Chapter 21

The porous medium equation

Prerequisites: *Chapters 7, 8, 20*

21.1 ▪ Introduction

The *porous medium equation*

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u^m}{\partial x^2} \qquad (m > 1) \tag{21.1}$$

is a model of diffusion of a substance, typically gas, through a porous medium. The unknown $u(x, t)$ is the gas density at the point x at time t. The exponent m—which is not necessarily an integer—is a physical property of the diffusing material. Equation (21.1) arises in other contexts as well. Have a look at this chapter's appendix to see how it comes about as a model in population dynamics. You will find a quite readable treatise on the subject, along with an extensive bibliography, in Vázquez [77]. One thing you should know is that if the initial condition $u(x, 0)$ is nonnegative, then the solution $u(x, t)$ is nonnegative for all $t > 0$, and therefore the exponentiation makes sense in that case. See Section 21.3 for a generalization to solutions of varying sign.

The goal in this chapter is to develop a finite difference scheme to solve initial/boundary value problems corresponding to (21.1). As we will see, the *Seidman sweep* scheme introduced in Section 20.5 leads to quite a simple implementation.

21.2 ▪ Barenblatt's solution

When $m = 1$, (21.1) reduces to the heat equation; cf. (20.1) on page 251. The condition $m > 1$, however, makes the porous medium equation a totally different beast compared to the heat equation. You may intuit a sign of trouble if you express (21.1) in the equivalent form $\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(mu^{m-1}\frac{\partial u}{\partial x}\right)$, which makes it clear that it is a nonlinear diffusion equation, as in $\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\kappa\frac{\partial u}{\partial x}\right)$, whose diffusion coefficient, $\kappa = mu^{m-1}$, varies with the density. Since $m > 1$, κ tends to zero as density tends to zero. Thus, the diffusion process *degenerates* near zero densities. The porous medium equation is the archetype of the class of *degenerate parabolic equations*.

The bulk of the theory of parabolic equations concerns the nondegenerate case. Just about everything breaks down when degeneracy occurs. One of the striking facts about

the porous medium equation—a consequence of the degeneracy—is that its solutions exhibit a *finite speed of propagation* into vacuum (i.e., a zero density region). This is very much in contrast to the heat equation whose solutions spread out at infinite speed. The finite speed of propagation is evident in the class of self-similar solutions to (21.1) constructed by Barenblatt:

$$u(x,t) = \frac{1}{(t+\delta)^\beta}\left(\left[c - \gamma\left(\frac{x}{(t+\delta)^\beta}\right)^2\right]_+\right)^\alpha, \qquad -\infty < x < \infty, \quad t \ge 0, \tag{21.2}$$

where

$$\alpha = \frac{1}{m-1}, \quad \beta = \frac{1}{m+1}, \quad \gamma = \frac{m-1}{2m(m+1)},$$

and where $c \ge 0$ and $\delta \ge 0$ are arbitrary constants. The notation $[\,\cdot\,]_+$ means $\max(\cdot, 0)$. We see that at any time $t > 0$ the support[83] of the solution is

$$|x| < \sqrt{\frac{c}{\gamma}}\,(t+\delta)^\beta,$$

which propagates at a finite (but nonconstant) speed, as asserted.

In the special case of $m = 3$, Barenblatt's solution takes on a particularly simple form

$$u(x,t) = \frac{1}{(t+\delta)^{1/4}}\sqrt{c - \frac{x^2}{12(t+\delta)^{1/2}}}, \tag{21.3}$$

or equivalently,

$$\frac{x^2}{12c(t+\delta)^{1/2}} + \frac{u^2}{c(t+\delta)^{-1/2}} = 1.$$

Therefore, the graph of u versus x (for any fixed t) is precisely the upper half of an ellipse with semimajor and semiminor axes lengths of $\sqrt{c(t+\delta)^{-1/2}}$ and $\sqrt{12c(t+\delta)^{1/2}}$. The ellipse flattens and spreads out as t increases. The area under the ellipse, which is proportional to the mass of the diffusing substance, remains constant at $\sqrt{3}\pi c$. Figure 21.1 shows snapshots of the graphs of $u(x,t)$ for several choices of t.

