

# PFURO 2023

ZOOM: <https://rochester.zoom.us/j/92332283530>

Research Project: 1D Initial Plasma Formation Modeling

Objective: implement a wave-based approach to model light wave propagation in materials

Plans:

- June 5(1st week)
  - attend online classes
  - write/familiar with explicit, implicit finite difference scheme for EM wave propagation. (optional spectrum method)
- June 12 (2nd week)
  - attend online classes
  - start to write a python module to model EM wave propagation
- June 18 (3rd week)
  - continue to finish the 1D EM wave propagation solver
- June 25 (4th week)
  - nearly finish the 1D EM wave propagation solver for non-damping wave
- July 2 (5th week)
  - continue to finish 1D field solver for damping wave case
- July 10 (6th week) : able to model decaying wave in material
- APS DEADLINE: July 14th.
- July 16 (7th week) : integrate in IPF code together. Make sure CH code is working (ray trace result).
- July 23 (8th week) : Begin brainstorming/draft of poster, send to Jack. Get simulation result
- July 30 (9th week) : prepare poster
- Aug 3 (10th week) : PFURO poster presentation
- Aug: (After 10 weeks) CH simulation
- Aug 24 : LLE presentation
- November : APS virtual poster presentation

Suggested Meeting Times

- Monday : 11 am EST
- Tuesday : 4 pm EST
- Wednesday : 11 am EST (3 pm EST)
- Thursday : 3 pm EST (11 am EST)
- Friday : 3 pm/5 pm EST (11 am EST)

## Numerical Method

### Electric Field Solver

In this project, both **explicit** and **implicit** Finite-Difference Time-Domain (FDTD) schemes will be implemented to model the electromagnetic wave propagation in medium. The word “explicit” means that the time-step size “dt” is strictly controlled by Courant–Friedrichs–Lewy

(CFL) condition so that the wave propagation distance within an allowed time-step size cannot exceed the grid size “dx”. Any form of **hyperbolic PDEs** are updated explicitly such as Euler equations in the computational fluid dynamics (CFD). The allowed wave propagation step size within a grid size “dx” is controlled by the CFL number. Typically, CFL ~ 0.5.

$$v_{\text{wave}} \times dt < \text{CFL} \times dx \dots \text{Eq. (0)}$$

An implicit FDTD has the advantage to treat large time-step size with unconditional numerical stability. Any form of **parabolic PDEs** are updated implicitly, in which they have infinite characteristic wave speeds. The goal of this project is to model the laser absorption phenomenon in one dimensional. Because of the fast light signal, the allowed time-step size given by Eq. (0) is very small, compared to other time scales in ICF plasma. Hence, implicit FDTD relieves the harsh condition on the time-step size problem in modeling EM waves.

#### Task 0 in PFURO program

- Report and compare the numerical stability between explicit and implicit FDTD schemes in modeling EM wave propagation for the initial plasma formation process.

Conventional inertial confinement fusion (ICF) codes adopt the geometric ray trace approach to model the light wave propagation in ICF plasma. The ray trace approach in 2D or 3D simulations are highly noisy in modeling the laser energy deposition, because light wave energies are dumped along their geometric paths. A wave-based approach, on the other hand, provides a smooth laser energy deposition modeling in multi-dimensional simulations, as well as the capability to model the initial plasma formation process consistently during the solid-to-plasma transition.

The modeling equation in this project is defined below. It is the electromagnetic (EM) wave equation in nonlinear optical media.

$$\nabla^2 E - \frac{n^2}{c^2} \frac{\partial^2 E}{\partial t^2} = \frac{1}{\epsilon_0 c^2} \frac{\partial^2 P_{\text{NL}}}{\partial t^2} + \mu \sigma \frac{\partial E}{\partial t} \dots \text{Eq. (1)}$$

Here  $P_{\text{NL}}$  is the nonlinear polarization in media. As EM waves propagate into a matter, molecules are polarized and oscillate along the direction of the electric field vector “E”. Depending on crystalline structure, the induced polarization can react with electric fields linearly or nonlinearly. In Eq. (1.0),  $\chi_1$  is known as the linear susceptibility, while  $\chi_2$  means for the second order susceptibility.

$$P = \epsilon_0 (\chi_1 E + \chi_2 E^2 + \dots) = \epsilon_0 \chi_1 E + P_{\text{NL}} \dots \text{Eq. (1.0)}$$

The light speed c and the complex refractive index n relate to the permittivity of free space “ $\epsilon_0$ ”, the permeability of free space  $\mu_0$ , as well as their correspondence in matter.

$$c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}, \quad n = \sqrt{\frac{\mu \epsilon}{\mu_0 \epsilon_0}} \quad \dots \text{Eq. (1.1)}$$

In this project, a non-magnetized plasma is assumed such that ( $\epsilon_0 = 8.85418782 \times 10^{-12}$  in SI unit)

$$\mu \approx \mu_0 \quad \dots \text{Eq. (1.2)}$$

The **complex** refractive index in Eq. (1) is

$$n \approx \sqrt{\epsilon/\epsilon_0} = \sqrt{1 + \chi_1} = \sqrt{\epsilon_r} \quad \dots \text{Eq. (1.3)}$$

where the linear susceptibility  $\chi_1$  is defined in Eq. (1.0). The refractive index is a complex number. Its real part accounts for reflections while the imaginary part accounts for absorptions. For Aluminium at 300 K, its refractive index is,

$$n(\text{Al}; 300K) = n_r + in_i = 0.85 + i6.48 \quad \dots \text{Eq. (1.3.0)}$$

The term  $\epsilon_r$  is known as the relative permittivity. It can be represented by a classical Lorentz model analytically.

$$\epsilon_r \equiv 1 + \chi_1 \quad \dots \text{Eq. (1.4)}$$

The last term in Eq. (1) refers to the formation of transient current characterized by the electric conductivity  $\sigma$ . The induced current is damped by the material resistance.

#### Task 1 during the first two weeks in PFURO program

- Derive an explicit finite scheme to discretize Eq. (1) using the second-order central finite differencing in time and space.
- Assume a **linear susceptibility** in Eq. (1) so that  $P_{NL} = 0$
- Ignore the transient current term in Eq. (1).
- Assume a room-temperature complex refractive index for Aluminum in Eq. (1.3.0)

$$\nabla^2 E - \frac{n^2}{c^2} \frac{\partial^2 E}{\partial t^2} = 0 \quad \dots \text{Eq. (2)}$$

Using the central differencing, the first-order spatial derivative, the second-order spatial derivative and the second-order time derivative are

$$\frac{\partial E(t^n)}{\partial x} \approx \frac{E^n(x_{i+1/2}) - E^n(x_{i-1/2})}{\Delta x} = \frac{E_{i+1/2}^n - E_{i-1/2}^n}{\Delta x} \quad \dots \text{Eq. (3.0)}$$

$$\frac{\partial^2 E(t^n)}{\partial x^2} = \frac{E_{i+1}^n - 2E_i^n + E_{i-1}^n}{\Delta x^2} \quad \dots \text{Eq. (3.1)}$$

$$\frac{\partial^2 E(x_i)}{\partial t^2} = \frac{E_i^{n+1} - 2E_i^n + E_i^{n-1}}{\Delta t^2} \dots \text{Eq. (3.2)}$$

Substituting Eqs. (3.1)-(3.2) into Eq. (2), the explicit update for the electric field is

$$E_i^{n+1} = \left( \frac{\Delta t}{\Delta x} \frac{c}{n} \right)^2 (E_{i+1}^n - 2E_i^n + E_{i-1}^n) - (-2E_i^n + E_i^{n-1}) \dots \text{Eq. (4.0)}$$

The first bracket in Eq. (4) is required to satisfy Courant condition as stated in Eq. (0)

$$\frac{\Delta t}{\Delta x} \frac{c}{n} < 1 \dots \text{Eq. (4.1)}$$

As the light waves propagate inside a material, the light wave interacts with electrons so that electrons are set into oscillations along the electric field vector direction. The oscillating electrons then collide with neighboring atoms so that the light wave energy is transferred from the electric field to electrons and ions eventually. This process is known as the collisional damping of light waves and is described by a complex refractive index. The incident electric field is described by a complex number

$$E_L = E_0 e^{-i(\omega t - kx)} = E_0 \cos(kx - \omega t) + iE_0 \sin(kx - \omega t) \dots \text{Eq. (4.2)}$$

