Forty years on — the Hall Effect with constant-voltage drive

Peter Fenwick

July 12, 2006

Abstract

This report describes the solution of a problem which arose about 40 years ago and seemed eminently suitable for computer solution. But numerical limitations made it quite unsuitable for complete solution on the computers then available. It is only now that a solution is feasible on generally-available machines.

The problem itself involves the solution of potentials over a Hall-effect plate driven with a constant voltage instead of the more usual constant current. The change from current-drive to voltage-drive greatly reduces the temperature sensitivity of the Hall output voltage, but at the cost of non-linear response at high magnetic fields. The problem is to find the shape of the response.

1 Background

This is the problem that introduced me to computing. For reasons that will become apparent, its solution languished and it is only now, 40 years later, that I have at last solved it.

In the summer of 1963–4 I was working as a casual laboratory technician in the University of Auckland Physics Department. Much of my work seemed to be with magnets for Advanced Laboratory experiments — I remember working with two large electromagnets, both weighing 10s of kilogrammes. One produced a high field of about 1 Tesla (10,000 gauss for the old-fashioned) for studying the Zeeman effect. The other produced a lower field, about 0.3 T, but very uniform, for demonstrating Nuclear Magnetic Resonance.

Also working with things magnetic I built a Hall-effect fluxmeter to measure magnetic fields. For reasons given in Section 2, the Hall element was driven by a constant voltage, using AC at about 1 kHz, with a switched gain amplifier and synchronous rectification. Its maximum full-scale sensitivity was about 100μT, or 1 gauss, comparable to the Earth's field.

But I knew that the output saturated at high fields (say $B > 0.1$T) and wondered just how the output varied with a high magnetic field. The University had then just got its first

computer, an IBM 1620 with all of 20k decimal digits of storage and a FP multiply time of 10 ms. Murray Johns suggested that I solve my problem on the computer as it seemed a good one for numerical solution. So I learned Fortran (with Format).

Prof Cecil Segedin had taught me Numerical Analysis in 1962 and suggested a semi-analytical approach using sinh and cosh functions. I quickly abandoned this approach in favour of a much simpler standard numerical relaxation (and one which I understood!) It worked, but the solution became unstable just as the saturation appeared.

The answer to *that* problem was a brute-force solution of the set of linear equations corresponding to the relaxation. Unfortunately the 1620 could handle only about 40 equations and even they took 20 minutes to solve[1]. With symmetries the 40 equations could handle a grid of say 6 points by 12 (solve only half), but even that was barely adequate for a sensible solution.

So the problem languished, and languished, and languished, and had competition from MSc coursework, and then PhD and teaching and so on. It was only in late 2004, 40 years later, that I felt able to return with a sensible attack.

2 The Physics

The Hall Effect is illustrated in Figure 1. A current I flowing *along* a conducting slab through a *vertical* magnetic field B causes a *transverse* potential to appear across the slab; this is the Hall voltage $V_H \propto I \times B$. (Strictly we should use vector quantities rather than scalars, but with perpendicular vectors we can ignore the difference.)

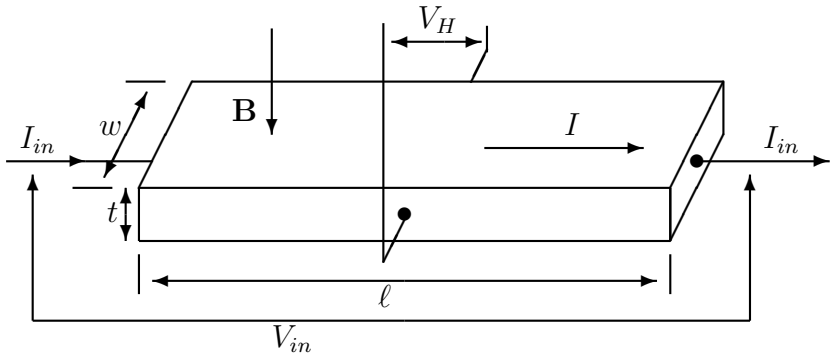

Figure 1: The Hall Effect

Assume now that the current is carried through the plate by a cloud of carriers with density n, each of charge q and moving with velocity v. Each charge carrier experiences a force $B \times q \times v$ across the plate; the resultant excess of charge on one side and deficit on the

