Numerical Programming Assignment 4

By Paige Meyer

## Euler’s Method

|  |
| --- |
| # Euler Method  def yPrime(t, y):  return -t \* y + 4\*t/y  #return y - t\*\*2 + 1  def w(prev\_w, h, i, a):  return prev\_w + h\*(yPrime(a+i\*h, prev\_w))  a = 0.0  b = 1.0  h = 0.1  wi = 1.0  for i in range(int((b-a)/h)+1):  wi = w(wi, h, float(i),a)  print("t{}: {}".format(b,wi)) |

## Taylor Method of Order 2

Note: derived using Wolfram alpha

**Input**: derive -t\*y(t) + 4\*t/y(t) **Output**: -(t (y(t)^2 + 4) y'(t) + y(t) (y(t)^2 - 4))/y(t)^2

|  |
| --- |
| '''Order Two Taylor Method'''  from math import exp  def frange(x, y, jump):  tmplist = []  while x <= y:  tmplist.append(x)  x += jump  return tmplist  def f(t,y):  return -t \* y + 4\*t/y  def fprime(t, y):  return -(t \*(y\*\*2 + 4) \*f(t,y) + y \*(y\*\*2 - 4))/y\*\*2  def t2(t,y, h):  return f(t,y) + (h/2) \* fprime(t,y)  def w(ti, wi, h):  return wi + (h) \* t2(ti, wi, h)  a = 0.0  b = 1.0  h = 0.1  w0 = 1.0  for i in frange(a+h, b, h):  w0 = w(i-h, w0, h)  print("t{:.3f}: {}".format(b,w0)) |

## Taylor’s Method of Order 4

Note: derived using Wolframalpha

**Input**: derive -(t (y(t)^2 + 4) y'(t) + y(t) (y(t)^2 - 4))/y(t)^2

**Output**: -(t y(t) (y(t)^2 + 4) y''(t) + 2 y'(t) (-4 t y'(t) + y(t)^3 + 4 y(t)))/y(t)^3

**Input**: derive -(t y(t) (y(t)^2 + 4) y''(t) + 2 y'(t) (-4 t y'(t) + y(t)^3 + 4 y(t)))/y(t)^3

**Output**: (-t (y(t)^2 + 4) y^(3)(t) y(t)^2 + 24 y'(t)^2 (y(t) - t y'(t)) - 3 y(t) (-8 t y'(t) + y(t)^3 + 4 y(t)) y''(t))/y(t)^4

|  |
| --- |
| '''Order Four Taylor Method'''  from math import exp  def frange(x, y, jump):  tmplist = []  while x <= y:  tmplist.append(x)  x += jump  return tmplist  def f(t,y):  return -t \* y + 4\*t/y  def fprime(t, y):  return -(t \*(y\*\*2 + 4) \*f(t,y) + y \*(y\*\*2 - 4))/y\*\*2  def fprime2(t,y):  # -(t y(t) (y(t)^2 + 4) y''(t) + 2 y'(t) (-4 t y'(t) + y(t)^3 + 4 y(t)))/y(t)^3  return -(t \* y \*(y\*\*2 + 4)\* fprime(t,y) + 2 \*f(t,y) \*(-4\* t \*f(t,y) + y\*\*3 + 4\* y))/y\*\*3  def fprime3(t,y):  # (-t (y(t)^2 + 4) y^(3)(t) y(t)^2 + 24 y'(t)^2 (y(t) - t y'(t)) - 3 y(t) (-8 t y'(t) + y(t)^3 + 4 y(t)) y''(t))/y(t)^4  return (-t \*(y\*\*2 + 4) \* fprime2(t,y) \* y\*\*2 + 24 \*f(t,y)\*\*2 \*(y - t \*f(t,y)) - 3 \*y \*(-8 \*t\* f(t,y) + y\*\*3 + 4\* y)\* fprime(t,y))/y\*\*4  def t2(t,y, h):  return f(t,y) + (h/2) \* fprime(t,y)  def t4(t,y, h):  return t2(t,y,h) + (h\*\*2/6)\*fprime2(t,y) + (h\*\*3/24)\*fprime3(t,y)  def w(ti, wi, h):  return wi + h \* t4(ti, wi, h)  a = 0.0  b = 1.0  h = 0.1  w0 = 1.0  for i in frange(a+h, b, h):  w0 = w(i-h, w0, h)  print("t{:.3f}: {}".format(b,w0)) |