The built-in Python function “complex” can be used to store complex-valued numbers, so that the incident electric field is represented by

$$E_L = \mathcal{C}(E_0 \cos(kx - \omega t), E_0 \sin(kx - \omega t)) \dots \text{Eq. (4.3)}$$

The complex-valued incident electric field provides the initial boundary condition in the ghost cells. The electric field inside the computational domain is initialized as

$$E(t, x) = \mathcal{C}(E_R, E_I) = \mathcal{C}(0, 0) \dots \text{Eq. (4.4)}$$

Similarly, the complex-valued refractive index is given by

$$n = \mathcal{C}(n_R, n_I) \dots \text{Eq. (4.5)}$$

By substituting complex-valued representations Eqs. (4.3) - (4.5) into Eq. (4.0), the wave propagation in matter is modeled by

$$\begin{aligned} \mathcal{C}(E_R, E_I)_i^{n+1} = & \mathcal{C}(r_R, r_I) (\mathcal{C}(E_R, E_I)_{i+1}^n - 2\mathcal{C}(E_R, E_I)_i^n + \mathcal{C}(E_R, E_I)_{i-1}^n) \\ & - (-2\mathcal{C}(E_R, E_I)_i^n + \mathcal{C}(E_R, E_I)_i^{n-1}) \dots \text{Eq. (4.6)} \end{aligned}$$

where the variable  $r$  becomes a complex constant

$$r = \left( \frac{\Delta t}{\Delta x} \frac{c}{\mathcal{C}(n_R, n_I)} \right)^2 = \mathcal{C}(r_R, r_I) \dots \text{Eq. (4.7)}$$

However, as demonstrated by Nick's work in the July 11 meeting, the direct implementation of a complex refractive index leads to oscillatory numerical solutions. The Fourier analysis is applied to understand this phenomenon. By substituting the following electric field form into Eq. (2),

$$E(x, t) = E_0 e^{-i(\omega t - kx)} \dots \text{Eq. (4.8)}$$

the second-order time derivative becomes

$$\partial_{tt} E = -\omega^2 E \dots \text{Eq. (4.9)}$$

The wave equation in Eq. (2) becomes

$$\nabla^2 E + \frac{\omega^2 n^2}{c^2} E = 0 \dots \text{Eq. (4.10)}$$

In 1-D the reduced wave equation is

$$\partial_{xx} E = -\frac{\omega^2 n^2}{c^2} E \dots \text{Eq. (4.11)}$$

When the refractive index is real (denoted by the subscript "R"), Eq. (4.11) accepts a sinusoidal wave solution

$$E(x) = E_0 e^{i(\omega n_R/c)x} = E_0 e^{ik_R x} \dots \text{Eq. (4.12)}$$

where the wavenumber is

$$k_R \equiv n_R(\omega/c) \dots \text{Eq. (4.13)}$$

When the refractive index is purely imaginary (denoted by the subscript "I"),

$$n = in_I \dots \text{Eq. (4.14)}$$

the 1-D wave equation in Eq. (4.11) becomes

$$\partial_{xx} E = \frac{\omega^2 n_I^2}{c^2} E \dots \text{Eq. (4.15)}$$

The above equation accepts an exponentially decaying or growing solution

$$E(x) = E_0 e^{\pm(\omega n_I/c)x} = E_0 e^{\pm k_I x} \dots \text{Eq. (4.16)}$$

where the wavenumber is

$$k_I \equiv n_I(\omega/c) \dots \text{Eq. (4.17)}$$

A positive sign in Eq. (4.14) corresponds to a growing solution whereas the negative sign corresponds to a decaying solution. For laser propagations in matter, the negative sign is the correct solution to describe an attenuating wave. During the numerical update, the selection of the sign depends on the boundary condition. To see this, by substituting Eq. (3.1) into Eq. (4.15), the numerical update is

$$\frac{E_{i+1} - 2E_i + E_{i-1}}{\Delta x^2} = k_I^2 E_i \dots \text{Eq. (4.18)}$$

such that

$$E_{i+1} = (\Delta x^2 k_I^2 + 1)E_i + (E_i - E_{i-1}) \dots \text{Eq. (4.19)}$$

The first bracket in the above equation represents a growing term if the local electric field is positive

$$(\Delta x^2 k_I^2 + 1)E_i > E_i, \quad \text{if } E_i > 0 \dots \text{Eq. (4.20)}$$

It also represents a decaying term if the local electric field is negative

$$(\Delta x^2 k_I^2 + 1)E_i < E_i, \quad \text{if } E_i < 0 \dots \text{Eq. (4.21)}$$

The sign of the first bracket in Eq. (4.19) only depends on the information of the local electric field, whereas the sign of the second bracket in Eq. (4.19) reads the boundary condition, depending on the sign of the local slope right at the boundary

$$E_i - E_{i-1} > 0, \quad \text{if } dE/dx > 0 \dots \text{Eq. (4.22)}$$

$$E_i - E_{i-1} < 0, \quad \text{if } dE/dx < 0 \dots \text{Eq. (4.23)}$$

Equations (4.20) - (4.23) produce the oscillatory numerical solution! To model the wave propagation with a decaying wave amplitude, Eq. (2) is updated by two solvers separately. The first kernel is called “Propagator”, which models the wave propagation in matter without changing the instantaneous wave amplitude.

$$E_i^{n+1(*)} = \left( \frac{\Delta t}{\Delta x} \frac{c}{n_R} \right)^2 (E_{i+1}^n - 2E_i^n + E_{i-1}^n) - (-2E_i^n + E_i^{n-1})$$

... Eq. (4.24)

The second kernel is called “Absorber”, which models the wave attenuation

$$\boxed{E_i^{n+1} = E_i^{n+1(*)} \times e^{-k_I x_i}} \quad \dots \text{Eq. (4.25)}$$

In the above treatment, the 1-D analytic solution is used. Ideally, the absorption kernel can be solved as

$$E_i^{n+1} = E_i^{n+1(*)} \times A_i^{n+1} \quad \dots \text{Eq. (4.26)}$$

where the wave amplitude component is updated by solving an separate ODE

$$\partial_{xx} A = \frac{\omega^2 n_I^2}{c^2} A = k_I^2 A \quad \dots \text{Eq. (4.27)}$$

where the initial condition for the wave amplitude must be have a negative slope at the boundary, where the wave enter from vacuum into the matter

$$\frac{dA}{dx} < 0, \quad \text{at B. C. : } x = 0 \quad \dots \text{Eq. (4.28)}$$

In 3-D, the wave attenuation is updated by solving an separate PDE

$$\nabla^2 A = \frac{\omega^2 n_I^2}{c^2} A = k_I^2 A \quad \dots \text{Eq. (4.29)}$$

#### Nick's Work for Week of June 5:

I followed the derivation of Eq (4.0) by subtracting Eq (3.1) and Eq. (3.2) [Eqs. (3.1)-(3.2)] and tried to replicate it but for **implicit finite scheme** but I kept getting strange answers. So I went on Wikipedia/Google and found a formula so that is what I have written down.

From [Wikipedia](#), we have the following implicit finite difference formula

$$u_j^n = (1 + 2r)u_j^{n+1} - ru_{j-1}^{n+1} - u_{j+1}^{n+1}$$

Where  $r = k/h^2$  which are constants that stem back from the derivation of the formula. I

believe in our case 'r' would be  $\left(\frac{\Delta t \ c}{\Delta x \ n}\right)^2$  to satisfy Courant condition. So now we basically just replace 'u' with our 'E' and we have our implicit finite scheme

$$E_j^n = (1 + 2r)E_j^{n+1} - rE_{j-1}^{n+1} - E_{j+1}^{n+1}$$

From Jack : As long as the PDEs are identical, the same results should be obtained.

