1 Parabolic PDEs

1.1 `diff_adv_2D.py`

Calculates the solution to the two-dimensional diffusion-advection/heat-convection equation

$$\frac{\partial u}{\partial t} = D(x, y)\ \nabla^2 u - \hat{\mathbf{v}} \cdot \nabla u + q(x, y, t),$$

using cartesian coordinates on a uniform mesh, on the domain Ω bound by $x \in [0, L_x]$ and $y \in [0, L_y]$, where

$$\hat{\mathbf{v}} = v_x \hat{\imath} + v_y \hat{\jmath}$$

describes the direction of advection/convection, $D(x, y)$ is the diffusivity/conductivity and $q(x, y, t)$ is a source/sink term. The program assumes Robin boundary conditions

$$a\ u(x, y, t) + b\ \frac{\partial u}{\partial n} = g(x, y)$$

where a and b are real scalars, $g(x, y)$ is an arbitrary function on the boundary and ∂n denotes differentiation in the direction of a normal to the boundary. The initial condition $u_0(x, y) = u(x, y, 0)$ can be any real valued function. The solution is calculated using the finite difference method.

1.2 `mc_stefan.py`

Calculates the solution to the classical Stefan problem

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \alpha\ \frac{\partial^2 u}{\partial x^2}, \qquad 0 < x < s(t), \quad t > 0 \\
u(0, t) &= 1, \\
u(x, 0) &= 0, \\
u(s(t), t) &= T_m \\
s(0) &= 0 \\
L\ \rho\ \frac{ds}{dt} &= k\ \left.\frac{\partial u}{\partial x}\right|_{x=s(t)}
\end{aligned}$$

using a monte carlo, where $u(x, t)$ is the temperature, $s(t)$ is the position of the moving boundary, α is the thermal diffusivity, T_m is the melting temperature, L is the latent heat, ρ is the density and k is the thermal conductivity. The solution is calculated on the domain Ω bound by $x \in [0, L_x]$ and $t > 0$. The code models the free boundary using the method proposed by Stoor [**stoor**].

The answer is compared to the analytical solution derived by Neumann[1]

$$u(x,t) = 1 - \frac{\mathrm{erf}\left(\frac{x}{2\sqrt{t}}\right)}{\mathrm{erf}\,(\lambda)},$$
$$s(t) = 2\lambda\sqrt{t},$$

where λ satisfies

$$\beta\sqrt{\pi}\lambda \mathrm{e}^{\lambda^2}\mathrm{erf}\,(\lambda) = 1, \tag{1}$$

where β is the Stefan number.

2 Hyperbolic PDEs

2.1 `wave_2D.py`

Calculates the solution to the two-dimensional acoustic wave equation

$$\frac{\partial^2 p}{\partial t^2} + \nu(x,y)\,\frac{\partial p}{\partial t} = c^2\,\nabla^2 u + q(x,y,t),$$

using cartesian coordinates on a uniform mesh, on the domain Ω bound by $x \in [0, L_x]$ and $y \in [0, L_y]$, where $\nu(x,y)$ is a damping term, c is the speed of sound and $q(x,y,t)$ is a source/sink term. The program assumes Robin boundary conditions

$$a\;u(x,y,t) + b\,\frac{\partial u}{\partial n} = g(x,y),$$

where a and b are real scalars, $g(x,y)$ is an arbitrary function on the boundary and ∂n denotes differentiation in the direction of a normal to the boundary. The initial pressure $p_0(x,y) = p(x,y,0)$ can be any real valued function. The solution is calculated using the finite difference method.

2.2 `wave_2D_PML.py`

Calculates the solution to the two-dimensional acoustic wave equation

$$\frac{\partial \hat{\mathbf{v}}}{\partial t} = -\frac{1}{\rho}\nabla p,$$
$$\frac{\partial p}{\partial t} + \nu(x,y)\;p = -c^2\rho\nabla\cdot\hat{\mathbf{v}} + q(x,y,t),$$

where $\hat{\mathbf{v}}(x,y,t)$ describes the velocity at each point in the mesh and $p(x,y,t)$ describes the pressure at each point in the mesh. Solution is calcualted using cartesian coordinates and a uniform mesh, on the domain Ω bound by $x \in [0, L_x]$ and $y \in [0, L_y]$, where $\nu(x,y)$ is a damping term, c is the speed of sound

[1] A similarity solution derived using the variable $\xi = \frac{x}{\sqrt{t}}$.

and $q(x, y, t)$ is a source/sink term. The program assumes Robin boundary conditions

$$a\ u(x, y, t) + b\ \frac{\partial u}{\partial n} = g(x, y),$$

where a and b are real scalars, $g(x, y)$ is an arbitrary function on the boundary and ∂n denotes differentiation in the direction of a normal to the boundary. The initial pressure $p_0(x, y) = p(x, y, 0)$ can be any real valued function. The solution is calculated using the finite difference method. Absorbing boundary conditions can optionally be turned on or off to simulate far-field conditions. Absorbing boundary conditions are implemented by stretching the coordinates of the governing equations into the complex domain in Fourier-transform-space using the method proposed by Berenger [**Berenger1994**].

3 Elliptical PDEs

3.1 `helmholtz.py`

Calculates the solution to the nonhomogenous Helmholtz equation

$$(\nabla^2 + k(x, y)^2)\ f(x, y, t) = \psi(x, y),$$

using cartesian coordinates on a uniform mesh, on the domain Ω bound by $x \in [0, L_x]$ and $y \in [0, L_y]$, where $k(x, y)$ and $\psi(x, y)$ are real valued functions. The program assumes Robin boundary conditions

$$a\ u(x, y, t) + b\ \frac{\partial u}{\partial n} = g(x, y),$$

where a and b are real scalars, $g(x, y)$ is an arbitrary function on the boundary and ∂n denotes differentiation in the direction of a normal to the boundary. The solution is calculated using the finite difference method.

`keywords: Laplace's equation, Poisson's equation.`

4 References