Normal modes

1 Assignment Description

These assignment sheets describe the modelling of strings like guitar and piano strings. They provide a brief description of the topic and set problems some of which must be solved using python programs.

The assignment consists in writing an essay that contains an answer to each question and which expand on the material included in these notes. The references are there to provide you with extra source of information and they are not exhaustive.

The essay must be readable on its own without reference to the assignment sheets. The essay must not necessarily follow the same order as the notes and does not need to answer the questions in the same order either. As a matter of fact using a different structure and order is a bonus. Do describe the interpretation or consequences of the results that you obtain. We also expect you to focus more on some aspects of the material presented in these notes. This could be a more detailed discussion of the derivation of the equations or algorithm described in the notes or to better explain the interpretation of the results. You can also solve problems which are not set and which answer some further questions that you might have.

Some part of the assignment sheet will need to be included in your essay, but do not copy these section verbatim, use your own words. The equations do not need to be changed, but you can provide more detailed explanations or derivations when appropriate.

You have to submit 3 files: 1 pdf file for the essay and 3 python programs. We advise you to typeset your essay in LaTeX but this is not compulsory.

The essay must have a maximum of 12 pages (using as a reference the provided LaTeX style file).

2 Introduction

In these notes we model the vibration of strings showing the difference between thin strings, like guitar strings, and thick string, like piano strings. We start by modelling the longitudinal and then transversal vibration of masses connected by string and then show that this is a good approximation to model strings. We also model strings using partial differential equations.

3 Masses and springs

We start by considering a simple example of N nodes of mass m_i connected by springs of elastic coefficient k. The end spheres are both connected to fixed walls. The masses are at the position X_i while the walls are located at the fixed positions X_0 and X_{N+1}.

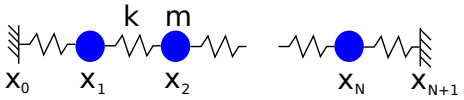

Figure 1: N masses connected by springs.

If the elongation at rest of each spring is d (the length of each spring when not connected to any mass), then the force acting on each mass i located at X_i is given by

$$F_i = k(X_{i+1} - X_i - d) - k(X_i - X_{i-1} - d) = k(X_{i+1} + X_{i-1} - 2X_i) \tag{1}$$

and the equation for these masses is given by

$$m_i \frac{d^2 X_i}{dt^2} = k(X_{i+1} + X_{i-1} - 2X_i) \qquad i = 1, \ldots, N. \tag{2}$$

When the system is at rest, $dX_i/dt = d^2X_i/dt^2 = 0$, the distances between the masses will be equal to

$$l_0 = \frac{X_{N+1} - X_0}{N+1}. \tag{3}$$

Notice that in general l_0 will be different from d. We can then perform the change of variable

$$X_i = X_0 + il_0 + y_i \tag{4}$$

where the y_i describe the displacement of the masses with respect to their static configuration. Substituting (4) into (2) we obtain

$$m_i \frac{d^2 y_i}{dt^2} = k(y_{i+1} + y_{i-1} - 2y_i) \qquad i = 1, \ldots, N \tag{5}$$

where $y_0 = y_{N+1} = 0$.

Equation (5) can be written in matrix form as

$$\ddot{\mathbf{y}} = k\mathbf{M}^{-1}\mathbf{A}\mathbf{y} \tag{6}$$

where

$$\mathbf{M} = \text{diag}(m_i), \qquad \mathbf{A} = \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & . & . & . & . & . \\ 1 & -2 & 1 & 0 & 0 & . & . & . & . & . \\ 0 & 1 & -2 & 1 & 0 & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & 0 & 1 & -2 & 1 & 0 \\ . & . & . & . & . & 0 & 0 & 1 & -2 & 1 \\ . & . & . & . & . & 0 & 0 & 0 & 1 & -2 \end{pmatrix}. \tag{7}$$

To solve this system of equations we seek very special solutions of the form

$$\mathbf{y}_\mu = \mathbf{v}_\mu \sin(\omega_\mu t), \tag{8}$$

where μ, or other Greek indices, labels the solution, not the vector components, and after inserting (8) into (6) we have to solve the algebraic equation

$$-\omega_\mu^2 \mathbf{v}_\mu = k\mathbf{M}^{-1}\mathbf{A}\mathbf{v}_\mu. \tag{9}$$

This is an eigen value problem which can easily be solved. As A is an N by N matrix, there are exactly N solutions.

The solutions of type (8) are called the normal mode solutions of (5) and notice that they do correspond to solutions where all the masses oscillate at the same frequency. The ω_μ are called the angular frequencies, while the frequencies of oscillations are given by $\nu_\mu = \omega_\mu/2\pi$.