---

To derive an implicit update, the EM wave equation in Eq. (2) is written as

$$\frac{\partial^2 E}{\partial t^2} = \frac{c^2}{n^2} \nabla^2 E^{n+1} \quad \dots \text{Eq. (5)}$$

The right-hand-side variables are evaluated at the next time-step  $t^{n+1}$ . To discretize Eq. (5), the second-order time derivative is replaced with Eq. (3.2) while the second-order spatial

derivative is replaced with Eq. (3.1). The Laplacian operator in Eq. (5) is the gradient vector that contains spatial derivatives in three-dimensional,

$$\vec{\nabla} = \partial_x \hat{x} + \partial_y \hat{y} + \partial_z \hat{z} \dots \text{Eq. (5.0)}$$

such that

$$\vec{\nabla} \cdot \vec{\nabla} = \nabla^2 = \partial_x^2 + \partial_y^2 + \partial_z^2 \dots \text{Eq. (5.0.1)}$$

In one-dimensional, the Laplacian operator is

$$\nabla^2 \rightarrow \partial_x^2 \dots \text{Eq. (5.0.2)}$$

so that the 3-D EM wave equation in Eq. (5) is reduced to

$$\frac{\partial^2 E}{\partial t^2} = \frac{c^2}{n^2} \partial_x^2 E^{n+1} \dots \text{Eq. (5.0.3)}$$

The required second-order time and spatial derivatives are

$$\partial_{tt} E = \Delta t^{-2} (E_i^{n+1} - 2E_i^n + E_i^{n-1}) \dots \text{Eq. (5.1)}$$

$$\partial_{xx} E = \Delta x^{-2} (E_{i+1}^{n+1} - 2E_i^{n+1} + E_{i-1}^{n+1}) \dots \text{Eq. (5.2)}$$

By defining

$$r \equiv (\Delta t / \Delta x)^2 \cdot (c/n)^2 \dots \text{Eq. (5.3)}$$

the implicit update for Eq. (5) becomes

$$-2E_i^n + E_i^{n-1} = rE_{i+1}^{n+1} - (1 + 2r)E_i^{n+1} + rE_{i-1}^{n+1} \dots \text{Eq. (6)}$$

The solution for the electric field in the next time-step in Eq. (6) is solved by a matrix inversion, with appropriate boundaries in both time and spatial domains.

## Laser-Matter Interaction

After obtaining the numerical solution for electric fields, the laser intensity in matter is

$$I(x, t) = \frac{c \cdot n(x, t) \cdot \epsilon_0}{2} E(x, t)^2 \dots \text{Eq. (7)}$$



where  $c$  is the light speed,  $n$  is the refractive index, and  $\epsilon_0$  is the permittivity of free space. The intensity  $I$  carries the unit of  $\text{J/m}^2/\text{s}$ . EM wave energies are absorbed continuously as the light wave propagates deeper in matter. The rate of laser energy deposition per unit volume, which has the unit of  $\text{J/s/m}^3$ , satisfies Beer-Lambert law

$$S(x, t) = -2k_i I - W_{\text{PI}} E_g \dots \text{Eq. (8)}$$

Here  $k_{\text{imaginary}}$  is the imaginary part of the wavevector

$$k_i = \frac{\omega_L}{c} n_i = \frac{1}{cn} \frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \dots \text{Eq. (9)}$$

which is given by the product of the imaginary part of the material refractive index  $n_i$  and the wavevector in free space  $\omega_L/c$ . Here  $\omega_L$  is the angular frequency of the light wave,  $\omega_p$  is the plasma frequency, and  $\nu_c$  is the electron-phonon momentum relaxation rate in collisions.  $W_{\text{PI}}$  is the multi-photoionization rate per unit volume while  $E_g$  is the band gap of dielectrics. The resulting laser energy deposition in Eq. (8) becomes

$$S(x, t) = -\frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \cdot \epsilon_0 E(x, t)^2 - W_{\text{PI}} E_g \dots \text{Eq. (10)}$$

The first term on the right hand side is the well-known Joule heating

$$\left( \frac{dU}{dt} \right)_{\text{JH}} = \frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \cdot \epsilon_0 E(x, t)^2 \dots \text{Eq. (11)}$$

The multi-photoionization rate per unit volume is given by

$$W_{\text{PI}} = \sigma_0 I^n$$

where the positive integer  $n$  is the number of photons required to excite an electron from the valence to the conduction bands. It is given by the integer part of the ratio of the band gap energy to the photon energy

$$n = \langle E_g / (\hbar\omega) + 1 \rangle$$

## Heat Transfer

After obtaining the laser energy deposition from Eq. (10), the source term  $S$  is coupled to the electron heat transfer equation

$$c_{\text{ve}} \frac{dT_e}{dt} = \frac{\partial}{\partial x} \kappa \frac{\partial T_e}{\partial x} - G(T_e - T_i) + S \dots \text{Eq. (12)}$$

Here  $c_{ve}$  is the electron specific heat capacity at the constant volume, in which  $c_{ve}$  has the unit of  $1/m^3$ .  $\kappa$  is the solid-state electron thermal conductivity

$$\kappa = \kappa_0 \left( \frac{T_e}{T_i} \right) \dots \text{Eq. (13)}$$

which is proportional to the ratio of the electron temperature  $T_e$  to the phonon temperature  $T_i$ . Both temperatures have the units of J.  $G$  is the electron-phonon heat transfer rate. After obtaining the electron temperature update from Eq. (12), the phonon temperature  $T_i$  is updated

$$c_{vi} \frac{dT_i}{dt} = G(T_e - T_i) \dots \text{Eq. (14)}$$

where  $c_{vi}$  is the phonon specific heat capacity at the constant volume.

## Initial Plasma Formation

In summary, the governing equations for 1D initial plasma formation (IPF) modeling are

$$\begin{aligned} \partial_{tt} E &= (c/n)^2 \partial_{xx} E \\ c_{ve} \frac{dT_e}{dt} &= \frac{\partial}{\partial x} \kappa \frac{\partial}{\partial x} T_e - G(T_e - T_i) + S \\ c_{vi} \frac{dT_i}{dt} &= G(T_e - T_i) \end{aligned}$$

where the laser energy deposition source term is

$$\begin{aligned} S &= -\frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \cdot \epsilon_0 E^2 - W_{PI} E_g \\ W_{PI} &= \sigma_0 I^n \\ n &= \langle E_g / (\hbar \omega) + 1 \rangle \end{aligned}$$

The original 1D-IPF code contains existing subroutines to calculate

$$\begin{aligned} c_{ve}(T_e) &\text{ by a quantum model} \\ c_{vi}(T_i) &\text{ by a Debye model} \\ \nu_c &\text{ by a harmonic mean model} \end{aligned}$$

The rest parameters can be taken as constants including

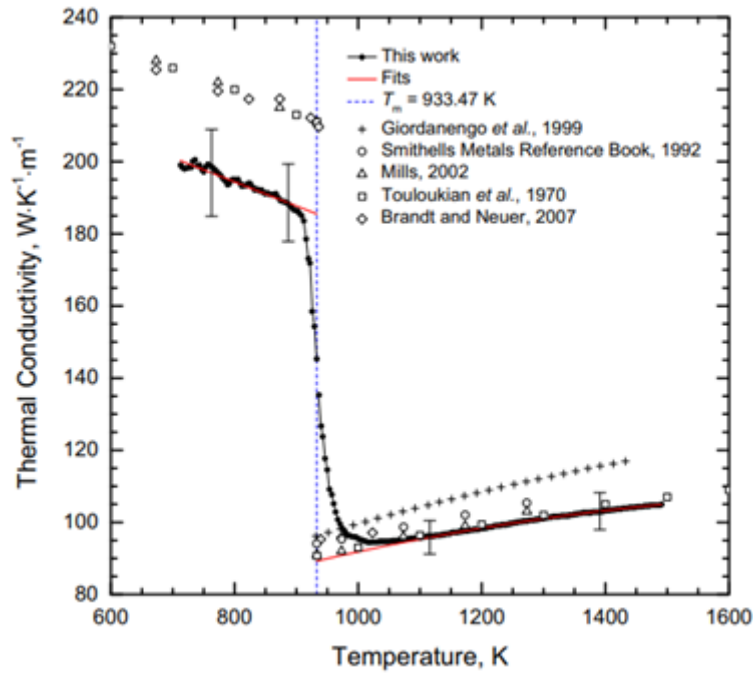
$$G, \kappa_0, \sigma_0, E_g$$

# Aluminum

The previous model is general for laser light propagation in dielectrics. The governing equations for laser light propagation in metals is simpler because there is no band gap term.

$$\begin{aligned}\partial_{tt}E &= (c/n)^2 \partial_{xx}E \\ S &= -\frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \cdot \epsilon_0 E^2 \\ c_{ve} \frac{dT_e}{dt} &= \frac{\partial}{\partial x} \kappa \frac{\partial}{\partial x} T_e - G(T_e - T_i) + S \\ c_{vi} \frac{dT_i}{dt} &= G(T_e - T_i)\end{aligned}$$

The electron thermal conductivity  $\kappa$  decreases with the lattice temperature before the melting point (933.47 Kelvin), and drops to about a half after transiting into liquid Aluminum. The variation of  $\kappa$  as a function of lattice temperatures can be seen from [Ref. Leitner, M., Leitner, T., Schmon, A. et al. Metall Mater Trans A 48, 3036–3045 (2017)]. At room temperature (300 Kelvin), the electron thermal conductivity is 237 W/m/K. It decreases linearly to 185.5 W/m/K at the melting point (933.47 K). After the phase transition into the liquid state, the electron thermal conductivity is 89.5 W/m/K, and grows gradually towards the vaporization point (2740 K).



