|epcc|

THE UNIVERSITY OF EDINBURGH

SS-2000-04

Inference of fluid flow at the top of the Earth's liquid core from geomagnetic data

Mark Madden

Abstract

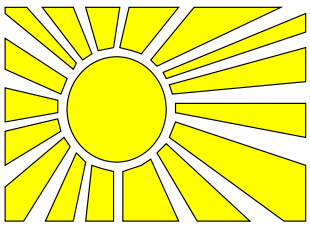
This project concerns the adaptation of code written to calculate flow patterns at the top of the Earth's liquid outer core.

Several stages of work were involved:

Incorporation of the `velnems` routine into the code. This provides an alternative method of assembling what are known as the equations of condition, by far the most computationally expensive stage of estimating the flow.

Parallelisation of this code.

Investigation of alternative methods to determine the flow model that best fits the data.



Contents

1 Introduction

The geomagnetic field can provide much information about the physical processes which occur deep within the Earth. For instance, taken together with its time evolution (secular variation or SV), it may be used to estimate flow patterns at the top of the liquid core. We shall see that although these estimates must assume a significant amount, they can prove quite accurate in modelling the secular variation and other phenomena.

We begin with a whistlestop tour of the inner Earth, which consists of three main layers (Figure 1). Innermost is the solid core extending to a radius of about 1200km. Following this is the liquid section which has a radius of 3485 km. The third, outermost part is the mantle, a broadly solid bed upon which the oceans and continents sit. The mantle may rotate at a different speed to the core, and indeed the two may exchange angular momentum. It is believed that this type of exchange is the main agent responsible for decadal changes in length of day (ΔLOD); the time taken for the mantle to make a complete revolution. ΔLOD has been observed over many timescales, shorter-term fluctuations being attributed to angular momentum exchange between the mantle and the atmosphere, or the oceans. There is som interest in finding flow models which may be able to explain both decadal ΔLOD and geomagnetic observations.

The task of finding a flow that fits such observations is an example of an *inverse problem*. There is also a corresponding *forward problem* of predicting, in this case, what SV a given flow will generate. This forward problem is relatively simple. Going in the other direction is not as easy. We shall illustrate this by describing the broad strategy applied in this particular code.

1.1 Physical model used by the code

Using spherical polar coordinates, we begin with the magnetic induction equation:

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}) + \eta \nabla^2 \mathbf{B} \tag{1}$$

with η a constant known as the magnetic diffusivity, $\mathbf{B}$ the magnetic field and $\mathbf{u}$ the velocity field we seek. Both of these vector quantities depend on time in the general case, but for now our notation avoids any explicit statement of this.

It turns out that we may only use the induction equation to calculate flow at the top of the core, at the core-mantle boundary (CMB). This is so because, to determine flow inside the core, we would need information about the magnetic field inside it. Unfortunately, our observations from Earth's exterior only reflect the field at the the core surface. This makes it hard to say anything about radial flow simply by looking at the data - we are constrained to movement on that surface. This is simply something that must be accepted.

First, because the magnetic diffusivity is small, we make the *frozen flux approximation*, reducing Equation (1) to

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}). \tag{2}$$

Physically, this may be interpreted by saying that the magnetic field is 'frozen into' the fluid and that it is carried along by the flow.