Barenblatt's solution is quite handy as a test case for our finite difference solver. We set up a problem with initial and boundary data derived from Barenblatt's solution, and then we expect that the solution produced by our solver will agree with Barenblatt's.

21.3 ▪ Generalizations

Since the exponent m in (21.1) is not necessarily an integer, the expression u^m is not well defined if u is negative. The extension

$$\frac{\partial u}{\partial t} = \frac{\partial^2(|u|^{m-1}u)}{\partial x^2} \tag{21.4}$$

of (21.1) admits negative u, is well-posed as a partial differential equation, and reduces to (21.1) when u is nonnegative. For this reason, most of the literature on the porous medium equation addresses (21.4) rather than the special case (21.1). Yet a further generalization of (21.4) is the equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 \phi(u)}{\partial x^2}, \tag{21.5}$$

[83] The *support* of a function is the closure of the set where it is nonzero.

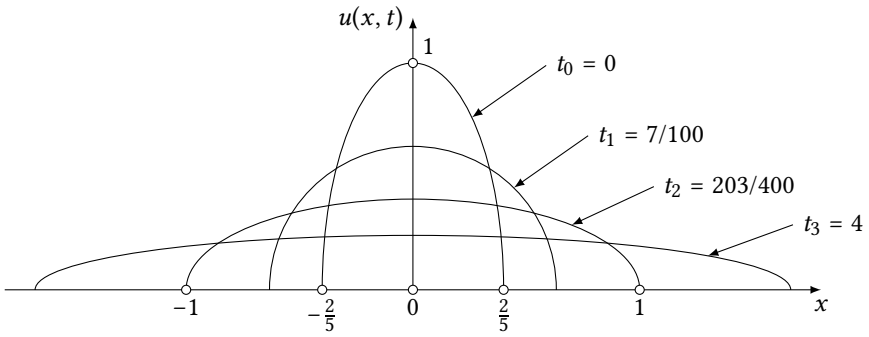

Figure 21.1: These flattening and spreading ellipses are snapshots of Barenblatt's solution, (21.3), of the porous medium equation with $m = 3$ at various times. The parameters $\delta = 1/75$ and $c = \sqrt{3}/15$ are chosen so that the solution at time $t_0 = 0$ has amplitude 1 and support $[-2/5, 2/5]$. I will leave it to you to verify that (a) at time $t_1 = 7/100$ the solution curve takes the form of a semicircle of radius $\sqrt{2/5}$, and (b) the solution front arrives at $x = \pm 1$ at time $t_2 = 203/400$. The solution curve at $t_3 = 4$ is also shown to give a better feel for how the solution spreads out in time.

where $\phi : \boldsymbol{R} \longrightarrow \boldsymbol{R}$ is some smooth monotonically increasing function with $\phi(0) = 0$ and $\phi'(0) = 0$. Much of the theory pertaining to the porous medium equation may be developed in the context of (21.5) with very minimal assumptions on ϕ. See, for instance, the articles [2, 9] and the book [77]. Clearly, (21.4) is the special case of (21.5) with $\phi(u) = |u|^{m-1}u$. I will state and explain this chapter's algorithm in the context of (21.5) for the sake of generality but will drop to the simple case of (21.1) for illustrations.

21.4 ▪ The finite difference scheme

The porous medium equation's counterpart of the initial/boundary value problem (20.2) is the following:

Find $u = u(x, t)$ so that

$$\frac{\partial u}{\partial t} = \frac{\partial^2 \phi(u)}{\partial x^2}, \qquad x \in (a, b),\ t > 0, \tag{21.6a}$$

$$u(x, 0) = u_0(x), \qquad x \in (a, b), \tag{21.6b}$$

$$u(a, t) = u_L(t), \quad u(b, t) = u_R(t), \qquad t > 0. \tag{21.6c}$$

As in the case of Chapter 20's heat equation, the initial condition $u_0(x)$ and the left and right boundary conditions $u_L(t)$ and $u_R(t)$ serve to define a unique solution $u(x, t)$ in the semi-infinite strip $a \le x \le b$ and $t > 0$ in the x-t plane. Also as before, in a finite difference approximation we replace the interval $a \le x \le b$ with a collection of $n + 1$ equally spaced points $x_0 < x_1 < \cdots < x_n$, where $x_0 = a$ and $x_n = b$, and we let $\Delta x = (b - a)/n$. Similarly, we discretize the time into "time-slices" $t_0 < t_1 < t_2 \cdots$, where $t_0 = 0$, and the spacing between the slices is a prescribed Δt. Figure 20.2 on page 253 shows the resulting finite difference grid.

