FD1D CODE FAMILY

MALGORZATA PESZYNSKA, PRIMARY DEVELOPER,

AND MICHAEL RENNE, CODE AND DOCUMENTATION EDITOR

---

1. Introduction

This document describes a family of templates for setting up the solution of a few application problems in 1d using mass-conservative cell-centered finite differences, and implicit time stepping.

1.1. Credits and use.

The code and documentation are made publicly available through the `GitHub` platform at [1]. If you use this code, we ask that you acknowledge it according to the rules of Creative Commons CC BY-NC-ND 4.0.

1.2. Contents.

(1) Section 2 describes `fd1d.m`
(2) Section 3 describes `fd1d_heat.m`
(3) Section 4 describes `compressible_single_phase.m`

Remark 1. *The MATLAB code provided in the* `fd1d` *family templates is written in a "flat" way which allows for easy portability, debugging, and experimentation. In particular, (i) we do not attempt to improve efficiency by vectorization, (ii) we write loops explicitly, and (iii) we use standard solvers. (Efficient simulation requires that the steps (i-iii) and possibly others are implemented.) Some data is hard-coded and some data is passed on as parameters. See preamble and code for how to use data and flags.*

2. fd1d.m

This code solves the two-point boundary value problem (Poisson equation in 1d).

Date: 2005-2017.
Support by NSF is gratefully acknowledged.

2.1. **Problem.**

$$-(ku')' = f, \quad x \in (a, b) \tag{1a}$$

with appropriate boundary conditions of either Dirichlet type

$$u(a) = u_a, u(b) = u_b \tag{1b}$$

or Neumann type where (1b) is replaced by $u'(a) = u'_a, u'(b) = u'_b$. In the code, we prefer to use the flux condition for better convergence.

Here $k = k(x)$ is the coefficient of conductivity, permeability, or diffusivity (depending on the application). The term $f = f(x)$ is the source/sink term.

2.2. **Data.** The user must provide or hard-code the values as in Table 1.

TABLE 1. Data for model in `fd1d`

a, b	interval	parameter
u_a, u_b	values of boundary conditions	parameter
k	permeability	hard-coded
f	source/sink term	hard-coded
nx	number of sub-intervals	parameter
flag1/2	type of boundary conditions	parameter

2.3. **Discretization.** The numerical method for (1) is a cell-centered finite difference method (CCFD).

2.4. **Solver.** The code uses the MATLAB "backslash" \ operator for the linear solver.

2.5. **Code preamble and parameters.**

```
function [u,h,xx,dirval1,dirval2,qflux1,qflux2] = ...
    fd1d (nx,a,b,val1,val2,flag1,flag2,flagf)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FD1D
%% fd1d(nx,a,b,val1,val2,flag1,flag2,flagf)
%% 1d FD cell-centered solution to Poisson equation on (a,b)
%% user must code permfun, rhsfun, exfun (at bottom of code)
%% nx: Number of sub-intervals of (a,b).
%% val1, val2 are values of bdary data on left and right endpoints
%% flag1/2 == 0: val1/2 is a Dirichlet value
%% flag1/2 == -1: val1/2 is a Neumann value
%% flagf == 0: Set rhs = 0 (optional argument)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Outputs:
%% u is the numerical solution at xx, the cell centers.
%% h is the maximum step size between nodes
```

```
%% dirval1/2 are the dirichlet data
%% qflux1/2 are the boundary fluxes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.6. **Hard-coded parameters.** The user hard-codes the permeability coefficients "permfun," the right-hand side of the Poisson equation, f, "rhsfun," and the exact solution, u, "exfun."

If the exact solution is known, it can be plotted and compared to the numerical solution.

2.7. **Examples.**

Example 2.7.1. $(0, 1)$ *split into 1000 sub-intervals with Dirichlet boundary conditions.*

