1. Upload last week’s notebook.

1. import numpy as np
3. ## y'' = -0.1\*y' - x, y(0) = 0, y'(0) = 1
5. ## Define the first orde equations. For ordef 2 we have two variables.
6. ## yvec = [y0 , y1 ], where y0 = y, y1 = y'
7. ## yvec'= [y0', y1'] = F(x, yvec)
9. ## Let's solve in a pedestrian manner starting from x = 0
10. ## Let's set h = 0.2
12. ## Solve for y0(0)
13. x = []; y0 = []; y1 = []
14. x.append(0); y0.append(0); y1.append(1)
16. y1p = lambda x,y0,y1:-0.1\*y1 - x
18. h = 0.2
20. ## Let's evaluate at 0 + h
21. x.append(h)
22. y0.append( y0[0] + y1[0]\*h )
23. y1.append( y1[0] + y1p(x[0],y0[0],y1[0])\*h )
25. ## Let's caculate at 0 + 2h
26. x.append(2\*h)
27. y0.append( y0[1] + y1[1]\*h )
28. y1.append( y1[1] + y1p(x[1],y0[1],y1[1])\*h )
30. print(' x y0 y1')
31. print(np.transpose([x,y0,y1]))
32. print('\n-----------------------\n')
33. ##
34. from euler import \*
35. import matplotlib.pyplot as plt
37. def F(x,y):
38. F = np.zeros(2)
39. F[0] = y[1]
40. F[1] = -0.1\*y[1] - x
41. return F
43. x = 0.0
44. xStop = 2.0
45. y = np.array([0.0, 1.0])
46. h = 0.2
48. # Call the integration routine
49. X,Y = integrate(F,x,y,xStop,h)
50. yExact = 100.0\*X - 5.0\*X\*\*2 + 990.0\*(np.exp(-0.1\*X) - 1.0)
52. from printSoln import \*
53. freq = 2
54. printSoln(X,Y,freq)
56. # Plotting tool
57. plt.plot(X,Y[:,0],'o',X,yExact,'-')
58. plt.grid(True)
59. plt.xlabel('x'); plt.ylabel('y')
60. plt.legend(('Numerical','Exact'),loc=0)
61. plt.show()

x y0 y1

[[0. 0. 1. ]

[0.2 0.2 0.98 ]

[0.4 0.396 0.9204]]

-----------------------

x y[ 0 ] y[ 1 ]

0.0000e+00 0.0000e+00 1.0000e+00

4.0000e-01 3.9600e-01 9.2040e-01

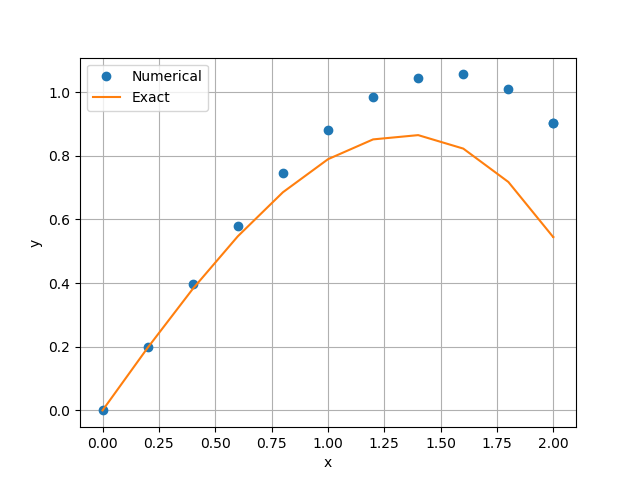
8.0000e-01 7.4448e-01 6.8555e-01

1.2000e+00 9.8396e-01 3.0160e-01

1.6000e+00 1.0554e+00 -2.2554e-01

2.0000e+00 9.0208e-01 -8.9021e-01

2.0000e+00 9.0208e-01 -8.9021e-01



1. def F(x,y):
2. F = np.zeros(2)
3. F[0] = y[1]
4. F[1] = -0.1\*y[1] - x
5. return F

F is derivative vector. F = [ y’, y’’ ]

1. from euler import \*
2. # Call the integration routine
3. X,Y = integrate(F,x,y,xStop,h)

euler.integrate function is reiteration of the following codes.

