Python code and analysis

```
import numpy as np

## Comparing different algorithms to solve dy/dx = f(x,y)

def euler_method(function,x0,y0,x,stepsize):
    """
    function = dy/dx; x0,y0 are initial conditions; x is the point where y needs to be calculated.
    """
    x_array = np.arange(x0, x + stepsize, stepsize)
    y_array = np.zeros(len(x_array))
    y_array[0] = y0
    for i in range(1,len(x_array)):
        y_array[i] = y_array[i-1] + stepsize*function(x_array[i-1],y_array[i-1])

    return y_array[-1]

def partialx(function,x0,y0,stepsize):
    """
    Returns partial first derivative of the function "function" at the point x0,y0 with respect to x by considering points, one and two steps on either side of x0.
    """
    return (function(x0 - 2*stepsize, y0) - 8*function(x0 - stepsize, y0) + 8*function(x0 + stepsize, y0) - function(x0 + 2*stepsize, y0))/(12*stepsize)

def partialy(function,x0,y0,stepsize):
    """
    Returns partial first derivative of the function "function" at the point x0,y0 with respect to y by considering points, one and two steps on either side of y0.
    """
    return (function(x0, y0 - 2*stepsize) - 8*function(x0, y0 - stepsize) + 8*function(x0, y0 + stepsize) - function(x0, y0 + 2*stepsize))/(12*stepsize)

def taylor_method(function,x0,y0,x,stepsize):
    """
    function = dy/dx; x0,y0 are initial conditions; x is the point where y needs to be calculated.
    """
    x_array = np.arange(x0, x + stepsize, stepsize)
    y_array = np.zeros(len(x_array))
    y_array[0] = y0
    for i in range(1,len(x_array)):
        y_array[i] = y_array[i-1] + stepsize*function(x_array[i-1],y_array[i-1]) + (stepsize**2.0)*(partialx(function,x_array[i-1],y_array[i-1],10**(-7)) + function(x_array[i-1],y_array[i-1])*partialy(function,x_array[i-1],y_array[i-1],10**(-7)))

    return y_array[-1]

def adam_bashford_method(function,x0,y0,x,stepsize):
    """
    function = dy/dx; x0,y0 are initial conditions; x is the point where y needs to be calculated.
    """
    x_array = np.arange(x0, x + stepsize, stepsize)
    y_array = np.zeros(len(x_array))
    y_array[0] = y0
    y_array[1] = y0
    for i in range(2,len(x_array)):
        y_array[i] = y_array[i-1] + stepsize*(1.5*function(x_array[i-1],y_array[i-1]) - 0.5*function(x_array[i-2],y_array[i-2]))

    return y_array[-1]

def runge_kutta_method(function,x0,y0,x,stepsize):
    """
    function = dy/dx; x0,y0 are initial conditions; x is the point where y needs to be calculated.
    """
    x_array = np.arange(x0, x + stepsize, stepsize)
    y_array = np.zeros(len(x_array))
    y_array[0] = y0
    for i in range(1,len(x_array)):
        y_step = stepsize*function(x_array[i-1], y_array[i-1])
        y_array[i] = y_array[i-1] + stepsize*function(x_array[i-1] + (stepsize/2.0), y_array[i-1] + (y_step/2.0))

    return y_array[-1]
```

Results for y(1) and y(3) are given below (Δy is the difference from the analytical value):

h	Euler y(1)	Δy(1)	Taylor y(1)	Δy(1)	Adam y(1) Bashford	Δy(1)	Runge y(1) Kutta	Δy(1)
0.5	0.75	0.1435	0.421875	0.1847	0.625	0.01847	0.587891	0.01864
0.2	0.652861	0.04633	0.55319	0.05334	0.607383	0.0008524	0.604186	0.002345
0.1	0.628157	0.02163	0.583102	0.02343	0.606581	5.053e-05	0.605987	0.0005433
0.05	0.616984	0.01045	0.59562	0.01091	0.606522	9.105e-06	0.6064	0.0001309
0.02	0.610629	0.004098	0.602359	0.004172	0.606527	3.604e-06	0.60651	2.05e-05
0.01	0.608566	0.002035	0.604477	0.002054	0.60653	1.082e-06	0.606526	5.09e-06
0.005	0.607545	0.001014	0.605512	0.001019	0.60653	2.932e-07	0.606529	1.268e-06
0.002	0.606936	0.0004049	0.606125	0.0004056	0.606531	4.909e-08	0.60653	2.025e-07
0.001	0.606733	0.0002023	0.606328	0.0002025	0.606531	1.245e-08	0.606531	5.058e-08