```
>> fd1d(1000,0,1,0,0,0,0)
```

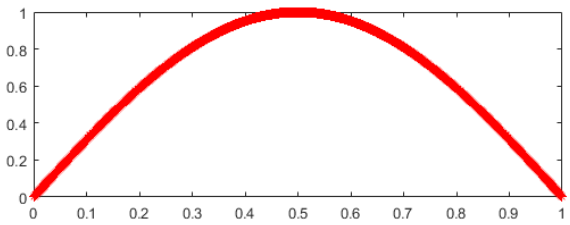

FIGURE 1. Plot of both the numerical and exact solution for Example 2.7.1

Example 2.7.2. $(0, 1)$ *split into 30 sub-intervals, with Dirichlet boundary conditions at left, and Neumann boundary conditions at right.*

```
>> fd1d(30,0,1,0,-pi,0,-1)
```

3. FD1D_HEAT.M

This code solves the initial value two-boundary point diffusion equation in 1d (heat-equation).

3.1. **Problem.**

$$\phi \frac{\partial c}{\partial t} - \nabla \cdot (k \nabla c) = f, \quad x \in (0, 1) \tag{2a}$$

with initial condition

$$c(x, 0) = g(x), \tag{2b}$$

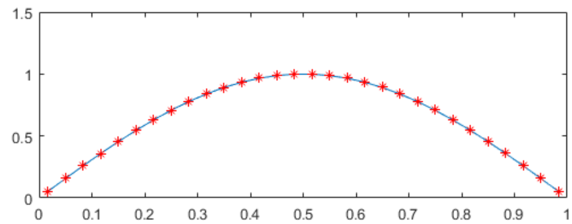

FIGURE 2. Plot of both the numerical and exact solution for Example 2.7.2

and boundary conditions of either Dirichlet type or Neumann type for $t \in (t_1, t_2)$.

Here, $\phi = \phi(x)$ is the porosity, $k = k(x)$ is the coefficient of conductivity, permeability, or diffusivity (depending on the application), and $f = f(x,t)$ is the source/sink term.

3.2. **Data.** The user must provide or hard-code the values as in Table 2.

TABLE 2. Data for `fd1d_heat`

t_1, t_2	time interval	hard-coded as $(0, 1)$
ϕ	porosity	hard-coded
k	permeability	hard-coded
f	source/sink term	hard-coded
nx	number of sub-intervals	parameter
dt	time step	parameter
bdaryflag	type of boundary conditions	parameter
outflag	compute error	parameter

3.3. **Discretization.** The numerical method for (2) is a cell-centered finite difference method (CCFD).

3.4. **Solver.** The code uses the MATLAB "backslash" \ operator for the linear solver.

3.5. **Code preamble and parameters.**

```
function [xplot,nsols] = fd1d_heat (nx,dt,bdaryflag,outflag)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% fd1d_heat (nx,dt,bdaryflag,outflag)
%% 1D FD cell-centered solution to heat equation on (0,1)
%% user must code the functions permfun, porfun, rhsfunt,
%%     initfun, exfun, dexfun INSIDE the code (below)
%% nx is the number of sub-intervals of (0,1)
%% dt is the time step
```

```
%% dt == 0: steady state solution only
%% bdaryflag == 0: Dirichlet (nonhomogeneous) BC on both ends
%% bdaryflag != 0: Neumann (nonhomogeneous) BC on both ends
%% note: user must code exfun/dexfun to deliver the BC values
%% outflag == 0: only plot solution in time
%% outflag != 0: consider exact solution, compute error etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Outputs:
%% xplot : the cell-centers
%% nsols : the numerical solution
%%   the code will also plot the numerical solution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3.6. **Hard-coded parameters.** The user hard-codes the porosity "`porfun`," permeability coefficients "`permfun`," the right-hand side of the heat equation, f, "`rhsfunt`," the initial condition "`initfun`," the exact solution, c, "`exfun`," and c_x, "`dexfun`."

Note: the user must code `exfun` and `dexfun` to deliver the boundary condition values.