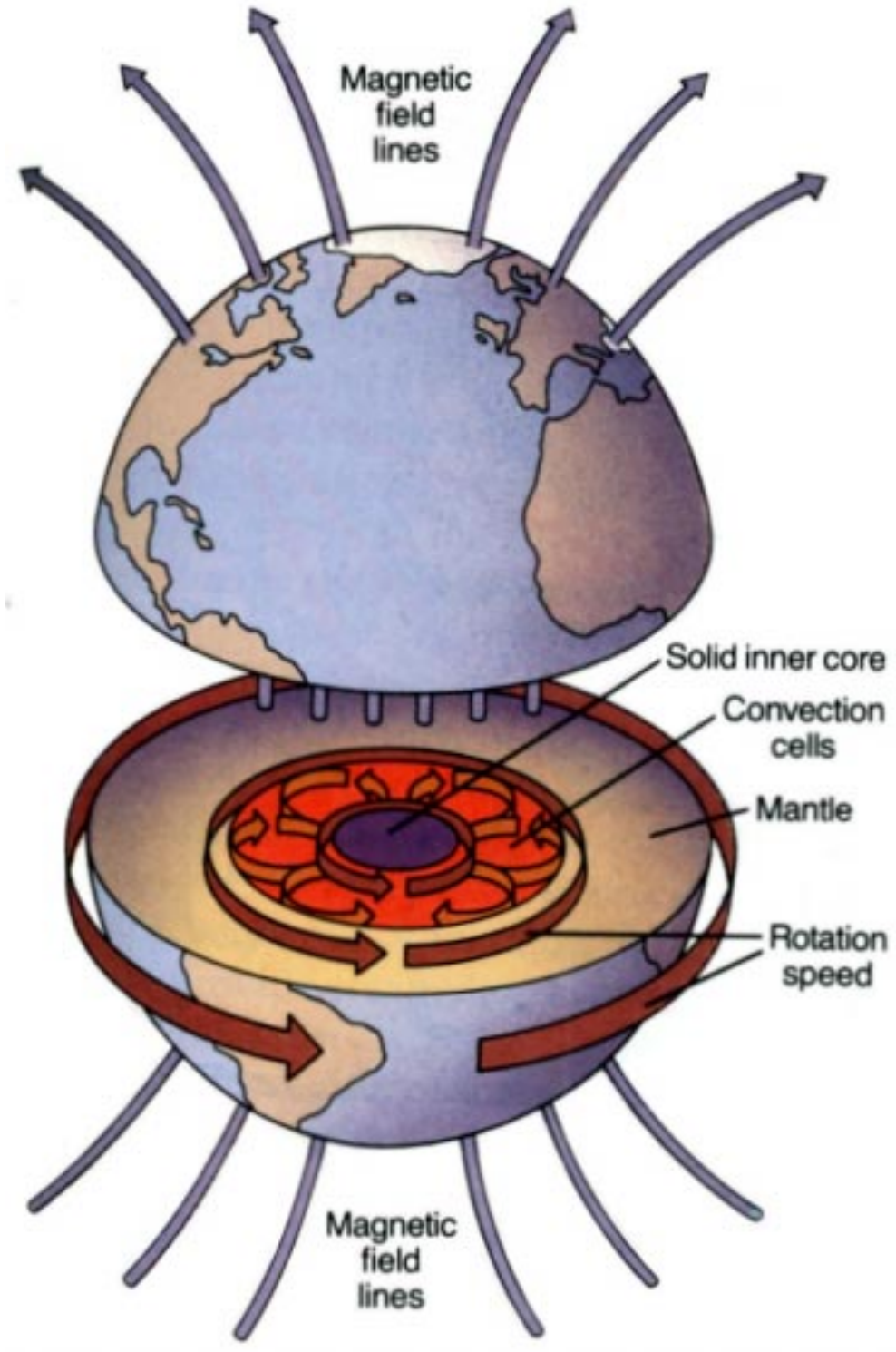


Figure 1: The Earth. Diagram taken from page 451 of Earth's Dynamic Systems, written by W.K. Hamblin and E.H. Christiansen.

Next, we consider only the radial component, which states

$$\frac{\partial B_r}{\partial t} + B_r \nabla_H \cdot \mathbf{u} + \mathbf{u} \cdot \nabla_H B_r = 0. \tag{3}$$

It might seem at this point that we have thrown away essential information by considering only one part of Equation (2). One way of seeing this is to note that, using this one equation, we wish to solve for the two 'horizontal' components of the surface flow. In actual fact the problem cannot be solved uniquely in any case; it can be shown that considering other parts of Equation (2) generally tends to introduce more unknowns than it resolves (see, for example, page 101 of [3]). Other methods must be used to address this inherent non-uniqueness and we shall come to them shortly.

Returning to Equation (3), we now perform some rather involved algebraic manipulation, expressing all vector quantities in terms of their toroidal and poloidal defining scalars (page 99 of [3]) and expanding these in spherical harmonic infinite series which must clearly be truncated. After a little more work, we eventually arrive at a set of simultaneous matrix equations covering the time period of interest. Each equation, known as an *equation of condition*, represents a specific instance of (3) at a particular time point t_i, i running from 1 to some arbitrarily chosen N:

$$\begin{aligned} \dot{\mathbf{g}}(t_1) &= \mathbf{A}(t_1)\mathbf{m}(t_1) \\ \dot{\mathbf{g}}(t_2) &= \mathbf{A}(t_2)\mathbf{m}(t_2) \\ &\vdots \\ \dot{\mathbf{g}}(t_N) &= \mathbf{A}(t_N)\mathbf{m}(t_N). \end{aligned} \tag{4}$$

Here, $\dot{\mathbf{g}}(t)$ is a column vector of SV coefficients, with $\mathbf{A}(t)$ a matrix known as the condition matrix. The shape of $\mathbf{A}$ is determined by the truncation level of the SV and velocity field and it is by no means certain that it is square. The last term, $\mathbf{m}(t)$, is a vector of coefficients storing an arbitrary surface velocity field.

Now, by introducing assumptions about the nature of this velocity field, it is possible to lift the non-uniqueness encountered earlier. In this code, there is something of a hierarchy of assumptions ordered by complexity, each building on the first:

- The simplest choice is to require the flow to be steady, completely unchanged over whatever period data are taken from, be it ten years, eighty years or a hundred and fifty. This assumption is attractive because of its simplicity and it can account for the gross scale of the SV.
- The next option is to extend the steady drift idea through the introduction of one extra input parameter, a constant 'drift velocity'. This choice allows the core to turn about the Earth's axis of rotation at a different rate to the mantle, so that the steady flow on the CMB now appears to drift around at a steady rate as the mantle watches it. In other words, observers on the mantle see a flow that has a specific kind of time-dependence. For the slightly greater cost in complexity, this achieves a much better fit to the finer detail of the SV.

- The most complex extension comes from allowing this input drift to be time-dependent rather than steady. On the basis of improving the fit to the SV, this option is hard to justify because it does not give a very significant improvement. However, with a steady flow, a time-dependent drift is the only type that allows for ΔLOD considerations: only if the rate of rotation of the core with respect to the mantle is allowed to vary may the two exchange angular momentum. Parts of the code are designed to examine such effects, because ascribing ΔLOD effects to a time-varying drift remains a somewhat contentious issue and one worthy of study. Length-of-day investigation was not, however, the thrust of this project.

