An efficient data space conjugate gradient Occam’s method for three-dimensional magnetotelluric inversion

Weerachai Siripunvaraporn$^{1,2}$ and Weerachai Sarakorn$^3$

$^1$Department of Physics, Faculty of Science, Mahidol University, Rama 6 Road, Rachathewi, Bangkok 10140, Thailand. E-mail: scwsp@mahidol.ac.th
$^2$ThEPI, Commission of Higher Education, 328 Si Ayutthaya Road, Bangkok 10400, Thailand
$^3$Department of Mathematics, Faculty of Science, Mahidol University, Rama 6 Road, Rachathewi, Bangkok 10400, Thailand

Accepted 2011 May 14. Received 2011 May 14; in original form 2010 May 24

SUMMARY
In this paper, we start with the implementation of a data space conjugate gradient (DCG) method for 3-D magnetotelluric (MT) data. This code will be referred to as WSDCG3DMT. It is an extension of the 2-D method previously developed. Several experiments on both synthetic and real data sets show that WSDCG3DMT usually needs more computational time than the data space Occam’s inversion (OCCAM) for which the corresponding code is referred to as WSINV3DMT. However, the memory requirement of WSDCG3DMT is only a fraction of that of WSINV3DMT. Based on the knowledge gained from several studies of both codes, we have created a new hybrid scheme called the DCG Occam’s inversion (DCGOCC) and the corresponding code, WSDCGOCC3DMT, from combining aspects of the OCCAM and DCG methods. As with OCCAM, the DCGOCC method divides the inversion into two phases. In Phase I the misfit is brought down to a desired level. In Phase II unnecessary structures are smoothed out. Because its mathematical basis is of a similar form to that of DCG, its memory requirement is similarly low but more stable. However, DCGOCC is significantly faster than both methods. We demonstrate the computational performances with comparisons of all three methods with both synthetic and EXTECH field data sets.

Key words: Numerical solutions; Inverse theory; Numerical approximations and analysis; Magnetotelluric; Electromagnetic theory.

1 INTRODUCTION
Recently, the number of 3-D magnetotelluric (MT) surveys has substantially increased worldwide (e.g. Tuncer et al. 2006; Patro & Egbert 2008, among many others). This might be due to the fact that MT is increasingly being accepted by many geophysicists and seismologists. Another major factor is the improvements of the data acquisition units, the measurement sensors and their accessories. Examples of MT uses include geothermal explorations (e.g. Heise et al. 2008; Arnason et al. 2010), volcanoes and tectonic studies (Uyeshima 2007; Patro & Egbert 2008; Hill et al. 2009; Ingham et al. 2009) and ore explorations (Tuncer et al. 2006; Farquharson & Craven 2008; Türkoğlu et al. 2009; Goldak & Kosteniuk 2010). All of these have led to a higher demand for 3-D MT inversion codes for interpretation.

A number of 3-D MT inversion algorithms have been developed (e.g. Mackie & Madden 1993; Newman & Alumbaugh 2000; Zhidanov et al. 2000; Sasaki 2001; Mackie, personal communication 2002; Siripunvaraporn et al. 2004 2005; Sasaki & Meju 2006; Han et al. 2008; Lin et al. 2008, 2009; Farquharson & Craven 2008; Avdeev & Avdeeva 2009; Siripunvaraporn & Egbert 2009). All algorithms are designed to find the ‘best’ model that fits the data and is ‘geologically interpretable’. Recent reviews of these algorithms can be found in Siripunvaraporn (2011). One of the 3-D codes (and the only one currently available to the MT community) is the WSINV3DMT program by Siripunvaraporn et al. (2005). The algorithm is based on the Occam’s style inversion (OCCAM) introduced for 1-D MT data by Constable et al. (1987). OCCAM is known for its robustness and efficiency. However, its disadvantage is the large memory requirements and extensive computational time, particularly when it is applied to 2-D and 3-D modelling (Siripunvaraporn & Egbert 2000; Siripunvaraporn et al. 2005). To reduce both storage and calculation time, Siripunvaraporn & Egbert (2000) and Siripunvaraporn et al. (2004, 2005) transformed the original Occam’s inversion, which is a model space method, into the data space Occam’s algorithm. The transformation makes it practical for 3-D MT inversion on most computers. However, OCCAM still requires substantial memory to store the $N \times M$ sensitivity matrix, where $N$ and $M$ are the number of data and model parameters, respectively. Siripunvaraporn & Egbert (2007) used 2-D MT data to show that the large storage requirements can be avoided by using a data space conjugate gradient (DCG) approach.

In this paper, our first objective is therefore to extend the 2-D DCG method to 3-D cases. Hereafter, the 3-D data space CG program will be referred to as WSDCG3DMT. The second objective is to demonstrate the effectiveness of a new scheme, called the
DCG Occam’s inversion (DCGOCC) method, which is derived from the DCG method, inheriting its low memory usage while having a superior efficiency to both the OCCAM and DCG algorithms.

We start the paper with a brief review of both Occam’s inversion (WSINV3DMT) and the DCG method (WSDCG3DMT). Differences between both techniques will be clearly demonstrated through both theory and numerical experiments with synthetic data. Several observations from both WSINV3DMT and WSDCG3DMT are used to develop the hybrid scheme WSDGOC3DMT. To demonstrate their performance robustness and parallel implementation, all three codes are applied to the EXTECH data set. The results are compared and discussed.