The electron-phonon momentum relaxation rate  $\nu_c$  in Eq. (9) is the inverse of the characteristic time scale  $\tau_c$  for a single electron to relax its momentum from the excited state to the unperturbed state during the head-on collision with a vibrating lattice. It is also known as the collision frequency for the momentum exchange. Its behavior over a wide range of lattice temperatures can be seen from [Ref. K. Eidmann et al. Phys. Rev. E 62, 1202 (2000)].

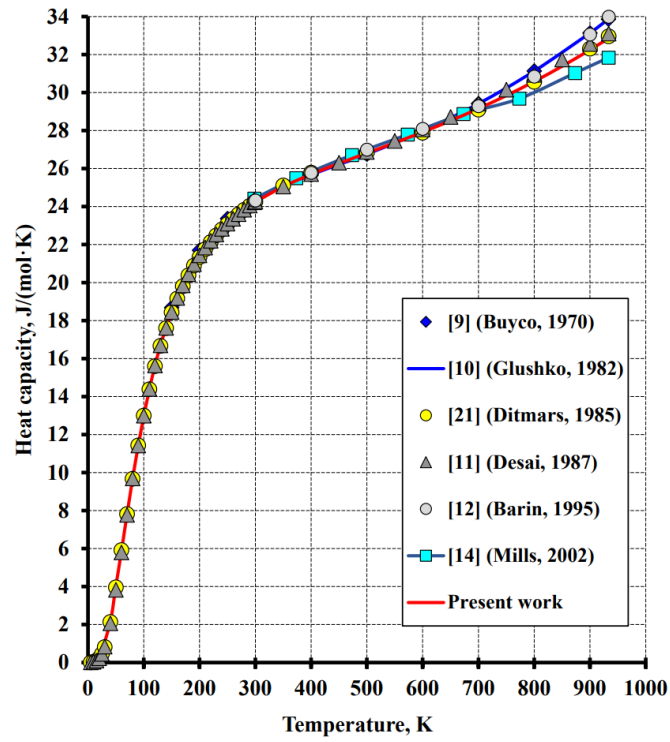
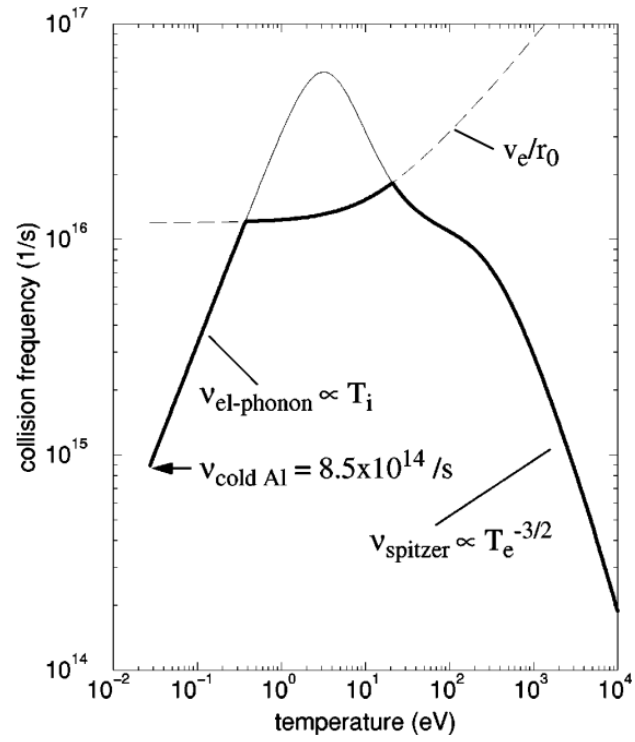


Figure 1. Isobaric heat capacity of solid aluminum.

## 5/26/2023 Meeting

- use Google Doc for research communication
- give a tutorial for the IPF code in the next meeting


## 5/30/2023 Meeting

- not sure for virtual in APS DPP
  - From Jack : email DPP staff to confirm the availability for virtual poster for undergraduate research
  - From Nick : they were not sure either.
- begin deriving an explicit finite difference scheme to solve Eq. (2)
- send 1D-IPF code to Nick
- next meeting during the week of June 5
- 27 hours driving from May 31 to June 4

## 06/07/2023 Meeting

- Pre-meeting questions/notes:
  - How did you know to switch to a FDTD? I understand the advantages because you explained them, but I never would have thought to do this. Interested in why/how you knew what scheme to choose.
    - ☐ Same thing as Finite Difference.
  - In “Task 1” notes, it mentions I should derive an explicit finite scheme. Should that say explicit finite time domain scheme?
  - **Note:** Paper/research note that has example of 1D FDTD code.  
<https://my.ece.utah.edu/~ece6340/LECTURES/lecture%2014/FDTD.pdf>
- Getting information for LLE resources. VPN is important.
- Want to derive an implicit finite difference scheme for week of June 5

# 06/14/2023 Meeting

- Pre-meeting questions/notes:
  - Best way to share my work in my notebook?
    - ☐ One idea is github.  
[https://github.com/nickdavila/PFURO-2023/blob/main/Code/finite\\_difference\\_schemes.ipynb](https://github.com/nickdavila/PFURO-2023/blob/main/Code/finite_difference_schemes.ipynb)
  - Commented question about deriving equation Eqs (5.1) and (5.2) 
- Action Items:
  - ☒ ~~Work on explicit scheme for Week of June 12, 2023~~
  - ☒ ~~Change u variable to E~~
  - ☒ ~~Add bottom point to diagram~~
  - ☒ ~~change h and k to delta\_x and delta\_t~~
  - ☒ ~~Another four arrays: I, T\_e, T\_i, N\_e, rho~~
  - ☒ ~~Add ghost cells for x. One on left one on right~~
  - ☒ ~~From Jack : email DPP staff to confirm the availability for virtual poster for undergraduate research~~
    - Virtual option is available!
  - ☐ Get deliverable in writing

# 06/22/2023 Meeting

- Pre-meeting questions/notes:
  - Ask about sizing of arrays, do I also add ghost cells for time array?.
  - Make sure to have correct number for index of refraction
  - Ask for the physics deliverable in writing
- TIMES WHEN JACK IS FREE:
  - Morning:
    - ☐ Wednesday and Thursday at 11 am
  - Afternoon:

☐ Tuesday, Wednesday, and Thursday at 3 pm

- Action Items:

- ☐ Get deliverable in writing

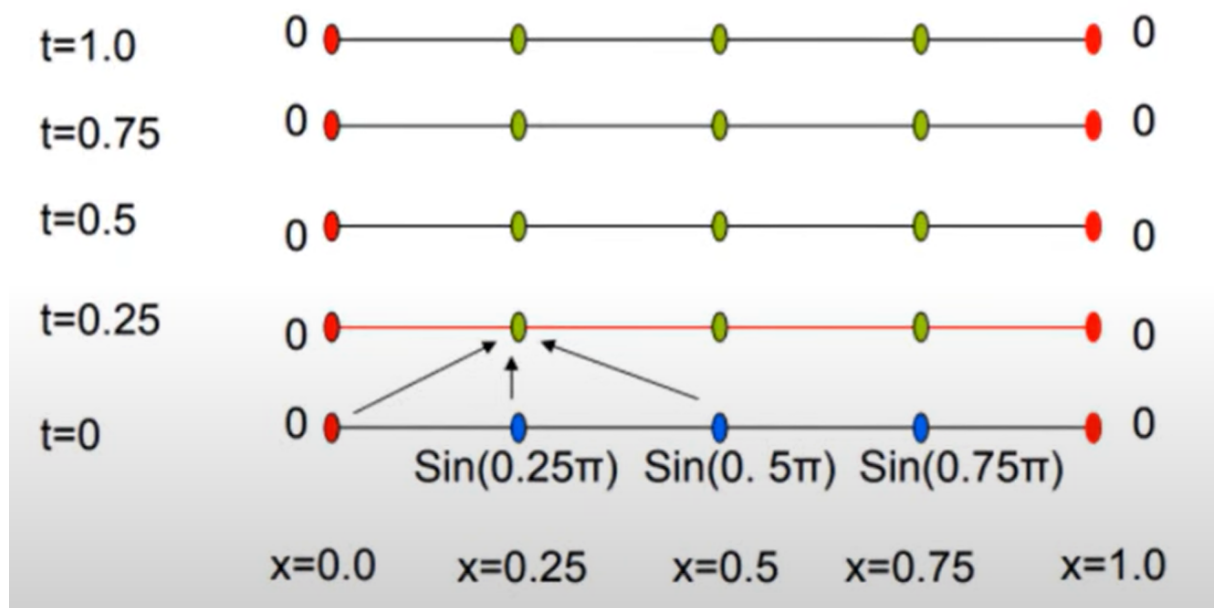
- ☐ Get the abstract done in one week.

