Name - Rakshya Sharma

Subject - ASTR-119

Assignment - Homework 6

Date - November 28, 2021

References - Professor Brant's code from session 11/4/2021

Performing the Cash-Karp implementation of a Runge-Kutta implementation

Cash-Karp Runge-Kutta method is the method of using weighted sum of the different derivative approximations that we make inorder to get very-high order ordinary differential equation integrator and we will do that with adaptive step-size control.

In [65]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

Defining a coupled set of ODEs to integrate

OED = Ordinary Differential Equations

In [66]:
```
def dfdx(x,f):

    y = f[0]
    z = f[1]

    dydx = np.zeros_like(f)
    dydx[0] = z
    dydx[1] = -1*y

    return dydx
```

Defining the core of the Cash-karp method

In [67]:
```python
def cash_karp_core_mv(x_i,y_i,nv,h,f):
    #cash karp is defined in terms
    #of weighting variables

    ni = 7
    nj = 6
    ci = np.zeros(ni)
    aij = np.zeros( (ni, nj) )
    bi = np.zeros(ni)
    bis = np.zeros(ni)

    #input values for ci, aij, bi, bis
    ci[2] = 1./5.
    ci[3] = 3./10.
    ci[4] = 3./5.
    ci[5] = 1.
    ci[6] = 7./8.

    #j = 1
    aij[2,1] = 1./5
    aij[3,1] = 3./40.
    aij[4,1] = 3./10.
    aij[5,1] = -11./54.
    aij[6,1] = 1631./55296.

    #j = 2
    aij[3,2] = 9./40.
    aij[4,2] = -9./10.
    aij[5,2] = 5./2.
    aij[6,2] = 175./512.

    #j = 3
    aij[4,3] = 6./5.
    aij[5,3] = -70./27.
    aij[6,3] = 575./13824.

    #j = 4
    aij[5,4] = 35./27.
    aij[6,4] = 44275./110592.

    #j = 5
    aij[6,5] = 253./4096.

    #bi
    bi[1] = 37./378.
    bi[2] = 0.
    bi[3] = 250./621.
    bi[4] = 125./594.
    bi[5] = 0.0
    bi[6] = 512./1771.


    #bis
    bis[1] = 2825./27648.
    bis[2] = 0.0
    bis[3] = 18575./48384.
    bis[4] = 13525./55296.
    bis[5] = 277./14336.
    bis[6] = 1./4.

    #define the k array
    ki = np.zeros((ni,nv))

    #compute ki
    for i in range(1,ni):
        #computer xn+1 for i
        xn = x_i + ci[i]*h

        #compute temp y
        yn = y_i.copy()
        for j in range(1,i):
            yn += aij[i,j]*ki[j,:]

        #get k
        ki[i,:] = h*f(xn,yn)

    #get ynpo, ynpo*
    ynpo = y_i.copy()
    ynpos = y_i.copy()
    #print("ni = ",ni, ynpo, ynpos)
    for i in range(1,ni):
        ynpo  += bi[i] *ki[i,:]
        ynpos += bis[i]*ki[i,:]

        #print(i, ynpo[0],ynpos[0])
        #print(i,ynpo[0], ynpos[0], bi[i]*ki[i,0],bis[i],*ki[i,0])

    #get erroe
    Delta = np.fabs(ynpo-ynpos)

    #print("INSIDE Delta", Delta, ki[:,0], ynpo, ynpos)
```

```
    #return new y and delta
    return ynpo, Delta
```

