Apéndice A

Tutorial de Python

A.1. Generalidades

Python es un lenguaje de programación de alto nivel, multi-paradigma y multi-propósito. Es un lenguaje interpretado donde se enfatiza la legibilidad para construir expresiones efectivas en pocas líneas de código. Estos últimos aspectos se pueden evidenciar con el tradicional programa "Hola Mundo", que utiliza solamente una línea de código en Python:

```
1 print("Hola Mundo")
```

Al ejecutar el anterior código, se obtiene, como es de esperar:

Salida

```
Hola Mundo
```

A.2. Funciones

La construcción de las funciones se hace con la palabra clave `def`, seguida del nombre de la función y sus argumentos entre paréntesis. La construcción termina con `:` y en la siguiente línea, *con un sangrado* (por convención de cuatro espacios en blanco) se prosigue al bloque donde se encuentra el cuerpo de la función. Como se verá más adelante en las estructuras de control, el sangrado es fundamental en Python.

```
1 def␣f(entrada):
2 ␣␣␣␣return␣entrada**3
3 print(f(7))
```

Salida

```
343
```

A.3. Estructuras básicas de control

A.3.1. if

```
x = 5
y = 8
if x == y:
    print("x es igual a y")
elif x < y:
    print("x es menor a y")
else:
    print("x es mayor a y")
```

Salida

```
x es menor a y
```

A.3.2. while

```
i = 1
while i < 5:
    print(6*i + 1)
    i += 1
```

Salida

```
7
13
19
25
```

A.3.3. for

```
for i in range(4):
    print(i*i + 3)
```

Salida

```
3
4
```

```
7
12
```

A.4. Ejemplos

A.4.1. Factorial

```
def␣factorial(n):
␣␣␣␣if␣n␣==␣0:
␣␣␣␣␣␣␣␣return␣1
␣␣␣␣else:
␣␣␣␣␣␣␣␣return␣n*factorial(n-1)

print(factorial(5))
print(factorial(10))
```

Salida

```
120
3628800
```

A.4.2. GCD

```
def␣gcd(a,␣b):
␣␣␣␣if␣a␣==␣b:
␣␣␣␣␣␣␣␣return␣a
␣␣␣␣if␣b␣<␣a:
␣␣␣␣␣␣␣␣return␣gcd(a-b,␣b)
␣␣␣␣if␣b␣>␣a:
␣␣␣␣␣␣␣␣return␣gcd(a,␣b-a)

print(gcd(75,␣60))
print(gcd(7,␣13))
print(gcd(1024,␣888))
```

Salida

```
15
1
8
```

A.4.3. ¿Es palíndromo?

```
def espalin(palabra):
    if len(palabra) < 2:
        return True
    elif palabra[0] != palabra[-1]:
        return False
    else:
        return espalin(palabra[1:-1])

print(espalin("python"))
print(espalin("no"))
print(espalin("y"))
print(espalin("anilina"))
```

Salida

```
False
False
True
True
```

A.4.4. NumPy

`numpy` es un poderoso módulo de `Python` para cálculo científico con arreglos de datos multidimensionales. Brinda a `Python` de una gran funcionalidad y es ampliamente usado por la comunidad científica. Recientemente ha contribuido a la generación de la primera imagen de un agujero negro y también en la detección de ondas gravitacionales.

```
import numpy as np
from numpy.linalg import solve, inv

a = np.array([[-1.1, 2.5], [1.3, 4.2]])  # matriz forma (2,2)
print(a)
print(a.T)  # transpuesta
print(inv(a))  # inversa

b = np.array([[2], [-3]]) # vector forma (2,1)
print(b)
s = solve(a, b)  # solucionar el sistema de ecuaciones
print(s)
```

Salida

```
[[-1.1  2.5]
 [ 1.3  4.2]]
```

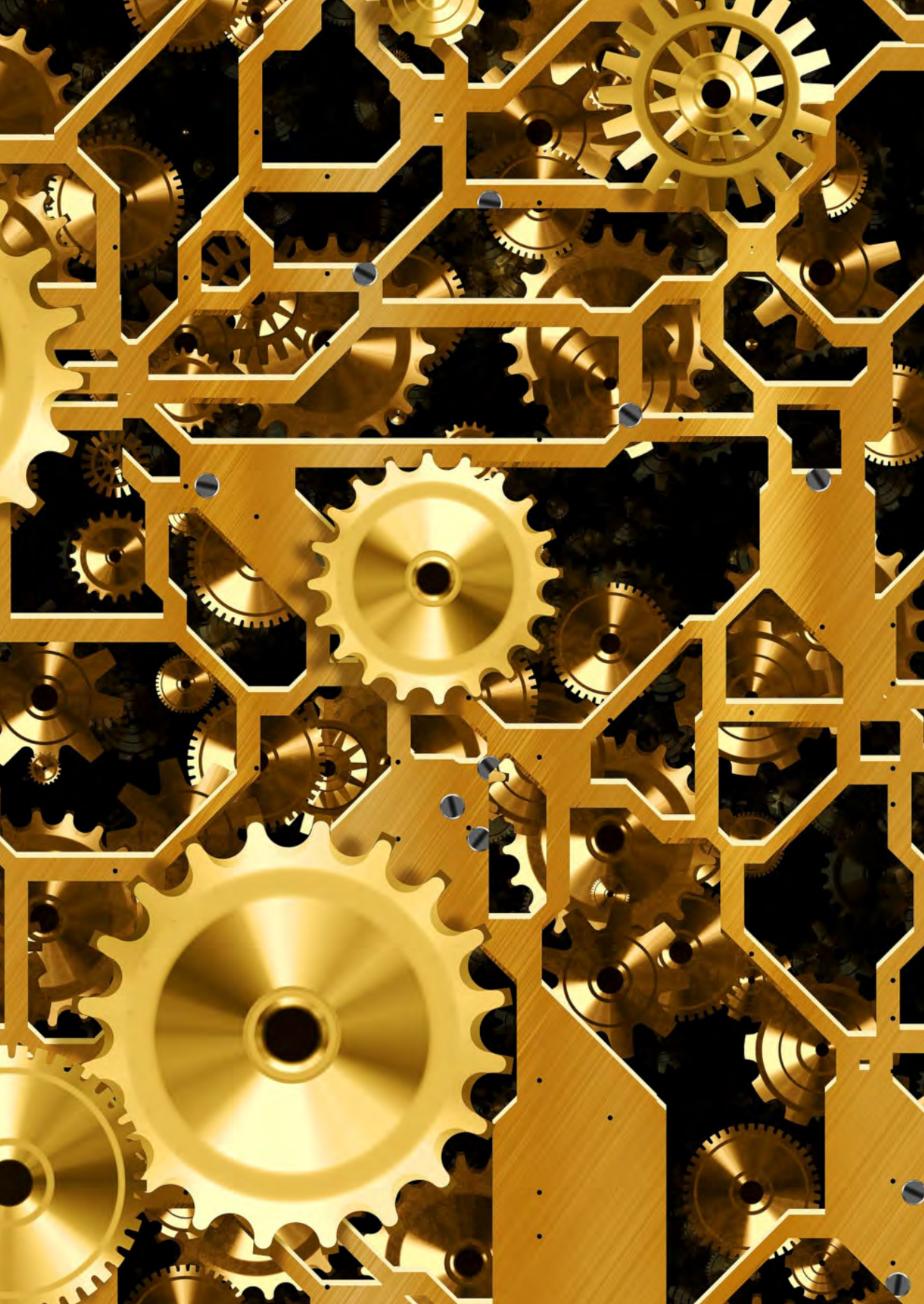
```
[[-1.1  1.3]
 [ 2.5  4.2]]
[[-0.53367217  0.31766201]
 [ 0.16518424  0.13977128]]
[[ 2]
 [-3]]
[[-2.02033037]
 [-0.08894536]]
```

Ejercicios 20

1. Calcular el valor de la expresión $\ln\left(4+\cos\left(\frac{\pi}{4}\right)\right)+\sqrt{\arctan\left(7+\frac{\pi}{3}\right)}$
2. Diseñar una función que imprima los n primeros números triangulares, esto es, números de la forma $\dfrac{i(i+1)}{2}$ donde i es un entero positivo.
3. Dado un entero positivo n, diseñar una función que retorne la suma de sus dígitos.
4. Encontrar la solución del sistema de ecuaciones

$$\begin{aligned} 2x+y-z &= 2\\ 2x-y+2z &= -1\\ x+y-z &= 3 \end{aligned}$$

5. Dada la función $f(n)=\begin{cases} n/2 & \text{si } n \text{ es par}\\ 3n+1 & \text{si } n \text{ es impar}\end{cases}$, diseñar un procedimiento que retorne el i-ésimo término de la sucesión $a_i=\begin{cases} m & \text{para } i=0\\ f(a_{i-1}) & \text{para } i>0\end{cases}$, donde m es un entero positivo arbitrario.