## The Midpoint Method

|  |
| --- |
| '''Midpoint Method'''  def frange(x, y, jump):  tmplist = []  while x <= y:  tmplist.append(x)  x += jump  return tmplist  def f(t, y):  return -t \* y + 4\*t/y  #return y - t\*\*2 + 1  def w(ti,wi, h):  return wi + (h) \* (f(ti+(h/2),wi+(h/2)\*f(ti,wi)))  a = 0.0  b = 1.0  h = 0.1  w0 = 1.0  for i in frange(a+h, b, h):  w0 = w(i-h, w0, h)  print("t{:.3f}: {}".format(b,w0)) |

## Modified Euler’s Method

|  |
| --- |
| '''Modified Euler Method'''  def frange(x, y, jump):  tmplist = []  while x <= y:  tmplist.append(x)  x += jump  return tmplist  def f(t, y):  return -t \* y + 4\*t/y  #return -y +t\*(y\*\*0.5)  def w(ti,wi, h):  return wi + (h/2) \* (f(ti,wi)+f(ti+h, wi + h\*f(ti,wi)))  a = 0.0  b = 1.0  h = 0.1  w0 = 1.0  for i in frange(a+h, b, h):  w0 = w(i-h, w0, h)  print("t{:.3f}: {}".format(b,w0)) |

## Heun’s Method

|  |
| --- |
| '''Heun's Method'''  def frange(x, y, jump):  tmplist = []  while x <= y:  tmplist.append(x)  x += jump  return tmplist  def f(t, y):  return -t \* y + 4\*t/y  #return -y +t\*(y\*\*0.5)  def w(ti,wi, h):  return wi + (h/4) \* (f(ti,wi)+ 3\* f(ti+(2/3)\*h,wi + (2/3)\*h\*f(ti+(h/3),wi+h/3\*f(ti,wi))))  a = 0.0  b = 1.0  h = 0.1  w0 = 1.0  for i in frange(a+h, b, h):  w0 = w(i-h, w0, h)  print("t{:.3f}: {}".format(b,w0)) |

## Runge-Kutta Order 4

|  |
| --- |
| '''Runge-Kutta Order Four Method'''  def frange(x, y, jump):  tmplist = []  while x <= y:  tmplist.append(x)  x += jump  return tmplist  def f(t, y):  return -t \* y + 4\*t/y  def w(ti,wi, h):  k1 = h\*f(ti,wi)  k2 = h\*(f(ti+(h/2),wi+k1/2))  k3 = h\*(f(ti+(h/2),wi+k2/2))  k4 = h\*(f(ti+h, wi+k3))  return wi + (1/6) \* (k1+ 2\* k2 + 2\*k3 + k4)  a = 0.0  b = 1.0  h = 0.1  w0 = 1.0  for i in frange(a+h, b, h):  w0 = w(i-h, w0, h)  print("t{:.3f}: {}".format(b,w0)) |

# A. h = 0.1

## Euler’s Method

|  |
| --- |
| t1.0: 1.7002148697864552 |

## Taylor Method of Order 2

|  |
| --- |
| t1.000: 1.7043394627441557 |

## Taylor’s Method of Order 4

|  |
| --- |
| t1.000: 1.7016942058924507 |

## The Midpoint Method

|  |
| --- |
| t1.000: 1.702247783424931 |

## Modified Euler’s Method

|  |
| --- |
| t1.000: 1.7002102953788956 |

## Heun’s Method

|  |
| --- |
| t1.000: 1.7018558380809625 |

## Runge-Kutta Order 4

|  |
| --- |
| t1.000: 1.7018677085421234 |

# 2. B. h=0.01

## Euler’s Method

|  |
| --- |
| t1.0: 1.7016942058924507 |

## Taylor Method of Order 2

|  |
| --- |
| t1.000: 1.7018922896192503 |

## Taylor’s Method of Order 4

|  |
| --- |
| t1.000: 1.7018700519754908 |

## The Midpoint Method

|  |
| --- |
| t1.000: 1.7018728288694795 |

## Modified Euler’s Method

|  |
| --- |
| t1.000: 1.7018534308891726 |

## Heun’s Method

|  |
| --- |
| t1.000: 1.7018700445748118 |

## Runge-Kutta Order 4

|  |
| --- |
| t1.000: 1.701870052530176 |

# 3. a.

Checking initial condition y(0) = 1.

