SCHRODINGER EQUATION TO NUMERICAL ODE

DR. MATTHEW HOGAN

We start with the Schrodinger equation,

$$\frac{\hbar}{i}\frac{\partial}{\partial t}\psi(r,t) = \hat{H}\psi(r,t). \tag{1}$$

We assume that our generated neutrinos experience no electro-weak potential and no gravitational potential. Furthermore, we will solve the Schrodinger equation in the neutrino's rest frame. For numerical integration, we assign all physical constants to one (1). This means that the partial derivatives become regular derivatives and we can drop the position r dependence. We now have the following equation

$$\frac{1}{i}\frac{d}{dt}\psi(t) = \hat{H}\psi(t). \tag{2}$$

We assume that there exists neutrinos of definite mass and we will solve for their dynamics in this basis called the mass-basis. Each neutrino has a distinct mass in this basis. Thus the Hamiltonian $\hat{H}$ acting on $\psi(t)$ results in the eigenvalue equation

$$\hat{H}\psi(t) = E\psi(t), \tag{3}$$

where $E = \sqrt{p^2 + m^2}$ is the energy of the neutrino, p is its momentum, and m is its mass. Assuming that neutrinos are nearly massless and an experiment can create neutrinos of measured energy E_ν, then the energy is approximately

$$E \approx E_\nu + \frac{m^2}{2E_\nu}, \tag{4}$$

where we have assumed that $p \approx E_\nu$ for near zero mass or highly-relativistic particle. The Schrodinger equation is now given by

$$\frac{1}{i}\frac{d}{dt}\psi(t) = \left(E_\nu + \frac{m^2}{2E_\nu}\right)\psi(t). \tag{5}$$

We now drop the E_ν term at this point because the term cancels out when calculating the oscillation probability in the exact solution $\psi(t) \sim \exp(iEt)$. Said another way, the oscillation probability is calculated by taking the inner product of two wave-functions that share a common phase term $\sim \exp(iE_\nu t)$ of which cancel.

Since neutrinos are known to interact with matter via the flavor basis (e, μ, and τ), we must switch to that basis. We assume there is a unitary mixing matrix U that converts neutrinos from the mass basis to the flavor basis. Let us multiply the Schrodinger equation by U on both sides.

$$U\frac{1}{i}\frac{d}{dt}\psi(t) = \frac{m^2}{2E_\nu}U\psi(t). \tag{6}$$

If we let ψ represent a vector of mass eigenstates, then $m^2 \to M$ is now a diagonal matrix of the neutrino masses. The equation now reads

$$U\frac{1}{i}\frac{d}{dt}\psi(t) = \frac{1}{2E_\nu}MU\psi(t). \tag{7}$$

If we multiply both sides of the equation by $U^{-1} = U^{*T}$ (transpose-conjugate), we now get

$$\frac{1}{i}\frac{d}{dt}\psi(t) = \frac{1}{2E_\nu}U^{*T}MU\psi(t). \tag{8}$$

For simplicity, let $\Delta = U^{*T}MU$. To solve the following ODE, divide ψ and multiply by i on both sides of the equation to get

$$\frac{1}{\psi(t)}\frac{d}{dt}\psi(t) = \frac{d}{dt}\log\psi(t) = \frac{i}{2E_\nu}\Delta. \tag{9}$$

The standard ODE technique is to integrate both sides between $t = 0$ and $t = t'$ (and drop the prime for simplicity), then exponentiate both sides. The solution reads

$$\psi(t) = \exp\left(\frac{i\Delta}{2E_\nu}t\right)\psi(t=0) \tag{10}$$

The exponential of a matrix is

$$\exp(A) = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + ..., \tag{11}$$

where I is the identity matrix and A is any square-matrix. We can go a step farther and say the simplified version of this equation is

$$\psi(t) = G\psi(t=0). \tag{12}$$

where G is the time propagation operator matrix.

For a simple two-flavor mixing example, we would get the following oscillatory behavior as shown in the figure below.

The code to generate this plot is included below also.