Defining an adaptive step size driver for Cash-Karp

In [68]:

```
def cash_karp_mv_ad(dfdx,x_i,y_i,nv,h,tol):

    #define a safey scale
    SAFETY = 0.9
    H_NEW_FAC = 2.0

    #set a maximum number of iterations
    imax = 1000

    #set an iteration variable
    i = 0

    #create an error
    Delta = np.full(nv,2*tol)

    #remember the step
    h_step = h

    #adjust the step
    while(Delta.max()/tol>1.0):

        #get our new y and error estimate
        y_ipo, Delta = cash_karp_core_mv(x_i, y_i, nv, h_step, dfdx)

        #if the error is too large, take a smaller step
        if(Delta.max()/tol>1.0):


            #our error is too large, take a smaller step
            h_step *= SAFETY * (Delta.max()/tol)**(-0.25)

        #check iteration
        if(i>=imax):
            print("Too many iterations in cash_karp_mv_ad()")
            raise StopIteration("Ending after i = ", i)

        #iterate
        i += 1

    #next time, try a bigger step
    h_new = np.fmin(h_step * (Delta.max()/tol)**(-0.9), h_step * H_NEW_FAC)


    #return the answer and step info
    return y_ipo, h_new, h_step
```

Defining a wrapper for Cash-Karp

In [69]:
```
def cash_karp_mv(dfdx, a,b,y_a,tol,verbose=False):

    #dfdx is the derivstive wrt x
    #a is the lower bound
    #b is the upper bound
    #y_a are the boundary condition at a
    #tol is the tolerance

    #define our starting step
    xi = a
    yi = y_a.copy()

    #define an initial starting step
    h = 1.0e-4 * (b-a)

    #set a max number of iterations
    imax = 1000

    #set a iteration variable
    i = 0

    #how many variables?
    nv = len(y_a)

    #set the initial conditions
    x = np.full(1,a)
    y = np.full( (1,nv), y_a)

    #set a flag
    flag = True

    #LOOP UNTIL WE REACH B
    while(flag):

        #calculate  y_i+1, step info
        y_ipo, h_new, h_step = cash_karp_mv_ad(dfdx, xi, yi, nv, h, tol)

        #update the step for next time
        h = h_new

        #prevent an overshoot
        if(xi+h_step>b):

            #limit step to end at b
            h = b - xi

            #recompute y_i+1
            y_ipo, h_new, h_step = cash_karp_mv_ad(dfdx,xi,yi,nv,h,tol)

            #we're done
            flag = False

        #update the values
        xi += h_step
        yi = y_ipo.copy()

        #add the step
        x = np.append(x,xi)
        y_ipo = np.zeros((len(x), nv))
        y_ipo[0:len(x)-1,:]= y[:]
        y_ipo[-1,:] = yi[:]
        del y
        y = y_ipo


        #prevent too many iterations
        if(i>=imax):
            print("Maximum iterations reached.")
            raise StopIteration("Iteration number = ",i)

        #iterate
        i += 1

        #output some information
        if(verbose):
            s = "i = %3d\tx = %9.8f\ty = %9.8f\th = %9.8f\tb = %9.8f" % (i, xi, yi[0],h_step,b)
            print(s)


        #if we're done, exit
        if(xi==b):
            flag = False

        #return the answer
    return x, y
```