Apéndice B

Compendio de programas

B.1. Búsqueda de raíces

B.1.1. Bisección

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ----------------------------------------------------------------------

# Implementación del método de bisección y algunos casos de salida.

from math import *


```

```
25 def pol(x):
26     """Función de prueba"""
27     return x**3 + 4*x**2 - 10  # retorna pol(x) = x^3 + 4x^2 - 10
28
29
30 def trig(x):
31     """Función de prueba"""
32     return x*cos(x-1) - sin(x)  # retorna trig(x) = x cos(x - 1) - sin(x)
33
34
35 def bisec(f, a, b, tol, n):
36     """
37     Implementación método de bisección
38     Entradas:
39     f -- función
40     a -- inicio intervalo
41     b -- fin intervalo
42     tol -- tolerancia
43     n -- número máximo de iteraciones
44
45     Salida:
46     p aproximación a cero de f
47     None en caso de iteraciones agotadas
48     """
49     i = 1
50     while i <= n:
51         p = a + (b - a)/2
52         print("i = {0:<2}, p = {1:.12f}".format(i, p))
53         if abs(f(p)) <= 1e-15 or (b - a)/2 < tol:
54             return p
55         i += 1
56         if f(a)*f(p) > 0:
57             a = p
58         else:
59             b = p
60     print("Iteraciones agotadas: Error!")
61     return None
62
63
64 # pol(x), a = 1, b = 2, TOL = 10^-8, N_0 = 100
65 print("Bisección función pol(x):")
66 bisec(pol, 1, 2, 1e-8, 100)
67
68 # trig(x), a = 4, b = 6, TOL = 10^-8, N_0 = 100
69 print("Bisección función trig(x):")
```

```
70 bisec(trig, 4, 6, 1e-8, 100)
```

Salida

```
Bisección función pol(x):
i = 1 , p = 1.500000000000
i = 2 , p = 1.250000000000
i = 3 , p = 1.375000000000
i = 4 , p = 1.312500000000
i = 5 , p = 1.343750000000
i = 6 , p = 1.359375000000
i = 7 , p = 1.367187500000
i = 8 , p = 1.363281250000
i = 9 , p = 1.365234375000
i = 10, p = 1.364257812500
i = 11, p = 1.364746093750
i = 12, p = 1.364990234375
i = 13, p = 1.365112304688
i = 14, p = 1.365173339844
i = 15, p = 1.365203857422
i = 16, p = 1.365219116211
i = 17, p = 1.365226745605
i = 18, p = 1.365230560303
i = 19, p = 1.365228652954
i = 20, p = 1.365229606628
i = 21, p = 1.365230083466
i = 22, p = 1.365229845047
i = 23, p = 1.365229964256
i = 24, p = 1.365230023861
i = 25, p = 1.365229994059
i = 26, p = 1.365230008960
i = 27, p = 1.365230016410
Bisección función trig(x):
i = 1 , p = 5.000000000000
i = 2 , p = 5.500000000000
i = 3 , p = 5.750000000000
i = 4 , p = 5.625000000000
i = 5 , p = 5.562500000000
i = 6 , p = 5.593750000000
i = 7 , p = 5.609375000000
i = 8 , p = 5.601562500000
i = 9 , p = 5.597656250000
i = 10, p = 5.599609375000
i = 11, p = 5.598632812500
i = 12, p = 5.599121093750
i = 13, p = 5.599365234375
```

```
i = 14, p = 5.599243164062
i = 15, p = 5.599304199219
i = 16, p = 5.599334716797
i = 17, p = 5.599319458008
i = 18, p = 5.599311828613
i = 19, p = 5.599315643311
i = 20, p = 5.599313735962
i = 21, p = 5.599312782288
i = 22, p = 5.599313259125
i = 23, p = 5.599313020706
i = 24, p = 5.599313139915
i = 25, p = 5.599313080311
i = 26, p = 5.599313050508
i = 27, p = 5.599313035607
i = 28, p = 5.599313028157
```

B.1.2. *Regula falsi*

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ------------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ------------------------------------------------------------------------

# Implementación del método de regula falsi y algunos casos de salida.

from math import *


def pol(x):
    """Función de prueba"""
```

```
27      return x**3 + 4*x**2 - 10  # retorna pol(x) = x^3 + 4x^2 - 10
28
29
30  def trig(x):
31      """Función de prueba"""
32      return x*cos(x-1) - sin(x)  # retorna trig(x) = x cos(x - 1) - sin(x)
33
34
35  def pote(x):
36      """Función de prueba"""
37      return pow(7, x) - 13  # retorna pote(x) = 7^x - 13
38
39
40  def regula(f, p0, p1, tol, n):
41      """
42      Implementación método de regula falsi
43      Entradas:
44      f -- función
45      p0 -- aproximación inicial
46      p1 -- aproximación inicial
47      tol -- tolerancia
48      n -- número máximo de iteraciones
49
50      Salida:
51      p aproximación a cero de f
52      None en caso de iteraciones agotadas
53      """
54      i = 0
55      while i <= n:
56          q0 = f(p0)
57          q1 = f(p1)
58          p = p1-(q1*(p1 - p0))/(q1 - q0)
59          print("Iter = {0:<2}, p = {1:.12f}".format(i, p))
60          if abs(p - p1) < tol:
61              return p
62          i += 1
63          q = f(p)
64          if q*q1 < 0:
65              p0 = p1
66              q0 = q1
67          p1 = p
68          q1 = q
69      print("Iteraciones agotadas: Error!")
70      return None
71
```

```
72
73 # pol(x), a = 1.0, b = 2.0, TOL = 10^-8, N_0 = 100
74 print("Regula falsi función pol(x):")
75 regula(pol, 1, 2, 1e-8, 100)
76
77 # trig(x), a = 4.0, b = 6.0, TOL = 10^-8, N_0 = 100
78 print("Regula falsi función trig(x):")
79 regula(trig, 4, 6, 1e-8, 100)
80
81 # pote(x), a = 0, b = 2.0, TOL = 10^-8, N_0 = 100
82 print("Regula falsi función pote(x):")
83 regula(pote, 0, 2, 1e-8, 100)
```

Salida

```
Regula falsi función pol(x):
Iter = 0 , p = 1.263157894737
Iter = 1 , p = 1.338827838828
Iter = 2 , p = 1.358546341825
Iter = 3 , p = 1.363547440042
Iter = 4 , p = 1.364807031827
Iter = 5 , p = 1.365123717884
Iter = 6 , p = 1.365203303663
Iter = 7 , p = 1.365223301986
Iter = 8 , p = 1.365228327026
Iter = 9 , p = 1.365229589674
Iter = 10, p = 1.365229906941
Iter = 11, p = 1.365229986660
Iter = 12, p = 1.365230006692
Iter = 13, p = 1.365230011725
Regula falsi función trig(x):
Iter = 0 , p = 5.235657374722
Iter = 1 , p = 5.569477410510
Iter = 2 , p = 5.597623035312
Iter = 3 , p = 5.599220749873
Iter = 4 , p = 5.599307996036
Iter = 5 , p = 5.599312749633
Iter = 6 , p = 5.599313008600
Iter = 7 , p = 5.599313022708
Iter = 8 , p = 5.599313023477
Regula falsi función pote(x):
Iter = 0 , p = 0.500000000000
Iter = 1 , p = 0.835058241104
Iter = 2 , p = 1.045169783424
Iter = 3 , p = 1.168847080360
Iter = 4 , p = 1.238197008244
```

```
Iter = 5 , p = 1.275862101477
Iter = 6 , p = 1.295933755010
Iter = 7 , p = 1.306516642232
Iter = 8 , p = 1.312064439413
Iter = 9 , p = 1.314963818299
Iter = 10, p = 1.316476640694
Iter = 11, p = 1.317265325509
Iter = 12, p = 1.317676311539
Iter = 13, p = 1.317890428220
Iter = 14, p = 1.318001965936
Iter = 15, p = 1.318060064553
Iter = 16, p = 1.318090326416
Iter = 17, p = 1.318106088665
Iter = 18, p = 1.318114298545
Iter = 19, p = 1.318118574700
Iter = 20, p = 1.318120801950
Iter = 21, p = 1.318121962020
Iter = 22, p = 1.318122566245
Iter = 23, p = 1.318122880958
Iter = 24, p = 1.318123044876
Iter = 25, p = 1.318123130253
Iter = 26, p = 1.318123174722
Iter = 27, p = 1.318123197884
Iter = 28, p = 1.318123209948
Iter = 29, p = 1.318123216231
```