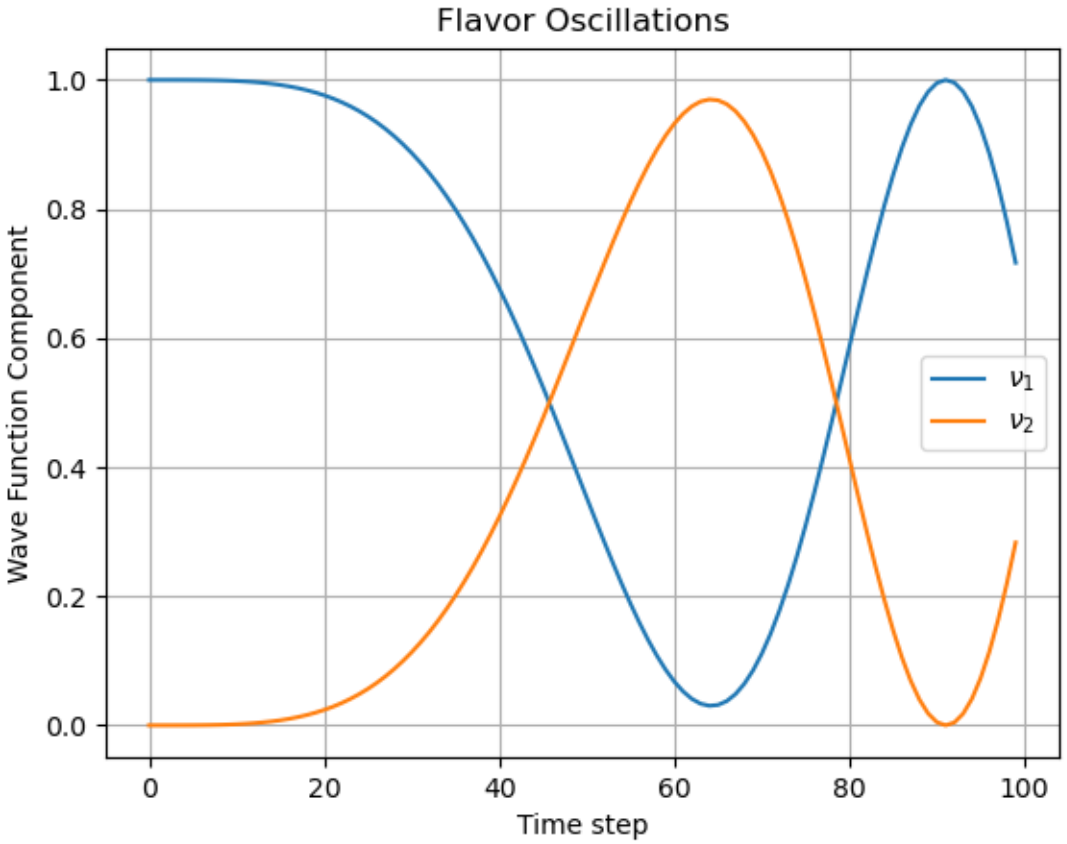

FIGURE 1. default

```
from scipy.linalg import expm
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

matplotlib.use("TkAgg")  # depends on OS


class MassStates(object):
    class Nu1(object):
        mass: float = 1.0
```

```
    class Nu2(object):
        mass: float = 2.0

    mass_mat: np.ndarray = np.array([[Nu1.mass, 0], [0, Nu2.mass]])
    mass2_mat: np.ndarray = mass_mat ** 2


class FlavorStates(object):

    mixing_angle: float = np.deg2rad(40.0)
    mixing_mat: np.ndarray = np.array(
        [
            [np.cos(mixing_angle), -np.sin(mixing_angle)],
            [np.sin(mixing_angle), np.cos(mixing_angle)],
        ]
    )


def time_propagator_mat(energy_nu: float, time_step: float) -> np.ndarray:
    """The time propagator matrix"""
    delta_mat: np.ndarray
    mixing_mat_conj_trans: np.ndarray
    delta_mat: np.ndarray
    exp_arg: np.ndarray
    exp_mat: np.nddary

    mixing_mat_conj_trans = np.conjugate(FlavorStates.mixing_mat).T
    delta_mat = np.matmul(mixing_mat_conj_trans, MassStates.mass2_mat)
    delta_mat = np.matmul(delta_mat, FlavorStates.mixing_mat)
    exp_arg = (0 + 1j) / (2.0 * energy_nu) * time_step * delta_mat
    exp_mat = expm(exp_arg)
    return exp_mat


time_steps: int = 100
time_step_size: float = 0.01
energy_nu: float = 10.0

initial_state_vector: np.ndarray = np.array([1, 0]).T
# Store all the state vector values
```

```
state_vectors: np.ndarray = np.zeros(shape=(time_steps, 2),
                                     dtype=complex)
state_vectors[0, :] = initial_state_vector.T

for step_index in range(1, time_steps):
    previous_step_index: int
    previous_state_vector: np.ndarray
    time_step: float
    time_propagator: np.ndarray
    new_state_vector: np.ndarray

    previous_step_index = step_index - 1
    previous_state_vector = state_vectors[previous_step_index, :]
    time_step = step_index * time_step_size
    time_propagator = time_propagator_mat(energy_nu, time_step)
    new_state_vector = np.matmul(time_propagator,
                                 previous_state_vector.T)
    state_vectors[step_index, :] = new_state_vector.T

# The fraction of each eigenstate in the wave-function
# is the absolute value squared
state_fractions: np.ndarray = np.abs(state_vectors ** 2)

fig: plt.Figure
ax: plt.Axes
fig, ax = plt.subplots()
ax.grid()
ax.plot(range(time_steps),
        state_fractions[:, 0],
        label=r"$\nu_1$")
ax.plot(range(time_steps),
        state_fractions[:, 1],
        label=r"$\nu_2$")
ax.set_title("Flavor Oscillations")
ax.set_ylabel("Wave Function Component")
ax.set_xlabel("Time step")
ax.legend()
fig.show()
```