2 THEORY OF THE DATA SPACE INVERSION ALGORITHMS

In this section, we briefly review the two main algorithms of this paper, which are the data space Occam’s algorithm for WSINV3DMT and the DCG method for WSDCG3DMT. For more details, see Siripunvaraporn & Egbert (2000) and Siripunvaraporn et al. (2005) for WSINV3DMT and Siripunvaraporn & Egbert (2007) for WSDCG3DMT, and Siripunvaraporn (2011) for a review of both methods and their algorithms.

2.1 WSINV3DMT: The data space Occam’s inversion

The idea of the Occam’s inversion (OCCAM) is to search for the minimum structure model generating the misfit at the target level. Mathematically, this is equivalent to seeking the stationary point of the unconstrained functional

\[ U(m, \lambda) = (m - m_0)^T C_m^{-1} (m - m_0) + \lambda^{-1} \{(d - F[m])^T C_d^{-1} (d - F[m]) - X^2\}, \]  

where \( m \) is a resistivity or conductivity model of dimension \( M \), \( m_0 \) is a base or prior model, \( C_m \) is a model covariance matrix which defines the model norm, \( d \) is the observed model with dimension \( N \), \( F[m] \) is the forward model response, \( C_d \) is a data covariance matrix, \( X^2 \) is the desired level of misfit and \( \lambda^{-1} \) is a Lagrange multiplier. Notice that the term OCCAM has been misused in many other publications. Here, we strictly follow the scheme first introduced by Constable et al. (1987).

Because \( U \) is a function of both \( m \) and \( \lambda \), finding its stationary point is not straightforward. A penalty functional

\[ W_1(m) = (m - m_0)^T C_m^{-1} (m - m_0) + \lambda^{-1} \{(d - F[m])^T C_d^{-1} (d - F[m])\}, \]  

is therefore minimized instead of \( U \). This is because when \( \lambda \) is fixed, \( \partial U/\partial m \) and \( \partial W_1/\partial m \) yield the same result. Hence minimizing \( W_1 \) with a series of \( \lambda \), and choosing \( \lambda \) for which the smallest minimum \( W_1 \) is achieved, is equivalent to minimizing the original functional \( U \).

To efficiently minimize \( W_1 \), its form as given in (2) must be first transformed into a data space penalty functional (Siripunvaraporn & Egbert 2000; Siripunvaraporn et al. 2005). The model response \( F[m] \) is linearized about model \( m_0 \) before it is differentiated with respect to the data space coefficient vector (see Siripunvaraporn & Egbert 2000; Siripunvaraporn et al. 2005, for more details). This yields the series of iterative approximate solutions

\[ m_{k+1} - m_0 = C_m^{-1} C_d^{-1} R^{-1} C_d^{-1} d, \]  

where \( J = \partial F/\partial m \) is an \( N \times M \) sensitivity matrix, \( d = d - F[m] + J(m - m_0) \), \( R = [\lambda I + C_m^{-1} C_d^{-1} C_d^{-1} C_m^{-1}] \) is the identity matrix and \( \lambda \) is the inversion iteration number.

To solve (3), OCCAM explicitly forms an \( N \times M \) matrix \( J \) and an \( N \times N \) matrix \( R \) and stores both in memory. \( R \) will later be decomposed with Cholesky decomposition and then solved with backward and forward substitutions. This scheme therefore requires a substantial amount of RAM to store the two matrices. This could prohibit a run on very large data sets, particularly in 3-D cases.

To search for stationary point of \( U \) using (1), as with the original Occam’s inversion (Constable et al. 1987), the inversion is divided into two phases. In Phase I, after forming \( J \) and \( R \), (3) is solved for various values of \( \lambda \) to find the model with the minimum RMS. The minimum RMS model is then used to compute the sensitivity matrix \( J \) for the next iteration. The iterative process is repeated until the misfit is brought down to the target misfit \( X^2 \). Once \( X^2 \) is achieved, Phase II varies \( \lambda \) while keeping the misfit at the desired level to search for the minimum norm model, that is the model with the minimum value of \( \|(m - m_0)^T C_m^{-1}(m - m_0)\| \).

2.2 WSDCG3DMT: the data space conjugate gradient inversion

WSDCG3DMT is the extension of the 2-D data space CG (DCG) inversion of Siripunvaraporn & Egbert (2007) to 3-D problems. The goal is to minimize \( W_1 \) using a fixed \( \lambda \) via iterative solutions given by (3). Instead of solving (3) with Cholesky decomposition, DCG solves (3) with a CG method because \( R \) is symmetric. One clear advantage of using CG to solve (3) is that the large \( N \times M \) sensitivity matrix \( J \) and cross-product matrix \( R \) are not explicitly formed and stored in the computer memory. Only a product of \( J \) or \( J^T \) with an arbitrary vector is required by solving one forward problem per period (see Mackie & Madden 1993; Newman & Alumbaugh 2000; Rodi & Mackie 2001; Siripunvaraporn & Egbert 2007; Lin et al. 2008). This allows DCG to be used for much larger data sets than OCCAM.

DCG can be divided into two main iterative loops. The outer inversion loop is designed to minimize \( W_1(m) \) in the same way as OCCAM except that \( J \) and \( R \) are not formed. The inner loop is used to minimize \( Rx = b \) in (3) with a conjugate gradient method where \( b = C_d^{-1} d \) and \( m_{k+1} - m_0 = C_m^{-1} C_d^{-1} x \) (see Barrett et al. 1994 for Preconditioned Conjugate Gradient algorithm). The full algorithm is summarized in fig. 2 of Siripunvaraporn & Egbert (2007).