B.1.3. Newton

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
```

```
17  # along with this program.  If not, see <http://www.gnu.org/licenses/>
18  # ----------------------------------------------------------------------
19
20  # Implementación del método de Newton y algunos casos de salida.
21
22  from math import *
23
24
25  def expo(x):
26      """Función de prueba"""
27      return x**2 + exp(-2*x) - 2*x*exp(-x)
28      # retorna expo(x) = x^2 + e^{-2x} - 2xe^{-x}
29
30
31  def expoprima(x):
32      """Derivada función de prueba"""
33      return 2*x - 2*exp(-2*x) - 2*exp(-x) + 2*x*exp(-x)
34      # retorna expoprima(x) = d/dx expo(x)
35
36
37  def trig(x):
38      """Función de prueba"""
39      return cos(x) - x  # retorna trig(x) = cos(x) - x
40
41
42  def trigprima(x):
43      """Derivada función de prueba"""
44      return -sin(x) - 1  # retorna trigprima(x) = d/dx trig(x)
45
46
47  def newton(f, fprima, p0, tol, n):
48      """
49      Implementación método de Newton
50      Entradas:
51      f -- función
52      fprima -- derivada función f
53      p0 -- aproximación inicial
54      tol -- tolerancia
55      n -- número máximo de iteraciones
56
57      Salida:
58      p aproximación a cero de f
59      None en caso de iteraciones agotadas
```

```
    """
    i = 1
    while i <= n:
        p = p0 - f(p0)/fprima(p0)
        print("Iter = {0:<2}, p = {1:.12f}".format(i, p))
        if abs(p - p0) < tol:
            return p
        p0 = p
        i += 1
    print("Iteraciones agotadas: Error!")
    return None


# trig(x), trigprima(x), p0 = π/4, TOL = 10^-8, N0 = 100
print("Newton función trig(x):")
newton(trig, trigprima, pi/4, 1e-8, 100)

# expo(x), expoprima(x), p0 = 4.0, TOL = 10^-8, N0 = 100
print("Newton función expo(x):")
newton(expo, expoprima, 4, 1e-8, 100)
```

Salida

```
Newton función trig(x):
Iter = 1 , p = 0.739536133515
Iter = 2 , p = 0.739085178106
Iter = 3 , p = 0.739085133215
Iter = 4 , p = 0.739085133215
Newton función expo(x):
Iter = 1 , p = 2.044965524905
Iter = 2 , p = 1.196901548785
Iter = 3 , p = 0.853320871938
Iter = 4 , p = 0.703487910307
Iter = 5 , p = 0.633704735236
Iter = 6 , p = 0.600031374819
Iter = 7 , p = 0.583490458626
Iter = 8 , p = 0.575292817860
Iter = 9 , p = 0.571212060259
Iter = 10, p = 0.569176179405
Iter = 11, p = 0.568159361242
Iter = 12, p = 0.567651232450
Iter = 13, p = 0.567397238091
Iter = 14, p = 0.567270258416
Iter = 15, p = 0.567206772954
Iter = 16, p = 0.567175031318
Iter = 17, p = 0.567159160772
```

```
Iter = 18, p = 0.567151225569
Iter = 19, p = 0.567147257985
Iter = 20, p = 0.567145274200
Iter = 21, p = 0.567144282303
Iter = 22, p = 0.567143786354
Iter = 23, p = 0.567143538379
Iter = 24, p = 0.567143414504
Iter = 25, p = 0.567143352393
Iter = 26, p = 0.567143321397
Iter = 27, p = 0.567143305350
Iter = 28, p = 0.567143297786
```

B.1.4. Secante

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ----------------------------------------------------------------------

# Implementación del método de la secante y algunos casos de salida.

from math import *


def trig(x):
    """Función de prueba"""
    return sin(2/x)  # retorna trig(x) = sin(2/x)


def pol(x):
```

```
31      """Función de prueba"""
32      return x**3 - 2  # retorna pol(x) = x^3 - 2
33
34
35  def secante(f, p0, p1, tol, n):
36      """
37      Implementación método de la secante
38      Entradas:
39      f -- función
40      p0 -- aproximación inicial
41      p1 -- aproximación inicial
42      tol -- tolerancia
43      n -- número máximo de iteraciones
44
45      Salida:
46      p aproximación a cero de f
47      None en caso de iteraciones agotadas
48      """
49      i = 2
50      while i <= n:
51          p = p1 - (f(p1)*(p1 - p0))/(f(p1) - f(p0))
52          print("Iter = {0:<2}, p = {1:.12f}".format(i, p))
53          if abs(p - p1) < tol:
54              return p
55          p0 = p1
56          p1 = p
57          i += 1
58      print("Iteraciones agotadas: Error!")
59      return None
60
61
62  # pol(x), p0 = -3.0, p1 = 3.0, TOL = 10^-8, N0 = 100
63  print("Secante función pol(x):")
64  secante(pol, -3, 3, 1e-8, 100)
65
66  # trig(x), p0 = 1.1, p1 = 0.8, TOL = 10^-8, N0 = 100
67  print("Secante función trig(x):")
68  secante(trig, 1.1, 0.8, 1e-8, 100)
```

Salida

```
Secante función pol(x):
Iter = 2 , p = 0.222222222222
Iter = 3 , p = 0.426937738247
Iter = 4 , p = 6.313558779887
Iter = 5 , p = 0.471912789303
```

```
Iter = 6 , p = 0.515915680782
Iter = 7 , p = 3.059385436425
Iter = 8 , p = 0.682161118454
Iter = 9 , p = 0.823408229082
Iter = 10, p = 1.668975857749
Iter = 11, p = 1.121425751966
Iter = 12, p = 1.221126314305
Iter = 13, p = 1.264621145111
Iter = 14, p = 1.259773686642
Iter = 15, p = 1.259920501483
Iter = 16, p = 1.259921049959
Iter = 17, p = 1.259921049895
Secante función trig(x):
Iter = 2 , p = 0.316169553146
Iter = 3 , p = 0.279163965987
Iter = 4 , p = 0.318328356367
Iter = 5 , p = 0.318309856297
Iter = 6 , p = 0.318309886186
Iter = 7 , p = 0.318309886184
```

B.1.5. Punto fijo

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ---------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ---------------------------------------------------------------------

# Implementación del método del punto fijo y algunos casos de salida.

from math import *
```

```
23
24
25  def pote(x):
26      """Función de prueba"""
27      return pow(2, -x)  # retorna pote(x) = 2^{-x}
28
29
30  def pol(x):
31      """Función de prueba"""
32      return (x**2 - 1)/3  # retorna pol(x) = (x^2-1)/3
33
34
35  def puntofijo(f, p0, tol, n):
36      """
37      Implementación método de punto fijo
38      Entradas:
39      f -- función
40      p0 -- aproximación inicial
41      tol -- tolerancia
42      n -- número máximo de iteraciones
43
44      Salida:
45      p aproximación a punto fijo de f
46      None en caso de iteraciones agotadas
47      """
48      i = 1
49      while i <= n:
50          p = f(p0)
51          print("Iter = {0:<2}, p = {1:.12f}".format(i, p))
52          if abs(p - p0) < tol:
53              return p
54          p0 = p
55          i += 1
56      print("Iteraciones agotadas: Error!")
57      return None
58
59
60  # pol(x), p_0 = 0.9, TOL = 10^{-8}, N_0 = 100
61  print("Punto fijo función pol(x):")
62  puntofijo(pol, 0.9, 1e-8, 100)
63
64  # pote(x), p_0 = 0.5, TOL = 10^{-8}, N_0 = 100
65  print("Punto fijo función pote(x):")
66  puntofijo(pote, 0.5, 1e-8, 100)
```

Salida

