Application of the Crank-Nicolson Finite Difference Numerical Method to the 1-Dimensional Acoustic Wave Equation

Emily Williams

AE 370 Project 2

Contents

1 Introduction

The wave equation serves as a timeless model used in various simulations for different engineering applications. Sound waves can be modeled using the acoustic wave equation. Piano tuners also utilize acoustic waves to match frequencies of certain keys to keep pianos in tune. Oftentimes, pianos and other instruments are tuned using A440 (the A key just above middle C) as a reference [7]. This study will look into why the A note is used to tune instruments rather than middle C or any other note and whether there is an underlying logical reason for this choice.

2 Solution Approach

2.1 Adapted Variant

The 1D wave equation takes the following form for this study:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} + g(x,t), \qquad 0 \leq t \leq T, \quad a \leq x \leq b \tag{1}$$

where time t spans from 0 to final time T, position x spans from a to b, $g(x,t)$ is the appropriate source term dependent on the exact solution, and c is the speed of sound. For this study, c is assumed to be about 343 meters per second, the speed of sound at around room temperature.

2.2 Method of Manufactured Solutions

To generate an exact solution in order to test the correctness of approximation methods used, the method of manufactured solutions is implemented using a very well-chosen solution to the wave equation as prescribed:

$$u(x,t) = u_m \sin(\omega t - kx) \tag{2}$$

where u_m is the amplitude of the wave, ω is the angular frequency, and k is the wave number [3]. For the case of a piano, u_m is assumed to reach a maximum of 50 decibels. To solve for the source term $g(x,t)$, the generated exact solution can be plugged into equation

(1) using the following derivatives:

$$\frac{\partial u}{\partial t} = u_m \omega \cos(\omega t - kx) \tag{3}$$

$$\frac{\partial^2 u}{\partial t^2} = -u_m \omega^2 \sin(\omega t - kx) \tag{4}$$

$$\frac{\partial u}{\partial x} = -u_m k \cos(\omega t - kx) \tag{5}$$

$$\frac{\partial^2 u}{\partial x^2} = -u_m k^2 \sin(\omega t - kx) \tag{6}$$

Plugging the above second derivatives into equation (1):

$$-u_m \omega^2 \sin(\omega t - kx) = -u_m c^2 k^2 \sin(\omega t - kx) + g(x,t) \tag{7}$$

$$g(x,t) = u_m (c^2 k^2 - \omega^2) \sin(\omega t - kx) \tag{8}$$

And thus the source term $g(x,t)$ is obtained and the complete form of the wave equation can be rewritten:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} + u_m (c^2 k^2 - \omega^2) \sin(\omega t - kx) \tag{9}$$

where $t \in [0, T]$ and $x \in [a, b]$.

2.3 Initial Boundary Value Problem

Initial boundary value problems are partial differential equations that depend on time and space [1]. Using the generated exact solution for the wave equation, initial conditions and boundary conditions are inherently prescribed:

$$u(x, t = 0) = \eta(x) = u_m \sin(-kx) \tag{10}$$

$$u(x = a, t) = g_a(t) = u_m \sin(\omega t - ka) \tag{11}$$

$$u(x = b, t) = g_b(t) = u_m \sin(\omega t - kb) \tag{12}$$

To solve the wave equation, a finite difference method can be implemented by using local interpolation in a procedure known as method of lines [1]. Local spectral methods can also be used as seen in finite element solutions. Finite difference methods are analogous to interpolation of functions, whereas spectral methods are analogous to least-squares approximation of functions. For modeling the wave equation, either method can be used. For this study, the finite difference method is considered because only the 1D wave equation is being analyzed. Finite element methods are especially useful for irregular domain regions because the domain can be partitioned into any simple subregion in 2D or 3D [5]. When solving initial boundary value problems (IBVPs), methods for solving time-dependent problems must be combined with methods for solving space-dependent problems. However, to get a resulting initial value problem (IVP) that can be solved using a numerical method, the IBVP must first be discretized in space using the method of lines. Breaking up the continuous space variable into a finite number of pieces will result in a discretization of the spatial domain:

$$x_j = a + \frac{(b-a)(j-1)}{n}, \qquad j = 1, \dots, n+1 \tag{13}$$

Now, the approximated infinite-dimensional solution $u(x,t)$ must be found by using local interpolation and approximating the spatial dependence of u in terms of a finite number of locally defined basis functions in space. Approximating this solution in space can be achieved by using a centered representation in terms of Lagrange polynomials. Using a second-order polynomial, $u(x,t)$ can be approximated over the interval $x_{j-1} \leq x \leq x_{j+1}$:

$$u(x,t) \approx \sum_{i=j-1}^{i=j+1} b_i(t) L_i^{(j)}(x) \tag{14}$$

where the $L_i^{(j)}(x)$ represents the Lagrange basis polynomials and the $b_i(t)$ represent the unknown coefficients in the expansion that can be simplified by looking at any instance in time:

$$u(x_j,t) \approx \sum_{i=j-1}^{i=j+1} b_i(t) L_i^{(j)}(x_j) \tag{15}$$

$$= b_j(t) \tag{16}$$

thus letting $b_j(t)$ be an approximation of $u(x_j,t)$. So, for finite difference methods:

$$u(x,t) \approx \sum_{i=j-1}^{i=j+1} u_i(t) L_i^{(j)}(x) \tag{17}$$

where $u_i(t) \approx u(x_i,t)$. So, computing the various coefficients $b_i(t)$ is equivalent to calculating the values $u_i(t)$ that approximate that exact solution u at the grid point x_i at some instance in time. With replacing the continuous spatial variable x by the $n+1$ points $\{x_1, \dots, x_{n+1}\}$, the spatial dependence of the function u is restricted to be a linear combination of the Lagrange polynomials. The result of this spatial discretization is to create an initial value problem. Plugging the above approximation into the 1D wave equation:

$$\sum_{i=j-1}^{j+1} \ddot{u}_i(t) L_i^{(j)}(x_j) = c^2 \sum_{i=j-1}^{j+1} u_i(t) \left.\frac{d^2 L_i^{(j)}}{dx^2}\right|_{x=x_j} + g(x_j,t) \qquad (j = 2, \dots, n) \tag{18}$$

$$\ddot{u}_j(t) = c^2 \sum_{i=j-1}^{j+1} u_i(t) \left.\frac{d^2 L_i^{(j)}}{dx^2}\right|_{x=x_j} + g(x_j,t) \qquad (j = 2, \dots, n) \tag{19}$$

Simplifying the summation term on the right-hand side:

$$\begin{aligned}\sum_{i=j-1}^{j+1} u_i(t) \frac{d^2 L_i^{(j)}}{dx^2}\bigg|_{x=x_j} = \quad & u_{j-1}(t)\frac{d^2}{dx^2}\left(\frac{1}{2\Delta x^2}(x-x_j)(x-x_{j+1})\right) + \\ & u_j(t)\frac{d^2}{dx^2}\left(-\frac{1}{\Delta x^2}(x-x_{j-1})(x-x_{j+1})\right) + \\ & u_{j+1}(t)\frac{d^2}{dx^2}\left(\frac{1}{2\Delta x^2}(x-x_{j-1})(x-x_j)\right)\end{aligned} \tag{20}$$

for:

$$L_{j-1}^{(j)} = \frac{(x-x_j)(x-x_{j+1})}{(x_{j-1}-x_j)(x_{j-1}-x_{j+1})} = \frac{1}{2\Delta x^2}(x-x_j)(x-x_{j+1}) \tag{21}$$

$$L_{j}^{(j)} = \frac{(x-x_{j-1})(x-x_{j+1})}{(x_j-x_{j-1})(x_j-x_{j+1})} = -\frac{1}{\Delta x^2}(x-x_{j-1})(x-x_{j+1}) \tag{22}$$

$$L_{j+1}^{(j)} = \frac{(x-x_{j-1})(x-x_j)}{(x_{j+1}-x_j)(x_{j+1}-x_{j-1})} = \frac{1}{2\Delta x^2}(x-x_{j-1})(x-x_j) \tag{23}$$

where $\Delta x = x_j - x_{j-1} = x_{j+1} - x_j$. Upon simplifying:

$$\sum_{i=j-1}^{j+1} u_i(t) \frac{d^2 L_i^{(j)}}{dx^2}\bigg|_{x=x_j} = \frac{1}{\Delta x^2}[u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)] \tag{24}$$

Plugging the above expression back into the new initial value problem to get the final form:

$$\ddot{u}_j(t) = \frac{c^2}{\Delta x^2}\Big[u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)\Big] + g(x_j, t), \quad j = 2, \ldots, n \tag{25}$$

In matrix form $\ddot{\boldsymbol{u}} = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{g}$:

$$\begin{bmatrix} \ddot{u}_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \frac{c^2}{\Delta x^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}\begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} g(x_2, t) + \frac{c^2 g_a(t)}{\Delta x^2} \\ g(x_3, t) \\ \vdots \\ g(x_{n-1}, t) \\ g(x_n, t) + \frac{c^2 g_b(t)}{\Delta x^2} \end{bmatrix} \tag{26}$$

with the following associated initial condition:

$$\boldsymbol{u}(t=0) = \begin{bmatrix} \eta(x_2) \\ \eta(x_3) \\ \vdots \\ \eta(x_{n-1}) \\ \eta(x_n) \end{bmatrix} \tag{27}$$

And thus the resulting IVP is of the form $\ddot{\boldsymbol{u}} = \boldsymbol{f}(\boldsymbol{u}, t)$ with $\boldsymbol{f}(\boldsymbol{u}, t) = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{g}(t)$.

2.4 State-Space Form

In order to solve the resulting IVP, a state-space representation is used by relating a system of first-order differential equations [4]. Setting some vector $\boldsymbol{v} = \dot{\boldsymbol{u}}$, the system becomes:

$$\dot{\boldsymbol{u}} = \boldsymbol{v} \tag{28}$$

$$\dot{\boldsymbol{v}} = \boldsymbol{f}(\boldsymbol{u}, t) = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{g} \tag{29}$$

Using state-space representation:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} \tag{30}$$

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \dot{\boldsymbol{u}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{t}) \end{bmatrix} = \tilde{\boldsymbol{f}}(\boldsymbol{z}, t) \tag{31}$$

thus resulting in a first-order ordinary differential equation where $\boldsymbol{f}(\boldsymbol{u}, t) = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{g}(t)$ with the following new system:

$$\dot{\boldsymbol{z}} = \boldsymbol{B}\boldsymbol{z} + \boldsymbol{p} \tag{32}$$

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{t}) \end{bmatrix} = \begin{bmatrix} 0 & \boldsymbol{I} \\ \boldsymbol{A} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{g} \end{bmatrix} \tag{33}$$

with the associated initial condition:

$$\boldsymbol{z}(t = 0) = \begin{bmatrix} \boldsymbol{u}(t = 0) \\ \dot{\boldsymbol{u}}(t = 0) \end{bmatrix} \tag{34}$$

$$\boldsymbol{z}(t = 0) = \begin{bmatrix} \boldsymbol{u}(t = 0) \\ \boldsymbol{u}_t(t = 0) \end{bmatrix} \tag{35}$$

With the first-order form in hand, the solution can be stepped through and approximated using nearly any numerical method of choice.

