9-1992

Numerical Solution of Laplace's Equation

Per Brinch Hansen
Syracuse University, School of Computer and Information Science, pbh@top.cis.syr.edu

SU-CIS-92-17

Numerical Solution of Laplace's Equation

Per Brinch Hansen

September 1992

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, NY 13244-4100

Numerical Solution of Laplace's Equation[1]

PER BRINCH HANSEN

Syracuse University, Syracuse, New York 13244

September 1992

This tutorial discusses Laplace's equation for steady state heat flow in a two-dimensional region with fixed temperatures on the boundaries. The equilibrium temperatures are computed for a square grid using successive overrelaxation with parity ordering of the grid elements. The numerical method is illustrated by a Pascal algorithm. We assume that the reader is familiar with elementary calculus.

Categories and Subject Descriptors: G.1.8 [**Numerical Analysis**]: Partial Differential Equations—*difference methods*

General Terms: Algorithms

Additional Key Words and Phrases: Laplace's equation

CONTENTS

INTRODUCTION

Physical phenomena that vary continuously in space and time are described by *partial differential equations*. The most important of these is *Laplace's equation*, which defines gravitational and electrostatic potentials as well as stationary flow of heat and ideal fluid [Feynman 1989].

This tutorial discusses Laplace's equation for steady state heat flow in a two-dimensional region with fixed temperatures on the boundaries. The equilibrium temperatures are computed on a square grid using *successive overrelaxation* with *parity ordering* of the grid elements [Young 1971, Press et al. 1989].

The numerical method is illustrated by a Pascal algorithm. A parallel version of this algorithm has been tested on a Computing Surface [Brinch Hansen 1992].

We assume that the reader is familiar with elementary calculus.

1. THE HEAT EQUATION

We will illustrate Laplace's equation by defining the equilibrium temperatures in a thin plate of homogeneous material and uniform thickness. The faces of the plate are perfectly insulated. On the boundaries of the plate, each point is kept at a known fixed temperature. As heat flows through the plate from warm towards cold boundaries, the plate eventually reaches a state where each point has a steady (time-independent) temperature maintained by the heat flow. The problem is to define the equilibrium temperature $u(x, y)$ for every point (x, y) on the plate.

We will study the heat flow through a small rectangular element of the plate with origin (x, y) and sides of length Δx and Δy (Fig. 1).

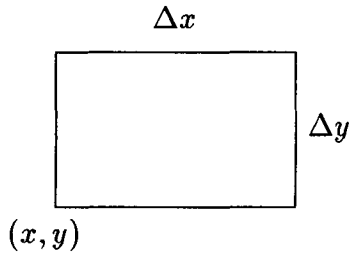

Fig. 1 A small element

Due to the insulation of the faces, the heat flows in two dimensions only. At every point (x, y) the velocity v of the heat flow has a horizontal component v_x and a vertical component v_y (Fig. 2). We will examine the horizontal and vertical flows separately.

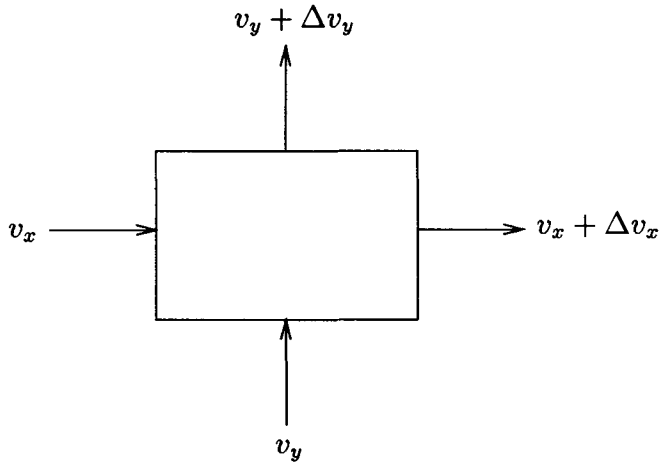

Fig. 2 Horizontal and vertical flow

The horizontal heat flow enters the element through the left boundary with velocity v_x and leaves the element through the right boundary with velocity $v_x + \Delta v_x$. Since these boundaries have length Δy, the element receives heat at the rate