```
Punto fijo función pol(x):
Iter = 1 , p = -0.063333333333
Iter = 2 , p = -0.331996296296
Iter = 3 , p = -0.296592819749
Iter = 4 , p = -0.304010899758
Iter = 5 , p = -0.302525790943
Iter = 6 , p = -0.302826048605
Iter = 7 , p = -0.302765461429
Iter = 8 , p = -0.302777691789
Iter = 9 , p = -0.302775223118
Iter = 10, p = -0.302775721422
Iter = 11, p = -0.302775620839
Iter = 12, p = -0.302775641142
Iter = 13, p = -0.302775637044
Punto fijo función pote(x):
Iter = 1 , p = 0.707106781187
Iter = 2 , p = 0.612547326536
Iter = 3 , p = 0.654040860042
Iter = 4 , p = 0.635497845813
Iter = 5 , p = 0.643718641723
Iter = 6 , p = 0.640061021177
Iter = 7 , p = 0.641685807043
Iter = 8 , p = 0.640963537178
Iter = 9 , p = 0.641284509067
Iter = 10, p = 0.641141851472
Iter = 11, p = 0.641205252450
Iter = 12, p = 0.641177074529
Iter = 13, p = 0.641189597767
Iter = 14, p = 0.641184031979
Iter = 15, p = 0.641186505614
Iter = 16, p = 0.641185406241
Iter = 17, p = 0.641185894842
Iter = 18, p = 0.641185677690
Iter = 19, p = 0.641185774200
Iter = 20, p = 0.641185731307
Iter = 21, p = 0.641185750370
Iter = 22, p = 0.641185741898
```

B.2. Interpolación

B.2.1. Lagrange

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ------------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ------------------------------------------------------------------------

# Implementación del interpolador de Lagrange y algunos casos de salida.

from math import *


def LagrangePol(datos):
    """
    Implementación del interpolador de Lagrange
    Entradas:
    datos -- lista de puntos (x, y) en el plano

    Salida:
    P -- función de interpolación
    """

    def L(k, x):
        """Implementación funciones L_k(x)"""
        # pol L_k(x) = \prod_{i≠k} (x-x_i)/(x_k-x_i)
        out = 1
        for i, p in enumerate(datos):
            if i != k:
```

```
                out *= (x - p[0])/(datos[k][0] - p[0])
        return out

    def P(x):
        """Implementación polinomio P(x)"""
        # polinomio P(x) = \sum_k f(x_k)L_k(x)
        lag = 0
        for k, p in enumerate(datos):
            lag += p[1]*L(k, x)
        return lag

    return P


# datos para f(x) = 1/x con x_0 = 2, x_1 = 2.75 y x_2 = 4
datosf = [(2, 1/2), (11/4, 4/11), (4, 1/4)]
Pf = LagrangePol(datosf)
print("Polinomio de Lagrange en x = 3:")
print("{0:.12f}".format(Pf(3)))

# datos g(x) = sin(3x), x_0 = 1, x_1 = 1.3, x_2 = 1.6, x_3 = 1.9 y x_4 = 2.2
datosg = [(1, 0.1411), (1.3, -0.6878), (1.6, -0.9962),
          (1.9, -0.5507), (2.2, 0.3115)]
Pg = LagrangePol(datosg)
print("Polinomio de Lagrange en x = 1.5:")
print("{0:.12f}".format(Pg(1.5)))
```

Salida

```
Polinomio de Lagrange en x = 3:
0.329545454545
Polinomio de Lagrange en x = 1.5:
-0.977381481481
```

B.2.2. Diferencias divididas de Newton

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
```

```
9  # it under the terms of the GNU General Public License as published by
10 # the Free Software Foundation, either version 3 of the License, or
11 # (at your option) any later version.
12 # This program is distributed in the hope that it will be useful,
13 # but WITHOUT ANY WARRANTY; without even the implied warranty of
14 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
15 # GNU General Public License for more details.
16 # You should have received a copy of the GNU General Public License
17 # along with this program.  If not, see <http://www.gnu.org/licenses/>
18 # ----------------------------------------------------------------------
19
20 # Implementación del interpolador de Newton y algunos casos de salida.
21
22 from math import *
23 from pprint import pprint
24
25
26 def NewtonPol(dat):
27     """
28     Implementación del interpolador de Newton
29     Entradas:
30     dat -- lista de puntos (x, y) en el plano
31
32     Salidas:
33     F -- tabla de diferencias divididas
34     P -- función de interpolación
35     """
36     n = len(dat)
37     F = [[0 for x in dat] for x in dat]  # crear tabla nula
38
39     for i, p in enumerate(dat):  # condiciones iniciales
40         F[i][0] = p[1]
41
42     for i in range(1, n):  # tabla de diferencias divididas
43         for j in range(1, i+1):
44             F[i][j] = (F[i][j-1]-F[i-1][j-1])/(dat[i][0]-dat[i-j][0])
45
46     def L(k, x):
47         """Implementación funciones L_k(x)"""
48         # polinomio L_k(x) = \prod_{i<=k} (x - x_i)
49         out = 1
50         for i, p in enumerate(dat):
51             if i <= k:
52                 out *= (x - p[0])
```

```
53          return out
54
55      def P(x):
56          """Implementación polinomio P(x)"""
57          # P(x) = f[x_0] + sum_{k=1}^{n} f[x_0, x_1, ..., x_k] L_{k-1}(x)
58          newt = 0
59          for i in range(1, n):
60              newt += F[i][i]*L(i-1, x)
61          return newt + F[0][0]
62
63      return F, P
64
65
66  datost = [(-1, 3), (0, -4), (1, 5), (2, -6)]
67  T, P = NewtonPol(datost)
68  print("Tabla de diferencias divididas:")
69  pprint(T)
70  print("Evaluar el polinomio en x = 0:")
71  print(P(0))
72
73  datosf = [(2, 1/2), (11/4, 4/11), (4, 1/4)]
74  T, P = NewtonPol(datosf)
75  print("Tabla de diferencias divididas:")
76  pprint(T)
77  print("Evaluar el polinomio en x = 3:")
78  print(P(3))
```

Salida

```
Tabla de diferencias divididas:
[[3, 0, 0, 0], [-4, -7.0, 0, 0], [5, 9.0, 8.0, 0], [-6, -11.0, -10.0, -6.0]]
Evaluar el polinomio en x = 0:
-4.0
Tabla de diferencias divididas:
[[0.5, 0, 0],
 [0.36363636363636365, -0.1818181818181818, 0],
 [0.25, -0.09090909090909091, 0.04545454545454544]]
Evaluar el polinomio en x = 3:
0.3295454545454546
```

B.2.3. Trazadores cúbicos

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
```

```
# ------------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ------------------------------------------------------------------------

# Trazadores cúbicos naturales y algunos casos de salida.

from math import *


def CubicSplines(datos):
    """
    Implementación trazadores cúbicos
    Entradas:
    datos -- lista de puntos (x, y) en el plano ordenados por x

    Salidas:
    a -- vector de coeficientes (constantes)
    b -- vector de coeficientes (lineales)
    c -- vector de coeficientes (cuadráticos)
    d -- vector de coeficientes (cúbicos)
    """
    n = len(datos)-1
    # Inicializar vectores auxiliares
    A = [x[1] for x in datos]
    X = [x[0] for x in datos]
    H = [0.0 for x in range(n)]
    B = [0.0 for x in range(n+1)]
    C = [0.0 for x in range(n+1)]
    D = [0.0 for x in range(n+1)]
    alpha = [0.0 for x in range(n)]
    mu = [0.0 for x in range(n+1)]
    lo = [1.0 for x in range(n+1)]
    z = [0.0 for x in range(n+1)]
```

```
49
50      # Crear vector H
51      for i in range(n):
52          H[i] = X[i+1]-X[i]
53
54      # Crear vector α
55      for i in range(1, n):
56          alpha[i] = (3/H[i])*(A[i+1]-A[i])-(3/H[i-1])*(A[i]-A[i-1])
57
58      # Solucionar sistema tridiagonal
59      for i in range(1, n):
60          lo[i] = 2*(X[i+1]-X[i-1])-H[i-1]*mu[i-1]
61          mu[i] = H[i]/lo[i]
62          z[i] = (alpha[i]-H[i-1]*z[i-1])/lo[i]
63
64      # Solucionar sistema tridiagonal
65      for j in range(n-1, -1, -1):
66          C[j] = z[j]-mu[j]*C[j+1]
67          B[j] = (A[j+1]-A[j])/(H[j])-H[j]*(C[j+1]+2*C[j])/3
68          D[j] = (C[j+1]-C[j])/(3*H[j])
69
70      # Retornar vectores A, B, C, D
71      return A[:-1], B[:-1], C[:-1], D[:-1]
72
73
74  # Datos de prueba (1,2), (2,3), (3,5)
75  datosPrueba = [(1, 2), (2, 3), (3, 5)]
76  a, b, c, d = CubicSplines(datosPrueba)
77  print("Vectores de coeficientes:")
78  print("A =", a)
79  print("B =", b)
80  print("C =", c)
81  print("D =", d)
82
83  # Datos de prueba (0,1), (1,e), (2,e^2) y (3,e^3)
84  datosPrueba = [(0, exp(0)), (1, exp(1)),
85                 (2, exp(2)), (3, exp(3))]
86  a, b, c, d = CubicSplines(datosPrueba)
87  print("Vectores de coeficientes:")
88  print("A =", a)
89  print("B =", b)
90  print("C =", c)
91  print("D =", d)
```

