Seismic reflectivity inversion using an L1-norm basis-pursuit method and GPU parallelisation

Ruo Wang¹, Yanhua Wang² and Ying Rao³,*

¹ The Key Laboratory of Unconventional Oil & Gas Geology, Oil and Gas Survey, China Geological Survey, Beijing 100083, China
² Centre for Reservoir Geophysics, Resource Geophysics Academy, Imperial College London, London, SW7 2BP, UK
³ State Key Laboratory of Petroleum Resources and Prospecting, China University of Petroleum (Beijing), Beijing 102249, China

*Corresponding author: Ying Rao. E-mail: raoying@cup.edu.cn

Received 24 April 2020, revised 21 May 2020
Accepted for publication 3 June 2020

Abstract
Seismic reflectivity inversion problem can be formulated using a basis-pursuit method, aiming to generate a sparse reflectivity series of the subsurface media. In the basis-pursuit method, the reflectivity series is composed by large amounts of even and odd dipoles, thus the size of the seismic response matrix is huge and the matrix operations involved in seismic inversion are very time-consuming. In order to accelerate the matrix computation, a basis-pursuit method-based seismic inversion algorithm is implemented on Graphics Processing Unit (GPU). In the basis-pursuit inversion algorithm, the problem is imposed with a L1-norm model constraint for sparsity, and this L1-norm basis-pursuit inversion problem is reformulated using a linear programming method. The core problems in the inversion are large-scale linear systems, which are resolved by a parallelised conjugate gradient method. The performance of this fully parallelised implementation is evaluated and compared to the conventional serial coding. Specifically, the investigation using several field seismic data sets with different sizes indicates that GPU-based parallelisation can significantly reduce the computational time with an overall factor up to 145. This efficiency improvement demonstrates a great potential of the basis-pursuit inversion method in practical application to large-scale seismic reflectivity inversion problems.

Keywords: basis pursuit, conjugate gradient method, GPU, L1-norm, parallelisation, reflectivity inversion, seismic inversion

1. Introduction
In seismic reflectivity inversion, a seismic profile can be treated as a number of layers and a reflection represents an interface separating layers of the subsurface media (Wang & Wang 2016). When using a basis-pursuit method to formulate a seismic reflectivity inversion problem, a reflector pair that defines a layer is decomposed into an even and an odd dipole. The decomposition for all reflectors forms a huge-size dictionary (Zhang & Castagna 2011), which leads to the very large seismic response matrix that is needed for the inversion problem. Therefore, the long computational time for resolving such a large-scale system is a considerable factor for seismic reflectivity inversion, especially when using the basis-pursuit method to form the inverse problem.

The basis-pursuit method is a technology for decomposing a seismic signal into an optimal superposition of dictionary elements (Zhang et al. 2012; Sen & Biswas 2015; Ostoori et al. 2018). Applying dipole decomposition can avoid the interference of the seismic response between...
the layers, and the retrieved result will have an increased resolution. The optimal quality here refers to the inversion resolution, defined by an L1-norm of the resultant reflectivity series. This L1-norm constrained inversion method will produce a sparse solution for a linear system (Wang 2003). Therefore, we develop seismic reflectivity inversion algorithm by using the basis-pursuit method to decompose reflectors and the L1-norm constraint for the sparsity of the solution.

To improve the algorithm efficiency by reducing the calculation time, a direct way is to develop a parallel code for implementation. Among various possible implementations, a Graphics Processing Unit (GPU) is a good candidate due to its highly parallel structure. It is more effective at processing large blocks of data than a general-purpose CPU. Nowadays GPU-accelerated algorithms are widely used in computational physics. In the area of geophysics and geoscience, the applications of GPUs are also of great interest. For example, in seismic geophysics, the GPU-based algorithm has been developed for solving 3D wavefield simulations (Weiss & Shragge 2013; Xu & Liu 2018), 3D reverse-time migration (Shi & Wang 2016; Li et al. 2018; Zhou et al. 2018) and geostatistical inversion (Liu & Grana 2019). These implementations have employed multiple GPUs to get a satisfactory result with much lower time consumption.

The GPU is a highly parallel structure that can manipulate a computer image efficiently. It is much faster than a CPU because it has thousands of cores that are optimised for executing multiple tasks at the same time, while a CPU only has a few. Thus, GPU parallelisation is more suitable for parallel computing while a CPU is good at sequential serial processing (Sanders & Kandrot 2010). In this paper, while we experiment the GPU-based basis-pursuit method for seismic reflectivity inversion, we will study the influence of matrix size on the algorithm efficiency analytically. We shall demonstrate that the computational time of the basis-pursuit method is drastically reduced, while precision of inversion is maintained.

