Homework 12

November 27, 2020

---

1 Problem 3

From the wave equation, we will determine an appropriate numerical solution by obtaining the first and second order differentiation matrices. To describe our general strategy, we will make use of the finite difference package that was discussed in OH, in which we will obtain the first and second order difference matrices that respectively correspond to the first and second derivatives of our solution to the wave equation. In particular, it is important to note that we first must define the Gauss-Lobatto Grid over the interval $[-1, 1]$ over which we want to determine a solution to the wave equation.

As shown in our code at the end, we were able to implement this grid successfully. Besides defining the grid, we made use of the function, finitediff.get_weights, which allowed us to recover individual columns from the first and order second difference matrices ($D^{(1)}$, $D^{(2)}$). Indeed, our code demonstrates that the differentiation matrices that we got, in the case below for $N = 8$, were of the form

$$\begin{bmatrix} -9.98274915\text{e}+00 & 1.77501971\text{e}+01 & -1.49848302\text{e}+01 & 1.42194634\text{e}+01 & -1.39024380\text{e}+01 & 1.37501971\text{e} \\ -1.19891237\text{e}+00 & -2.70710678\text{e}+00 & 6.49660576\text{e}+00 & -4.82842712\text{e}+00 & 4.3318213\text{e}+00 & -4.12132034 \\ 2.27844661\text{e}-01 & -1.46247780\text{e}+00 & -1.39574364\text{e}+00 & 4.23304033\text{e}+00 & -2.92647737\text{e}+00 & 2.53752220\text{e} \\ \vdots & & & & & \end{bmatrix}$$

From the matrix above, we have displayed what we introduced at $D1$, which in the case for $N = 8$ that we carried out successfully mirrored behavior in the derivative of the actual solution $u(x, t)$ to the wave equation.

At more of a superficial glance, we observe that the differentiation matrices clearly differ for different N. For clarification, we have included more of the matrices for different N that we tried in the Appendix. But even from the differentiation matrix above, we can see how applying the get weights method from Python to the individual points on the Gauss-Lobatto Grid that we defined gives us a clear result on which we can transform the PDE to an ODE that we can solve componentwise, with the usual ivp solvers that we made use of in HWS 7,8 , etc. Below, we have produced some plots that give us several perspectives from which we can analyze the effectiveness of the spectral method for the wave equation. As expected, for certain values of N that we tried, we saw that our method was more effective for higher N, in which we even clearly saw from the entries of each differentiation matrix that our numerical solution that we obtained for the wave equation was more smooth, for higher N.

We proceeded to solve the given PDE by reducing it to an ODE, to which we applied the odeint method from Python. This was done by expanding the solution in x,

$$y(x, t) = \sum_j C_j(x) y_j(t)$$

and writing the x derivative in vector form

$$\frac{\partial y}{\partial x}|_i = \sum_j D_{ij}^{(1)} y_j(t)$$

then substituting into the wave equation,

$$\frac{\partial^2 y}{\partial t^2} = \frac{\partial^2 y}{\partial x^2}$$
$$\ddot{y}_i = \sum_j D_{ij}^{(2)} y_j$$

