STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **4**, September 2016, pp 194–213.
Published online in International Academic Press (www.IAPress.org)

A Comparison of Compressed Sensing and Sparse Recovery Algorithms Applied to Simulation Data

Ya Ju Fan *, Chandrika Kamath

Lawrence Livermore National Laboratory, CA, USA

(Received: 9 March 2016; Accepted: 12 June 2016)

Abstract The move toward exascale computing for scientific simulations is placing new demands on compression techniques. It is expected that the I/O system will not be able to support the volume of data that is expected to be written out. To enable quantitative analysis and scientific discovery, we are interested in techniques that compress high-dimensional simulation data and can provide perfect or near-perfect reconstruction. In this paper, we explore the use of compressed sensing (CS) techniques to reduce the size of the data before they are written out. Using large-scale simulation data, we investigate how the sufficient sparsity condition and the contrast in the data affect the quality of reconstruction and the degree of compression. We provide suggestions for the practical implementation of CS techniques and compare them with other sparse recovery methods. Our results show that despite longer times for reconstruction, compressed sensing techniques can provide near perfect reconstruction over a range of data with varying sparsity.

Keywords Compressed Sensing, Sparse Recovery, Simulation Data, Algorithms

AMS 2010 subject classifications 94Axx, 68U20

DOI: 10.19139/soic.v4i3.207

1. Background and Motivation

The analysis of data from scientific simulations is typically done by writing out the variables of interest at each time step and analyzing them at a later time. For simulations run on massively parallel machines, the volume of data written out frequently approaches terabytes and beyond. In some problems, writing out the data could be more time consuming than the computations done at each time step. As we move to exascale computing, it is expected that the I/O subsystem will not be able to provide the needs of the simulations [8]. Instead of writing out the data for analysis, it has been proposed that the analysis algorithms be moved *in situ*, and only the analysis results, which are hopefully much smaller, be written out. This is a viable option that works when we know the analysis algorithm we want to use and its parameters. However, in many cases, the choice of analysis algorithms and their parameters may depend on what we find as we analyze the simulation output. This is especially true when the motivation for the analysis is scientific understanding and discovery. In such cases, the analysis cannot be done *in situ* and alternate approaches have to be considered to address this problem of limited I/O bandwidth.

One such approach is to reduce the size of the simulation output by compressing the data before they are written out. There are several ways of compressing data, ranging from the traditional compression schemes, such as JPEG, that are used for images, to others that are based on clustering techniques from data mining. For the data set and problem considered in this paper, we require loss-less compression techniques that can be applied to data that are irregularly spaced in two or three dimensions and are distributed across multiple processors (or files). We propose

*Correspondence to: Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94551, USA.
E-mail:fan4,kamath2@llnl.gov

ISSN 2310-5070 (online) ISSN 2311-004X (print)

addressing this problem of compressing simulation output using compressed sensing (CS) and sparse recovery algorithms. We have three goals. First, we want to understand how we would apply compressed sensing techniques in practice to simulation data, given the irregular spacing of points and their distribution across multiple files. Second, we are interested in how well these techniques preserve the features of interest in the data so that we can perform our analysis on data that were compressed and reconstructed, while obtaining results identical or close to what we would have obtained using the original, uncompressed data. Finally, we want to compare compressed sensing with other sparse recovery algorithms and evaluate their suitability for our data sets using the metrics of reconstruction accuracy, compression time, and reconstruction time.

We note that our interest in compression is for the purpose of reducing the amount of data generated for quantitative analysis, not for visualization. In other words, the fact that the original and the compressed, then reconstructed, data are visually the same is not sufficient; we also want the quantitative results of the analysis to be the same for the two data sets. Therefore, we need perfect or near-perfect reconstruction of the compressed data.

This paper is organized as follows: First, in Section 2, we discuss compression techniques currently in use to reduce the size of scientific data sets, especially those from simulations. Next, we describe the specific characteristics of our data set in Section 3 to provide the background on the constraints that influence our choice of compression algorithm. We describe the algorithms considered in Section 4 and the pre-processing of the data in Section 5, followed by the experimental results in Section 6. We first identify a performance metric to use to evaluate the effectiveness of compression, followed by an evaluation of various preprocessing steps and the use of sparse vs. dense measurement matrices. We then compare compressed sensing with other sparse recovery algorithms and identify which algorithms best meet the needs of our problem. We conclude with a summary in Section 7.

2. Related work

Data compression techniques, such as JPEG and MPEG [28], have long been used to reduce the size of images and video to make it easier to store and transmit them. These techniques are lossy, that is, the data compressed using these techniques cannot be reconstructed exactly. However, if the compression preserves key aspects of the images such that the image or video after reconstruction is visually the same as the original, such compression may be acceptable in some applications while, at the same time, enabling a large reduction in the volume of the data.

However, for scientific data, visual quality is often not the only factor influencing the choice of a compression technique. For example, the wavelet-based Hcompress software [16] used in compressing astronomy images has both a lossy and a loss-less capability; the former is used to generate pre-view images, where fine details may be irrelevant, and the latter is used in scientific analysis where such details are important. Similarly, for medical images, loss-less compression is used to preserve the information in the data, though a hybrid approach that uses lossy compression in regions that are not of interest has been proposed [15] to reduce the size.

Our interest is in scientific data generated not as images, but as a result of scientific simulations run on high-performance computers. These simulations typically generate multi-terabytes of data in the form of double-precision variable values that are specified at grid or mesh points in a two- or three-dimensional spatial domain. The grid could be regular or irregular and could be either fixed or change over time. Since the volume of these data are large, the visualization community has developed several compression schemes to reduce the costs of data transfer and storage. Many of these schemes are lossy, and range from simple approaches, such as storing a double precision number in single precision, to more complex techniques involving sophisticated encoding [14, 24]. These lossy techniques are often sufficient to meet the end goal of qualitative analysis through visual display of the data, especially as the data sizes are much larger than can be accommodated on the displays used to visualize the data. However, to understand the effects of loss of information, efforts have been made to evaluate how various physically-motivated metrics change as a result of compression [22]. A different approach to lossy compression of simulation data borrows ideas from graph-based clustering techniques. It represents the mesh data as a graph, and identifies vertex-sets that can be approximated by a constant value within a user-specified error tolerance [20].

Compressed sensing techniques, the subject of this paper, can also be implemented in a lossy manner by including quantization in the process [29, 26], or by combining it with wavelets [27].

As mentioned earlier, our interest is in loss-less compression of simulation data using compressed sensing and sparse recovery methods. There has been relatively little work in this area. The visualization community has investigated, albeit to a limited extent, the loss-less compression of the meshes used to represent three-dimensional geometric models [19] as well as the floating-point data themselves [25]. The work in this paper is an extension of our early work that was performed to understand if compressed sensing could be applied to unstructured mesh data [12].

3. Description of the Data

We conduct our investigation into compressed sensing and sparse recovery algorithms using data from a fusion simulation [23]. These data describe how the plasma within a tokamak evolves over time. The computational domain is a three-dimensional toroid, representing the tokamak. It is composed of 32 poloidal planes, equi-spaced along the toroid (see schematic in Figure 1(a)). Each poloidal plane has 591,745 grid points; these are arranged in concentric circles around a center that represents the magnetic axis that runs around the toroid (see Figure 1(c)). To correctly simulate the physical processes in fusion without unnecessarily increasing the number of grid points, and hence the computational cost of the simulation, the distances between poloidal planes is larger than the distances between grid points within a poloidal plane. This means that the physics of interest occurs within each poloidal plane and we can analyze each plane independently. The grid points are not regularly spaced, so their spatial coordinates are required for analysis. This also means that compression techniques that assume the data to be available in a two-dimensional regular grid, as in an image, are not applicable to our data set. To reduce the size of the simulation output, the data are written out every five time steps.

Since the computational domain is so large, with nearly 19,200,000 grid points across the 32 poloidal planes, the simulation is run on massively parallel machines. The points in each poloidal plane are sub-divided into smaller domains, as shown in Figure 1(b), with roughly equal number of points in each, and each domain is assigned to one processor. As we shall see later in this section, this distribution of points across multiple processors, causes an additional complication in applying the compression algorithms to our data.

At each time step in the simulation, there are several variables that are written out for each grid point; each variable represents a different physical quantity. In our work, we consider just one of these variables; compressing all the data at a time step can be done by processing each variable separately. The grid points in our problem remain fixed over time, so their coordinates, in three-dimensions, are written out only once.

In our evaluation of compressed sensing and sparse recovery algorithms, we want to ensure that the analysis results obtained using the re-constructed compressed data are very close to the results using the original, uncompressed data. Specifically, we focus on the analysis of coherent structures in the data. Coherent structures are essentially groups of grid points that behave as a coherent whole. The grid points are spatially near each other, have similar values of a variable, and evolve in a similar way over time. Identifying coherent structures and characterizing their behavior over time is a common analysis task in many simulations, so our work is broadly applicable in many domains.