Mathematically, as (9) is linear, the norm of the eigen vector $\mathbf{v}_\mu$ are arbitrary (they can be multiplied by any number) and they correspond to the amplitude of motion of the masses. (In reality, if the amplitude is too large, equation (2) stops being valid, the springs would be permanently damaged for example). When one computes the normal modes, it is common and convenient to chose the amplitudes so that

$$(\mathbf{v}_\mu.\mathbf{v}_\mu) = |\mathbf{v}_\mu|^2 = 1. \tag{10}$$

Notice also that $\mathbf{A}$ is symmetric: $\mathbf{A} = \mathbf{A}^t$. If all the m_i are identical, then M is a multiple of the identity matrix and $\mathbf{C} = k\mathbf{M}^{-1}\mathbf{A}$ is also symmetric and, as a result, the eigen vector $\mathbf{v}$ are orthogonal to each other. Indeed,

$$\mathbf{v}_\rho^t \mathbf{C} \mathbf{v}_\mu = -\omega_\mu^2 \mathbf{v}_\rho^t \mathbf{v}_\mu, \tag{11}$$

and as $\mathbf{C}$ is symmetric,

$$\begin{aligned}\mathbf{v}_\rho^t \mathbf{C}\mathbf{v}_\mu &= (\mathbf{v}_\rho^t \mathbf{C}\mathbf{v}_\mu)^t \\ &= \mathbf{v}_\mu^t \mathbf{C}\mathbf{v}_\rho \\ &= -\omega_i^2 \mathbf{v}_\mu^t \mathbf{v}_\rho \\ &= -\omega_j^2 \mathbf{v}_\rho^t \mathbf{v}_\mu. \end{aligned} \tag{12}$$

So $(\omega_\rho - \omega_\mu)(\mathbf{v}_\rho.\mathbf{v}_\mu) = 0$ and if $\omega_\rho \neq \omega_\mu$, then $(\mathbf{v}_\rho.\mathbf{v}_\mu) = 0$. Sometimes, a system can have multiple normal frequencies, i.e. we can have 2 eigen vectors with the same frequency. In that case, we have a plane of solutions with the same frequency and we can always pick 2 vectors that are perpendicular to each other.

If all the normal vector $\mathbf{v}_j$ satisfy (10) they form an orthonormal base of $\mathbb{R}^N$.

Another property of (6) is that any linear combination of its solutions is also a solution. We can thus generate a very large class of solutions by taking the most general linear combination of the normal modes:

$$\mathbf{Y}(t) = \sum_{\mu=0}^{N-1} a_\mu \sin(\omega_\mu t + \phi_\mu)\mathbf{v}_\mu \tag{13}$$

where a_μ are constants. One can prove that all solutions of (6) can actually be written as (13) and so the special solutions we have computed allow us to compute the most general solutions.

Equation (6) is normally solved as an initial value problem, meaning that given $Y(t=0)$ and $\frac{dY}{dt}(t=0)$ the solution is uniquely determined.

Then, using the orthogonality of the eigen vectors, we see that

$$\begin{aligned}(\mathbf{v}_\rho.\mathbf{Y}) &= \sum_{\mu=0}^{N-1} a_\mu \sin(\omega_\mu t + \phi_\mu)(\mathbf{v}_\rho.\mathbf{v}_\mu) \\ &= a_\rho \sin(\omega_\rho t + \phi_\rho) \end{aligned} \tag{14}$$

and

$$\begin{aligned}(\mathbf{v}_\rho.\dot{\mathbf{Y}}) &= \sum_{\mu=0}^{N-1} a_\mu \omega_\mu \cos(\omega_\mu t + \phi_\mu)(\mathbf{v}_\rho.\mathbf{v}_\mu) \\ &= a_\rho \omega_\rho \cos(\omega_\rho t + \phi_\rho). \end{aligned} \tag{15}$$

Evaluating (14) and (15) at $t = 0$ gives us a system of $2N$ equations with $2N$ unknowns, a_ρ and ϕ_ρ, which has a unique solution.

Coding task 1:

`eigen_val.py`:

Write a python program, called `eigen_val.py`, to compute the normal frequencies and the normal modes of equation (5) when all the masses are equal, $m_i = m$. You actually need to solve equation (9). The function `linalg.eig` from the `numpy` library take a square matrix as its argument and computes its eigen values and eigen vectors. The function returns a pair of array `Ev,V` where `Ev` is the list of eigen values and `V` a square array where column l is the eigen vector corresponding to the eigen value `Ev[l]`, or, more explicitly, the eigen vector $\mathbf{v}_\mu$ is given by `V[:,mu-1]` and the corresponding eigen value λ_μ by `Ev[mu-1]` (we label the eigen vectors starting at $\mu = 1$ but python starts at 0).

The eigen values returned by`linalg.eig` are not ordered, but the following code

```
(Ev , V) = np.linalg.eig(C)
idx = Ev.argsort()[::-1]
Ev = Ev[idx]
V = V[:,idx]
```

computes the eigen values λ_μ and eigen vectors $\mathbf{v}_\mu$ of $\mathbf{C}$, *i.e.* satisfying $\mathbf{C}\,\mathbf{v}_\mu = \lambda_\mu \mathbf{v}_\mu$, (where in our case $\lambda_\mu = -\omega_\mu^2$).

The code above saves the eigen values in `Ev` in decreasing order and orders the corresponding eigen vectors in `V`. Notice also that the eigen vectors, `V`, returned by `numpy` are normalised to 1.