[1]With experience you could follow the solution progress on control panel lights, and even see the individual floating point divisions of the Gaussian elimination.

other produces the Hall voltage V_H. In equilibrium, the Hall voltage produces a transverse electric field E_T which acts on the carriers to balance the magnetic force. Thus

$$Bqv = E_T q \text{ or } E_T = B \times v$$

The current is $I = wtnqv$, where w and t are the width and thickness of the plate (wt is the cross-sectional area). Then

$$v = \frac{I}{wtnq}$$

and

$$H_V = E_T w = \frac{BwtI}{wtnq} = \frac{BI}{tnq}$$

Several points may be noted –

- The Hall voltage is proportional to the product of the magnetic field B and the plate current I (strictly a vector product as noted above).
- The Hall voltage is independent of the *width*; a wider plate has the carriers moving slower, decreasing E_T, but the reduction is exactly offset by E_T generating a potential over the greater width.
- The Hall voltage is *inversely* proportional to the charge carrier density n. (With a lower density the fewer carriers must move faster to maintain the same current, and the higher velocity increases V_H.) Good Hall sensitivity therefore demands a *poor* conductor and preferably a semiconductor rather than a metal. (The units in this work used gallium arsenide.)

But, the charge carrier density n is critically dependent on temperature, making it difficult to use the Hall effect for accurate measurements without excellent temperature compensation.

Now consider that the external current I_{in} (which is equal to the internal current I) produces a voltage drop V_{in} along the plate, with a corresponding *longitudinal* electric field $E_\ell = V_{in}/\ell$. With a carrier mobility μ, the carrier velocity is $v = \mu E_\ell = \mu V_{in}/\ell$, the current is

$$I = vwtnq = \frac{\mu V_{in}}{\ell} wtnq = \frac{wt\mu V_{in} nq}{\ell}$$

and

$$H_V = \frac{BI}{n} = \frac{B}{n} \cdot \frac{wtq\mu V_{in} n}{\ell} = \frac{wtq\mu}{\ell} BV_{in}$$

Replacing the conventional constant-current drive with a constant-voltage drive makes the Hall voltage a function, not of the carrier density n, but of the carrier mobility μ which is much less temperature dependent. With a constant voltage drive the output clearly becomes non-linear as the Hall output approaches the input drive voltage; this is the price that must be paid for the reduced temperature dependence. (An interesting alternative might be to

use a constant-current drive and monitor *both* the Hall voltage and the end-to-end voltage. This has not been investigated.)

The question now is "How does the Hall voltage depend on the magnetic field with constant-voltage drive?" This involves solving the potential over the plate, given constant voltages at the end and other conditions as stated below.

3 The numerical equations

The problem is attacked by setting up a grid of points across the Hall surface with appropriate boundary conditions and solving for the potential distribution $\phi_{i,j}$ over the grid. Initially the equations were solved by Liebmann relaxation, iterating over the grid until the solution converges.

$$
\begin{array}{cc}
 & \textit{Edge of plate} \\
-V & \left\| \begin{array}{ccccc}
\phi_{i-2,0} & \phi_{i-1,0} & \phi_{i,0} & \phi_{i+1,0} & \cdots \\
\phi_{i-2,1} & \phi_{i-1,1} & \phi_{i,1} & \phi_{i+1,1} & \cdots \\
\phi_{i-2,2} & \phi_{i-1,2} & \phi_{i,2} & \phi_{i+1,2} & \cdots \\
\phi_{i-2,3} & \phi_{i-1,3} & \phi_{i,3} & \phi_{i+1,3} & \cdots \\
\phi_{i-2,4} & \phi_{i-1,4} & \phi_{i,4} & \phi_{i+1,4} & \cdots \\
\cdots
\end{array} \right.
\end{array}
$$

Figure 2: Potentials near one corner

There are three areas to consider to set up the equations (two with mirror-symmetry variants). All can be seen in a corner view as in Figure 2 using as a reference column one near the end..