2. The basis-pursuit inversion method

A seismic trace is a time-domain convolution:

\[ d = W r, \]

where \( d \) is a vector representing the seismic trace, \( W \) is a wavelet operator in which each row is a time-shifted version of the wavelet and \( r \) is the vector of the reflectivity series.

The basis-pursuit method presents a reflector pair \( r = [r_1, r_2]^T \) as a weighted sum of an even pair and an odd pair:

\[ r = ar_{even} + b r_{odd}, \]

where \( r_{even} \) and \( r_{odd} \) are dipoles defined by delta function \( \delta(t) \), and \( a \) and \( b \) are two weighting coefficients. As illustrated in figure 1, the even–odd pair decomposition is implemented on the reflectivity coefficients of the same layer and, thus, the reflectivity pair \( (r_1, r_2) \) refer to the top and base reflectors of any single layer.

Considering the entire time range of a seismic trace, the reflector dipoles are expressed as

\[
\begin{align*}
    r_{\text{even}}(t, l, n) &= \delta(t - l\Delta t) + \delta(t - l\Delta t + n\Delta t), \\
    r_{\text{odd}}(t, l, n) &= \delta(t - l\Delta t) - \delta(t - l\Delta t + n\Delta t),
\end{align*}
\]

(3)

where \( \Delta t \) is the time sampling rate, \( t - l\Delta t \) represents the time scale at the bottom of a layer, \( n\Delta t \) is the time thickness between the two reflectors, and thus \( t - l\Delta t + n\Delta t \) represents the corresponding time scale at the top of the same layer. The time scale \( l \) ranges from the first sample point to the last one of a seismic trace.

Then, the reflectivity series becomes

\[
r \equiv r(t) = \sum_{n=1}^{N} \sum_{l=1}^{L} [a_{nl} r_{\text{even}}(t, l, n) + b_{nl} r_{\text{odd}}(t, l, n)],
\]

(4)

where \( a_{nl} \) and \( b_{nl} \) are the coefficients corresponding to the even and odd reflectivity pairs, respectively, \( L \) is the length of a seismic trace and \( N \) represents the maximum layer thickness of the pairs. The thickness parameter \( n \) ranges from one to a constant \( N \). Note that a single impulse is also included for properly representing a layer that has an interface be outside of the available time range (Zhang & Castagna 2011; Zhang et al. 2012).

Equation (4) is rewritten in a vector-matrix form as

\[
r = Hm,
\]

(5)

where \( H \) is a matrix formed by a collection of dipole reflectors, and \( m \) is a vector consisted of coefficients \( \{a_{ij}, b_{ij}\} \).

The seismic trace, based on the convolution model of equation (1), becomes

\[
d = Gm,
\]

(6)
where $G = WH$ is the seismic response matrix that represents the convolution of each dipole vector in $H$ with the known wavelet.

For solving equation (6), an objective function can be set up conventionally as

$$\phi(m) = \|d - Gm\|_2^2 + \mu \|m\|_1, \quad (7)$$

where $\|d - Gm\|_2^2$ is the L2 norm of data misfit, $\|m\|_1$ is the L1-norm model constraint and $\mu$ is a trade-off parameter for balancing the contribution of the data misfit term and the model constraint. Once the model $m$ is obtained, the reflectivity series $r$ is generated using equation (5).

3. The linear programming problem

The L1-norm inverse problem (7) can be reformulated as a linear programming problem in a canonical form (Dantzig 1982; Dantzig & Thapa 1997):

$$\min (c^T x), \quad \text{subject to} \quad Ax = d, \quad x_i \geq 0,$$

where

$$A = [G - G], \quad x = [u \quad v], \quad m = u - v. \quad (9)$$

Vector $c$ in equation (8) represents the regularisation parameter $\mu$ for the L1-norm model constraint:

$$\mu \|m\|_1 = \mu \|u - v\|_1 \leq \mu (\|u\|_1 + \|v\|_1) = \mu (u + v) = c^T x. \quad (10)$$

Thus, $c = \mu e$ and $e$ is a vector with all elements being equal to a value of 1. Note that the derivation in equation (10) follows the bound $x_i \geq 0$ presented in equation (8).