Salida

```
Vectores de coeficientes:
A = [2, 3]
B = [0.75, 1.5]
C = [0.0, 0.75]
D = [0.25, -0.25]
Vectores de coeficientes:
A = [1.0, 2.718281828459045, 7.38905609893065]
B = [1.465997614174723, 2.222850257027689, 8.809769654506473]
C = [0.0, 0.756852642852966, 5.830066754625817]
D = [0.252284214284322, 1.6910713705909506, -1.9433555848752724]
```

B.2.4. Recta de ajuste mínimos cuadrados

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ------------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ------------------------------------------------------------------------

# Implementación de la recta de mínimos cuadrados.

from math import *


def RectaMinSq(datos):
    """
    Implementación recta de mínimos cuadrados
    Entradas:
    datos -- lista de puntos (x, y) en el plano

```

```
31      Salida:
32      P -- recta de mínimos cuadrados
33      """
34      X = sum([p[0] for p in datos])
35      Y = sum([p[1] for p in datos])
36      XX = sum([(p[0])**2 for p in datos])
37      XY = sum([p[0]*p[1] for p in datos])
38      m = len(datos)
39
40      def P(x):
41          """Recta de mínimos cuadrados"""
42          a0 = (Y*XX - X*XY)/(m*XX - X**2)
43          a1 = (m*XY - X*Y)/(m*XX - X**2)
44          return a0 + a1*x
45
46      return P
47
48
49  def ErrorSq(f, datos):
50      """Calcular error cuadrático"""
51      E = sum([(p[1] - f(p[0]))**2 for p in datos])
52      return E
53
54
55  # datos de prueba
56  datos = [(-1, 2), (0, -1), (1, 1), (2, -2)]
57  f = RectaMinSq(datos)
58  print("Recta de ajuste. Evaluar en x = 1:")
59  print("{0:.10f}".format(f(1.0)))
60
61  # datos de prueba
62  datos = [(1.0, 1.3), (2.0, 3.5), (3.0, 4.2), (4.0, 5.0), (5.0, 7.0),
63           (6.0, 8.8), (7.0, 10.1), (8.0, 12.5), (9.0, 13.0)]
64  f = RectaMinSq(datos)
65  print("Recta de ajuste. Evaluar en x = 1:")
66  print("{0:.10f}".format(f(1.0)))
```

Salida

```
Recta de ajuste. Evaluar en x = 1:
-0.5000000000
Recta de ajuste. Evaluar en x = 1:
1.3066666667
```

B.3. Diferenciación e integración numérica

B.3.1. Extrapolación de Richardson

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ----------------------------------------------------------------------

# Implementación extrapolación de Richardson y algunos casos de salida.

from math import *


def pol(x):
    """Función de prueba"""
    return x**3 + 4*x**2 - 10  # retorna pol(x) = x^3 + 4x^2 - 10


def trig(x):
    """Función de prueba"""
    return x*cos(x-1) - sin(x)  # retorna trig(x) = x cos(x - 1) - sin(x)


def dercentrada(f, x, h):
    """Retorna diferencias centradas"""
    return (f(x + h) - f(x - h))/(2*h)


def richardson(f, x, h):
```

```
41      """
42      Implementación extrapolación de Richardson
43      Entradas:
44      f -- función
45      x -- punto
46      h -- paso
47
48      Salida:
49      d -- aproximación a la derivada
50      """
51      d = (4/3)*dercentrada(f, x, h/2)-(1/3)*dercentrada(f, x, h)
52      return d
53
54
55  # pol(x), x = 1.5, h = 0.1
56  print("Derivada función pol(x):")
57  print("{0:.12f}".format(richardson(pol, 1.5, 0.1)))
58
59  # trig(x), x = 4, h = 0.2
60  print("Derivada función trig(x):")
61  print("{0:.12f}".format(richardson(trig, 4, 0.2)))
```

Salida

```
Derivada función pol(x):
18.750000000000
Derivada función trig(x):
-0.900812732439
```

B.3.2. Regla del trapecio

```
1   #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   # ------------------------------------------------------------------------
5   # Compendio de programas.
6   # Matemáticas para Ingeniería. Métodos numéricos con Python.
7   # Copyright (C) 2020 Los autores del texto.
8   # This program is free software: you can redistribute it and/or modify
9   # it under the terms of the GNU General Public License as published by
10  # the Free Software Foundation, either version 3 of the License, or
11  # (at your option) any later version.
12  # This program is distributed in the hope that it will be useful,
13  # but WITHOUT ANY WARRANTY; without even the implied warranty of
14  # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
15 # GNU General Public License for more details.
16 # You should have received a copy of the GNU General Public License
17 # along with this program.  If not, see <http://www.gnu.org/licenses/>
18 # ----------------------------------------------------------------------
19
20 # Implementación de la regla del trapecio y algunos casos de salida.
21
22 from math import *
23
24
25 def pol(x):
26     """Función de prueba"""
27     return x**3 + 4*x**2 - 10  # retorna pol(x) = x^3 + 4x^2 - 10
28
29
30 def trig(x):
31     """Función de prueba"""
32     return x*cos(x-1) - sin(x)  # retorna trig(x) = x cos(x - 1) - sin(x)
33
34
35 def trapecio(f, a, b, n):
36     """
37     Implementación regla del trapecio
38     Entradas:
39     f -- función
40     a -- inicio intervalo
41     b -- fin intervalo
42     n -- número de pasos
43
44     Salida:
45     abc -- aproximación área bajo la curva
46     """
47     h = (b - a)/n
48     acum = 0
49     for j in range(1, n):
50         acum += 2*f(a + h*j)
51     abc = (h/2)*(f(a) + acum + f(b))
52     return abc
53
54
55 # pol(x), a = 1, b = 2, N = 10
56 print("\nÁrea bajo la curva pol(x):\n")
57 print("{0:.12f}".format(trapecio(pol, 1, 2, 10)))
58
59 # trig(x), a = 4, b = 6, N = 20
```

```
60 print("\nÁrea bajo la curva trig(x):\n")
61 print("{0:.12f}".format(trapecio(trig, 4, 6, 20)))
```

Salida

```
Área bajo la curva pol(x):

3.097500000000

Área bajo la curva trig(x):

-3.425574350843
```

B.3.3. Regla de Simpson

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # ------------------------------------------------------------------------
5  # Compendio de programas.
6  # Matemáticas para Ingeniería. Métodos numéricos con Python.
7  # Copyright (C) 2020 Los autores del texto.
8  # This program is free software: you can redistribute it and/or modify
9  # it under the terms of the GNU General Public License as published by
10 # the Free Software Foundation, either version 3 of the License, or
11 # (at your option) any later version.
12 # This program is distributed in the hope that it will be useful,
13 # but WITHOUT ANY WARRANTY; without even the implied warranty of
14 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
15 # GNU General Public License for more details.
16 # You should have received a copy of the GNU General Public License
17 # along with this program.  If not, see <http://www.gnu.org/licenses/>
18 # ------------------------------------------------------------------------
19
20 # Implementación de la regla de Simpson y algunos casos de salida.
21
22 from math import *
23
24
25 def pol(x):
26     """Función de prueba"""
27     return x**3 + 4*x**2 - 10  # retorna pol(x) = x^3 + 4x^2 - 10
28
29
```

```
30 def trig(x):
31     """Función de prueba"""
32     return x*cos(x-1) - sin(x)  # retorna trig(x) = x cos(x − 1) − sin(x)
33
34
35 def simpson(f, a, b, n):
36     """
37     Implementación regla de Simpson
38     Entradas:
39     f -- función
40     a -- inicio intervalo
41     b -- fin intervalo
42     n -- número de pasos (par)
43
44     Salida:
45     abc -- aproximación área bajo la curva
46     """
47     h = (b - a)/n
48     oddsum = 0
49     evensum = 0
50     for j in range(1, n):
51         x = a + h*j
52         if j % 2 == 0:
53             evensum += 2*f(x)
54         else:
55             oddsum += 4*f(x)
56     abc = (h/3)*(f(a) + evensum + oddsum + f(b))
57     return abc
58
59
60 # pol(x), a = 1, b = 2, N = 10
61 print("Área bajo la curva pol(x):")
62 print("{0:.12f}".format(simpson(pol, 1, 2, 10)))
63
64 # trig(x), a = 4, b = 6, N = 20
65 print("Área bajo la curva trig(x):")
66 print("{0:.12f}".format(simpson(trig, 4, 6, 20)))
```

Salida