Figure 1(d) shows the values of the variable of interest at the grid points in plane 00 at time step 500. Since each plane has a large number of grid points (nearly 600,000), we progressively zoom-in into the details of this plane in Figure 2, where panel (a) shows a pie-shaped wedge, panel (b) shows the distinct grid points illustrating the coherent structures, and panel (c) shows a region with no structures. The coherent structures can be seen in panel (b) as the groups of grid points with values of the variable higher than the surrounding grid points, where the values are nearly zero. At this time step, there are very few coherent structures on the left half of the poloidal plane and near the center (which is the location of the magnetic axis of the tokamak). As a result, when the plane is sub-divided into domains as shown in Figure 1(b), there are some domains that have a large number of coherent structures, while others have none. Consequently, each domain has a different range of values of a variable, with

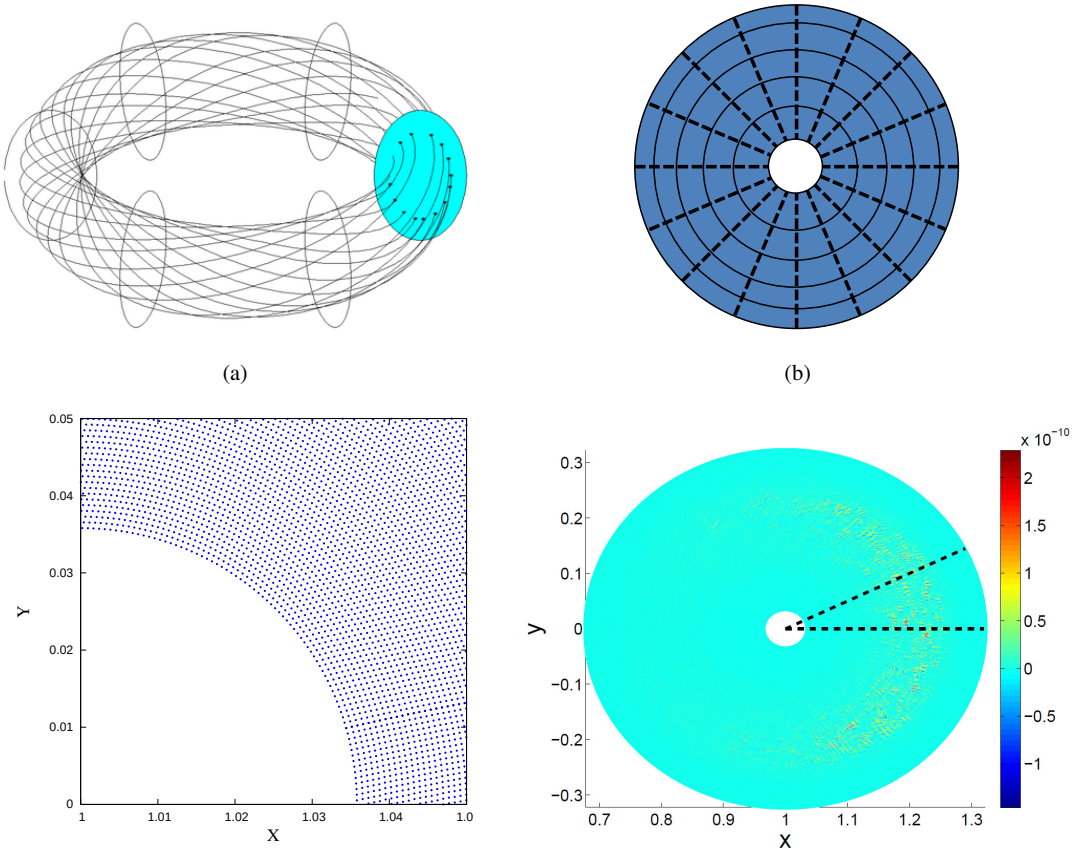


Figure 1. (a) A schematic showing a three-dimensional toroid representing the tokamak; the plane highlighted in blue is a poloidal plane. (b) Black lines show the division of a poloidal plane into regions with roughly equal areas; each region is assigned to a different processor. The center of the circle is the magnetic axis of the toroid. (c) A zoomed-in view showing the locations of the points near the center of the poloidal plane. (d) An example of the data in a poloidal plane at time step 500. The dotted lines indicate a slice of the data that contains $\frac{1}{16}$ of the grid points in this plane.

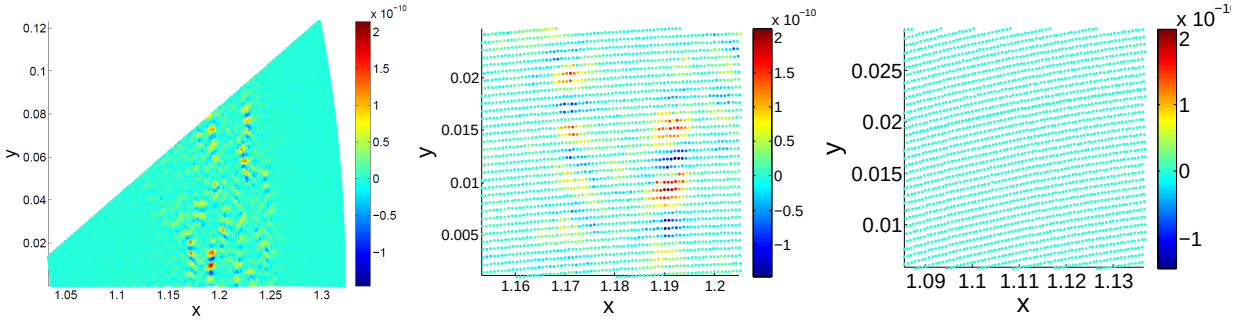


Figure 2. (a) A segment of the data at time step 500 showing the variable of interest. Zoomed-in views showing (b) the patterns of interest and the grid points and (c) the background region with no structures.

domains with many coherent structures having a larger variation. Given this variation in the data, we expect that determining CS parameters, such as the size of the compressed data, will be a challenge.

3.1. Dividing the data into sub-domains

To mimic the decomposition of each poloidal plane among processors, we divided the data on a poloidal plane into parts, or sub-domains, of equal area so that each sub-domain has almost the same number of data points, n. Since the data are in the form of a circular disk, we use the radius r and angle θ for the split. The range of the angle θ is $[-\pi, \pi]$. We make 16 cuts into the circle from the center out. Each part has the same angle $\theta = \frac{\pi}{8}$ in radians. We then divide each part into five pieces of equal area. Let r_0 and r be the closest and farthest distance from the data to the center of the circle, respectively. We compute the radii, $r_1, r_2, \ldots, r_4$, that equally separate the part into five based on r_0 and r. That is $r_i = \sqrt{\frac{1}{5}(i \times r^2 + (5-i) \times r_0^2)}$. Figure 1(b) shows how the data are divided into 80 sub-domains that are almost equal in size.

In practice, the number of sub-domains is determined by the size of the problem and the number of processors used to run the simulation. However, the division into sub-domains shown in Figure 1(b) is typical as it minimizes the communication costs associated with sharing data across sub-domains in neighboring processors.

4. Description of the compression algorithms

We next describe the compressed sensing and sparse reconstruction algorithms used in our work. The ideas behind compressed sensing have been studied extensively [11]. In essence, the method begins by sampling a sparse signal at a low rate and then uses computational techniques to obtain its exact reconstruction. This makes compressing data a simple task and most of the effort is spent on reconstructing the original signal.

Given a signal $\mathbf{u}$, in the form of an n-dimensional vector, we want to compress the signal to a smaller length $m(< n)$ so that we can reduce the cost of moving or transmitting this signal, which in our case is the data from the simulation, from the computer to the disk. We make m measurements, each amounting to a projection of $\mathbf{u}$ on a known vector. These measurements with a linear transformation result in a vector $\mathbf{f}$ of length m, that is, $\mathbf{f} = \mathbf{A}\mathbf{u}$, where the known matrix $\mathbf{A}$ is of size $m \times n$. The vector $\mathbf{f}$ is referred to as a sketch of $\mathbf{u}$ in theoretical computer science [17].

Once the m values of the vector $\mathbf{f}$, which is smaller than the original signal $\mathbf{u}$, have been transmitted, we need to uncompress the vector and recover the original signal. In general, this is a difficult task. Compressed sensing works only when signal $\mathbf{u}$ is sufficiently sparse, with k number of non-zero elements, where $k \ll n$ [7, 5]. The matrix $\mathbf{A}$ contains randomly generated numbers, which, in the conventional CS approach, are drawn independently from a Gaussian distribution. From a computational viewpoint, we do not need to explicitly store the matrix $\mathbf{A}$. Instead, we can store the random seed used to create the matrix and generate the matrix entries as needed, saving on memory requirements for the algorithm. In our work, we focus on the case where the non-zero components are real numbers and the measurements are linearly independent, that is, A is a full-rank matrix. The CS reconstruction problem then reduces to finding a $\mathbf{u}$ that satisfies the linear system $\mathbf{f} = \mathbf{A}\mathbf{u}$.

To make exact signal reconstruction possible in principle, we need at least $m \geq k + 1$. We can conduct an exhaustive search on all $\binom{n}{k}$ possible selections of searches on non-zero components of $\mathbf{u}$ to find the only one possible linear system that can be inverted. However, when our data set is very large, with a large n, using the enumeration method quickly becomes impractical.

In practice, the reconstruction step in CS is solved by forming an L_1-norm optimization problem [6, 11] defined as

$$\min_{\hat{\mathbf{u}}} \|\hat{\mathbf{u}}\|_1 \quad \text{subject to} \quad \mathbf{A}\hat{\mathbf{u}} = \mathbf{f} \tag{1}$$

where $\|\hat{\mathbf{u}}\|_1 = \sum_{i=1}^{n} |\hat{\mathbf{u}}_i|$. Equation (1) is also known as the basis pursuit problem, whose solution is the vector with smallest L_1 norm that describes the sketch $\mathbf{f}$.

CS is an example of a sparse recovery algorithm that approximates a sparse signal $\mathbf{u}$ from its lower-dimensional sketch $\mathbf{A}\mathbf{u} \in \Re^m$. However, unlike the conventional CS approach, sparse recovery algorithms do not require a solution of the optimization problem in Equation (1), nor do they require the matrix $\mathbf{A}$ to be dense or have entries drawn from a Gaussian distribution.

In this paper, we compare the performance of CS and sparse recovery algorithms for the compression of the data set described in Section 3. We consider six different methods: three of them - L_1 magic [4], Gradient Projection for Sparse Reconstruction (GPSR) [13] and YALL1 [10] - are conventional CS methods where the matrix $\mathbf{A}$ is dense, with values drawn from a Gaussian distribution. The other three methods - Sparse Matching Pursuit (SMP) [2], Sequential Sparse Matching Pursuit (SSMP) [1], and Count-Min sketch [9] - have a sparse measurement matrix $\mathbf{A}$. Numerous methods have been proposed for sparse recovery by different research communities, including signal processing, theoretical computer science, and applied mathematics. These methods vary in the accuracy of reconstruction, computational time, and the length of the sketch. The six methods we selected are illustrative of different approaches. We briefly describe them in the following; more details are available in the references.