1. At the left-hand end all grid points have the same potential of $-V$, and similarly at the right hand end, with $+V$.

$$\text{Here,} \quad \phi_{i-2,j} = -V \quad \forall j$$

2. In the body of the grid, the potentials satisfy Poisson's equation

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$$

for which the potential at each point is average of its 4 immediate neighbours

$$\phi_{i,j} = \frac{\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1}}{4}$$

3. For the edge boundary condition, the equipotentials are no longer perpendicular to the current flow, but are inclined by the magnetic field. But as the current flow at the edge *must* be parallel to the physical edge, so must the equipotentials be inclined to the edge. This condition can be written

$$\frac{\partial \phi}{\partial y} = B \frac{\partial \phi}{\partial x}$$

where B is an increasing function of the transverse magnetic field.

Assuming that the grid points have unit separation in both directions, it is easy to see that at the point $(i, 0)$

$$\left.\frac{\partial \phi}{\partial x}\right|_{i,0} = \frac{\phi_{i+1,0} - \phi_{i-1,0}}{2}$$

Now assume that the y derivative is a simple linear combination of three values, as

$$\left.\frac{\partial \phi}{\partial y}\right|_{i,0} = m = a\phi_{i,0} + b\phi_{i,1} + c\phi_{i,2}$$

To determine the coefficients a, b and c we assume 3 simple functions $\phi_j = F(j)$ and solve the resulting set of linear equations for the edge point derivative in each case[2]. The constant value and the quadratic both have zero derivative at the edge, while the linear has a slope $= m$. (Any quadratic can of course be formed as a linear combination of these functions.)

$F(j) =$ constant	$a + b + c = 0$	constant has zero slope
$F(j) = j$	$b + 2c = 1$	slope $= 1$
$F(j) = j^2$	$b + 4c = 0$	zero slope at $j = 0$

whence $a = -3/2, \quad b = 2, \quad c = -1/2$ and

$$\left.\frac{\partial \phi}{\partial y}\right|_{i,0} = -\frac{3}{2}\phi_{i,0} + 2\phi_{i,1} - \frac{1}{2}\phi_{i,2}$$

$$\begin{array}{lrcl} \text{As} & \frac{\partial \phi}{\partial y} & = & B\frac{\partial \phi}{\partial x} \\ \text{then} & -\frac{3}{2}\phi_{i,0} + 2\phi_{i,1} - \frac{1}{2}\phi_{i,2} & = & B\left[\frac{\phi_{i+1,0} - \phi_{i-1,0}}{2}\right] \\ \text{or} & 3\phi_{i,0} - 4\phi_{i,1} + \phi_{i,2} + B(\phi_{i+1,0} - \phi_{i-1,0}) & = & 0 \end{array}$$

4 Relaxation results

The first attempts used a conventional Liebmann relaxation over the grid, calculating interior points as the average of their neighbours (a standard solution of Poisson's equation). Edge points were calculated by a simple application of the edge formulæ.

Figure 3 shows the results for a relaxation over a 20×10 plate (or a 21×11 grid). Three lines are shown –

[2]This method, evaluating a few simple and well-chosen cases, was a particular favourite of Cecil Segedin

1. A linear fit to the other results, asymptotic at the origin.
2. A "second-order" result, corresponding to the equations derived above. It becomes numerically unstable beyond about $B = 1.9$.
3. A "first-order" result, using a simple 1st-order approximation to the partial derivatives. It is used mainly to check correctness of the more complex second-order equations and becomes unstable at about $B = 1.2$.

It is pleasing that the two results are almost indistinguishable where both give solutions, but it is really desirable to extend results to higher fields, even beyond the $B = 1.9$ limit.

But it is unfortunate that instability occurs at higher fields, precisely where saturation arises and the solution becomes "interesting".