$$v_x \Delta y$$

and loses heat at the rate

$$(v_x + \Delta v_x)\Delta y$$

So the horizontal flow removes heat from the element at the rate

$$\Delta v_x \Delta y = \frac{\partial v_x}{\partial x} \Delta x \Delta y$$

Similarly, the vertical flow removes heat from the element at the rate

$$\Delta v_y \Delta x = \frac{\partial v_y}{\partial y} \Delta y \Delta x$$

The combined rate of heat loss is the sum of the horizontal and vertical loss rates:

$$\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \Delta x \Delta y$$

In equilibrium, the element holds a constant amount of heat, which keeps its temperature $u(x, y)$ constant. Since the rate of heat loss is zero in the steady state, we have the *heat conservation* law:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} = 0 \tag{1}$$

Now, heat flows towards decreasing temperatures at a rate proportional to the temperature gradient:

$$v_x = -k\frac{\partial u}{\partial x} \qquad v_y = -k\frac{\partial u}{\partial y} \tag{2}$$

where k is a constant [Feynman 1989]. This is the law of the *velocity potential.*

By combining the conservation and potential laws, we obtain *Laplace's equation* for equilibrium temperatures:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{3}$$

This partial differential equation is also known as the *heat equation* or the *equilibrium equation.* It is often written as follows:

$$\nabla^2 u = 0 \tag{4}$$

A function $u(x, y)$ that satisfies this equation is called a *potential function.* The *boundary conditions* determine a potential function.

2. DIFFERENCE EQUATIONS

As an example, we will solve the heat equation for a *square region*, where the temperature is fixed at each boundary (Fig. 3).

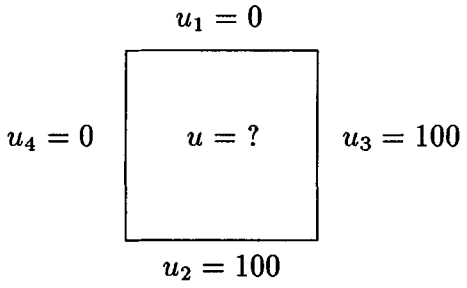

Fig. 3 A square region

A numerical solution can be found only for a *discrete* form of the heat equation. The first step is to divide the square region into a *square grid* of elements with uniform *spacing* h (Fig. 4). The center of each element represents a single point in the region. There are three kinds of grid elements:

1) *Interior elements*, marked "?", have unknown temperatures, which satisfy a discrete heat equation.

2) *Boundary* elements, marked "+", have fixed, known temperatures.

3) *Corner elements*, marked "−", are not used.

−	+	+	+	+	−
+	?	?	?	?	+
+	?	?	?	?	+
+	?	?	?	?	+
+	?	?	?	?	+
−	+	+	+	+	−

Fig. 4 A square grid

A numerical solution of the heat equation is based on the observation that the heat flow through an interior element is driven by *temperature differences* between the element and its immediate neighbors. Figure 5 shows an interior element and four adjacent elements. The five elements are called c (center), n (north), s (south), e (east), and w (west).

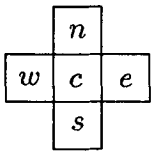

Fig. 5 Adjacent elements

These elements have the following *coordinates* and temperatures:

$$u_n = u(x, y + h)$$

$$u_w = u(x - h, y) \quad u_c = u(x, y) \qquad u_e = u(x + h, y)$$

$$u_s = u(x, y - h)$$

If the grid spacing h is small enough, we can approximate the temperature u_e by the first three terms of a *Taylor expansion* of the function u in the neighborhood east of (x, y) [Courant and John 1989]:

$$u_e \approx u_c + h\frac{\partial u}{\partial x} + \frac{1}{2}h^2\,\frac{\partial^2 u}{\partial x^2}$$

A Taylor expansion in the western neighborhood of (x, y) gives an approximation of the temperature u_w:

$$u_w \approx u_c - h\frac{\partial u}{\partial x} + \frac{1}{2}h^2\,\frac{\partial^2 u}{\partial x^2}$$