Use two variables to store the values of k and m at the beginning of your program. The program must be written to work for any values of these parameters but take $k = 1$, $m = 1$ for the requested output. Use a system of $N = 201$ masses.

The program must then compute all the frequencies $\nu_\mu = \sqrt{-\lambda_\mu}/2\pi$ as well as the ratio ν_μ/ν_0 where ν_0 is the lowest frequency. This is to check if the frequencies are multiple of each others.

The program must generate the following output.

- a figure $\mathbf{v}_\mu$, as a function of the node index i. (Each $\mathbf{v}_\mu$ is a vector of size n whose components are labelled by the index i). The figures must show the amplitude of displacement, $\mathbf{v}_\mu$,of the 7 lowest frequencies (7 lowest smallest μ), on the same figure but in different colours. Notice that by (8), $\mathbf{v}_\mu[i] = \mathbf{y}_\mu(t = 1/4\nu_\mu)[i]$, corresponding to the largest displacements of the mass at $X_i = X_0 + il_0$.
- a second figure (once the first one is closed) showing the following two functions: 1) ν_μ/ν_0 (in blue) as a function of the eigen value index μ, where ν_0 is the lowest normal frequency. 2) the identity function μ (in red) (both starting with $\mu = 1$).
- The program must also output on the screen the difference between the computed frequencies, ν_μ, and the integer multiple of the lowest frequencies, $\mu\nu_0$, expressed in cent, for the lowest 20 frequencies. (See section 5.4 in [4] or section 6.2 below for the definition of a cent).

3.1 Transverse vibrations

We will now consider the same system as above but where the masses move transversally in the y direction while their x coordinate remains unchanged. Assuming that the fixed horizontal distance between nodes is $l_0 = x_{i+1} - x_i$ for all i and that the relative transverse displacement, $y_{i+1} - y_i$, are much smaller than l_0, the potential energy is then given by

$$V = \frac{k}{2}\sum_{i=0}^{N}\left((l_0^2 + (y_{i+1} - y_i)^2)^{1/2} - d\right)^2 \tag{16}$$

$$\approx \sum_{i=0}^{N}\left(v_0 + \frac{\kappa}{2}(y_{i+1} - y_i)^2\right) \tag{17}$$

and the equations for y_i are then

$$m_i\frac{\partial^2 y_i}{\partial t^2} = -\frac{\partial V}{\partial y_i} = \kappa(y_{i+1} + y_{i-1} - 2y_i) \tag{18}$$

with $y_0 = y_{N+1} = 0$.

Question 1:

Derive the expression for the potential V, (16), and its approximation as a quadratic function (17) where one must assume that $y_{i+1} - y_i << l_0$ and $y_{i+1} - y_i << l_0 - d$. Determine the expression for κ and v_0. Detail the computation of $\frac{\partial V}{\partial y_i}$ in (18).

We note that equation (18) is identical to equation (5) except for the coefficient in front of the spatial derivatives. This means that, if the springs are stretched, the longitudinal and transversal displacements of the masses obey the same equation but with different coefficients.

Question 2:

> What is the physical condition that the string must be subjected to before it can vibrate (hint: look at equation (18) after having determined κ).

If the number of masses is very large, this system can be used to approximately model a guitar string.

4 Guitar string

If the number of nodes is very large, we can take the continuum limit of eq (5) or (18) to transform it into a partial differential equation. To do this, we assume that the end points are fixed, $X_0 = 0$, $X_{N+1} = L$ so that $l_0 = L/(N+1)$ and so l_0 decreases as N increases. We also define $x_i = il_0$ as the rest position of the nodes. If we assume that the displacements $y_i = y(x_i)$ do not vary too much between nodes, we can use the following approximation:

$$\begin{aligned} y(x_i + l_0) &= y(x_i) + l_0 \frac{\partial y}{\partial x} + \frac{l_0^2}{2}\frac{\partial^2 y}{\partial x^2} + \frac{l_0^3}{6}\frac{\partial^3 y}{\partial x^3} + o(l_0^4) \\ y(x_i - l_0) &= y(x_i) - l_0 \frac{\partial y}{\partial x} + \frac{l_0^2}{2}\frac{\partial^2 y}{\partial x^2} - \frac{l_0^3}{6}\frac{\partial^3 y}{\partial x^3} + o(l_0^4) \end{aligned} \tag{19}$$

Substituting (19) into (18) and assuming all the mass identical, $m_i = m$, we obtain

$$\frac{\partial^2 y}{\partial t^2} = \frac{\kappa l_0^2}{m}\frac{\partial^2 y}{\partial x^2} + o(l_0^4). \tag{20}$$

which we rewrite as

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2} \tag{21}$$

where

$$c = l_0\sqrt{\frac{\kappa}{m}}. \tag{22}$$

Question 3:

> Solve equation (21) using the method of separation of variables, proceeding as follows. Write $y = g(t)f(x)$ and substitute this into (21). Compute the most general solution for $g(t)$ and $f(x)$ when imposing the condition that $f(0)$ and $f(L) = 0$. Determine all the values that the angular frequency ω_n can take. What is $f_n(x)$ for each of these ω_n? Write down the normal mode frequencies $\nu_n = \omega_n/2\pi$.