As with other non-Occam’s inversions (e.g. Newman & Alumbaugh 2000; Rodi & Mackie 2001), the DCG method only minimizes \( W_1 \) for a pre-selected \( \lambda \) (Siripunvaraporn & Egbert 2007). Thus, to make it equivalent to (1), several values of \( \lambda \) must be used. In contrast to OCCAM, (3) is iteratively solved with each fixed value of \( \lambda \) until the misfit is reduced to the desired misfit \( X^2 \). The process is repeated with other pre-selected \( \lambda \) values. After evaluating the misfits corresponding to several values of \( \lambda \), it will be found that some models share the same desired misfit, \( X^2 \). To be equivalent to (1), the model with the smallest norm \( \|(m - m_0)^T C_m^{-1}(m - m_0)\| \) is chosen as the final solution to the inverse problem.

3 NUMERICAL EXPERIMENT ON WSINV3DMT AND WSDCG3DMT

Both codes have been run on many examples of synthetic and real data and similar results are obtained. Here, we use one typical synthetic data set as an example to demonstrate the concepts of both methods. Knowledge gained in this section will lead to the
new hybrid algorithm. Another example with real data will be given later in Section 5 to help demonstrate the efficiency of the new code.

As with Siripunvaraporn et al. (2005) and Siripunvaraporn & Egbert (2009), we use the same synthetic model to generate a synthetic data set for testing our codes. The synthetic model consists of two anomalies with resistivities of 1 and 100 Ωm, buried next to each other inside a 10 Ωm layer lying on top of a 100 Ωm half-space (Fig. 1). The model mesh for the inversion was discretized into 30 × 44 × 21 blocks in the x-, y- and z-directions, respectively, with seven air layers. This mesh is different from the forward modelling mesh used to generate the synthetic data. We invert the full complex-valued impedance data, $Z_{xx}$, $Z_{yy}$, $Z_{xy}$ and $Z_{yx}$ ($N_a = 8$). We simulate these data for 16 periods ($N_p = 16$) in a range from 0.031 to 1000 s at 40 sites ($N_s = 40$) located at regular intervals covering the two anomalies (solid dots in Fig. 1). Eight per cent Gaussian noise calculated from the magnitude of the data ($|Z_{xy}|$) was added to the impedance data. With this configuration, the number of model parameters, $M$ and the number of data $N$, are equal to $30 \times 44 \times 21 = 27720$ and $40 \times 16 \times 8 = 5120$, respectively.

In this experiment, all runs were performed on a single processor machine (Intel Core Two Duo 6400, 2.13 GHz machine with 2 GB of RAM). A larger model mesh or data set would not be possible on this machine in the case of WSINV3DMT. A 50 Ωm half-space was used as an initial model and prior model in all inversions.

### 3.1 WSINV3DMT: a synthetic example

WSINV3DMT was first run on this synthetic model. The convergence behaviour of Phases I and II are shown in Fig. 2. In Phase I, the values of $\lambda$ that produce the minimum RMS in each iteration are 0.1, 0.01 and 0.003 for the first, second and third iterations, respectively (solid squares for the first and second iterations in Fig. 2). However, on the third iteration, the minimum RMS is below the target misfit of 1. The code therefore searches for a $\lambda$, which yields a misfit of RMS = 1. The search process found that $\lambda = 0.17$ produces a model with a misfit of around RMS = 1 and a norm of 7981 (Fig. 3). Phase II starts on the fourth iteration from the model in Fig. 3. By varying $\lambda$, it can be seen that there are at least two models that produce a misfit of around RMS = 1. Because a larger $\lambda$ always produces a model with a smaller norm, a model from a larger $\lambda$ is therefore chosen as shown in iterations 4 and 5 of Fig. 2. Phase II helps reduce the model norms to 7558 and 7417 for iterations 4 and 5, respectively, while keeping the misfit at RMS = 1. The program is terminated at the next iteration because the code cannot find further models with RMS = 1. The CPU time is shown in Fig. 4 where Phase I finished within 575 min. The code spends about 450 min to smooth out the unnecessary structures. The inverted model from Phase I and the final inverted model from Phase II of WSINV3DMT are shown in Figs 3(a)–(c) for plane-views at different depths and Fig. 3d for a cross-section view, and Figs 5(a)–(d), respectively. Fig. 3 shows that the inverted model is rougher than the inverted model after passing through Phase II (Fig. 5).

### 3.2 WSDCG3DMT: a synthetic example

In the next experiment, we applied WSDCG3DMT to the same synthetic data. The computational time of WSDCG3DMT is mainly controlled by the stopping criteria used inside the CG algorithm when solving $\mathbf{Rx} = \mathbf{b}$ in eq. (3). Normally, the CG solver is terminated when the relative error ($r = \|\mathbf{Rx} - \mathbf{b}\|/\|\mathbf{b}\|$) reaches a given tolerance. The conditioned number of (3) is inversely proportional to the $\lambda$ used. A large conditioned number means the system requires more iterations (and more CPU time) to converge to a given tolerance. To get a ‘exact’ solution as obtained from a direct method, $r_{tol}$ typically must be set to less than $10^{-6}$. To reach this level, particularly for small $\lambda$, would require a significantly larger CPU time than by solving with a direct method in our 3-D case. This is also true in the 2-D case (Siripunvaraporn & Egbert 2007), although the system is a lot smaller.

Thus, to reduce the CPU time to a practical level, we run the code with different values of $r_{tol}$ for several values of $\lambda$. This experiment is similar to the experiments we conducted in the 2-D cases. The results obtained in the 3-D case are not significantly different from the 2-D studies so we briefly summarize them. At $r_{tol} = 10^{-2}$, the
Figure 2. Convergence behaviour of Phases I and II as a function of $\lambda$ of WSINV3DMT on synthetic data. Solid squares mark the values of $\lambda$ used for the next iterations.