According to the duality in linear programming, the minimisation problem (8) can be converted into a maximisation problem (Khachivan 1979; Karmarkar 1984; Matoušek & Gärtner 2007):

$$\max (d^T y), \quad \text{subject to} \quad A^T y + z = c, \quad z_i \geq 0,$$

where $y$ and $z$ are two dual variable vectors. The variable vector $y$ has the same number of elements as seismic trace vector $d$, and the variable vector $z$ has the same number of elements as $x$.

For this primal-dual problem, there are three measurements:

$$\begin{align*}
\|Ax - d\|_2^2 \\
\|A^T y + z - c\|_2^2 \\
z^T x \approx 0.
\end{align*} \quad (12)$$

The first two measurements are the infeasibility of the primal and dual problems, respectively. The last one is the duality gap between the primal objective $\min (c^T x)$ and the dual objective $\max (d^T y)$ and is derived from $c^T x - d^T y = c^T x - y^T d = (c - A^T y)\frac{x}{2} = z^T x$. These measurements cannot be set to be zero directly. However, using a barrier method or an interior-point method (Gill et al. 1991; Potra & Wright 2000; Boyd & Vandenberghe 2004; Wright 2005), small perturbations are introduced to these measurements and establish a linear system of equations as the following:

$$F(x, y, z) = \begin{pmatrix} A x - d + y_2^2 y \\ A^T y + z - c - y_2^2 x \end{pmatrix} = 0, \quad (13)$$

where $y_2^2$ and $y_2^2$ are small positively-valued parameters, $Z$ is a diagonal matrix representing vector $z$ and $\lambda$ is the barrier parameter. Note that each of the three measurements in equation (12) is a scalar and is presented as an inner product of two vectors, and a small perturbation in equation (13) is added to each multiplied component of the inner products.

Given an initial solution estimate $(x, y, z)$, linear system (13) can be solved using a first-order perturbation method:

$$F(x, y, z) + \begin{pmatrix} \frac{\partial F}{\partial x} & \frac{\partial F}{\partial y} & \frac{\partial F}{\partial z} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = 0, \quad (14)$$

where $(\Delta x, \Delta y, \Delta z)$ are the updates to the solution estimate $(x, y, z)$. Equation (14) may be written explicitly as

$$\begin{pmatrix} A \Delta x + y_2^2 \Delta y \\ A^T \Delta y + \Delta z - y_2^2 \Delta x \\ Z \Delta x + X \Delta z \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}, \quad (15)$$

where $X$ is a diagonal matrix representing vector $x$, and

$$\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} d - Ax - y_2^2 y \\ c + y_2^2 x - A^T y - z \end{pmatrix} \begin{pmatrix} \lambda e - Z x \end{pmatrix}. \quad (16)$$

This column vector is equal to $-F(x, y, z)$. The linear system of equations (15) can be re-arranged to (Chen et al. 1998)

$$\Delta y = (ADA^T + y_2^2 I)^{-1} \left[q_1 + AD (q_2 - X^{-1} q_3)\right], \quad (17)$$

$$\Delta x = D(A^T \Delta y - q_2 + X^{-1} q_3), \quad (18)$$
\[ \Delta z = -X^{-1}(Z\Delta x - q), \]

where
\[ D = (X^{-1}Z + \mu^2I)^{-1}. \]

In each iteration, \( \Delta y \) is solved from equation (17) using a conjugate gradient method, and then \( \Delta x \) and \( \Delta z \) are derived in sequence.

For the initial solution \( x \), equation (6) is presented as a least-squares problem (Wang 2016)
\[ m = (G^TG)^{-1}G^Td, \]
and solved also by the conjugate gradient method. Then, the initial solution \( x \) is set as
\[ u_i = \begin{cases} m_i, & \text{for } m_i \geq 0, \\ 0, & \text{otherwise}. \end{cases} \]
\[ v_i = \begin{cases} -m_i, & \text{for } m_i < 0, \\ 0, & \text{otherwise}. \end{cases} \]

Once the final solution \( x \) is obtained, the final model estimate is given by \( m = u - v \), subtracting the second half part of \( x \) from the first half part of \( x \).