## 06/27/23 Meeting

GITHUB LINK:

[https://github.com/nickdavila/PFURO-2023/blob/main/Code/finite\\_difference\\_schemes.ipynb](https://github.com/nickdavila/PFURO-2023/blob/main/Code/finite_difference_schemes.ipynb)

- **Boundary conditions:** I was going off of a video/book where their explicit grid looked like this:



- So using this grid, the boundary conditions are where the left and right 'walls' are where the red dots are. I know we created ghost cells so would our boundary conditions be the beginning of the ghost cells?
- **Initial conditions:** For my initial conditions I chose a random number, but I'm very sure that is wrong. How do I pick a good initial condition?
- **Building my E matrix:** So I just used the length of my spatial and time domains to create my E matrix, but is there a better way to do this?



## Problem Solving after meeting 06/27/23

```
# Assignment of boundary conditions in space:

# Instantaneously defining the boundary conditions/values of ghost cells on the RIGHT:
E[n, 0] = E_0 * np.sin( angular_frequency * t_sim * -wavenumber * x[0] )
E[n, 1] = E_0 * np.sin( angular_frequency * t_sim * -wavenumber * x[1] )

# We have no laser on the left, so boundary conditions are just 0
E[n, -1] = 0
E[n, -2] = 0

# Initial conditions in time:
E[1, 2:] = 0
for i in range(2, len(x) - 3):
    print('index', index, 'n', n, 'i', i)
    E[n+1, i] = (r_const ** 2) * (E[n, i-1] + (1 - 2*r_const) * E[n, i] + r_const * E[n, i+1]) - (-2 * E[n, i] + E[n-1, i])
    index += 1
```

index 113 n 13 i 5  
index 114 n 13 i 6  
index 115 n 13 i 7  
index 116 n 13 i 8  
index 117 n 13 i 9  
index 118 n 13 i 10  
index 119 n 13 i 11  
index 120 n 14 i 2

```
-----
IndexError                                Traceback (most recent call last)
Cell In[23], line 39
    37 for i in range(2, len(x) - 3):
    38     print('index', index, 'n', n, 'i', i)
--> 39     E[n+1, i] = (r_const ** 2) * (E[n, i-1] + (1 - 2*r_const) * E[n, i] + r_const * E[n, i+1]) - (-2 * E[n, i] + E[n-1, i])
    40     index += 1

IndexError: index 15 is out of bounds for axis 0 with size 15
```

In [24]: E.shape  
Out[24]: (15, 1005)

- Finished off the meeting with an out of bounds error. I realized that the error stemmed from how we defined our E matrix (  $E = \text{np.zeros}((\text{len}(x), \text{len}(t)))$  ). So since our x domain is only 15 large, when  $E[n+1, i]$  is calculated and  $n = 14$ , we get an out of bounds error because the shape of our E matrix is (15, 1005).
- This is because our 'n' goes through time for us, so since the length of our time array is 1005 (1000 timesteps plus 2 cells on the left and 3 on the right), this loop (image below) will go from 2 to 996 and so will inevitably go over 15.

```
index = 0
for n in range(2, len(t) - 3):
    t_sim = t[n] # in seconds
```

- My first instinct was to change the definition of the E matrix to  $E = \text{np.zeros}((\text{len}(t), \text{len}(x)))$  which fixed the issue of the index out of bounds. But now I was left with another “possible” problem.

```

index += 1

index 9986 n 1000 i 8
index 9987 n 1000 i 9
index 9988 n 1000 i 10
index 9989 n 1000 i 11
index 9990 n 1001 i 2
index 9991 n 1001 i 3
index 9992 n 1001 i 4
index 9993 n 1001 i 5
index 9994 n 1001 i 6
index 9995 n 1001 i 7
index 9996 n 1001 i 8
index 9997 n 1001 i 9
index 9998 n 1001 i 10
index 9999 n 1001 i 11

C:\Users\nickd\AppData\Local\Temp\ipykernel_2376\2860737529.py:39: RuntimeWarning: overflow encountered in scalar multiply
E[n+1, i] = (r_const) * (E[n, i-1] + (1 - 2*r_const) * E[n, i] + r_const * E[n, i+1]) - (-2 * E[n, i] + E[n-1,i])
C:\Users\nickd\AppData\Local\Temp\ipykernel_2376\2860737529.py:39: RuntimeWarning: invalid value encountered in scalar subtra
ct
E[n+1, i] = (r_const) * (E[n, i-1] + (1 - 2*r_const) * E[n, i] + r_const * E[n, i+1]) - (-2 * E[n, i] + E[n-1,i])

In [97]: plt.plot(E)
#plt.legend(t)
plt.show()

def _locator(self, vmin, vmax)
2141     vmin = -vmax
2142     vmin, vmax = mtransforms.nonsingular(
2143         vmin, vmax, expander=1e-13, tiny=1e-14)
-> 2144     locs = self._raw_ticks(vmin, vmax)
2146     prune = self._prune
2147     if prune == 'lower':

File ~\OneDrive\Documents\GitHub\PFURO-2023-PRIV\Code\pfurovenv\Lib\site-packages\matplotlib\ticker.py:2127, in MaxNLocator._
raw_ticks(self, vmin, vmax)
2125     low = edge.le(_vmin - best_vmin)
2126     high = edge.ge(_vmax - best_vmin)
-> 2127     ticks = np.arange(low, high + 1) * step + best_vmin
2128     # Count only the ticks that will be displayed.
2129     nticks = ((ticks <= _vmax) & (ticks >= _vmin)).sum()

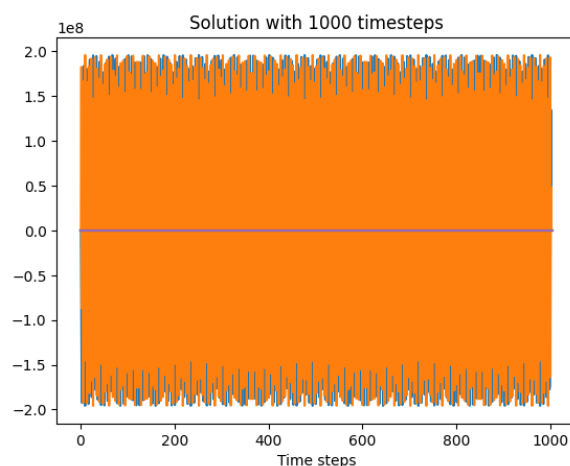
ValueError: range: cannot compute length

<Figure size 640x480 with 1 Axes>

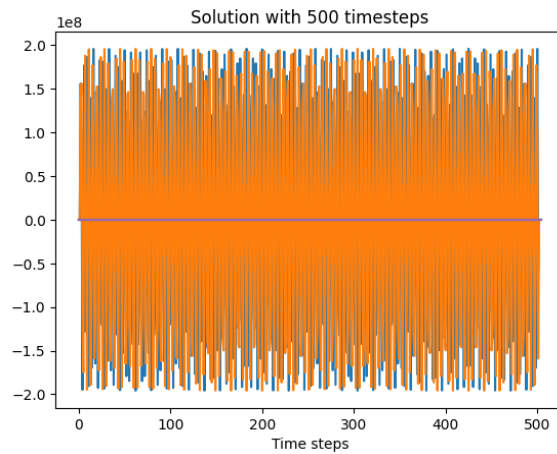
```

- So my code runs, but now it won't plot. I figured out it's because I have some 'nan' values in my array. So once I removed them I was able to plot what was going on.