Figure 3. The inverted models from Phase I of WSINV3DMT in plan-view at surface (a), at 3 km depth (b) and at 7.5 km depth (c). The cross-section views cutting across the two anomalies at $X = 0$ km is shown in (d). The solution is shown only in the central area around the anomalies, not for the full model domain.
A hybrid DCG–OCCAM inversion for 3-D MT data

Figure 4. Convergence of WSINV3DMT (squares) and WSDCG3DMT for various values of \( \lambda \) as a function of CPU time. Each symbol indicates one iteration.

Figure 5. The final inverted models from WSINV3DMT in plan-view at surface (a), at 3-km depth (b) and at 7.5-km depth (c). The cross-section views cutting across the two anomalies at \( X = 0 \) km is shown in (d). The solution is shown only in the central area around the anomalies, not for the full model domain.
number of CG iterations is significantly less than when solving with a smaller $r_{\text{tol}}$. The models obtained with $r_{\text{tol}} = 10^{-3}$ differ by less than 2 per cent from those generated with $r_{\text{tol}} = 10^{-2}$, but are more than 10 per cent different from models obtained with $r_{\text{tol}} = 10^{-1}$. Thus, $r_{\text{tol}} = 10^{-2}$ is assigned as the ‘optimal tolerance’ value providing acceptable accuracy with reasonable computational time.

For $r_{\text{tol}} = 10^{-2}$, the convergence behaviour of WSDCG3DMT from several $\lambda$ are shown in Fig. 6 as a function of $\lambda$ and in Fig. 4 as a function of CPU time. Fig. 6 shows that after a few iterations of WSDCG3DMT with small $\lambda$ (e.g. $\lambda < 0.01$), DCG breaks down as shown by a question mark in the third iterations of $\lambda = 0.1$ and 0.01. At this inversion iteration, after a few tens of CG iterations, the relative error becomes large to unreasonably large and no convergence to the solutions occurs. The cause for this divergence inside the CG loop is probably the loss in the orthogonality of matrix $R$. The relative errors becoming unreasonably large after a number of iterations (e.g. after 15 CG iterations) can be used as a criterion to detect the failure of the CG solver. Whenever it happens, the program is terminated to avoid wasting CPU time.

For very small $\lambda$, if DCG does not break down, we believe that it can reduce the misfit to the desired level as this has happened in some other examples not shown here. Although smaller $\lambda$ can produce models with misfits at the target level, the models often look very rough, containing many spurious features which are not geologically interpretable. We should therefore avoid using small $\lambda$.

For intermediate levels of $\lambda$ between 0.1 and 10, DCG can often converge to or even exceed the desired level of accuracy and it does not easily break down. In the case shown in Fig. 6, the range of $\lambda$ that leads to convergence is between 0.1 and 2. For larger $\lambda$ (>3 in Fig. 6), WSDCG3DMT can only reduce the misfit in the early iterations before idling above the target level.

To be equivalent to minimizing $U$, the model with the smallest norm is selected from the list of the models with misfits of about RMS = 1. In general from several synthetic studies, models from large $\lambda$ with misfit of RMS = 1 always produce smaller model norms than those from smaller $\lambda$. In our example here, $\lambda = 1$ is selected because it generates a model with a norm of 7454. This is almost the same level as obtained from WSINV3DMT in the previous experiment where the final model with the smallest norm at 7417 is generated from $\lambda \approx 2$ with a misfit of RMS = 1. This validates the equivalence of both methods in achieving the same stationary point of a functional $U$.

### 3.3 Comparisons of WSINV3DMT and WSDCG3DMT on a synthetic example: memory and CPU time

Most of the memory requirement of WSINV3DMT is from storing $J$ and $R$ matrices. With double precision this is around $8NM + 8N\pi$ bytes. This is about 1.4 GB in our test case. The code also requires less than 0.3 GB for storing $S$ (the forward modelling coefficient matrix), $\partial S / \partial m$, and other parts for miscellaneous computations. For WSDCG3DMT, we do not store $J$ and $R$ in memory and the algorithm requires only about 0.4 GB to store the many different matrices and vectors. This is approximately the same as the memory used for the miscellaneous computations in WSINV3DMT.

Fig. 4 shows the CPU time of both WSINV3DMT and WSDCG3DMT for each $\lambda$. WSINV3DMT converges to the desired level within 575 min and the final model is obtained after a total of 1025 min. WSDCG3DMT with $\lambda = 1$ converges to RMS = 1 after about 728 min which is shorter than the total time of WSINV3DMT, but longer than Phase I of WSINV3DMT. WSDCG3DMT with larger $\lambda$ takes less computational time than with a smaller $\lambda$. However, it will not converge. We have seen that to obtain the final model equivalent to minimizing $U$ of WSINV3DMT, one needs to run WSDCG3DMT with several values of $\lambda$ to select the model with the smallest norm producing the misfit at about RMS = 1. The total running time for all $\lambda$ used (assuming all converge) is significantly larger than the total time of WSINV3DMT. However, if one only

![Figure 6](https://academic.oup.com/gji/article-abstract/186/2/567/587531) Convergence behaviour of WSDCG3DMT on synthetic data. For each $\lambda$, the first iteration is at the top. The question marks indicate the breakdown of WSDCG3DMT at the next iteration. Dash-dot line indicates the minimum possible RMS that WSDCG3DMT can attain.
cares about the misfit, running WSDCG3DMT for just one value of \( \lambda \) may be enough and could produce a model that fits the data at an acceptable level.

As we stated earlier, the system of equations of DCG with large \( \lambda \) results in a small conditioned number. This is why solving (3) with smaller \( \lambda \) requires significantly more CPU time (due to the CG solver needing more iterations) than when solving with larger \( \lambda \). This differs from WSINV3DMT where the system is directly solved with the result that the CPU time is almost independent of \( \lambda \).