```
Área bajo la curva pol(x):
3.083333333333
Área bajo la curva trig(x):
-3.430561834182
```

B.4. Sistemas de ecuaciones

B.4.1. *LU*

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ----------------------------------------------------------------------

# Implementación del método LU y algunos casos de salida.

from math import *
from pprint import pprint


def lu(A):
    """
    Implementación del método LU
    Entradas:
    A -- matriz cuadrada

    Salidas:
    L, U -- matrices de la descomposición
    None -- en caso de no ser posible la descomposición
    """
    n = len(A)
    # crear matrices nulas
    L = [[0 for x in range(n)] for x in range(n)]
    U = [[0 for x in range(n)] for x in range(n)]

```

```
    # Doolittle
    L[0][0] = 1
    U[0][0] = A[0][0]

    if abs(L[0][0]*U[0][0]) <= 1e-15:
        print("Imposible descomponer")
        return None

    for j in range(1, n):
        U[0][j] = A[0][j]/L[0][0]
        L[j][0] = A[j][0]/U[0][0]

    for i in range(1, n-1):
        L[i][i] = 1
        s = sum([L[i][k]*U[k][i] for k in range(i)])
        U[i][i] = A[i][i] - s

        if abs(L[i][i]*U[i][i]) <= 1e-15:
            print("Imposible descomponer")
            return None

        for j in range(i+1, n):
            s1 = sum([L[i][k]*U[k][j] for k in range(i)])
            s2 = sum([L[j][k]*U[k][i] for k in range(i)])
            U[i][j] = A[i][j] - s1
            L[j][i] = (A[j][i] - s2)/U[i][i]

    L[n-1][n-1] = 1.0
    s3 = sum([L[n-1][k]*U[k][n-1] for k in range(n)])
    U[n-1][n-1] = A[n-1][n-1] - s3

    if abs(L[n-1][n-1]*U[n-1][n-1]) <= 1e-15:
        print("Imposible descomponer")
        return None

    print("Matriz L:")
    pprint(L)
    print("Matriz U:")
    pprint(U)
    return L, U


A = [[4, 3], [6, 3]]
print("Matriz A:")
pprint(A)
```

```
lu(A)

A = [[0, 1], [1, 1]]
print("Matriz A:")
pprint(A)
lu(A)

A = [[3, 1, 6], [-6, 0, -16], [0, 8, -17]]
print("Matriz A:")
pprint(A)
lu(A)

A = [[1, 2, 3], [2, 4, 5], [1, 3, 4]]
print("Matriz A:")
pprint(A)
lu(A)
```

Salida

```
Matriz A:
[[4, 3], [6, 3]]
Matriz L:
[[1, 0], [1.5, 1.0]]
Matriz U:
[[4, 3.0], [0, -1.5]]
Matriz A:
[[0, 1], [1, 1]]
Imposible descomponer
Matriz A:
[[3, 1, 6], [-6, 0, -16], [0, 8, -17]]
Matriz L:
[[1, 0, 0], [-2.0, 1, 0], [0.0, 4.0, 1.0]]
Matriz U:
[[3, 1.0, 6.0], [0, 2.0, -4.0], [0, 0, -1.0]]
Matriz A:
[[1, 2, 3], [2, 4, 5], [1, 3, 4]]
Imposible descomponer
```

B.4.2. Jacobi

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
```

```
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ----------------------------------------------------------------------

# Implementación del método de Jacobi y algunos casos de salida.

from math import *
from pprint import pprint


def distinf(x, y):
    """Implementación distancia dada por la norma infinito"""
    return max([abs(x[i] - y[i]) for i in range(len(x))])


def Jacobi(A, b, x0, TOL, MAX):
    """
    Implementación del método de Jacobi
    Entradas:
    A -- matriz cuadrada
    b -- vector
    x0 -- aproximación inicial
    TOL -- tolerancia
    MAX -- número máximo de iteraciones

    Salida:
    x -- aproximación a solución del sistema Ax = b
    None -- en caso de agotar las iteraciones o presentar errores
    """
    n = len(A)
    x = [0.0 for x in range(n)]
    k = 1
    while k <= MAX:
        for i in range(n):
            if abs(A[i][i]) <= 1e-15:
```

```
                print("Imposible iterar")
                return None
            s = sum([A[i][j]*x0[j] for j in range(n) if j != i])
            x[i] = (b[i] - s)/A[i][i]
        pprint(x)
        if distinf(x, x0) < TOL:
            print(r"Solución encontrada")
            return x
        k += 1
        for i in range(n):
            x0[i] = x[i]
    print("Iteraciones agotadas")
    return None


A = [[2, 1], [5, 7]]
b = [11, 13]
x0 = [1, 1]
print("Matriz A:")
pprint(A)
print("Vector b:")
pprint(b)
print("Semilla x0:")
pprint(x0)
print("Iteración de Jacobi")
# TOL = 10^-5, MAX = 50
Jacobi(A, b, x0, 1e-5, 50)


A = [[10, -1, 2], [-1, 11, -1], [2, -1, 10]]
b = [6, 25, -11]
x0 = [0, 0, 0]
print("Matriz A:")
pprint(A)
print("Vector b:")
pprint(b)
print("Semilla x0:")
pprint(x0)
print("Iteración de Jacobi")
# TOL = 10^-10, MAX = 50
Jacobi(A, b, x0, 1e-10, 50)
```

Salida

```
Matriz A:
[[2, 1], [5, 7]]
```

```
Vector b:
[11, 13]
Semilla x0:
[1, 1]
Iteración de Jacobi
[5.0, 1.1428571428571428]
[4.928571428571429, -1.7142857142857142]
[6.357142857142857, -1.6632653061224494]
[6.331632653061225, -2.683673469387755]
[6.841836734693878, -2.6654518950437316]
[6.832725947521865, -3.0298833819241984]
[7.014941690962099, -3.0233756768013325]
[7.0116878384006665, -3.1535297792586428]
[7.076764889629321, -3.151205598857619]
[7.075602799428809, -3.197689206878086]
[7.098844603439043, -3.1968591424491493]
[7.098429571224575, -3.213460431027888]
[7.106730215513944, -3.2131639794461244]
[7.106581989723062, -3.2190930110813887]
[7.109546505540695, -3.2189871355164734]
[7.1094935677582365, -3.221104646814782]
[7.110552323407391, -3.2210668341130266]
[7.110533417056513, -3.221823088148137]
[7.110911544074068, -3.221809583611795]
[7.110904791805897, -3.22207967433862]
[7.11103983716931, -3.2220748512899258]
[7.111037425644962, -3.2221713122637925]
[7.1110856561318965, -3.2221695897464024]
[7.111084794873201, -3.222204040094212]
[7.111102020047106, -3.22220342490943]
[7.111101712454715, -3.2222157286050757]
[7.111107864302538, -3.2222155088962245]
Solución encontrada
Matriz A:
[[10, -1, 2], [-1, 11, -1], [2, -1, 10]]
Vector b:
[6, 25, -11]
Semilla x0:
[0, 0, 0]
Iteración de Jacobi
[0.6, 2.272727272727273, -1.1]
[1.0472727272727274, 2.227272727272727, -0.9927272727272728]
[1.0212727272727273, 2.277685950413223, -1.0867272727272728]
[1.0451140495867768, 2.266776859504132, -1.0764859504132231]
[1.0419748760330578, 2.2698752817430505, -1.0823451239669422]
```