With non-uniqueness taken care of in one of the ways described above, the ensemble of equations (4) can then be solved. It is important to notice that, whatever type of drift is assumed, the code will always find a steady flow solution - the steady flow that best fits the constraints of *all* equations in (4). Thus, for example, the flow that is best found to describe the period 1900-1980 will not, in general, be the best fitting flow for the time period 1950-1960, say.

The solution proceeds by first accounting for the drift angle at each timepoint. This is done by changing the frame of reference of equations (4) from the mantle to the core, so that the flow now appears the same at all timepoints:

$$\begin{aligned} \dot{\mathbf{g}}'(t_1) &= \mathbf{A}'(t_1)\mathbf{m} \\ \dot{\mathbf{g}}'(t_2) &= \mathbf{A}'(t_2)\mathbf{m} \\ &\vdots \\ \dot{\mathbf{g}}'(t_N) &= \mathbf{A}'(t_N)\mathbf{m}. \end{aligned} \tag{5}$$

Here, primes denote that we have 'rotated' our SV coefficient vectors and our condition matrices into the drifting core reference frame.

Now, with $\mathbf{m}$ the same for each timepoint, we may condense equations (5) into one:

$$\dot{\mathbf{G}} = \mathcal{A}\mathbf{m} \tag{6}$$

Where $\dot{\mathbf{G}}$ stores all our rotated SV coefficients and $\mathcal{A}$ contains the rotated condition matrices.

We next multiply Equation (6) by the transpose of $\mathcal{A}$, and in so doing turn the condition equations into their corresponding *normal equations*.

$$\mathcal{A}^T\dot{\mathbf{G}} = \mathcal{A}^T\mathcal{A}\mathbf{m} \tag{7}$$

This is desirable because $\mathcal{A}^T\mathcal{A}$ is guaranteed to be square, symmetric and *positive definite*, making it is possible to solve Equation (7) by standard methods.

1.2 Assessing flow solutions

Once the code has a solution, it is useful to be able to assess it so it may be compared with other flow solutions obtained using, for example, a different drift rate. In this code, such an

assessment is made using the following *objective function*, a pure number whose size reflects the relative favourability of a given flow solution:

$$\mathcal{O} = \int_{t_0}^{t_1} [\int_{r_e} (\dot{B}_r - \dot{B}^0{}_r)^2 \, dS + \lambda_d (\ddot{\psi})^2] \, dt + \lambda_v N. \tag{8}$$

This equation is given for completeness; it suffices to note that our objective function is in fact a linear combination of three important contributors to flow solutions:

- The surface integral over the CMB, $\int_{r_e} (\dot{B}_r - \dot{B}^0{}_r)^2 \, dS$, measures the mean square misfit between the radial component of the SV predicted by the flow solution in question and that observed.
- The term $(\ddot{\psi})^2$ measures the drift angle acceleration. Physically, we do not expect that having the core oscillate wildly will represent a terribly favourable solution and this term imposes such a constraint. The damping parameter λ_d reflects how much relative importance we wish to attatch to this.
- N is the *flow norm*. Crudely, this is a measure of the steady flow's complexity or its strength. Again from a physical standpoint, we expect that flows with a higher flow norm will be less attractive than others. This ingredient is weighted with the damping parameter λ_v, which is also used to influence the complexity of the flow obtained by the code in the first place. There is something of a trade-off between damping the flow and fitting the data; a 'smoother' flow may have a higher misfit than a more turbulents one.

Our goal is to minimise $\mathcal{O}$ for the time period of interest. There is clearly much scope for doing this; simply changing the value of, say, λ_v will change not only $\mathcal{O}$ but also the flow upon which it in part depends. In later parts of this project, the general emphasis rested on examining how $\mathcal{O}$ depended on drift rate, with other parameters fixed.

2 Work undertaken

As briefly described, work on this project saw a number of stages. We now discuss these in some detail.

2.1 Decreasing run-time

It was requested that the first stage of work be concerned with making the initial Fortran code run faster. This was accomplished by providing a swifter routine for assembling the equations of condition, and then performing a relatively trivial parallelisation.

2.1.1 Incorporation of `velnems` routine into the code

Examining the profile obtained for a simple run of the code as it was initially presented (Table 1), we see that the great majority of the run-rime is spent inside subroutine `nmlvsub` or related routines `thrj` and `gntels`. All of these are concerned with forming the normal equations. The subroutine `gntels` is called by `nmlvsub`, and in turn calls `thrj` several times.

Routine	% time	time (s)
nmlvsub	42.8	258
thrj	39.6	239
gntels	5.3	32
Total	87.7	529
Program run-time:		603

Table 1: Three of the top four contributors to run-time

Accordingly, as an alternative option to calling nmlvsub, the functionally equivalent routine nems3 was added:

```
#ifdef RICHORIG
        call nmlvsub(e,f,ndata,time,ssr,grot,svrot,lmfin,lsvin,ivar,
     &lsvin,gerr)
#else
        lsvir = lsvin
        LSVIR=MIN0(LSV,LSVIR)
        NSV=LSVIR*(LSVIR+2)

        call nems3(e,f,ndata,time,ssr,grot,svrot,lmfin,lsvin,ivar,
     &lsvir,nsv,lsvin,gerr)
#endif
```

The #ifdef...#endif that encloses the above fragment is an fpp (Fortran Pre-Processor) construct, allowing one to choose at compile-time which clause should be executed. See [2] for more details.