3 Crank-Nicolson Method

The Crank-Nicolson method is based on the trapezoidal rule, giving second-order convergence in time. The algorithm for Crank-Nicolson is a combination of the forward Euler method at some k and the backward Euler method at some $k+1$ [2]. Algorithms for the forward Euler method and backward Euler method are known:

$$z_{k+1} = z_k + \Delta t \tilde{\boldsymbol{f}}(z_k, t_k) \tag{36}$$

$$z_{k+1} = z_k + \Delta t \tilde{\boldsymbol{f}}(z_{k+1}, t_{k+1}) \tag{37}$$

3.1 Derivation

The derivation of the Crank-Nicolson method begins by integrating the IVP from t_k to t_{k+1}:

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{z}} dt = \int_{t_k}^{t_{k+1}} \tilde{\boldsymbol{f}}(\boldsymbol{z}(t), t) dt \tag{38}$$

Evaluating the left-hand side directly and using the Lagrange basis to approximate the right-hand side as a line over the interval $t \in [t_k, t_{k+1}]$:

$$\tilde{\boldsymbol{f}}(\boldsymbol{z}(t), t) \approx \tilde{\boldsymbol{f}}(\boldsymbol{z}(t_k), t_k) \left(\frac{t - t_{k+1}}{t_k - t_{k+1}} \right) + \tilde{\boldsymbol{f}}(\boldsymbol{z}(t_{k+1}), t_{k+1}) \left(\frac{t - t_k}{t_{k+1} - t_k} \right) \tag{39}$$

Substituting and letting $\Delta t = t_{k+1} - t_k$:

$$\boldsymbol{z}(t_{k+1}) - \boldsymbol{z}(t_k) \approx \int_{t_k}^{t_{k+1}} \tilde{\boldsymbol{f}}(\boldsymbol{z}(t_k), t_k) \left(\frac{t - t_{k+1}}{-\Delta t} \right) + \tilde{\boldsymbol{f}}(\boldsymbol{z}(t_{k+1}), t_{k+1}) \left(\frac{t - t_k}{\Delta t} \right) dt \tag{40}$$

Evaluating the integrals on the right-hand side:

$$\boldsymbol{z}(t_{k+1}) - \boldsymbol{z}(t_k) \approx \frac{\tilde{\boldsymbol{f}}(\boldsymbol{z}(t_k), t_k)}{-\Delta t} \int_{t_k}^{t_{k+1}} (t - t_{k+1}) dt + \frac{\tilde{\boldsymbol{f}}(\boldsymbol{z}(t_{k+1}), t_{k+1})}{\Delta t} \int_{t_k}^{t_{k+1}} (t - t_k) dt \tag{41}$$

$$\boldsymbol{z}(t_{k+1}) - \boldsymbol{z}(t_k) \approx \frac{\Delta t}{2} (\tilde{\boldsymbol{f}}(\boldsymbol{z}(t_k), t_k) + \tilde{\boldsymbol{f}}(\boldsymbol{z}(t_{k+1}), t_{k+1})) \tag{42}$$

This suggests the following method:

$$\boldsymbol{z}_{k+1} - \boldsymbol{z}_k = \frac{\Delta t}{2} (\tilde{\boldsymbol{f}}(\boldsymbol{z}_k, t_k) + \tilde{\boldsymbol{f}}(\boldsymbol{z}_{k+1}, t_{k+1})) \tag{43}$$

thus resulting in the algorithm for the Crank-Nicolson method. In order to implement the implicit method in `MATLAB` or similar software, the solution must be algebraically manipulated due to the unknown $\boldsymbol{z}_{k+1}$ also being in the right-hand side:

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \frac{\Delta t}{2} (\tilde{\boldsymbol{f}}(\boldsymbol{z}_k, t_k) + \tilde{\boldsymbol{f}}(\boldsymbol{z}_{k+1}, t_{k+1})) \tag{44}$$

Substituting $\tilde{\boldsymbol{f}}(\boldsymbol{z}, t) = \boldsymbol{B}\boldsymbol{z} + \boldsymbol{p}$:

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \frac{\Delta t}{2} (\boldsymbol{B}\boldsymbol{z}_k + \boldsymbol{p}_k + \boldsymbol{B}\boldsymbol{z}_{k+1} + \boldsymbol{p}_{k+1}) \tag{45}$$

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \frac{\Delta t}{2} \boldsymbol{B}\boldsymbol{z}_k + \frac{\Delta t}{2} \boldsymbol{p}_k + \frac{\Delta t}{2} \boldsymbol{B}\boldsymbol{z}_{k+1} + \frac{\Delta t}{2} \boldsymbol{p}_{k+1} \tag{46}$$

$$\boldsymbol{z}_{k+1} - \frac{\Delta t}{2} \boldsymbol{B}\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \frac{\Delta t}{2} \boldsymbol{B}\boldsymbol{z}_k + \frac{\Delta t}{2} (\boldsymbol{p}_k + \boldsymbol{p}_{k+1}) \tag{47}$$

$$\boldsymbol{z}_{k+1} = \left(\boldsymbol{I} - \frac{\Delta t}{2} \boldsymbol{B} \right)^{-1} \left(\boldsymbol{z}_k + \frac{\Delta t}{2} \boldsymbol{B}\boldsymbol{z}_k + \frac{\Delta t}{2} (\boldsymbol{p}_k + \boldsymbol{p}_{k+1}) \right) \tag{48}$$

An analogous process can be followed with the backward Euler method to achieve a form fit for implementation:

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \Delta t \tilde{\boldsymbol{f}}(\boldsymbol{z}_{k+1}, t_{k+1}) \tag{49}$$

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \Delta t(\boldsymbol{B}\boldsymbol{z}_{k+1} + \boldsymbol{p}_{k+1}) \tag{50}$$

$$\boldsymbol{z}_{k+1} - \Delta t \boldsymbol{B}\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \Delta t \boldsymbol{p}_{k+1} \tag{51}$$

$$\boldsymbol{z}_{k+1} = \big(\boldsymbol{I} - \Delta t \boldsymbol{B}\big)^{-1}\big(\boldsymbol{z}_k + \Delta t \boldsymbol{p}_{k+1}\big) \tag{52}$$

Both the Crank-Nicolson method and backward Euler methods are implicit methods. For the 1D wave equation, unlike the heat equation, it is possible to use forward Euler and similar explicit methods to generate an approximate solution. However, there are enticing benefits of choosing implicit methods that will be explored.

3.2 Appeal of Implicit Methods

Methods such as the backward Euler method and the Crank-Nicolson method that involve the unknown $\boldsymbol{z}_{k+1}$ in the right-hand side are called implicit because advancing the approximate solution requires the solution of a nonlinear algebraic system of equations. The additional complexity is compensated for by the significantly better stability properties. Through implementation, it was proven that for time steps that are too large, the forward Euler method provided exceedingly large estimates to $\boldsymbol{z}_{k+1}$. Other explicit methods that provide more promising results, such as fourth-order Runge-Kutta, are explored later. However, implicit methods are primarily used in this study due to greater stability that is noticeable through analyzing regions of absolute stability.

3.3 Stability

Absolute stability regions can be generated easily through consideration of the following general IVP:

$$\dot{\boldsymbol{u}} = \boldsymbol{\Lambda}\boldsymbol{u} \tag{53}$$

$$\boldsymbol{u}(t_0) = \boldsymbol{u}_0 \tag{54}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix. The backward Euler method for this IVP becomes:

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \Delta t \boldsymbol{\Lambda}\boldsymbol{u}_{k+1} \tag{55}$$

$$\boldsymbol{u}_{k+1} = \big(\boldsymbol{I} - \Delta t \boldsymbol{\Lambda}\big)^{-1}\boldsymbol{u}_k \tag{56}$$

$$\boldsymbol{u}_{k+1} = \Big[\big(\boldsymbol{I} - \Delta t \boldsymbol{\Lambda}\big)^{-1}\Big]^{k+1}\boldsymbol{u}_0 \tag{57}$$

For the diagonal matrix $\boldsymbol{\Lambda}$, the j^{th} entry in $\boldsymbol{u}_{k+1}$ can be expressed as:

$$(u_{k+1})_j = \frac{1}{(1 - \Delta t \lambda_j)^{k+1}}(u_0)_j \tag{58}$$

and hence the backward Euler method is absolutely stable when $|1 - \Delta t \lambda_j| > 1$. An analogous process can be followed for finding the absolute stability region of the forward Euler method. Resulting figures (1) and (2) again reiterate the stability benefits of implicit methods.

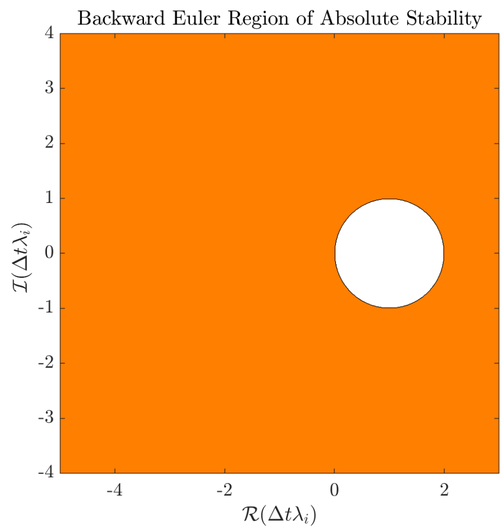

Figure 1: Absolute stability region of backward Euler method

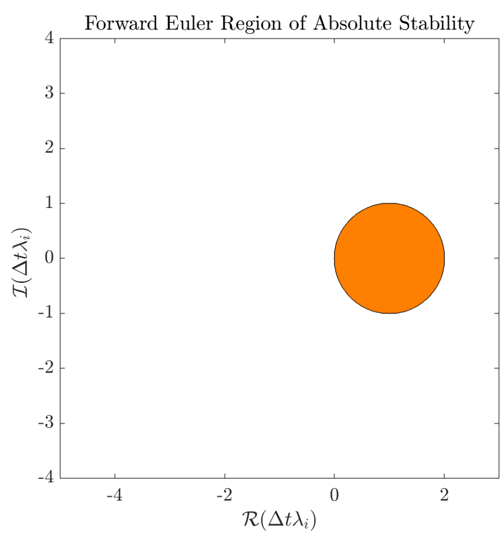

Figure 2: Absolute stability region of forward Euler method

The same procedure can be followed to achieve the stability region for the Crank-Nicolson method:

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2}(\boldsymbol{\Lambda}\boldsymbol{u}_k + \boldsymbol{\Lambda}\boldsymbol{u}_{k+1}) \tag{59}$$