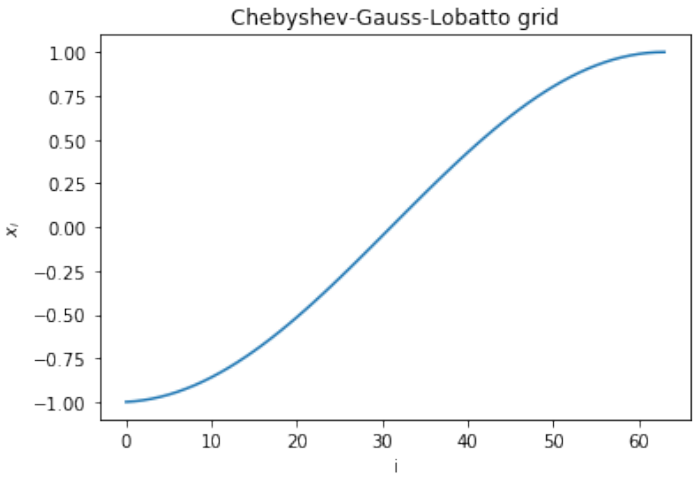
or equivalently as a vector equation

$$\ddot{\vec{y}} = D^{(2)} \cdot \vec{y}$$

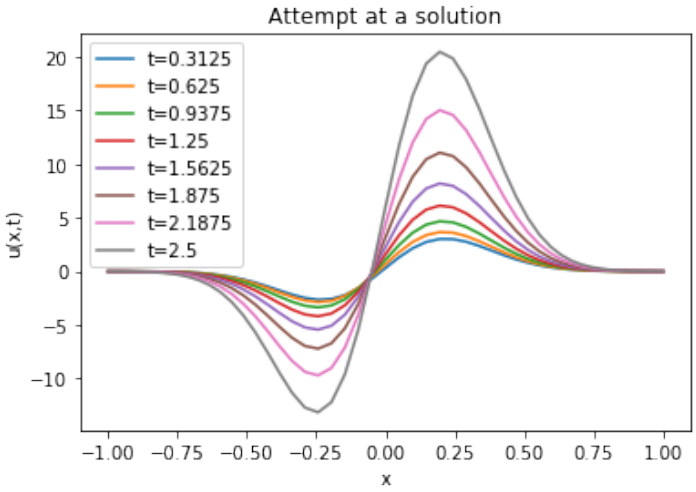
This results in N coupled second order ODEs, which we decomposed into $2N$ first order ODEs by writing $\vec{y_1} = \vec{y}$ and $\vec{y_2} = \dot{\vec{y}}$. So we have $\dot{\vec{y_1}} = \vec{y_2}$ and $\dot{\vec{y_2}} = D^{(2)} \cdot \vec{y_1}$. We then used scipy.integrate.odeint to solve these equations given the initial condition of $u(x,0) = e^{-\alpha x^2}$ and $u'(x,0) = 2\alpha x e^{-\alpha x^2}$, $\alpha = 10$.

However, we found that we could not implement this correctly to produce a good result. This first approach we took was to input arrays into odeint and do the matrix multiplication to get $\dot{y}_1$ and $\dot{y}_2$. This resulted in nonsensical answers, where the solution suddenly grew to enormous values (10^{200} for example), or would vanish to zero almost immediately. So we then tried to integrate each element of $\vec{y}$ forward in time. This time we got an answer with a reasonable order of magnitude but completely wrong behavior. We suppose this elementwise approach fails because it doesn't correctly (and cannot) recalculate the second x derivative when u is updated, because u is hidden within the integrator method.

With this general strategy and implementation that we use to get the differentiation matrices of first and second order, we will now show a few plots of our numerical results. First, we exhibit the plot below.

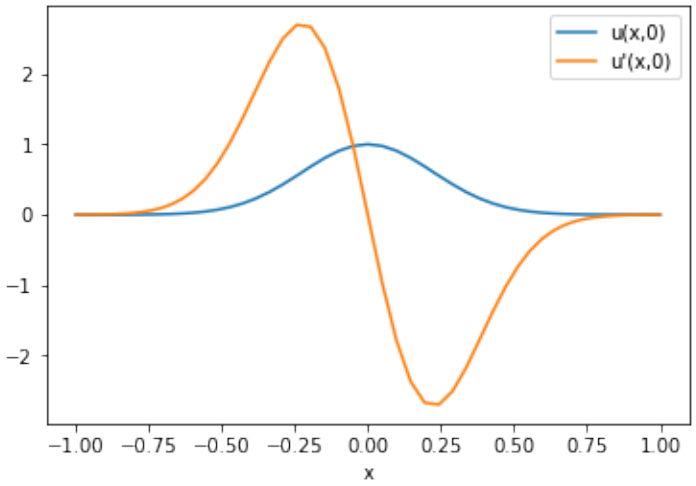


This plot above shows how we created the appropriate grid before applying any of the methods from the finitediff package in Python. Next, we will show a few plots of the numerical results that we obtained. By and large, we observe that the spectral method is clearly accurate for larger N, but we nevertheless experienced difficulty in implementing the last step of solving the PDE that we got from the differentiation matrices.