The subroutine nems3 was taken from an earlier code, ITERTRAN, written for much the same purpose as the current one (for details, see [2], [5] and [4]). In the course of work performed in [2], velnems was converted from a standalone program to a subroutine called in nems3, and also optimised somewhat. It calculates the condition matrices by means of a spherical transform approach.

Using the optimisation options provided by that work, a simple run of the newly modified code was found to run at 77.7% of the time taken by the original code under the same conditions. The results it gave were in agreement with the original, after a conflict involving normalisation of input data was eventually resolved.

2.1.2 Parallelisation of the code

The other way in which overall runtime was decreased was through the use of parallelism in the assembly of the normal equations. In the code, this is performed as a loop over the timepoints t_i in which, among other things, nmlvsub or nems3 is called as in the code fragment above. It follows from Table 1 that this is the computationally dominant loop.

Now, calculations for each timepoint are independent of one another, making it possible to split the overall time interval into a number of sub-intervals and then to assign each sub-interval to a separate processor. Once each processor has calculated its share of the ensemble of equations (7), the whole ensemble is broadcast to every processor. A given processor holds its part of

equations (7) in two arrays, one storing $\mathcal{A}^T\mathcal{A}$ and the other containing $\mathcal{A}^T\dot{\mathbf{G}}$ for that processor's time interval. The relevant contributions from each time point are summed into these arrays.

Note that, before and after calculation of the normal equations, each processor possesses identical data so at the beginning and end of execution we are in fact running multiple copies of the same program.

This is acceptable since so much of the run-time is spent inside this one loop. Parallelisation of the code outside this loop was deemed to be not worth the effort at the current time.

Parallelisation was implemented using the *Message Passing Interface* (MPI) standard. Figure 2 shows how the parallel code scales with the number of processors.

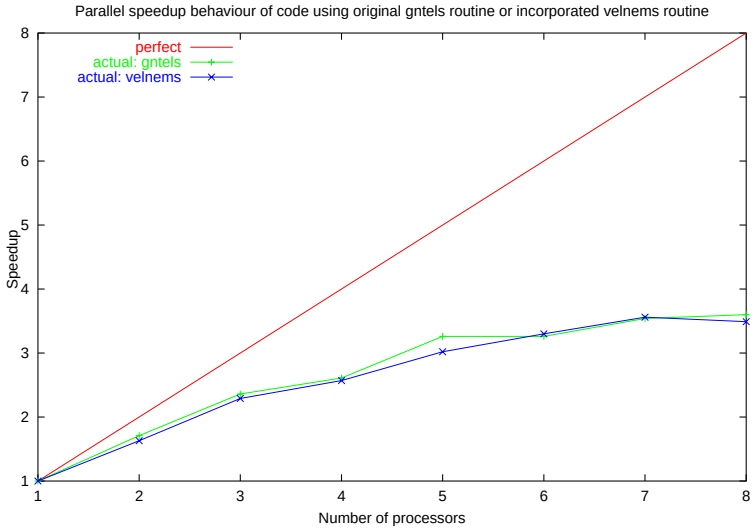


Figure 2: Relative parallel performance of the code.

It can be seen from this that the speedup, while acceptable, is somewhat less then ideal. Two main reasons might be attributed to this:

- Without further parallelisation effort, there will *always* be at least a 10% proportion of the code that is entirely serial. In accordance with *Amdahl's Law* ([1]), the code can only ever go as fast as this 10%.
- As we increase the number of processors, we increase the amount of communication work that must be done. For instance, the array in the code that stores the matrices $\mathcal{A}^T\mathcal{A}$ for a particular processor's sub-interval consists of over one hundred thousand double precision numbers. It does not seem unreasonable to suppose that communications involving this will become a significant overhead.

2.2 Investigation of steady drifts

With a reasonable parallel code obtained, attention turned to methods for attempting to determine what drift would serve to minimise $\mathcal{O}$. The cases of both steady and time-dependent drifts were examined.

We first consider how changing the value of a steady drift affects $\mathcal{O}$. Investigation of this involved the addition of a new option to the code, that of scanning through a given range of drifts. For each drift velocity, the code would find its flow solution and calculate $\mathcal{O}$. The drift velocity would be incremented and the calculation repeated until the interval had been covered. Figures 3 and 4 illustrate the results obtained.

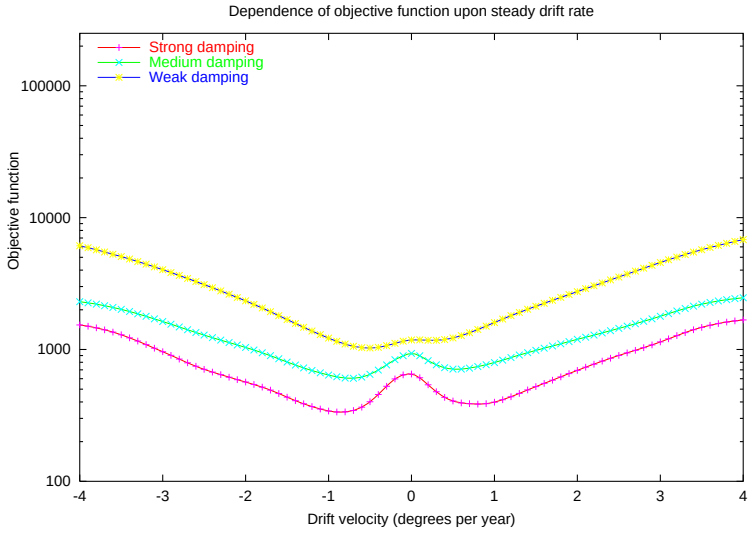


Figure 3: Plot of $\mathcal{O}$ as a function of 81 uniform drift values ranging from -4 degrees per year to $+4$ degrees per year (the eastward direction is defined to be positive). Three curves are plotted; 'Strong damping' ($\lambda_v = 2.0 \times 10^{-3}$), 'medium damping' ($\lambda_v = 2.0 \times 10^{-4}$) and 'weak damping' ($\lambda_v = 2.0 \times 10^{-5}$)