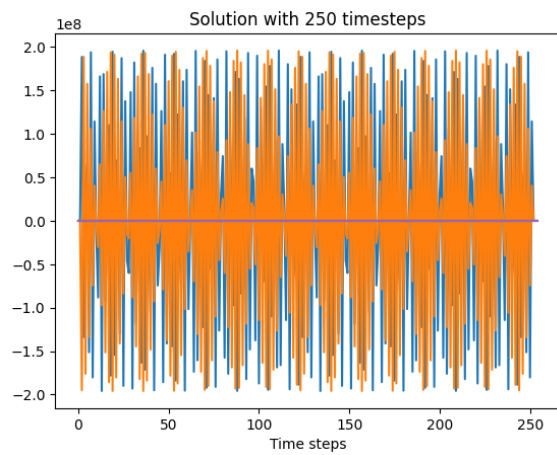
- These are the results once I removed the 'nan' values:



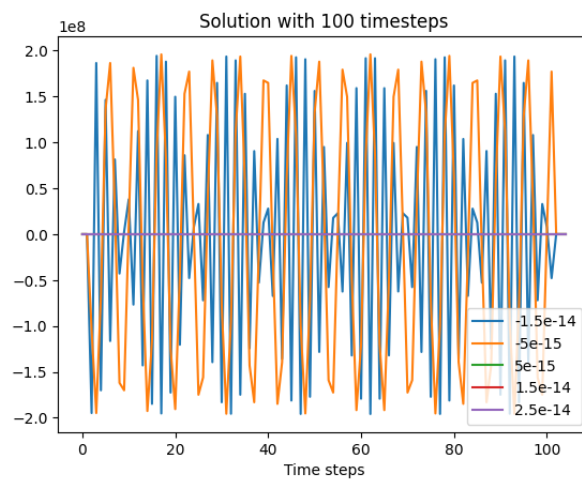
○



○



○



○

- Used a legend on the last one to see what was going on. Not too sure if I understand the values.
- Also, not sure what to use for my axis label.

- Finished off trying to animate but having issues. Will continue to work on it!

# 06/30/23 Meeting

```
# These Loops will go through the grid. We start at 1 so as to not calculate the
# value when the a time or spatial point is equal to 0.
index = 0
for n in range(1, (len(t) - 3)*1 + 0):
    t_sim = t[n] # in seconds

    # Initial conditions in time:
    E[n-1, 0] = E_0 * np.sin( angular_frequency * t[n-1] - wavenumber * x[0] )
    E[n-1, 1] = E_0 * np.sin( angular_frequency * t[n-1] - wavenumber * x[1] )

    # Instantenously defining the boundary conditions/values of ghost cells on the RIGHT:
    E[n, 0] = E_0 * np.sin( angular_frequency * t[n] - wavenumber * x[0] )
    E[n, 1] = E_0 * np.sin( angular_frequency * t[n] - wavenumber * x[1] )

    # We have no Laser on the Left, so boundary conditions are just 0
    E[n, -1] = 0
    E[n, -2] = 0

    # Initial conditions in time:
    E[1, 2:] = 0
    for i in range(2, len(x) - 3):
        E[n+1, i] = (r_const) * ( E[n, i+1] - 2*E[n, i] + E[n, i-1] ) - ( -2*E[n, i] + E[n-1, i] )

        # if the ratio is not bounded to from -1 to 1, then we have overshoot and are out of simulation physical bounds
        if abs(E[n+1, i]/E_0) > 1:
            print('stop')
            print('index', index, 'n+1', n+1, 'i', i, 'E[n+1, i]', E[n+1, i], 'E[n+1, i]/E_0', E[n+1, i]/E_0, (r_const) * (E[n, i]
            break

        index += 1

stop
index 2420 n+1 244 i 2 E[n+1, i] 195931975.39770085 E[n+1, i]/E_0 1.0002072100414854 -19443.842576992043 -195951419.24027783 99
46.015775949434
```

- 
- We finished the meeting with a problem where our ratio of  $|E[n+1, i]/E_0|$  was not strictly less than 1, if a number is greater than 1, it means we overshoot our physical boundaries. Everytime we lowered our CFL, the number of values that overshoot decreased, but we had to deal with increased computational time.
- Jack and I are going to look online/think about how to fix our spatial boundaries, because our time boundaries are good.
  - The reason our spatial boundaries are having issued is because this is a 2nd order
- Github link:  
[https://github.com/nickdavila/PFURO-2023/blob/main/Code/finite\\_difference\\_scheme\\_s.ipynb](https://github.com/nickdavila/PFURO-2023/blob/main/Code/finite_difference_scheme_s.ipynb)

## Investigation for boundary conditions:

- Paper [1] link:  
<https://asmedigitalcollection.asme.org/fluidsengineering/article-abstract/107/4/523/409883>
  - This paper talks about using second-order explicit finite-difference schemes for waterhammer analysis. Waterhammer is basically water crashing against a pipe and causing problems if a valve is suddenly closed. The paper mentions three different finite-difference schemes, but I mainly focused on the boundary conditions section. The author focused on two methods of including boundary conditions into the analysis.
    - **1. Using the characteristic equations**

- The characteristic equations can be used at the boundaries in conjunction with the conditions imposed by the boundary. The partial derivatives may be determined using the same finite-difference approximations as for the interior points. We just plug in the values we have for the boundaries. Now, the next  $+i$  may be determined from this equation by using a forward finite-difference approximation.
- 2. Extrapolating fluxes beyond the boundaries and then using extrapolated values (then using the extrapolated values in the difference scheme as values not at the boundaries)
  - I understood this as we calculate values from outside of the bounds and then use these outside of bound values in the difference scheme. Not too sure how this would work.
- Paper [2] link: <https://link.springer.com/article/10.1007/s10915-011-9531-1>
  - “A Fourth Order Accurate Finite Difference Scheme for the Elastic Wave Equation in Second Order Formulation” This one was harder to understand, but had a lot of discussion on boundaries and even used ghost cells like we do. Also had references to other papers that discussed boundaries.

# 07/07/2023 Meeting

The case for  $n_r = 1$ ,  $n_i = 0.1$

stop

index 17982  $n+1$  20  $i$  2  $E[n+1, i]$  (17155406.319050565+199574759.62439826j)  $\text{abs}(E[n+1, i])/\text{abs}(E_0)$  1.022560234000526

$E_0$  195891384.73574314

$(r_{\text{const\_complex}}) * (E[n, i+1] - 2 * E[n, i] + E[n, i-1])$  (-406372.605797941-1215926.982570957j)  
 $\text{abs}((r_{\text{const\_complex}}) * (E[n, i+1] - 2 * E[n, i] + E[n, i-1]))$  1282036.3183962929

$(-2 * E[n, i] + E[n-1, i])$  (-17561778.924848508-200790686.6069692j)  
 $\text{abs}(-2 * E[n, i] + E[n-1, i])$  201557227.3750147

$\text{abs}(r_{\text{const\_complex}})$  0.009900990099009903

$E[n, i+1]$  (18418536.3053871+56898007.1294487j)  $E[n, i]$   
(18550070.79522132+190637020.10283095j)  $E[n, i-1]$   
(2769256.96247854+195871809.68572965j)  $E[n-1, i]$   
(19538362.665594134+180483353.5986927j)  
0.30529620787061923 0.9777735145886043 0.9999999999999999 0.9267270407144016

$\cos\_term\_0\_t\_nm1$  3876835.7714562863  
 $\sin\_term\_0\_t\_nm1$  195853018.25115716  
 $\cos\_term\_1\_t\_nm1$  2646186.5362760285  
 $\sin\_term\_1\_t\_nm1$  195873510.9975368  
195891384.73574314  
195891384.73574314  
195891384.73574314  
195891384.7357431

The case for  $n_r = 1$ ,  $n_i = 0$

stop

index 17982  $n+1$  20  $i$  2  $E[n+1, i]$  (1556461.7303728838+198825071.1484092j)  $\text{abs}(E[n+1, i])/\text{abs}(E_0)$  1.015007186486191

