***LAB EXPERIMENTS FOR XDIFFERENTIAL AND DIFFERENCE EQUATIONS***

***AAKASH-AGGARWAL***

***18BLC1060***

**Python 3.7**

**Compiler: Anaconda Navigator/Spyder version 4.0.1**

**Diagonalization of a given matrix**

**CODE:**

from sympy import \*

R = int(input("Enter the number of rows in matrix"))

C= int(input('enter the number of columns in matrix'))

# Initialize matrix

matrix = []

print("Enter the coefficients of variables entries row wise(press enter after each entry) :")

# For user input

for i in range(R):

a =[]

for j in range(R):

a.append(int(input()))

matrix.append(a)

A=Matrix(matrix)

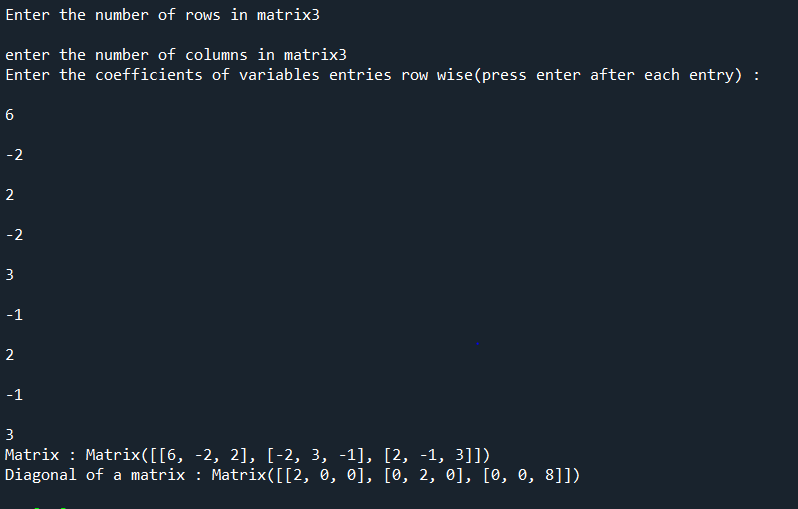
n=len(A)

print("Matrix : {} ".format(A))

P, D = A.diagonalize(A)

print("Diagonal of a matrix : {}".format(D))

**OUTPUT:**



**Quadratic to Canonical Form**

**CODE:**

import scipy.linalg as la

from sympy import \*

u,v,w=symbols('u v w')

a=int(input('Enter the coefficient of x1^2:'))

b=int(input('Enter the coefficient of x2^2:'))

c=int(input('Enter the coefficient of x3^2:'))

p=int(input('Enter the coefficient of x1\*x2:'))

q=int(input('Enter the coefficient of x1\*x3:'))

r=int(input('Enter the coefficient of x2\*x3:'))

A=[[a,p/2,q/2],[p/2,b,r/2],[q/2,r/2,c]]

print('matrix A',A)

results = la.eig(A)

#print('eigenvalues',results[0])

#print('eigen vectors',results[1])

eigvals, eigvecs = la.eig(A)

print('eigenvalues',eigvals)

print('eigen vectors',eigvecs)

eigvals = eigvals.real

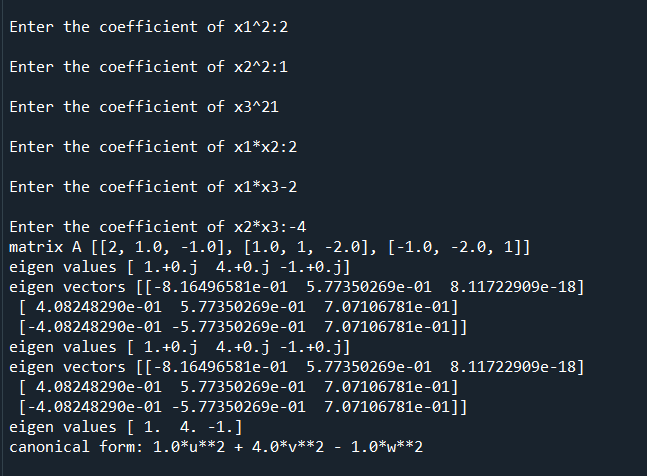
print('eigen values with only real part:',eigvals)

#print(results[0][0])

canonical\_form=(eigvals[0])\*(u\*\*2) + (eigvals[1])\*(v\*\*2) + (eigvals[2])\*(w\*\*2)

print('canonical form:',canonical\_form)