- **Compressed sensing using L_1 magic:** L_1 magic [4] is a public domain solver that obtains solutions of Equation (1) using efficient linear programming techniques. In our work, we use the routine `l1eq_pd`, which is a primal-dual algorithm for linear programming. A system of nonlinear equations is formulated using the Karush-Kuhn-Tucker conditions corresponding to Equation (1). The system is linearized and solved using the classical Newton's method. This algorithm starts at an interior point and applies the maximum step size that just keeps the point in the interior. The algorithm iterates and identifies the solution point as close to being optimal when the slackness condition is small and the surrogate duality gap has decreased below a given tolerance. In the conventional CS method, the measurement matrix $\mathbf{A}$ is dense.
- **Compressed sensing using Gradient Projection for Sparse Reconstruction (GPSR):** Gradient projection for sparse reconstruction (GPSR) [13] solves the general optimization problem formulated as

$$\min_{\hat{\mathbf{u}}} \frac{1}{2}\|\mathbf{f} - \mathbf{A}\hat{\mathbf{u}}\|_{\mathbf{2}}^{\mathbf{2}} + \tau\|\hat{\mathbf{u}}\|_{\mathbf{1}}, \tag{2}$$

  which is a convex unconstrained optimization problem with a quadratic term. τ is a nonnegative parameter and $\|\mathbf{v}\|_{\mathbf{2}}$ denotes the Euclidean norm of a vector $\mathbf{v}$. Equation (2) falls in the category of a sparse recovery problem since the use of the L_1 norm on $\hat{\mathbf{u}}$ tends to reconstruct the signal $\mathbf{u}$ to be as sparse as possible, given the sketch $\mathbf{f}$ and the measurement matrix $\mathbf{A}$. GPSR is a gradient projection algorithm in which the search path from each iterate is obtained by projecting the negative-gradient onto the feasible set from each iteration. GPSR has been shown to perform well in a range of applications and is significantly faster than competing methods [13]. The measurement matrix $\mathbf{A}$ is typically dense.
- **Group sparse reconstruction using alternating direction method (YALL1):** YALL1 (Your ALgorithm for L1) [10] is a solver for group sparse reconstruction using alternating direction method (ADM) [31, 3]. The algorithm makes use of the group sparsity structure in the data. Components within a group are likely to be either all zeros or all nonzeros. Formulating the group sparsity reduces the degrees of freedom in the solution, and hence improves reconstruction performance. Let groups $\{\mathbf{u}_{g_i} \in \Re^{n_i} : i = 1, \ldots, s\}$ be a partition of $\mathbf{u}$, where $g_i \subseteq \{1, 2, \ldots, n\}$ is a set of instance indices contained in the i-th group, and $\mathbf{u}_{g_i}$ are the n_i instances of $\mathbf{u}$ that are in the i-th group. YALL1 solves the following basis pursuit model:

$$\min_{\hat{\mathbf{u}}} \|\hat{\mathbf{u}}\|_{w,2,1} \quad \text{subject to} \quad \mathbf{A}\hat{\mathbf{u}} = \mathbf{f} \tag{3}$$

  where $\|\hat{\mathbf{u}}\|_{w,2,1} := \sum_{i=1}^{s} w_i \|\hat{\mathbf{u}}_{g_i}\|_2$ is the weighted $l_{2,1}$-norm and $w_i \geq 0$ for $i = 1, \ldots, s$ are weights associated with each group. YALL1 applies the classic ADM to both the primal and dual forms of the $l_{w,2,1}$-problem (3) whose objective functions are separable, and constructs an efficient first-order algorithm using augmented Lagrangian that updates both the primal and dual variables at each iteration.
- **Sparse matching pursuit (SMP):** Sparse matching pursuit (SMP) is a method designed for practical, near-optimal, sparse recovery in the L_1 norm [2]. Instead of using a dense matrix generated from a Gaussian distribution in the L_1 norm, as in Equation (1), SMP uses a sparse measurement matrix $\mathbf{A}$ with binary elements, that is, the matrix elements take the values 0 or 1. It is derived from a very fast recovery algorithm, called Expander Matching Pursuit (EMP) [18], which, in turn, is derived from the Orthogonal Matching Pursuit (OMP) [30]. Like these methods, SMP is a greedy iterative algorithm that estimates the vector $\mathbf{y} = (\mathbf{u}^t - \mathbf{u})$, which is the difference between the current approximation $\mathbf{u}^t$ and the signal $\mathbf{u}$ from the sketch

$\mathbf{Ay} = \mathbf{Au}^t - \mathbf{Au}$. The matrix $\mathbf{A}$ can be interpreted as an adjacency matrix of a bipartite graph G, with left vertex set = $\{1, 2, \ldots, n\}$ and right vertex set = $\{1, 2, \ldots, m\}$. Each column of $\mathbf{A}$ contains d elements that are one, that is, every vertex in the left part of the graph has exactly d neighbors in the right part. Each coordinate $\mathbf{y}_i^*$ in the estimation $\mathbf{y}^*$ of $\mathbf{y}$ is the median of the coordinates of the sketch vector $\mathbf{Ay}$ that are neighbors of i in the graph G. The approximated signal $\mathbf{u}^t$ is updated by $\mathbf{y}^*$ and the process is repeated. The idea behind the algorithm is to select the neighbors of i using a voting-like mechanism. Compared to EMP, SMP is slightly slower, but performs successful recovery from a significantly smaller number of measurements, and is therefore more suitable for our application in reducing the cost of data movement.

- **Sequential sparse matching pursuit (SSMP):** Sequential sparse matching pursuit (SSMP) [1] is derived from the SMP method. Both SSMP and SMP use a sparse binary matrix for measurement, and voting-like mechanism in iterative search for signal approximation. The difference is that SSMP updates the approximation signals sequentially while SMP updates them in parallel. SMP works only when the input parameters, the sparsity k and the number of measurements m, are within the theoretically guaranteed region. SSMP does not have such convergence problems and is designed for arbitrary input signals.
- **Count-Min and Count-Median sketch:** The Count-Min sketch algorithm [9] also utilizes a sparse random matrix $\mathbf{A}$ with binary elements. It only works under the assumption that all elements in the signal are greater or equal to zero. That is, the elements of $\mathbf{u}$, $\mathbf{u}_i \geq 0$ for all i. The method divides matrix $\mathbf{A}$ into d segments, $\mathbf{A}(h_1), \mathbf{A}(h_2), \ldots, \mathbf{A}(h_d)$. Each segment of $\mathbf{A}$ corresponds to a function h, from the set H of all functions $h : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, w\}$. We independently and uniformly choose these d random functions. Each such function generates a binary matrix $\mathbf{A}(h)$ of size $w \times n$ by setting $(\mathbf{A}(h))_{j,i} = 1$ if $j = h(i)$ and $(\mathbf{A}(h))_{j,i} = 0$ otherwise. The one-to-one mapping makes each column have exactly one 1. Combining the d segments, the algorithm constructs the matrix $\mathbf{A}$ whose number of rows is equal to $m = wd$. To approximate $\mathbf{u}^*$ from $\mathbf{Au}$, the algorithm makes use of the construction of matrix $\mathbf{A}$. Observe that for any signal $\mathbf{u}$,

$$(\mathbf{Au})_{(l-1)w+j} = (\mathbf{A}(h_l)\mathbf{u})_j = \sum_{i:h_l(i)=j} \mathbf{u}_i$$

for $l = 1, 2, \ldots, d$ and $j = 1, 2, \ldots, w$. Using this property, computing the approximation of signal $\mathbf{u}$ is fairly simple. It requires only pairwise independent hash functions. Under the assumption of $\mathbf{u} \geq \mathbf{0}$, we define elements of the approximation of $\mathbf{u}^*$ to be:

$$\mathbf{u}_i^* = \min_l \quad (\mathbf{A}(h_l)\mathbf{u})_{h_l(i)} = \min_l \quad \sum_{i':h_l(i')=h_l(i)} \mathbf{u}_{i'}.$$

We cannot directly apply Count-Min to our data sets as they contain negative values. Instead we use the extended Count-Min algorithm, which is also called the Count-Median algorithm, for our problem. This method replaces $\mathbf{u}_i^*$ by

$$(\mathbf{u}_{med}^*)_i = \operatorname*{median}_l \quad (\mathbf{A}(h_l)\mathbf{u})_{h_l(i)}.$$

Removing minimization constraints allows negative estimators. The algorithm then works for general signals with both positive and negative values.

Count-Min and Count-Median sketches are algorithms designed by the database community to handle fast queries on data streams [9]. Therefore, their recovery time is expected to be very fast, but as only a summary of the data is stored, the algorithms are not intended for use in exact recovery. However, we have included them in our study to evaluate how much the accuracy degrades in the context of our problem.