$E_0$  195891384.73574314

$(r_{\text{const\_complex}}) * (E[n, i+1] - 2 * E[n, i] + E[n, i-1])$   
(2547.4075012792005-1247708.7153800316j)  
 $\text{abs}((r_{\text{const\_complex}}) * (E[n, i+1] - 2 * E[n, i] + E[n, i-1]))$  1247711.3158580658

$(-2 * E[n, i] + E[n-1, i])$  (-1553914.3228716047-200072779.86378923j)  
 $\text{abs}(-2 * E[n, i] + E[n-1, i])$  200078814.20117196

abs(r\_const\_complex) 0.010000000000000004

E[n, i+1] (311514.19959899725+60165544.36604696j) E[n, i]

(1413015.2059748087+190404112.79488987j) E[n, i-1]

(2769256.96247854+195871809.68572965j) E[n-1, i]

(1272116.0890780126+180735445.7259905j)

0.3071413829278647 0.9720149565026492 0.9999999999999999 0.9226537595968505

cos\_term\_0\_t\_nm1 3876835.7714562863

sin\_term\_0\_t\_nm1 195853018.25115716

cos\_term\_1\_t\_nm1 2646186.5362760285

sin\_term\_1\_t\_nm1 195873510.9975368

195891384.73574314

195891384.73574314

195891384.73574314

195891384.7357431

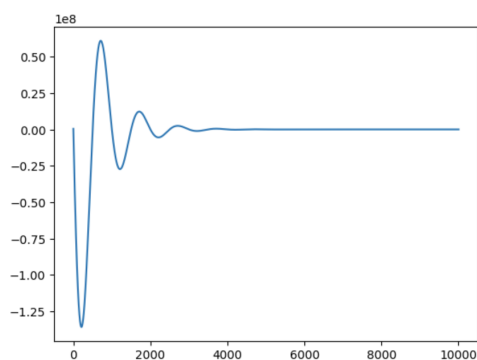
# Plots before IPF code:

## Plot of 10 cycles: Finding when function hits zero.

Process 1: I wasn't sure which way to do it. So I had two processes. One was simply running the code and seeing when the last value was closest to 0.0, which in our case is roughly when  $n_i = 0.255$ .

```
In [222]: # re-defining complex index of refraction (adding imaginary part to force damping using analytical solution)
n_r = 1
n_i = 0.255
complex_refractive_index = complex(n_r, n_i)
complex_wavenumber = (angular_frequency / speed_of_light) * complex_refractive_index

In [223]: plt.plot(np.exp(-complex_wavenumber.imag * x) * E_complex[int(9999 * lambda_mult), :].real)
Out[223]: [ <matplotlib.lines.Line2D at 0x26b6ac0d750>]
```



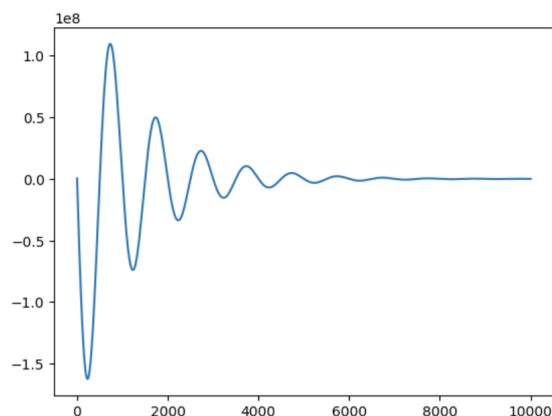
```
In [224]: sto = np.exp(-complex_wavenumber.imag * x) * E_complex.real

In [225]: sto[100001, 10001]
Out[225]: 0.0854228381453693
```

Process 2: My other process was visually inspecting the plot and seeing when it looked like the end was close to zero and not looking at any actual values. For this process the  $n_i$  value was roughly 0.125. **From Jack: this is nice.**

```
In [236]: # re-defining complex index of refraction (adding imaginary part to force damping using analytical solution)
n_r = 1
n_i = 0.125
complex_refractive_index = complex(n_r, n_i)
complex_wavenumber = (angular_frequency / speed_of_light) * complex_refractive_index

In [237]: plt.plot(np.exp(-complex_wavenumber.imag * x) * E_complex[int(9999 * lambda_mult), :].real)
Out[237]: [ <matplotlib.lines.Line2D at 0x26b6aee67d0>]
```





## Plot of S(x,t):

I am plotting Eq. (10) from the Google doc:

$$S(x, t) = -\frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \cdot \epsilon_0 E(x, t)^2 \quad \dots \text{Eq. (10)}$$

Using definitions from Eq (9), I rearranged Eq. (10) and substituted

$$k_i = \frac{\omega_L}{c} n_i = \frac{1}{cn} \frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \quad \dots \text{Eq. (9)}$$

$$\frac{\omega_p^2 \nu_c}{\omega_L^2 + \nu_c^2} \text{ for } -(k_i * cn)$$

So now my new Eq. (10) reads as

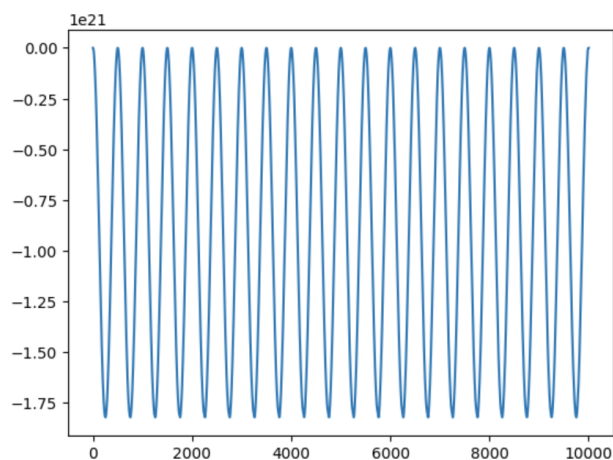
$$S(x, t) = -(k_i * cn) \cdot \epsilon_0 E(x, t)^2$$

Process: I wasn't totally sure how to fix my plots. My first try I just followed my modified Eq. (10) and multiplied it by the entire solution of our E\_complex.

$$S(x, t) = -(k_i * cn) \cdot \epsilon_0 E(x, t)^2 \quad (2)$$

```
In [12]: # S_xt = -(complex_wavenumber * speed_of_light * RefractiveIndex) * permittivity_free_space * (E_complex[int(9999 * Lambda_mult),
# S_xt = -(complex_wavenumber * speed_of_light * RefractiveIndex) * permittivity_free_space * (E_complex.real)**2
S_xt = -(complex_wavenumber * speed_of_light * RefractiveIndex) * permittivity_free_space * (E_complex_10_cycles_data.real)**2

In [16]: plt.plot(S_xt[int(9999 * lambda_mult), :])
plt.show()
```



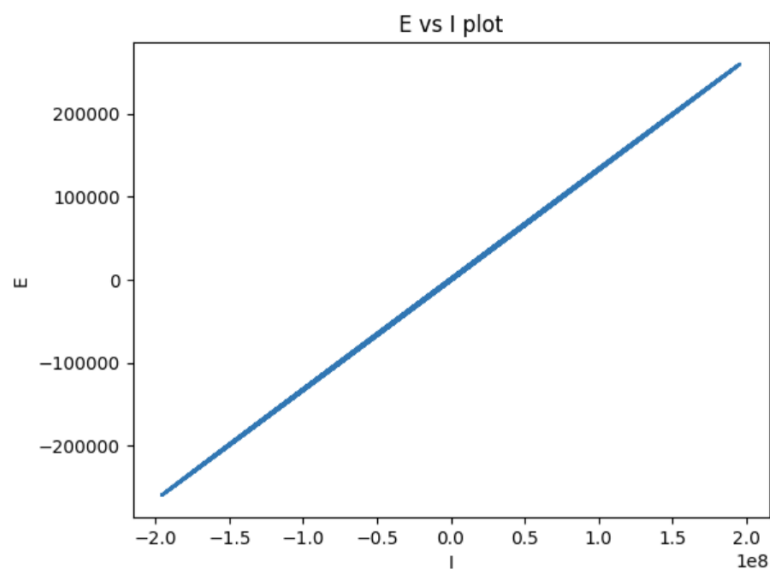
## Plot of E vs I:

Process: Since my other plots were clearly not correct, this one came out wrong as well. However I think with a quick fix they could all be correct.

