ColdxrayPy User's Guide

Installation Steps

Follow these steps to install the coldxraypy package.

1. Open the command line or Powershell (whichever can be used to call pip).
2. Type "pip install coldxrayPy".
3. Press the enter key.
4. Pip should notify of installation completion.

Importing in Python

After installation of the package, the coldxraypy package can be imported into code where it's to be used by following these steps.

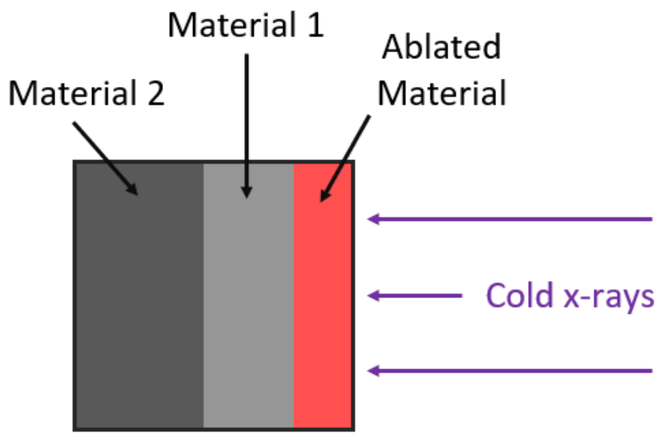
1. In the code, prior to calling methods and classes from this package, include the following code. "import coldxraypy".

An example is shown below.

```python
#Import the coldxrayPy package using the following code
import coldxrayPy
```

Purpose and Usage Intent

The purpose of this package is to model a high, sudden dosage of cold x-rays impinging on a material in space. The model is 1-dimensional, and x-rays are assumed to be traveling normal to the material's outer surface. The material to be analyzed may be composed of multiple layers of dissimilar materials (e.g. painted aluminum or a photovoltaic cell). The phenomenon captured in this package is the energy absorption of the x-rays by the material using the mass attenuation coefficients, ablation of material, then a transient thermal re-equilibration to determine how long the system takes to stabilize.



Units

Unless otherwise specified, all units for this package are in base metric (e.g., meter, second, Joule).

Example of Usage

To highlight how this package can be used, we’ll analyze a 3mm thick aluminum plate exposed to a cold x-ray dose as defined in M.S. Smith et al [1]. The code can analyze multi-layer configurations, but a single material composition is used here to compare to the analytical solution.
First, start by defining the required geometry (node quantity and thickness of each node), physical characteristics, and cold x-ray dosage parameters. The xraydb package is included to make it easy to obtain mass attenuation factors.

```
#Configure x-ray profile
x_flux = 1.7332e21 #[Total x-rays], Total x-rays from event
x_energy = 13.5e3 #[eV], Average energy of x-rays from event

#Pull material data for aluminum
al_mac = xraydb.mu_elam('Al', x_energy) #[cm^2/g]
al_mac = al_mac / 10 #[m^2/kg]

al_dens = xraydb.atomic_density('Al') #[g/cm^3]
al_dens = al_dens * 1000 #[kg/m^3]

#Configure the test points
e_qty = 100 #[-]
e_thick = 0.00001 #[m]

#Configure the material properties
al_k = 237 #[W/m*k] - not used for this verification
e_area = 0.1 #[m^2] - not used for this verification
ic_temp = 293.0 #[K] - not used for this verification
al_ab_temp = 1000.0 #[K] - placeholder, not used for this verification
al_c = 897.0 #[J/kg*C] - not used for this verification
```

Next, instantiate the material section and call the “add_matl” function to build the material layer by layer, starting from the surface that will be exposed to the x-rays first.

```
#Create a section of material representing 1mm thick Aluminum
matl_section = Section()
matl_section.add_matl(quantity=e_qty,
                      t_conductivity=al_k,
                      spec_heat=al_c,
                      cs_area=e_area,
                      ic_temp=ic_temp,
                      element_length=e_thick,
                      ablation_temp=al_ab_temp,
                      rad_abs_coefficient=al_mac,
                      density=al_dens)
```