4. Conjugate gradient method with GPU parallelisation

As discussed in the last section, two core problems to be solved during the process of the basis-pursuit method are the calculation of the initial model by the least-squares method in equation (21) and the calculation of Newton search directions \( (\Delta x, \Delta y, \Delta z) \) in equation (17). If assuming that the maximum interval of the dipoles is \( N \), and the number of sample points of the input data is \( n_1 \), the size of matrix \( G \) in equation (21) will be \( n_1 \cdot [2(N + 1) \cdot n_1 - (N + 1) \cdot N] \), which is much larger than the size of the input data. The size of the matrix \( A \) in equation (17) is \( n_1 \cdot [2 \cdot (2N + 1) \cdot n_1 - 2 \cdot (N + 1) \cdot N] \), which is double in the column size when compared with matrix \( G \). For solving these large-size problems, we use the GPU to accelerate computation.

A GPU has a high computational density and memory bandwidth, and has been popular for two decades. It was originally used for real-time rendering to relieve stress from the CPU. Nowadays, the GPU-based parallel technique has gained broad attention in computational physics because of its suitability to floating point operation and parallel operation. In 2007 Nvidia released an application programming interface (API), called CUDA. This parallel computing architecture for NVIDIA GPU implementation supports a standard language such as C and is supported by most operating systems such as Windows and Linux (Nvidia 2010).

CUDA Basic Linear Algebra Subroutines (CUBLAS) is a library on top of CUDA runtime and allows users to get access to the computational resources of GPU easily. It should be noted that to call CUBLAS functions, the input matrices must be first transformed into vectors with column major in GPU memory space. The obtained result is also a column-major stored vector and must be transformed back into matrix form with row-major storage before being used by other non-CUDA functions (Nvidia 2017).

For a CPU, only a few cores are employed in the structures, making them easier to adopt for serial processing, while GPUs contain far more cores enabling them to handle multiple tasks simultaneously. Due to their different features, it is powerful to combine these two types of processing units in many algorithms, where the sequential part is processed by CPUs and computation of intensive functions is allocated to GPUs.

Figure 2 shows the flowchart of a GPU-accelerated seismic reflectivity inversion using the basis-pursuit method. We use a GPU to accelerate the inversion mainly for the conjugate gradient method for solving equations (17) and (21) (Hestenes & Stiefel 1952). We denote these two equations in a general form of \( Ax = b \), where \( b \) is a column vector, \( x \) is the solution we seek, and \( A \) is a general square matrix and is not the matrix defined in equation (9). A fully parallelised conjugate gradient method is summarised as Algorithm 1.

This parallelised implementation has an identical pseudo code as the implementation of the conventional conjugate gradient method (Wang 2016). However, each operation is fully parallelised using CUBLAS:

1) \( \text{cublasSgemv}(A, p^{(k)}, u^{(k)}) \) is for the matrix-vector multiplication \( u^{(k)} = Ap^{(k)} \);
2) \( \text{cublasSdot}(a, b) \) is for the inner product of any two vectors \((a, b)\);
3) \( \text{cublasSaxpy}(a, p^{(k)}, x^{(k)}) \) is the implementation of vector update \( x^{(k+1)} = x^{(k)} + a p^{(k)} \);
4) \( \text{cublasSscal}(\beta, p^{(k)}) \) and \( \text{cublasSaxpy}(1, r^{(k)}, p^{(k)}) \) together is for another type of vector update \( p^{(k+1)} = \beta p^{(k)} + r^{(k+1)} \).
Algorithm 1. GPU-parallelised conjugate gradient method

Initialise the method by setting \( (x^{(0)}, p^{(0)}, r^{(0)}) \).

Then, for \( k = 1, 2, \ldots \),

\[
\begin{align*}
\alpha_k &= \text{cublasSdot}(r^{(k)}, r^{(k)}) / \text{cublasSdot}(p^{(k)}, u^{(k)}) , \\
x^{(k+1)} &= \text{cublasSaxpy}(\alpha_k, p^{(k)}, x^{(k)}) , \\
r^{(k+1)} &= \text{cublasSaxpy}(-\alpha_k, u^{(k)}, r^{(k)}) , \\
\beta_k &= \text{cublasSdot}(r^{(k+1)}, r^{(k+1)}) / \text{cublasSdot}(r^{(k)}, r^{(k)}) , \\
\gamma_k &= \text{cublasScal}(\beta_k^{(k)}), \\
p^{(k+1)} &= \text{cublasSaxpy}(1, r^{(k+1)}, p^{(k+1)}) , \quad \text{and}
\end{align*}
\]

Check the convergence for whether to stop the iteration.

In addition, matrix–matrix multiplications wherever needed are calculated using \text{cublasSgemm}.