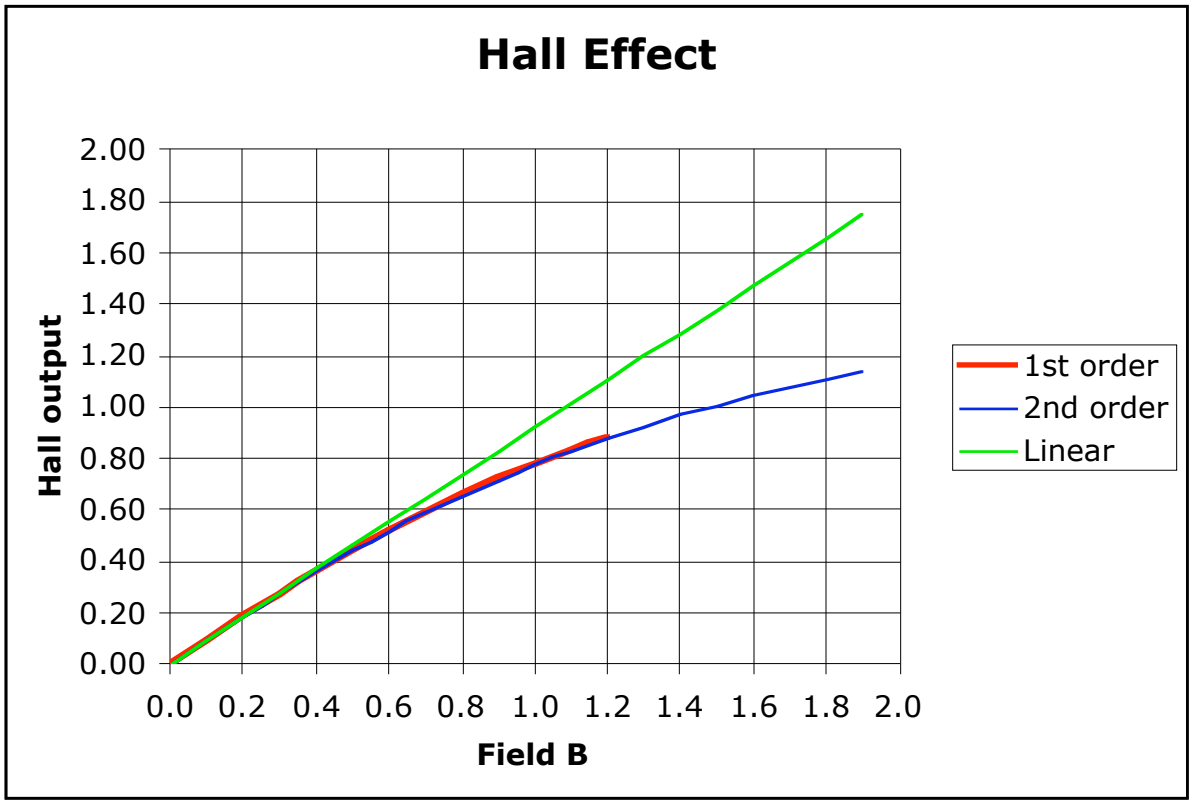

Figure 3: Relaxation Results

5 Direct solution

Going beyond the results of Section 4 requires a direct solution of the set of linear equations, rather than the iterative solution used until now. This much was realised in 1964, but the

limited capacity of the computer meant that only about 40 equations could be solved. Even with symmetries about the midline and recognising that the end-points have constant values, the largest plate, about 13×6 (a 6×6 grid), was barely adequate for a sensible solution.

But things have changed since then. The total capacity of primary memory has grown from 2000 real numbers to 32 million (on a 256 Mbyte machine), and multiplication times have decreased from 10 ms (yes, 100 products per second, or 20 divisions) to little more than 10 ns. It is the speed increase of 6 orders of magnitude that is the most important.

If we multiply the plate linear dimensions by n, the number of grid points and therefore the number of equations changes as n^2. But the work to solve a set of m linear simultaneous equations varies as m^3 and therefore as n^6 in our example here. A speed improvement of 10^6 is entirely consumed by a 10-fold increase in plate size or grid resolution! The corresponding increase in storage for the coefficients (to 90 megabytes in the largest example) is insignificant in comparison.