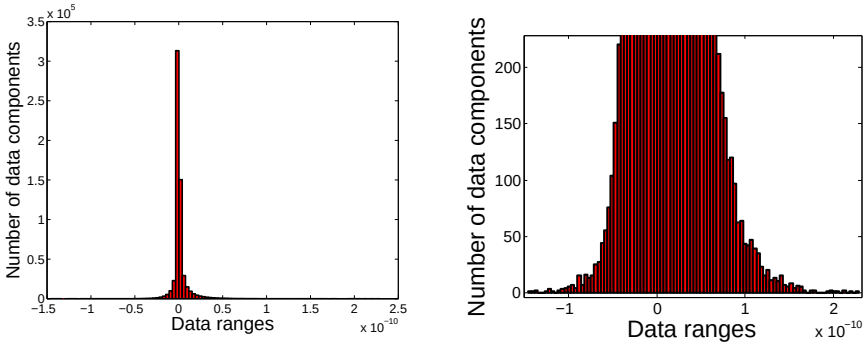
5. Preprocessing the data

Before we could apply the six compression algorithms to our simulation data, we found that the data required some preprocessing. Our data set contains a large number of values that are almost zero as seen in the background region

in Figure 2. When we first applied CS to the data, we were not successful in recovering the original signal even though the algorithm had converged (see Figure 5). On examining the results, we found that a majority of the data components have to be exactly zero in order to satisfy the sufficient sparsity condition of CS. This suggests that we need to threshold the data first and set all values below the threshold to zero. From the analysis viewpoint, this is acceptable because one of the first steps in the analysis of coherent structures in the data is to perform such a thresholding as it is the grid points with the larger values that are of interest [21].

The use of thresholding as a pre-processing step is also motivated by a constraint on k, the number of non-zeros, and m, the number of measurements, that must be satisfied for reconstructing the signal. As the results by Candes et al. [5] show, if $k \leq \alpha \cdot m \cdot (\log n)^{-1}$, where $\alpha > 0$ is a constant, then the L_1 problem (1) will find the unique solution that is exactly equal to $\mathbf{u}$ with very high probability. Their empirical study showed that for n between a few hundred and a few thousand, with $k \leq m/8$, the recovery rate is above 90%. These results also suggest that $m^* = k \log n$ is an appropriate choice for the number of rows of $\mathbf{A}$. However, there is still slight chance that we will not get the exact $\mathbf{u}$ from solving the L_1 problem. Since we are looking for exact reconstruction of the patterns in the data, in addition to thresholding values that are very near zero, we need to investigate values around m^* and find a better condition for CS for our simulation data.

Another issue we need to address is the contrast in our data set, that is the range of intensity values at the grid points in the simulations. The data over an entire poloidal plane at early time steps have low contrast, with a relatively small range of intensities. Not only are the majority of the data values almost zero, forming the background, but the rest of the data values, which are part of the coherent structures of interest, are also close to zero. For example, at time step 500, Figure 1 indicates that the absolute values in the poloidal plane are less than 3×10^{-10}. The corresponding histogram of values shown in Figure 3 indicates that most of the values are near zero (the background), with a few grid points (the coherent structures) having slightly larger values, as shown in Figure 3(b). At later time steps in the simulation, the absolute values of the variable in a coherent structure within a poloidal plane can go as high as 10^{-5}, while the background remains at 10^{-10} or smaller. Further, some sub-domains, such as the ones near the center or on the left side of the poloidal plane, continue to have a small range of values as they are composed of mainly the background. The results at later time steps of the simulation allow us to compare the methods for higher dynamic range data with varying sparsity.



(a) (b)

Figure 3. (a) Histogram of the data at time step 500 showing majority of the data values very close to zero. (b) Zoomed-in view of the histogram.

To address this problem of sub-domains in a poloidal plane having a large range of values either within a sub-domain or across sub-domains, we consider the option of scaling the data within a poloidal plane. Scaling just the values in each sub-domain to [0.0,1.0] will result in amplifying the noise in the sub-domains that have all grid points with values very near zero. To scale across the entire poloidal plane, we first use a gather/scatter operation to find the minimum and maximum values across all sub-domains that form a poloidal plane. Then, each sub-domain scales its grid points so that the values in the entire poloidal plane are in the [0.0, 1.0] range. After reconstruction, we only need the two limits to scale the data back to their original values without any loss of information.

Increasing the contrast of the data helps the performance of CS and the sparse recovery algorithms in many ways. First, it improves our ability to control the convergence of these methods. Recall that L_1 magic, GPSR

and YALL1 are iterative optimization algorithms and all have convergence conditions with a given tolerance. In practice, the tolerance is set to be some very small number. However, when the contrast or the values of the data are relatively small, the algorithm could terminate prematurely due to the convergence conditions being satisfied, even though the optimal solution has not been obtained. While the user can adjust the tolerance for the algorithm to terminate at an acceptable optimal, it is difficult to determine such a value when the data set contains very small numbers and has a small range of intensity values. Therefore, scaling the data to [0.0,1.0] helps standardize the input data for convergence conditions. This scaling, by improving the contrast in the data values, also helps the sparse recovery algorithms that are iterative in nature. Finally, when multiple algorithms are being compared, as in our study, scaling the data helps us to evaluate the performance of the algorithms more objectively.

6. Experimental Results

We next describe the experiments we conducted to understand how best to apply compressed sensing and sparse recovery algorithms to our data set. Given a data set of n components containing k non-zero entries, we have two main goals: The first is to determine if CS can be applied successfully to our data and if our ideas for scaling the data and selecting the size of the compressed data result in sufficient compression and near-perfect reconstruction. In particular, we want to understand how well the features of interest in our data are preserved during reconstruction. Second, we want to compare CS with other compression methods described in Section 4 in terms of reconstruction accuracy, compression time and reconstruction time to identify those suitable for our problem.

We conduct our experiments in two parts. Our main focus is compressed sensing as it allows perfect reconstruction of the data. Therefore, in the first set of experiments we evaluate different performance metrics and pre-processing ideas with CS using L_1 magic as the compression algorithm. In the second set of experiments, we compare CS with the sparse recovery algorithms. Since the SMP, SSMP and Count-Min methods require a sparse measurement matrix, for a fair comparison, we consider both a dense matrix and a sparse matrix in L_1 magic, GPSR and YALL1. Recall that these three methods were originally proposed with a dense measurement matrix.

6.1. Reconstruction accuracy of compressed sensing

The first evaluated how well CS could reconstruct compressed data using the L_1 magic software. Recall that we divided the data on each poloidal plane into equal sub-domains to mimic the way in which the data are decomposed into domains for parallel processing. As observed earlier, some of these domains have a large number of coherent structures, while others are mainly background, with relatively small values that are likely to be ignored during thresholding. So, the percentage of non-zeros, k/n, in each domain is different. To evaluate the performance of CS on each sub-domain of the data, we vary the percentage of compression, m/n for each sub-domain, and evaluate a metric that determines the accuracy of reconstruction. The ratio k/n gives the percentage of non-zeros in each sub-domain, while the ratio m/n gives the reduction in size of the data for that sub-domain.

We considered three metrics to measure the accuracy of reconstruction: the L_2 norm, the peak signal-to-noise ratio (PSNR), and R^2. The L_2 norm measures the Euclidean distance between the original data, $\mathbf{u}_i, i = 1, \ldots, n$, and the data, $\hat{\mathbf{u}}_i, i = 1, \ldots, n$, after CS and reconstruction. PSNR is derived from the mean squared error (MSE) of the two data sets; it is often used in image compression where removing noise is the desired outcome. Let

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}_i - \hat{\mathbf{u}}_i)^2.$$

Then, PSNR is defined as:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\max(\{\mathbf{u}_i\}_{i=1,\ldots,n})}{\text{MSE}}\right).$$

The coefficient of determination, denoted by R^2, is widely used in statistics in comparing two data sets and is defined as:

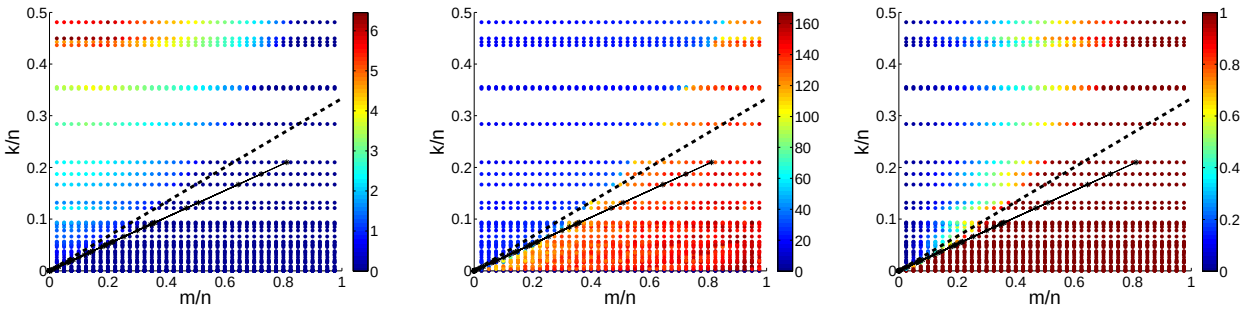
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\mathbf{u}_i - \hat{\mathbf{u}}_i)^2}{\sum_{i=1}^{n}(\mathbf{u}_i - \frac{\sum_{i=1}^{n}\mathbf{u}_i}{n})^2}.$$

Since we are looking for exact reconstruction, we prefer smaller L_2, larger PSNR and an R^2 value that is close to 1.

6.1.1. Evaluating performance metrics We compared the three performance metrics, the L_2 norm, PSNR, and R^2, by evaluating their effectiveness using the data in all sub-domains from time step 500 in the simulation. First, we scaled the data in each sub-domain using the maximum and the minimum values from the 00 poloidal plane that spans all sub-domains. Then, based on the histogram of the data, we set components whose absolute values are less than 0.3 to zero as a threshold that separates the coherent structures of interest from the background.

Table 1. Data size n and number of non-zero components k for the 11 sub-domains (out of a total of 80) that are non-sparse at time step 500. CS fails to reconstruct these sub-domains using $m < n$.

Pie #	Slice #	n	k
1	2	7454	4171
1	3	7508	4553
2	2	7381	3866
2	3	7448	4914
3	3	7458	4505
4	3	7459	3587
13	3	7468	3933
14	3	7458	4898
15	2	7380	3756
15	3	7459	4435
16	3	7451	4752



(a) L_2 (b) PSNR (c) R^2

Figure 4. Performance of L_1 magic with a sparse measurement matrix on data from time step 500. The values of the three metrics - L_2, PSNR and R^2 - are shown in color for different levels of compression (m/n) for sub-domains with different levels of sparsity (k/n) . The dotted black line represents the slope of $\frac{1}{3}$. The straight line with star markers is the theoretically suggested selection of m^* with respect to k.