In the rest of this section I will use the notation and ideas introduced in Section 20.5 without further elaboration. The forward and reverse difference formulas (20.16) (on

page 261) now take the form

$$v_j - u_j = -r\big(\phi(v_j) - \phi(v_{j-1})\big) - r\big(\phi(u_j) - \phi(u_{j+1})\big), \qquad j = 1, 2, \dots, n-1, \tag{21.7a}$$

$$w_j - v_j = -r\big(\phi(v_j) - \phi(v_{j-1})\big) - r\big(\phi(w_j) - \phi(w_{j+1})\big), \qquad j = n-1, \dots, 2, 1, \tag{21.7b}$$

where

$$r = \frac{\Delta t}{2(\Delta x)^2},$$

as in (20.15). (I have changed the notation from r' to r here since there is no chance of confusion within this chapter.) We rearrange the equations (21.7) into

$$v_j + r\phi(v_j) = r\phi(v_{j-1}) + u_j - r\phi(u_j) + r\phi(u_{j+1}), \qquad j = 1, 2, \dots, n-1, \tag{21.8a}$$

$$w_j + r\phi(w_j) = r\phi(v_{j-1}) + v_j - r\phi(v_j) + r\phi(w_{j+1}), \qquad j = n-1, \dots, 2, 1. \tag{21.8b}$$

During the forward sweep, all the terms on the right-hand side of (21.8a) are known. We solve the nonlinear equation $v_j + r\phi(v_j) =$ "known" to find v_j. Similarly, during the reverse sweep, all the terms on the right-hand side of (21.8b) are known. We solve the nonlinear equation $w_j + r\phi(w_j) =$ "known" to find w_j. Thus, in comparison with the heat equation of Chapter 20, the only extra effort is in solving a nonlinear equation of the form $\xi + r\phi(\xi) = c$ at each step. This may be accomplished through a Newton's iteration without much trouble by starting with an initial guess ξ_0. Since ϕ is expected to be a monotonically increasing function, any reasonable choice for ξ_0 will do. I suggest taking $\xi_0 = c$ and leave its implementation as an instructive project. In the rest of this chapter, however, I will focus on the special case of $\phi(u) = u^3$, which avoids Newton's iteration altogether. Here is why.

To solve the cubic equation $\xi + r\xi^3 = c$ for ξ, we multiply it through by $r^{1/2}$ to get $r^{1/2}\xi + r^{3/2}\xi^3 = r^{1/2}c$. Letting $\eta = r^{1/2}\xi$ and $k = r^{1/2}c$, we arrive at the cubic equation $\eta + \eta^3 = k$. You may verify that the unique real root of $\eta + \eta^3 = k$ has the explicit form

$$\eta = \frac{\gamma}{6} - \frac{2}{\gamma}, \quad \text{where} \quad \gamma = \Big[108k + 12\sqrt{12 + 81k^2}\Big]^{1/3}. \tag{21.9}$$

Thus, we evaluate η and then set $\xi = r^{-1/2}\eta = \eta/\sqrt{r}$.

21.5 ▪ The program

The rest of this chapter is devoted to details of the implementations of the Seidman sweep for solving the initial/boundary value problem (21.6) in the special case when $\phi(u) = u^3$. The case of a general ϕ is left as a project. Our program relies on *xmalloc.[ch]* from Chapter 7 to allocate memory, the file *array.h* from Chapter 8 to construct vectors and matrices, and the files `plot3d.[ch]` from Chapter 20 to generate plotting scripts. We also adapt Chapter 20's *heat-solve.h* header file, with minor alterations, to the case in hand, and name it *pme-solve.h.* Thus, following the suggestions in Chapters 2 and 6, the contents of the project's directory will look like this:

```
$ cd pme
$ ls -F
Makefile  demo1.c  plot3d.c@  pme-solve.c  xmalloc.c@
array.h@  demo2.c  plot3d.h@  pme-solve.h  xmalloc.h@
```