In a previous 2-D study (Siripunvaraporn & Egbert 2007), we did not directly compare the CPU time, but compared the number of forward modelling calls of each algorithm. In 3-D, comparing the number of forward calls can be misleading. As with the 2-D case, WSINV3DMT requires a fixed number of calls per inversion iteration to form the sensitivity and compute the misfit. This number is \( N_p N_s N_m + N_p (N_s + 1) \) where \( N_s \) is the number of values of \( \lambda \). In our WSINV3DMT experiment here, for the first iteration, \( N_s = 5 \), and so the number of forward modelling calls is 2656. For WSDCG3DMT, in each iteration the number of forward modelling calls depend on the number of CG iterations \( (N_{cg}) \) in the DCG inner loop, and is equal to \( 4N_p + 2N_p N_{cg} \) per inversion iteration as we previously discussed. In our WSDCG3DMT experiment, for the case \( \lambda = 1 \) and \( r_{tol} = 10^{-2} \), \( N_{cg} = 45 \) for the first iteration, and so the number of forward modelling calls is 1504.

Although for the first iteration WSDCG3DMT makes about a thousand less forward modelling calls than WSINV3DMT, its computational time is actually 206 min longer (Fig. 4). WSINV3DMT and WSDCG3DMT spend on average about 3.1 and 14 s, respectively, for each forward call. This is because the system of equations in the forward routines becomes more complicated when computing \( J_p \) or \( J^T q \) in WSDCG3DMT than when just forming \( J \) in WSINV3DMT. Hence testing the efficiency of the inversion by just counting the number of forward modelling calls, as was done in earlier publications (see, e.g. Newman & Alumbaugh 1997), can be misleading.

3.4 Comparisons of WSINV3DMT and WSDCG3DMT on a synthetic example: inverted models

The final inverted model from WSDCG3DMT with \( \lambda = 1 \) is shown in Figs 7(a)–(c) for plane-views at different depths and Fig. 7(d) for a cross-section view for comparison with those from WSINV3DMT (Fig. 5). Both models generate the same misfit of about RMS = 1. Both methods can recover the anomalies quite well. Some differences can be observed particularly the near surface. Resistivity values of the conductor from WSDCG3DMT are lower than those from WSINV3DMT, whereas resistivity values of the resistor from both are about the same. Because the inverted model from WSINV3DMT (Fig. 5) passed phase II of the inversion, it is slightly smoother than the model from WSDCG3DMT, which does not have Phase II of the inversion.

**Figure 7.** The final inverted models from WSDCG3DMT with \( \lambda = 1 \) in plan-view at surface (a), at 3 km depth (b) and at 7.5 km depth (c). The cross-section views cutting across the two anomalies at \( X = 0 \) km is shown in (d). The solution is shown only in the central area around the anomalies, not for the full model domain.
According to both theory and experiments, we have seen that DCG uses less memory than OCCAM because it does not explicitly form and store the sensitivity matrix, but DCG is slower. Here we propose a new scheme, the DCG Occam’s inversion (DCGOCC). We refer to the corresponding code as WSDCGOCC3DMT. The method combines techniques from both algorithms and results in a scheme that is faster than both and uses the same amount of memory as DCG but significantly less than OCCAM.

One way to use less memory is to solve (3) with the CG method as performed in WSDCG3DMT. Although memory usage can be significantly reduced, a direct application of the CG solver to OCCAM could be prohibited due to either the failure of the CG method if $\lambda$ is too small or the whole process taking significantly longer than the original WSINV3DMT code. This, along with the following observations from our several experiments, is the key to the formulation of the new scheme.

1. $\lambda$ too large: DCG may not be able to converge to the desired misfit (Fig. 6).
2. $\lambda$ too small: DCG could cause the CG solver to break down (Fig. 6).
3. CPU time in DCG is inversely related to $\lambda$ (Fig. 4).
4. For a given $\lambda$ in DCG, the CPU time for each iteration always decreases. For example, for $\lambda = 1$, the CPU time for the first, second and third iterations are 342, 209 and 95 min, respectively (Fig. 4).
5. Phase I in OCCAM varies $\lambda$ to decrease the RMS misfit (Fig. 2). However, if $\lambda$ is fixed, the RMS misfit can also be reduced as in DCG (Fig. 6).
6. The minimum RMS misfits of both methods are approximately the same (Figs 2 and 6).

(7) In Phase II of OCCAM, for constant misfit the model norm decreases with increasing $\lambda$ (Fig. 2).

DCGOCC uses the same mathematics as in the DCG algorithm to maintain the memory advantage over OCCAM. To improve the CPU time, the $\lambda$ value will be varied from iteration to iteration but in a different way from that done in OCCAM. A flowchart of the WSDCGOCC3DMT scheme is shown in Fig. 8.

As with OCCAM, the DCGOCC scheme is divided into two phases. Phase I is to bring the misfit down to the target misfit. Phase II is to smooth out the unnecessary structure. However, the main difference from OCCAM is how we vary $\lambda$. To minimize the computational time, only one $\lambda$ is used in each iteration for both phases (Fig. 8). OCCAM searches for the $\lambda$ that minimizes the misfit. This $\lambda$ value is not predictable. To avoid wasting CPU time calculating many different values of $\lambda$ as with the DCG method, we select a value of $\lambda$ that will result in a low misfit and require the least computational time.