Figure 4 actually reproduces a little of the work in [6]. This was taken to be a useful check that the code had not been 'broken'. Figure 3 was intended to complement that work by determining if any minima were to be found outside the region $[-1.6, 1.6]$ degrees per year. We see that this is not the case unless all such minima happen to occur for slightly different values of steady drift than were sampled. Under the assumption that none exist, we may extrapolate the smooth curves underlying Figures 3 and 4.

2.3 Investigation of time-dependent drifts

Steady drifts are attractive because, given an initial value, we need only one other number to describe the drift angle for all time: a constant drift velocity.

We now consider more general time-dependent drifts which clearly cannot be represented by one single velocity. In order to do this, we must first discuss how the code actually deals with drifts.

2.3.1 Spline interpolation

In order to transform our equations of condition (4) into the drifting reference frame, we need to know the drift angle and drift velocity at each timepoint. A very simple solution to this

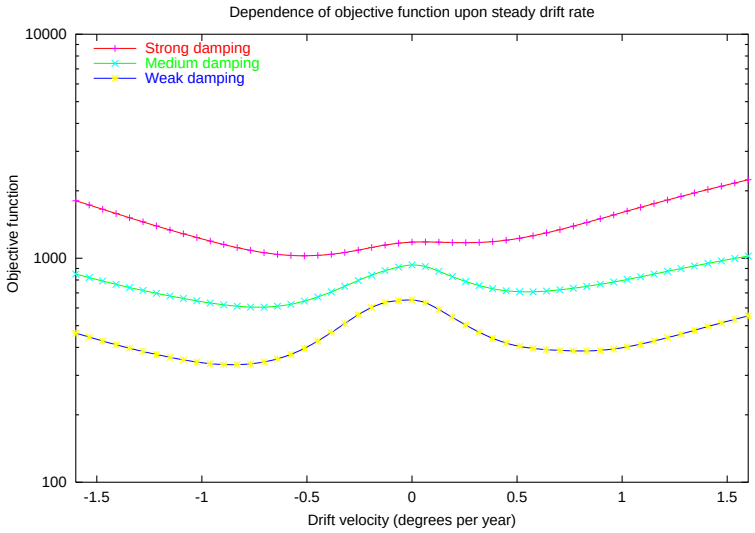


Figure 4: Plot of $\mathcal{O}$ as a function of 51 uniform drift values ranging from -1.6 degrees per year to $+1.6$ degrees per year. All other parameters are the same as Figure 3; this figure represents a slightly more detailed scan of the apparent region where $\mathcal{O}$ has its minima.

problem would be to store a look-up table of the necessary values; however this is clearly rather inflexible. For instance, if we decide that we wish to change the spacing of the timepoints, we must recalculate our table. It is thus desirable to have some way of specifying a drift curve which can tell us the drift angle and velocity at any time.

Here, this is handled by *spline interpolation* (Figure 5). Such an approach actually deals with steady and fully time-dependent drifts in exactly the same way, since a constant drift velocity is merely a special case of time-dependence for which the drift curve is simply a straight line.

The method proceeds as follows. We first specify an initial dataset which gives values of the drift angle [1] at an arbitrary number of evenly spaced times (the + signs in Figure 5). This dataset is passed to a section of the code which calculates a number of *spline coefficients* representing a continuous curve which passes as near as it can to each original point of the dataset. In our case we obtain 22 such coefficients and these allow us to extrapolate the required drift angle (the $\times$ signs) and drift velocity for each timepoint of our equations of condition (4). It is important to realise that, as far as the calculation of equations (5) is concerned, the code only 'sees' these extrapolated points - the continuous curve is drawn for illustrative purposes.

Note that as Figure 5 shows it, the times at which the initial datapoints occur coincide with the times at which the drift angle and also the drift velocity are calculated. This choice was made for simplicity; there is no requirement for the two to coincide. The choice of data point spacing (or more precisely the number of points in our time period) will prove to have an interesting effect later.

Finally, notice that the curve does not always pass through the data points specified; some 'smoothing' has occurred. Again, this will become significant later.

[1] Alternatively, we may specify a velocity dataset; the interpolation code can handle either. In either case, the end result is an interpolated drift *angle* curve. This is chosen to be so because, to get the drift velocity, we have only to differentiate this curve, which is an easy task. Earlier work using steady drifts, such as that in ITERTRAN, stored the drift velocity and had to integrate this to get the corresponding drift angle. This is a less attractive option.