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2}\boldsymbol{\Lambda}\boldsymbol{u}_k + \frac{\Delta t}{2}\boldsymbol{\Lambda}\boldsymbol{u}_{k+1} \tag{60}$$

$$\boldsymbol{u}_{k+1} = \left(\boldsymbol{I} - \frac{\Delta t}{2}\boldsymbol{\Lambda}\right)^{-1}\left(\boldsymbol{I} + \frac{\Delta t}{2}\boldsymbol{\Lambda}\right)\boldsymbol{u}_k \tag{61}$$

$$\boldsymbol{u}_{k+1} = \left[\left(\boldsymbol{I} - \frac{\Delta t}{2}\boldsymbol{\Lambda}\right)^{-1}\left(\boldsymbol{I} + \frac{\Delta t}{2}\boldsymbol{\Lambda}\right)\right]^{k+1}\boldsymbol{u}_0 \tag{62}$$

$$(u_{k+1})_j = \left[\left(1 - \frac{\Delta t}{2}\lambda_j\right)^{-1}\left(1 + \frac{\Delta t}{2}\lambda_j\right)\right]^{k+1}(u_0)_j \tag{63}$$

And thus the Crank-Nicolson method is absolutely stable when $|1 + \frac{\Delta t}{2}\lambda_j| / |1 - \frac{\Delta t}{2}\lambda_j| < 1$:

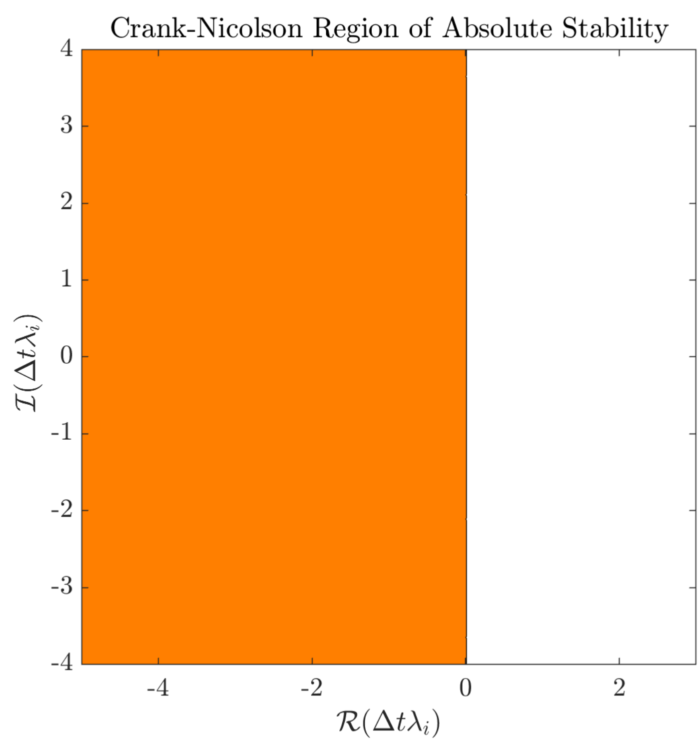

Figure 3: Absolute stability region of Crank-Nicolson method

3.4 Error

There are two sources of error: the truncation error introduced at a single time increment and the cumulative error inherited over several previous time increments. The truncation error and cumulative error together make up the global error. Looking at how the truncation error scales can provide insight into the accuracy of which implicit method between backward Euler and Crank-Nicolson performs better. The truncation error associated with a certain numerical method is the error in applying the method to advance the exact solution by one time increment Δt. For both forward Euler and backward Euler methods, truncation error scales with $O(\Delta t)$ as first-order methods. Looking at the Crank-Nicolson method:

$$\boldsymbol{\tau}_k = \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} - \frac{1}{2}(\boldsymbol{f}(\boldsymbol{u}(t_k), t_k) + \boldsymbol{f}(\boldsymbol{u}(t_{k+1}), t_{k+1})) \tag{64}$$

Using the IVP:

$$\boldsymbol{\tau}_k = \frac{\boldsymbol{u}(t_{k+1}) - \boldsymbol{u}(t_k)}{\Delta t} - \frac{1}{2}(\dot{\boldsymbol{u}}(t_k) + \dot{\boldsymbol{u}}(t_{k+1})) \tag{65}$$

The following Taylor series expansions can be implemented in order to express all quantities evaluated at t_{k+1} in terms of t_k:

$$\boldsymbol{u}(t_{k+1}) = \boldsymbol{u}(t_k) + \Delta t \dot{\boldsymbol{u}}(t_k) + \frac{\Delta t^2}{2}\ddot{\boldsymbol{u}}(t_k) + \frac{\Delta t^3}{6}\dddot{\boldsymbol{u}}(t_k) + H.O.T. \tag{66}$$

$$\dot{\boldsymbol{u}}(t_{k+1}) = \dot{\boldsymbol{u}}(t_k) + \Delta t \ddot{\boldsymbol{u}}(t_k) + \frac{\Delta t^2}{2}\dddot{\boldsymbol{u}}(t_k) + H.O.T. \tag{67}$$

And upon substitution:

$$\boldsymbol{\tau}_k = \frac{\Delta t \dot{\boldsymbol{u}}(t_k) + \frac{\Delta t^2}{2}\ddot{\boldsymbol{u}}(t_k) + \frac{\Delta t^3}{6}\dddot{\boldsymbol{u}}(t_k)}{\Delta t} - \frac{1}{2}\left(2\dot{\boldsymbol{u}}(t_k) + \Delta t \ddot{\boldsymbol{u}}(t_k) + \frac{\Delta t^2}{2}\dddot{\boldsymbol{u}}(t_k)\right) + H.O.T. \tag{68}$$

$$\boldsymbol{\tau}_k = -\frac{\Delta t^2}{6}\dddot{\boldsymbol{u}}(t_k) + H.O.T. \tag{69}$$

$$\boldsymbol{\tau}_k = O(\Delta t^2) \tag{70}$$

And thus proving that the Crank-Nicolson method is indeed a second-order method and, therefore, is more accurate than the Euler methods. For a given Δt, the Crank-Nicolson method will provide a better approximate solution than either of the Euler methods. This notion of accuracy is further demonstrated through implementation.

3.5 Justification

The choice for using the Crank-Nicolson method for approximating the wave equation is based on both stability and accuracy. As shown, implicit methods are proven to be more stable than explicit methods, even with the additional complexity. While the backward Euler method has a larger region of absolute stability than the Crank-Nicolson method, it is shown to be less accurate due to being first-order. While other explicit methods, like fourth-order Runge-Kutta, are more accurate than Crank-Nicolson, these methods often require a much smaller Δt in order to achieve convergence, which is not always desirable for runtime efficiency. Overall, the Crank-Nicolson method is shown to be more accurate than backward Euler, while also having an impressive stability region.

4 Implementation

Approximating the solution to the wave equation further demonstrates that the Crank-Nicolson method exhibits the expected convergence rates in both space and time. Upon verifying correct implementation, comparing results against external data when considering the use of sound waves and frequencies when tuning pianos is considered in the following sections.

The convergence rate in space is equal to the order of the spatial discretization used. Because a second-order polynomial was used to spatially discretize, the spatial error should scale as $O(\Delta x^2)$ for all time stepping methods used. The temporal convergence rate, or the convergence rate in time, is dependent on which method is used for the IVP. This produces the implication that a method that converges differently in space and time could be used, resulting in different convergence rates. This implication is another reason that the Crank-Nicolson method is the primary focus of this study, as it will be demonstrated that the orders of this method in space and time indeed match.

So, it is verified throughout this section that correct implementation is achieved using the solution generated from the method of manufactured solutions, which can remain as the applicable source term with thoughtfully-chosen constants when analyzing the tuning of pianos using small enough values of Δt and Δx due to tuning being only frequency dependent. Again, comparison to this external data is covered in detail in the next section.

4.1 Spatial Convergence

Performing a spatial convergence test demonstrates that the Crank-Nicolson algorithm scales as $O(\Delta x^2)$ by using a fixed small Δt of $\Delta t = 10^{-1}$ for $n = [20, 40, 80, 100]^T$. Using a fixed small value of Δt helps to ensure that the temporal error contribution is small, so the spatial error is dominant. Waterfall plots for the exact and finite difference solutions for the finest value of n show that the solutions appear to agree.

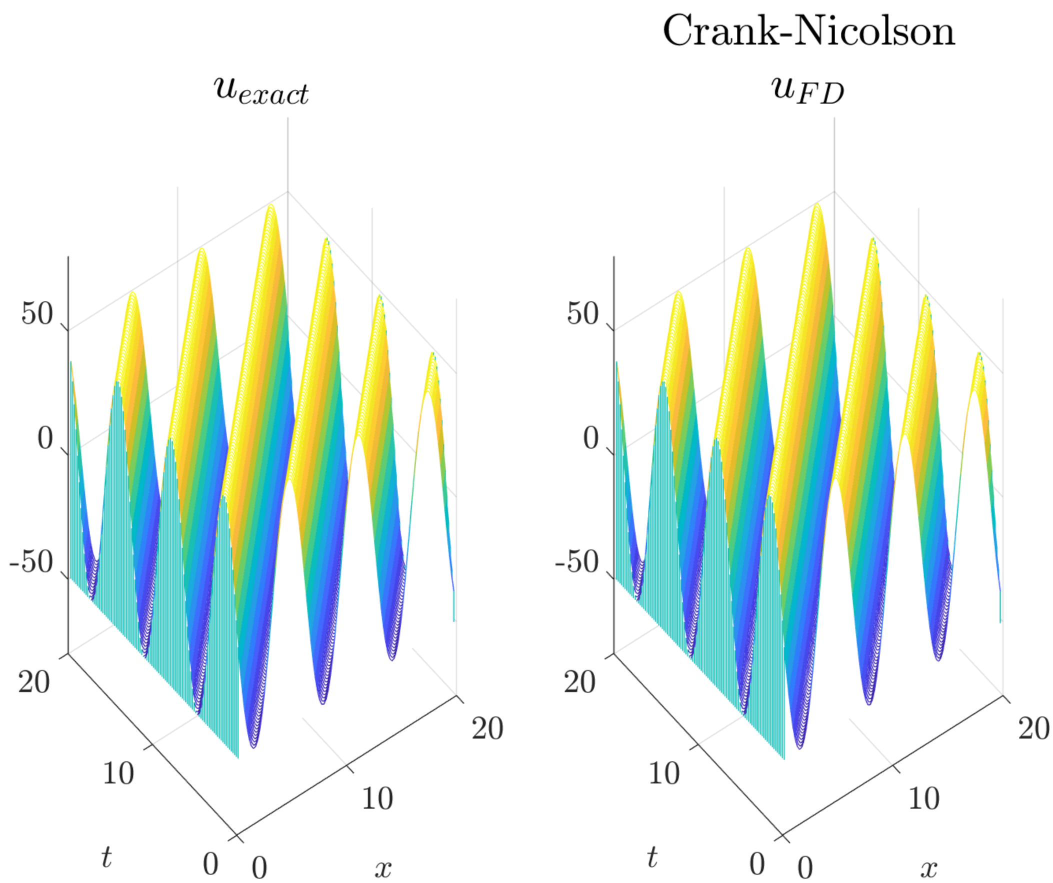

Figure 4: Waterfall plots showing evolution of the solution to the wave equation using $\Delta t = 10^{-1}$ and $n = 100$

Further demonstration of the correctness at $T = 10$ shows that the convergence rate in space is equal to the order of the spatial discretization used (2, in this case). The spatial convergence test in figure (5) shows that both the backward Euler error and the Crank-Nicolson error scale as $O(\Delta x^2)$ since a second-order polynomial was used in the spatial discretization. Note also that the Crank-Nicolson error is again smaller than the backward Euler error, showing that the Crank-Nicolson method is more accurate. While the forward Euler method yields inaccurate results due to its very limited stability, other explicit methods, like the fourth-order Runge-Kutta, again demonstrate the proper scaling for spatial convergence, as shown in figure (6).