1. ## Let's evaluate at 0 + h
2. x.append(h)
3. y0.append( y0[0] + y1[0]\*h )
4. y1.append( y1[0] + y1p(x[0],y0[0],y1[0])\*h )
6. ## Let's caculate at 0 + 2h
7. x.append(2\*h)
8. y0.append( y0[1] + y1[1]\*h )
9. y1.append( y1[1] + y1p(x[1],y0[1],y1[1])\*h )

05.28.py

1. #!/usr/bin/python
2. ## y'' +3yy' = 0
4. import numpy as np
5. import matplotlib.pyplot as plt
7. from run\_kut4 import \*
8. from ridder import \*
9. from printSoln import \*
11. def initCond(u): # Init. values of [y,y']; use 'u' if unknown
12. return np.array([0.0, u])
14. def r(u): # Boundary condition residual
15. X,Y = integrate(F,xStart,initCond(u),xStop,h)
16. y = Y[len(Y) - 1]
17. r = y[0] - 1.0
18. return r
20. def F(x,y): # First-order differential equations
21. F = np.zeros(2)
22. F[0] = y[1]
23. F[1] = -3.0\*y[0]\*y[1]
24. return F
26. xStart = 0.0
27. xStop = 2.0
28. u1 = 1.0
29. u2 = 2.0
30. # Start of integration
31. # End of integration
32. # 1st trial value of unknown init. cond.
33. # 2nd trial value of unknown init. cond.
34. # Step size
35. # Printout frequency
36. h = 0.1
37. freq = 2
38. u = ridder(r,u1,u2) # Compute the correct initial condition
39. X,Y = integrate(F,xStart,initCond(u),xStop,h)
40. printSoln(X,Y,freq)
42. plt.plot(X,Y[:,0],'o-',X,Y[:,1],'-')
43. plt.xlabel('x')
44. plt.legend(('y','dy/dx'),loc = 1)
45. plt.grid(True)
46. plt.show()

x y[ 0 ] y[ 1 ]