When the data in a sub-domain are not sufficiently sparse, CS performs poorly. Table 1 shows the numbers of non-zero components and the sizes of the data for the 11 sub-domains that are not sparse enough and are excluded in our work. The results indicate that in general when k is more than 50% of n, CS fails in reconstruction using $m < n$.

Figure 4 shows the performance of L_1 magic with a sparse measurement matrix on each sub-domain at time step 500 using the three different performance metrics. This measurement matrix was generated as described in the Count-Min sketch algorithm and in Section 6.1.4. There are 80 sub-domains on each poloidal plane (as shown in

Figure 1(c)), each with roughly the same number of grid points, but a different number, k, of non-zero values. We ignore the low sparsity sub-domains listed in Table 1. Before applying CS, the data were first scaled so the poloidal plane had values between [0.0, 1.0], then thresholded to identify the background points, which were set to zero. The dotted black line starting from origin represents the slope of $\frac{1}{3}$. Points below that line could be compressed better by writing out just the x–y coordinates and values of the grid points that remain after the thresholding. The solid straight line with star markers is the theoretically suggested selection of m^* with respect to k.

Each horizontal line in the figure represents the performance of CS on one sub-domain. Taking Figure 4(c) as an example, as we vary m, the number of measurements, for a sub-domain that has k non-zero values, the performance of CS evaluated using R^2 is represented by colors on a horizontal dotted line. The change in color clearly shows that the R^2 increases as the size of compressed data (m/n) increases. Recall that we prefer smaller L_2, larger PSNR and an R^2 value that is close to 1.

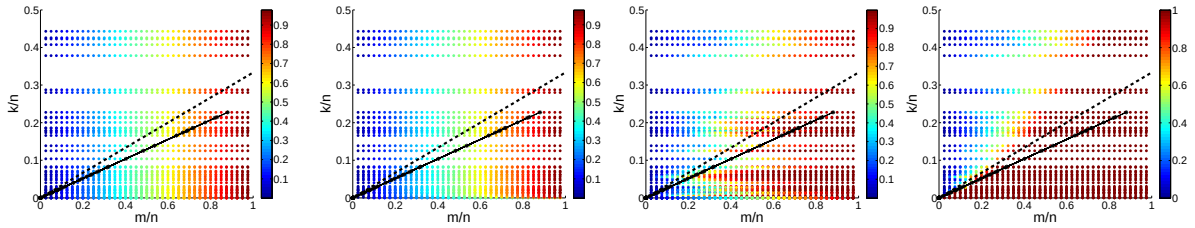
Among the three metrics, R^2 gives the most gradual and distinct difference in the performance as we vary m. Regardless of the data used, R^2 is always in the range $[0, 1]$, unlike the L_2 norm and PSNR metrics, whose maximum will vary with the data, making it difficult to compare compression results across data from different time steps or poloidal planes. A value of $R^2 = 1$ obtained using data before and after compression indicates that they are identical. Therefore, we propose using R^2 as our metric of choice in measuring CS for exact reconstruction. We will only show R^2 values in the rest of this paper.

6.1.2. Evaluating the effectiveness of compression In Figure 4, we include two lines to help evaluate the effectiveness of the compression using CS. The thick dotted black line starting from the origin has a slope of $\frac{1}{3}$. Points on the line represent $m = 3k$, indicating that carrying the three values - x–y coordinates of a grid point and its value - for all non-zero points that remain after thresholding requires the same size of storage as the compressed data. Points below the line represent $m > 3k$, where the compressed data have a larger size than the size of k non-zero values and their corresponding x–y locations. Therefore, the sub-domains that benefit from CS are the ones that have $R^2 = 1$ and lie above the dotted line.

The straight line with markers connects the values when $m = m^*$. As mentioned earlier, theory suggests a value of $m^* = k \log n$ for reconstruction using the L_1 norm. To test how accurate this number m^* works in practice, especially in the context of the R^2 metric, we considered several different values of m around m^* for various k. Most points below the line with markers have $R^2 = 1$ and only a few points with small k below the line do not. This indicates that the theoretical selection of m^* works very well when k is not too small, i.e. less than 5% of n in Figure 4(c). Also, we do not need as large an m^* for exact reconstruction when k is large, especially above 10% of n as indicated by the $R^2 = 1$ points that lie to the left of the line with markers.

6.1.3. Evaluating the performance of preprocessing Figure 5 shows the results of L_1 magic with a sparse measurement matrix on data from time step 1000 in the simulation with different preprocessing options. We considered a later time step as by this time the simulation has progressed and there are more coherent structures, leading to data that allows a better evaluation of the preprocessing options. Recall that we need exact reconstruction, so a value of $R^2 = 1$ is desired.

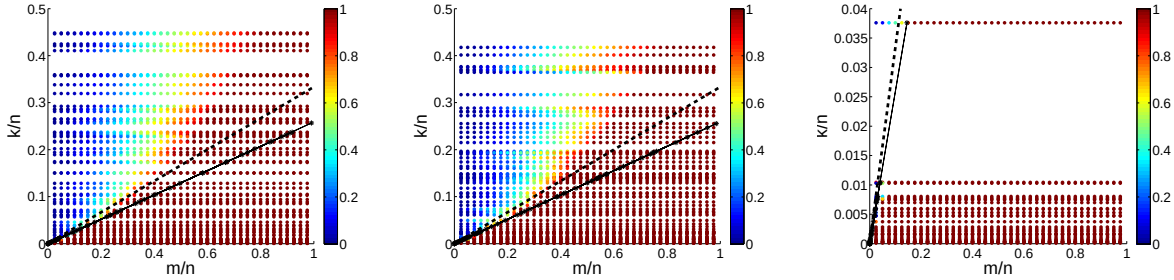
Figure 5(a) shows the results with no preprocessing, followed by the results with only thresholding, only scaling, and both thresholding and scaling, in Figures 5(b)-(d), respectively. To make it easier to compare the plots, each horizontal line in Figures 5(a) and (c) corresponds to the same sub-domain as the corresponding line in (d); however, the number of non-zero components is higher when no thresholding is used. We make several observations. The results with no preprocessing and only thresholding appear very similar. Neither has any sub-domains with perfect reconstruction. The maximum R^2 from the data without any preprocessing is 0.9982, while the maximum R^2 from only thresholding is 0.9977. The use of both scaling and thresholding results the most sub-domains with $R^2 = 1$, while using just scaling, but no thresholding, results in a few sub-domains with $R^2 = 1$. However, all these sub-domains in Figure 5(c) are below the black dotted line, the one at $\frac{1}{3}$ slope, indicating that compression is not preferred. From these observations, we conclude that scaling has very significant effect on the performance of CS, and thresholding improves the performance only when the data are scaled.



(a) Without preprocessing (b) Thresholding only (c) Scaling only (d) Scaling and thresholding

Figure 5. Performance of L_1 magic with a sparse measurement matrix on data at time step 1000 with different preprocessing options. (a) Results with no scaling or thresholding of the data. (b) Results with thresholding, no scaling. (c) Results with scaling, but no thresholding of the data. (d) Results using both scaling and thresholding. For ease of comparison, each horizontal line in the four plots corresponds to the same sub-domain as indicated in (d), though the sub-domain has a higher number of non-zeros when thresholding is not used.

We next considered the performance of L_1 magic at other time steps. We know that in the simulation, the number of coherent structures changes over time. At early time steps, there is a small number of coherent structures, which grow larger in size over time, until they split into multiple structures. Thus the sparsity of the data reduces over time, and we would expect that CS techniques would start performing poorly at late time. Figure 6 shows the performance of L_1 magic with a sparse measurement matrix evaluated using R^2 on data at time steps 2000, 3000 and 4000 using scaling and thresholding. These results suggest that compression using CS is an option when the sparsity is more than 15% and less than 50%. For sparsity lower than the lower limit, it is preferable to just store the non-zero values and their locations in three dimensions. For sparsity greater than the upper limit, CS does not give perfect reconstruction.



(a) Time step 2000 (b) Time step 3000 (c) Time step 4000

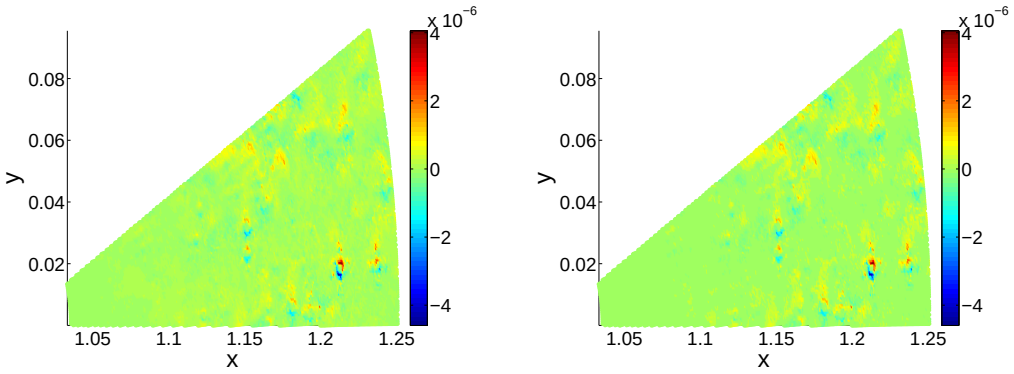
Figure 6. Performance of L_1 magic with a sparse measurement matrix at time steps 2000, 3000 and 4000 evaluated using R^2.

Finally, we evaluated the effectiveness of CS in preserving small but unusual structures in the data. Our earlier work had shown that, at time step 3000, there are some small structures, composed of a few points, with large negative values of the variable, unlike other structures in the data that have positive values. When the simulations are run for scientific discovery, it is important that any compression preserve such structures so they can be analyzed further. Figure 7 and Table 2 show that this is possible using CS, even with a compression ratio of 72%.