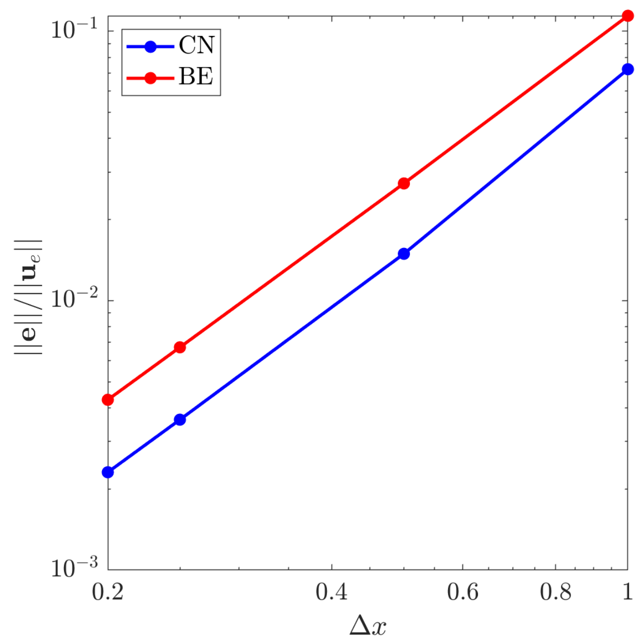

Figure 5: Spatial convergence of the error at $T = 10$ using a fixed $\Delta t = 10^{-1}$

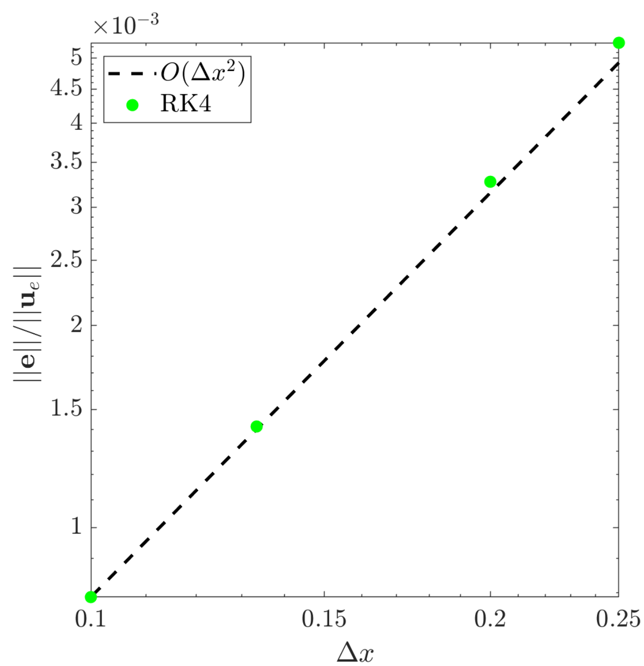

Figure 6: Spatial convergence of the error at $T = 0.25$ using a fixed $\Delta t = 10^{-4}$

4.2 Temporal Convergence

The convergence rate in time is equal to the convergence rate of the time stepping method used for the IVP. For the Crank-Nicolson method, it should scale as $O(\Delta t^2)$, while backward Euler would yield an $O(\Delta t)$ convergence rate, confirmed in figure (7). Figure (8) shows the temporal convergence rate of the Crank-Nicolson method, which scales accurately as $O(\Delta t^2)$.

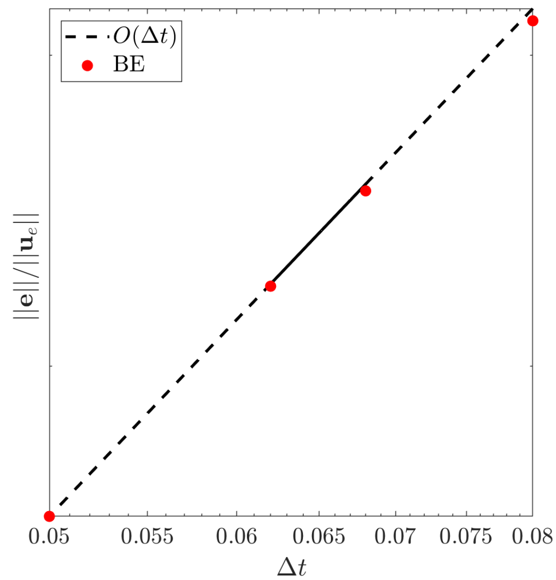

Figure 7: Temporal convergence of backward Euler error at $T = 20$ using a fixed $n = 3000$

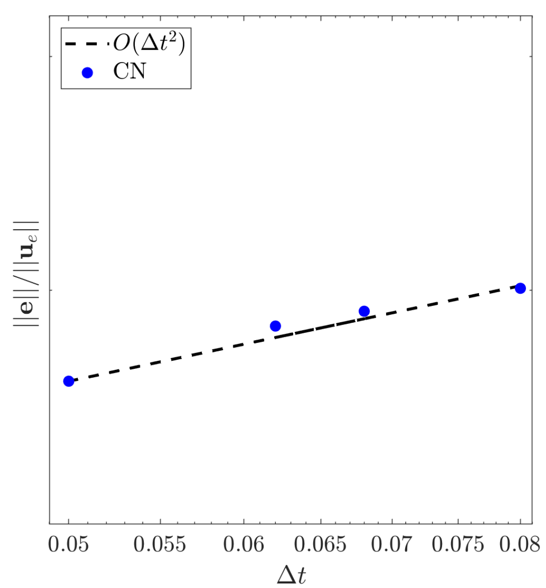

Figure 8: Temporal convergence of Crank-Nicolson error at $T = 20$ using a fixed $n = 3000$

4.3 Animation

The spatial propagation of the solution to the wave equation can also be animated in a digestible form to be able to analyze the fluctuation of the particles. Using the Crank-Nicolson method, it is a straightforward process to plot the approximation as the sound wave propagates. Figure (9) shows this spatial propagation as a scatter plot, demonstrating the change in position of the particle.

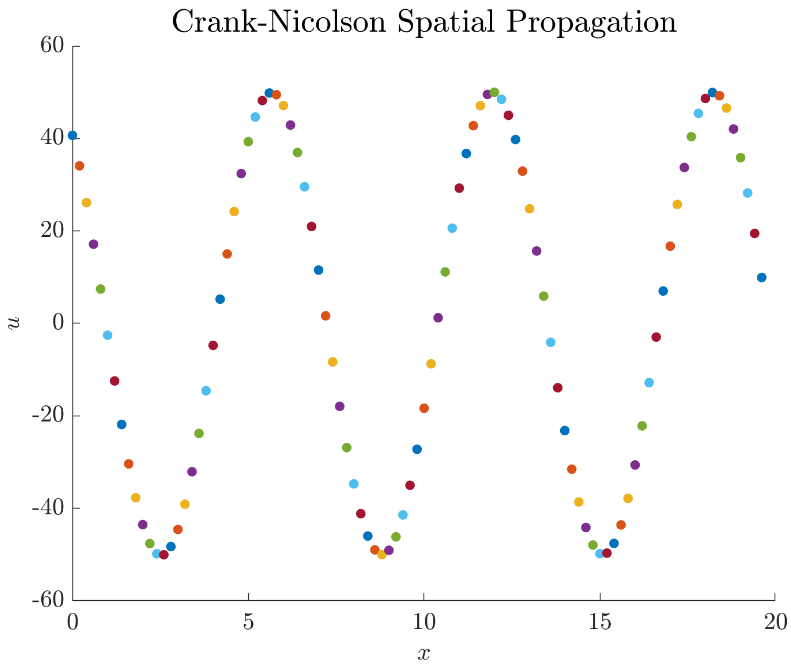

Figure 9: Spatial propagation of Crank-Nicolson approximation

A visual animation is available at the following link: `https://youtu.be/U30_Fubq9EQ`. This video shows the propagation of the particle from a fixed reference frame. Overall, various different methods to test error convergence as well as visual representations of the approximation in action demonstrate that the implementation has been achieved accurately.

Following verification and validation of the successful implementation of the approximation generated by the Crank-Nicolson method, focus can now be directed towards the use of this model to analyze the tuning methods of pianos and other instruments.

Again, with careful consideration of relevant constants and forcing terms that more accurately encompass the solution of the wave equation as applied to acoustic waves, it becomes clear why piano tuners show favoritism towards A440.

5 Piano Tuning

5.1 Keys and Frequencies

So why do piano tuners use A440 as the primary reference? This topic can be explored by looking at the shape of sound waves of certain keys as the wave propagates in space. For example, G_4 (the G key just above middle C, shown in figure (11)) has an audio frequency of 391.9954 Hertz (Hz) and has the following spatial propagation [6]:

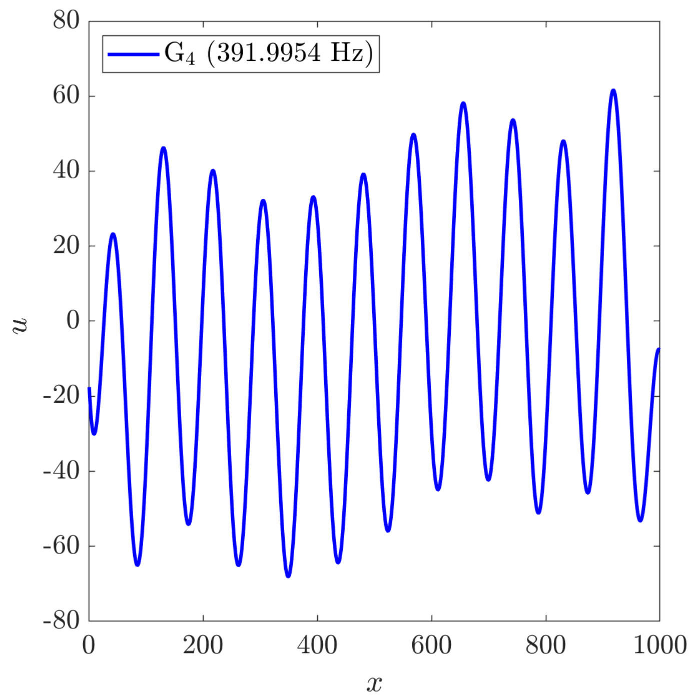

Figure 10: Spatial wave propagation of G_4 for $\Delta x = 10^{-2}$ and $\Delta t = 10^{-1}$