h	Euler y(3)	Δy(3)	Taylor y(3)	Δy(3)	Adam y(3) Bashford	Δy(3)	Runge y(3) Kutta	Δy(3)
0.5	0.0	0.01111	0.094551	0.08344	0.02124	0.01013	0.02999	0.01888
0.2	0.00459	0.006519	0.026157	0.01505	0.012549	0.00144	0.012587	0.001478
0.1	0.007791	0.003318	0.016062	0.004953	0.01148	0.0003708	0.011409	0.0002999
0.05	0.009444	0.001665	0.01313	0.002021	0.011202	9.28e-05	0.011177	6.824e-05
0.02	0.010443	0.0006664	0.011828	0.0007191	0.011124	1.479e-05	0.011119	1.035e-05
0.01	0.010776	0.0003333	0.011455	0.0003461	0.011113	3.689e-06	0.011112	2.543e-06
0.005	0.010942	0.0001666	0.011279	0.0001698	0.01111	9.212e-07	0.01111	6.302e-07
0.002	0.011042	6.665e-05	0.011176	6.716e-05	0.011109	1.473e-07	0.011109	1.003e-07
0.001	0.011076	3.333e-05	0.011142	3.345e-05	0.011109	3.681e-08	0.011109	2.504e-08

Fortran90 code and analysis

```fortran
function f(x,y) result(z)
    double precision x, y, z
    z = -x*y
end function f

function euler_method(x0,y0,x,stepsize) result(e)
    double precision f, x0, y0, x, stepsize, x1, y1, e
    integer n, i
    n = (x - x0)/stepsize
    do 10 i = 0, n
        y1 = y0 + stepsize*f(x0, y0)
        x1 = x0 + stepsize
        x0 = x1
        y0 = y1
    10 continue
    e = y1
end function euler_method

function partialx(x0,y0,stepsize) result(ddx)
    double precision f, x0, y0, stepsize, ddx
    ddx = (f(x0-2.0*stepsize,y0)-8.0*f(x0-stepsize,y0)+8.0*f(x0 + stepsize,y0)-f(x0+2.0*stepsize,y0))/(12.0*stepsize)
end function partialx

function partialy(x0,y0,stepsize) result(ddy)
    double precision f, x0, y0, stepsize, ddy
    ddy = (f(x0,y0-2.0*stepsize)-8.0*f(x0,y0-stepsize)+8.0*f(x0,y0+stepsize)-f(x0,y0+2.0*stepsize))/(12.0*stepsize)
end function partialy

function taylor_method(x0,y0,x,stepsize) result(t)
    double precision f, x0, y0, x, stepsize, x1, y1, t, partialx, partialy, h
    integer n, i
```

```
    h = 10E-7
    n = (x - x0)/stepsize
    do 20 i = 0, n
        y1 = y0 + stepsize*f(x0,y0) + (stepsize**2.0)*(partialx(x0,y0,h) + f(x0,y0)*partialy(x0,y0,h))
        x1 = x0 + stepsize
        x0 = x1
        y0 = y1
    20 continue
    t = y1
end function taylor_method

function adam_bashford_method(x0,y0,x,stepsize) result(a)
    double precision f, x0, y0, x, stepsize, x1, y1, x2, y2, a
    integer n, i
    n = (x - x0)/stepsize
    x1 = x0 + stepsize
    y1 = y0 + stepsize*f(x0,y0)
    do 30 i = 1 ,n
        y2 = y1 + stepsize*(1.5*f(x1,y1) - 0.5*f(x0,y0))
        x2 = x1 + stepsize
        x0 = x1
        x1 = x2
        y0 = y1
        y1 = y2
    30 continue
    a = y2
end function adam_bashford_method

function runge_kutta(x0,y0,x,stepsize) result(r)
    double precision f, x0, y0, x, x1, y1, stepsize, k, r
    integer n, i
    n = (x - x0)/stepsize
    do 40 i = 1,n
        k = stepsize*f(x0,y0)
        y1 = y0 + stepsize*f(x0 + stepsize*0.5, y0 + k*0.5)
        x1 = x0 + stepsize
        x0 = x1
        y0 = y1
    40 continue
    r = y1
end function runge_kutta

program solve_diff_eqn

implicit none
double precision x0,y0,x,stepsize
double precision euler_method, taylor_method, adam_bashford_method, runge_kutta

x0 = 0
y0 = 1
x = 1
stepsize = 0.001

print*,euler_method(x0, y0, x, stepsize)
x0 = 0
y0 = 1

print*,taylor_method(x0, y0, x, stepsize)

x0 = 0
y0 = 1

print*,adam_bashford_method(x0, y0, x, stepsize)

x0 = 0
y0 = 1

print*,runge_kutta(x0, y0, x, stepsize)

end program solve_diff_eqn
```

