     11
     12 plt.figure()

TypeError: only size-1 arrays can be converted to Python scalars

BUSCAR EN STACK OVERFLOW
```

A continuación, daremos una conclusión sobre el código y los test que hicimos con los mismos, sobre el código es muy eficiente y elegante, tanto en su programación, así como se muestra en la terminal, es muy amigable, ya que muestra las soluciones numéricas y graficas. Es una gran herramienta para seguir probando funciones, analizar su convergencia y resultados. Tiene algunas sutilezas, como cuando las funciones no son bien portadas, el programa tiene errores, asi como no hay una alternativa cuando en algún método se requiere una segunda derivada y esta es cero, pero en términos generales están muy bien los programas.

Sobre los test a las funciones, pudimos notar que para algunas funciones, los resultados de convergencia son mas rápidos en un método de convergencia que en otro, en otros, no tenía un resultado concreto y en las funciones trigonométricas complejas, no había manera de ver su convergencia, pero en las demás funciones nos dimos cuenta de ello, mientras unos métodos para encontrar el resultado, necesitaban miles de iteraciones, en otro método eran menos de cien iteraciones, también nos dimos cuenta que el resultado no solo depende de la complejidad de la función, sino también del método que se usa, ya que unos son más óptimos que otros, un ejemplo es con el método Newton – Raphson, converge más rápido que otros métodos. Esas son mis conclusiones.