0.0000e+00 0.0000e+00 1.5145e+00

2.0000e-01 2.9404e-01 1.3848e+00

4.0000e-01 5.4170e-01 1.0743e+00

6.0000e-01 7.2187e-01 7.3287e-01

8.0000e-01 8.3944e-01 4.5752e-01

1.0000e+00 9.1082e-01 2.7013e-01

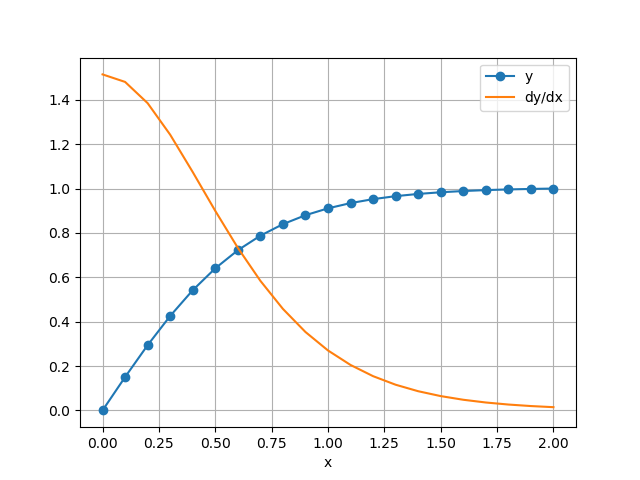
1.2000e+00 9.5227e-01 1.5429e-01

1.4000e+00 9.7572e-01 8.6471e-02

1.6000e+00 9.8880e-01 4.7948e-02

1.8000e+00 9.9602e-01 2.6430e-02

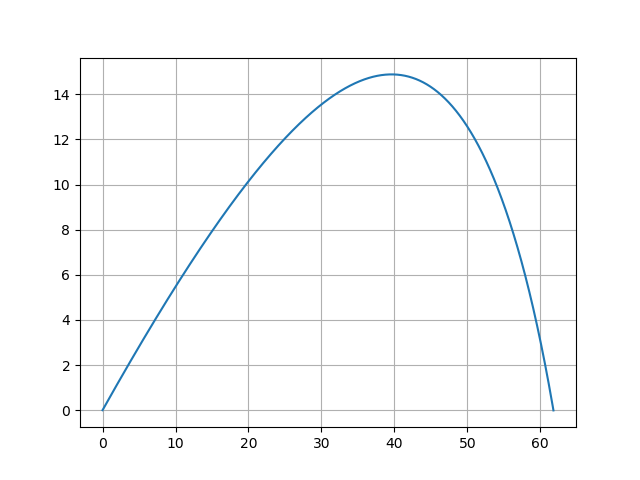
2.0000e+00 1.0000e+00 1.4522e-02



problem7.1.13.py

1. import numpy as np
2. from run\_kut4 import \*
4. theta = np.pi/6
5. m = 0.25 # kg
6. v0 = 50 # m/s
7. C = 0.03 # kg/(m s)^0.5
8. g = 9.80665 # m/s^2
10. ## Function for using integrate function of run\_kut4
11. ## x = [x, x', y, y']
12. def F(t,x):
13. F = np.zeros(4)
14. v = (x[1]\*\*2 + x[3]\*\*2)\*\*0.5
15. F[0] = x[1]
16. F[1] = -C/m\*x[1]\*(v\*\*0.5)
17. F[2] = x[3]
18. F[3] = -C/m\*x[3]\*(v\*\*0.5) - g
19. return F
21. h = 0.001
22. x0 = [0,v0\*np.cos(theta),0,v0\*np.sin(theta)]
23. t, x = integrate(F,0,x0,10,h)

26. import matplotlib.pyplot as plt
27. from printSoln import \*
29. xs = []; ys = []
30. for i in range(len(t)):
31. xs.append(x[i,0]); ys.append(x[i,2])
32. if x[i,2] < 0.: break
34. n = len(xs)-1
35. #print(t[n], xs[n], ys[n])
36. plt.plot(xs,ys,'-')
37. plt.grid()
38. plt.show()



2.

problem8.1.19.py

1. import numpy as np
2. import matplotlib.pyplot as plt
4. m = 20 # kg
5. c = 3.2e-4 # kg/m
6. g = 9.80665 # m/s^2
8. ## x'' = -(c/m)\*v\*x'
9. ## --- Rewrite the equations ---
10. ## y0' = y1
11. ## y1' = -(c/m)\*v\*y1
13. ## y'' = -(c/m)\*v\*y - g
14. ## --- Rewrite the equations ---
15. ## y2' = y3
16. ## y3' = -(c/m)\*v\*y3 - g
18. ## x0(0) = 0.0, x1(0) = v\*cos(theta), x2(0) = 0.0, x3(0) = v\*sin(theta)
19. ## x0(10) = 8000, x1(10) = ?, x2(10) = 0, x3(10) = ?
21. def F(t,x):
22. F = np.zeros(4)
23. v = ( x[1]\*\*2 + x[3]\*\*2 )\*\*0.5
24. F[0] = x[1]
25. F[1] = -(c/m)\*v\*x[1]
26. F[2] = x[3]
27. F[3] = -(c/m)\*v\*x[3] - g
28. return F
30. def r(u):
31. r = np.zeros(len(u))
32. ts,xs = integrate(F,tStart,initCond(u),tStop,h)
33. y = xs[len(ts) - 1]
34. r[0] = y[0] - 8000 ## x(t=10) = 8000
35. r[1] = y[2] - 0.0 ## y(t=10) = 0
36. return r
38. initCond = lambda u: np.array([0.0,u[0],0.0,u[1]])
40. from run\_kut4 import \*
41. ## Use newtonRaphson2 module for finding v0, theta
42. from newtonRaphson2 import \*
44. tStart = 0.0
45. tStop = 10.0
46. h = 0.001
48. ## Initial value for newtonRaphson2
49. v0 = 50
50. theta = np.pi/6
51. u = [v0\*np.cos(theta), v0\*np.sin(theta)]
53. ## Consider initial value
54. u = newtonRaphson2(r,u)
56. ts,xs = integrate(F,tStart,initCond(u),tStop,h)
58. #from printSoln import \*
59. #printSoln(ts,xs,freq)
61. ## Write new txt file for save solution
62. f = open('problem8.1.19.txt','w')
63. f.write('u '+str(u))
64. f.write('\nv0 '+str((u[0]\*\*2 + u[1]\*\*2)\*\*0.5))
65. f.write('\ntheta '+str(np.arctan(u[1]/u[0])))
66. f.write('\nx(10) '+str(xs[len(ts)-1,0]))
67. f.write('\nr(u) '+str(r(u)))
68. f.close()
70. plt.plot(xs[:,0],xs[:,2],'-')
71. plt.grid()
72. plt.savefig('problem8.1.19.png')

problem8.1.19.txt

1. u [853.48977441 50.14976188]
2. v0 854.9618667727328
3. theta 0.05869099740746082
4. x(10) 7999.999999999947
5. r(u) [-5.27506927e-11 3.42487076e-13]

v\_0 = 856 m/sec, theta\_0 = 0.0587 rad = 3.36 degree

problem8.1.19.png