**OUTPUT:**



**Solving linear second order Homogeneous Ordinary  Differential Equations (ODE) with constant coefficients by method of variation of parameters**

**CODE:**

from sympy import \*

x,m,c1,c2,p,q,f,y1,y2=symbols('x m c1 c2 p q f y1 y2')

p1=int(input('Enter the coefficient of D2y:'))

p2=int(input('Enter the coefficient of Dy:'))

p3=int(input('Enter the coefficient of y:'))

f=input('Enter the non-homogeneous part:')

f=sympify(f)

expr=p1\*(m\*\*2)+p2\*m+p3

sol=solve(expr,m)

print(sol)

q=im(sol[0])

if len(sol)==1:

y1=exp(sol[0]\*x)

y2=t\*exp(sol[0]\*x)

elif(q!=0):

p=[re(sol[0]),re(sol[1])]

q=[im(sol[0]),im(sol[1])]

###for another solution,change p[0] -> p[1] and q[0] -> q[1] in (q!=0)condition if roots are complex

y1=exp(p[0]\*x)\*cos(q[0]\*x)

y2=exp(p[0]\*x)\*sin(q[0]\*x)

else:

y1=exp(sol[0]\*x)

y2=exp(sol[1]\*x)

print('y1 =',y1)

print('y2 =',y2)

y\_h=c1\*y1+c2\*y2

print(y\_h)

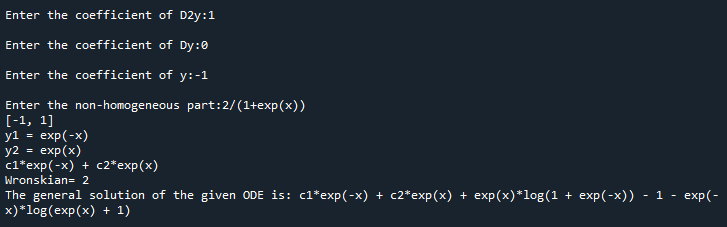
W=simplify(y1\*diff(y2,x)-y2\*diff(y1,x))

print('Wronskian=',W)

y\_p=-y1\*integrate(y2\*f/W,x)+y2\*integrate(y1\*f/W,x)

Y=(y\_h+y\_p)

print('The general solution of the given ODE is:',Y)

**OUTPUT:**

**Solving Cauchy-Euler differential equation**

**CODE:**

from sympy import \*

x,y,r,c1,c2,t=symbols('x y r c1 c2 t ')

p1=int(input('Enter the coefficient of x^2 d^2y/dx^2:'))

p2=int(input('Enter the coefficient of x dy/dx:'))

p3=int(input('Enter the coefficient of y:'))

f=input('Enter the function in terms of x:')

f=sympify(f)

f1=f.subs(x,exp(t))

expr=p1\*(r\*\*2)+(p2-p1)\*r+p3

sol=solve(expr,r)

print(sol)

q=im(sol[0])

if len(sol)==1:

y1=exp(sol[0]\*t)

y2=t\*exp(sol[0]\*t)

elif(q!=0):

p=[re(sol[0]),re(sol[1])]

q=[im(sol[0]),im(sol[1])]

###for another solution,change p[0] -> p[1] and q[0] -> q[1] in (q!=0)condition if roots are complex

y1=exp(p[0]\*t)\*cos(q[0]\*t)

y2=exp(p[0]\*t)\*sin(q[0]\*t)

else:

y1=exp(sol[0]\*t)

y2=exp(sol[1]\*t)

print('y1 =',y1)

print('y2 =',y2)

yc=y1\*c1+y2\*c2

print('yc =',yc)

w=y1\*diff(y2,t)-y2\*diff(y1,t)

print('w =',simplify(w))

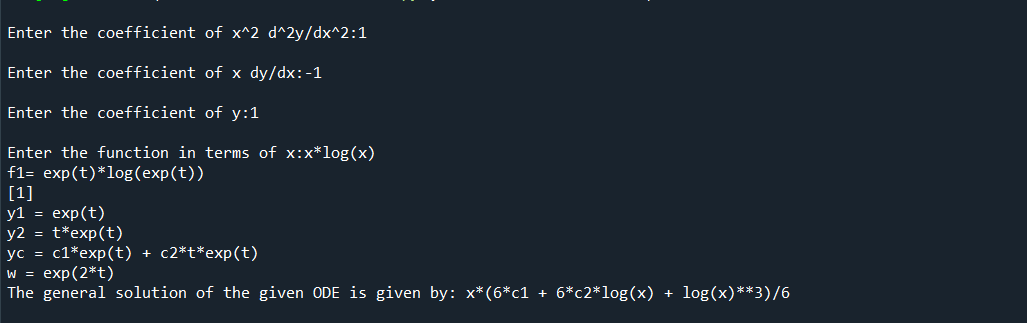
yp=-y1\*integrate(y2\*f1/(p1\*w),t)+y2\*integrate(y1\*f1/(p1\*w))

yy=yc+yp

y3=yy.subs(t,log(x))

print('The general solution of the given ODE is given by:',simplify(y3))