With the criteria we used, the starting $\lambda$, which we denote by $\lambda_{ini}$, should be large to obtain a low misfit and CPU time. If a large value of $\lambda$ results in failure to converge to the desired level, $\lambda$ in the next iteration is reduced by a factor of $\epsilon_I$ (e.g. $\epsilon_I = 10$). This is repeated until the calculation converges to the desired level or $\lambda$ reaches $\lambda_{min}$ (e.g. $\lambda_{min} = 0.1$). Whenever $\lambda$ is below $\lambda_{ini}$, it will set equal to $\lambda_{min}$ (Fig. 8). This is to avoid a very rough model and the breaking down of the CG solver due to too small a value of $\lambda$. Whenever the CG solver breaks down, $\lambda$ will be increased by the same factor $\epsilon_I$. A reduction in $\lambda$ was used before in Avdeev & Avdeeva (2009), Kelbert et al. (2008), Haber et al. (2000) and Newman & Hoversten (2000) for different reasons. For example, Kelbert et al. (2008) reduced it when their inversion algorithm failed to reduce the misfit.

When the misfit is below the desired level, Phase II starts (Fig. 8). Based on our observation that the model norm is smaller for larger

Figure 8. Flowchart of WSDCGOCC3DMT. The scheme is divided into two phases. Phase I is shown on the left in black. Phase II is on the right in blue.
\( \lambda \), the value of \( \lambda \) is then increased by a factor of \( \varepsilon_2 \) (e.g. \( \varepsilon_2 = 3 \)). The goal is to obtain a model with a smaller norm while maintaining the misfit at the desired level. If the larger value of \( \lambda \) cannot produce the model with the desired misfit, there is an option to set \( \lambda \) to the average of the previous two values of \( \lambda \). The process of Phase II is repeated until no larger \( \lambda \) can produce the model at the desired misfit.

### 4.1 WSDCGOCC3DMT: a synthetic example

To check the efficiency of the WSDCGOCC3DMT code, we applied it to the same synthetic data set generated from the model in Fig. 1 as was used to test WSINV3DMT and WSDCG3DMT. Four values of \( \lambda_{\text{ini}} \) are used (\( \lambda_{\text{ini}} = 10000, 1000, 100 \) and 10) with \( \varepsilon_1 = 10 \) and \( \varepsilon_2 = 3 \). The convergence rates of DCGOCC in Fig. 9 are compared with those of OCCAM and DCG with \( \lambda = 1 \).

All WSDCGOCC3DMT runs converge to or below the desired misfit within three to four iterations. All runs spend two more iterations in Phase II. The most important information learned from this experiment is that all WSDCGOCC3DMT runs are faster than both WSINV3DMT and WSDCG3DMT with \( \lambda = 1 \).

The lowest computational time is achieved when \( \lambda_{\text{ini}} = 100 \). For the first iteration, DCGOCC behaves the same way as DCG with \( \lambda = 100 \) in both CPU time and misfit. Then, \( \lambda \) is decreased to 10 and 1 and 0.1 in the second, third and fourth iterations, and the resulting misfits are 2.4, 1.3 and 0.8, respectively. This completes Phase I of the inversion. Phase II starts where \( \lambda \) is increased to 0.3 and 0.9 in the fifth and sixth iterations, respectively. Both iterations in Phase II can generate a model with RMS = 1 but with a smaller model norm. When \( \lambda \) is 2.7, it fails to produce a model with RMS = 1. The program is therefore terminated.

When \( \lambda_{\text{ini}} \) is much larger than 100, redundant computations occur in the first few iterations resulting in a slightly longer CPU time. This is because an iteration takes a shorter time for larger \( \lambda \). However, if \( \lambda_{\text{ini}} = 10000 \), the misfit is not greatly reduced. For the next iteration, when \( \lambda \) is decreased to 1000, the misfit happens to be almost the same as that from \( \lambda_{\text{ini}} = 10000 \). The first iteration of \( \lambda_{\text{ini}} = 1000 \) is therefore redundant. This also occurred for \( \lambda_{\text{ini}} = 1000 \). Using \( \lambda_{\text{ini}} \leq 10 \) also requires more computational time than with \( \lambda_{\text{ini}} = 100 \). In this small \( \lambda_{\text{ini}} \) case, the first iteration is much slower and \( \lambda \) decreases quickly to \( \lambda_{\text{min}} \) in the next few iterations. This would demand a large CPU time. This is therefore less effective and is

![Image](https://example.com/image.png)

**Figure 9.** Convergence behaviour as a function of time of WSDCGOCC3DMT applied to the synthetic data with various \( \lambda_{\text{ini}} \) between 10 and 10 000. Results from WSINV3DMT (square) and WSDCG3DMT (circle) with \( \lambda = 1 \) are also shown for comparison.
similar to running DCG with $\lambda = \lambda_{\text{ini}}$. We should thus avoid a $\lambda_{\text{ini}}$ in DCGOCC which is too small or too large.

In many other examples, we found that $\lambda_{\text{ini}}$ between 100 and 1000 seems to be very reasonable for the WSDCGOCC3DMT code. In addition, $\epsilon_I$ around 10 is suitable in many cases. If $\epsilon_I$ is too small, redundant computations can also occur. If it is too large, WSDCGOCC3DMT would carry out more CG iterations making it less efficient.

The final inverted model from WSDCGOCC3DMT with $\lambda_{\text{ini}} = 100$ is shown in Figs 10(a)–(c) for plane-views at different depths and Fig. 10(d) for a cross-section view in comparison with those from WSINV3DMT (Fig. 5) and WSDCG3DMT (Fig. 7). As with the other method, WSDCGOCC3DMT can recover both anomalies. The inverted model from WSDCGOCC3DMT is similar to that from WSINV3DMT because both models are the results from the end of the Phase II of the inversion.