Like the normal modes of the discrete equation (5), the normal modes of (21) are orthogonal to each other. The proof is the same as for equation (5) except that to compute the scalar product between 2 functions $f_i(x)$ and $f_j(x)$, we must compute the integral

$$< f_i . f_j >= \int_0^L f_i(x) f_j(x) dx. \tag{23}$$

We then use the fact that $\frac{d^2 f_i(x)}{dx^2} = -\omega_i f_i(x)$, and we show that

$$\int_0^L f_i(x)\frac{d^2 f_j(x)}{dx^2}dx = \int_0^L \frac{d^2 f_i(x)}{dx^2} f_j(x) dx \tag{24}$$

by integrating by part twice and by using the boundary condition imposing that at all the $f_i(x)$ vanish at $x = 0$ and $x = L$.

We have shown above that the continuum limit of eq (18) is the so called wave equation (21). The opposite is true as well: equation (18) can be seen as a discretised version of

the wave equation (21). As we have seen, the wave equation is easy to solve when c is a constant, but when it depends on x, the problem is much harder to solve.

When this is the case, one can solve the equation numerically by noticing that eq (18) is a system of ordinary differential equations. In this project we will restrict ourself to constant c or k, but the programs we will write would be very easy to modify to consider more general and more complex cases. We will now consider a string with both ends fixed, $y = 0$, at $x = 0$ and $x = L$, and approximate it as N masses connected by springs. We can then find approximate solutions of (21) by solving (18) with

$$m_i = 1, \qquad \kappa = \frac{c^2}{l_0^2}, \qquad l_0 = \frac{L}{N-1}. \tag{25}$$

Equation (21) is what is called an initial value problem. One can prove that to solve it uniquely, one needs to know the initial profile of the string, $y(x, t = 0)$, and its initial speed, $\dot{y}(x, t = 0)$. When a guitar string is pinched, it is stretched into an approximately triangular shape (which we will consider more later). The triangular shape gives us $y(x, t = 0)$ and as the string is simply released $\dot{y}(x, t = 0) = 0$ everywhere. A piano string on the other hand is hit by a hammer. In that case, $y(x, t = 0) = 0$ everywhere and $\dot{y}(x, t = 0)$ is a bell shape function around the point of impact of the hammer.

Question 4:

Convert the system of N second order differential equations (18) into a system of $2N$ first order differential equations.

Coding task 2:

`OdeString.py`:

Write a computer program that solves the system of ODE (18) numerically using the module `ode_rk4.py` (this extended version of the module used for the electron problem is described in section 6.3.) We will do this to model a guitar or harpsichord string and thus use the parameters (25) and take $c = 329.6m/s$, $L = 0.63m$.

The code you will write is similar to the one you have already written for the electron problem except that the number of equation is not fixed but depends on the number of points we decide to consider.

To write the code, edit the provided file `OdeString.py` and proceed as follows:

- Notice that the `__init__` function already sets some useful parameters: the class variable `dx` is nothing but l_0 and `dt` is also set correctly.
- Complete the function `reset(self,t,y,doty)` to set the initial condition. `y` is the initial value for y_i and `doty` the initial value for $\frac{dy_i}{dt}$. The class variable `self.v` must be an array of `2N` elements. The first `N` values of `self.v` must be the values of y_0 to y_{N-1} and the last `N` values must be $\frac{dy_0}{dt}$ to $\frac{dy_{N-1}}{dt}$. Use numpy arrays of type `dtype='float64'` for better accuracy.
- Complete the function `f(self,t,v)` describing the equation using the answer from question 4. Instead of using loops to compute the right hand side of the equation, use the indexing functionality of the `numpy` library described in section 6.4.2 below. Notice also that the end points do not move: $d^2y_0/dt^2 = d^2y_{n-1}/dt^2 = 0$. So the values `F` for these 2 points must be set to `0`.
- Below the class definition, set the initial values of L and c.
- Create an object of type `StringOde`.
- Create the initial value, at $t = 0$,for y and $\dot{y}$, taking $y(x) = \sin(\pi x/L)$ and $dy/dt(x) = 0$, where $x \in [0, L]$. Which of the normal mode computed in question 3 does this correspond to?

  Notice that the class variable `dx` is the distance between nodes which should be used to computes the `x` coordinates, ranging from 0 to L, of the lattice points.