5.1 The Equations

We illustrate with a small example of 4 rows by 5 columns, shown in Figure 4. There are 20 points over the grid and therefore a 20×20 matrix, the points numbered according to a raster scan, left to right and top to bottom. Each matrix row, in this order, corresponds to the equation defining one point and, within that row, each column corresponds to the coefficient of another point in the equation.

$$\begin{array}{c|c|c} & \begin{array}{|rrrrr|} \hline 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ \hline \end{array} & \\ V_{in} & & V_{in} \\ = -1 & & = +1 \end{array}$$

Figure 4: The "raster scan" over the Hall effect plate

Figure 5 shows the complete matrix, and right hand side, for the 4 rows by 5 columns of Figure 4. The parameter B is equal to 2 for this example.

Thus the first matrix row corresponds to the "top left" point of the grid, with a constant value of -1 (in the right hand side). In the 5th matrix row, the 5th column similarly defines the right hand edge with a constant potential of $+1$. The intervening 3 matrix rows generate the "top edge" pattern according to the earlier equation, (remember $B = 2$)

$$3\phi_{i,0} - 4\phi_{i,1} + \phi_{i,2} + 2\phi_{i+1,0} - 2\phi_{i-1,0} = 0$$

Matrix rows 6–10 and 11–15 define the left end, three internal points, and the right edge for the two interior rows of grid points. Matrix rows 16–20 reflect the first four, for the bottom edge.

Results from explicitly solving the equations are shown in Figure 6. Results from the relaxation solution coincide with the equations, until instability arises

	Row 1					Row 2					Row 3					Row 4					RHS
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
2	2	3	-2	0	0	0	-4	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	2	3	-2	0	0	0	-4	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	2	3	-2	0	0	0	-4	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
7	0	1	0	0	0	1	-4	1	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	1	-4	1	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	1	0	0	0	1	-4	1	0	0	0	1	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1
12	0	0	0	0	0	0	1	0	0	0	1	-4	1	0	0	0	1	0	0	0	0
13	0	0	0	0	0	0	0	1	0	0	0	1	-4	1	0	0	0	1	0	0	0
14	0	0	0	0	0	0	0	0	1	0	0	0	1	-4	1	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-1
17	0	0	0	0	0	0	1	0	0	0	0	-4	0	0	0	-2	3	2	0	0	0
18	0	0	0	0	0	0	0	1	0	0	0	0	-4	0	0	0	-2	3	2	0	0
19	0	0	0	0	0	0	0	0	1	0	0	0	0	-4	0	0	0	-2	3	2	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Figure 5: Equations for 4 rows, 5 columns

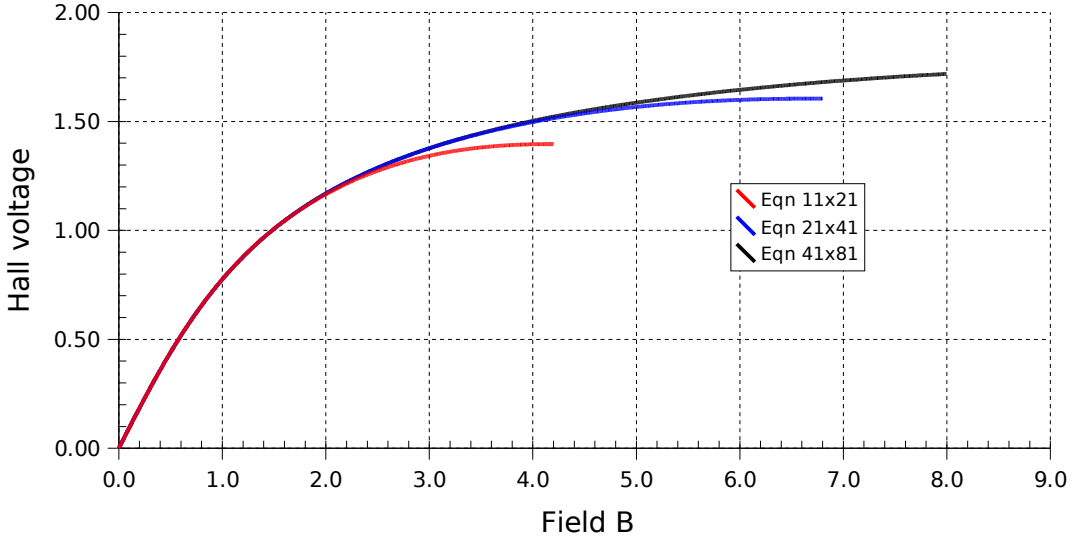

Figure 6: Results from direct solution of the mesh equations

Equations 11×21 ($11 \times 21 = 231$ equations.) It agrees with the other results for $B < 2.5$ and then continues to flatten off as expected, but perhaps a little too quickly when compared with the last result.