The files *demo1.c* and *demo2.c* contain the specifications of two initial/boundary value problems for the porous medium equation, both with $\phi(u) = u^3$. These are the following:

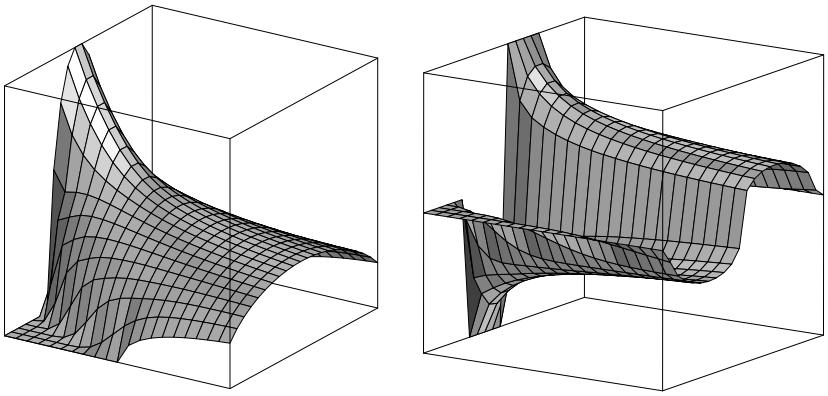

Figure 21.2: Graphs of the solutions $u(x, t)$ of problems `pme1` (left) and `pme2` (right) as surfaces in three dimensions computed by our finite difference solver and rendered in GEOMVIEW. Compare the graph of `pme1` to the corresponding snapshots in Figure 21.1.

.

Problem pme1: This produces a finite difference approximation to Barenblatt's solution (21.2) of the porous medium equation with $m = 3$, $c = \sqrt{3}/15$, and $\delta = 1/75$ on the bounded interval $-1 \le x \le 1$. The values of c and δ are chosen so that the initial condition (whose graph is the upper half of an ellipse) has amplitude 1 and is supported on the interval $[-2/5, -2/5]$. Figure 21.1 shows snapshots of the solution at a few selected times. Figure 21.2(left) shows the graph of the solution $u(x, t)$ as a surface in three dimensions.

Implementing the problem involves writing a function to evaluate Barenblatt's solution $u(x, t)$ in (21.2) and then extracting its initial and boundary data by evaluating $u(x, 0)$ and $u(\pm 1, t)$, as explained in the "reverse engineering" idea in (20.19) on page 263.

Problem pme2: This solves the initial/boundary value problem (21.6) on the interval $-1 \le x \le 1$ with the boundary data $u_L(t) = u_R(t) \equiv 0$ and the initial data

$$u_0(x) = \begin{cases} 1/2 & \text{if } 0 < x < 1/2, \\ -1/2 & \text{if } -1/2 < x < 0, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, this tests the scheme's ability to handle initial data of variable sign. Figure 21.2(right) shows the graph of the solution $u(x, t)$ as a surface in three dimensions. No exact solution is available in this case.

Listing 21.1 shows a transcript of a sample interactive session. The largish error is due to the steep (actually infinite) slope of the solution $u(x, t)$ at the edge of its propagating front.

21.6 ▪ The file *pme-solve.h*

The file *pme-solve.h*, shown in Listing 21.2, is a slight modification of the file *heat-solve.h* of Chapter 20. Specifically, the declaration of **`enum`** `method` has been removed since

Listing 21.1: A transcript of a sample interactive session.

```
$ ./demo1
Usage: ./demo1 T n m
   T : solve over 0 ≤ t ≤ T
   n : number of subintervals in the x direction
   m : number of subintervals in the t direction

$ ./demo1 1 20 40
r = 1.25
Geomview script written to zz.gv
absolute error = 0.197579
```