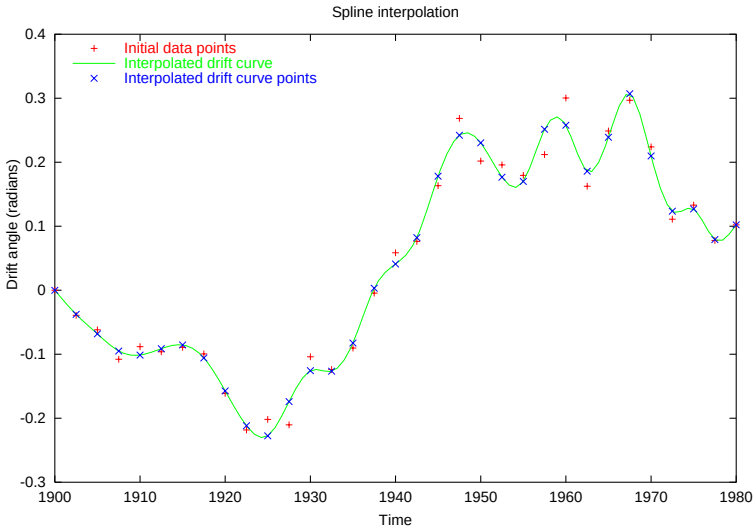


Figure 5: Spline interpolation. We specify an initial dataset consisting of, in our case, values of the drift angle at various times. It is then possible to obtain a set of *spline coefficients* which describe the smooth curve from which we obtain our actual drift curve points.

We now move to discuss a number of ways in which it was attempted to minimise $\mathcal{O}$ by varying the shape of the drift curve. Broadly, two approaches were taken: a *Monte Carlo* approach in which lots of pseudorandom curves were tested, and a *simulated annealing* method in which a number of pseudorandom curves were generated and then perturbed repeatedly. In all cases, for convenience the drift angle was defined to be zero at the start time, fixed for this study to be 1900.

This work complements previous minimisation efforts which used *Powell's method* ([6], [7]). The reliance upon pseudorandom trials reflects an attempt to sample more varied drifts than perhaps Powell's method examined. The reader may judge how successful this attempt was.

2.3.2 Monte Carlo sampling of drift curves

The philosophy behind this approach is quite simple. An initial dataset was generated randomly (for routines used, see [8]) subject to some constraints and the corresponding drift curve interpolated. The value of $\mathcal{O}$ for this drift was calculated and the process repeated.

Two methods of dataset sampling were tried, in which the constraints placed on the generation of the drift curve were different.

The first method of generation involved choosing sequential random displacements for the time points. In other words, we would begin with the first point after the start time, choosing for it a normally-distributed random 'jump', move to the next time point, select how far to jump away from the previous drift angle and carry on until the end time was reached. The distribution was chosen to have zero mean and a variance that reflected the judgement that a drift angle jump of anything more than 4.5 degrees per year was unphysical and should be made highly improbable. This condition was empirically verified when a run was made with the tolerance level set at 20 degrees per year. The run produced objective functions about an order of magnitude larger.

Figure 6 gives the drift curve which yielded the lowest value of $\mathcal{O}$ found by this sampling method when the number of data points coincided with the number of condition equation time points. 2000 trials were tried, using a medium-damped flow.

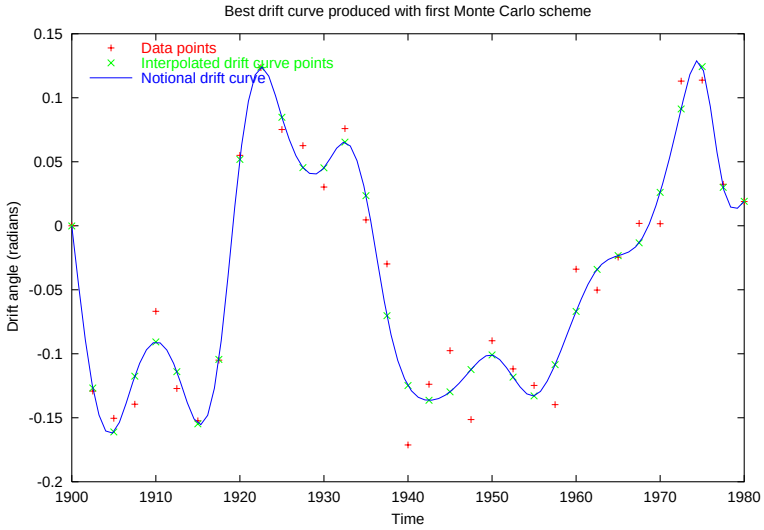


Figure 6: Best random drift curve generated by first Monte Carlo sampling method with 33 data points - the same as the number of condition equations. This curve yielded a value for $\mathcal{O}$ of 36387.

We see from Figure 6 that the performance of this method is not all we might wish for. One possible explanation is that, as can be seen, this way of sampling allows the drift angle to change direction extremely frequently. Perhaps it is possible that this rapid oscillation does not allow for a very good fit to the data - certainly it is the misfit term in Table (2) which is dominant.

Misfit to observed SV	Damped drift norm	Damped flow norm	Total $\mathcal{O}$
.755E+05	57.3	267	75781
.358E+06	250	.485E+04	363253
.652E+05	34.1	.136E+04	66593
.202E+06	170	743	202609
.110E+06	83.2	533	110642
.123E+06	43.1	.190E+04	124771

Table 2: Contributions to objective function for a randomly selected sample of results generated by the first Monte Carlo scheme.

It was therefore decided to try another drift curve sampling technique which chose ‘jump’ displacements as before, but which this time enforced a direction with a given probability of a change in direction at every timepoint. Two results are shown in Figure 7.

As can be seen, this drift curve selection gives rather better results. Two things are notable about Figure 7:

- The number of initial dataset points was set to be 200 for this set of results. This gave a substantially lower value of the objective function than other runs using 33 dataset points. This effect was also noticable when the first Monte Carlo scheme was retried in a similar

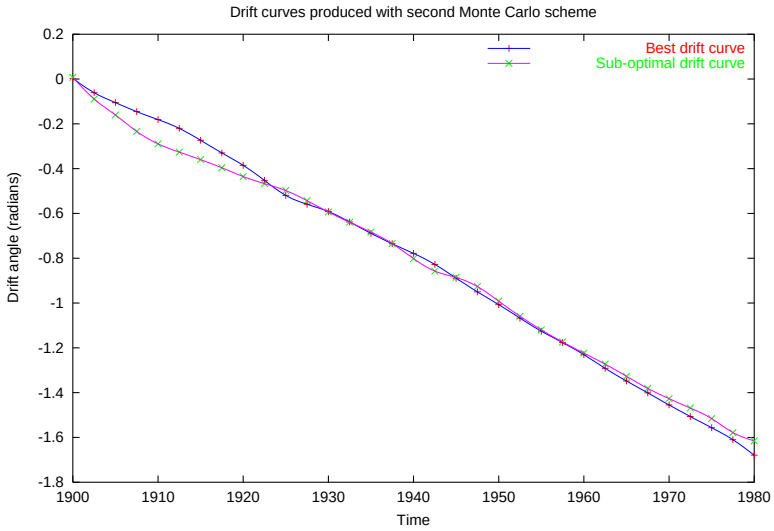


Figure 7: Best random drift curve generated by second Monte Carlo sampling method along with a worse curve for comparison. In this figure, the drift was strongly damped and the number of initial dataset points was set to 200. The best curve yielded a value for $\mathcal{O}$ of 5998; the other shown here returned a value of 16290.

way.

- It appears that the objective function is quite sensitive to the precise shape of a given drift curve. There is apparently not a great deal of difference between the two curves shown, yet their respective objective function values differ significantly. Perhaps this effect arises from the differences in the drift acceleration of the two curves, seen only subtly in the figure.

However, it seems likely that we are constraining the sample variety rather too much in this case - the type of drift curve selected with this approach is quite restricted. Ideally we should like to admit more varied kinds of drifts. With such a desire in mind, we turn to the other method used in this part of the project.

2.3.3 Simulated annealing of drift curves

The approach taken here built on the earlier method of random curve generation. A random dataset would be created using the first Monte Carlo scheme and its corresponding objective function value found. Then, however, a randomly selected point in this dataset would be displaced up or down subject to the condition that its new position should not violate the same '4.5 degree per year' constraint as before. The new value of $\mathcal{O}$ would be calculated. If the perturbed dataset was found to have a reduced $\mathcal{O}$ it would be stored over the old one, and a new perturbation applied to it. This sequence would repeat up to some arbitrary limit, whereupon a completely new dataset would be generated and a similar process repeated.

Now, if a perturbation were found to have increased $\mathcal{O}$, it would not necessarily be doomed to rejection. Rather, its probability of being accepted regardless was defined to be

$$p(acceptance) = e^{(\frac{-\Delta\mathcal{O}}{T})} \tag{9}$$

where $\Delta\mathcal{O}$ is given by subtracting the unperturbed objective function from the perturbed one, and T is a notional 'temperature' which is decreased as the number of perturbations increases.

The idea behind this approach is that, by occasionally accepting an unfavourable change (in our case an increase in the value of the objective function), we may eventually arrive at something approaching a global minimum of our function. For a full discussion, see [7].

Various runs were tried, yielding the following points of interest:

- 'Temperatures' of 2×10^6 or more result in the acceptance of ridiculously unfavourable perturbations, leading to final drift curves whose objective functions were larger than their initial ones. Reasonable minimisation was found to occur for temperatures of the order of 2×10^4 or lower.
- Final results obtained lay somewhere in between those of the first and second Monte Carlo approaches. However, the scale of minimisation achieved could be quite significant, reducing $\mathcal{O}$ by a factor of about 7 in many cases. Such minimisations seemed to take anything up to 4000 perturbations.
- The effect of perturbations on $\mathcal{O}$ seems extremely hard to predict. Some changes resulted in an increase of 10^4 or more, others had little or no effect.

As an extension to this work, an option was added to enable the loading of previously 'minimised' datasets, so they could be put through another run (simply performing a larger run in one go would have been unfeasible owing to availability of computer time). An example of this is the initial dataset shown in Figure 8, to be compared with the same dataset midway through such a 'compound' annealing, after 3200 perturbations (Figure 9). This has the added effect of periodically raising the temperature as a new run begins (although the initial temperatures at the start of later runs were lower compared to earlier initial temperatures).

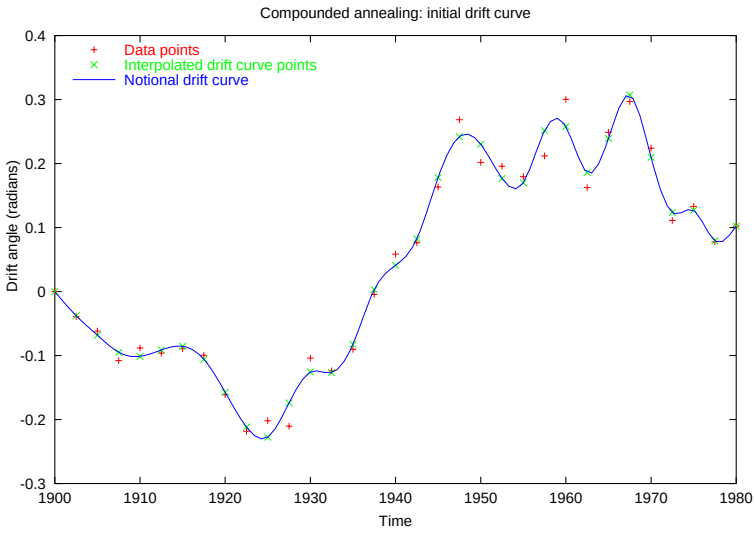


Figure 8: Initial dataset and extrapolated curve before any annealing. This curve gave an objective function of 146925.