Equations 21×41 (861 equations) This is the finest grid that is reasonable for most computers, with each point taking about 8 seconds on a 2,5GHz computer. It agrees with the previous results up to about $B = 2.0$

Equations 41×81 (A system of 3321 equations, with 11,029,041 coefficients and requiring nearly 89 Mbytes.) This is the finest practicable grid, taking about 8 minutes per point on the fastest available computers (3.5GHz Pentium, and 1.8 GHz iMac G5). The complete solution requires about 100 points, or 13 hours CPU time. (Remember that the time scales as the 6th power of the grid resolution, so that doubling the resolution multiplies the solution time by 64. A further doubling would need about 5 CPU-weeks.) It agrees with the 21×41 solution up to about $B = 4.0 - 4.5$.

The coarser grid solutions drop off rather sooner and to a lower asymptote than those with the finer grids.

5.2 The equipotentials

Figure 7 shows the equipotentials for $B = 1.0$. The end-points have potentials of $-1V$ and $+1V$ and potentials are shown at $0.1V$ intervals. Even at this relatively low field there is considerable crowding of equipotentials at the top left and bottom right corners, leading to considerable errors in the finite-difference approximations to the partial derivatives. Some of the values in these corners are indeed seen to be quite anomalous if the detailed potentials are examined for high fields. It is the back-propagation of these errors which probably leads to the earlier drop-offs for the coarser grids in Figure 6.

These problems can be reduced by choosing better approximation formulæ(those used have $O(h^4)$ errors in conventional terms) or by using a variable resolution with finer spacing in critical areas. Neither improvement seemed to be justified here.

5.3 Numerical errors

Two obvious forms of error are truncation and rounding in the arithmetic, and quantisation from the finite spacing of the grid elements (this discussed already in section 5.2).

All calculations are done in *double* (64-bit) format, with an inherent precision of about 15 digits. Truncation and rounding errors can be estimated from looking at the outputs. Referring back to Figure 4, the pairs (1, 20), (2, 19), (3, 18) and (4, 17) should each be numeric complements from symmetry; each pair should sum to zero, with any deviation indicating the numerical error. The rms error over the two edges is always around 5×10^{-18}, irrespective of the number of equations. (This is when solving the equations using LU