- Call the class function `reset()` to copy the initial conditions into the `StringOde` object.
- Call the class function `solve_until()` with the parameters `tmax` and `fig_dt` already initialised in the program.
- Add some code to generate on the same figure the profile $y(x)$ at $t = 0$ (in black), $t = T/10$ (in red), $t = 2T/10$ (in green),$t = 3T/10$ (in blue), $t = 4T/10$ (in magenta) and $t = 5T/10$ (in cyan) where `T` is the period set in the template program.
- Add some code to plot the function $y(L/3, t)$ as a function of time, to be displayed as a second figure when the first one has been closed. t must range from 0 to $20T$. Hint, use the plot function of the `OdeRk4` class.
- Add some code to compute the Fourier transform of the vibrating string proceeding as follows:
  - Add a function called `FFT_mod(self,x)` to the class `StringOde`. It must compute the modulus of Fourier transform ($\sqrt{a_i^2 + b_i^2}$) of the time function $y(x, t)$ for the given x, using the values stored in the class `OdeRk4` variable `l_f`. As $y(x, t)$ is real we only want to return the values corresponding to the positive frequencies (see the FFT project). The number of data points does not need to be a power of 2. The function must return the result, $\sqrt{a_i^2 + b_i^2}$ in increasing value of i, as a numpy array.
  - Add a function called `plot_modfft(self,modFFT,freq_max)` which plots the positive defined array `modFFT`, which will be an average values of modulus of Fourier transform, as a function of the frequency (which you can compute using `np.fft.fftfreq`). Only plot the frequencies up to `freq_max` and use the function `loglog` to plot a logarithmic plot of both argument.
  - At the end of `OdeString` add some code calling `FFT_mod` 5 times with the argument `x=L/13`, `x=L/17`, `x=L/23`, `x=L/31` and `x=L/43`, compute the average value of the arrays returned by `FFT_mod` and use `plot_modfft` to plot that average. Set `freq_max` to 400Hz. Notice that if we compute the Fourier Transform only at one point, we will miss all the normal modes which have a node at that point. This is why we use a few points whose distance from the origin are not multiple of each other.

Expected code output:

- A graphic of the profile $y(x)$ at the 6 times specified above and in the specified colours.
- A graphic, as a separate figure, of $y(L/3, t)$ as a function of time from $t = 0$ up to $t = 20\,T$ where $T = 2\,L/c$.
- A logarithmic plot of the average modulus of the Fourier Transform of the vibrating string averaged over the 5 position on the string described above.

Question 5:

Use the program `OdeString.py` to integrate equation (21) using the parameters and the initial conditions specified in coding task 2.

a) Generate the profile $y(x)$ at the time specified in coding task 2.

b) Generate the figure of $f(L/3)$ as a function of t for the interval $t \in [0, 20\,T]$.

c) Generate the logarithmic plot of the averaged modulus of the Fourier Transform of the vibrating string.

d) The Fourier Transform for a normal mode should be zero for all frequencies except the normal mode frequency. Why is the spectrum spread out?

Coding task 3:

`eigen_val_guitar.py` :

Make a copy your program `eigen_val.py`, calling it `eigen_val_guitar.py`, and modify it so that it computes the spectrum of a guitar string by solving the eigen value problem resulting from eq (21). For this you must set the mass $m = 1$ and add a new variables `L` and `c` for the length and rigidity of the string. Take $L = 0.63m$ and $c = 329.6m/s$, compute `dx=L/(N+1)` and set $\kappa = c^2/dx^2$.

The program must generate the same 2 figures as in coding task 1. It must also output on the screen

- The lowest 20 frequencies of the string.
- The difference between the computed frequencies, ν_j, and the integer multiple of the lowest frequencies, $j\nu_0$, expressed in cent, for the lowest 100 frequencies.

Question 6:

- As equation (5) is an approximation of (21) we can expect the normal frequencies of (9) to match the normal frequencies obtained in question 3, but there are differences between the two and, for a given frequency, that difference decreases when the number of points `N` increases. What is the minimum number of points, using only multiple of 100, that you must use so that the lowest 100 frequencies are within 10 cents of an integer multiple of the lowest frequency? A cent is defined and explained in section 6.2.
- Which value do you get for the lowest frequency of the string. Use the same number of points as the one found in the previous question. How does that value compare with the lowest frequency found in question (3) ?

Question 7:

Use `eigen_val_guitar.py` to generate the 7 lowest normal mode profiles using the parameters specified in that coding task. On a second figure plot the frequencies of the normal mode, as a function of the normal mode index, for the discretised string, computed using for `N` the value found in question 6, (in blue) as well as the expected spectrum for a guitar string in (red).

The initial condition we have used in coding task 2 is simple mathematically, but as a matter of fact very hard to excite on a guitar. When a guitar string is plucked, it is pulled by the finger of the player to form a triangular shape and then it is released suddenly. Here is how to simulate this excitation of the guitar string:

Coding task 4:

Make a copy your program `OdeString.py` and call it `OdeStringPlucked.py`.

- Set the initial value for y so that it correspond to the triangular shape taken by a guitar string when it is plucked, using

$$y(x) = \begin{cases} \frac{3}{L}x & 0 \le x \le \frac{L}{3} \\ \frac{3}{2}(1 - x/L) & \frac{L}{3} \le x \le L. \end{cases} \tag{26}$$

  Keep $\dot{y}(x) = 0$.
- Set `N` to the number of points found in question 6
- Modify the function so that it plots the profiles at the times `10*T`, `10*T+2*T/10`, ... `10*T+5*T/10`.

Question 8:

Use the program `OdeStringPlucked.py` to create figures of snapshots of the profile of the guitar string at the same times set in coding task 4, as well as the function $y(x = L/3, t)$ corresponding to the vibration of the string at $x = 3/L$.