Consequently,

$$u_e + u_w \approx 2u_c + h^2 \frac{\partial^2 u}{\partial x^2} \tag{5}$$

Similarly,

$$u_n + u_s \approx 2u_c + h^2 \frac{\partial^2 u}{\partial y^2} \tag{6}$$

From (5) and (6) we obtain the important approximation:

$$\nabla^2 u \approx (u_n + u_s + u_e + u_w - 4\ u_c)/h^2$$

According to the heat equation (4), the left-hand side is zero for steady state heat flow. Consequently, the discrete heat equation is a system of *difference equations* of the form:

$$4\ u_c - u_n - u_s - u_e - u_w \approx 0 \tag{7}$$

There is a separate equation for each of the $n \times n$ internal elements.

In thermal *equilibrium*, the temperature of each grid element is simply the *average* of the temperatures of the four surrounding elements:

$$u_c \approx (u_n + u_s + u_e + u_w)/4 \tag{8}$$

3. NUMERICAL SOLUTION

On an $n \times n$ grid, the heat equation is replaced by n^2 linear equations in the unknown temperatures. Solving this system by a direct method requires $O(n^6)$ run time on a sequential computer. For a grid of, say, 250×250 elements, we obtain 62,500 linear equations with 4×10^9 coefficients. If the run time is measured in units of 1 μsec, the computation takes 8 years. This is clearly impractical.

Each equation (7) has five nonzero coefficients only. For such a *sparse* linear system, *iterative* methods are far more efficient than direct methods. Iterative solution of linear equations is known as *relaxation.* In the following, we develop a Pascal algorithm that uses relaxation to solve the discrete heat equation on a square grid with fixed boundary values.

A *grid* u is represented by an $(n+2) \times (n+2)$ real array:

```
const n = 250;
type row = array [0..n+1] of real;
  grid = array [0..n+1] of row;
var u: grid;
```

It consists of $n \times n$ interior elements surrounded by a border of $4n$ boundary elements as shown in Fig. 4.

The *Laplace procedure* initializes the boundary elements with fixed temperatures u_1, u_2, u_3, and u_4, and the interior elements with the tentative temperature u_5. This is followed by a fixed number of relaxation steps (Algorithm 1).

```
procedure laplace(var u: grid; u1, u2,
  u3, u4, u5: real; steps: integer);
var k: integer;
begin
  newgrid(u);
  for k := 1 to steps do relax(u)
end
```

Algorithm 1

A *new grid* holds known values in the boundary elements and reasonable values in the interior elements based on educated guessing (Algorithm 2).

```
procedure newgrid(var u: grid);
var i, j: integer;
begin
  for i := 0 to n + 1 do
    for j := 0 to n + 1 do
      u[i,j] := initial(i, j)
end
```

Algorithm 2

The known and estimated temperatures shown in Fig. 3 are defined by the same *initial* function (Algorithm 3). The temperatures of the four corner elements are arbitrary (and irrelevant).

```
function initial(i, j: integer)
  : real;
begin
  if i = 0 then
    initial := u1
  else if i = n + 1 then
    initial := u2
  else if j = n + 1 then
    initial := u3
  else if j = 0 then
    initial := u4
  else
    initial := u5
end
```

Algorithm 3

The operation

```
laplace(u, 0.0, 100.0,
  100.0, 0.0, 50.0, n)
```

solves the heat flow problem shown in Fig. 3. As a reasonable guess, the interior temperatures are initially set at the average of the boundary temperatures:

$$(0 + 100 + 100 + 0)/4 = 50$$

4. RELAXATION METHODS

A relaxation step replaces the temperature of every inner element by a better approximation based on the previous temperature of the element and the temperatures of its neighbors (see Fig. 5).