$$S(x, t) = -2 * k_i * I, \text{ we can rearrange for } I \text{ (laser intensity) and get } I = \frac{S(x, t)}{-2 * k_i}$$

```
In [31]: laser_intensity = S_xt / (-2 * complex_wavenumber)

In [38]: plt.plot(E_complex_10_cycles_data[int(9999 * lambda_mult), :].real, laser_intensity)
plt.title("E vs I plot")
plt.xlabel("I")
plt.ylabel("E")
plt.show()
```



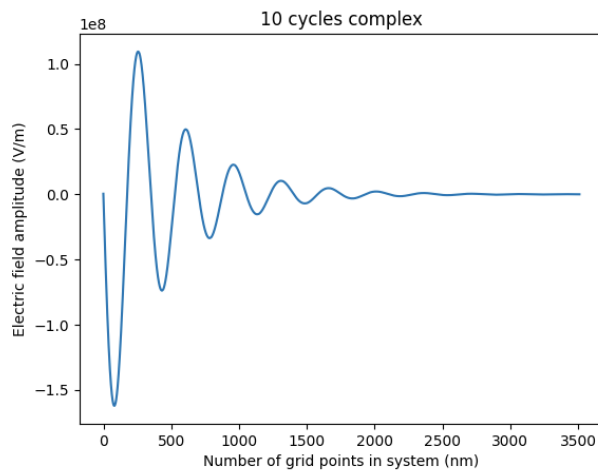
## Questions:

1. Do we still need to keep the code for the non-complex solution?
  - a. From Jack: I think no, because we model the laser absorption separately from the Fourier analysis.
2. What are good units for the x and y axis for the plots once we get them working properly?
  - a. From Jack: To be discussed in the meeting.

# 07/14/2023 Meeting

## Notes:

For Aluminum let's try to model 100 nm. 1 cycle because the wave immediately decays because SKIM length for Al is 20nm. and for CH let's try to model 1000 nm.



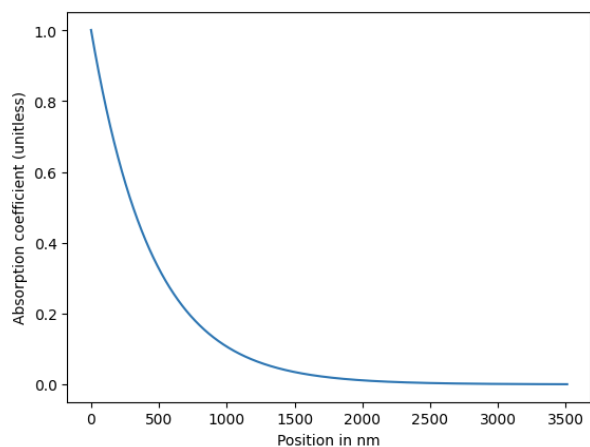
## NEXT TIME: Plot this curve for aluminum as well!

Useful for presentation. Talk about real and imaginary part, and talk about how some propagation from real part. Aluminum will have a very short skin length.

Will first model ultrafast for aluminum. And CH in much later time

This is absorption coefficient inside of the system. Useful parameter for presenting

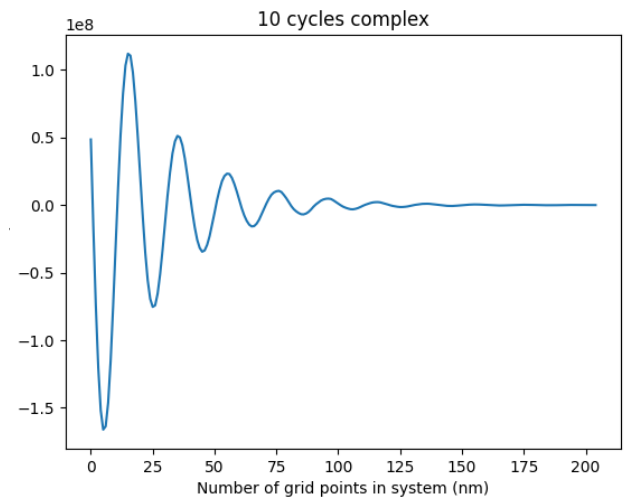
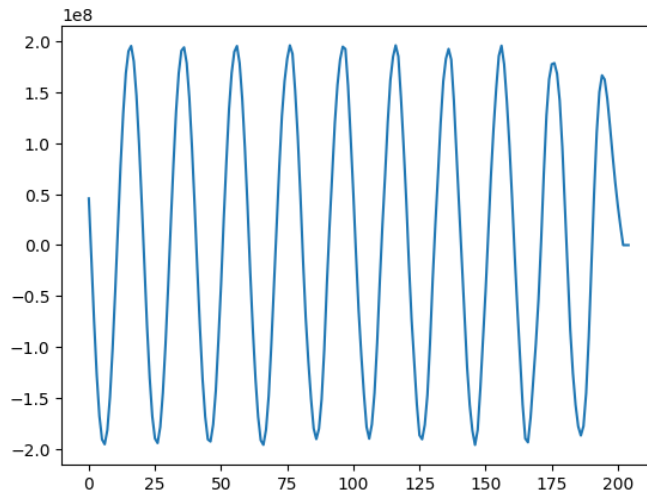
```
[ ]: # plt.plot(np.exp(-complex_wavenumber.imag * x))
plt.ylabel("Absorption coefficient (unitless)")
plt.xlabel("Position in nm")
plt.plot(E_complex_10_cycles_xdata/nm, np.exp(-complex_wavenumber.imag * E_complex_10_cycles_xdata))
plt.show()
```



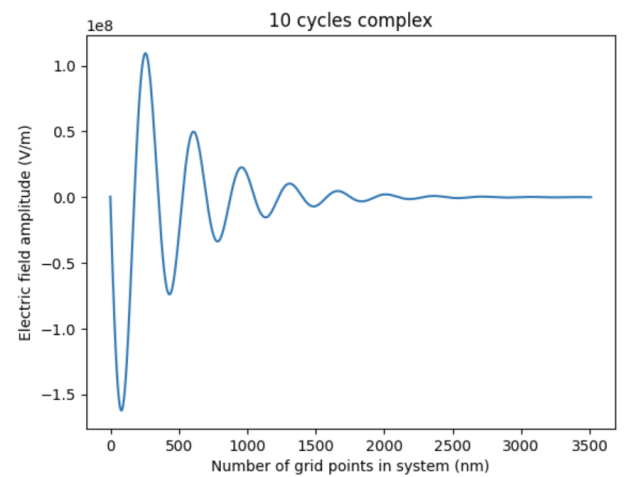
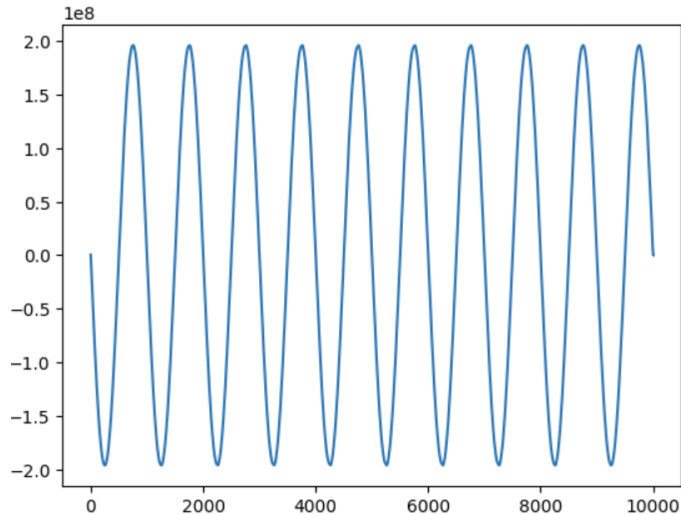
# 07/17/2023 Meeting:

## Pre-Meeting notes:

**CFL = 0.5,  $\Delta x = 0.05 \cdot \text{wavelength}$ . Runtime: Less than 1 second.**



**CFL = 0.1  $\Delta x = 0.001 \cdot \text{wavelength}$ . Runtime: About 21 minutes.**



### Plot of Laser intensity over space

```
In [97]: plt.plot(x, laser_intensity[400,:])  
plt.title("Plot of Laser Intensity '$I$' over space")  
plt.xlabel("Space (x)")  
plt.ylabel("Laser Intensity")  
plt.show()
```