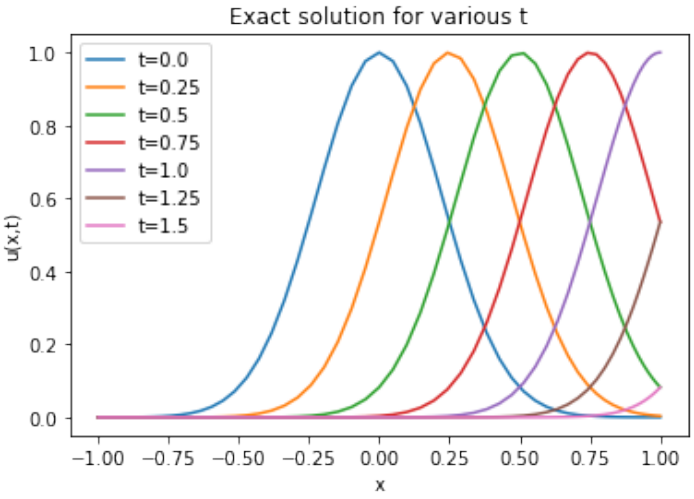
This plot above demonstrates different solutions to the wave equation that we have numerically determined for different t. From individual plots of each solution for some fixed t, we observe that we are able to obtain accurate results for the t that we choose for an initial starting time. Furthermore, we also observe that the plots of our numerical solutions above are evidence to the accuracy of the differentiation matrices of the type (such as for $N = 8$ and other N) that we have given at the beginning.

In addition to this plot, it is important to take into account the initial condition for the derivative. To impose this condition, we also compared plots of the solution u and its derivative u' for $x = 0$. We have the following plot below.



This plot above demonstrates to us that our numerical method of PS collocation can obtain much higher accuracy and efficiency with these results. But then again, the method is pseudo-spectral because it can only execute the procedures that we have given in the code in our Appendix because we can only deal with a finite number of rows and columns from the differentiation matrix.

Another plot below demonstrates a slightly different view of the curvature for the numerical solution that we obtained from the array. To establish some points of comparison with our numerical results, we decided to include plots of the exact solutions, for different t, below.



This final plot above shows us that the numerical solution that we have obtained from the PS Collocation method, again from differential initial times that we have prescrbied onto the PDE, are accurate to some degree. This completes our analysis of the PDE, in which we were unable to implement the last part with solving the ODE from the differentiation matrices.

2 Code

```
import numpy as np
import matplotlib.pyplot as plt
from finitediff import get_weights
from scipy.integrate import odeint

N=64
mesh=np.cos(np.pi*np.arange(N)/N)
mesh.sort()

a=10
x=mesh
u0=np.exp(-a*x**2)
up0=-2*a*x*np.exp(-a*x**2)

D1=np.zeros((N,N))
D2=np.zeros((N,N))
for i in range(N):
    D1[i,:]=get_weights(x,x[i],maxorder=1)[:,1]
    D2[i,:]=get_weights(x,x[i],maxorder=2)[:,2]

# Per element
def f(y,t):
    return([y[1],y[0]])
ts=np.linspace(0,2.5,9)
u=np.zeros((N,len(ts)))
for i in range(N):
```

```
    u[i,:]=(odeint(f,[-up0[i],u0[i]],ts,rtol=1e-10))[:,0]
for j in range(1,9):
    plt.plot(x,u[:,j],label='t='+str(ts[j]))
plt.title('Attempt at a solution')
plt.ylabel('u(x,t)')
plt.xlabel('x')
plt.legend(loc=2)

# Exact
ts=np.linspace(0,1.5,7)
for t in ts:
    uexact=np.exp(-a*(x-t)**2)
    plt.plot(x,uexact,label='t='+str(t))
plt.legend()
plt.title('Exact solution for various t')
plt.ylabel('u(x,t)')
plt.xlabel('x')

# Vectors
def F(y,t):
    u=y[N:] #u(t)
    z=y[:N] #u'(t)
    # return u'=z and z'=D2*u
    return(np.concatenate((z,np.dot(D2,u))))
ts=np.linspace(0,2.5,26)
y=odeint(f,np.concatenate((u0,up0)),ts,rtol=1e-8)
```