Next, call the “prop_xray_energy” method and use the total x-ray energy dosage [eV] as the input. This will use a first order difference method to determine how much x-ray energy is absorbed by each element and transport the remaining energy into the subsequent element. This process will be repeated from the forward element until the final, back element is computed.

```
#Propagate the xrays through the material and return the amount
#remaining at each element to compare to the analytical result
matl_section.prop_xray_energy(x_flux*x_energy)
```

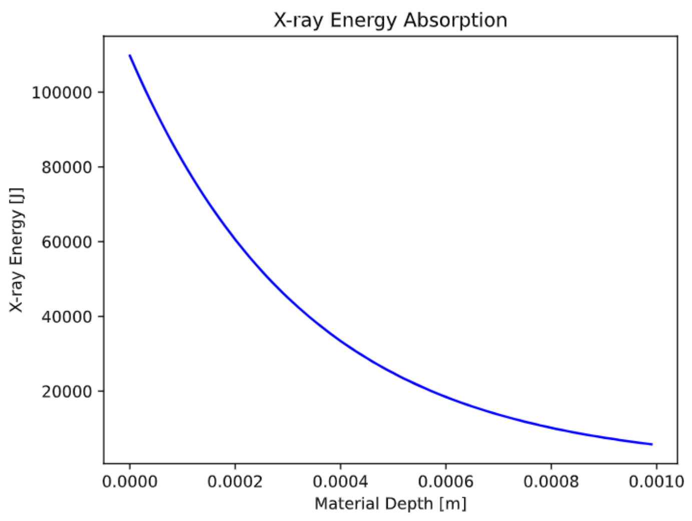
Next, call the "eval_ablation" method. This method will look at each element's temperature post x-ray event and assign a value identifying whether the element has been ablated.

```python
#Evaluate the elements for ablation
matl_section.eval_ablation()
```

The analysis is now complete and results can be retrieved by calling the methods like the ones shown below. Consult the class and function descriptions section for a complete list of available methods. The elements are stored as a list in the Section class and can be accessed directly if needed.

```python
#Retrieve the results of the simulation

#Returns a list of the depth from the outermost surface for each element [m].
#Helpful in generating the x-axis for many plots.
node_depths = matl_section.get_node_depths()

#Returns a list of the energy absorbed by each element [J].
#Expressed in Joules to facilitate easier application to thermal calculations.
absorbed_energys = matl_section.get_absorbed_energy()

#Returns a list of energy that is transmitted through at the start of each element [eV]
#Evaluated at the start of each element to better match analytical solution where
#I(x=0) = I0
non_absorbed_energys = matl_section.get_rem_energy()

#Returns a list containing each element's temperature right after x-ray exposure [K].
temps_post_xrays = matl_section.get_temperatures()
```

The results of energy absorbed should match the plot shown below.



Class and Function Descriptions

Class: InternalElement

This represents a single element of material and is used primarily to store material parameters (e.g., thermal conductivity) and state (e.g., temperature). The user shouldn't have to interact with this directly unless provided Section class accessors are insufficient.

Method: constructor

```
def __init__(self,
             t_conductivity,
             spec_heat,
             cs_area,
             ic_temp,
             element_length,
             ablation_temp,
             rad_abs_coefficient,
             density):
```

Inputs

- t_conductivity <float>: Thermal conductivity of the element's material
- spec_heat <float>: Specific heat of the element's material
- cs_area <float>: The cross-sectional area of the element
- ic_temp <float>: The initial condition for temperature of the element
- element_length <float>: The length of the element (colinear w/ x-ray direction)
- ablation_temp <float>: The temperature at which the material ablates
- rad_abs_coefficient <float>: The mass attenuation factor of the material
- density <float>: The density of the material

Outputs

- None

Method: set_ics

```
def set_ics(self, initial_temp):
```

Inputs

- initial_temp <float>: The initial condition for temperature of the element

Outputs

- None

Class: Section

This class is used to represent a cross-section of one or more materials which will absorb x-rays and undergo thermal changes. This is what the user should interact with primarily to set up the problem, execute the simulation, and retrieve results.

Method: constructor

Primarily used to configure internal state and initialize objects. No input required from the user at this step.

```
def __init__(self):
```

Inputs

- None

Outputs

- None

Method: add_matl

This function is used to add a new thickness of material. Call this function multiple times to add additional material layers. Subsequent calls will add material to the back (away from initial x-ray impingement surface)

```
def add_matl(self,
             quantity,
             t_conductivity,
             spec_heat,
             cs_area,
             ic_temp,
             element_length,
             ablation_temp,
             rad_abs_coefficient,
             density):
```

Inputs

- quantity <int>: Number of elements in the section of material
- t_conductivity <float>: The material's thermal conductivity
- spec_heat <float>: The material's specific heat
- cs_area <float>: The cross sectional area for the 1D material section (normal to x-rays)
- ic_temp <float>: Initial temperature of the material
- element_length <float>: The length of each element in the material layer
- ablation_temp <float>: The temperature at which the material will ablate
- rad_abs_coefficient <float>: The mass attenuation factor of the material
- density <float>: The density of the material

Outputs

- None

Method: prop_xray_energy

After all the layers of materials have been added, call this function to pass x-ray energy through the material section to calculate how much each element absorbs and the accompanying temperature change.

```
def prop_xray_energy(self, tot_x_ray_energy):
```

Inputs

- tot_x_ray_energy <float>: Energy of all the x-rays to pass through the material's section [eV]

Outputs

- None

Method: get_absorbed_energy

After passing the x-ray energy through the material, use this function to get a list of the energy absorbed by each element [J].

```
def get_absorbed_energy(self):
```

Inputs

- None

Outputs

- Absorbed energy of each element <List of floats>

Method: get_rem_energy

After passing the x-ray energy through the material, use this function to get a list of the energy that passes through each previous element [eV]. Previous elements are reported so that numerical solution matches analytical solution better at depth = 0.

```
def get_rem_energy(self):
```

Inputs

- None

Outputs

- Remaining energy passing through each element <List of floats>

Method: get_temperatures

After passing the x-ray energy through the material, use this function to get a list of the temperature for each element [K].

```
def get_temperatures(self):
```

Inputs

- None

Outputs

- Temperature of each element <List of floats>

Method: get_node_depths
After configuring the material section, use this function to return a list of the depth for each element. This is useful when plotting performance and parameters of the section as a function of distance [m]. The depth is measured from the outermost surface (i.e., where the radiation will impinge first).

```
def get_node_depths(self):
```

Inputs

- None

Outputs

- Depth of each element <List of floats>

Method: get_macs
After configuring the material section, use this function to return a list of the mass attenuation coefficient for each element.

```
def get_macs(self):
```

Inputs

- None

Outputs

- Mass attenuation coefficient of each element <List of floats>

Method: get_densities
After configuring the material section, use this function to return a list of the density [kg/m^3]

```
def get_densities(self):
```

Inputs

- None

Outputs

- Density of each element <List of floats>

Method: eval_ablation
After calling the prop_xray_energy method, call this method to determine which elements will be lost to ablation from extreme heats.

```
def eval_ablation(self):
```

Inputs

- None

Outputs

- Whether or not the element is ablated <List of booleans>

Method: transient_thermal_analysis

After the ablation has occurred, call this method to equilibrate the plate to it's steady state temperature

```
def transient_thermal_analysis(self, L, T0, Tinf, rho, c, k, dx, dt, t_max, emissivity, solar_flux):
```

Inputs:

- L: length of remaining plate in m
- T0: Initial temperature of the plate in K
- Tinf: Temperature black body space in K
- rho: Density of material kg/m^3
- c: Specific heat capacity of material in J/kg*K
- k: Thermal conductivity in W/m*k
- dx: Spatial step in m
- dt: Time step in s
- t_max: Total time in s
- emissivity: Emissivity of material
- solar_flux: Low earth orbit solar flux in W/m^2

Outputs:

- Steady state temperature at the midpoint of the plate.

References

[1] M. S. Smith, T. J. Burns, and J. O. Johnson, “Shield Optimization Program, Part V: A Hydrodynamic Comparison Using HULL and PUFF-TFT for a One-Dimensional Aluminum Slab,” Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Jun. 1989.