**OUTPUT :**



**Solving Cauchy-Legendre differential equation**

**CODE:**

from sympy import \*

x,c1,c2,r,t=symbols('x c1 c2 r t')

k1=int(input('Enter the coefficient of (ax+b)^2 d^2y/dx^2:'))

k2=int(input('Enter the coefficient of (ax+b) dy/dx:'))

k3=int(input('Enter the coefficient of y:'))

a=int(input('Enter the value of a:'))

b=int(input('Enter the value of b:'))

f=input('Enter f in terms of x:')

f=sympify(f)

f1=f.subs(x,(exp(t)-b)/a)

expr=(k1\*(a\*\*2))\*r\*\*2+((a\*k2)-(a\*\*2\*k1))\*r+k3

sol=solve(expr,r)

print('sol',sol)

q=im(sol[0])

if len(sol)==1:

y1=exp(sol[0]\*t)

y2=t\*exp(sol[0]\*t)

elif(q!=0):

p=[re(sol[0]),re(sol[1])]

q=[im(sol[0]),im(sol[1])]

##for another solution,change p[0] -> p[1] and q[0] -> q[1] in (q!=0)condition if roots are complex

y1=exp(p[1]\*t)\*cos(q[1]\*t)

y2=exp(p[1]\*t)\*sin(q[1]\*t)

else:

y1=exp(sol[0]\*t)

y2=exp(sol[1]\*t)

print('y1 =',y1)

print('y2 =',y2)

yc=c1\*y1+c2\*y2

w=y1\*diff(y2,t)-y2\*diff(y1,t)

print('Wronskian =',simplify(w))

p1=-y1\*integrate((y2\*f)/((a\*\*2)\*k1\*w),t)

p1=simplify(p1)

p2=y2\*integrate((y1\*f)/((a\*\*2)\*k1\*w),t)

p2=simplify(p2)

#print('p1:',p1)

#print('p2:',p2)

yp=p1+p2

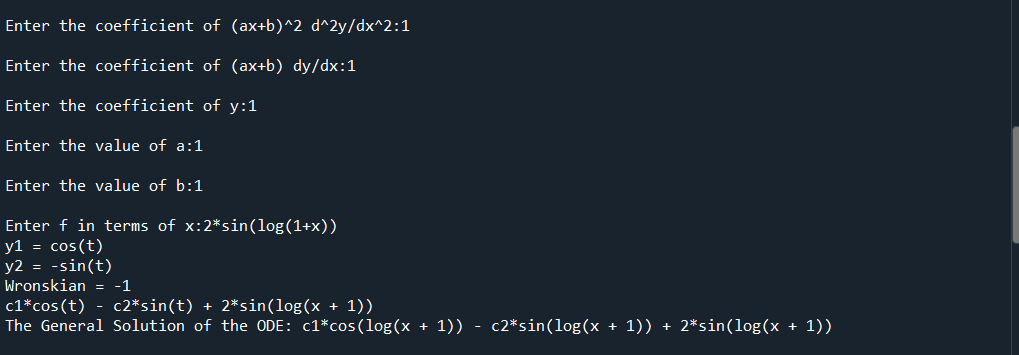
yy=yc+yp

yy=simplify(yy)

print(yy)

y3=yy.subs(t,log(a\*x+b))

print('The General Solution of the ODE:',simplify(y3))

**OUTPUT:** 

**Solving system of DE’s by matrix method**

**CODE:**

from sympy import \*

import scipy.linalg as la

import numpy as np

x,c1,c2,c3,c4=symbols('x c1 c2 c3 c4')

c=[c1,c2,c3,c4]

R = int(input("Enter the number of equations"))

# Initialize matrix

matrix = []

print("Enter the coefficients of variables entries row wise:") # eq1=6x1-x2 ; eq2=x1+x2 => 6(enter) -1(enter) 1(enter) 1(enter)

# For user input

for i in range(R):

a =[]

for j in range(R):

a.append(int(input()))

matrix.append(a)