The simplest method is *Jacobi relaxation*, which conceptually updates every temperature simultaneously. This procedure must keep a copy of the current grid until the next temperatures have been computed (Algorithm 4).

```
procedure relax(var u: grid);
var u0: grid; i, j: integer;
begin
  u0 := u;
  for i := 1 to n do
    for j := 1 to n do
      u[i,j] := next(u0[i,j],
        u0[i−1,j], u0[i+1,j],
        u0[i,j+1], u0[i,j−1])
end
```

Algorithm 4

The most obvious idea is to replace every approximate temperature u_c by the average of the surrounding approximate temperatures u_n, u_s, u_e, and u_w (Algorithm 5).

```
function next(uc, un, us,
  ue, uw: real): real;
begin
  next := (un + us + ue +
    uw)/4.0
end
```

Algorithm 5

Algorithm 6 uses a new temperature as soon as it has been computed. This in-place computation cuts the memory requirement in half. The method is called

Gauss-Seidel relaxation, "even though Gauss didn't know about it and Seidel didn't recommend it" [Strang 1986].

```
procedure relax(var u: grid);
var i, j: integer;
begin
  for i := 1 to n do
    for j := 1 to n do
      u[i,j] := next(u[i,j],
        u[i−1,j], u[i+1,j],
        u[i,j+1], u[i,j−1])
end
```

Algorithm 6

Since new values are better approximations than old ones, Gauss-Seidel converges twice as fast as Jacobi. However, both methods require $O(n^2)$ relaxation steps and $O(n^4)$ run time [Press et al. 1989]. If the run time is measured in units of, say, 10 μsec, the computation of 250×250 equilibrium temperatures takes 11 hours. The slow convergence of these methods makes them impractical for scientific computing.

By 1950 engineers had discovered that convergence is much faster if temperatures are *overrelaxed*. Instead of approximating a new temperature by the average temperature

$$\text{aver} = (u_n + u_s + u_e + u_w)/4.0$$

we make the next temperature a weighted sum of the old temperature and the surrounding temperatures

$$\text{next} = (1 - f) * u_c + f * \text{aver}$$

The *relaxation factor* f is 1 for Gauss-Seidel. For overrelaxation, $f > 1$. This *ad hoc* method converges only if $f < 2$ [Press et al. 1989].

It was a breakthrough when David Young [1954] developed a theory of overrelaxation. For the heat equation with fixed boundary values on a large $n \times n$ square grid, the fastest convergence is obtained with the following relaxation factor:

$$f_{opt} = 2 - 2\pi/n$$

For $n = 250$, the optimal factor is $f_{opt} = 1.97$.

It is illuminating to look at overrelaxation from a different point of view. The *residual*

$$\text{res} = \text{aver} - u_c$$

is a measure of how well an approximate temperature u_c satisfies the discrete heat equation (7). In overrelaxation, the correction of each temperature is proportional to its residual:

$$\text{next} = u_c + f * \text{res}$$

where $1 < f < 2$.

The method of *successive overrelaxation* is based on this accidental discovery supported by Young's theory (Algorithm 7).

```
function next(uc, un, us,
  ue, uw: real): real;
var res: real;
begin
  res := (un + us + ue +
    + uw)/4.0 − uc;
  next := uc + fopt*res
end
```

Algorithm 7

The method requires n *steps* and $O(n^3)$ run time to achieve 3-figure accuracy [Press et al. 1986]. If the run time is measured in units of 10 μsec, a 250×250 grid requires less than 3 minutes of computing.

The theory of optimal overrelaxation is valid only if the grid elements are updated in an appropriate order. The row-wise updating of Gauss-Seidel is acceptable. However, it is inherently a sequential method. Instead, we will use an ordering that can be adapted for parallel computing [Young 1971, Barlow and Evans 1982, Brinch Hansen 1992].

Every element $u[i, j]$ has a *parity*

$$(i + j) \textbf{ mod } 2$$

which is either *even* (0) or *odd* (1). Even and odd elements alternate on the grid like black and white squares on a chessboard. Every interior element is surrounded by four elements of the opposite parity (Fig. 6). So, the updating of the even elements depends only on the odd elements, and vice versa.