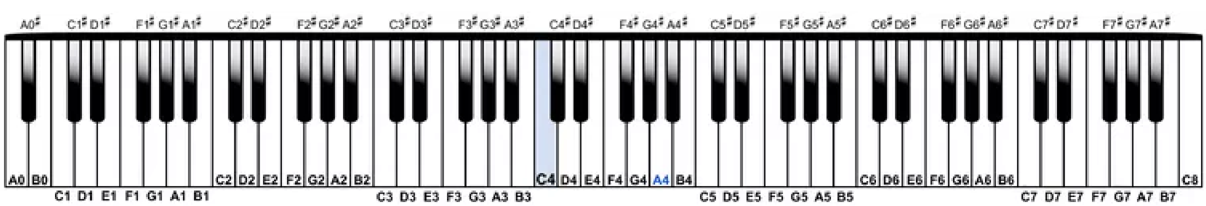

Figure 11: Typical 88-key piano with labeled keys

The same approximation can be applied to different notes, just by changing the frequencies. To acquire the rest of the wave properties needed to plot accurately, the following relations can be used to relate frequency f to angular frequency ω and wave number k:

$$f = 391.9954\text{Hz} \tag{71}$$
$$\omega = 2\pi f \tag{72}$$
$$c = \frac{w}{k} \tag{73}$$

where c is the speed of sound at around 343 meters per second. Wave number k could alternatively be found by using wavelength λ:

$$c = \lambda f \tag{74}$$
$$k = \frac{2\pi}{\lambda} \tag{75}$$

Amplitude u_m can be approximated at about 50 decibels. (When playing *fortissimo*, the piano should peak at about 80 decibels.) And thus all parameters of the wave equation are accurately and thoughtfully determined. Just by changing the frequencies of the musical pitches of different notes, the spatial wave propagation can be approximated. Looking at middle C and then tenor C (C_5 - one octave above middle C), the propagations can be compared:

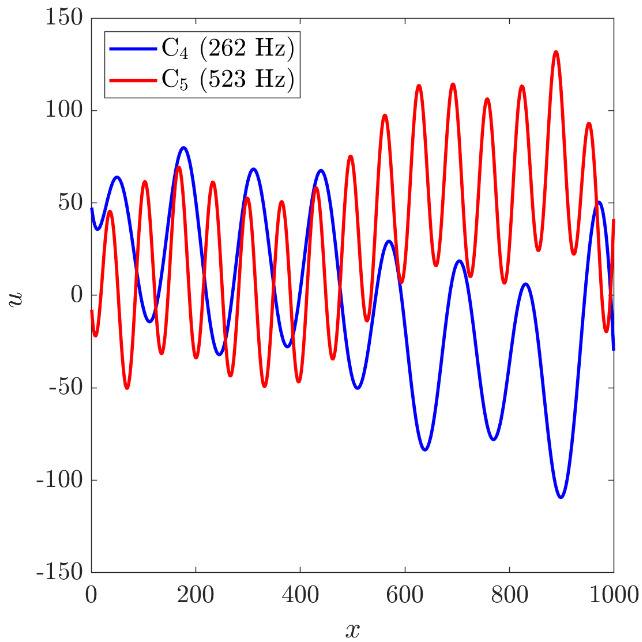

Figure 12: Spatial wave propagation of middle and tenor C for $\Delta x = 10^{-2}$ and $\Delta t = 10^{-1}$

Two completely different notes with different pitches can also be compared, such as F$\sharp_5$ and D$\flat_6$:

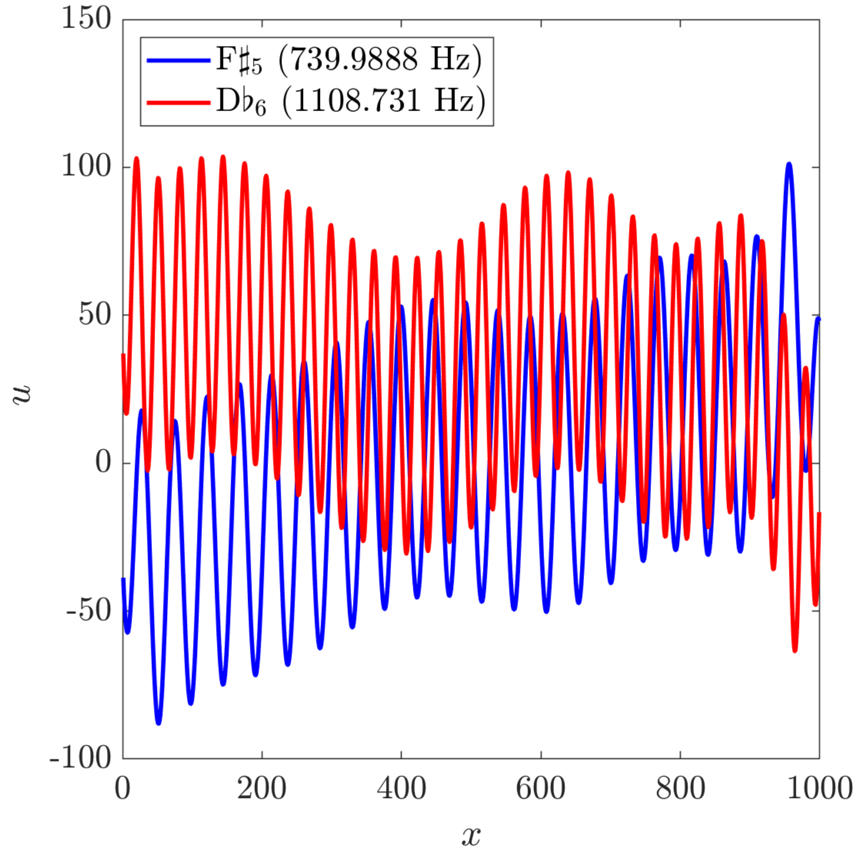

Figure 13: Spatial wave propagation of F$\sharp_5$ and D$\flat_6$ for $\Delta x = 10^{-2}$ and $\Delta t = 10^{-1}$

Looking into the uniqueness of A440 (the A note just above middle C) in comparison to A$_5$ (one octave above), again the spatial propagations can easily be approximated using the Crank-Nicolson method:

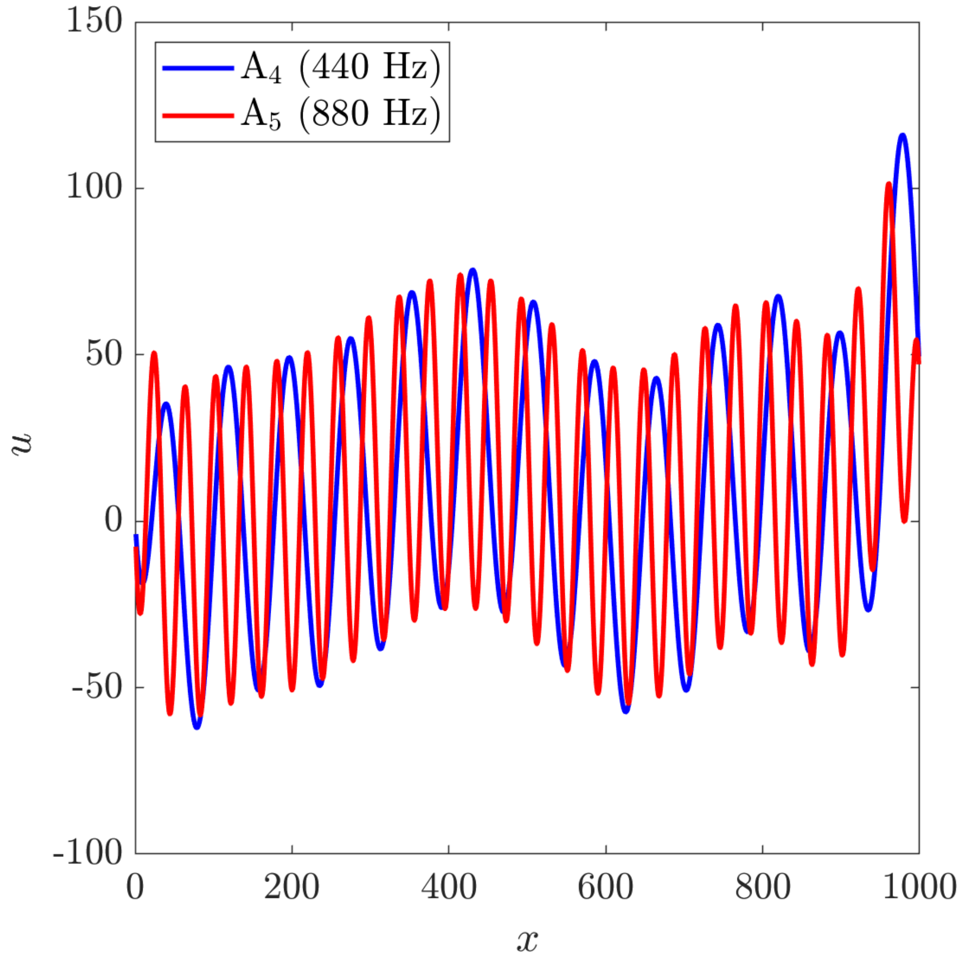

Figure 14: Spatial wave propagation of A440 and A$_5$ for $\Delta x = 10^{-2}$ and $\Delta t = 10^{-1}$

This phenomenon of the A note propagating almost identically regardless of octave is seen across all frequencies of the musical pitch A. Modern practices also show that A440 is often used as a tuning reference due to *just intonation*, which is the tuning of musical intervals as whole number ratios of frequencies (seen with the two A keys compared, at 880 Hz and 440 Hz, a 2:1 ratio) [7]. So, it makes sense that, because of the identical spatial wave propagations with 2:1 frequency ratios regardless of specific octave, it is easier and more straightforward for tuners to tune pianos by looking at different musical pitches in relation to their nearest neighbor A note. Perhaps more useful, it is common to first make sure all A-keys are in tune, which is easy by matching the demonstrated similar propagations between octaves. A more complete tabulated set of piano A-notes and their associated frequencies in comparison to middle C can be found in table (1). Spatial wave propagations of various A-notes are demonstrated in figures (15) and (16).