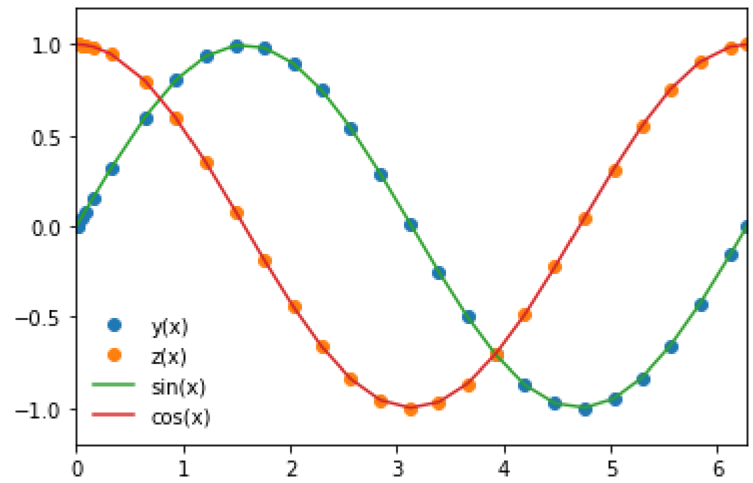
Perform the integration

In [70]:
```
a = 0.0
b = 2.0*np.pi
y_0 = np.zeros(2)
y_0[0] = 0.0
y_0[1] = 1.0
nv = 2

tolerance = 1.0e-6
x, y = cash_karp_mv(dfdx, a, b, y_0, tolerance, verbose = True)
plt.plot(x,y[:,0],'o',label='y(x)')
plt.plot(x,y[:,1],'o',label='z(x)')
plt.plot(x,np.sin(x),label='sin(x)')
plt.plot(x,np.cos(x),label= 'cos(x)')
plt.xlim([0,2*np.pi])
plt.ylim([-1.2,1.2])
plt.legend(frameon=False)
```

```
i =   1 x = 0.00062832  y = 0.00062832  h = 0.00062832  b = 6.28318531
i =   2 x = 0.00188496  y = 0.00188495  h = 0.00125664  b = 6.28318531
i =   3 x = 0.00439823  y = 0.00439822  h = 0.00251327  b = 6.28318531
i =   4 x = 0.00942478  y = 0.00942464  h = 0.00502655  b = 6.28318531
i =   5 x = 0.01947787  y = 0.01947664  h = 0.01005310  b = 6.28318531
i =   6 x = 0.03958407  y = 0.03957373  h = 0.02010619  b = 6.28318531
i =   7 x = 0.07979645  y = 0.07971180  h = 0.04021239  b = 6.28318531
i =   8 x = 0.16022123  y = 0.15953660  h = 0.08042477  b = 6.28318531
i =   9 x = 0.32107077  y = 0.31558279  h = 0.16084954  b = 6.28318531
i =  10 x = 0.64276986  y = 0.59941495  h = 0.32169909  b = 6.28318531
i =  11 x = 0.93739384  y = 0.80601851  h = 0.29462398  b = 6.28318531
i =  12 x = 1.20675386  y = 0.93446544  h = 0.26936002  b = 6.28318531
i =  13 x = 1.49426997  y = 0.99707369  h = 0.28751611  b = 6.28318531
i =  14 x = 1.76344767  y = 0.98150050  h = 0.26917769  b = 6.28318531
i =  15 x = 2.03076151  y = 0.89606839  h = 0.26731385  b = 6.28318531
i =  16 x = 2.29758389  y = 0.74731321  h = 0.26682237  b = 6.28318531
i =  17 x = 2.56844699  y = 0.54227799  h = 0.27086310  b = 6.28318531
i =  18 x = 2.84768738  y = 0.28969237  h = 0.27924039  b = 6.28318531
i =  19 x = 3.12869484  y = 0.01289739  h = 0.28100745  b = 6.28318531
i =  20 x = 3.39732623  y = -0.25295549 h = 0.26863139  b = 6.28318531
i =  21 x = 3.66411066  y = -0.49906425 h = 0.26678443  b = 6.28318531
i =  22 x = 3.93155133  y = -0.71032491 h = 0.26744067  b = 6.28318531
i =  23 x = 4.20391433  y = -0.87348904 h = 0.27236300  b = 6.28318531
i =  24 x = 4.48601277  y = -0.97448720 h = 0.28209844  b = 6.28318531
i =  25 x = 4.76275451  y = -0.99873304 h = 0.27674175  b = 6.28318531
i =  26 x = 5.03073667  y = -0.94975497 h = 0.26798215  b = 6.28318531
i =  27 x = 5.29729916  y = -0.83376264 h = 0.26656249  b = 6.28318531
i =  28 x = 5.56553774  y = -0.65761502 h = 0.26823858  b = 6.28318531
i =  29 x = 5.83966631  y = -0.42912109 h = 0.27412857  b = 6.28318531
i =  30 x = 6.12503238  y = -0.15749453 h = 0.28536607  b = 6.28318531
i =  31 x = 6.28318531  y = 0.00000015  h = 0.15815293  b = 6.28318531
```

Out[70]: <matplotlib.legend.Legend at 0x7f3c332a3b50>



In [ ]:

In [ ]:

In [ ]:

In [ ]: