Model Order Reduction for 3D Wave Equation

Victor Pereyra

January 9, 2015

1 Introduction

In recent work we have shown that model order reduction (MOR) is useful for shot interpolation and extrapolation in two-dimensions, acoustic wave equation modeling [5, 13, 25]. The requirement there is accuracy and thus we cannot stray too far from the shots that produced the snapshots for MOR. We have shown that extrapolating one shot in each direction from the basis shot or interpolating two interior shots from two boundary base producing shots works well when the data thus generated is used in an imaging process. Taking into account the overhead, this doubles the speed at which we can perform these simulations, where many thousands of shots and corresponding simulations are required.

The challenge now is to extend this process to three-dimensions in a competitive manner. To fix our ideas, let us consider a typical 3D simulation that is performed today on a single multicore machine with 128 Gb of fast memory:

The number of nodes in each direction of the spatial mesh is $nx = ny = 1541$, $nz = 613$, for a total of $n = 1,455,679,453$ mesh points. That is to say, the size of one snapshot in single precision would be approximately $5.8Gb$. From our experience in 2D and for the type of source frequencies desired, we expect to need of the order of 1000 snapshots, i.e., just to hold a copy of the whole set in core would require $5.8TB$, far more than what is available for a modern single machine. Each simulation takes three hours in Sandy bridge (16 cores, 128 GB memory, 277 GHz). This corresponds to 16" of simulation with a time step $dt = 0.00191$ (9424 time steps).

We observe that a full survey simulation requires many thousands of shots and that is where parallelism in a distributed system is employed. So, that is not the answer for our single shot problem that fits nicely in core for high fidelity simulations.

Currently we have tried two approaches to obtain a well conditioned orthogonal basis for the snapshots: truncated SVD's and a progressive QR algorithm. Both require all the snapshots in fast memory to be competitive. Once an orthogonal basis U is generated we need to project the high

fidelity discretization of the wave equation in order to obtain the reduced order system:

$$a^*_{tt} = U^T AU a^* + U^T \mathbf{b} u(t) - 2U^T D(\epsilon) U a^*_t.$$

Here A is the discrete Laplacian. For our $8th$ order discretization, this is an $N \times N$ sparse, structured matrix, with only 25 nonzero elements per row (in 3D) that is stored in sparse mode. $\mathbf{b}$ is a vector that describes where the forcing function $u(t)$ is applied. For a point source, $\mathbf{b}$ is all zeroes except at the source index, where it is equal to 1. Finally $D(\epsilon)$ is a diagonal matrix and that term is part of the absorbing boundary conditions. Thus, the main pre-processing task is to calculate $U^T AU$.

The reduction procedure can also be attained without orthogonalization, as we explain in the section on Slanted Projection. We also describe an old algorithm [14, 20] for finding a well conditioned basis of columns of a general matrix and calculating recursively the matrix of normal equations. This is then used to implement a version of the slanted projection algorithm and is demonstrated in some of our example problems. An intriguing possibility is that of doing MOR using only a limited number of rows, which we explore in Section 6. Additional savings can be obtained by using only partial integration when generating the snapshots.

We also consider the use of randomized algorithms [2, 8, 7] for the linear algebra steps that are required to project the high fidelity equations into the reduced ones. Because of the availability of fast least squares solvers for large matrices we can also consider applying this to the slanted projection algorithm.

In order to reduce the computing cost further we also investigate in Section 9 the use of incoherent encoded sources or super-shots. At this point, the winning combination seems to be MOR with slanted projection (i.e., no orthogonalization) and the use of the compacted snapshots with the receiver/source rows or Monte Carlo abbreviated multiplication, to generate the coefficients and reduce the size of the least squares problems that arise. That is what we will investigate in more detail.

Numerical comparisons and performance are shown for 2 and 3D cases. The conclusion is that this approach can save considerable computer time if implemented properly. The codes we use are neither optimized nor paralellized.

2 Randomized algorithms for large scale linear algebra

In recent times there has been considerable activity in the area of probabilistic algorithms for constructing approximate decompositions for very large

matrices, as those that occur in big data mining, computer learning, neural networks, page ranking (Google search) and genome calculations, to name a few.

We are interested in obtaining U, an orthogonal basis for the range of the matrix S of snapshots, for the case in which neither S nor U fit the fast memory of our computer.

2.1 Randomly sub-sampling rows and columns

Let S be the $n \times k$ matrix of snapshots, so large that it does not fit in fast memory. However, n is such that we can read a few snapshots at a time and fit them in fast memory. The following algorithm will enable the projection of the high fidelity system of equations:

1. Sample $s = r + p \leq k$ columns of S at random, with probability of chosing column S_i, $p_i^{col} = \|S_i\|_2^2 / \|S\|_F^2$. Here r is the target rank and p is a small over-sampling number. Normalize the numbers p_i^{col} so that they add up to one and use them to define a probability distribution P.

2. Form $Y_{n\times s} = \frac{\|S\|_F}{\sqrt{s}} \left[\frac{S_{i_1}}{\|S_{i_1}\|_2}, ..., \frac{S_{i_s}}{\|S_{i_s}\|_2} \right]$.

3. Sample s rows of Y using the distribution P and form the matrix $W_{s\times s}$, similarly as we did with the columns.

4. Orthogonalize the columns of W to obtain $V_{s\times s}$. Calculate a good basis for the approximate range of S by multiplying by Y. Use YV to reduce the high fidelity equations by assuming that $u(t) = YVa(t)$:

$$YVa_{tt} = AYVa + \cdots ,$$

$$a_{tt} = V^T(Y^TY)^{-1}Y^TAYVa + \cdots .$$

The first observation is that $YV_{n\times s}$ is not orthogonal and usually s will be larger than the number of columns that can fit in memory, so in principle it seems that we have not made much progress. However, the critical and potentially expensive solution of the normal equations $(Y^TY)^{-1}$ corresponds to a small $s \times s$ system. Also, for the large matrix-matrix multiplications exist fast approximate randomized algorithms that are trivially parallelizable [1]. So, in principle this would solve our problem. The question is to asses its cost, test it and validate it on realistic problems. There may be other, better approaches and we are exploring that with Michael Mahoney an expert in the subject from the Statistics Department at University of California, Berkeley.

A second observation is that it seems that we could have applied the same idea directly to the original matrix S; however there is no guarantee that the selected set of snapshots has full rank or is well conditioned, so the normal

equations are potentially dangerous to use, under those circumstances. Since the resulting problem now is manageable we can use truncated SVD's instead.

With regards to the random sampling of rows or columns we need first to create the cumulative density function C:

$$C_j = \sum_{i=1}^{j} p_i, \quad j = 1, \ldots, n \text{ (or } m),$$

where $C_0 = 0$ and C_n or $C_m = 1$ (columns or rows). Then we get a random number $0 \leq r \leq 1$ and choose j such that

$$C_{j-1} \leq r < C_j.$$

3 Low-rank approximation via interpolative decompositions

A similar approach is advocated by [2], with the additional advantage that they provide a public implementation in [9]. We are interested here in the fixed-precision problem, where the rank of the approximation is determined by the software, given a computational tolerance. Once a basis has been selected, then we can use this as the S in Section 5.

4 Randomized matrix products

As indicated above, there exist fast Monte Carlo algorithms for various matrix operations, which may help in handling very large matrices, both in terms of efficiency and memory requirements [1, 8].

Given a $m \times n$ matrix A and a $n \times p$ matrix B, the key to the method of abbreviated multiplication is the identity:

$$AB = \sum_{i=1}^{n} A^{(i)} B_{(i)},$$

where $A^{(i)}$ is the ith column of A and $B_{(i)}$ is the ith row of B. The algorithm now corresponds to selecting at random c terms in this sum ($c << n$), according to a probability distribution $\{p_i\}_{i=1}^{n}$, and scaling each term in an appropriate manner. In other words, the product AB is approximated by an abbreviated product

$$CR = \sum_{t=1}^{c} C^{(t)} R_{(t)} = \sum_{t=1}^{c} \frac{1}{cp_{i_t}} A^{(i_t)} B_{(i_t)}.$$

The probability distribution could be uniform (that does not require a priori access to the matrices) or the one indicated in the previous section. Drineas et al [1] show in Lemma 11 that

$$\|AB - CR\|_F = \mathcal{O}(\|A\|_F \|B\|_F / \sqrt{c}).$$

5 MOR without orthogonality

In the vein of the previous section, we consider the use of the snapshots directly, without obtaining an orthogonal basis. We assume then that $u(t) = Sa(t)$ and replace this expression in the original wave equation:

$$Sa_{tt} = ASa + \mathbf{b}u(t) - 2D(\epsilon)Sa_t. \tag{5.1}$$

Multiplying by S^T we get:

$$(S^T S)a_{tt} = S^T[ASa + \mathbf{b}u(t) - 2D(\epsilon)Sa_t].$$

We observe that $(S^T S)^{-1} S^T = S^+$ is the pseudoinverse of S and since $(S^T S)$ is not large we can use a truncated SVD procedure to solve the implied least squares problem. Alternatively, the matrix coefficient of the first term in the righ-hand-side can be obtained by solving the matrix least squares problem:

$$SX = AS,$$

where X is a $k \times k$ matrix. It turns out that in a yet unpublished paper, Xiangrui Meng, Michael A. Saunders and Michael W. Mahoney [10] address exactly this problem and have produced a publicly available code that works even in the case that S is rank deficient.

Unfortunately, in the case that S is ill-conditioned, this idea does not apply directly. One would need to have a procedure to whittle the columns of S into an $\tilde{S}$ that is well-conditioned and then apply the procedure mentioned above. A procedure to select an independent set of columns from an arbitrary matrix and compute its pseudoinverse or solve a matrix least square problem has been described in [14, 20]. Also, we can use the Monte Carlo procedures of Section 3.

Rewritting the above equation, we need to calculate:

$$a_{tt} = (S^+ AS)a + (S^+ \mathbf{b})u(t) - 2(S^+ D(\epsilon)S)a_t.$$

Below we discuss a modern implementation of the procedure in [14, 20]. As a bonus of this recursive procedure we end up with S^+. So, in general, we will have to inspect the snapshots first to produce a well-conditioned basis. This can be done by the following (expensive) procedure:

1. For the first snapshot we simply need to normalize it and incorporate it into the basis (if its norm is above a small threshold).

2. From there on we need to calculate the 2-norm of the component orthogonal to the existing basis. This can be done by solving the least squares problem:

$$B_q x = s,$$

where B_q contains the current basis, s is the snapshot under inspection and $x = B_q^+ s$. Then, we calculate the residual (actually, the least square solver returns r in the components of x after s):

$$r = (I - B_q B_q^+)s$$

and if $\|r\|_2^2 > thresh$, then we accept s into the basis. At the end of this process we should have a well-conditioned basis S and we can proceed as above. This is an expensive procedure that adds up to the overhead. Plausibly, we can use updating techniques to make the least square solves less expensive [12, 14, 20, 22].

6 Reduction of MOR to the receiver net

In certain applications, such as survey simulation, QC, etc. one only needs the time response at a set of receivers. This set is usually one dimension smaller than the whole spatial field, i.e., a line for a 2D problem or a 2D array for a 3D problem. This is indicated by adding the equation:

$$w = Cu,$$

where the matrix C, $nr \times n$, selects the elements of u that correspond to the nr selected spatial positions (we assume that the source positions are included in that set) and packs them on a vector w of size nr. Now, if we multiply equation 5.1 by C the reduction proceeds in the usual fashion if we assume $u = Sa$:

$$CSa_{tt} = CASa + C\mathbf{b}u(t) - 2CD(\epsilon)Sa_t.$$

This essentially picks the nr equations corresponding to the selected spatial positions. Observe that the product CAS with the full matrix S can be performed columnwise in the High Fidelity code, as the snapshots are generated and then only nr rows need to be saved to be read by the MOR code. The only proviso here is that the adaptive selection of a well-conditioned basis needs to be done before calculating this product and saving the snapshots. In fact, since even in the 3D case the sizes can be moderate, we can go back to using adaptive QR to obtain an orthogonal basis U in the reduced space. In other words, we can calculate $CS = UR$, where U is orthonormal and R

is upper triangular and well conditioned and S is a selected subset of all the snapshots inspected.

The other two terms in the right-hand-side only require the compacted nr rows and therefore they can be calculated in the MOR code. This eliminates totally the large dimension n and should make possible and very economical to run the MOR algorithm. Replacing we get:

$$URa_{tt} = CASa + C\mathbf{b}u(t) - 2CD(\epsilon)Sa_t.$$

Finally:

$$a_{tt} = R^{-1}U^TCASa + R^{-1}U^TC\mathbf{b}u(t) - 2R^{-1}U^TCD(\epsilon)Sa_t$$

and

$$w = CSa,$$

where R^{-1} stands for solving an upper triangular system of linear equations. Observe that no abbreviated matrix multiplications are necessary, so in principle, there are no additional approximations. An intriguing thought is: would it be possible to reconstruct the whole field if we save S instead of only CS? In any case it would be wise to compare the results of this MOR with simple trace interpolation.

7 Implementation and testing in 2D

We have implemented the algorithm described in the previous section, first in 2D and with $C = I$, to test the novel formulation of the slanted projection. In addition, a new idea came about. What about doing only partial integrations in the HF code? That is to say, get enough snapshots in the basis to represent well the manifold where the whole solution lies and do that for every shot, in such a way that the total cost for, say, five shots is the equivalent of integrating fully for two shots. It turns out that this works well and now all the shots are accurately calculated by MOR/PSD (Proper Slanted Decomposition). We have experimented with integrating only for a few steps after the first shot and that works remarkably well, showing some small errors at the final quarter of the integration interval.

A conservative approach is to do a full integration for the first and last shot and then shorten the integration interval for the center shots. That works well also. Obviously there are a number of options here and we are still exploring the optimal combination that would give the greatest performance boost with good accuracy for all shots.

We present now some numerical results. They correspond to model *vel3l*, the three layers constant velocity 2D model we have used before. We consider a five shots sequence, with shots separated by 10 m and a 1 Hz Ricker wavelet as the source. In Figure 7.1 we crossplot the results of the High Fidelity code

Code	Time	ns	err^1	err^2	err^3	$err^\$$	err^5	thresh
HF	275							
HFslant	160	10^5						0.01
MOR	85.8		0.00018	0.0039	0.0057	0.0024	0.0021	
Ratio	0.89							
HFslant	93	50000						0.05
MOR	116		0.00089	0.0065	0.012	0.017	0.025	
Ratio	1.32							

Table 1: Performance of MOR with slanted projection. 5 shots.

Code	Time	Rank	Worst error	ϵ	reduce	S1 snaps	S2-S10
HF	1030						
HFS	192			0.1	0.9	150	25
MOR	160	99	0.054				
Ratio	2.93						

Table 2: Model vel3l 2D. 10 shots.

with those of MOR at a location on the free surface. The snapshots were produced by running full integrations at the end shots and only twenty steps at the interior shots. We measured the error as:

$$error = \sqrt{\Sigma[(THF_i - TMOR_i)/max(1, |THF_i|)]^2}.$$

For $error < 0.005$ there is no visible difference in a cross-plot of the two traces. The total number of mesh points is $n = 123201$ and ns is the number of selected rows. For $ns = 25000$ there were noticeable errors for the last two shots.

We try now with 10 shots. The best results for a limited exploration of the tuning parameters space are shown in Table 2 and Figure 7.3.

8 LSRN and a variant

In [10] the authors consider algorithm LSRN for solving large least squares problems using a random projection to produce a pre-conditioner for an iterative method. They demonstrate theoretically the quality of this pre-conditioner, independently of the chosen random projection.

Let the $n \times k$ matrix A be the matrix of coefficients of the least squares problem and r its rank. In our case we are interested in problems for which $n >>> k \geqq r$, and therefore we will look at the first algorithm LSRN.

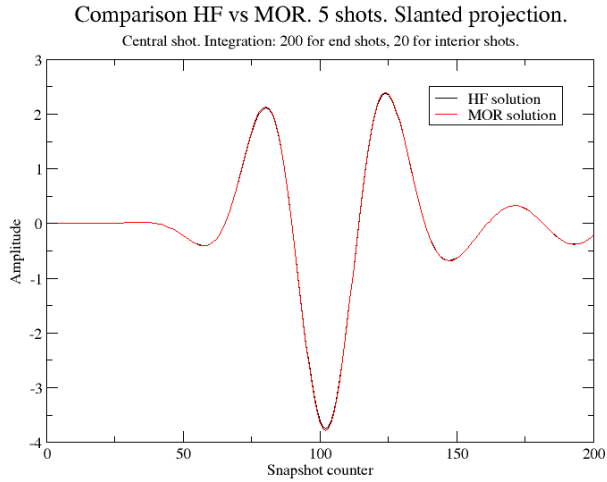

Figure 7.1: Crossplot of one trace for central shot (maximum error of 0.0057). $ns = 100000$.

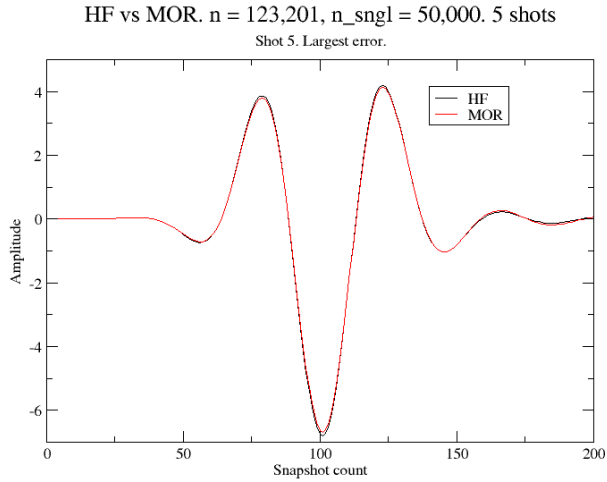

Figure 7.2: Crossplot of one trace for last shot (maximum error of 0.025). $ns = 50000$.

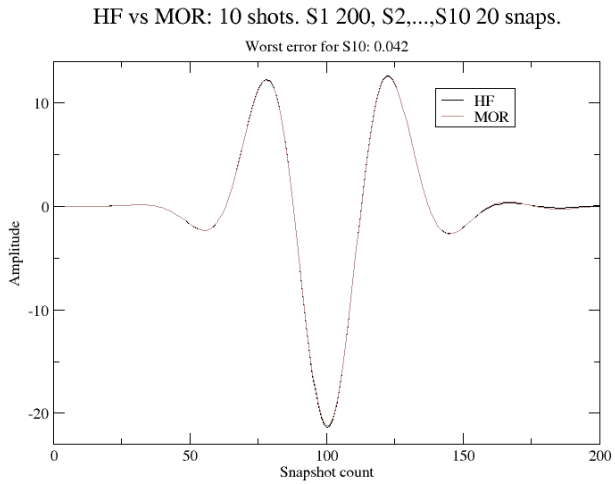

Figure 7.3: Model vel3l 2D. 10 shots.Worst error.

Algorithm LSRN

1. Choose an oversampling factor $\gamma > 1$ and set $s = \lceil \gamma nk \rceil$.
2. Generate $G = random(s, n)$, i.e., an $s \times n$ random matrix whose entries are independent random variables following the standard normal distribution.
3. Compute $\tilde{A} = GA$.
4. Compute $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $r = rank(\tilde{A})$, $\tilde{U}$ is a $s \times r$ orthogonal matrix (not needed), $\tilde{\Sigma}$ is a $r \times r$ diagonal matrix and $\tilde{V}$ is orthogonal $k \times r$.
5. Let $N = \tilde{V}\tilde{\Sigma}^{-1}$.
6. Compute the minimal length solution to $min_x \|ANy - b\|_2$ using an iterative method. Let $\hat{y}$ be the solution.
7. Calculate $\hat{x} = N\hat{y}$.

One additional difficulty in our problem is that n can be so large that A does not fit in fast memory. Worse still, G is even larger, so the product GA will probably require one of the techniques for approximate multiplication [8]. Also, distributed computing is only allowed at the level of a multi-core box, since the complete simulation or imaging problem requires thousands of

shots which are distributed to a network of multi-core machines. In essence, we are seeking to speed up the global calculation by making each simulation (or group of simulations) more efficient, without sacrificing accuracy.

If the generation of random numbers is sufficiently fast, an alternative to storing A and G is to process A by columns, re-generating the random matrix G always starting from the same seed.

9 The Rosen-Pereyra algorithm revisited

We discuss now in detail the algorithm of [14, 20]. This algorithm essentially executes the procedure described above in a more economical, recursive manner. The first step is as indicated above. Let us assume that we have constructed a well conditioned basis B_q with q column vectors and that we have calculated $(B_q^T B_q)^{-1}$. Let a_{q+1} be a candidate new vector. As indicated in step 2. we solve the least squares problem $B_q x = a_{q+1}$ as $x = (B_q^T B_q)^{-1} B_q^T a_{q+1}$ and calculate the residual $r = (a_{q+1} - B_q x)$. Departing slightly from the original algorithm, we check if

$$\|r\|/\|x\| > tanang,$$

in which case we accept the new vector temporarily in the basis and update the inverse $(B_{q+1} B_{q+1})^{-1}$. Observe that $\|r\|/\|x\|$ is the tangent of the angle that the new vector forms with the current basis, since x is its projection on it. Thus if we choose $tanang = tan(20^0)$, say, we will be accepting a vector that is far from dependent from the existing ones. To update the inverse we use the formula:

$$(B_{q+1}^T B_{q+1})^{-1} = \begin{pmatrix} (B_q^T B_q)^{-1} + xx^T/\kappa & -x/\kappa \\ -x^T/\kappa & 1/\kappa \end{pmatrix},$$

where $\kappa = a_{q+1}^T r$. Another safeguard recommended in the original paper is to keep the size of the elements of this matrix small. An easy way to do that at this point is to control the size of $1/\kappa$. If $1/\kappa > lev$, then we reject the column. So, given $(B_q^T B_q)^{-1}$ from a previous step, the algorithm progresses by calculating:

1. $x = (B_q^T B_q)^{-1} B_q^T a_{q+1},\ r = a_{q+1} - B_q x.$

2. If $\|r\|/\|x\| > tanang$ then a_{q+1} is accepted in the basis and the inverse is updated, provided the restriction on κ is satisfied.

3. Continue until all the vectors have been inspected. Observe that the objective here is to construct a well conditioned basis, not to obtain the best numerical rank, as it was proposed in [14, 20].

Code	One shot	Preprocess	Total
HFWave	98.24	-	687.65
MORslant	1	100.6	204.0
Ratio	98		3.37

Table 3: Comparisons for model vel3l. 7 shots.

10 Comparisons

Having implemented the Rosen-Pereyra algorithm of [14] (as MORslant) that allow us to choose a well conditioned deterministic basis, we go now to compare it with the full fidelity code HFWave. We use the three layer model vel3l. We run seven shots using snapshots from the first and last one. From these 800 snapshots 94 are chosen when we use thresholds of 0.01 for the $\tan(\phi)$ and a maximum of 200 for κ. The statistics are shown in Table 3. The total time for MORslant includes the HF shots required to generate the snapshots. We recall that for this problem, pre-processing time using the SVD to obtain an orthogonal basis was 296.62 (Report 1, 10/27/2011). With this basis the accuracy at the center shot is very good in phase as well as in amplitude.

We count one HF run as part of the MOR cost. Recall that a 2D survey consists of firing a line of shots. We divide the total number of shots by 7. Once the process has started each new interval only requires to calculate the end shot to complete the snapshots for the MOR basis, since the first shot is the last of the previous interval.

10.1 Derivation of updating formula in the Rosen-Pereyra algorithm

This is obtained simply by 2×2 Gaussian elimination. First we eliminate the subdiagonal term by multiplying the first equation by $a_{q+1}^T B_q (B_q^T B_q)^{-1}$ and replace the second equation by substracting it from the modified first equation, obtaining:

$$\begin{pmatrix} B_q^T B_q & B_q^T a_{q+1} \\ 0 & -\kappa \end{pmatrix} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} = \begin{pmatrix} I & 0 \\ x^T & -1 \end{pmatrix},$$

where $\kappa = a_{q+1}^T r$.

Now we perform the back substitution, that gives:

$$\gamma = -x^T/\kappa;\ \delta = 1/\kappa;\ \alpha = (B_q^T B_q)^{-1} + xx^T/\kappa;\ \beta = -x/\kappa.$$

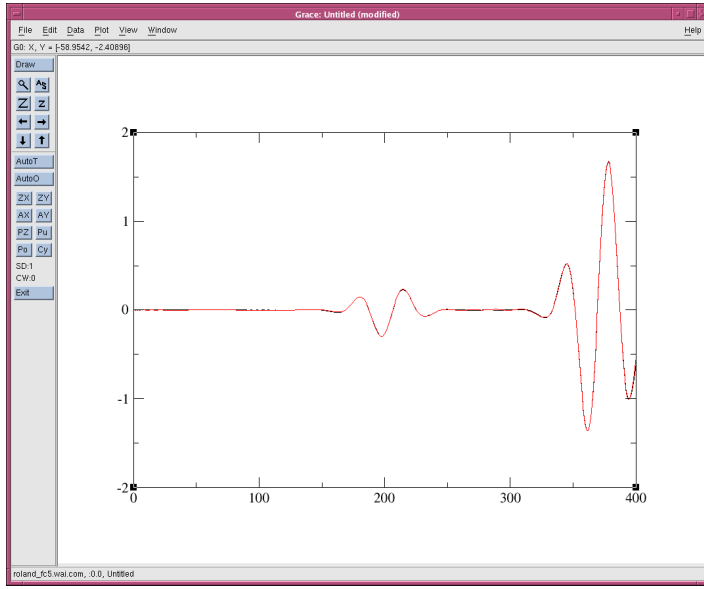

Figure 10.1: Crossplot of seismograms at first receiver for HF and MOR

11 3D implementation

The extension of HF and MORslant to 3D is in principle straightforward. In the case of HF we just add an eight order formula for the y direction. This requires extensive modifications to the routine GENMAT that generates the sparse matrix representation of the spatial discretization. A bonus of our current organization is that that routine is common with MORslant and therefore, once debugged and validated advances us a long way into the implementation of MORslant3D. A number of small tweaks need to be made in order to accomodate the new coordinate direction, including some in the main input file.

Recalling that their share the same input file, similar changes need to be made to implement MORslant3D, although all the matrix manipulations in the straightforward extension of the 2D code remain the same. Only their sizes will grow considerably and therefore eventually a different implementation along the lines sketched before will be necessary.

Once these implementations are coded and debugged we need to validate them. For that we will consider first a problem with a known solution, involving spherical waves. Let $r = \sqrt{x^2 + y^2 + z^2}$, and consider the wave equation in the (t, r) space (for the half semi-space $r \geq 0$ with constant velocity v) and a point source function $s(t)$:

Method	Total time	Pre-process
HF	2506.43"	-
MOR	1311"	1302"
Ratio	1.91	-

Table 4: Spherical wave test

Method	Total time	Four shots
HF	7134.8	3171.0
MOR	1851	5022
Ratio	1.42	

Table 5: Comparison of HF3D and MORQR3D for model vel3l3D, 9 sources.

$$(ru)_{tt} = v^2(ru)_{rr} - v^2\delta(x)s(t).$$

The solution of this wave equation in one dimension is:

$$u(t,r) = -0.00003528 \times v^2/r \times s(t - r/v),$$

representing spherical traveling waves. For $t = 0$ we have the initial conditions $u(0,x,y,z) = 0,\ u_t(0,x,y,z) = 0$. We choose a Ricker wavelet as a source. Both HF3D and MORslant3D reproduce the exact solution, so they are validated. Performance is shown below. This corresponds to 200 snapshots. Rank for MORslant3D was $k = 33$.

Having verified the codes we now proceed to do some comparisons, on the style of what we did for 2D. We consider problem vel3l3D, a three layers, constant velocity model with the following specs:

$n_x = n_y = n_z = 176,\ n_t = 200.\ dx = dy = dz = 10,\ dt = 0.005.$

We consider 9 sources in a 3×3 uniform array and extract 800 snapshots from the corners of the array. The results are mediocre and they can probably be attributed to extensive paging due to lack of main memory. The accuracy is also not too good. So, we need to improve the coding, trying to save memory and introduce Open MP constructs to take advantage of multiple cores. In the meanwhile we go back to the best algorithm that we have: adaptive QR, and extend it to 3D.

12 MORQR3D

We extend MOR with adaptive QR to 3D. We test it with the 3D version of the three constant layers model vel3l3D (see previous section for the specs). First we try a 3×3 array of sources selecting 110 snapshots from the corners.

These results are considerably better than those using the slanted projection of the previous section, but the speedup is not very impressive. We try

Method	Total time	Five shots
HF	19948.55	3941.8
MOR	4045.96	7987.76
Ratio	2.5	

Table 6: Comparison of HF3D and MORQR3D for model vel3l3D, 25 sources.

a larger problem with a 5×5 array of sources, selecting 104 snapshots form the corners and the central shot. The speedup ratio is considerably better, but the accuracy is poor and deteriorates as the shot moves, which indicates that we are not selecting enough snapshots from the later shots. Thus we introduce a varying threshold that starts pretty strict and gets relaxed as we move along (we used a factor of 0.5 to multiply the tangent of the angle control, from one shot to the next, starting with 0.2). This strategy selects 146 snapshots and produces very accurate results over the whole set of shots. The worst RMS is 3.28×10^{-3} and it is located at shot 19. This gives a very good match at ploting accuracy. Observe that the maximum possible speedup factor is 5, so there is room for improvement.

13 Using encoded sources[1]

We have seen both in 2D and 3D that an important part of the cost of MOR comes from the need of obtaining snapshots from a number of HF calculations. An interesting idea used in 3D full waveform inversion may help in reducing this cost [6]. The idea is to activate multiple sources at once, thus requiring fewer simulations. This requires encoding and summing the data to produce equivalent single gathers, but that does not concern us here.

The question to be answered is if from these snapshots, generated by combining multiple sources in one simulation, one can obtain an appropriate basis for Model Order Reduction.

Encoded simultaneous-source methods have been proposed to improve the efficiency of seismic acquisition and processing [4, 6, 19]. It is argued that incoherently encoded gathers contain much more information than do a single point-source gather or a coherently encoded gather (i.e., a plane-wave gather). In particular, an incoherently encoded gather illuminates more of the model than a point-source one. It is likely that we may need to use multiple encodings to better represent the relevant subspace of snapshots.

Thus, the balance of all these factors need to be explored, in order to see if this idea provides a net gain from our previous implementation. We do

[1]We thank Josep de la Puente, Barcelona Supercomputer Center, for suggesting this approach and pointing us to Krebs et al paper.

Source	Error SS2	Error SS3	Error (succ. 1,9)	Error (succ. 3,7)
1	0.011	0.029	0.004	-
2	0.021	0.018	0.13*	-
3	0.036*	0.0098	0.069*	0.00068
4	0.033*	0.0023	0.057*	0.036*
5	0.028	0.0043	0.051*	0.051*
6	0.029	0.0042	0.046*	0.042*
7	0.015	0.023	0.049*	0.1*
8	0.027*	0.026	0.14*	-
9	0.028	0.022	0.013	-
10	0.038*	0.026	-	
11	0.051*	0.034*	-	
Time HF	1040.64	1040.64	851.43	478.95
Time MORQR	242	364.11	205.9	204
Ratio	4.3	2.86	4.1	2.34
Rank	94	102	93	83

Table 7: Error and time comparisons for several strategies. SS2 (two repetitions) and SS3 (three repetitions) activates sources 1,3,5,7,9,11 at once.

that first in 2D using model vel3l in a 2" integration with a $5Hz$ source.

After an implementation of a solver where several sources are activated at the same time with randomized signs, we summarize below the best results. The important main result is that with three HF simulations, we can have more sources calculated with MOR via this approach, with better accuracy than our previous one source at a time generation of snapshots. We see by comparing the different columns that one can obtain a speedup of 2.86 by this simple device, plus better accuracy. The error headings indicate which shots are included in a super-shot (ss) and similarly for the successive single-shot simulation (succ.).

The measure of the error we use is an average, relative RMS between the High Fidelity and the corresponding MOR results for one trace:

$$\sqrt{\Sigma_{i=1}^{N}(u_i^{HF} - u_i^{MOR})^2/(N \times max_i|u_i^{HF}|)}. \tag{13.1}$$

A level of error less than 0.03 corresponds roughly to acceptable small deviations in amplitude in cross-plots. There are three possible sources of error: amplitude, phase and unwanted oscillations. It would be good to be able to discriminate between them, but in the meanwhile it is prudent to check the cross-plots. We have marked with * those shots that may not be accurate enough. The best results for an 11 shot span correspond to three randomized repetitions of a supershot that includes every other shot.

Other strategies, such as activating more adjacent sources at the same

Init. Thresh.	0.05	0.1
Time HF	24457.5	24457.5
Time HFss	4281.13	4215.96
Time MORQR	2786.6	1274.92
Total MORQR	7067.73	5490.88
Ratio	3.46	4.45
Rank	150	97
Max. RMS	0.011	0.025

Table 8: Supershots in 3D

time, did not give good results, probably because of excesive cross-talk. Using one non-random super-shot (i.e., with all positive sources) gave reasonably good results, but it seems that the repetitions, which require randomization, are important to achieve the necessary variety of snapshots and correspondingly better accuracy. These results were obtained using the QR based algorithm for orthogonalizing the snapshots.

Observe that the supershot approach requires that MOR calculates all the individual shots, including those active in the supershot, since they are all mixed up and cannot be separated. However, since the MOR integration is so fast, the additional cost is negligible.

With supershots is not clear at this time how you simulate a whole survey economically. Maybe keep the last half of the basis from one interval to the next and reduce the number of repetitions after the first interval?

13.1 Supershots in 3D

We extend the implementation of supershots to the 3D solver HF3Dss. We still use HF3D to generate the baseline traces and timings for comparison. We use the 3D version of the 3 layers model, vel3l3D. We present the results for two good runs for a 5×5 molecule of shots. The supershots activate the same sources that we used for separate shots: i.e., $[1, 5, 13, 21, 25]$ and in both cases we use four repetitions, since with three not enough accuracy is achieved. The difference between the two runs is in the initial threshold to choose snapshots, $thresh = 0.05,\ 0.1$ that we multiply by 0.5 for each successive shot.

Observe that either result is an improvement over the separate shot approach and this improvement comes from the fact that we are using one less HF solve to generate the snapshot basis. In Figure 13.1 we show the less accurate result for $thresh = 0.1$.

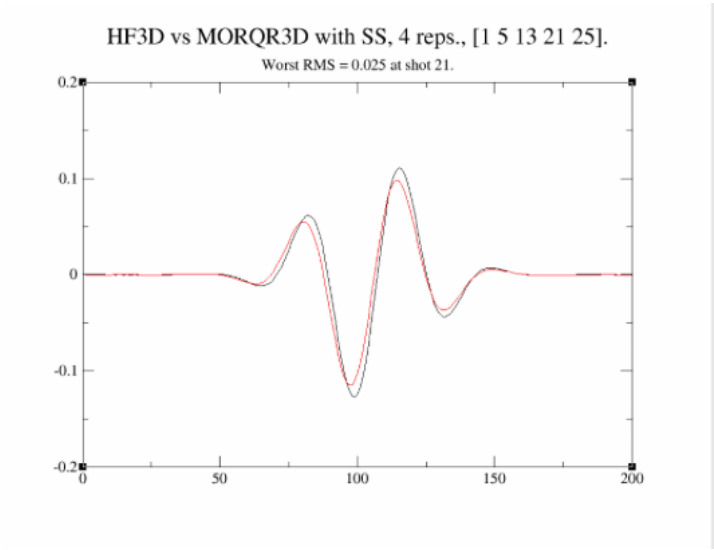

Figure 13.1: Crossplot for worst RMS in vel3l3D supershot run with thresh.=0.1

13.2 Calculating the matrix coefficients for large problems

As we observed in Section 6, there are problems in which a substantial reduction in dimensionality can be attained by working only on the receiver array. However, there are other applications, such as Reverse Time Migration, where the response in the whole spatial mesh is required. In this case and when all the snapshots do not fit in the memory available we need to choose carefully the algorithm and see how we can operate with one single snapshot at a time. At this point we can only say that SVD, the Rosen-Pereyra algorithm and progressive QR will not qualify since they require the presence of all the snapshots or their orthogonal proxies.

Therefore we take a fresh look into the slanted projection algorithm of section 5. Since the snapshots are not to be orthogonalized, the HF code only needs to produce and output them one at a time. As we will see below, if we use randomized algorithms, it turns out that we will only need to save and process a limited (and small) number of rows of the matrix of snapshots. The MOR code then will read this information and calculate the necessary matrix coefficients approximately, by using the algorithm of section 4. The two more complex coefficients to be calculated are: $S^T S$ and $S^T AS$.

Observe that the dimensions are apropriate for the algorithm of section 4: S^T is $k \times n$, with $k <<< n$. So, we proceed to select randomly a priori in

the HF code the indices of $c = \gamma k$ columns of S^T (i.e., rows of S) using an uniform distribution. The oversampling factor $\gamma > 1$ will have to be chosen experimentally, once the accuracy impact of this approximate computation is gauged. In any case, we expect $c << n$. From now on we add an ^{c} to a matrix name to indicate that the $ind(i),\ i = 1, ..., c$ rows are stored compacted in positions $i = 1, ..., c$.

Therefore, we need only to write and read a matrix S^c of size $c \times k$, which will not tax the memory. Likewise, only the appropriate c rows of A need to be generated and saved. Since A's rows only have at most 25 nonzero elements, this provides great savings, both in storage and in operations.

Thus, we first perform the random multiplication $A^c S$ and then random multiply the result with S^{Tc}. The least squares part requires the solution of a square system with the approximate normal equations $S^{Tc}S^c$ $(k \times k)$. This can be done with a truncated SVD algorithm, in case the basis is ill conditioned.

Even for $k = 1000$ and $c = 2000$ this is quite a feasible program, within the memory constraints we have. The remaining question is related to the accuracy of the final calculation. The interesting part is that the first product $A^c S$ can be performed exactly by generating only $(c \times 25)$ numbers, a very modest proposition. This needs to be done in the HF code, one column at a time, for each snapshot. By saving this product there is no need to generate the matrix A in the MOR code.

13.3 Accuracy of MC product

We offer now some insight on the accuracy that can be achieved with the algorithm of section 4. We consider two random matrices A (100×10000) and $B = 10000 \times 100$ and apply the algorithm for all values of c. The Frobenius norm that is plotted is: $\|AB - CR\|_F / \|A\|_F \|B\|_F$. According to the theory, we expect this to be of the order of $1/\sqrt{c}$. We observe in the results that either measure of the error decreases rapidly and after a few hundreds tapers down and then decreases more slowly.

In case there is any doubt about the necessity of all the details of the carefully crafted algorithm of [1, 8], we try some reasonable variants. For instance, we see that not allowing repetitions is OK at the begining, but eventually, not only this is worse than Drineas et al, but at the end does not even go to full roundoff accuracy. The sequential, non random algorithm, is not even worth discussing.

For our case, where $n \sim 10^{10}$, we can expect better accuracy relatively earlier. It remains to be proven how far we need to go in order to achieve the ultimate accuracy required by the overall problem. Observe that for the given size of current top box memory of 32×10^9 single precision numbers, we can stretch $\gamma k \approx 10^5$ in the approach above, without running out of memory.

This algorithm rational is based on the fact that these matrices have low

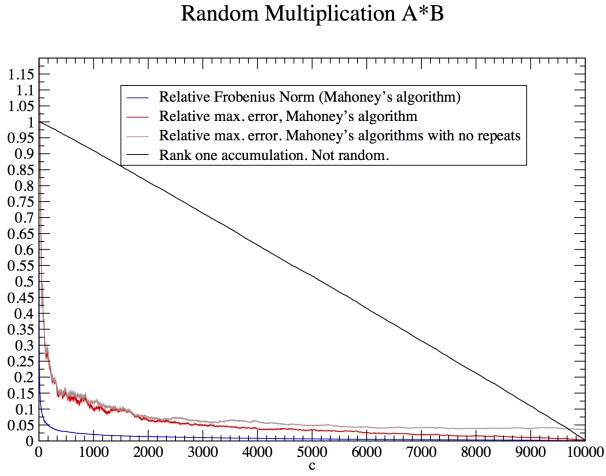

Figure 13.2: A (100×10000), B (10000×100)

rank ($r \leq k$) and therefore they contain much redundant information with regards to the subspaces that they span. That is why it is plausible that we can obtain a good approximation of the product if we sample a small number c of the large dimension lines and scale the sum of their rank one products appropriately.

In terms of selecting dynamically a well-conditioned subset of snapshots it is worth considering the results and algorithm of [23].

13.4 Random vs deterministic and working on the receiver net

One impossible dream, for applications in which we only need the response at the receivers (usually a net with dimension one less than the one for the whole model), is to work only in this sub-net. For wave propagation this is impossible, since we need to develop the whole space field in time given that everything is coupled and we need to account for the scattering in depth.

However, the above ideas lead to an interesting alternative that would amount to working only with the mesh of receivers in the MOR simulation. This is plausible since the snapshots would contain the full spatial information from the response in the model.

Unfortunately, this idea would require to select the rows of the matrix of snapshots that correspond to the receiver positions in the abbreviated multiplication process. That is, of course, not random, so the question is

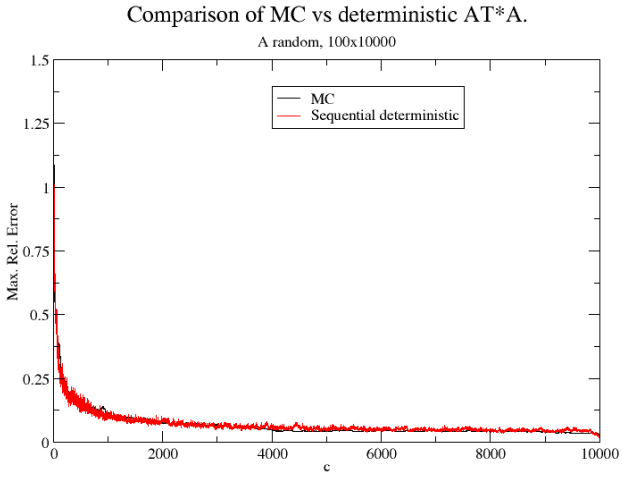

Figure 13.3: Crossplot of maximum relative error for MC and deterministic approximate matrix product

how the accuracy of the reduced products would be affected. In order to get an idea about this we have performed an experiment similar to the one above, in which we compare the MC product with one in which we use a deterministic (sequential) selection of columns/rows in the product $A^T A$, for a 100×10000 random matrix, for all values of c. The results can be viewed in Figure 13.3, where we crossplot the maximum relative error of the difference between the exact and approximate products. The two curves superpose, while the error for the deterministic approach seems to be more oscillatory as a function of c.

In reality, as we showed in Section 6, one could work completely in the receiver net, without resorting or thinking of the whole mesh at all. That would not require abbreviated products and would virtually reduce the dimensionality by one, making a 3D problem of 2D size. This is quite a novel idea so it needs to be tested.

13.5 Implementation details

The product $S^T S$ fits exactly the case described in detail in section 4, with $A = B^T$, so it requires no further explanation.
The product $S^T(AS)$ is somewhat more complicated, so we explain in detail how to perform it without ever having to handle matrices with the large dimension n. We recall that an ^{c} indicates that the matrix is compressed,

with the rows $ind(i)$ in the first i positions.

- In the HF code we generate the uniformly random distributed indices $ind(i)\epsilon N,\ i = 1, ..., c$. Here N is the set containing the integers from 1 to n. Save and output only these c rows as S^c (of dimension k). Calculate the first product A^cS exactly, saving the result in D^c $(c \times k)$, one column at a time, as the snapshots are generated:

$$D^c = A^cS;\ D^c_{jl} = \Sigma_{i=1}^{RowLength(j)} A^c_{j,colind(i,j)} S_{i,l},\quad j = 1, ..., c;\quad l = 1, ..., k.$$

- In MORslantMC (Model Order Reduction with slanted projection and Monte Carlo matrix multiplication), read in the matrix S^c $(c \times k)$. Read also D^c. Finally, perform the second product in the standard MC fashion: $S^T(AS) \sim S^{Tc}D^c = \Sigma_{i=1}^{c} S^{c(i)} D^c_{(i)}$. Observe that at no point we have needed to save a matrix with dimensions including n.

- Once the integration in time is completed we need to reconstruct the time response at the receivers. Here is where the observation of 13.4 comes handy. In fact, this observation needs to be applied from the begining, instead of the random choice, so we would need to evaluate its effect.

14 A large scale 2D numerical example

We consider now *vel4pc*, a 2001×2001 $2D$ model of complex geology. See Figure 14.1 for a depiction of its velocity mesh.

The number of time step stations (where we potentially inspect snapshots) is 400, with $dt = 0.005$", for a total integration interval of 2" and a 5 Hz Ricker wavelet as the source. We consider 10 shots spaced by 6.25 units and extract a trace at $[900, 5]$ for comparison of the HF and MOR codes. We exercise the options of using less than the total integration interval to select snapshots in code HFS, namely: for the end shots, S_1 and S_{10} we use the first 300 stations, while for the inner shots we only inspect the first 30 ones, for a total cost of about 2.1 full integrations. With a threshold at the end points of $\epsilon = 0.075$ and $\epsilon_i = 0.9^{i-1} * \epsilon,\ i = 2, ..., 9$ for the interior ones, a basis of 164 snapshots is selected. We also consider the matrix C so that only 200100 consecutive points are used in HFS and MOR, starting at 1000500. In Table 9 we list the maximum absolute errors for the difference between the HF and MOR traces and in Figure 14.2 we show crossplots for the best and worst cases (S_1,S_5).

Observe that for this example there is no action after station 300. If there were significant signal then we could not ignore this final interval when seeking useful snapshots from the end sources, as we have seen in other examples. Again we see that phases are very good, while there is a miss in

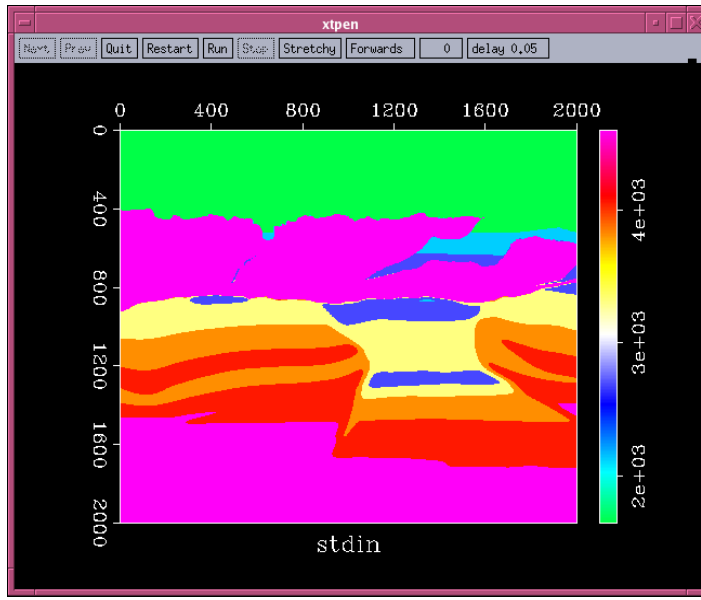

Figure 14.1: Model vel4pc velocity

1	2	3	4	5	6	7	8	9	10
0.00017	0.00053	0.00091	0.0012	0.0013	0.0013	0.0012	0.00095	0.00057	0.00025

Table 9: Maximum absolute errors for each shot at the control trace for 2D model vel4pc

amplitude matching, which is visible at this scale. We observe in Table 10 that a speed up by a factor of 5 has been achieved by been able to use a significantly larger number of shots to share the overhead. This should be even more pronounced in 3D. We have used no parallelization (one thread only for the three codes).

14.1 3D layered media example

We consider now a 3D version of model *vel3l, vel3l3D.* The specs are:

$nx = ny = nz = 176$, $nt = 200$, $n_{sngl} = 500000$,
$dx = dy = dz = 10m$, $dt = 0.005s$.
$threshold = 0.01$, $\delta threshold = 0.9$. 9 shots in a 3×3 square.
Frequency of Ricker source wavelet is $5Hz$.

The matrix C selects 500000 consecutive rows starting at 1982465 and the trace is collected at point 2182405. The sources are spaced by $10m$ in each

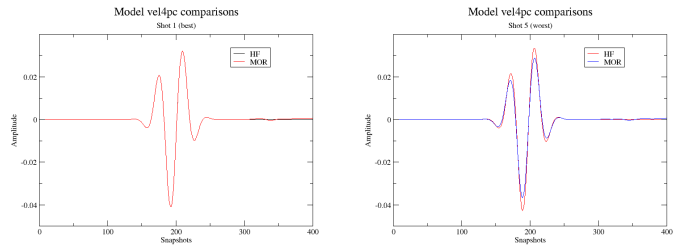

Figure 14.2: Crossplot of best and worst shot for 2D model vel4pc

Method	Time (sec.)
HF	249579
HFS	48510
MOR	1168
Ratio	5.0

Table 10: Performance for 2D model vel4pc

direction, with the first one at coordinates $(71, 71, 5)$, i.e., at point 2180645. We recall that the order of the mesh in vector form proceeds, from slow to fast indeces (x, y, z).

We collect snapshots from the corner shots $(\#1, 3, 7, 9)$ for all timesteps and also from the interior shots, but only for the first 10 timesteps, for a total of $4, 25$ HF integration intervals. Of, course, as we see in Table 11, only a subset of snapshots are selected. In conclusion, the limiting speedup for this example is $9/4.25 = 2.12$. The threshold is multiplied by $\delta threshold$ for each successive shot, in order to diminish the bias of the basis been more heavily populated with snapshots from earlier shots.

We are interested at this point in assesing how much we can reduce the number of rows (size of matrix C) and the effect of the adaptive snapshot selection strategy, including the short integration for the interior shots, all part of our novel algorithm. For this example we are reducing the number of rows by a factor of $176^3/500000 = 5451776/500000 = 10.9$.

In Table 11 we describe the results and in Table 12 we post the computing times for the different codes:

HF3D, the High Fidelity code that runs all the shots and produces and output trace for comparison only.

HFS3D, the High Fidelity code that runs the limited integrations, extracts adaptively the snapshots S, produces the QR decomposition of the abbreviated version $CS = QR$, calculates CAS and outputs these matrices for the MOR code $MORS3D$.

Shot	1	2	3	4	5	6	7	8	9
Rank	41	51	89	99	109	119	150	160	185
RMS	1.3(-5)	4.0(-3)	1.7(-5)	1.1(-3)	2.8(-3)	2.1(-3)	1.1(-4)	3.8(-3)	2.1(-3)

Table 11: Rank and accuracy for each shot in 3D model vel3l3D. $a.b(-c) = a.b \times 10^{-c}$.

Code	Time (sec)
HF3D	147096
HFS3D	52536
MORS3D	736
Ratio	2.76

Table 12: Comparative time performance for 3D model v3l33D. 9 shots.

Finally, $MORS3D$ reads all the necessary information, calculates the remaining matrix coefficients and performs the time integration in the projected spaces. The RMS error was defined in 13.1 and the rank corresponds to the number of snapshots selected at the end of the integration for each shot. In Figure we crossplot the corresponding time traces for Shot 2, the one with the worst accuracy in the whole calculation. We see that eyeball accuracy is quite good, and of course it is better for the more accurate shots. All the calculations were performed on an $8CPU's,\ 2300MHz$ machine with 3.3 GB of fast memory, with a single thread especified.

Next we consider a 5×5 square array of sources. The specs for this larger experiment are:

$\varepsilon = 0.025,\ \delta\varepsilon = 0.8,\ ncut = 10,\ n_{sngl} = 500000,\ nsteps = 200.$

We perform full integrations at the corners but use only 20 time steps for every other of the interior ones. In Table 13 we show the pattern:

We run this experiment in a 24 cores machine with 100 GB of memory. We use the parallelized versions in OpenMP of the integration loops of HF3D and HFS3D, activating 12 cores. Although MORS3D is not paralellized, the relative cost is essentially negligible, so this does not distort the comparisons

$$\begin{bmatrix} X & 0 & \triangle & 0 & X \\ 0 & 0 & 0 & 0 & 0 \\ \triangle & 0 & \triangle & 0 & \triangle \\ 0 & 0 & 0 & 0 & 0 \\ X & 0 & \triangle & 0 & X \end{bmatrix}$$

Table 13: Shot integration pattern: X full integration; 0 no integration; Δ $nsteps/ncut = 20$ steps.

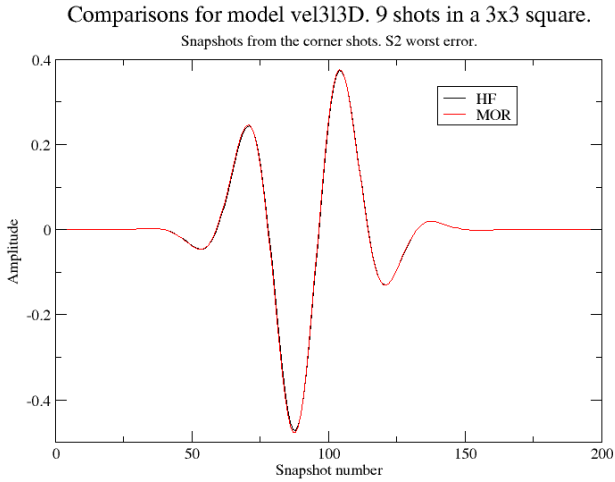

Figure 14.3: Crossplot of trace for worst error in model vel3l3D. Shot 2.

Code	Time (sec)
HF3D	26617.46
HFS3D	7338
MORS3D	661
Ratio	3.33

Table 14: Time comparisons for problem vel3l3D using 25 shots.

in any significant way. The rank is 220 and the maximum error of 0.015 occurs at Shot 10. There are small visible missmatches, but overal the accuracy is acceptable. The time performance is shown in Table 14.

15 A large 2.5 D complex model

We take a subset of model *vel4pc* of Section 14 and extend it laterally and constantly, generating a 3D version (actually, what is called a 2.5D model) of it that we call *vel4pc*25. The specs for this model are:

$nx = ny = 301,\ nz = 201.\ n = nx \times ny \times nz = 18,210,801.\ n_{sngl} = 1,800,000,\ tsteps = 200.$

$\epsilon = 0.01,\ \delta\epsilon = 0.8$, first source at $[15, 151, 5]$, trace at $[14, 154, 1]$. Resulting rank of snapshot basis is 63.

We run three shots to asses the scale of computing time. The results are

Method	1 Shot	3 Shots	Rank	25 Shots
HF3D	51747	176844		1,482,100
HFS3D	50113	55124	63	220,497
MORS3D	179	538		4,475
Ratio	1.02	3.17		6.6

Table 15: Results for three shots, model vel4pc25. Times in sec.

shown in Table 15.

The times for HFS3D correspond to the strategy for full survey simulation described in the next section and the 25 shots times are extrapolated from the smaller example. The maximum RMS error of 1.7×10^{-4} occurs at the end shots, and therefore the accuracy is excellent throughout.

16 Full survey simulation

A full survey simulation involving many thousands of shots will be partitioned in small subsets of the sort that we have tested above in order to use MOR. The easiest implementation is one in which the subsets are adjacent but do not overlap. In this case they can be calculated in parallel and no comunication will be necessary.

If we allow overlaps, considerable savings in computing time can be effected at the cost of programming complexity, increased storage and read/writes between fast memory and auxiliary storage. As usual these costs need to be balanced in order to attain a positive gain. We explain next a possible strategy that extends to a 2D array of sources what we did for a line of sources [25]. In the 2D modeling case we needed only to save the snapshots corresponding to the far end of the segment in order to reuse them for the next segment. Now we have two directions to attend to and things are more complex.

Algorithm

Collect the snapshots associated with each source in a single binary file with a header indicating the model name, shot number, length and number of snapshots. Assuming that we have a rectangle of subsets that form the whole shot array, the suggested algorithm proceeds as follows:

1. Start with the first row. After the first block is computed save the South (S) and East (E) snapshots.

2. For the next blocks in row one, re-use the E side of the previous block as the West (W) side of the new block, replace the old E side by the new one and save the new S side.

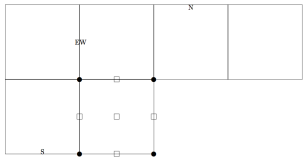

Figure 16.1: Schematic diagram of a full survey simulation

3. From block row 2 on, re-use S side of the block above and replace it by the new S side. Similarly with the E side (after the first block).

4. Once a row is finished go to 3 until the whole survey has been simulated.

For an interior 5×5 block as exemplified in Section 14, instead of 5 full integrations (assuming 1/10 of time steps for the every other shot in the interior) we would have only 1.333 integrations for a 3.75× save. **For that example that would imply a total speedup factor of approximately 12.5!** Of course, for an especific problem one would also have to consider the possible disk access penalties. In the diagram below we give an schematic of the geometry used.

17 Possible future improvements

The implementation we have of these three codes is pretty rough and could be considerably improved. It is remarkable that even with that handicap the results are fairly interesting and promising. We list now some thoughts on possible avenues of improvement in moving towards a production type code.

A good part of the slow performance of the HF codes for the larger examples comes from lack of fast memory and trashing between fast and disk memory, as evidenced . Although we were running on top of the line, large memory boxes, we have not been too carefull in the memory management. A blatant example is the storage of the sparse matrix that contains the discrete Laplacian. That matrix has $n \times 25$ non-zero elements and we use two copies to store the information on the coefficients and their locations. That is the same as the storage necessary to save 50 snapshots! With (quite) a bit of

effort this information could be generated on the fly and no storage would be necessary.

Another area that we had not had time to explore in any depth is the application of the randomized linear algebra techniques. Those again are potential space and computation savers, but we would have to assess their impact in accuracy and of course select the most appropriate implementations.

Finally, parallelization to take advantage of the multiple cores available has only been implemented for the integration loops in the HF codes. No use of the BLAS and most especially, BLAS3 have been effected. Thus the linear algebra implementation is probably inefficient in both memory use and parallelization.

References

[1] P. Drineas, R. Kannan and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication.* SIAM Journal on Computing **36**:132-157 (2006).

[2] N. Halko, P. G. Martinsson and J. A. Tropp, *Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review* **53**:217–288 (2011).

[3] P. C. Hansen, V. Pereyra and G. Scherer, Least Squares Data Fitting with Applications. Johns Hopkins University Press, Baltimore (2013).

[4] L. Ikelle, *Coding and decoding: Seismic data modeling, acquisition and processing.* SEG Extended Abstracts **77**:66-70 (2007).

[5] B. Kaelin and V. Pereyra, *Fast wave propagation by model order reduction.* ETNA **30**:406-419 (2008).

[6] J. R. Krebs, J. E. Anderson, D. Hinkley, R. Neelamani, S. Lee, A. Baumstein and M-D. Lacasse, *Fast full-wavefield inversion using encoded sources.* Geophysics **74**:WCC177-WCC188 (2009).

[7] M. W. Mahoney, *Approximate computation and implicit regularization for very large-scale data analysis.* Proc. of the 2012 ACM Symposium on Principles of Database Systems, 143-154 (2012).

[8] M. W. Mahoney, *Randomized algorithms for matrices and data, foundations and trends in machine learning*, NOW Publishers **3** (2011).

[9] P-G. Martinsson, V. Rokhlin, Y. Shkolnisky and M. Tygert, ID: A software package for low-rank approximation of matrices via interpolative decompositions. Version 0.4. Yale University, New Haven, Conn. (2014).

[10] Xiangrui Meng, Michael A. Saunders and Michael W. Mahoney, *LSRN: A parallel iterative solver for strongly over- or under-determined systems.* Submitted for publication (2013).

[11] R. Neelamani, C. E. Krhon, J. R. Krebs, M. Deffenbaugh, J. E. Anderson and J. K. Romberg, *Efficient seismic forward modeling using simultaneous random sources and sparsity.* SEG Expanded Abstracts **78**:2107-2111 (2008).

[12] Fabian Ojeda, Johan A.K. Suykens and Bart De Moor, *Low rank updated LS-SVM classifiers for fast variable selection.* Neural Networks **21**:437-449 (2008).

[13] V. Pereyra, *Wave equation simulation in two-dimensions using a compressed modeler.* American Journal of Computational Mathematics **3**:231-241 (2013).

[14] V. Pereyra and J. B. Rosen, *Computation of the pseudoinverse of a matrix of unknown rank.* Stanford University Computer Sciences Report SC18 (1964).

[15] V. Pereyra and G. Scherer, *Efficient computer manipulation of tensor products with applications in multidimensional approximation.* Math. Comp. **27**:595-605 (1973).

[16] V. Pereyra and G. Scherer, *Least Squares Scattered Data Fitting by Truncated SVD's.* Applied Numerical Mathematics, **40**:73-86 (2002).

[17] V. Pereyra and G. Scherer, *Large scale least squares data fitting.* Applied Numerical Math. **44**:225-239 (2003).

[18] V. Pereyra and G. Scherer, *Least Squares Collocation Solution of Elliptic Problems in General Regions.* Mathematics and Computers in Simulation **73**:226-230 (2006).

[19] L. A. Romero, D. C. Ghiglia, C. C. Ober and S. A. Morton, *Phase encoding of shot records in prestack migration.* Geophysics **65**:426-436 (2000).

[20] J. B. Rosen, *Minimum and basic solutions to singular linear systems.* SIAM J. **12**:152-162 (1964).

[21] K. Schwarz, *Darts, dice, and coins: sampling from a discrete distribution.* Manuscript (2011).

[22] M. Seeger, *Low rank updates for the Cholesky decomposition.* University of California at Berkeley, Tech. Rep, (2007).

[23] Joel A. Tropp, Column subset selection, matrix factorization, and eigenvalue optimization. SODA '09 Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, Philadelphia, PA pp. 978-986 (2009).

[24] M. D. Vose, *A linear algorithm for generating random numbers with a given distribution.* IEEE Trans. Soft. Eng. **17**:972-974 (1991).

[25] Chunling Wu, Dimitri Bevc and V. Pereyra, *Model order reduction for efficient seismic modeling.* SEG Annual Meeting (2013).