Scientific Pitch Notation (Note)	Frequency (Hz)
A8	7040
A7	3520
A6	1760
A5	880
A4 (A440)	440
C4 (Middle C)	261.6256
A3	220
A2	110
A1	55

Table 1: Notes and frequencies

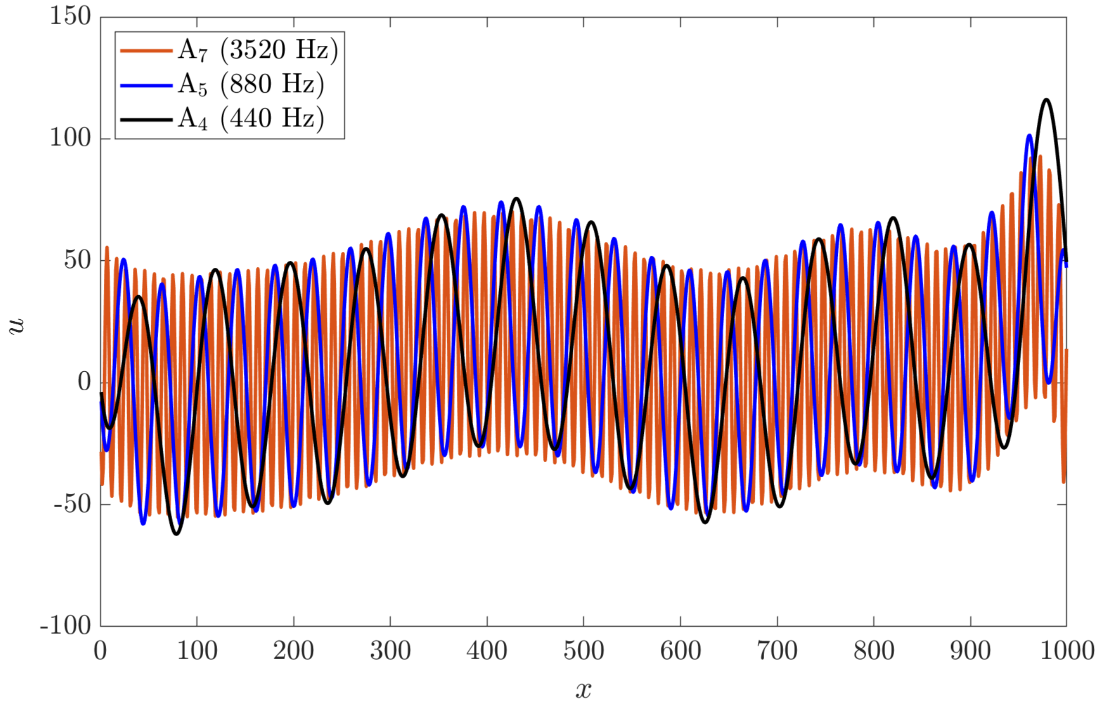

Figure 15: Spatial wave propagations of various A-notes for $\Delta x = 10^{-2}$ and $\Delta t = 10^{-1}$

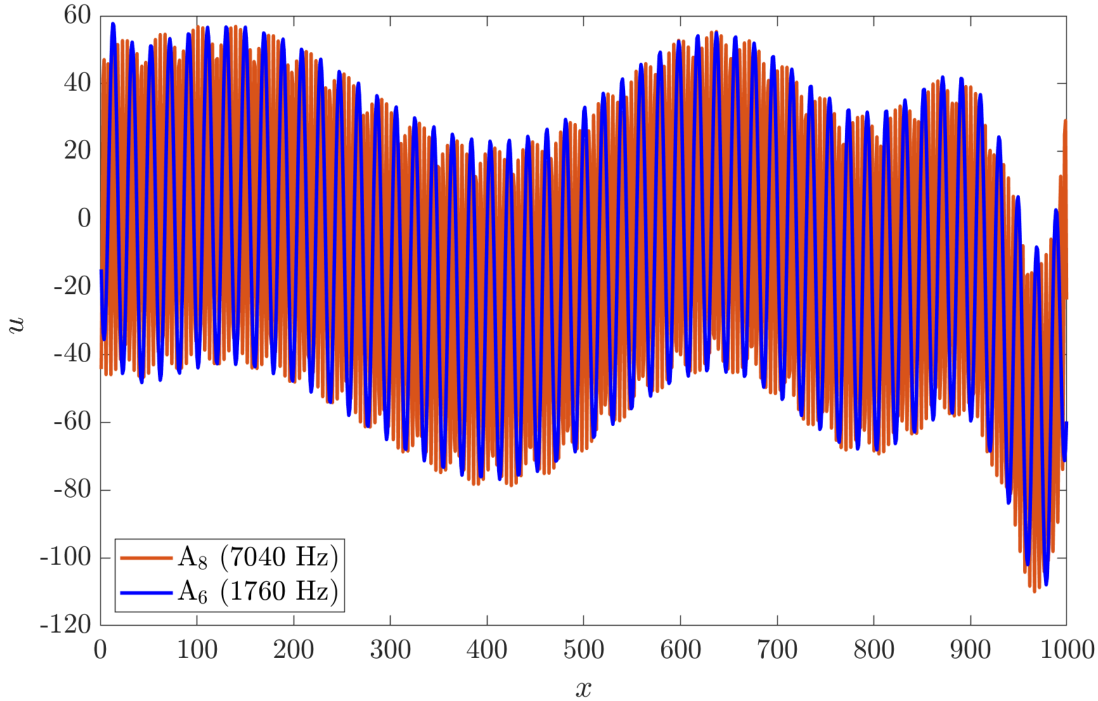

Figure 16: Spatial wave propagations of various A-notes for $\Delta x = 10^{-2}$ and $\Delta t = 10^{-1}$

5.2 Conclusions and Outlook

So what is special about A440? It is the first musical pitch of the A-note right above middle C, and, across octaves, all A-notes propagate remarkably similarly with other specific A-notes, making A440 a prime choice for piano tuners to use when tuning using frequencies. This study proved that the Crank-Nicolson method accurately approximates the solution to the wave equation as it is applicable to acoustic sound waves produced by pianos and other instruments. The Crank-Nicolson algorithm is a good candidate due to its large stability region and accuracy as compared with other methods such as RK4, backward Euler, and forward Euler. Opportunities for further research include using another explicit numerical method, other than RK4, to approximate the solution. Using an explicit method slightly increases runtime efficiency by avoiding more complex nonlinear systems needed to solve for the positions when advancing through time. Also, instead of using a finite difference method, finite element methods that find the best least-squares solution using spectral methods defined in terms of locally defined basis functions could be explored.

References

[1] A. Goza. *Typed Lecture Notes (Various) for AE 370*. 2020. `https://sites.google.com/illinois.edu/ae370/home?authuser=1`.

[2] H. Holden. *The Crank-Nicolson Method*. Norwegian University of Science and Technology, 2015. `https://www.math.ntnu.no/emner/TMA4135/2015h/notater/crank-nicolson/cn.pdf`.

[3] M. Jelinek and J. Mahaffy. *The Method of Manufactured Solutions*. Penn State University, Applied Physics Laboratory. `http://www.personal.psu.edu/jhm/ME540/lectures/VandV/MMS_summary.pdf`.

[4] J. Lehtinen. *Time-domain Numerical Solution of the Wave Equation*. Helsinki University of Technology, 2003. `https://www.cs.unm.edu/~williams/cs530/wave_eqn.pdf`.

[5] L. Olsen-Kettle. *Numerical solution of partial differential equations*. The University of Queensland, School of Earth Sciences, 2011. `http://espace.library.uq.edu.au/view/UQ:239427`.

[6] B. Suits. *Frequencies of Musical Notes, A4 = 440 Hz*. Michigan Tech University, Physics of Music, 1998. `https://pages.mtu.edu/~suits/notefreqs.html`.

[7] D. Wright. *Mathematics and Music*. American Mathematical Society, 2011.

Appendix

```
%% AE 370 Project 2
% Emily Williams
% Acoustic wave equation

%% Spatial Convergence Test

clear all
close all
clc

L = 20;

a = 0;
b = L;
c = 343;
ln = b - a;
T = 20;
dt = 0.1;
u_m = 50;
w = 1;
k = 1;

uex = @(x,t) u_m.*sin(w.*t-k.*x);

% g(x)
g = @(x,t) u_m.*(k.^2.*c.^2 - w.^2).*sin(w.*t-k.*x);

% u(x=a,t) = g_a(t)
g_a = @(t) u_m.*sin(w.*t-k.*a);

% u(x=b,t) = g_b(t)
g_b = @(t) u_m.*sin(w.*t-k.*b);

% initial condition
ueta = @(x) uex(x,0);
veta = @(x) u_m.*w.*cos(-k.*x);

%# of n points to use
nvect = [20; 40; 80; 100];