A=matrix

n=len(A)

#print('n',n)

eigvals, eigvecs = la.eig(A)

eigvals = eigvals.real

print('eigen values',eigvals)

print('eigen vectors',eigvecs)

#C = int(input("Enter the number of equations"))

# Initialize matrix

matrix2 = []

print("Enter the function entries rowwise:") # eq1=6x1-x2 ; eq2=x1+x2 => 6(enter) -1(enter) 1(enter) 1(enter)

# For user input

for i in range(R):

a =[]

for j in range(1):

a.append((input()))

matrix2.append(a)

# For printing the matrix

for i in range(R):

for j in range(1):

matrix2[i][j]=sympify(matrix2[i][j])

print(matrix2[i][j], end = " ")

print()

f=matrix2

inv\_eig=np.linalg.inv(eigvecs)

# Program to multiply two matrices (vectorized implementation)

P=np.zeros([R,1])

P = np.dot(inv\_eig,f)

y=[]

p=[]

for i in range(0,n):

p.append( c[i]\*exp( eigvals[i]\*x) + exp( eigvals[i]\*x )\*integrate( exp(-1\*eigvals[i]\*x)\*P[i]) )

y.append(p)

print('y matrix:',y)

trans\_y=np.transpose(y)

result=np.zeros([R,1])

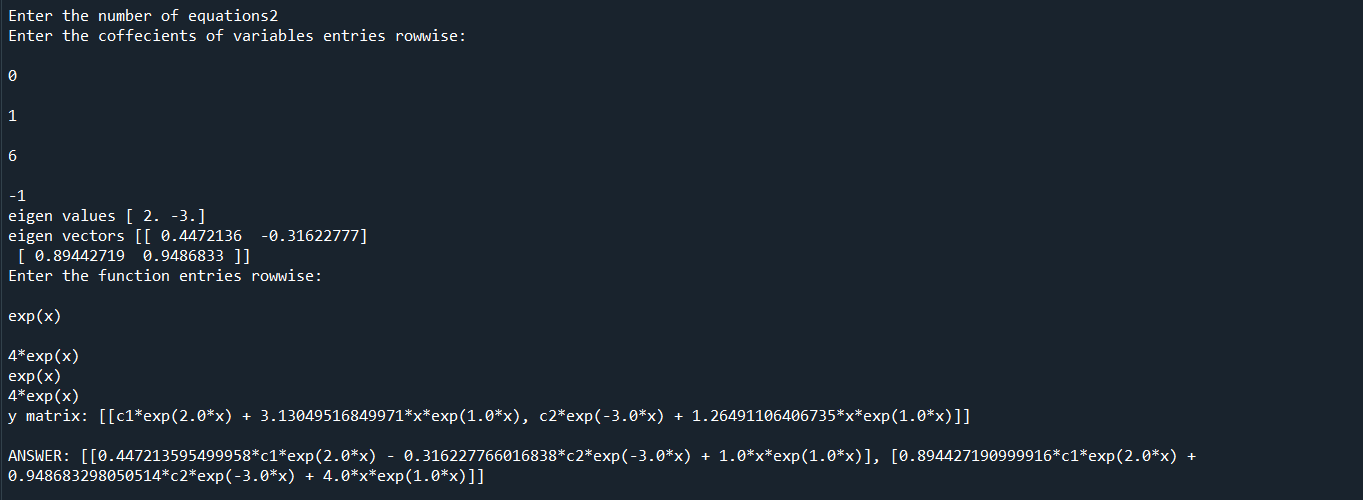
result = np.dot(eigvecs,trans\_y)

X=result

print(" ")

ANSWER=simplify(X)

print('ANSWER:',ANSWER)

**OUTPUT:**

**REPRESENTING A FUNCTION IN FOURIER SERIES**

**For symfit library**

#### **conda install**

* linux-64  v0.5.2
* osx-64  v0.5.2
* win-64  v0.5.2

To install this package with conda run one of the following:  
conda install -c conda-forge symfit  
conda install -c conda-forge/label/cf201901 symfit  
conda install -c conda-forge/label/cf202003 symfit

On compiler

**CODE:**

#valid for constant functions only

from symfit import parameters, variables, sin, cos, Fit

import numpy as np

import matplotlib.pyplot as plt

from sympy import \*

x,n=symbols('x n')

def fourier\_series(x, f, n=0):

"""

Returns a symbolic fourier series of order `n`.

:param n: Order of the fourier series.