Table 2. Performance of compressed sensing on sub-domain data at time steps 3000.

k/n	0.38	0.38	0.38	0.38	0.38	**0.38**	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38
m/n	0.60	0.63	0.65	0.68	0.70	**0.73**	0.75	0.78	0.80	0.83	0.85	0.90	0.95	0.98
R^2	0.87	0.89	0.91	0.93	0.95	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

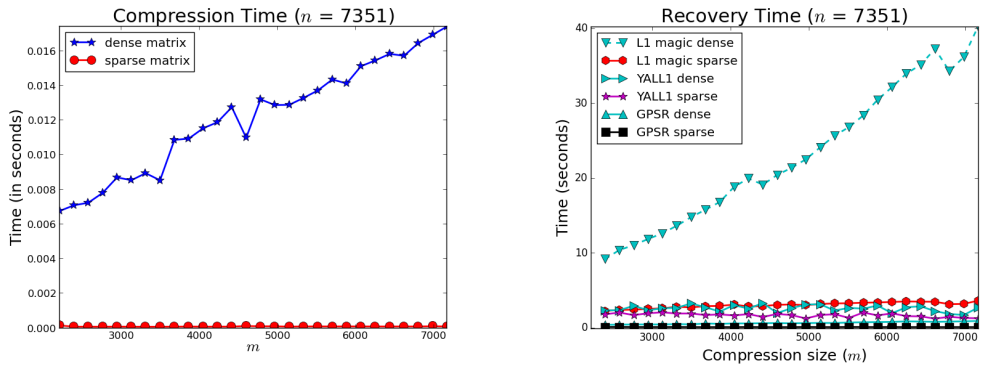
6.1.4. Comparing the use of dense vs. sparse measurement matrix Next, we compare the use of a dense matrix vs. a sparse matrix in L_1 magic, GPSR and YALL1, using the metrics of compression time, recovery time and reconstruction accuracy. The dense matrix is generated using random numbers drawn independently from a



(a) Original data (b) Reconstructed data using $m/n = 72\%$, giving $R^2 = 1$

Figure 7. Sub-domain data at time steps 3000 before and after applying preprocessing and compressed sensing. Using 72% compression rate, CS reconstructed the patterns perfectly with $R^2 = 1$ after the background grid points were set to zero. The small structures near the lower right with large positive and negative values next to each other are of scientific interest.

Gaussian distribution. The sparse matrix is generated using $d = 8$, which divides each column of $\mathbf{A}$ into d segments, randomly assigns one element in each division to be 1, and sets the rest to 0. This sparse matrix satisfies the requirements for SMP, SSMP and Count-Min algorithms to work and therefore, allows us to compare all methods on an equal footing.

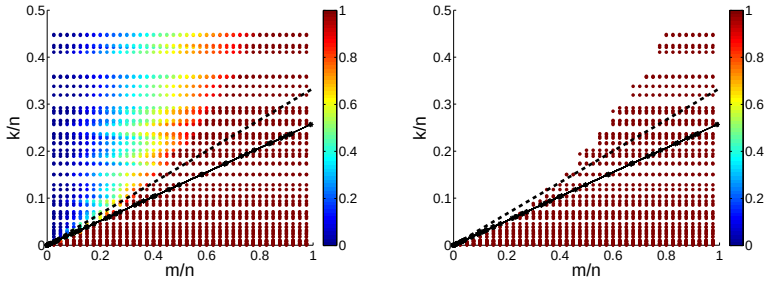


(a) (b)

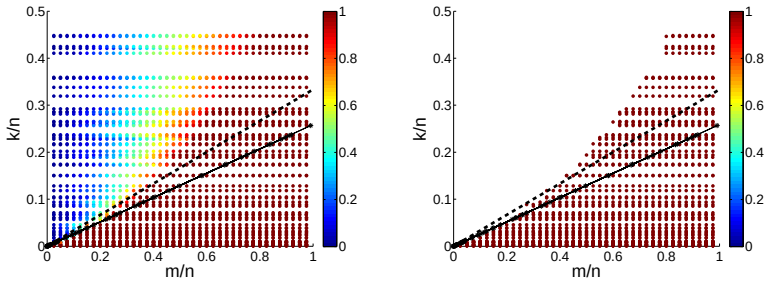
Figure 8. Comparison of (a) compression and (b) reconstruction times using L_1 magic, GPSR and YALL1 with a dense and a sparse matrix on a subdomain data of length $n = 7351$ and sparsity $k = 26$ at time step 2000.

Since we compress data $\mathbf{u}$ using a matrix $\mathbf{A}$, the compression time is the time for the calculation of $\mathbf{Au}$, and is the same for all methods. Figure 8(a) compares the compression time using a dense matrix and a sparse matrix on a subdomain of length $n = 7351$ and sparsity $k = 26$ at time step 2000, as the size of compression m is varied. The compression time using a dense matrix obviously increases linearly as m increases; in contrast, using a sparse matrix takes a smaller amount of time, which is almost constant with increasing m. The corresponding recovery times for L_1 magic, GPSR and YALL1 with both dense and sparse matrices on the same subdomain data at time step 2000 are shown in Figure 8(b), with additional details shown in Figure 13. Similarly using a dense matrix takes dramatically more time for the recovery, especially in the case of L_1 magic. In general, the recovery times increase as the compression size m increases, except for YALL1.

Figure 9(a) and Figure 9(c) display the reconstruction accuracy of L_1 magic using a dense matrix and a sparse matrix, respectively, for time step 2000. Figures 9(b) and (d) show just the points where $R^2 = 1$, that is, the points when L_1 magic gives perfect reconstruction. Focusing on the points with perfect reconstruction, we observe that the performance of a dense matrix is very similar to that of a sparse matrix. There are only a few points that are

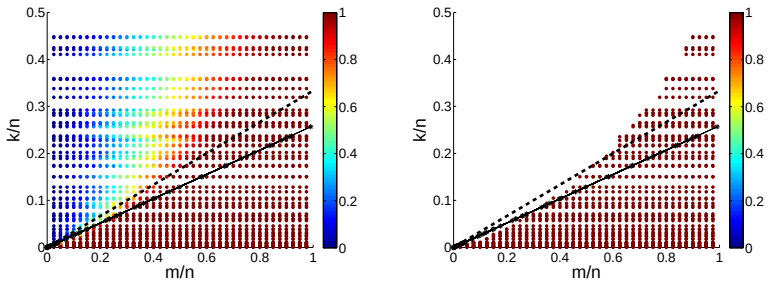


(a) $L1$-magic: dense (b) $L1$-magic: dense ($R^2 = 1$)

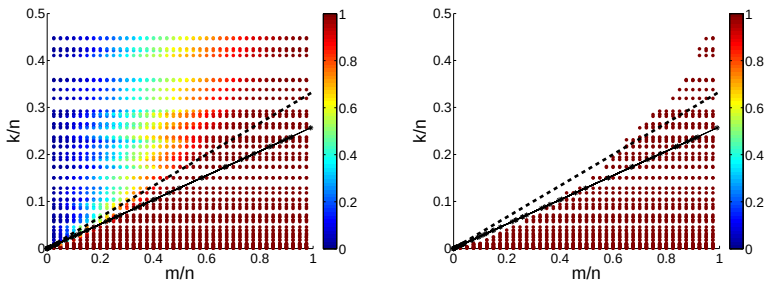


(c) $L1$-magic: sparse (d) $L1$-magic: sparse ($R^2 = 1$)

Figure 9. Comparison of reconstruction accuracy of $L1$-magic using (a) a dense matrix and (c) a sparse matrix. (b) and (d) display just the points that satisfy $R^2 = 1$.



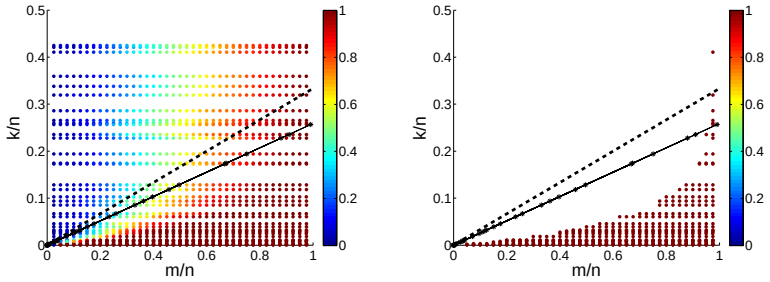
(a) GPSR: dense (b) GPSR: dense ($R^2 = 1$)



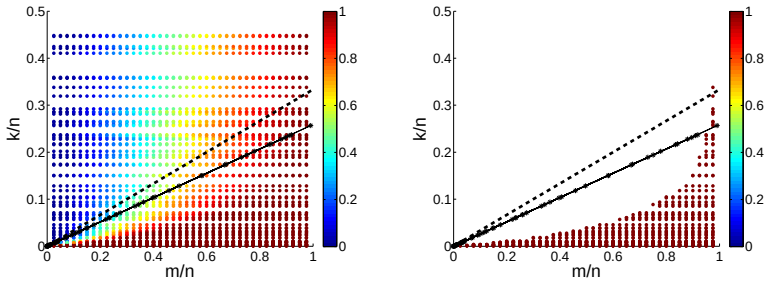
(c) GPSR: sparse (d) GPSR: sparse ($R^2 = 1$)

Figure 10. Comparison of reconstruction accuracy of GPSR using (a) a dense matrix and (c) a sparse matrix. (b) and (d) display just the points that satisfy $R^2 = 1$.

different. A similar result is observed for GPSR and YALL1 with dense and sparse matrices as shown in Figure 10 and Figure 11.