**5 APPLICATION OF WSDCGOCC3DMT, WSDCG3DMT AND WSINV3DMT TO EXTECH DATA**

Here, we compare the results of applying these three algorithms to real data. The real data used is the EXTECH data set (Tuncer et al. 2006) conducted around the McArthur River mine, Saskatchewan, Canada (fig. 2 of Tuncer et al. 2006). It consists of the impedance tensor ($Z_{xx}$, $Z_{xy}$, $Z_{yx}$ and $Z_{yy}$) and the vertical magnetic field transfer function ($T_{zx}$ and $T_{zy}$) for 131 stations and 16 periods (from 8000 to 5 Hz). Hence, $N = 25 152$. In previous runs using WSINV3DMT (Tuncer et al. 2006; Siripunvaraporn & Egbert 2009) the misfits cannot be reduced to the desired level of $\text{RMS} = 1$. To achieve the level of $\text{RMS} = 1$, the minimum error bars must be slightly increased to 20 per cent of $(|T_{xx}|^2 + |T_{xy}|^2)^{1/2}$ and 10 and 80 per cent of $|Z_{xy}Z_{yx}|^{1/2}$ for the off-diagonal terms and for the diagonal terms, respectively. The minimum error bars for the diagonal terms and VTF are set to be larger because the signal to noise ratios of these signals is very low for most sites after observing the data. Thus, in all runs, wherever the error bars obtained from the data processing program are smaller than the minimum error bars, they are replaced with the minimum error bars.

The slight increase of minimum error bars does not much alter the inverted solutions from the previous runs (Siripunvaraporn & Egbert 2009). Different minimum error bars can produce different results, particularly if large difference was used. However, these will occur in every inversion algorithm (e.g. NLCG of Rodi & Mackie 2001; Farquharson & Craven 2008; among many examples), not just to these three algorithms. Discussion on different minimum error bars to the inverted models is then not addressed here. However, by setting minimum error bars too small, a conjugate gradient solver inside WSDCG3DMT are prone to fail than with the direct solver used inside WSINV3DMT.

A 1000 $\Omega$m half-space is used as an initial model and a prior model ($m_0$) and is discretized by $56 \times 56 \times 33$ (+7 air layers) cells.
Hence, \( M = 103 \,488 \). A value of 1000 \( \Omega m \) was chosen because it produced a smallest misfit than any of the others half-space values tried.

### 5.1 Parallel implementation

Because the EXTECH data set and model mesh used are very large, all codes are impossible to run on a single processor machine with 2 GB of RAM. To execute the codes and at the same time to speed up the inversion, all codes must be parallelized and run on a cluster computer. Here, a cluster computer with eight processor nodes (\( N_{\text{pro}} = 8 \)) is used.

WSINV3DMT has already been parallel implemented by distributing forward modelling calls of \( N_p/N_{\text{pro}} \) frequencies to one node via message passing interface libraries (Siripunvaraporn & Egbert 2009). With 16-period data, two periods are therefore computed on each processor node. The distribution helps relieve the memory limitation problem and also decreases the runtime but less so than the theoretical speed up (Siripunvaraporn & Egbert 2009).

Although memory is not an issue for both DCG and DCGOCC methods, large computational runtime is a major challenge because of numerous calls to the forward modelling routine. Thus, as with WSINV3DMT, we parallelize our WSDCG3DMT and WSDCGOCC3DMT codes over frequencies. For DCG and DCGOCC, the parallelization is relatively simple; we just distribute the forward modelling calls of \( N_p/N_{\text{pro}} \) frequencies to each processor when computing the misfit, \( J_p \), and \( J^T_q \). Therefore, no data exchange between processors is required. This is in contrast to WSINV3DMT where it is necessary to form and store the cross-product matrix \( R \), which requires multiplications of data between processes (Siripunvaraporn & Egbert 2009).

Because the memory requirement is now distributed to each node, WSINV3DMT requires about 7 GB of RAM per node to store all matrices and vectors. A larger model domain would be impossible for WSINV3DMT because of memory limitations. In contrast, both WSDCG3DMT and WSDCGOCC3DMT use significantly less memory (about 1.5 GB of RAM per node) to perform the inversion. Both can therefore be applied to a large data set and a large mesh. However, here we use the same parameter values to allow comparisons to be made.

### 5.2 Convergence of WSINV3DMT, WSDCG3DMT and WSDCGOCC3DMT

The convergence of WSINV3DMT is shown in Fig. 11. WSINV3DMT finished Phase I within three iterations taking about 950 min, and with another iteration to complete Phase II taking an additional 380 min. As in Fig. 2 for the synthetic example, \( \lambda \) is varied in each iteration. In this case, the \( \lambda \) values that produce models generating the minimum RMS are 1, 0.3 and 0.3 for the first, second and third iterations, respectively. For the fourth iteration, \( \lambda \) moves to 1 to obtain the model with the smallest norm with the desired misfit.

We ran WSDCG3DMT with various values of \( \lambda \). As with the synthetic case, when \( \lambda > 1 \), DCG cannot converge to the desired level. When \( \lambda < 0.3 \), it breaks down. Because \( \lambda = 1 \) can produce a model with smallest norm with RMS = 1, this model is therefore used. The convergence of WSDCG3DMT with \( \lambda = 1 \) is also shown in Fig. 11. With \( \lambda = 1 \), WSDCG3DMT uses three iterations and spends about 1140 min, which is slightly longer than Phase I of WSINV3DMT, to converge to the desired RMS = 1. However, the total time for running WSDCG3DMT with all values of \( \lambda \) would be more than 3000 min.