```
[1.0434565529676934, 2.2690572501878283, -1.0814074470323065]
[1.043187214425244, 2.2692771914486713, -1.0817855855747558]
[1.0432848362598182, 2.269218329895499, -1.0817097237401818]
[1.0432637777375864, 2.269234101138149, -1.0817351342624137]
[1.0432704369662977, 2.269229876679561, -1.0817293454337025]
[1.0432688567546966, 2.269231008321145, -1.0817310997253036]
[1.0432693207771753, 2.26923070518449, -1.0817306705188248]
[1.043269204622214, 2.269230786387123, -1.0817307936369862]
[1.0432692373661094, 2.2692307646350205, -1.0817307622857304]
[1.0432692289206482, 2.2692307704618524, -1.08173077100972]
[1.043269231248129, 2.2692307689009934, -1.0817307687379443]
[1.0432692306376883, 2.269230769319108, -1.0817307693595264]
[1.043269230803816, 2.269230769207106, -1.081730769195627]
[1.0432692307598361, 2.269230769237108, -1.0817307692400526]
Solución encontrada
```

B.4.3. Gauss-Seidel

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ------------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ------------------------------------------------------------------------

# Implementación del método de Gauss-Seidel y algunos casos de salida.

from math import *
from pprint import pprint


def distinf(x, y):
```

```
27      """Implementación distancia dada por la norma infinito"""
28      return max([abs(x[i]-y[i]) for i in range(len(x))])
29
30
31  def GaussSeidel(A, b, x0, TOL, MAX):
32      """
33      Implementación del método de Gauss-Seidel
34      Entradas:
35      A -- matriz cuadrada
36      b -- vector
37      x0 -- aproximación inicial
38      TOL -- tolerancia
39      MAX -- número máximo de iteraciones
40
41      Salida:
42      x -- aproximación a solución del sistema Ax = b
43      None -- en caso de agotar las iteraciones o presentar errores
44      """
45      n = len(A)
46      x = [0.0 for x in range(n)]
47      k = 1
48      while k <= MAX:
49          for i in range(n):
50              if abs(A[i][i]) <= 1e-15:
51                  print("Imposible iterar")
52                  return None
53              s1 = sum([A[i][j]*x[j] for j in range(i)])
54              s2 = sum([A[i][j]*x0[j] for j in range(i+1, n)])
55              x[i] = (b[i] - s1 - s2)/A[i][i]
56          pprint(x)
57          if distinf(x, x0) < TOL:
58              print(r"Solución encontrada")
59              return x
60          k += 1
61          for i in range(n):
62              x0[i] = x[i]
63      print("Iteraciones agotadas")
64      return None
65
66
67  A = [[2, 1], [5, 7]]
68  b = [11, 13]
69  x0 = [1, 1]
70  print("Matriz A:")
71  pprint(A)
```

```
print("Vector b:")
pprint(b)
print("Semilla x0:")
pprint(x0)
print("Iteración de Gauss-Seidel")
# TOL = 10^-5, MAX = 50
GaussSeidel(A, b, x0, 1e-5, 50)


A = [[10, -1, 2], [-1, 11, -1], [2, -1, 10]]
b = [6, 25, -11]
x0 = [0, 0, 0]
print("Matriz A:")
pprint(A)
print("Vector b:")
pprint(b)
print("Semilla x0:")
pprint(x0)
print("Iteración de Gauss-Seidel")
# TOL = 10^-10, MAX = 50
GaussSeidel(A, b, x0, 1e-10, 50)
```

Salida

```
Matriz A:
[[2, 1], [5, 7]]
Vector b:
[11, 13]
Semilla x0:
[1, 1]
Iteración de Gauss-Seidel
[5.0, -1.7142857142857142]
[6.357142857142857, -2.683673469387755]
[6.841836734693878, -3.0298833819241984]
[7.014941690962099, -3.1535297792586428]
[7.076764889629321, -3.197689206878086]
[7.098844603439043, -3.213460431027888]
[7.106730215513944, -3.2190930110813887]
[7.109546505540695, -3.221104646814782]
[7.110552323407391, -3.221823088148137]
[7.110911544074068, -3.22207967433862]
[7.11103983716931, -3.2221713122637925]
[7.1110856561318965, -3.222204040094212]
[7.111102020047106, -3.2222157286050757]
[7.111107864302538, -3.2222199030732415]
Solución encontrada
```

```
Matriz A:
[[10, -1, 2], [-1, 11, -1], [2, -1, 10]]
Vector b:
[6, 25, -11]
Semilla x0:
[0, 0, 0]
Iteración de Gauss-Seidel
[0.6, 2.3272727272727276, -0.9872727272727273]
[1.0301818181818183, 2.276628099173554, -1.0783735537190082]
[1.043337520661157, 2.2695421788129226, -1.081713286250939]
[1.0432968751314802, 2.2692348717164132, -1.0817358878546546]
[1.0432706647425722, 2.269230434262538, -1.0817310895222607]
[1.043269261330706, 2.269230742891677, -1.0817307779769734]
[1.0432692298845623, 2.2692307683552353, -1.0817307691413889]
[1.0432692306638014, 2.2692307692293103, -1.0817307692098292]
[1.043269230764897, 2.2692307692322786, -1.0817307692297515]
[1.043269230769178, 2.269230769230857, -1.0817307692307498]
Solución encontrada
```

B.5. Ecuaciones diferenciales

B.5.1. Euler

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ----------------------------------------------------------------------

# Implementación del método de Euler y algunos casos de salida.
```

```
21
22 from math import *
23
24
25 def test1(t, y):
26     """Función de prueba"""
27     return y - t**2 + 1   # y' = y - t^2 + 1
28
29
30 def test2(t, y):
31     """Función de prueba"""
32     return 2 - exp(-4*t) - 2*y   # y' = 2 - e^{-4t} - 2y
33
34
35 def Euler(a, b, y0, f, N):
36     """
37     Implementación método de Euler
38     Entradas:
39     a -- inicio intervalo
40     b -- fin intervalo
41     y0 -- aproximación inicial
42     f -- función
43     N -- pasos
44
45     Salida:
46     w -- aproximación final
47     """
48     h = (b - a)/N
49     t = a
50     w = y0
51     print("t0  = {0:.2f}, w0  = {1:.12f}".format(t, w))
52     for i in range(1, N+1):
53         w = w + h*f(t, w)
54         t = a + i*h
55         print("t{0:<2} = {1:.2f}, w{0:<2} = {2:.12f}".format(i, t, w))
56     return w
57
58
59 # dy/dt = y - t^2 + 1, a = 0, b = 2, y0 = 0.5, N = 10
60 print("Método de Euler:")
61 Euler(0, 2, 0.5, test1, 10)
62
63 # dy/dt = 2 - e^{-4t} - 2y, a = 0, b = 1, y0 = 1, N = 20
64 print("Método de Euler:")
65 Euler(0, 1, 1, test2, 20)
```

Salida

```
Método de Euler:
t0  = 0.00, w0  = 0.500000000000
t1  = 0.20, w1  = 0.800000000000
t2  = 0.40, w2  = 1.152000000000
t3  = 0.60, w3  = 1.550400000000
t4  = 0.80, w4  = 1.988480000000
t5  = 1.00, w5  = 2.458176000000
t6  = 1.20, w6  = 2.949811200000
t7  = 1.40, w7  = 3.451773440000
t8  = 1.60, w8  = 3.950128128000
t9  = 1.80, w9  = 4.428153753600
t10 = 2.00, w10 = 4.865784504320
Método de Euler:
t0  = 0.00, w0  = 1.000000000000
t1  = 0.05, w1  = 0.950000000000
t2  = 0.10, w2  = 0.914063462346
t3  = 0.15, w3  = 0.889141113810
t4  = 0.20, w4  = 0.872786420624
t5  = 0.25, w5  = 0.863041330356
t6  = 0.30, w6  = 0.858343225262
t7  = 0.35, w7  = 0.857449192140
t8  = 0.40, w8  = 0.859374424729
t9  = 0.45, w9  = 0.863342156356
t10 = 0.50, w10 = 0.868742996309
t11 = 0.55, w11 = 0.875101932517
t12 = 0.60, w12 = 0.882051581347
t13 = 0.65, w13 = 0.889310525548
t14 = 0.70, w14 = 0.896665794082
t15 = 0.75, w15 = 0.903958711543
t16 = 0.80, w16 = 0.911073486970
t17 = 0.85, w17 = 0.917928028074
t18 = 0.90, w18 = 0.924466561769
t19 = 0.95, w19 = 0.930653719470
t20 = 1.00, w20 = 0.936469808930
```

B.5.2. Verlet

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ----------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
```

```
7  # Copyright (C) 2020 Los autores del texto.
8  # This program is free software: you can redistribute it and/or modify
9  # it under the terms of the GNU General Public License as published by
10 # the Free Software Foundation, either version 3 of the License, or
11 # (at your option) any later version.
12 # This program is distributed in the hope that it will be useful,
13 # but WITHOUT ANY WARRANTY; without even the implied warranty of
14 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
15 # GNU General Public License for more details.
16 # You should have received a copy of the GNU General Public License
17 # along with this program.  If not, see <http://www.gnu.org/licenses/>
18 # ----------------------------------------------------------------------
19
20 # Implementación del método de Verlet y algunos casos de salida.
21
22 from math import *
23
24
25 def test1(x):  # y'' = x
26     """Función de prueba"""
27     return x
28
29
30 def test2(x):  # y'' = -x
31     """Función de prueba"""
32     return -x
33
34
35 def Verlet(a, b, x0, v0, f, N):
36     """
37     Implementación método de Verlet
38     Entradas:
39     a -- inicio intervalo
40     b -- fin intervalo
41     x0 -- aproximación inicial
42     v0 -- aproximación inicial
43     f -- función
44     N -- pasos
45
46     Salida:
47     p1 -- aproximación final
48     """
49     h = (b - a)/N
50     p0 = x0
51     p1 = p0 + v0*h + 0.5*f(p0)*h**2
```

```
52      print("a0  = {0:0.2f}, p0  = {1:0.12f}".format(a, p0))
53      for i in range(1, N+1):
54          p = 2*p1 - p0 + f(p1)*h**2
55          s = a + i*h
56          print("a{0:<2} = {1:0.2f}, p{0:<2} = {2:.12f}".format(i, s, p1))
57          p0 = p1
58          p1 = p
59      return p1
60
61
62  # d^2x/dt^2 = x, a = 0, b = 1, x_0 = 1, v_0 = 1, N = 20
63  print("Método de Verlet:")
64  Verlet(0, 1, 1, 1, test1, 20)
65
66  # d^2x/dt^2 = -x, a = 0, b = 1, x_0 = 1, v_0 = 0, N = 20
67  print("Método de Verlet:")
68  Verlet(0, 1, 1, 0, test2, 20)
```