```

```
%initialize error vect
err_cn = zeros( size(nvect) );
err_be = zeros( size(nvect) );

for j = 1 : length( nvect )

        n = nvect(j);

        % discretize
        xj = (a:ln/n:b)';

        dx = ln/n;

        % A construction
        A = (c^2/dx^2)*(diag(-2*ones(n-1,1),0) + diag(ones(n-2,1),1) + diag(ones(n-2,1),-1));

        B = [ zeros(size(A)) eye(size(A)) ; A zeros(size(A)) ];

        p = @(t) [ zeros(size(g(xj(2:end-1),t))) ; g(xj(2),t)+(c^2/dx^2).*g_a(t) ; g(xj(3:end-2),t) ; g(xj(end-1),t)++(c^2/dx^2).*g_b(t) ];

        % f(z,t)
        f = @(z,t) [ v ; A*u + g(t) ];
    %--

    %-- initialize
        zk_cn = double([ (ueta(xj(2:end-1))) ; veta(xj(2:end-1)) ]);
        zk_be = double([ (ueta(xj(2:end-1))) ; veta(xj(2:end-1)) ]);
        tk = 0;
        tvect = dt : dt : T;

        nsnps = 100;
        ind = max( 1, round(length(tvect)/nsnps) );
        tsv = tvect( 1 : ind : end );

        z_cn = zeros( 2*n-2, length(tsv));
        z_be = zeros( 2*n-2, length(tsv));
        cnt = 1;
    %--

    %-- time stepping
```

```
    for jj = 1 : length( tvect )

        stat = [ num2str(j), ' out of ', num2str(length(nvect)), ': ', num2str(jj), ' out of ', num2str(length(tvect)) ];
        disp(stat)

        tkp1 = tk + dt;

        % kp1
        zkp1_cn = (eye(size(B)) - 0.5*dt*B)\(zk_cn + 0.5*dt*B*zk_cn + 0.5*dt*(p(tk)+p(tkp1)));
        zkp1_be = (eye(size(B)) - dt*B)\(zk_be + dt*p(tkp1));

        % update
        zk_cn = zkp1_cn;
        zk_be = zkp1_be;
        tk = tkp1;

        if min(abs( tkp1-tsv ) ) < 1e-8
            z_cn(:,cnt) = zk_cn;
            z_be(:,cnt) = zk_be;
            cnt = cnt + 1;
        end

    end
    %---

    err_cn(j) = norm( vpa(zkp1_cn(1:n-1,:)) - vpa(uex(xj(2:end-1),tk)) ) / norm( vpa(uex(xj(2:end-1),tk)) );
    err_be(j) = norm( vpa(zkp1_be(1:n-1,:)) - vpa(uex(xj(2:end-1),tk)) ) / norm( vpa(uex(xj(2:end-1),tk)) );

end

figure(100)
F(nsnps) = struct('cdata',[],'colormap',[]);
anim = VideoWriter('animation','MPEG-4');
anim.FrameRate = 10;
anim.Quality = 97;
open(anim)
for i = 1:nsnps
   plot(20,zkp1_cn(i,:),'b.','MarkerSize',20);
```

```
   F(i) = getframe;
   writeVideo(anim,F(i));
end
close(anim)

figure(1000)
for i = 1:nsnps-1
    scatter(xj(i),zkp1_cn(i,:),'filled'), hold on
end
set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
title('Crank-Nicolson Spatial Propagation', 'fontsize', 20, 'interpreter', 'latex')
xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
ylabel('$u$', 'fontsize', 15, 'interpreter', 'latex')
set(gcf, 'PaperPositionMode', 'manual')
set(gcf, 'Color', [1 1 1])
set(gca, 'Color', [1 1 1])
set(gcf, 'PaperUnits', 'centimeters')
set(gcf, 'PaperSize', [20 15])
set(gcf, 'Units', 'centimeters' )
set(gcf, 'Position', [0 0 20 15])
set(gcf, 'PaperPosition', [0 0 20 15])
svnm = 'animation';
print( '-dpng', svnm, '-r200' );


%-- waterfall
    [X,T] = meshgrid( xj(2:end-1), tsv );

    figure(1), subplot(1,2,1)
    waterfall( X,T, uex(X,T) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-80 80])

    subplot(1,2,2)
    y1 = z_cn';
    waterfall( X,T, y1(:,1:n-1) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
```

```
    title({'Crank-Nicolson','$u_{FD}$'}, 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-80 80])

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [20 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 20 15])
    set(gcf, 'PaperPosition', [0 0 20 15])
    svnm = 'waterfall';
    print( '-dpng', svnm, '-r200' );

    figure(2), subplot(1,2,1)
    waterfall( X,T, uex(X,T) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-80 80])

    subplot(1,2,2)
    y2 = z_be';
    waterfall( X,T, y2(:,1:n-1) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title({'Backward Euler','$u_{FD}$'}, 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-80 80])

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [20 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 20 15])
```

```matlab
    set(gcf, 'PaperPosition', [0 0 20 15])
%--

%-- error
    figure(4)
    loglog( ln./nvect, err_cn , 'b.-', 'markersize', 26, 'linewidth', 2 ), hold on
    loglog( ln./nvect, err_be , 'r.-', 'markersize', 26, 'linewidth', 2 )
    h = legend('CN','BE');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
    svnm = 'spat';
    print( '-dpng', svnm, '-r200' );
%--

%% Temporal Convergence Test CN

clear all
close all
clc

L = 10;

a = 0;
b = L;
c = 5;
ln = b - a;
T = 20;
```

```
u_m = 50;
w = 1;
k = 1;

uex = @(x,t) u_m.*sin(w.*t-k.*x);

n = 3000;
dx = (b-a)/n;

% g(x)
g = @(x,t) u_m.*(k.^2.*c.^2 - w.^2).*sin(w.*t-k.*x);

% u(x=a,t) = g_a(t)
g_a = @(t) u_m.*sin(w.*t-k.*a);

% u(x=b,t) = g_b(t)
g_b = @(t) u_m.*sin(w.*t-k.*b);

% initial condition
ueta = @(x) uex(x,0);
veta = @(x) u_m.*w.*cos(-k.*x);

% dt's
dtvect = [0.8e-1; 0.62e-1; 0.68e-1; 0.5e-1];

% error vect
err_cn = zeros( size( dtvect ) );

for j = 1 : length( dtvect )

        dt = dtvect(j);

        % discretize
        xj = (a:ln/n:b)';

        % A construction
        A = (c^2/dx^2)*(diag(-2*ones(n-1,1),0) + diag(ones(n-2,1),1) + diag(
            ones(n-2,1),-1));

        B = [ zeros(size(A)) eye(size(A)) ; A zeros(size(A)) ];

```

```
        p = @(t) [ zeros(size(g(xj(2:end-1),t))) ; g(xj(2),t)+(c^2/dx^2).*g_a(t) ; g(xj(3:end-2),t) ; g(xj(end-1),t)++(c^2/dx^2).*g_b(t) ];

        % f(z,t)
        f = @(z,t) [ v ; A*u + g(t) ];
    %---

    %--- initialize
        zk_cn = double([ ueta(xj(2:end-1)) ; veta(xj(2:end-1)) ]);
        tk = 0;
        tvect = dt : dt : T;
    %---

        M = inv(eye(size(B)) - 0.5*dt*B);

    %--- time stepping
    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        % kp1
        zkp1_cn = M*(zk_cn + 0.5*dt*B*zk_cn + 0.5*dt*(p(tk)+p(tkp1)));

        % update
        zk_cn = double(zkp1_cn);
        tk = tkp1;

    end
    %---

    stat = [ num2str(j), ' out of ', num2str(length(dtvect)) ];
    disp(stat)

    err_cn(j) = norm( vpa(zkp1_cn(1:n-1,:)) - vpa(uex(xj(2:end-1),tk)) ) / norm( vpa(uex(xj(2:end-1),tk)) );

end


%-- error plot
    figure(2)
    m = err_cn(end)*(1./dt^2);
```

```
    loglog( dtvect, m*(dtvect).^2, 'k--', 'linewidth', 2 ), hold on
    loglog( dtvect, err_cn , 'b.', 'markersize', 26 )
    ylim([1e-5 1.5e-3])
    xlim([0.049 0.081])
    yticks(1e-7:0.5e-1:1e-4)

    % formatting
    h = legend('$O(\Delta t^2)$', 'CN');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
    svnm = 'temp';
    print( '-dpng', svnm, '-r200' );
%--

%% Temporal Convergence Test BE

clear all
close all
clc

L = 10;

a = 0;
b = L;
c = 5;
ln = b - a;
T = 20;
u_m = 50;
```

```
w = 1;
k = 1;

uex = @(x,t) u_m.*sin(w.*t-k.*x);

n = 3000;
dx = (b-a)/n;

% g(x)
g = @(x,t) u_m.*(k.^2.*c.^2 - w.^2).*sin(w.*t-k.*x);

% u(x=a,t) = g_a(t)
g_a = @(t) u_m.*sin(w.*t-k.*a);

% u(x=b,t) = g_b(t)
g_b = @(t) u_m.*sin(w.*t-k.*b);

% initial condition
ueta = @(x) uex(x,0);
veta = @(x) u_m.*w.*cos(-k.*x);

% dt's
dtvect = [0.8e-1; 0.62e-1; 0.68e-1; 0.5e-1];

% error vect
err_be = zeros( size( dtvect ) );

for j = 1 : length( dtvect )

        dt = dtvect(j);

        % discretize
        xj = (a:ln/n:b)';

        % A construction
        A = (c^2/dx^2)*(diag(-2*ones(n-1,1),0) + diag(ones(n-2,1),1) + diag(
           ones(n-2,1),-1));

        B = [ zeros(size(A)) eye(size(A)) ; A zeros(size(A)) ];

        p = @(t) [ zeros(size(g(xj(2:end-1),t))) ; g(xj(2),t)+(c^2/dx^2).*g_a
           (t) ; g(xj(3:end-2),t) ; g(xj(end-1),t)++(c^2/dx^2).*g_b(t) ];
```

```

        % f(z,t)
        f = @(z,t) [ v ; A*u + g(t) ];
    %---

    %--- initialize
        zk_be = double([ ueta(xj(2:end-1)) ; veta(xj(2:end-1)) ]);
        tk = 0;
        tvect = dt : dt : T;
    %---

        M = inv(eye(size(B)) - dt*B);

    %--- time stepping
    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        % kp1
        zkp1_be = M*(zk_be + dt*p(tkp1));;

        % update
        zk_be = double(zkp1_be);
        tk = tkp1;

    end
    %---

    stat = [ num2str(j), ' out of ', num2str(length(dtvect)) ];
    disp(stat)

    err_be(j) = norm( vpa(zkp1_be(1:n-1,:)) - vpa(uex(xj(2:end-1),tk)) ) / norm( vpa(uex(xj(2:end-1),tk)) );

end


%-- error plot
    figure(2)
    m = err_be(end)*(1./dt);
    loglog( dtvect, m*(dtvect), 'k--', 'linewidth', 2 ), hold on
    loglog( dtvect, err_be , 'r.', 'markersize', 26 )
```

```
    yticks(1e-7:0.5e-1:1e-4)

    % formatting
    h = legend('$O(\Delta t)$', 'BE');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', '
       fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
    svnm = 'temp_be';
    print( '-dpng', svnm, '-r200' );
%--


%% Frequency Analysis

clear all
close all
clc

L = 10;

a = 0;
b = L;
c = 343;
ln = b - a;
T = 10;
u_m = 50; % dB
freq = [ 261.6256; 523.2511; 391.9954; 739.9888; 1108.731; 7040; 3520; 1760;
   880; 440; 220; 110; 55 ];
n = 1000;
```

```
dx = (b-a)/n;
dt = 0.1;

for j = 1:length(freq)

    w = 2*pi*freq(j);
    lambda = c/freq(j);
    k = 2*pi/lambda;

    uex = @(x,t) u_m.*sin(w.*t-k.*x);

    % g(x)
    g = @(x,t) u_m.*(k.^2.*c.^2 - w.^2).*sin(w.*t-k.*x);

    % u(x=a,t) = g_a(t)
    g_a = @(t) u_m.*sin(w.*t-k.*a);

    % u(x=b,t) = g_b(t)
    g_b = @(t) u_m.*sin(w.*t-k.*b);

    % initial condition
    ueta = @(x) uex(x,0);
    veta = @(x) u_m.*w.*cos(-k.*x);

    % discretize
    xj = (a:ln/n:b)';

    % A construction
    A = (c^2/dx^2)*(diag(-2*ones(n-1,1),0) + diag(ones(n-2,1),1) + diag(ones(n-2,1),-1));

    B = [ zeros(size(A)) eye(size(A)) ; A zeros(size(A)) ];

    p = @(t) [ zeros(size(g(xj(2:end-1),t))) ; g(xj(2),t)+(c^2/dx^2).*g_a(t) ; g(xj(3:end-2),t) ; g(xj(end-1),t)++(c^2/dx^2).*g_b(t) ];

    % f(z,t)
    f = @(z,t) [ v ; A*u + g(t) ];
    %---

    %--- initialize
    zk_cn = double([ ueta(xj(2:end-1)) ; veta(xj(2:end-1)) ]);
```

```
    tk = 0;
    tvect = dt : dt : T;
    %---

    M = inv(eye(size(B)) - 0.5*dt*B);

    %--- time stepping
    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        % kp1
        zkp1_cn = M*(zk_cn + 0.5*dt*B*zk_cn + 0.5*dt*(p(tk)+p(tkp1)));

        % update
        zk_cn = double(zkp1_cn);
        tk = tkp1;

    end
    %---

    z_cn(:,j) = zk_cn(1:n-1);

end


%-- plot
    figure(2)
    plot( z_cn(:,2), '-','color',[0.8500 0.3250 0.0980], 'linewidth', 2 ), hold on
    plot( z_cn(:,4), 'b-', 'linewidth', 2 ), hold on
    plot( z_cn(:,5), 'k-', 'linewidth', 2 )

    % formatting
    h = legend('A$_7$ (3520 Hz)','A$_5$ (880 Hz)','A$_4$ (440 Hz)','interpreter','latex');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$u$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
```

```
    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [25 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 25 15])
    set(gcf, 'PaperPosition', [0 0 25 15])
%     svnm = 'a2';
%     print( '-dpng', svnm, '-r200' );

    figure(6)
    plot( z_cn(:,1), '-','color',[0.8500 0.3250 0.0980],'linewidth', 2 ), hold on
    plot( z_cn(:,3), 'b-', 'linewidth', 2 )

    % formatting
    h = legend('A$_8$ (7040 Hz)','A$_6$ (1760 Hz)','interpreter','latex');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'SouthWest' )

    xlabel( '$x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$u$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [25 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 25 15])
    set(gcf, 'PaperPosition', [0 0 25 15])
%     svnm = 'a3';
%     print( '-dpng', svnm, '-r200' );

    figure(3)
    plot( z_cn(:,3), 'b-', 'linewidth', 2 ), hold on
    plot( z_cn(:,4), 'r-', 'linewidth', 2 )

    % formatting
```

```
    h = legend('C$_4$ (262 Hz)','C$_5$ (523 Hz)','interpreter','latex');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$u$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
%     svnm = 'C';
%     print( '-dpng', svnm, '-r200' );

    figure(4)
    plot( z_cn(:,5), 'b-', 'linewidth', 2 ), hold on
    plot( z_cn(:,4), 'r-', 'linewidth', 2 )

    % formatting
    h = legend('G$_4$ (391.9954 Hz)','interpreter','latex');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$u$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
%     svnm = 'G4';
%     print( '-dpng', svnm, '-r200' );
```

```
    figure(5)
    plot( z_cn(:,6), 'b-', 'linewidth', 2 ), hold on
    plot( z_cn(:,7), 'r-', 'linewidth', 2 )

    % formatting
    h = legend('F$\sharp_5$ (739.9888 Hz)','D$\flat_6$ (1108.731 Hz)','interpreter','latex');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$u$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
%     svnm = 'fs5';
%     print( '-dpng', svnm, '-r200' );
%--
```

```
%% Spatial Convergence Test RK4

clear all
close all
clc

L = 20;

a = 0;
b = L;
c = 343;
ln = b - a;
T = 0.25;
dt = 0.0001;
u_m = 50;
```

```
w = 1;
k = 1;

uex = @(x,t) u_m.*sin(w.*t-k.*x);

% g(x)
g = @(x,t) u_m.*(k.^2.*c.^2 - w.^2).*sin(w.*t-k.*x);

% u(x=a,t) = g_a(t)
g_a = @(t) u_m.*sin(w.*t-k.*a);

% u(x=b,t) = g_b(t)
g_b = @(t) u_m.*sin(w.*t-k.*b);

% initial condition
ueta = @(x) uex(x,0);
veta = @(x) u_m.*w.*cos(-k.*x);

%# of n points to use
nvect = [80; 100; 150; 200];

%initialize error vect
err_rk4 = zeros( size(nvect) );

for j = 1 : length( nvect )

        n = nvect(j);

        % discretize
        xj = (a:ln/n:b)';

        dx = ln/n;

        % A construction
        A = (c^2/dx^2)*(diag(-2*ones(n-1,1),0) + diag(ones(n-2,1),1) + diag(ones(n-2,1),-1));

        B = [ zeros(size(A)) eye(size(A)) ; A zeros(size(A)) ];

        p = @(t) [ zeros(size(g(xj(2:end-1),t))) ; g(xj(2),t)+(c^2/dx^2).*g_a(t) ; g(xj(3:end-2),t) ; g(xj(end-1),t)++(c^2/dx^2).*g_b(t) ];