Listing 21.2: The contents of the file *pme-solve.h*.

```
1  #ifndef H_PME_SOLVE_H
2  #define H_PME_SOLVE_H
3
4  struct pme_solve {
5      double a;                                    // left end at x = a
6      double b;                                    // right end at x = b
7      double T;                                    // solve for 0 < t < T
8      int n;                                       // number of x subintervals
9      int m;                                       // number of t subintervals
10     double (*ic)(double x);                      // initial condition
11     double (*bcL)(double t);                     // left boundary condition
12     double (*bcR)(double t);                     // right boundary condition
13     double **u;                                  // solution array
14     double (*exact_sol)(double x, double t); // exact solution, if any
15     double error;                                // error vs exact solution
16     char *maple_out;                             // output file for maple graphics
17     char *matlab_out;                            // output file for matlab graphics
18     char *geomview_out;                          // output file for geomview graph-
   ics
19 };
20
21 void show_usage_and_exit(char *progname);
22 void pme_solve(struct pme_solve *prob);
23
24 #endif /* H_PME_SOLVE_H */
```

the only method used in the chapter is the Seidman sweep. Furthermore, we have removed the `method` field from what was ***struct*** *heat_solve* and is now renamed ***struct*** *pme-solve*. And finally, near the end we have changed the name of the solver from `heat_solve()` to `pme_solve()`.

21.7 ▪ The file *pme-solve.c*

The file *pme-solve.c* also shares a number of common elements with Chapter 20's *heat-solve.c*. Listing 21.3 shows an outline. Here are a few comments on that listing.

Line 2. Here we define the function $\phi(x) = x^3$ as a preprocessor macro. Alternatively,

Listing 21.3: An outline of the file *pme-solve.c*.

```
1  ▸ #include ...
2  #define phi(x)  pow((x),3)
3  ▸ static void write_plotting_script(struct pme_solve *prob) ...
4  ▸ void show_usage_and_exit(char *progname) ...
5  ▸ static double error_vs_exact(struct pme_solve *prob) ...
6  ▸ static double croot(double k) ...
7  ▸ void pme_solve(struct pme_solve *prob) ...
```

we could have defined ϕ as a function. There is no significant difference between the two.

Line 3. The function `write_plotting_script()` is essentially identical to that form *heat-solve.c*. The only difference is the change in the argument type from what used to be **`struct`** `heat_solve` to **`struct`** `pme_solve`.

Lines 4 and 5. These functions are identical to those in *heat-solve.c*.

Line 6. The function `croot()` calculates and returns the unique real root of the cubic equation $\eta + \eta^3 = k$ according to the formula given in (21.9). I will leave it to you to write the code.

Line 7. The function `pme_solve()` is the main workhorse of this module. I will describe its contents in the next section.

21.8 ▪ The function `pme_solve()`

The function `pme_solve()` which occurs on line 7 of Listing 21.3 contains an adaptation of the Seidman sweep scheme to solving the porous medium equation. The bulk of the function is shown Listings 21.4. The first few lines of Listing21.4 are self-explanatory. Here are some comments on the rest of those listings.

Line 9. We set $s = \sqrt{r}$ for convenience since $\sqrt{r}$ appears in a few places in the calculations surround the equation (21.9).

Lines 10 and 12. The Seidman sweep scheme goes from time step i to time step $i + 1$ through an intermediate half-way-in-between time. The array `v` is a temporary storage to hold the values of the solution at that intermediate time.

Line 16. We initialize the solution array `u` by filling its first row with the values supplied by the initial condition.

Line 21. Seidman sweep's main iterative loop begins here and extends to line 40. It consists of the forward sweep (lines 24–30), and the reverse sweep (lines 33–35).

Line 24. The left boundary condition determines the value `v[0]` of the intermediate array `v`.

Lines 26 and 28. Here we solve the equation (21.8a) for v_j. As it was explained in Section 21.4, this amounts to solving the cubic equation $\xi + r\xi^3 = c$, or the transformed version $\eta + \eta^3 = r^{1/2}c$, where $\xi = \eta/r^{1/2}$. Therefore, on line 26 we calculate c, and then on line 28 we apply the function `croot()` to solve the cubic, divide the result by $r^{1/2}$, and assign the result to `v[j]`.