Results for y(1) and y(3) are given below (Δy is the difference from the analytical value):

h	Euler y(1)	Δy(1)	Taylor y(1)	Δy(1)	Adam Bashford y(1)	Δy(1)	Runge Kutta y(1)	Δy(1)
0.5	0.375	0.231531	0.210937	0.395593	0.28125	0.325281	0.587890	0.01864
0.2	0.652861	0.04633	0.55319	0.05334	0.607383	0.0008524	0.724096	0.117565
0.1	0.628157	0.02163	0.583102	0.02343	0.606581	5.053e-05	0.666451	5.99e-02
0.05	0.616984	0.01045	0.59562	0.01091	0.606522	9.105e-06	0.636701	3.02e-02
0.02	0.598416	8.114e-03	0.590311	1.621e-02	0.594394	1.213e-02	0.60651	2.05e-05
0.01	0.602480	4.05e-03	0.598432	8.098e-03	0.600463	6.066e-03	0.606526	5.09e-06
0.005	0.604507	2.023e-03	0.602484	4.046e-03	0.603497	3.032e-03	0.606529	1.268e-06
0.002	0.606936	0.0004049	0.606125	0.0004056	0.606531	4.909e-08	0.607743	1.21e-03
0.001	0.606733	0.0002023	0.606328	0.0002025	0.606531	1.245e-08	0.607137	6.06e-04

h	Euler y(3)	Δy(3)	Taylor y(3)	Δy(3)	Adam Bashford y(3)	Δy(3)	Runge Kutta y(3)	Δy(3)
0.5	0.0	0.01111	0.141826	0.130717	1.525e-03	9.583e-03	0.02999	0.01888
0.2	0.00459	0.006519	0.026157	0.01505	0.012549	0.00144	2.161e-02	1.050e-02
0.1	0.007791	0.003318	0.016062	0.004953	0.01148	0.0003708	1.525e-02	4.148e-03
0.05	0.009444	0.001665	0.01313	0.002021	0.011202	9.28e-05	1.296e-02	1.854e-03
0.02	9.816e-03	1.292e-03	1.115e-02	4.728e-05	1.047e-02	6.344e-04	0.011119	1.035e-05
0.01	1.045e-02	6.565e-04	1.112e-02	1.159e-05	1.078e-02	3.251e-04	0.011112	2.545e-06
0.005	1.077e-02	3.307e-04	1.111e-02	2.875e-06	1.094e-02	1.646e-04	0.01111	6.326e-07
0.002	0.011042	6.665e-05	0.011176	6.716e-05	0.011109	1.473e-07	0.011175	6.692e-05
0.001	0.011076	3.333e-05	0.011142	3.345e-05	0.011109	3.681e-08	0.011142	3.339e-05