Salida

```
Método de Verlet:
a0  = 0.00, p0  = 1.000000000000
a1  = 0.05, p1  = 1.051250000000
a2  = 0.10, p2  = 1.105128125000
a3  = 0.15, p3  = 1.161769070313
a4  = 0.20, p4  = 1.221314438301
a5  = 0.25, p5  = 1.283913092385
a6  = 0.30, p6  = 1.349721529200
a7  = 0.35, p7  = 1.418904269838
a8  = 0.40, p8  = 1.491634271150
a9  = 0.45, p9  = 1.568093358141
a10 = 0.50, p10 = 1.648472678527
a11 = 0.55, p11 = 1.732973180609
a12 = 0.60, p12 = 1.821806115642
a13 = 0.65, p13 = 1.915193565965
a14 = 0.70, p14 = 2.013369000203
a15 = 0.75, p15 = 2.116577856941
a16 = 0.80, p16 = 2.225078158322
a17 = 0.85, p17 = 2.339141155098
a18 = 0.90, p18 = 2.459052004762
a19 = 0.95, p19 = 2.585110484438
a20 = 1.00, p20 = 2.717631740325
Método de Verlet:
a0  = 0.00, p0  = 1.000000000000
a1  = 0.05, p1  = 0.998750000000
a2  = 0.10, p2  = 0.995003125000
```

```
a3  = 0.15, p3  = 0.988768742188
a4  = 0.20, p4  = 0.980062437520
a5  = 0.25, p5  = 0.968905976758
a6  = 0.30, p6  = 0.955327251054
a7  = 0.35, p7  = 0.939360207223
a8  = 0.40, p8  = 0.921044762873
a9  = 0.45, p9  = 0.900426706617
a10 = 0.50, p10 = 0.877557583594
a11 = 0.55, p11 = 0.852494566612
a12 = 0.60, p12 = 0.825300313213
a13 = 0.65, p13 = 0.796042809032
a14 = 0.70, p14 = 0.764795197827
a15 = 0.75, p15 = 0.731635598629
a16 = 0.80, p16 = 0.696646910433
a17 = 0.85, p17 = 0.659916604962
a18 = 0.90, p18 = 0.621536507978
a19 = 0.95, p19 = 0.581602569724
a20 = 1.00, p20 = 0.540214625046
```

B.5.3. RK4

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# ---------------------------------------------------------------------
# Compendio de programas.
# Matemáticas para Ingeniería. Métodos numéricos con Python.
# Copyright (C) 2020 Los autores del texto.
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>
# ---------------------------------------------------------------------

# Implementación del método RK4 y algunos casos de salida.

from math import *

```

```
24
25 def test1(t, y):  # y' = y - t^2 + 1
26     """Función de prueba"""
27     return y - t**2 + 1
28
29
30 def test2(t, y):  # y' = 2 - e^{-4t} - 2y
31     """Función de prueba"""
32     return 2 - exp(-4*t) - 2*y
33
34
35 def RK4(a, b, y0, f, N):
36     """
37     Implementación método RK4
38     Entradas:
39     a -- inicio intervalo
40     b -- fin intervalo
41     y0 -- aproximación inicial
42     f -- función
43     N -- pasos
44
45     Salida:
46     w -- aproximación final
47     """
48     h = (b - a)/N
49     t = a
50     w = y0
51     print("t0  = {0:.2f}, w0  = {1:.12f}".format(t, w))
52
53     for i in range(1, N+1):
54         k1 = h*f(t, w)
55         k2 = h*f(t + h/2, w + k1/2)
56         k3 = h*f(t + h/2, w + k2/2)
57         k4 = h*f(t + h, w + k3)
58         w = w + (k1 + 2*k2 + 2*k3 + k4)/6
59         t = a + i*h
60         print("t{0:<2} = {1:.2f}, w{0:<2} = {2:.12f}".format(i, t, w))
61     return w
62
63
64 # dy/dt = y - t^2 + 1, a = 0, b = 2, y_0 = 0.5, N = 10
65 print("Método RK4:")
66 RK4(0, 2, 0.5, test1, 10)
67
68 # dy/dt = 2 - e^{-4t} - 2y, a = 0, b = 1, y_0 = 1, N = 20
```

```
69 print("Método RK4:")
70 RK4(0, 1, 1, test2, 20)
```

Salida

```
Método RK4:
t0  = 0.00, w0  = 0.500000000000
t1  = 0.20, w1  = 0.829293333333
t2  = 0.40, w2  = 1.214076210667
t3  = 0.60, w3  = 1.648922017042
t4  = 0.80, w4  = 2.127202684948
t5  = 1.00, w5  = 2.640822692729
t6  = 1.20, w6  = 3.179894170232
t7  = 1.40, w7  = 3.732340072855
t8  = 1.60, w8  = 4.283409498318
t9  = 1.80, w9  = 4.815085694579
t10 = 2.00, w10 = 5.305363000693
Método RK4:
t0  = 0.00, w0  = 1.000000000000
t1  = 0.05, w1  = 0.956946773927
t2  = 0.10, w2  = 0.925794826349
t3  = 0.15, w3  = 0.903996935703
t4  = 0.20, w4  = 0.889504715870
t5  = 0.25, w5  = 0.880674661873
t6  = 0.30, w6  = 0.876191562614
t7  = 0.35, w7  = 0.875006100539
t8  = 0.40, w8  = 0.876284037659
t9  = 0.45, w9  = 0.879364861493
t10 = 0.50, w10 = 0.883728152457
t11 = 0.55, w11 = 0.888966251604
t12 = 0.60, w12 = 0.894762067259
t13 = 0.65, w13 = 0.900871071482
t14 = 0.70, w14 = 0.907106710988
t15 = 0.75, w15 = 0.913328599230
t16 = 0.80, w16 = 0.919432972496
t17 = 0.85, w17 = 0.925344987868
t18 = 0.90, w18 = 0.931012518523
t19 = 0.95, w19 = 0.936401165330
t20 = 1.00, w20 = 0.941490255536
```

Bibliografía

[1] R. Burden. *Análisis Numérico*. Grupo Editorial Iberoamericano. 2002.

[2] W.E Boyce, R.C. DiPrima. *Elementary Differential Equations and Boundary Value Problems*. Wiley. 2008.

[3] A.B. Downey. *Think Python*. Green Tea Press. 2012

[4] H. Th. Jongen, K. Meer, E. Triesch. *Optimization Theory*, Springer. 2004

[5] D. Kincaid, W. Cheney. *Análisis Numérico*. Addison Wesley. 1994

[6] E. Kreyszig. *Matemáticas avanzadas para Ingeniería*. Limusa Wiley. 2003.

[7] L. Leithold. *Álgebra y Trigonometría con Geometría Analítica*. Editorial Harla. 1987

[8] I. Mantilla Prada. *Análisis Numérico*. Universidad Nacional de Colombia. 2004

[9] H.M. Mora Escobar. *Introducción a* `C` *y a Métodos Numéricos*. Universidad Nacional de Colombia. 2004

[10] S. Nakamura. *Métodos Numéricos aplicados con* Software. Prentice Hall. 1998

[11] A. Nieves, F.C. Domínguez. *Métodos Numéricos aplicados a la Ingeniería*. Grupo Editorial Patria. 2014

[12] M. Pilgrim. *Dive Into Python*. Apress. 2004

El libro contiene métodos computacionales para resolver problemas esenciales en el campo de ingeniería o matemática aplicada. En cada método se pretende dar al lector una visión de su esencia, necesidad, ventajas y desventajas. En algunos casos, alejándose de presentaciones rigurosas, pero sin dejar de ser correctas. El objetivo final es proporcionar los elementos necesarios para la aplicación adecuada de los algoritmos.

ISBN: 978-958-5142-97-8