−	1	0	1	0	−
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
−	0	1	0	1	−

Fig. 6 Parity ordering

Algorithm 8 defines *successive overrelaxation* with *parity ordering.* A relaxation step consists of a scan of the even elements followed by a scan of the odd elements. A single scan updates all elements with the same *parity b.* We assume that n is *even.*

```
procedure relax(var u: grid);
var b, i, j, k: integer;
begin
  for b := 0 to 1 do
    for i := 1 to n do
      begin
        k := (i + b) mod 2;
        j := 2 - k;
        while j <= n - k do
          begin
            u[i,j] := next(u[i,j],
              u[i-1,j], u[i+1,j],
              u[i,j+1], u[i,j-1]);
            j := j + 2
          end
      end
end
```

Algorithm 8

Successive overrelaxation can be refined further by *Chebyshev acceleration*, which changes the relaxation factor after each scan [Press et al. 1989]. However, the fastest relaxation method is the *multigrid* technique, which computes n^2 temperatures recursively on finer and finer grids in $O(n^2)$ time [Press and Teukolsky 1991].

5. PASCAL ALGORITHM

The complete Pascal algorithm shown below is composed of Algorithms 1–3 and 7–8.

```
const n = 250 {even};
type row = array [0..n+1] of real;
  grid = array [0..n+1] of row;

procedure laplace(var u: grid; u1, u2,
  u3, u4, u5: real; steps: integer);
const pi = 3.14159265358979;
var fopt: real; k: integer;

  function initial(i, j: integer)
    : real;
  begin
    if i = 0 then
      initial := u1
    else if i = n + 1 then
      initial := u2
    else if j = n + 1 then
      initial := u3
    else if j = 0 then
      initial := u4
    else
      initial := u5
  end;

  function next(uc, un, us,
    ue, uw: real): real;
  var res: real;
  begin
    res := (un + us + ue +
      + uw)/4.0 - uc;
    next := uc + fopt*res
  end;

  procedure newgrid(var u: grid);
  var i, j: integer;
  begin
    for i := 0 to n + 1 do
      for j := 0 to n + 1 do
        u[i,j] := initial(i, j)
  end;
```

```
  procedure relax(var u: grid);
  var b, i, j, k: integer;
  begin
    for b := 0 to 1 do
      for i := 1 to n do
        begin
          k := (i + b) mod 2;
          j := 2 - k;
          while j <= n - k do
            begin
              u[i,j] := next(u[i,j],
                u[i-1,j], u[i+1,j],
                u[i,j+1], u[i,j-1]);
              j := j + 2
            end
        end
  end;

begin
  fopt := 2.0 - 2.0*pi/n;
  newgrid(u);
  for k := 1 to steps do relax(u)
end
```

SUMMARY

We have derived Laplace's equation for steady-state heat flow in two dimensions and have explained how the equation is solved by successive overrelaxation on a discrete grid. The numerical method was illustrated by a Pascal algorithm.

ACKNOWLEDGEMENTS

I thank Jonathan Greenfield for his comments.

REFERENCES

BARLOW, R. H., and EVANS, D. J. 1982. Parallel algorithms for the iterative solution to linear systems. *Computer Journal* **25**, 1, 56–60.

BRINCH HANSEN, P. 1992. Parallel cellular automata. School of Computer and Information Science, Syracuse University, Syracuse, NY.

COURANT, R., and JOHN F. 1989. *Introduction to Calculus and Analysis.* Vol. II. Springer-Verlag, New York, NY.

FEYNMAN, R. P., LEIGHTON, R. B., and SANDS, M. L. 1989. *The Feynman Lectures on Physics.* Addison-Wesley, Redwood City, CA.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., and VETTERLING, W. T. 1989. *Numerical Recipes in Pascal: The Art of Scientific Computing.* Cambridge University Press, Cambridge, MA.

PRESS, W. H., TEUKOLSKY, S. A., 1991. Multigrid methods for boundary value problems. I. *Computers in Physics* **5**, 5, 514–519.

STRANG, G. 1986. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, Wellesley, MA.

YOUNG, D. M. 1954. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society* **76**, 92–111.

YOUNG, D. M. 1971. *Iterative Solution of Large Linear Systems.* Academic Press, New York, NY.