- Using $N = 900$ as well as c and L as above, generate the 2 figures set in coding task 4: a) the profile $y(x)$ at $t = 0, T/10, 2T/10, 3T/10, 4T/10, 5T/10$. b) $y(L/3, t)$ from $t - 0$ until $T = 20T$ where $T = 2L/c$. Generate the same 2 figures with $N = 100$.
- What are the main differences between the evolution of the plucked profile compared to the evolution of the normal mode? (Look at the solution for $N = 900$ which is a very good approximation of the solutions of eq (21).)
- The analytical solution of equation (5) with the plucked initial condition can be easily computed using the solutions of this equation described in sec 3.2 of [4] and one expects the solution to be fully periodic in t. For small N the solution is not periodic any more. Can you explain why using the results of question 6? (The time integration of equation (5) is very accurate and can't be blamed for the results obtained here.)

5 Piano String

The equation for a piano string is slightly different from a guitar string. The difference comes from the fact that the piano string is thicker and made out of metal. One thus has to take into account the energy that one needs to bend it. The end result is the following equation[3]

$$\frac{\partial^2 f}{\partial t^2} = c_1^2 \frac{\partial^2 f}{\partial x^2} - c_2^2 \frac{\partial^4 f}{\partial x^4}. \tag{27}$$

Does the extra term proportional to c_2^2 affect the spectrum of the string? To answer that question we can approximate the equation above by a finite difference equation, and compute the spectrum by solving the corresponding eigen value problem.

Question 9:

Derive the expression for the finite order approximation $\Delta_i^4 f$ of $d^4 f/dx^4$ using the 5 points $x_i, x_{i\pm1}, x_{i\pm2}$. Start by computing the Taylor expansion of $f(x \pm dx)$ and $f(x \pm 2dx)$ up to the 5th order. Combine the obtained expression to express $d^4 f/dx^4(x_i)$ as a linear combination of f_i, $f_{i\pm1}$ and $f_{i\pm2}$.

Replacing $d^2 f/dx^2$ by $\Delta_i^2 f$, given by (37), and $d^4 f/dx^4$ by $\Delta_i^4 f$ in (27) we obtain

$$\frac{\partial^2 f}{\partial t^2} = c_1^2 \Delta_i^2 f - c_2^2 \Delta_i^4 f. \tag{28}$$

The equation can then be written in matrix form as

$$\frac{\partial^2 \mathbf{f}}{\partial t^2} = \frac{c_1^2}{dx^2}(A_2 + B_2)\mathbf{f} - \frac{c_2^2}{dx^4}(A_4 + B_4)\mathbf{f} \tag{29}$$

where A_2 and A_4 are the matrix representation of the finite difference expression of respectively $d^2 f/dx^2$ and $d^4 f/dx^4$ while B_2 and B_4 are boundary terms. Notice also that dx is the lattice spacing. The two ends of the piano strings are both fixed, which means that $f_0 = f_{N+3} = 0$. One must also impose the condition that $d^2 f/dx^2 = 0$ at the end of the string. This is called an hinged boundary condition. In finite difference terms this means that $f_0 - 2f_1 + f_2 = 0$ and this is encoded in the 2 boundary matrices B_2 and B_4 bellow. A_2 is identical to (7). Notice that the finite difference equation (28) does not make sense for f_0, f_1, f_{N+2} and f_{N+3}, but these functions are all determined by the boundary conditions.

Question 10:

Write down the expression for the matrices A_4. (You can write it for $N = 11$, *i.e.* as a 7×7 matrix)

The matrices B_2 and B_4 then take the following form:

$$\mathbf{B}_2 = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & . & . & . & . & . \\ 0 & 0 & 0 & 0 & 0 & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & 0 & 0 & 0 & 0 & 0 \\ . & . & . & . & . & 0 & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix} \tag{30}$$

$$\mathbf{B}_4 = \begin{pmatrix} -2 & 0 & 0 & 0 & 0 & . & . & . & . & . \\ 2 & -1 & 0 & 0 & 0 & . & . & . & . & . \\ 0 & 0 & 0 & 0 & 0 & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & 0 & 0 & 0 & 0 & 0 \\ . & . & . & . & . & 0 & 0 & 0 & -1 & 2 \\ . & . & . & . & . & 0 & 0 & 0 & 0 & -2 \end{pmatrix} \tag{31}$$

(32)

Coding task 5:

Make a copy your program `eigen_val.py`, calling it `eigen_val_piano.py`, and modify it so that it computes the spectrum of a piano string by solving the eigen value problem resulting from eq (29). At the bottom of `eigen_val_piano.py` add some code to plot the inharmonicity (in cent) of the first 100 normal modes of the piano string as a function of their frequency ν. (Use the `pyplot` function `semilogx`). Use for N the value determined in question 6. Be careful to compute dx correctly using L and N (it is different from the guitar case).

In [2], the values of c_1 and c_2 are given for a few piano string and for the note C4 we have $c_1 = 329.6m/s$, $c_2 = 1.25m^2/s$ and the length of the string is $0.63m$.