Listing 21.4: Here is the function `pme-solve()`. Lines 24–30 implement the forward sweep. You need to supply the missing code for the reverse sweep on lines 35–37.

```
1  void pme_solve(struct pme_solve *prob)
2  {
3      int m = prob→m;
4      int n = prob→n;
5      double **u = prob→u;
6      double dx = (prob→b - prob→a) / n;      // space-step
7      double dt = prob→T / m;                 // time-step
8      double r = dt/(2*dx*dx);
9      double s = sqrt(r);
10     double *v;
11
12     make_vector(v, n+1);
13
14     printf("r = %g\n", r);
15
16     for (int j = 0; j ≤ n; j++) {
17         double x = prob→a + j*dx;
18         u[0][j] = prob→ic(x);
19     }
20
21     for (int i = 1; i ≤ m; i++) {
22         double t = i*dt;
23
24         v[0] = prob→bcL(t-dt/2);         // forward sweep
25         for (int j = 1; j ≤ n-1; j++) {
26             double c = r*phi(v[j-1]) + u[i-1][j]
27                         - r*phi(u[i-1][j]) + r*phi(u[i-1][j+1]);
28             v[j] = croot(s*c)/s;
29         }
30         v[n] = prob→bcR(t-dt/2);         // not used
31
32
33         u[i][n] = prob→bcR(t);           //reverse sweep
34         for (int j = n-1; j ≥ 1; j--) {
35 ▸           // ... supply the missing code ...
36 ▸           ...
37 ▸           u[i][j] = ...
38         }
39         u[i][0] = prob→bcL(t);
40     }
41     free_vector(v);
42
43     write_plotting_script(prob);
44
45     if (prob→exact_sol ≠ NULL)
46         prob→error = error_vs_exact(prob);
47 }
```

Line 30. We complete the evaluation of the array `v` by assigning `v[n]` from the right boundary condition. Actually this is not necessary at all since the value of `v[n]` is not needed in the subsequent calculations. This line can be safely commented out.

Lines 33–39. In the reverse sweep we calculate the array entries `u[i][j]`, based on the values of the intermediate array `v`, by solving the equation (21.8b). You need to supply a couple of lines of missing code very much similar to those in the forward sweep case.

21.9 ▪ The file *demo1.c*

The file *demo1.c*, shown in an outline form in Listing 21.5, formulates and solve an initial boundary value for the porous medium equation with a known exact solution. Let's look at its details.

Line 3. The function `barenblatt()` encodes Barenblatt's solution given in (21.2). The evaluation depend on the independent variables x and t, and on the parameters m, c, and δ, which we supply as arguments.

Lines 7–11. The function `exact_sol()` evaluates and returns the Barenblatt's solution with the parameters $m = 3$, $c = \sqrt{3}/15$, and $\delta = 1/75$. The choice of these parameters was explained in the description of **Problem pme1** on page 284. We generate initial and boundary data based on this function, and then validate the solution produced by our finite difference solver by comparing to it.

Lines 13–15. The functions `ic()`, `bc_L()`, and `bc_R()` simply evaluate `exact_sol()` at $t = 0$ and $x = \pm 1$ to produce the problem's initial and left and right boundary conditions.

Line 21. Populate the entries of a **`struct`** `pme_solve`. See the complete description of that structure in Section 21.6.

Lines 28–30. Allocate an $(m+1)\times(n+1)$ array `u` to hold the values of the solution on the grid points, call `pme_solve()` to solve the problem, and then free the allocated memory.

Recalling our implementation of `pme_solve()`, it will produce scripts for plotting solutions if the `prob` structure provides file names for the scripts, and it will print to `stdout` the absolute error between the calculated and exact solutions if an exact solution is provided.

21.10 ▪ *Project Porous Medium*

Part 21.1. Complete the implementation of this chapter's finite difference solver and try it out with *demo1.c* which formulates **Problem pme1** on page 284. The output of the program is shown in Listing 21.1. The solution is plotted in Figure 21.2 on the left.

Part 21.2. Write a file *demo2.c* which formulates and solves **Problem pme2** on page 285.

Part 21.3. [Optional] Modify your program to handle a general $\phi : \boldsymbol{R} \longrightarrow \boldsymbol{R}$ assuming $\phi(0) = 0$, $\phi'(0) = 0$, and ϕ is monotonically increasing. You will replace the function `croot()` with a solver (using Newton's iteration) for the equation $\xi + r\phi(\xi) = c$.