:param x: Independent variable

:param f: Frequency of the fourier series

"""

# Make the parameter objects for all the terms

a0, \*cos\_a = parameters(','.join(['a{}'.format(i) for i in range(0, n + 1)]))

sin\_b = parameters(','.join(['b{}'.format(i) for i in range(1, n + 1)]))

print('a0',a0)

print('cos a wali:',\*cos\_a)

print('sin\_b wali:',sin\_b)

# Construct the series

series = a0 + sum(ai \* cos(i \* f \* x) + bi \* sin(i \* f \* x)

for i, (ai, bi) in enumerate(zip(cos\_a, sin\_b), start=1))

return series

x, y = variables('x, y')

w, = parameters('w')

no\_of\_functions=int(input('enter the number of functions:'))

f=[]

lower\_limit=[]

upper\_limit=[]

for i in range(no\_of\_functions):

f.append(input(f'enter functions {i} :'))

lower\_limit.append(float(input(f'Enter the lower limit of function(in float type) {i} :')))

upper\_limit.append(float(input(f'Enter the upper limit of function(in float type) {i} :')))

terms=int(input('Enter the number of terms'))

model\_dict = {y: fourier\_series(x, f=w, n=terms)}

print('series',model\_dict)

print('lower\_limit[0]',lower\_limit[0])

print('upper\_limit[0]',upper\_limit[no\_of\_functions-1])

l\_l=lower\_limit[0]

u\_l=upper\_limit[no\_of\_functions-1]

# Make step function data

xdata = np.linspace(l\_l,u\_l)

print('xdaata',xdata)

ydata = np.zeros\_like(xdata)

print('ydaata',ydata)

#ydata[xdata > 0] = 1

for i in range(no\_of\_functions):

l1=lower\_limit[i]

l2=upper\_limit[i]

ff=f[i]

ydata[np.logical\_and( xdata>=l1 , xdata<l2)]=ff

# Define a Fit object for this model and data

fit = Fit(model\_dict, x=xdata, y=ydata)

fit\_result = fit.execute()

#print('result:',fit\_result)

# Plot the result

plt.plot(xdata, ydata)

plt.plot(xdata, fit.model(x=xdata, \*\*fit\_result.params).y, ls=':')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

**OUTPUT:**

For terms=3:

enter the number of functions:3

enter functions 0 :-1

Enter the lower limit of function(in float type) 0 :-3.14

Enter the upper limit of function(in float type) 0 :0

enter functions 1 :2

Enter the lower limit of function(in float type) 1 :0

Enter the upper limit of function(in float type) 1 :1.57

enter functions 2 :1

Enter the lower limit of function(in float type) 2 :1.57

Enter the upper limit of function(in float type) 2 :3.14

Enter the number of terms3

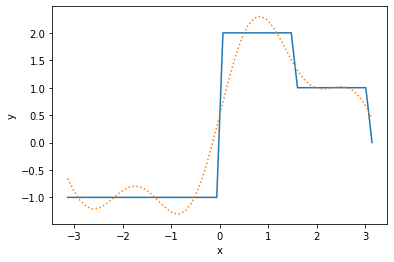
a0 a0

cos a wali: a1 a2 a3

sin\_b wali: (b1, b2, b3)

lower\_limit[0] -3.14

upper\_limit[0] 3.14



For terms=50:

enter the number of functions:3

enter functions 0 :-1

Enter the lower limit of function(in float type) 0 :-3.14

Enter the upper limit of function(in float type) 0 :0

enter functions 1 :2

Enter the lower limit of function(in float type) 1 :0

Enter the upper limit of function(in float type) 1 :1.57

enter functions 2 :1

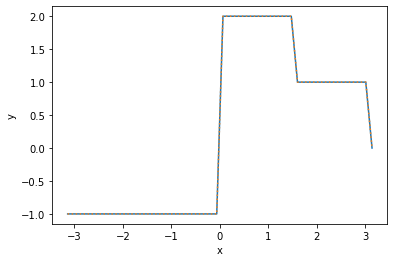
Enter the lower limit of function(in float type) 2 :1.57