5. Performance evaluation

As the GPU-version basis-pursuit algorithm has already been implemented, data of different sizes are adopted to test the performance of this parallel code and compared with the original CPU version.

Two computing environments are used to test the serial and parallel basis-pursuit code, respectively. The serial code is performed on the platform with an Intel Xeon Quad-Core X5647 (2.93 GHz) CPU and 3 GB dedicated memory (RAM), while the GPU-accelerated basis-pursuit algorithm is tested using a 448-core NVIDIA Tesla C2050 GPU.

The first numerical experiment uses matrices of different sizes, from 200 \( \times \) 200 to 1600 \( \times \) 1600, to test the performance of the conjugate gradient method. The maximum conjugate-gradient iteration number is set to \( 2 \times 10^5 \) to have a suitable calculation time for both the CPU- and GPU-based conjugate-gradient methods. The test shown in figure 3a indicates a significant reduction in the calculation time for the GPU-parallelised conjugate-gradient method. When increasing the matrix size, the time cost by the CPU-based conjugate-gradient method grows more rapidly compared to the GPU-based conjugate-gradient method. That is, the speedup factor is improved as the matrix size increases due to a better load balancing. The maximum speedup factor for the tested data is 92 when the matrix size is 1600 \( \times \) 1600. It can be concluded that the larger the matrix is, the more efficiently the GPU-based code operates when compared with the CPU-based code.

Moreover, the computational consumption for an individual iteration step of the basis-pursuit process is tested. The test data sets have 20 traces with the number of sample points ranging from 200 to 1600. Calculation times of the serial and parallel codes and the speedup factor are as shown in figure 3b. A similar conclusion can be drawn that the GPU-accelerated basis-pursuit process is far more efficient than the original CPU version. The speedup factor increases as the number of sample points of the input seismic traces increases.

We use a small seismic data set (figure 4a), which contains 701 traces, to further investigate the performance of the GPU-accelerated basis-pursuit algorithm. The sizes of matrices in the linear systems (17) and (21) are controlled by the number of sample points of the input data, in which 226 sample points per trace are used. As expected, the calculation time is reduced effectively due to GPU-parallelisation (Table 1).

The inversion result obtained by the GPU-parallelised code (figure 4b) has a similar precision to that of the CPU-based code (figure 4c). There is only minor difference caused by the difference in platforms (CPU and GPU). The normalised energy difference between the two inversion results is less than 0.01%.

Next, we use a large seismic data set (figure 5a) to test both the serial and parallel codes. The large field seismic data set has 1461 traces and 501 sample points. When this data
set contains more sample points (than that in the small data set), the speedup factor of the linear programming method is further improved to 147, because of an improved load balancing.

As listed in Table 1, the total computational time of the basis-pursuit inversion method is the sum of the time spent on the least-squares method for the initial estimate and the time spent on the linear programming method for the estimate updating. The total computational cost of the basis-pursuit inversion method with GPU parallelisation is speeded up 145 times. It is the GPU parallelisation that makes the basis-pursuit reflectivity inversion method pragmatically applicable.

### 6. Conclusions

In the basis-pursuit method, since the dictionary used to represent the seismic reflectivity is composed by large amounts of even and odd dipoles, matrices used in this method are huge and the matrix operations are tedious. As a result, the seismic reflectivity inversion process is very time-consuming. In this paper, we have developed a GPU-accelerated basis-pursuit algorithm using CUDA, to improve computational efficiency. Specifically, we have used GPU parallelisation for matrix multiplications, matrix-vector multiplications and for a parallelised conjugate gradient method to solve large-scale linear systems in inversion. We have presented the implementation details and evaluated the performance of the developed code using matrices and data of different sizes. Moreover, we have investigated performance using several field seismic data sets of different sizes. GPU-based parallelisation has significantly reduced the computational time by an overall factor up to 145. This efficiency improvement demonstrates the potential of the basis-pursuit inversion method in practical application to large-scale seismic reflectivity inversion problems.
Figure 5. (a) Input seismic traces of a large dataset. (b) Seismic reflectivity obtained by the CPU-based basis-pursuit algorithm. (c) Seismic reflectivity obtained by the GPU-based basis-pursuit algorithm.

Acknowledgements

The authors are grateful to the National Natural Science Foundation of China (grant nos. 41974118 and U19B6003), the sponsors of the Centre for Reservoir Geophysics, Resource Geophysics Academy, Imperial College London, and the Science Foundation of China University of Petroleum (Beijing) (grant no. 2462018BJC001), for supporting this research.