The program must generate the same 2 figures as in coding task 1. It must also

- Generate a figure of the inharmonicity of the piano string normal modes (in cent), for the first 100 normal modes, as a function of their frequency.
- Print on the screen the smallest frequency which is more than 100 cents of out tune as well as its frequency index (the lowest frequency has index 1).

Question 11:

Use the number of nodes obtained in question 6 to ensure that the values of the lowest 100 frequencies of equation 9 are within 10 cents of the corresponding frequencies of equation 21.

a) Using the values for c_1 and c_2 above, compute the frequencies, ν, of the normal modes of the C4 piano string. Then plot on the same figure the 100 lowest frequencies of the string, in blue, as well as the 100 first multiples of the lowest frequencies in red as a function of their index number.

b) Plot the inharmonicity of the string expressed in cent as a function of the frequency, ν, of each node and use a logarithmic scale for the frequency $\nu = \omega/2\pi$.

c) What is the frequency of the normal mode which is detuned by 1 semi tone and what is it index number (the lowest mode being counted as number 1).

Question 12:

> The result of question 11 might be surprising and some experimental values of the normal frequencies of piano strings can be found in [3].
>
> So is the piano string inharmonic? Would one expect a piano string to sound out of tune and why does it not?

To answer this question, you can copy the program `OdeStringPlucked.py` into a file called `OdeStringHit.py` and change in it the initial condition by setting $y(x, t = 0) = 0$ and $dy/dt(x, t = 0) = exp(-((x - L/3)/0.05)^2)$. This corresponds roughly to a string being hit by a piano hammer.

You can then use the program `OdeStringHit.py` to create a figure of the modulus of the Fourier coefficients, averaged over a few points, as a function of the frequency, setting `freq_max` to `20000`Hz. This corresponds to a guitar string that is hit like a piano string. While the equation is not exactly the same as the piano string equation, the relative amplitude of the excited mode will be a good estimate for a real piano string.

6 Some Useful Material

6.1 Finite differences

To derive a finite difference expression to approximate a differential operator, one must use the Taylor series. There are several way to do this and there are also several possible answers, but if one tries to use the smallest amount of adjacent points, then the answer is unique modulo some symmetry. For any derivative of order n we need at least $n + 1$ points.

We assume that we have the sampled values f_i of a function $f(x)$ where $f_i = f(x_i)$ where $x_i = idx$, for some dx and $i \in [0, N]$ are integers. For the first derivative we need 2 points, so to approximate $df/dx(x_i)$ we should use f_i and f_{i+1}. The we use the fact that

$$f_{i+1} = f(x_i + dx) = f(x_i) + dx\frac{df}{dx}(x_i) + O(dx^2) = f_i + dx\frac{df}{dx}(x_i) + O(dx^2) \tag{33}$$

and so

$$\frac{df}{dx}(x_i) = \frac{f_{i+1} - f_i}{dx} + O(dx) \tag{34}$$

and we can write

$$\frac{df}{dx}(x_i) \approx \frac{f_{i+1} - f_i}{dx}. \tag{35}$$

Notice that we could also use f_{i-1} instead of f_{i-1}.

For the second order derivative, we must take 3 points: x_{i-1}, x_i and x_{i+1}. We then write

$$\begin{aligned} f_{i+1} &= f(x_i + dx) = f(x_i) + dx\frac{df}{dx}(x_i) + \frac{dx^2}{2}\frac{d^2f}{dx^2}(x_i) + \frac{dx^3}{6}\frac{d^3f}{dx^3}(x_i) + O(dx^4) \\ f_{i-1} &= f(x_i - dx) = f(x_i) - dx\frac{df}{dx}(x_i) + \frac{dx^2}{2}\frac{d^2f}{dx^2}(x_i) - \frac{dx^3}{6}\frac{d^3f}{dx^3}(x_i) + O(dx^4). \end{aligned} \tag{36}$$

Adding these 2 equations together, moving f_i on the other side of the equation and dividing by dx we obtain

$$\Delta_i^2 = \frac{d^2f}{dx^2}(x_i) = \frac{f_{i+1} + f_{i-1} - 2f_i}{dx^2} + O(dx^2). \tag{37}$$

6.2 Musical Scale and Cent

The human hear likes sound which are periodic. Periodic sounds are made out of a superposition of sine waves with frequencies which are all multiple of the lowest frequency. Such sound are call harmonic. Sounds which are not harmonic are called inharmonic. Most

musical instruments produce harmonic sounds, the main exception being percussion instruments. The detuning between 2 notes can be described by their frequency ratio or, like musicians, using cents. The piano keyboard is split in section called octaves. An octave difference corresponds to a doubling of frequency. An octave is then split in 12 semi tones. If ν_0 is the frequencies of the lowest notes, the other notes in the octave will have the frequencies $\nu_i = \nu_0\, 2^{i/12}$. Notice that note 12 has the frequency $\nu_{12} = 2\nu_0$. The semitone is then divided in 100 intervals called cents. Their frequencies, within a given semitone, is given by

$$\nu_i = \nu_1\, 2^{i/1200}. \tag{38}$$

A well trained hear can detect a frequency difference of about 5 cents and most adult can easily spot a difference of about 25 cents.