Enter the upper limit of function(in float type) 2 :3.14

Enter the number of terms50

lower\_limit[0] -3.14

upper\_limit[0] 3.14



**Solution of Difference equation**

Solve

yn+2−2yn+1+yn

 with the initial conditions,

y(0)=1,y(1)=2

**CODE:**

import matplotlib.pyplot as plt

import numpy as np

from sympy import \*

n ,k1 ,k2 ,c\_1 ,c\_2 ,c\_3 ,c\_4 ,c\_5 ,s=symbols('n k1 k2 c\_1 c\_2 c\_3 c\_4 c\_5 s')

a=int(input('Enter the coefficient of y\_(n+2):'))

b=int(input('Enter the coefficient of y\_(n+1):'))

c=int(input('Enter the coefficient ofy\_n:'))

expr=a\*(s\*\*2)+b\*s+c

sol=solve(expr,s)

print(sol)

q=im(sol[0])

if len(sol)==1:

y1=sol[0]\*\*n

y2=n\*sol[0]\*\*n

elif(q!=0):

p=[re(sol[0]),re(sol[1])]

q=[im(sol[0]),im(sol[1])]

#

b=p[1]\*\*2 + q[1]\*\*2

rho=sqrt(b)

theta=np.arctan( abs( q[1]/p[1] ) )

y1=(rho\*\*n)\*cos(n\*theta)

y2=(rho\*\*n)\*sin(n\*theta)

else:

y1=sol[0]\*n

y2=sol[1]\*n

print('y1 =',y1)

print('y2 =',y2)

y\_0=int(input('y(0) :'))

y\_1=int(input('y(1):'))

yc=y1\*k1+y2\*k2

yc=simplify(yc)

print('yc',yc)

yc0=yc.subs(n,0);

#print('yc0;',yc0)

yc1=yc.subs(n,1);

#print('yc1;',yc1)

eq0=yc0-y\_0;

eq1=yc1-y\_1;

res=solve([eq0,eq1],(k1,k2))

#print('res',res)

k1\_1=res[k1]

k2\_1=res[k2]

#print('k1',k1)

y\_s=yc.subs(k1,k1\_1)

y\_s=y\_s.subs(k2,k2\_1)

#print('y\_s',y\_s)

def createList(r1, r2):

return np.arange(r1, r2+1, 1)

# Driver Code

r1=0

r2=int(input('enter the number of points(exanple 10):'))

m=createList(r1, r2)

#print(m)

y\_ss=[]

for i in range(r2+1):

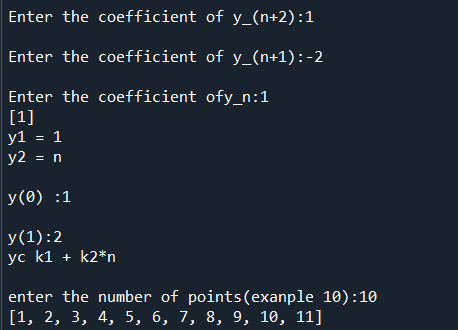
y\_ss.append( y\_s.subs(n,m[i]))

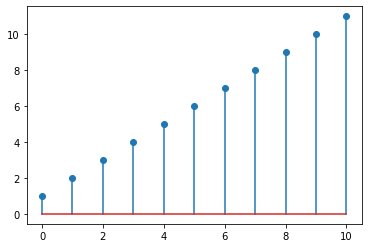
print(y\_ss)

plt.stem(m,y\_ss)

plt.show()

OUTPUT:





**Solving second order ODE by Laplace transform technique**

CODE

from sympy import \*

from sympy import symbols

from sympy.integrals import laplace\_transform

from sympy.integrals import inverse\_laplace\_transform

y,y1,y2,s,t,Y=symbols('y y1 y2 s t Y')

y=sympify('y(t)')

#y1=diff(y,t)

#print('y1:',y1)

a=int(input('enter the coefficient of second deri of y:'))

b=int(input('enter the coefficient of first deri of y:'))

c=int(input('enter the coefficient  of y:'))

f=(input('enter the RHS function:'))

d=int(input('enter the value of y(0):'))

e=int(input('enter the value of Dy(0):'))

f=sympify(f)

y1=diff(y,t)

y2=diff(y1,t)

eq=a\*y2 + b\*y1 + c\*y - f;

print('equation:',eq)

print(' ')

eqn=laplace\_transform(eq, t, s)

print('after laplace equation:',eqn)

print(' ')

eqn=sympify(eqn)

#eqn=eqn.subs(t,0)

#eqn=eqn.subs(y(0),d)

print('after replacing:',eqn)

print(' ')

eqn=eqn.subs(sympify('LaplaceTransform(y(t), t, s)'),Y)

print('after replacing:',eqn)

print(' ')

eqn=eqn.subs(sympify(y1),e)

print('after replacing:',eqn)

eqn=eqn.subs(sympify(y),d)

print(' ')

print('after replacing:',eqn)

print(' ')

collected\_expr = collect(eqn, Y)