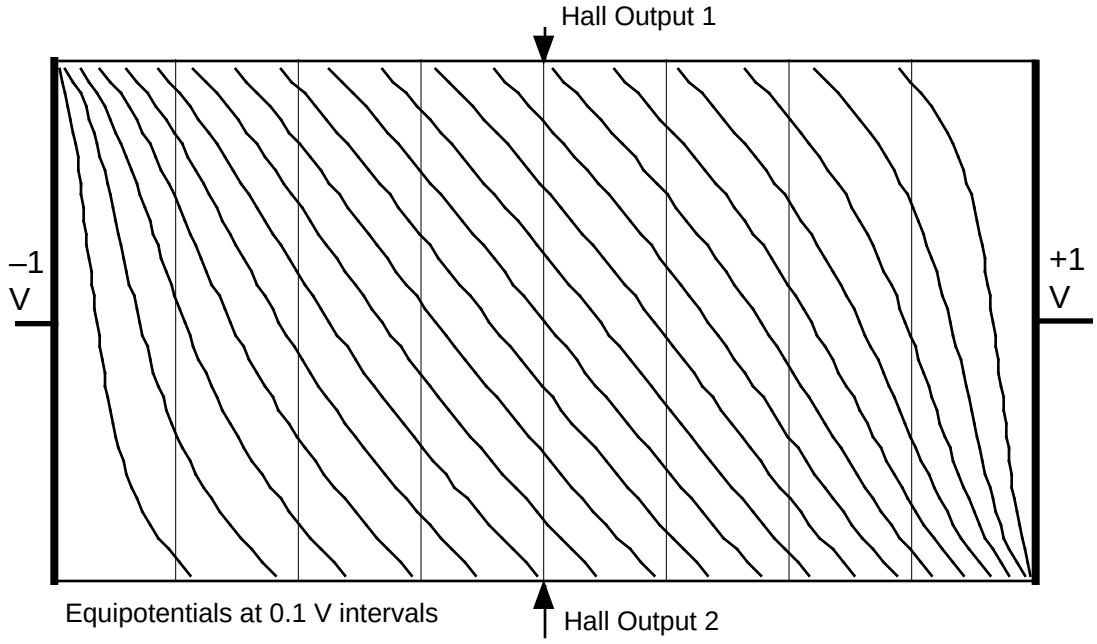

Figure 7: Hall plate equipotentials

decomposition. With the slower Gaussian elimination the RMS error is around 5×10^{-9}, still negligible but far greater than for LU decomposition.)

Truncation and rounding errors seem to be negligible. This is a great contrast to the early work with the 1620 computer which simply truncated floating point results to 8 decimal digits; a solution of the set of 40 equations usually had only about 4 digits of useful precision. Rounding is obviously handled much better now.

Another interesting point is that error values (such as 3.32194696e-18) are identical on both the iMac G5 processor and the Pentium 4. Floating point arithmetic is essentially identical on the two processors. Perhaps there is something to be said for standards!

6 Of Processors and Compilers

The program was written in C, using the two Numerical Recipes linear equation routines `gaussj` (Gauss-Jordan elimination) and `ludcmp/lubksb` (LU decomposition). It was run on several different systems as shown in Table 1 (sometimes with only a few tests on a system).

Columns give the processor, its clock speed (for what that's worth), the compiler, optimisation level, algorithm (Gauss-Jordan or LU decomposition) and the solution time for one set of equations. All systems had at least 256MB RAM; cache characteristics were ignored.

The comparisons are both expected and unexpected. LU decomposition is generally faster

than Gauss-Jordan, but not always by the expected factor of 3. Aggressive optimisation (–O3) gives up to twice the speed of minimal optimisation (–O0), but sometimes much less improvement. The raw clock speed is a poor predictor of speed.

The final row shows the times for the largest problem, with 3,321 simultaneous equations.

processor	clock MHz	compiler	opt lvl	eqn. alg	time sec	
iMac G4	700	gcc	-O0	GJ	80	
iMac G4	700	gcc	-O3	GJ	37	
Celeron	735	codewarrior	-O0	GJ	87	
Celeron	735	codewarrior	-O4	GJ	66	
Celeron	735	cygwin/gcc	-O	GJ	86	
Celeron	735	cygwin/gcc	-O3	GJ	68	
Pentium 4	3.5G	gcc	-O0	GJ	13.5	
Pentium 4	3.5G	gcc	-O3	GJ	8.4	
iMac G5	1.8G	gcc	-O3	GJ	7.9	
Athlon	2.2G	gcc	-O3	GJ	9.3	
iMac G4	700	gcc	-O0	LU	46.4	
iMac G4	700	gcc	-O3	LU	35.5	
Celeron	735	codewarrior	-O4	LU	8.5	
Celeron	735	cygwin/gcc	-O3	LU	13	
Celeron	735	cygwin/gcc	-O	LU	19	
Pentium 4	3.5G	gcc	-O0	LU	4.9	
Pentium 4	3.5G	gcc	-O3	LU	2.8	
Pentium 4	2.2G	gcc	-O3	LU	320	41×81 grid

Table 1: Solution times , mostly 21×41 grid

7 Final Comments

This work has been rather nostalgic, this one problem bracketing much of my career. Although the general methods of solution were obvious so long ago, and the numerical techniques were well known, it has taken four decades of technical progress to be able to actually solve it.

Apart from the lack of memory, all of the programs could have been written for and run on the IBM 1620 — perhaps not so easily on the original Fortran-with-Format, but certainly a little later when Fortran II introduced user-written subroutines. Assuming that the 11 million coefficients could be stored, the solution of 3300 linear equations would have taken over 10 CPU-years at the 1620 speed. And those 10 years produce just 1 point; I used 100 of them in the final run!