```

```
        % f(z,t)
        f = @(z,t) [ v ; A*u + g(t) ];
    %--

    %-- initialize
        zk_rk4 = double([ (ueta(xj(2:end-1))) ; veta(xj(2:end-1)) ]);
        tk = 0;
        tvect = dt : dt : T;

        nsnps = 100;
        ind = max( 1, round(length(tvect)/nsnps) );
        tsv = tvect( 1 : ind : end );

        z_rk4 = zeros( 2*n-2, length(tsv));
        cnt = 1;
    %--

    %-- time stepping
    for jj = 1 : length( tvect )

        stat = [ num2str(j), ' out of ', num2str(length(nvect)), ': ', num2str(jj), ' out of ', num2str(length(tvect)) ];
        disp(stat)

        tkp1 = tk + dt;

        % kp1
        y1 = B*zk_rk4 + p(tk);
        y2 = B*zk_rk4 + 0.5*dt*B*y1 + p(tk+0.5*dt);
        y3 = B*zk_rk4 + 0.5*dt*B*y2 + p(tk+0.5*dt);
        y4 = B*zk_rk4 + dt*B*y3 + p(tk+dt);
        zkp1_rk4 = zk_rk4 + (1/6)*dt*(y1+2*y2+2*y3+y4);

        % update
        zk_rk4 = zkp1_rk4;
        tk = tkp1;

        if min(abs( tkp1-tsv ) ) < 1e-8
            z_rk4(:,cnt) = zk_rk4;
            cnt = cnt + 1;
        end

```

```
    end
    %---

    err_rk4(j) = norm( vpa(zkp1_rk4(1:n-1,:)) - vpa(uex(xj(2:end-1),tk)) ) / norm( vpa(uex(xj(2:end-1),tk)) );

end

%-- waterfall
    [X,T] = meshgrid( xj(2:end-1), tsv );

    figure(1), subplot(1,2,1)
    waterfall( X,T, uex(X,T) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-80 80])

    subplot(1,2,2)
    y1 = z_rk4';
    waterfall( X,T, y1(:,1:n-1) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title({'RK4','$u_{FD}$'}, 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-80 80])

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [20 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 20 15])
    set(gcf, 'PaperPosition', [0 0 20 15])
%--

%-- error
    figure(4)
    m = err_rk4(end)/(dx^2);
    loglog( ln./nvect, m*(ln./nvect).^2, 'k--', 'linewidth', 2 ), hold on
```

```matlab
    loglog( ln./nvect, err_rk4 , 'g.', 'markersize', 26, 'linewidth', 2 ), hold on
    h = legend('$O(\Delta x^2)$', 'RK4');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])
    svnm = 'spat_rk4';
    print( '-dpng', svnm, '-r200' );
%--
```

```matlab
%% BE

wr = -5:0.01:3;
wi = -4:0.01:4;
[Wr,Wi] = meshgrid(wr,wi);
BE_sc = abs(1./(1-(Wr+1i*Wi)));
BE_sc(BE_sc<1) = 1; % stable
BE_sc(BE_sc>1) = 2;
cmap = [ 1 0.5 0 ; 1 1 1 ];
contourf(Wr,Wi,BE_sc,[1 2])
colormap(cmap), axis equal
xlabel( '$\mathcal{R}(\Delta t \lambda_i$)', 'interpreter', 'latex', 'fontsize', 16)
ylabel( '$\mathcal{I}(\Delta t \lambda_i$) ', 'interpreter', 'latex', 'fontsize', 16)
title( 'Backward Euler Region of Absolute Stability', 'interpreter', 'latex', 'fontsize', 16)
set(gca, 'TickLabelInterpreter','latex', 'fontsize', 14 )

```

```
set(gcf, 'PaperPositionMode', 'manual')
set(gcf, 'Color', [1 1 1])
set(gca, 'Color', [1 1 1])
set(gcf, 'PaperUnits', 'centimeters')
set(gcf, 'PaperSize', [15 15])
set(gcf, 'Units', 'centimeters' )
set(gcf, 'Position', [0 0 15 15])
set(gcf, 'PaperPosition', [0 0 15 15])
svnm = 'BE_stab';
print( '-dpng', svnm, '-r200' );

%% FE

wr = -5:0.01:3;
wi = -4:0.01:4;
[Wr,Wi] = meshgrid(wr,wi);
FE_sc = abs(1-(Wr+1i*Wi));
FE_sc(FE_sc<1) = 1; % stable
FE_sc(FE_sc>1) = 2;
cmap = [ 1 0.5 0 ; 1 1 1 ];
contourf(Wr,Wi,FE_sc,[1 2])
colormap(cmap), axis equal
xlabel( '$\mathcal{R}(\Delta t \lambda_i$)', 'interpreter', 'latex', 'fontsize', 16)
ylabel( '$\mathcal{I}(\Delta t \lambda_i$) ', 'interpreter', 'latex', 'fontsize', 16)
title( 'Forward Euler Region of Absolute Stability', 'interpreter', 'latex', 'fontsize', 16)
set(gca, 'TickLabelInterpreter','latex', 'fontsize', 14 )

set(gcf, 'PaperPositionMode', 'manual')
set(gcf, 'Color', [1 1 1])
set(gca, 'Color', [1 1 1])
set(gcf, 'PaperUnits', 'centimeters')
set(gcf, 'PaperSize', [15 15])
set(gcf, 'Units', 'centimeters' )
set(gcf, 'Position', [0 0 15 15])
set(gcf, 'PaperPosition', [0 0 15 15])
svnm = 'FE_stab';
print( '-dpng', svnm, '-r200' );

%% CN
```

```
wr = -5:0.01:3;
wi = -4:0.01:4;
[Wr,Wi] = meshgrid(wr,wi);
CN_sc = abs((1+0.5*(Wr+1i*Wi))./(1-0.5*(Wr+1i*Wi)));
CN_sc(CN_sc<1) = 1; % stable
CN_sc(CN_sc>1) = 2;
cmap = [ 1 0.5 0 ; 1 1 1 ];
contourf(Wr,Wi,CN_sc,[1 2])
colormap(cmap), axis equal
xlabel( '$\mathcal{R}(\Delta t \lambda_i$)', 'interpreter', 'latex', 'fontsize', 16)
ylabel( '$\mathcal{I}(\Delta t \lambda_i$) ', 'interpreter', 'latex', 'fontsize', 16)
title( 'Crank-Nicolson Region of Absolute Stability', 'interpreter', 'latex', 'fontsize', 16)
set(gca, 'TickLabelInterpreter','latex', 'fontsize', 14 )

set(gcf, 'PaperPositionMode', 'manual')
set(gcf, 'Color', [1 1 1])
set(gca, 'Color', [1 1 1])
set(gcf, 'PaperUnits', 'centimeters')
set(gcf, 'PaperSize', [15 15])
set(gcf, 'Units', 'centimeters' )
set(gcf, 'Position', [0 0 15 15])
set(gcf, 'PaperPosition', [0 0 15 15])
svnm = 'CN_stab';
print( '-dpng', svnm, '-r200' );
```