For the DCGOCC scheme, we used \( \lambda_{\text{ini}} = 100 \) and found that it converges to the desired RMS faster than other algorithms and also faster than with other values of \( \lambda_{\text{ini}} \) in other experiments. Its

---

**Figure 11.** Convergence behaviour as a function of time of WSINV3DMT, WSDCG3DMT with \( \lambda = 1 \), and WSDCGOCC3DMT with \( \lambda_{\text{ini}} = 100 \) applied to EXTECH data.
convergence is shown in Fig. 11 along with those of the OCCAM and DCG methods. As with DCG, it spends three iterations on Phase I, but its CPU time (500 min) is about half that of WSINV3DMT and WSDCG3DMT. The fourth iteration of WSDCGOCC3DMT, by moving to larger $\lambda = 3$, fails to generate a model with RMS = 1. The program is therefore stopped.

Although it appears that the convergence of WSDCG3DMT and WSDCGOCC3DMT to the EXTECH data is similar to those of the synthetic data, this is just a coincidence. In fact, in many applications that we have tested the algorithms on, the range of the $\lambda$ values that can converge to the desired level is between 0.05 and 5. Therefore, using a value of $\lambda_{ini}$ between 100 and 1000 for WSDCGOCC3DMT is reasonable for most cases.

5.3 Final model and fit to the data

The final models from the three algorithms are slightly different. Here, the inverted model from the third iteration of WSDCGOCC3DMT starting with $\lambda_{ini} = 100$ is shown in Fig. 12. There are two conductors. The first conductor is on the eastern part of the profiles lying in the NE–SW direction starting at a depth of 500 m. The other is the northern conductor running from a depth of 800 m to 1.3 km. The conductor is consistent with the existence of graphitic faults where the uranium usually deposits above. Detailed interpretation of Fig. 12 will be presented elsewhere. A discussion of this EXTECH data set can be found in Tuncer et al. (2006) and Farquharson & Craven (2008).

As with the synthetic data, the calculated off-diagonals and the vertical magnetic transfer function fit the observed data quite well. Because we also increase the minimum error bars of the on-diagonals, the calculated on-diagonals do not fit well with the observed data. Inverting without the on-diagonal terms would produce a model quite similar to that of Fig. 12. Here, we use all the terms just to demonstrate the efficiency of the code that can handle a large data set.

6 CONCLUSIONS

In this paper, we first implemented and extended the DCG inversion for 3-D magnetotelluric data (WSDCG3DMT). Numerical experiments on various 3-D synthetic and real data show that WDCG3DMT with $\lambda$ between 0.5 and 5 can converge to the desired level of misfit (RMS = 1). However, it often takes more CPU time than the data space Occam’s inversion (WSINV3DMT). Because the whole sensitivity matrix and cross-product matrix are not explicitly formed and stored as in WSINV3DMT, the memory usage of WDCG3DMT is minimal and a fraction of that of WSINV3DMT. This makes WDCG3DMT attractive for its practical use for large to very large data sets.

Based on the numerical experiments using WDCG3DMT, the number of CG iterations is inversely proportional to $\lambda$. Although a large $\lambda$ requires fewer CG iterations, it rarely converges to the desired misfit. In contrast, a smaller $\lambda$ requires more CG iterations per inversion iteration but it can converge to the desired level of misfit. However, if $\lambda$ is too small, it can diverge. The computational time depends greatly on the number of CG iterations. Hence, to reduce CPU time, the number of CG iterations per main inversion iteration must be minimized. In addition, a model generated from a large $\lambda$ has a lower norm than a model from a smaller $\lambda$.

Information learned from many studies has inspired us to develop the DCGOCC scheme that combines aspects of the DCG and the OCCAM methods. Mathematically, WSDCGOCC3DMT is almost identical to WSDCG3DMT. Therefore, the memory usage is already minimal. To speed up the process of convergence, $\lambda$ is reduced from a large value (e.g. $\lambda_{ini} = 100$) in the first iteration by a factor $\varepsilon_1$ (e.g. $\varepsilon_1 = 10$) to a smaller $\lambda$ in later iterations. The process is continued until a misfit below the desired level is obtained. This completes Phase I of the inversion. The reduction of $\lambda$ helps reduce the time taken to converge to the desired misfit to almost half that of both the DCG and OCCAM methods. If the objective of the inversion is just to find a model that fits the data to an acceptable

Figure 12. The inverse solution at various depths from the third iteration of the WSDCGOCC3DMT method with $\lambda_{ini} = 100$. The EXTECH data used here consists of the vertical magnetic transfer function and the full impedance tensor at 131 sites and 16 periods. The crosses indicate the locations of the stations.
level, WSDCGOCC3DMT can achieve that faster than both WSDC3DMT and WSINV3DMT. Phase II of WSDCGOCC3DMT, which is to search for the minimum structure, increases $\lambda$ to find a model that can produce a misfit at the desired level. Phase II is terminated when no larger $\lambda$ can be used to fit the data at the desired level. Phase II requires slightly less computational time than that of WSINV3DMT. Hence, the total CPU time for both phases is less than for the other methods.

**Acknowledgments**

This research has been supported by the Thai Center of Excellence in Physics (TheEP) and by the Thailand Research Fund (TRF: RMU5080025) to W. Siripunvaraporn and by a Development and Promotion of Science and Technology Talents Project (DPST) scholarship to W. Sarakorn. The authors thank Anna Avdeeva, another anonymous reviewer, and the editor, Oliver Ritter, for their comments to help improve the paper. The authors thank Jim Craven for allowing us to use the EXTTECH data sets and Michael Allen for editing the English of the paper.

**References**