Y1=simplify(solve(eqn,Y))

ans=inverse\_laplace\_transform(Y1,s,t)

OUTPUT:



1.Computing the harmonics  of  a continuous real-valued function

CODE:

from array import array

import numpy as np

import math

from sympy import \*

import matplotlib.pyplot as plt

x=symbols('x')

p=float(input('Enter the period length(in integer or float type only):'))

print('p',p)

l=p/2

n = int(input("Enter number of elements in vectors : "))

# Below line read inputs from user using map() function

X = list(map(float,input("\nEnter the X-vector : ").strip().split()))[:n]

Y= list(map(float,input("\nEnter the Y-vector : ").strip().split()))[:n]

#print("\nList is - ", X)

#print(Y)

N=len(X)

print(N)

r=int(input('Enter the number of terms in series:'))

a\_0=(2/N)\*sum(Y);

a=[]

b=[]

H=[]

add1=[]

add2=[]

for i in range(1,r+1):

    sum1=0

    sum2=0

    for j in range(N):

        sum1=sum1+(Y[j]\*math.cos(i\*math.pi\*X[j]/l))

        sum2=sum2+(Y[j]\*math.sin(i\*math.pi\*X[j]/l))

    add1.append(sum1)

    add2.append(sum2)

#print('add1',add1)

#print('add1',add2)

for i in range(r):

    a.append((2/N)\*add1[i])

    b.append((2/N)\*add2[i])

#print('a',a)

#print('b',b)

for i in range(r):

    H.append(a[i]\*cos((i+1)\*math.pi\*x/l) + b[i]\*sin((i+1)\*math.pi\*x/l))

print('H',H)

HS=(a\_0)/2+sum(H);

#print('Harmonic series is given by',HS)

HS=simplify(HS)

print('Harmonic series is given by',HS)

u = np.arange(0, p+1,0.1)

v=[]

for i in range(len(u)):

    v.append(HS.subs(x,u[i]))

plt.plot(u,v)

plt.plot(X,Y,'ro')

plt.xlim([0, p])

plt.show()

OUTPUT:

Enter the period length:6

p 6.0

Enter number of elements : 6

Enter the X-vector : 0 1 2 3 4 5

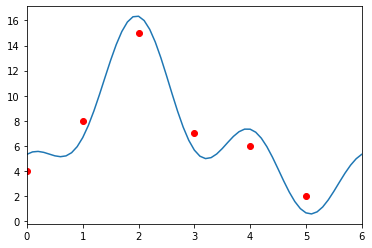
Enter the Y-vector : 4 8 15 7 6 2

6

Enter the number of terms in series:3

H [4.33012701892219\*sin(1.0471975511966\*x) - 2.83333333333333\*cos(1.0471975511966\*x), -0.866025403784437\*sin(2.0943951023932\*x) - 1.5\*cos(2.0943951023932\*x), 5.71914493359824e-16\*sin(3.14159265358979\*x) + 2.66666666666667\*cos(3.14159265358979\*x)]

Harmonic series is given by 4.33012701892219\*sin(1.0471975511966\*x) - 0.866025403784437\*sin(2.0943951023932\*x) + 5.71914493359824e-16\*sin(3.14159265358979\*x) - 2.83333333333333\*cos(1.0471975511966\*x) - 1.5\*cos(2.0943951023932\*x) + 2.66666666666667\*cos(3.14159265358979\*x) + 7.0



2. Power series of a ODE ( ordinary point)

CODE:

from sympy import \*

import sympy as sym

from sympy.solvers.solveset import linsolve

x,r,d\_0,c\_0,c\_1,c\_2,c\_3,c\_4,c\_5=symbols('x r d\_0 c\_0 c\_1 c\_2 c\_3 c\_4 c\_5')

print('Input the coefficients of DE as constants or function of x')

p1=input('Coefficient of D2y:')

p2=input('Coefficient of Dy:')

p3=input('Coefficient of y:')

c=[c\_0,c\_1,c\_2,c\_3,c\_4,c\_5]

p1=sympify(p1)

p2=sympify(p2)

p3=sympify(p3)

y=0

for i in range(0,6):

    y=y+c[i]\*x\*\*(i)

print('y=',y)

y=sympify(y)

dy=diff(y,x)

d2y=diff(dy,x)

ode=p1\*d2y+p2\*dy+p3\*y

ode=expand(ode)

print('ode',ode)

ps=collect(ode,x)

ps=simplify(ps)

print('ps',ps)

d=[]

for i in range (6+1):

    d.append(ps.coeff(x,i))

print('d',d)

fin\_list = solve((d[0],d[1],d[2],d[3]),(c\_2,c\_3,c\_4,c\_5))

c = fin\_list

z=y.subs(c\_2,c[c\_2])