3.7. **Examples.**

Example 3.7.1. $(0,1)$ *split into 100 sub-intervals, with 10 time steps, Dirichlet boundary conditions, and computed error.*

```
>> [xx,yy] = fd1d_heat(100,0.1,0,1)
```

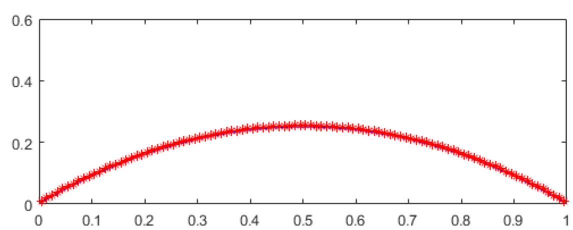

FIGURE 3. Plot of both the numerical and exact solution, at the final time step, for Example 3.7.1

Example 3.7.2. $(0,1)$ *split into 100 sub-intervals, with 10 time steps, Neumann boundary conditions, and computed error.*

```
>> [xx,yy] = fd1d_heat(100,0.1,1,1)
```

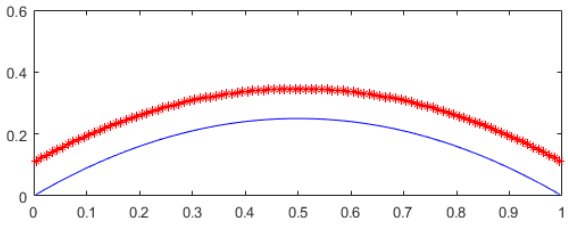

FIGURE 4. Plot of both the numerical and exact solution, at the final time step, for Example 3.7.2

4. COMPRESSIBLE_SINGLE_PHASE.M

This code solves slightly compressible Darcy fluid flow problem in subsurface.

4.1. **Problem.**

$$\phi\frac{\partial\rho}{\partial t} - \nabla\cdot\left(\frac{K}{\mu}\rho(\nabla p - G\rho\nabla D)\right) = f(x), \tag{3a}$$

with appropriate boundary and initial boundary conditions. Here K, μ, D, G, ϕ are data supplied by the user or hard-coded in the code.

The problem is solved assuming the fluid is slightly compressible

$$\rho(p) = \rho_{ref}\exp(c(p - p_{ref})) \tag{3b}$$

The values c, p_{ref}, ρ_{ref} are data provided by the user. Additionally, the user must supply boundary conditions in a manner similar to `fd1d.m`.

4.2. **Data.** The user must provide or hard-code the values as in Table 3.

TABLE 3. Data for single phase compressible code

a, b	spatial interval	hard-coded as $(0, 0.6)$
t_1, t_2	time interval	parameter
K	permeability	hard-coded
μ	viscosity	hard-coded
D	depth	hard-coded with `dfac`
G	gravity constant	hard-coded
ϕ	porosity	hard-coded
c	fluid compressibility	hard-coded
p_{ref}	reference pressure	hard-coded
ρ_{ref}	reference density	hard-coded
f	source/sink	$f = 0$
nx	number of subintervals of (a,b)	parameter
dfac	angle	parameter
dt0	time step	parameter
bdary1/2	type of boundary conditions	parameter
val1/2	boundary values	parameter

4.3. **Discretization.** The numerical method for equation (3) is cell-centered finite difference method (CCFD), with the particular discretization details following [2]. The time discretization is fully implicit, but the nonlinearity can be resolved in a sequential way, if desired, by setting `implicit_explicit` to 'explicit'.

The user must supply the desired spatial discretization parameter and the desired time step.

4.4. **Solver.** The code uses the MATLAB "backslash" \ operator for the linear solver.

Nonlinearities are solved with a Newton solver; the code provides a template for clearly coded handling of adaptive time stepping depending on the success of Newton iteration, use of absolute and relative tolerance, and so on.

