Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Spline-based parallel nonlinear optimization of function sequences

Tal Ben-Nun [a,b,*], Amnon Barak [a], Uri Raviv [b]

[a] *Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem 9190401, Israel*
[b] *Department of Chemistry, The Hebrew University of Jerusalem, Jerusalem 9190401, Israel*

HIGHLIGHTS

- Presents a scalable algorithm for optimizing sequences of functions in parallel.
- Develops an objective function for modeling nonlinear dynamical system optimization.
- Demonstrates three real-world applications and analyzes the results.
- Performance shows that the algorithm benefits from heterogeneous HPC clusters.
- Two distributed variants of the algorithm are proposed and evaluated.

ARTICLE INFO

Article history:
Received 6 January 2015
Received in revised form
22 September 2015
Accepted 21 April 2016
Available online 28 April 2016

Keywords:
Nonlinear optimization
Parallel optimization
Dynamical systems
Curve fitting

ABSTRACT

Nonlinear dynamical system optimization problems exist in many scientific fields, ranging from computer vision to quantitative finance. In these problems, the underlying optimized parameters exhibit a certain degree of continuity, which can be formulated as a discrete sequence of nonlinear functions. Traditionally, such problems are either solved by ad-hoc algorithms or via independent optimization of the underlying functions. The former solutions are difficult to define and develop, requiring expertise in the field, while the latter approach does not take advantage of the inherent sequential properties of the functions. This paper presents a parallel spline-based algorithm for nonlinear optimization of function sequences, with emphasis on dataset sequences that represent dynamically evolving systems. The presented algorithm provides results that are more coherent with fewer evaluations than independent optimization of the sequence functions. We elaborate on the heuristic approach, the motivation behind using splines to model dynamical systems, and the various tiers of concurrency built into the algorithm. Furthermore, we present two distributed variants of the algorithm and compare their convergence with the serial version. The performance of the algorithm is demonstrated on benchmarks and real-world problems in audio signal decomposition, small angle X-ray scattering analysis, and video tracking of arbitrary objects.

1. Introduction

Large-scale nonlinear optimization problems are recently gaining importance in the scientific community. One subset of these problems focuses on modeling dynamical systems. In this subset, the objective function contains an additional continuous dimension (e.g. time). Since real-world experiments consist of discrete observations, the problem can be formulated as a sequence of related functions to optimize. Examples of nonlinear dynamical system problems include analytical mechanics and physics simulations [23,25]; sensor-based (e.g. GPS) navigation [35]; nonlinear econometric models in financial time series [52]; and other time-evolving processes such as phase transitions in thermodynamic systems.

Current solutions to sequential problems are often area-specific [41,43,13,45], allowing little to no flexibility on the structure of the data (i.e., algorithms that operate on videos will usually not perform well on audio waves); do not leverage parallel and distributed systems; and are not always robust to various input conditions, such as noise. Furthermore, devising such solutions requires deep understanding of the research area, with which specifically-crafted heuristics are added to the algorithms. On the other hand, nonlinear modeling is simpler and only requires basic knowledge of the underlying process.

The importance of function sequence optimization stems from the fact that it estimates models more efficiently by using less parameters and assuming continuity. Moreover, the results of

* Corresponding author at: Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem 9190401, Israel.
E-mail address: talbn@cs.huji.ac.il (T. Ben-Nun).

http://dx.doi.org/10.1016/j.jpdc.2016.04.011
0743-7315/© 2016 Elsevier Inc. All rights reserved.

optimized long sequences (e.g., a month of video footage) can be automatically subdivided to segments using the underlying parameters [24,47,4], a process which aids in finding anomalies and the critical points of the dynamical system.

Traditional optimization of each sequence function independently does not guarantee the coherency of the solution with respect to the sequence, nor produce well-informed parameter estimates based on the continuous properties of the system. This results in an unnecessarily large amount of function evaluations, which may be costly for some problems (e.g., when using numerical integration). More importantly, such algorithms are not scalable with respect to the length of the sequence, with which the memory consumption can grow linearly and sometimes quadratically, depending on the method.

In this paper, we design a novel algorithm that we call Discrete Sequence Optimization (DSO). This algorithm provides a generalized solution to constrained and unconstrained optimization of nonlinear function sequences. As we shall show, DSO converges to better solutions with less evaluations when the underlying functions represent a dynamical system.

Parallel and distributed nonlinear optimization algorithms generally provide better results than their sequential counterparts [44] by evaluating many estimates simultaneously to evade sub-optimal local convergence. The DSO algorithm is inherently parallel and can be distributed across many computing nodes, essentially capable of achieving maximal performance on HPC clusters with multi-GPU machines. It is designed with three tiers of concurrency: data parallelism, task parallelism, and distributed computing. For the distributed computing tier, two different variants of DSO are presented in this paper. These variants utilize distributed global optimization principles to achieve more accurate results than the sequential version.

We also show that the memory complexity of DSO is independent of the length of the sequence. This is especially important for solving large-scale optimization problems on massively parallel architectures, which typically contain a limited amount of RAM and no memory swapping capabilities.

To demonstrate the effectiveness of DSO, we present two benchmarks and three real-world applications for sequence optimization. These applications cover a wide range of research areas, optimizing audio signals for source instrument decomposition, analyzing complex fluid X-ray scattering datasets, and fitting image sequences to track arbitrary objects in videos.