To compute the cent difference between 2 frequencies ν_1 and ν_2 one can invert eq (38) and find

$$i \;=\; 1200\,\frac{\log(\frac{\nu_1}{\nu_2})}{\log 2}. \tag{39}$$

So, for example, the freuqency difference between 450Hz and 440Hz is $1200\,\frac{\log(\frac{450}{440})}{\log 2} = 38.9$ cents.

6.3 Module ode_RK4.py

The python module `ode_RK4.py` is identical to the one used for the electron project, except that a couple of member functions have been added:

- `solve_until(self,tmax,fig_dt)`: this function calls the function `RK4_1step` for as long as `t<tmax`. It thus integrates the equation until the time `t_max`. Moreover at regular intervals it saves the function values in the class variable `l_f` which becomes a list of arrays containing the equation function values.

  The corresponding times are saved in the function variable `l_t` which is also a list. So `l_t[i]` is the time corresponding to the profile `l_f[i]`.

  The functions values are saved every multiple of `fig_dt`. If `fig_dt` is an integer multiple of `dt`, then `l_t[i]` will be equal to `i*fig_dt`, if not it will be the time nearest to it.

  In the string program `l_f[i][0:N]` will be an array with all the values of $y(x)$ at the time `l_t[i]`, while `l_f[i][N:]` will be an array containing all the values of $\dot{y}(x)$.

- `plot(self,i,j,format="k-")`: plots the function `v[j]` as a function of `v[i]` where `j` and `i` counts the fields starting from `1` not `0`. This is because when `i` or `j` are zero they refer to the time. `format` sets the plot format for the figure. `plot` uses the data saved in the class variables `l_t` and `l_f`.

  For example if `myode` is an object of type `ode_RK4.py`, `myode.plot(0,1)` plots the function `v[0]` as a function of time and `myode.plot(2,5,"r-")` plots `v[4]` as a function of `v[1]`.

  Notice that `plot()` does not call the `plt.show()` function. This means that one can call `plot()` several times to plot multiple functions on the same figure. It also means that `plt.show()` must be called after one or more calls of `plot()` to display the result.

6.4 Computing with numpy

6.4.1 Evaluating functions for arrays

Using numpy, one can evaluate functions for a range of values quite simply. For example, we we want to evaluate the function $\exp(\cos(x^2) + 3 * sin(4x))$ for 200 values of x ranging from 0 to 2π all we need to do is

```
1 import numpy as np
2 x = np.linspace(0,np.pi,200)
3 f = np.exp(np.cos(x*x)+3*np.sin(4*x))
```

6.4.2 Copying sub arrays

A common tasks when manipulating arrays is to copy one sub array into another one. One can be tempted to do this using loops, but this can be quite slow. Instead, `numpy` has indexing features which makes such manipulations quite fast and easy to write.

In the example below `A` and `B` are 6 component vectors and we compute the quantity `B[i] = A[i]+3*A[i-1]-A[i+1]` for the index values for which this makes sense: `i=1` to `4`:

```
1 import numpy as np
2 A = np.array([0,1,2,3,4,5])
3 B = np.zeros(6)
4 n = 6
5 B[1:n-1] = A[1:n-1]+3*A[0:n-2]-A[2:n]
6 print("A=",A)
7 print("B=",B)
```

Notice that each element sub-range correspond to exactly 4 elements: `[1:n-1]` corresponds to the range `1,2,3,4`, `[0:n-2]` to the range `0,1,2,3` and `[2:]` to the range `2,3,4,5`.

In the example, below, we replace a *square* of values inside `A` by the values of `B`: `A[1,2]=B[0,0]`, `A[2,2]=B[1,0]`, `A[1,3]=B[0,1]`, `A[2,3]=B[1,1]`.

```
1 import numpy as np
2 A = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
3 B = np.array([[10,20],[30,40]])
4 print(A)
5 print(B)
6 A[1:3,2:]=B = B
7 print(A)
```

6 References

[1] T.D. Pollard, W.C. Earnshaw, J. Lippincott-Schwartz, *Cell Biology*, ISBN : 9781416022558

[2] J. Bensa, S. Bilbao, R. Kronland-Martinet, J.O. Smith *The simulation of piano string vibration: From physical models to finite difference schemes and digital waveguides.* J. Acoust. Soc. Am. 114 (2003)

[3] H. Fletcher, E. Donell Blackham and R. Stratton *Quality of Piano Tones* J. Acoust. Soc. Am. 34 (1962)

[4] Dave Benson *Music: a Mathematical Offering*. Freely downloadable from https://homepages.abdn.ac.uk/mth192/pages/html/maths-music.html

7 To submit:

- One pdf file for the essay, including the figures for questions 5, 7, 8, 11 and 12. Ensure all your figures have axis labels which are not tiny. Give all your figures a caption describing their content and refer to them in the text by number.
- Python code 2: `OdeString.py`
- Python code 3: `eigen_val_guitar.py`
- Python code 5: `eigen_val_piano.py`