(a) YALL1: dense (b) YALL1: dense ($R^2 = 1$)



(c) YALL1: sparse (d) YALL1: sparse ($R^2 = 1$)

Figure 11. Comparison of reconstruction accuracy of YALL1 using (a) a dense matrix and (c) a sparse matrix. (b) and (d) display just the points that satisfy $R^2 = 1$.

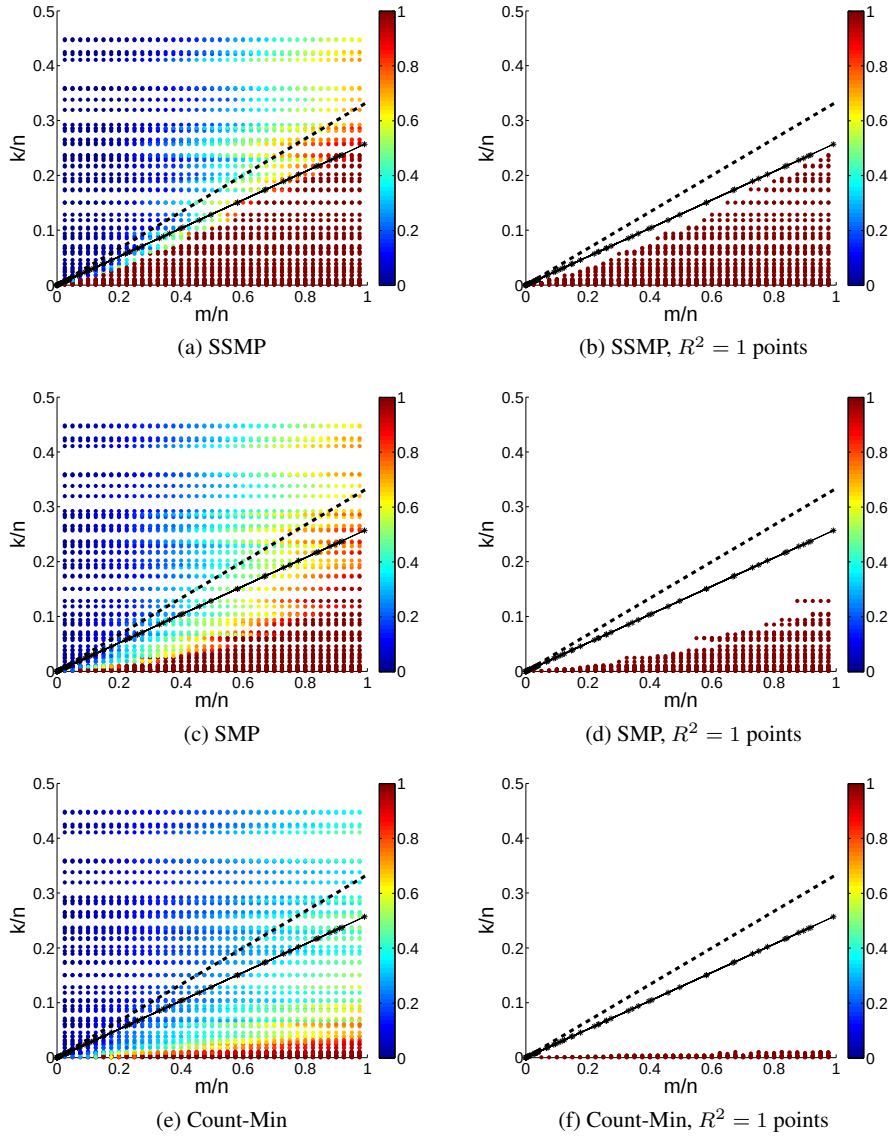
Our results suggest that for L_1 magic, GPSR and YALL1, using a sparse matrix is a better choice compared to using a dense matrix. The use of a sparse matrix gives similar reconstruction accuracy and requires a significantly smaller amount of time in compression and recovery.

6.2. Comparison of CS with other sparse recovery algorithms

We next compare the performance of CS using L_1 magic, GPSR and YALL1 with the other sparse recovery methods described in Section 4. Recall that, unlike compressed sensing algorithms, the sparse recovery algorithms have no guarantee of exact reconstruction. Figure 12 displays the reconstruction accuracy of SMP, SSMP and Count-Min using a sparse matrix for time step 2000, along with the points with value $R^2 = 1$ where we have perfect reconstruction. SSMP gives perfect reconstruction with compression size m almost aligned with the theoretically suggested compression size m^* (the straight line in Figure 12(b)) for CS. The results also show that SMP does not perform as well as SSMP, and Count-Min only works well when data are extremely sparse. All three sparse recovery methods and the group sparse recovery method (YALL1) have fewer points with $R^2 = 1$ when compared with L_1 magic and GPSR.

Next, in Figure 13, we compare the recovery time for the different methods. For a fair comparison, we use the same subdomain as in the case of L_1 magic from time step 2000, the results for which are shown in Figure 9. We also consider a range of m values such that we have full recovery ($R^2 = 1$) for some points for all the methods. The subdomain contains $n = 7351$ signals with sparsity $k = 26$, which gives $\frac{k}{n} = 0.0035$. Since all our sub-domains have roughly the same signal size $n \approx 7400$, we did not vary n as a factor in the recovery time. These results show that L_1 magic with a sparse matrix, and GPSR with a dense matrix, require more time for the recovery step, with the time increasing as the compression size m increases. Recovery times for YALL1 with both dense and sparse matrices are mostly in between the times for L_1 magic and GPSR. The remaining methods in decreasing order of recovery times are SSMP, GPSR sparse, SMP and Count-Min; all methods are almost indifferent to the compression size m.

Based on the performance of all methods using compression time, recovery time and accuracy, we make the following observations: GPSR is a good method if a shorter recovery time is desired; Count-Min is the fastest



(a) SSMP (b) SSMP, $R^2 = 1$ points

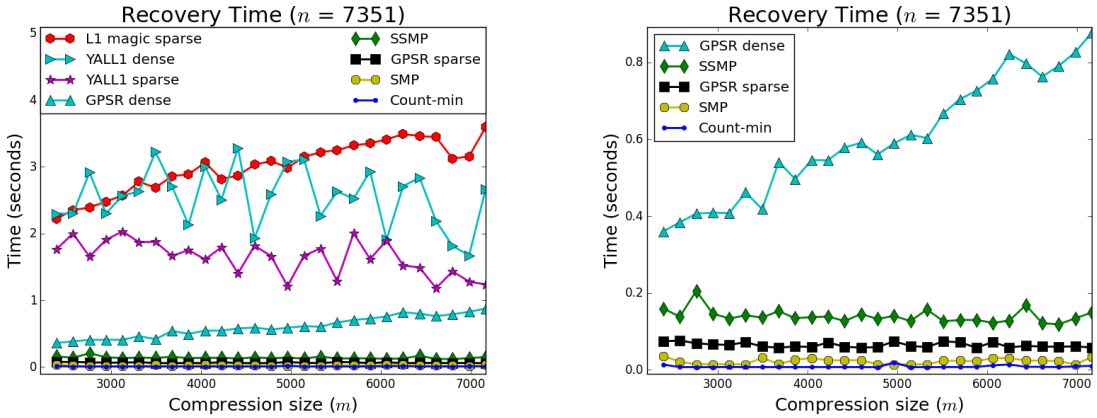
(c) SMP (d) SMP, $R^2 = 1$ points

(e) Count-Min (f) Count-Min, $R^2 = 1$ points

Figure 12. Performance comparison of SSMP, SMP and Count-Min using sparse matrix for time step 2000, showing just the points with $R^2 = 1$ in the right column.

method but only works well when data are extremely sparse; and SSMP is particularly better than SMP in reconstruction accuracy but takes longer time in recovery. L_1 magic gives the best performance in accuracy but takes a much longer time than the other methods. It also takes longer to compress and recover the signal using a dense matrix than using a sparse matrix for L_1 magic, GPSR and YALL1. We suggest using L_1 magic if the goal is to obtain perfect reconstruction as it has more points with $R^2 = 1$ that meet the criterion of using less storage than the non-zero values and their locations.

Finally, we compare how well different methods perform in preserving the small structures of interest in time step 3000 (see Figure 7). In Figure 14, we show the region around the structures after thresholding and the difference between this and the reconstruction by each of the methods. The reconstructed data by each method appears visually similar to the thresholded data, so we present the difference plot to clearly show the quality of reconstruction for



(a) Sparse recovery

(b) A zoomed-in view of (a)

Figure 13. Comparisons of recovery times for the different methods. The values of m are chosen such that all six methods give full recovery ($R^2 = 1$). Results for L_1 magic with a dense matrix are shown in Figure 8.

each method. For each plot, we list the minimum and maximum values in the thresholded data or the difference data sets. We observe that even though the worst performing method appears to have a large difference from the thresholded data in Figure 14(h), this difference is in the range of [-7.6e-12, 5.1e-12], while the thresholded data are in the range [-4.6e-6, 4.1e-6]. This suggests that the sparse recovery methods can be an alternative when perfect reconstruction is not required.

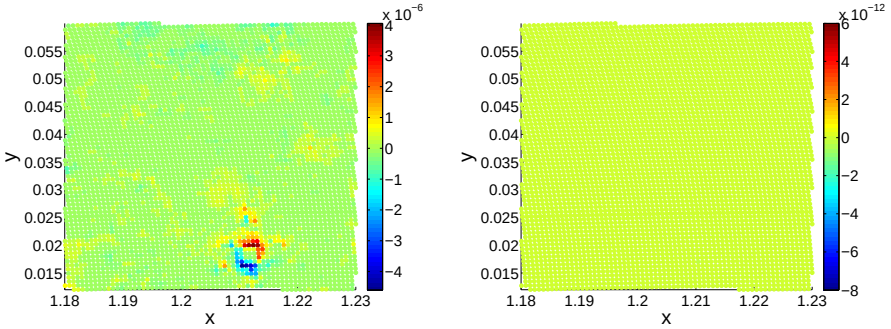
7. Conclusions

In this paper, we considered practical issues in the application of compressed sensing to simulation data. We were motivated by the need to reduce the size of the data being written out from simulations run on massively-parallel processors, while enabling accurate reconstruction of the data for quantitative analysis. Using data from a plasma physics simulation, we showed how we could scale and threshold the data that are distributed across processors and determine the amount of compression that would enable near perfect reconstruction. We found that a successful application of CS is bounded by the percentage of sparsity in the data - the data have to be sparse enough for compression using CS, but not so sparse that it is more cost effective to just write out the locations and values of the non-zero data points.

Our experiments, conducted using the R^2 metric, show that scaling the data is very helpful and thresholding helps both with compression and the coherent structure analysis that is performed on the data. After comparing the performance of CS and sparse recovery algorithms, we suggest using L_1 magic with a sparse measurement matrix for CS if the goal is to obtain near perfect reconstruction, even though the method takes longer time for the recovery task. Our future work will focus on determining to what extent we can relax the perfect reconstruction requirement and still obtain results close to what we would get by analyzing the original data.

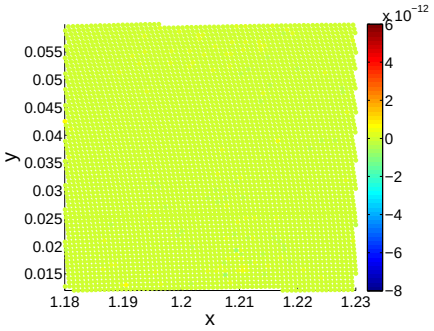
Acknowledgment

LLNL-JRNL-681293. We thank Zhihong Lin, UC Irvine, for providing access to the data that was generated as part of the GSEP SciDAC project. This work was done as part of the Exa-DM project, funded by Dr. Lucy Nowell, program manager, ASCR, US Department of Energy. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

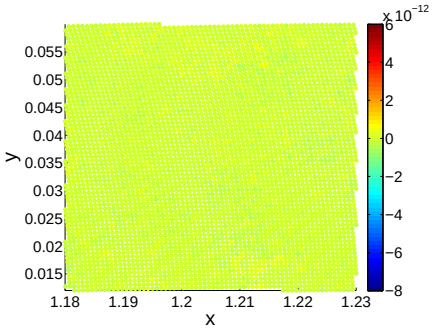


(a) Data after thresholding
[-4.6e-6, 4.1e-6]

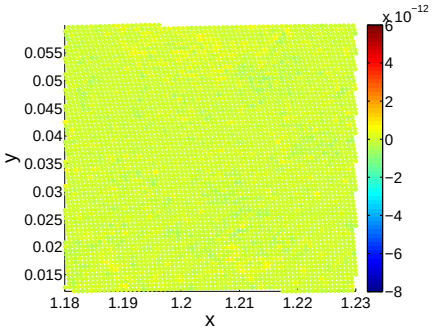
(b) L1 magic dense ($R^2 = 1.0$)
[-2.6e-13, 1.8e-13]



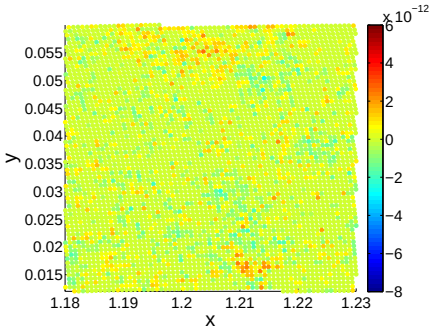
(c) L1 magic sparse ($R^2 = 0.9990$)
[-7.5e-13, 6.7e-13]



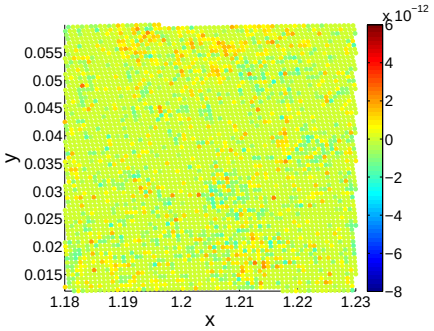
(d) GPSR dense ($R^2 = 0.9986$)
[-2.5e-13, 2.4e-13]



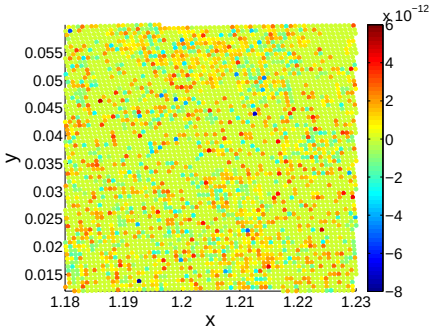
(e) GPSR sparse ($R^2 = 0.9895$)
[-6.5e-13, 7.8e-13]



(f) SMP ($R^2 = 0.8347$)
[-2.7e-12, 2.7e-12]



(g) SSMP ($R^2 = 0.8260$)
[-2.7e-12, 2.7e-12]



(h) Count-min ($R^2 = 0.5898$)
[-7.6e-12, 5.1e-12]

Figure 14. A slice from sub-domain data at time step 3000 before and after applying compression methods. (a) displays the thresholded data. (b)-(h) show the difference between the thresholded data and the reconstructed data for each method. Numbers in $[\min, \max]$ are the minimum and maximum values in each plot.

REFERENCES

[1] R. Berinde and P. Indyk. Sequential sparse matching pursuit. In *Communication, Control, and Computing, 2009 47th Annual Allerton Conference on*, pages 36–43, Sept 2009.

[2] R. Berinde, P. Indyk, and M. Ruzic. Practical near-optimal sparse recovery in the L1 norm. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 198–205, Sept 2008.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, January 2010.

[4] E. Candes and J. Romberg. 1-magic: Recovery of sparse signals via convex programming. http://users.ece.gatech.edu/~justin/l1magic/, 2005.

[5] E.J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489–509, Feb 2006.

[6] E.J. Candes and T. Tao. Decoding by linear programming. *Information Theory, IEEE Transactions on*, 51(12):4203–4215, Dec 2005.

[7] E.J. Candes and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *Information Theory, IEEE Transactions on*, 52(12):5406–5425, Dec 2006.

[8] H. Childs et al. In Situ Processing. In E. Wes Bethel, Hank Childs, and Charles Hansen, editors, *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 171–198. CRC Press/Francis–Taylor Group, Boca Raton, FL, USA, 2012.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58 – 75, 2005.

[10] W. Deng, W. Yin, and Y. Zhang. Group sparse optimization by alternating direction method. volume 8858, pages 88580R–88580R–15, September 2013.

[11] D.L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, April 2006.

[12] Y. J. Fan and C. Kamath. Practical considerations in applying compressed sensing to simulation data. In *Proceedings of the IEEE Data Compression Conference*, page 479, Piscataway, NJ, USA, 2015.

[13] M.A.T. Figueiredo, R.D. Nowak, and S.J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *Selected Topics in Signal Processing, IEEE Journal of*, 1(4):586–597, Dec 2007.

[14] N. Fout and K.-L. Ma. Transform coding for hardware-accelerated volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1600–1607, Nov 2007.

[15] S. B. Gokturk, C. Tomasi, B. Girod, and C. Beaulieu. Medical image compression based on region of interest, with application to colon CT images. In *Proceedings of the 23rd Annual EMBS International Conference*, pages 2453–2456, Piscataway, NJ, USA, 2001. IEEE.

[16] Hcompress image compression software Web page, 2015. http://www.stsci.edu/software/hcompress.html.

[17] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *Proceedings, Conference on Very Large Data Bases*, pages 363–372, 2000.

[18] P. Indyk and M. Ruzic. Near-optimal sparse recovery in the L1 norm. In *Proceedings, 49th Symposium on Foundations of Computer Science*, pages 199–207, October 2008.

[19] M. Isenburg, P. Lindstrom, and J. Snoeyink. Lossless compression of predicted floating-point geometry. *Comput. Aided Des.*, 37(8):869–877, July 2005.

[20] J. Iverson, C. Kamath, and G. Karypis. Fast and effective lossy compression algorithms for scientific datasets. In *Proceedings of the 18th International Conference on Parallel Processing*, Euro-Par'12, pages 843–856, Berlin, Heidelberg, 2012.

[21] C. Kamath, J. Iverson, R. Kirk, and G. Karypis. Detection of coherent structures in extreme-scale simulations. In *Proceedings of the Exascale Research Conference*, April 2012.

[22] D. Laney, S. Langer, C. Weber, P. Lindstrom, and A. Wegener. Assessing the effects of data compression in simulations using physically motivated metrics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 76:1–76:12, New York, NY, USA, 2013. ACM.

[23] Z. Lin, T. S. Hahm, W. W. Lee, W. M. Tang, and R. B. White. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science*, (1835):281, 1998.

[24] P. Lindstrom. Fixed-rate compressed floating-point arrays. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2674–2683, Dec 2014.

[25] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1245–1250, Sep-Oct 2006.

[26] R. Saab, R. Wang, and O. Yilmaz. Near-optimal compression for compressed sensing. In *Data Compression Conference (DCC), 2015*, pages 113–122, April 2015.

[27] M. Salloum, N. Fabian, D. M. Hensinger, and J. A. Templeton. Compressed sensing and reconstruction of unstructured mesh datasets. arXiv:1508.06314, August 2015.

[28] K. Sayood. *Introduction to Data Compression (3rd Ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[29] R. Soundararajan and S. Vishwanath. Quantization using compressive sensing. In *Information Theory and Applications Workshop (ITA), 2011*, pages 1–6, Feb 2011.

[30] J.A. Tropp and A.C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, Dec 2007.

[31] J. Yang and Y. Zhang. Alternating direction algorithms for l_1-problems in compressive sensing. *SIAM Journal on Scientific Computing*, 33(1):250–278, 2011.