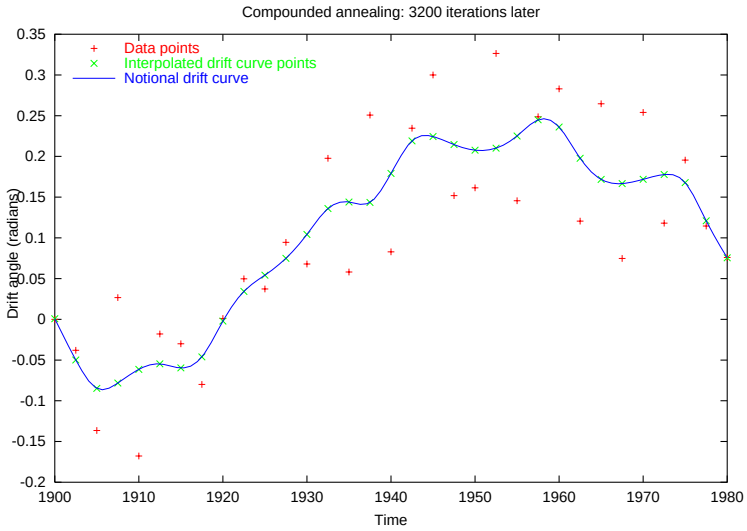


Figure 9: Dataset and extrapolated curve after 3200 perturbations. This curve gave an objective function of 26534.

Note from Figure 9 that the correspondence between individual dataset points and corresponding interpolated drift curve points has to a great extent been lost. Rather, an averaging effect seems to be coming out.

2.4 Further work

It could be inferred from previous sections that later work has so far been somewhat inconclusive. This is true to the extent that attempts to minimise $\mathcal{O}$ were not as effective as might have been hoped. We here outline a number of ways in which the Monte Carlo and simulated annealing methods might be made to work better.

- Instead of splining or perturbing a drift angle dataset, perhaps using drift velocities would have proven superior.
- All perturbations made so far involved moving only one point at a time. Entire sections of the curve could instead be shifted by specifying less dataset points than normal equation timepoints.
- A final thought that occurs is that it might be worth investigating the random generation or perturbation of the spline coefficients, as these are what the earlier minimisation efforts focused upon. Perhaps bringing pseudorandomness to this area would prove fruitful.

3 Conclusion

This report has discussed work performed as part of the SSP 2000. Specifically, we have

- Described relevant aspects of the operation of the original code.

- Outlined changes made to this code.
- Discussed the code's treatment of drift velocities and how steady drifts affect the objective function.
- Outlined a preliminary study of Monte Carlo and simulated annealing methods as applied to our situation. These were found to be less effective than the first approach used by the code, at least at present.
- Suggested how these methods might be altered or improved.

References

[1] G.S. Almasi and A. Gottlieb. *Highly parallel computing*, chapter 1. The Benjamin/Cummings publishing company, second edition, 1994.

[2] R.M. Baxter. ITERTRAN: Analysis and optimisation report. Technical Report EPCC-PG-GLG-2 1.0, EPCC, 1996.

[3] J. Bloxham and A. Jackson. Fluid flow near the surface of Earth's outer core. *Reviews of Geophysics*, 29(1):97–120, February 1991.

[4] R.G. Davis and K.A. Whaler. Determination of a steady velocity field in a rotating frame of reference at the surface of the Earth's core. *Geophysical Journal International*, 126:92–100, February 1996.

[5] E. Degoli. Parallelisation options for the core-mantle boundary flow code ITERTRAN. Project report for the EPCC Summer Scholarship Programme, 1997. Project number EPCC-SSP-1997.

[6] R. Holme and K.A. Whaler. Steady core flow in an azimuthally drifting reference frame. *Geophysical Journal International*, (submitted), 2000.

[7] S.A. Teukolsky W.T. Vetterling, W.H. Press and B.P. Flannery. *Numerical Recipes in Fortran*, chapter 10. Cambridge Universtiy Press, second edition, 1992. Volume 1.

[8] S.A. Teukolsky W.T. Vetterling, W.H. Press and B.P. Flannery. *Numerical Recipes in Fortran*, chapter 7. Cambridge Universtiy Press, second edition, 1992. Volume 1.

Mark is about to enter the final year of his Mathematics with Physics joint degree at York University. The fact that he considers this exciting should probably be regarded as a worrying sign.

Supervisors:
Terence Sloan, EPCC
Dr Rob Baxter, EPCC
Professor Kathy Whaler, Department of Geology and Geophysics

This veritable plethora of supervisors deserve thanks for their time, suggestions and general support, particularly as far as dealing with foolish programming issues, suggesting ways to proceed and repeatedly reading drafts are concerned.