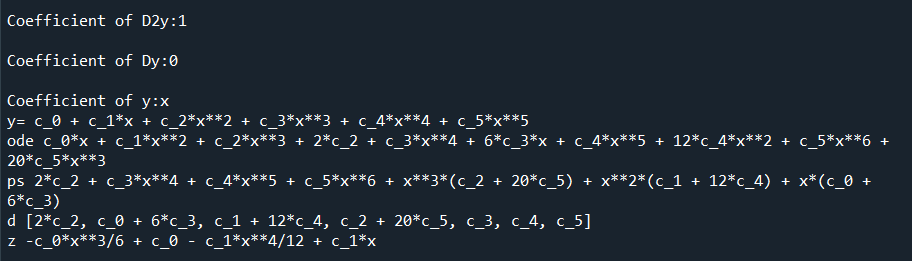
z=z.subs(c\_3,c[c\_3])

z=z.subs(c\_4,c[c\_4])

z=z.subs(c\_5,c[c\_5])

print('z',z)

OUTPUT:



3. Power series of an ODE about a regular singular point

**CODE:**

from sympy import \*

import sympy as sym

x,r,d\_0,c\_0,c\_1,c\_2,c\_3,c\_4,c\_5=symbols('x r d\_0 c\_0 c\_1 c\_2 c\_3 c\_4 c\_5')

print('Input the coefficients of DE as constants or function of x')

p1x=input('Coefficient of D2y:')

p1x=sympify(p1x)

p2x=input('Coefficient of Dy:')

p2x=sympify(p2x)

p3x=input('Coefficient of y:')

p3x=sympify(p3x)

c=[c\_0,c\_1,c\_2,c\_3,c\_4,c\_5]

y=0

#p=0

for i in range(0,6):

    y=y+c[i]\*x\*\*(i)

print('y=',y)

Px=p2x/p1x

print('Px=',Px)

Qx=p3x/p1x

print('Qx=',Qx)

px=simplify(x\*Px)

print('px=',px)

qx=simplify((x\*\*2)\*Qx)

print('qx=',qx)

a\_0=px.subs(x,0)

print('a\_0',a\_0)

b\_0=qx.subs(x,0)

print('b\_0',b\_0)

expr=r\*(r-1)+a\_0\*r+b\_0

sol=solve(expr,r)

print('sol',sol)

print(sol[0])

d=[]

y\_y=[]

for i in range(2):

    y1=y\*x\*\*sol[i]

    y1=expand(y1)

    print('y1',y1)

    y\_y.append(y1)

    dy1=diff(y1,x)

    d2y1=diff(dy1,x)

    ode1=p1x\*d2y1+p2x\*dy1+p3x\*y1

    ode1=ode1/x\*\*sol[i];

    ode1=expand(ode1)

    #print('ode1',ode1)

    ps1=collect(ode1,x)

    ps1=simplify(ps1)

    #print('ps1',ps1)

    d1=[]

    for i in range (6+1):

        d1.append(ps1.coeff(x,i))

    #print('d1',d1)

    d.append(d1)

print(' ')

print('d',d)

fin\_list = solve((d[0][0],d[0][1],d[0][2],d[0][3],d[0][4]),(c\_1,c\_2,c\_3,c\_4,c\_5))

c1 = fin\_list

z1= y\_y[0].subs(c\_1,c1[c\_1])

z1=z1.subs(c\_2,c1[c\_2])

z1=z1.subs(c\_3,c1[c\_3])

z1=z1.subs(c\_4,c1[c\_4])

z1=z1.subs(c\_5,c1[c\_5])

print(' ')

print('The particular solution of the given ODE around x=0 is given by: ')

print('z1',z1)

print(' ')

fin\_list = solve((d[1][0],d[1][1],d[1][2],d[1][3],d[1][4]),(c\_1,c\_2,c\_3,c\_4,c\_5))

c2 = fin\_list

z2=y\_y[1].subs(c\_1,c2[c\_1])

z2=z2.subs(c\_2,c2[c\_2])

z2=z2.subs(c\_3,c2[c\_3])

z2=z2.subs(c\_4,c2[c\_4])

z2=z2.subs(c\_5,c2[c\_5])

print('The particular solution of the given ODE around x=0 is given by: ')

print('z2',z2)

**OUTPUT:**