# 3. b. Absolute Error using h=0.1

Actual value of y(1) = 1.7018700527612773

### a. Euler’s Method

|y(1) - 1.7002148697864552 | = 0.0016551829748221447

### b. Taylor’s Method of Order Two

|y(1)-1.7043394627441557| = 0.0024694099828783678

### c. Taylor’s Method of Order Four

|y(1)-1.7018615673417952| = 8.48541948217374e-06

### d. Midpoint Method

|y(1) - 1.702247783424931| = 0.0003777306636536526

### e. Modified Euler’s Method

|y(1)-1.7002102953788956| = 0.0016597573823817768

### f. Heun’s Method

|y(1)-1. 7018558380809625| = 1.4214680314816874e-05

### g. Runge-Kutta Order 4

|y(1)-1.7018677085421234| = 2.344219153904703e-06

# 3. c. Absolute Error Using h=0.01

Actual value of y(1) = 1.7018700527612773

### a. Euler’s Method

|y(1) - 1.7016942058924507 | = 0.0001758468688266568

### b. Taylor’s Method of Order Two

| y(1) - 1.7018700519754908 | = 2.223685797297925e-05

### c. Taylor’s Method of Order Four

| y(1) - 1.6953404638524072 | = 7.857865469418357e-10

### d. Midpoint Method

| y(1) - 1.7018728288694795| = 2.7761082022070838e-06

### e. Modified Euler’s Method

| y(1) - 1.7018534308891726 | = 1.6621872104716218e-05

### f. Heun’s Method

| y(1) - 1.7018700445748118 | = 8.186465505488627e-09

### g. Runge-Kutta Order 4

| y(1) - 1.701870052530176| = 2.3110136027071349e-10

# 3. d.

Note: each of these have h=0.1 / h=0.01 so a number greater than 1 indicates convergence and a number smaller than 1 indicates a less precise h=0.01 number.

### a. Euler’s Method

0.0016551829748221447 / 0.0001758468688266568 = 9.412638313473535

Since we decreased h by a factor of ten and error decreased by factor of 10 (approximately), so seems to be linear.

### b. Taylor’s Method of Order Two

0.0024694099828783678 / 2.223685797297925e-05 = 111.05031051954508

Since we decreased h by a factor of ten and error decreased by factor of 100 (approximately), so seems to be quadratic,

### c. Taylor’s Method of Order Four

8.48541948217374e-06 / 7.857865469418357e-10 = 10798.63165791999

Since we decreased h by a factor of ten and error decreased by factor of 10000 (approximately), so seems to be decreased by 10^4 approximately, , .

### d. Midpoint Method

0.0003777306636536526 / 2.7761082022070838e-06 = 136.06482029531347

Since we decreased h by a factor of ten and error decreased by factor of 100 (approximately), so seems to be quadratic.

### e. Modified Euler’s Method

0.0016597573823817768 / 1.6621872104716218e-05 = 99.85381742354066

Since we decreased h by a factor of ten and error decreased by factor of 100 (approximately), so seems to be quadratic, .

### f. Heun’s Method

1.4214680314816874e-05 / 8.186465505488627e-09 = 1736.363550947185

Since we decreased h by a factor of ten and error decreased by factor of 1000 (approximately), so seems to be quadratic,.

### g. Runge-Kutta Order 4

2.344219153904703e-06 / 2.3110136027071349e-10 = 10143.683927946902

Since we decreased h by a factor of ten and error decreased by factor of 10000 (approximately), so seems to be decreased by 10^4 approximately, .