Listing 21.5: An outline of the file *demo1.c*.

```
▸ #include ...
// Barenblatt's solution
▸ static double barenblatt(double x, double t, double m,
        double c, double delta) ...

// pme1: Barenblatt's solution with m = 3, and special choices of c and δ
static double exact_sol(double x, double t)
{
    double c = sqrt(3)/15, delta = 1.0/75;
    return barenblatt(x, t, 3, c, delta);
}

▸ static double ic(double x) ...
▸ static double bc_L(double t) ...
▸ static double bc_R(double t) ...

int main(int argc, char **argv)
{
    ▸ extract the command-line arguments T, n, m
      call show_usage_and_exit() if invalid arguments
    ▸ struct pme_solve prob = {
        .a      = -1,
        .b      = 1,
        .T      = T,
        ...
    };

    make_matrix(prob.u, m+1, n+1);
    pme_solve(&prob);
    free_matrix(prob.u);

    return EXIT_SUCCESS;
}
```

21.11 ▪ Appendix: The porous medium equation as a population dynamics model

It is not out of place to show how (21.6a) arises out of a simple population model. Although the treatment of this chapter has been limited to a one-dimensional space, the derivation of the model works more transparently in an n-dimensional setting, and that's what I will do. At the very end you may set $n = 1$ if you like.

Consider a certain hypothetical population that lives in the n-dimensional space $\boldsymbol{R}^n$. Assume that there exists a *population density function* $u(\boldsymbol{x}, t)$ so that the population in any arbitrary region $\omega \subset \boldsymbol{R}^n$ at time t is given by $\int_\omega u(\boldsymbol{x}, t)\, d\boldsymbol{x}$. Also assume that there exists a vector function $\boldsymbol{v}(\boldsymbol{x}, t)$ that gives the velocity of the movement of the individuals at the point $\boldsymbol{x}$ at time t. For simplicity's sake, let's assume that the members of the population don't reproduce and don't die. Then the rate of increase of the population in ω equals

exactly the inflow of the population through its boundary, that is,

$$\frac{d}{dt}\int_\omega u\,d\boldsymbol{x} = -\int_{\partial\omega} u\boldsymbol{v}\cdot\boldsymbol{n}\,da,$$

where $\boldsymbol{n}$ is the outward unit normal to the boundary $\partial\omega$ of ω.

We move the time derivative to under the integration sign. That's permissible since ω is independent of time. On the right-hand side, we may apply the *Divergence Theorem* (see, e.g., the section titled "Curl and Divergence" in Stewart [65]) from multivariable calculus to change the boundary integral to a volume integral. We get

$$\int_\omega \frac{\partial u}{\partial t}\,d\boldsymbol{x} = -\int_\omega \operatorname{div}(u\boldsymbol{v})\,da.$$

Since ω is arbitrary, we conclude that

$$\frac{\partial u}{\partial t} + \operatorname{div}(u\boldsymbol{v}) = 0.$$

What we have obtained is the *equation of conservation of mass*. That we derived it in the context of population dynamics is quite irrelevant. The density u and velocity $\boldsymbol{v}$ could have been those of the exhaust gases of a rocket, the air flowing around an airplane's wings, blood coursing through an animal's veins, or a vibrating metal plate. The equation of the conservation of mass links a material's velocity and density functions, assuming that the material is neither created nor destroyed.

Returning to the population model, assume that the species are averse to living in high density areas. More precisely, if the population density is not constant in a particular individual's neighborhood, the individual runs to a lower density area; that is, it moves in the opposite direction of the density gradient, ∇u, and the speed of the movement is a function of the density itself, let's say $\boldsymbol{v} = -\psi(u)\nabla u$. Substituting this in the equation of conservation of mass results in

$$\frac{\partial u}{\partial t} = \operatorname{div}\left[u\psi(u)\nabla u\right].$$

To connect this to the porous medium equation, introduce a function ϕ through $\phi(u) = \int_0^u \sigma\psi(\sigma)\,d\sigma$, that is, $\phi'(u) = u\psi(u)$ and $\phi(0) = 0$. Then the expression inside the square brackets becomes $\phi'(u)\nabla u$, that is, $\nabla\phi(u)$, and we conclude that

$$\frac{\partial u}{\partial t} = \operatorname{div}\nabla\phi(u) = \nabla^2\phi(u).$$

In the one-dimensional case this reduces to (21.6a).