This program numerically solves flavor evolution of neutrinos and anti-neutrinos emmitted from an infinite line sources. The system, that is, occupation numbers and fluxes are assumed to be stationary and homogeneous. The partial differential equation describing this setup is as follows

$$i(v_x\partial_x + v_z\partial_z)\rho_{E,\mathbf{v}} = [H, \rho_{E,\mathbf{v}}]$$

where

$$H = \frac{M^2}{2E} + v^\mu \Lambda_\mu + \int d\Gamma'\ v^\mu v'_\mu \rho_{E',\mathbf{v}'}$$

An infinite source cannot be put into a computer so we impose a periodic boundary condition.

In [187]:

```
import matplotlib
%matplotlib inline
import numpy as np
from scipy.integrate import ode
import matplotlib.pyplot as plt
import matplotlib.lines as mline
import seaborn as sns
```

Following part describes the M^2 matrix. 'theta' is the vacuum mixing angle.

In [209]:

```
m1 = 10*0.25*0.25
m2 = 0
theta = 0.00
U = np.array([[np.cos(theta), np.sin(theta)], [-np.sin(theta), np.cos(theta)]])
vacuum = np.dot(U, np.dot(np.diag([m1,m2]), np.transpose(U)))
vacuum = np.diag(np.diag(vacuum))
print(vacuum)
```

```
[[0.625 0.   ]
 [0.    0.   ]]
```

The matter term assuming that vector part of Λ is zero.

In [210]:

```
factor = 0
n_e = 1
n_mu = 0.5
matter = factor*np.diag([n_e, n_mu])
```

All the modes present in the system. 'mu' represents the density of the neutrinos. 'Kin_vars' stores energy and the velocity of all the modes. 'Vel_arr_4' has four velocity of all the modes.

In [211]:

```
mu = 20/0.72
Nmodes = 4
Kin_vars = np.ones((Nmodes,3)) # eneergy, v_x, v_z
Kin_vars[0] = [1, 0.6, 0.8]
Kin_vars[1] = [-1, 0.6, 0.8]
Kin_vars[2] = [1, -0.6, 0.8]
Kin_vars[3] = [-1, -0.6, 0.8]
Vel_arr_4 = Kin_vars.astype(np.complex64) # i, v_x, v_z
Vel_arr_4[:,0] = 1j
# print(Kin_vars)
# print(Vel_arr_4)
```

The initial occupations numbers for all the modes. (Can be read from a data file)

In [212]:

```
occupations = np.zeros((Nmodes,2))
occupations[0] = [1.25/2, 0]
occupations[1] = [-0.75/2, 0]
occupations[2] = [1.25/2, 0]
occupations[3] = [-0.75/2, 0]
init_dens = []
for i in range(Nmodes):
    init_dens.append(np.reshape(np.diag(occupations[i]), -1))
init_dens = np.array(init_dens).astype(np.complex64)
# print(init_dens)
```

Discretizing in x and defining initial density matrix with seeded perturbations at $x_i's$.

In [213]:

```
L = 100
Ndivs = 1000
deltax = L/Ndivs
x_arr = np.arange(0, L, deltax)

def perturb(mat, div_num):
    N_mats = np.reshape(mat, (Nmodes, 2, 2)).astype(np.complex64)
    k = 2*np.pi/L
    for i in range(Nmodes):
#         print(x_arr[div_num])
        N_mats[i][0][1] = (np.random.uniform(-0.005,0.005) + 1j*np.random.uniform(-0.005,0.
        N_mats[i][1][0] = np.conj(N_mats[i][0][1])
    return np.reshape(N_mats, -1)

tot_init_vec = []
for i in range(Ndivs):
    tot_init_vec.extend(perturb(init_dens, i))
tot_init_vec = np.array(tot_init_vec)
print(tot_init_vec)
tot_init_vec = np.reshape(tot_init_vec, -1)
```

```
[ 6.2500000e-01+0.0000000e+00j -7.2431883e-10-2.7365681e-08j
 -7.2431883e-10+2.7365681e-08j ... -3.7566465e-08+1.5937744e-08j
 -3.7566465e-08-1.5937744e-08j  0.0000000e+00+0.0000000e+00j]
```

Vacuum and matter hamiltonians at each of the $x_i's$ for all modes.

In [214]:

```
vacPart = []
for t1 in range(Ndivs):
    for t2 in range(Nmodes):
        vacPart.append(vacuum/Kin_vars[t2, 0])

vacPart = np.reshape(np.array(vacPart), (Ndivs, Nmodes, 2, 2))
print(vacPart.shape)
# print(vacPart)

matterPart = []
for t1 in range(Ndivs*Nmodes):
    matterPart.append(matter)

matterPart = np.reshape(np.array(matterPart), (Ndivs, Nmodes, 2, 2))
print(matterPart.shape)
# print(matterPart)
```

```
(1000, 4, 2, 2)
(1000, 4, 2, 2)
```

After discretiziation in x the differential equations become

$$v_z \partial_z \rho_{E,\mathbf{v}}(x_i) = -i[H, \rho_{E,\mathbf{v}}] - v_x \frac{\rho_{E,\mathbf{v}}(x_{i+1}) - \rho_{E,\mathbf{v}}(x_{i-1})}{2\Delta x}$$

The next piece is the rhs function of the above equation for all the modes and points. 'vel_tensor' contains $\int d\Gamma v_\mu \rho_{E,\mathbf{v}}(x_i)$ at all the points on x axis, so it is a (Ndiv x 3 x 2 x 2) tensor. 'SelfPart' is $\int d\Gamma'\ v^\mu v'_\mu \rho_{E',\mathbf{v}'}(x_i)$.