4.5. **Code preamble and parameters.**

```
function compressible_single_phase (nx,dfac,dt0,t1,t2,...
    bdary1,bdary2,val1,val2,...
    implicit_explicit,...
    pmin,pmax, ...
    ifpause)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 1D FD cell-centered solution to compressible flow equation
%% PRESSURE UNKNOWN
%% <nx, dfac, dt0, t1, t2>: physical domain
%%    nx: Number of sub-intervals of (a,b) (hard-coded(0,0.6))
%%    dfac: sin(angle), angle=0 horizontal, angle= pi/2 vertical
%%    dt0: time step over the time interval (t1,t2).
%% dfac == 0: depth = 0,     dfac == 1: depth = x;
%% bdary1/2 == 0: Dirichlet (nonhomogeneous) values val1/2
%% bdary1/2 == 1: Neumann flux values val1/2
%% implicit_explicit == 'explicit': Solve sequentially
%% <pmin, pmax, ifpause>: parameters for plotting
%% Note: This is hard-coded with SI units, K with [m^2],
%%        pressure with [Pa].
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Outputs:
%%   the code will plot the numerical solution over time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.6. **Hard-coded parameters.** The user hard-codes permeability K, porosity ϕ, viscosity μ, depth D, gravity G, fluid compressibility c, reference pressure p_{ref}, and reference density ρ_{ref}.

Both permeability and porosity are hard-coded as constants, with the option of hard-coding them as functions, `permfun` and `porfun`, respectively (at the bottom of the code). Furthermore, depth is hard-coded with `dfac`.

4.6.1. *Units.* We use SI units here, so, in particular, K has units of [m^2], and pressures come out as [Pa].

4.7. **Examples.**

Example 4.7.1. *Vertical flow example, no flow at the bottom*

```
>> compressible_single_phase(10,1,0.1,0,1,0,1,2000,0,0,0,5e4,1)
```

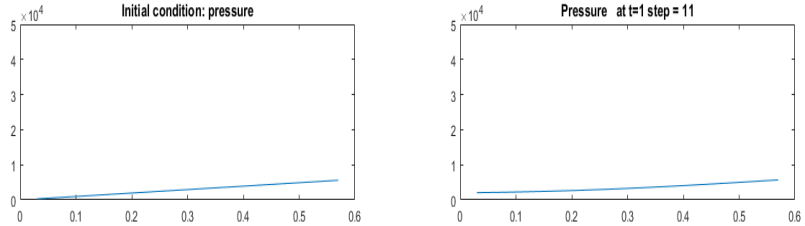

FIGURE 5. Plots of the initial pressure and the numerical solution at the last time step for Example 4.7.1.

Example 4.7.2. *Horizontal case, Dirichlet conditions*

```
>> compressible_single_phase(10,0,0.1,0,1,0,0,0,1000,0,0,1e3,1)
```

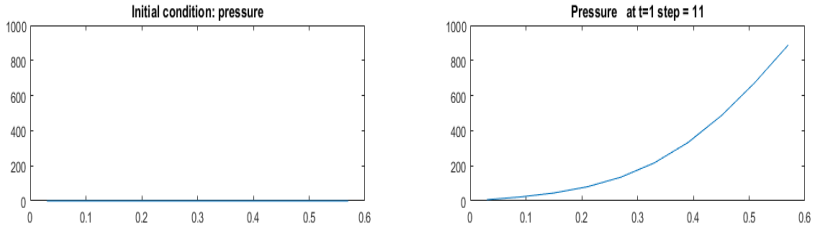

FIGURE 6. Plots of the initial pressure and the numerical solution at the last time step for Example 4.7.2.

REFERENCES

[1] M. Peszynska (Primary developper). fd1d code family, 2005-2017. URL `https://github.com/mpesz/fd1d`. retrieved 2017.

[2] M. Peszynska, E. Jenkins, and M. F. Wheeler. Boundary conditions for fully implicit two-phase flow model. In Xiaobing Feng and Tim P. Schulze, editors, *Recent Advances in Numerical Methods for Partial Differential Equations and Applications*, volume 306 of *Contemporary Mathematics Series*, pages 85–106. American Mathematical Society, 2002. doi: http://dx.doi.org/10.1090/conm/306/05250.

DEPARTMENT OF MATHEMATICS, OREGON STATE UNIVERSITY, CORVALLIS, OR 97331, USA