Conflict of interest statement

None declared.

References

Boyd, S. & Vandenberghe, L., 2004. Convex Optimization, 1st edn, Cambridge University Press, New York, NY.
Chen, S.S., Donoho, D.L. & Saunders, M.A., 1998. Atomic decomposition by basis pursuit, *SIAM Journal on Scientific Computing*, 20, 33–61.
Dantzig, G.B., 1982. Reminiscences about the origins of linear programming, *Operations Research Letters*, 1, 43–48.
Dantzig, G.B. & Thapa, M.N., 1997. *Linear Programming*, 1st edn, Springer, Berlin.
Gill, P.E., Murray, W. & Wright, M.H., 1991. *Numerical Linear Algebra and Optimization*, Vol. 1, 1st edn, Addison-Wesley, Redwood City, CA.
Hestenes, M.R. & Stiefel, E., 1952. Methods of conjugate gradients for solving linear systems, *Journal of Research of the National Bureau of Standards*, 49, 409–4360.
Karmarkar, N., 1984. A new polynomial-time algorithm for linear programming, *Combinatorica*, 4, 373–395.
Khachivan, L., 1979. A polynomial algorithm for linear programming, *Doklady Akademii Nauk SSSR* (The Proceedings of the USSR Academy of Sciences), 224, 1093–1096.
Li, F., Gao, J., Jiang, X. & Sun, W., 2018. A causal imaging condition for reverse time migration using the Discrete Hilbert transform and its efficient implementation on GPU, *Journal of Geophysics and Engineering*, 16, 894–912.
Liu, M. & Grana, D., 2019. Accelerating geostatistical seismic inversion using TensorFlow: a heterogeneous distributed deep learning framework, *Computers & Geosciences*, 124, 37–45.
Matoušek, J. & Gärtner, B., 2007. *Understanding and Using Linear Programming*, Springer, Berlin.
Nvidia Co., 2010. *Nvidia CUDA C Programming Guide*, version 3.2 (September 2010).
Nvidia Co., 2017. *CUBLAS Library User Guide*, version DU-06702-001_v8.0 (June 2017).
Ostoori, R., Goudarzi, A. & Oskooi, B., 2018. GPR random noise reduction using BPD and EMD, *Journal of Geophysics and Engineering*, 15, 347–353.
Potra, F.A. & Wright, S.J., 2000. Interior-point methods, *Journal of Computational and Applied Mathematics*, 124, 281–302.
Sanders, J. & Kandrot, E., 2010. *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley, Reading, MA.
Sen, M.K. & Biswas, R., 2015. Choice of regularisation weight in basis pursuit reflectivity inversion, *Journal of Geophysics and Engineering*, 12, 70–79.
Shi, Y. & Wang, Y.H., 2016. Reverse time migration of 3D vertical seismic profile data, *Geophysics*, 81, 31–38.
Wang, R. & Wang, Y.H., 2016. Multichannel algorithms for seismic reflectivity inversion, *Journal of Geophysics and Engineering*, 14, 41–50.
Wang, Y., 2003. Sparseness-constrained least-squares inversion: application to seismic wave reconstruction, *Geophysics*, 68, 1633–1638.
Wang, Y.H., 2016. *Seismic Inversion: Theory and Applications*, Wiley Blackwell, Oxford.
Weiss, R.M. & Shragge, J., 2013. Solving 3D anisotropic elastic wave equations on parallel GPU devices, *Geophysics*, 78, 7–15.
Wright, M.H., 2005. The interior-point revolution in optimization: history, recent developments, and lasting consequences, *Bulletin of the American Mathematical Society*, 42, 39–56.
Xu, S. & Liu, Y., 2018. 3D acoustic wave modelling with a time-space-domain temporal high-order finite-difference scheme, *Journal of Geophysics and Engineering*, 15, 1963–1976.
Zhang, R. & Castagna, J., 2011. Sparse-layer inversion with basis pursuit decomposition, *Geophysics*, 76, 147–158.
Zhang, R., Sen, M.K., Phan, S. & Srinivasan, S., 2012. Stochastic and deterministic seismic inversion methods for thin-bed resolution, *Journal of Geophysics and Engineering*, 9, 611–618.
Zhou, Z., Jia, X. & Qiang, X., 2018. GPU-accelerated element-free reverse-time migration with Gauss points partition, *Journal of Geophysics and Engineering*, 15, 718–728.