In [229]:

```
def ConvToMat(arr):
    return np.reshape(arr, (2,2))

def func(t, tot_vec):
    tot_mat = np.apply_along_axis(ConvToMat, 2, np.reshape(tot_vec, (Ndivs, Nmodes, 4))) #r
    vel_tensor = np.einsum('ij,ki...->kj...', Vel_arr_4, tot_mat)
    SelfPart = -np.einsum('ij,kj...->ki...', Vel_arr_4, vel_tensor)
    Ham_array = mu*SelfPart+matterPart+vacPart
    Commutators = -1j*(np.einsum('ijkl,ijlm->ijkm', Ham_array, tot_mat) - np.einsum('ijkl,i
    Commutators1 = np.einsum('ij...,j->ij...', Commutators, 1/(Vel_arr_4[:,2]))
    gradient = np.zeros((Ndivs, Nmodes, 2, 2)).astype(np.complex64)
    gradient[1:Ndivs-1] = (tot_mat[2:Ndivs] - tot_mat[0:Ndivs-2])/(2*deltax)
    gradient[0] = (tot_mat[1] - tot_mat[-1])/(2*deltax)
    gradient[Ndivs-1] = (tot_mat[0] - tot_mat[Ndivs-2])/(2*deltax)
    gradient1 = np.einsum('ij...,j->ij...', gradient, (Vel_arr_4[:,1])/Vel_arr_4[:,2])

    answer = np.reshape(-gradient1 + Commutators1, -1)
    return answer
```

The main body of the code. All the data is stored in 'arr'.

In [246]:

```
solver = ode(func).set_integrator(name = 'zvode', method = 'BDF', nsteps = 1000000, atol =
solver.set_initial_value(tot_init_vec, 0)
z_final = 13
deltaz = 0.1
z_arr = np.arange(0, z_final, deltaz)

arr = []
for i in range(len(z_arr)):
    if(i%10 == 0):
        print(i)
    solver.integrate(solver.t+deltaz)
    arr.append(np.abs(solver.y))
arr = np.array(arr)
```

```
0
10
20
30
40
50
60
70
80
90
100
110
120
```

To present the off-diagonal element of one of the modes as a heatmap.

In [267]:

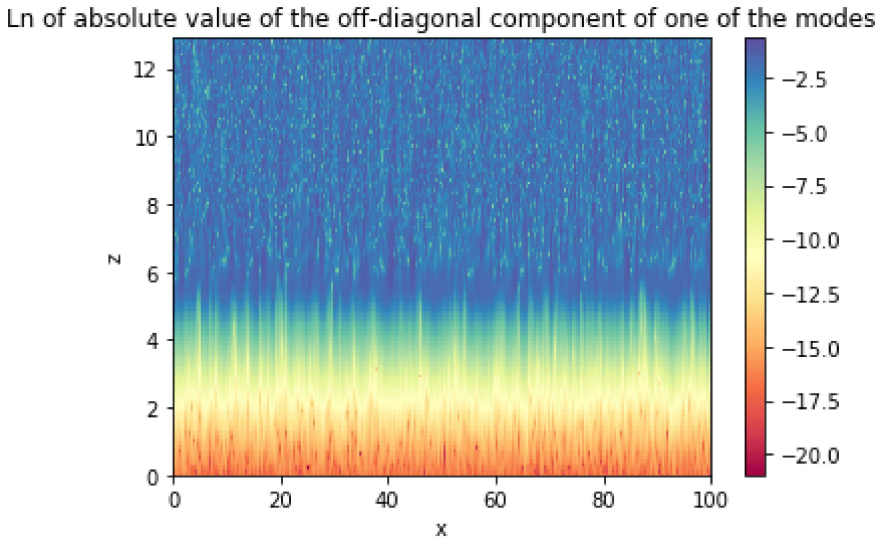
```
arr = np.reshape(arr, (len(z_arr), Ndivs, Nmodes, 2, 2))
print(arr.shape)
offdiag_arr = np.log(arr[:, :, 3, 0, 1])
diag_arr = arr[:, :, 3, 0, 0]
print(offdiag_arr.shape)

x_data, z_data = np.meshgrid(x_arr, z_arr)
plt.pcolormesh(x_data, z_data, offdiag_arr, cmap = plt.get_cmap("Spectral"))
plt.colorbar()
plt.xlabel("x")
plt.ylabel("z")
plt.title("Ln of absolute value of the off-diagonal component of one of the modes")
# plt.plot(z_arr, diag_arr[:,0])
# print(np.polyfit(z_arr[25:55], offdiag_arr[25:55,0], 1))
# heat = sns.heatmap(offdiag_arr)
# plt.show()
```

```
(130, 1000, 4, 2, 2)
(130, 1000)
```

Out[267]:

```
Text(0.5,1,'Ln of absolute value of the off-diagonal component of one of the
modes')
```



In [ ]: