Coverage for **diffusion2d.py** : 60%

84 statements | 50 run | 34 missing | 0 excluded

```python
"""
Solving the two-dimensional diffusion equation

Example acquired from https://scipython.com/book/chapter-7-matplotlib/examples/the-two-dimensional-diffusion-equation/
"""

import numpy as np
import matplotlib.pyplot as plt


class SolveDiffusion2D:
    def __init__(self):
        """
        Constructor of class SolveDiffusion2D
        """
        # plate size, mm
        self.w = None
        self.h = None

        # intervals in x-, y- directions, mm
        self.dx = None
        self.dy = None

        # Number of discrete mesh points in X and Y directions
        self.nx = None
        self.ny = None

        # Thermal diffusivity of steel, mm^2/s
        self.D = None

        # Initial cold temperature of square domain
        self.T_cold = None

        # Initial hot temperature of circular disc at the center
        self.T_hot = None

        # Timestep
        self.dt = None

    def initialize_domain(self, w=10., h=10., dx=0.1, dy=0.1):
        # Check if the parameters are floats
        assert isinstance(w, float), 'w must be a float'
        assert isinstance(h, float), 'h must be a float'
        assert isinstance(dx, float), 'dx must be a float'
        assert isinstance(dy, float), 'dy must be a float'

        self.w = w
        self.h = h
        self.dx = dx
        self.dy = dy
        self.nx = int(w / dx)
        self.ny = int(h / dy)

    def initialize_physical_parameters(self, d=4., T_cold=300., T_hot=700.):
        assert isinstance(d, float), 'd must be a float'
        assert isinstance(T_cold, float), 'T_cold must be a float'
        assert isinstance(T_hot, float), 'T_hot must be a float'

        self.D = d
        self.T_cold = T_cold
        self.T_hot = T_hot

        # Computing a stable time step
        dx2, dy2 = self.dx * self.dx, self.dy * self.dy
        self.dt = dx2 * dy2 / (2 * self.D * (dx2 + dy2))

        print("dt = {}".format(self.dt))
```

```python

    def set_initial_condition(self):
        u = self.T_cold * np.ones((self.nx, self.ny))

        # Initial conditions - circle of radius r centred at (cx,cy) (mm)
        r, cx, cy = 2, 5, 5
        r2 = r ** 2
        for i in range(self.nx):
            for j in range(self.ny):
                p2 = (i * self.dx - cx) ** 2 + (j * self.dy - cy) ** 2
                if p2 < r2:
                    u[i, j] = self.T_hot

        return u.copy()

    def do_timestep(self, u_nm1):
        u = u_nm1.copy()

        dx2 = self.dx * self.dx
        dy2 = self.dy * self.dy

        # Propagate with forward-difference in time, central-difference in space
        u[1:-1, 1:-1] = u_nm1[1:-1, 1:-1] + self.D * self.dt * (
                (u_nm1[2:, 1:-1] - 2 * u_nm1[1:-1, 1:-1] + u_nm1[:-2, 1:-1]) / dx2
                + (u_nm1[1:-1, 2:] - 2 * u_nm1[1:-1, 1:-1] + u_nm1[1:-1, :-2]) / dy2)

        return u.copy()

    def create_figure(self, fig, u, n, fignum):
        fignum += 1
        ax = fig.add_subplot(220 + fignum)
        im = ax.imshow(u.copy(), cmap=plt.get_cmap('hot'), vmin=self.T_cold, vmax=self.T_hot)
        ax.set_axis_off()
        ax.set_title('{:.1f} ms'.format(n * self.dt * 1000))

        return fignum, im


def output_figure(fig, im):
    fig.subplots_adjust(right=0.85)
    cbar_ax = fig.add_axes([0.9, 0.15, 0.03, 0.7])
    cbar_ax.set_xlabel('$T$ / K', labelpad=20)
    fig.colorbar(im, cax=cbar_ax)
    plt.show()


def main(solver: SolveDiffusion2D = None):
    if solver is None:
        DiffusionSolver = SolveDiffusion2D()
        DiffusionSolver.initialize_domain()
        DiffusionSolver.initialize_physical_parameters()
    else:
        DiffusionSolver = solver

    u0 = DiffusionSolver.set_initial_condition()

    # Number of timesteps
    nsteps = 101

    # Output 4 figures at these timesteps
    n_output = [0, 10, 50, 100]

    fig_counter = 0
    fig = plt.figure()

    im = None

    # Time loop
    for n in range(nsteps):
        u = DiffusionSolver.do_timestep(u0)

        # Create figure
```

```
        if n in n_output:
            fig_counter, im = DiffusionSolver.create_figure(fig, u, n, fig_counter)

        u0 = u.copy()

    # Plot output figures
    output_figure(fig, im)


if __name__ == "__main__":
    main()
```

« index   coverage.py v5.5, created at 2022-01-19 19:18 +0000

normal