The paper is organized as follows. Section 2 depicts the theoretical background for the algorithm, used throughout the paper. Section 3 presents DSO in detail, along with its distributed counterparts. Case studies that demonstrate the accuracy of DSO are presented in Section 4, and various performance parameters of the algorithm are evaluated in Section 5. Related work and conclusions are discussed in Sections 6 and 7 respectively.

2. Background

This section provides theoretical background for the algorithm presented in this paper.

2.1. Statement of the problem

The problem of unconstrained nonlinear dynamical system optimization can be formally defined as follows. Given an objective function $f : \mathbb{R}^{d+1} \to \mathbb{R}$, denoted as $f(\tau, x)$ where $x \in \mathbb{R}^d$ is some parameter vector, f is defined in the range $\tau \in [a, b]$ and smooth with respect to τ; let g be a scalarization function, which quantifies the values of f in the range $\tau \in [a, b]$ to a real value; find a continuously differentiable parameter function $\mathcal{X}^*(\tau) : \mathbb{R} \to \mathbb{R}^d$ that satisfies:

$$\mathcal{X}^* = \arg\min_{\mathcal{X}:\mathbb{R}\to\mathbb{R}^d} g\left(f, \mathcal{X}, a, b\right).$$

A specific instance of this problem, where $g(f, \mathcal{X}, a, b) \equiv \int_a^b f(\tau, \mathcal{X}(\tau), \mathcal{X}'(\tau))d\tau$, solves the Euler–Lagrange equation [16], which is prominently used in analytical mechanics.

This paper focuses on the discrete version of this problem, finding the parameter matrix $X^* \in \mathbb{R}^{m\times d}$ that satisfies the equation:

$$X^* = \arg\min_{X\in\mathbb{R}^{m\times d}} g\begin{pmatrix} f_1\left(X_{1,*}\right) \\ \vdots \\ f_m\left(X_{m,*}\right)\end{pmatrix}, \tag{1}$$

where $f_t : \mathbb{R}^d \to \mathbb{R}$, $t \in [m]$ is a discrete sequence of m related functions, exhibiting sequential continuity (see below); X^* is sequentially continuous along its rows; and $X_{t,*}$ is equivalent to row t of X.

Since optimizing the f_t functions independently can produce incoherent results with respect to the sequence, it is necessary to find an objective function $g : \mathbb{R}^m \to \mathbb{R}$ that constrains the parameter values along the sequence in order to enforce continuity. To address this issue, we define a metric for sequential continuity as the discrete version of function smoothness, using finite differences instead of derivatives. A parameter sequence is continuous if the sum of its absolute second-order differences, $\|\ddot{X}_t\|$ (defined in Appendix A.1) for $1 < t < m$, is minimal. The scalarization function g is therefore given by:

$$g(\vec{f}, X, \lambda) \equiv \sum_{t=1}^{m} f_t\left(X_{t,*}\right) + \lambda \sum_{t=2}^{m-1}\sum_{i=1}^{d} |c_i\ddot{X}_{t,i}|, \tag{2}$$

where $c \in \mathbb{R}^d$ is a fixed non-negative vector that normalizes the dimensions of the parameters, and $\lambda \geq 0$ is a regularization parameter for the second term. In this paper, the two parts of Eq. (2) are referred to as the data and smoothness terms respectively. The second term functions as an ℓ_1-norm based regularization, applied via Lagrangian relaxation [27]. This technique follows the principle of Tikhonov regularization [50], adding the sequential continuity constraint to the minimized function directly.

2.2. Constrained sequence optimization

The respective constrained problem of discrete sequence optimization is defined by:

$$\begin{aligned} X^* = \arg\min_{X\in\mathbb{R}^{m\times d}} \quad & g(\vec{f}, X, \lambda), \\ \text{s.t.} \quad & X \in C_g \ \wedge\ \forall_{t\in[m]} : X_{t,*} \in C_\ell, \end{aligned}$$

where $C_g \subset \mathbb{R}^{m\times d}$ is the feasible subset of the result matrix, representing global constraints; and $C_\ell \subset \mathbb{R}^d$ is the feasible subset of individual results, representing local constraints. Global constraints enforce sequential coherency of the optimization results, whereas local constraints restrict the optimization of each function independently.

For example, linear box constraints, which are extensively used throughout the paper, can be defined by $C_g = \left\{x' \in \mathbb{R}^{dm} : A_g x' \leq b_g\right\}$ and $C_\ell = \left\{x \in \mathbb{R}^d : Ax \leq b\right\}$, where n, k are the number of global and local constraints respectively, $A_g \in \mathbb{R}^{n\times dm}$, $b_g \in \mathbb{R}^n$, x' is the vector representation of X (having $X_{i,j} = x'_{id+j}$), $A \in \mathbb{R}^{k\times d}$ and $b \in \mathbb{R}^k$.

2.3. Curve fitting of dataset sequences

A common sub-problem of nonlinear dynamical system optimization is time series analysis, in which the input is a single model function $F : \mathbb{R}^{d+1} \to \mathbb{R}$ with d parameters, denoted as $F(q, \vec{\theta})$, where q corresponds to a data point in the model and $\vec{\theta}$ corresponds to a parameter vector. F is fit to a sequence of given datasets (or observations), defined by a matrix $D \in \mathbb{R}^{m\times k}$ consisting of m datasets with k points, using a vector $\vec{q} \in \mathbb{R}^k$ as source points for all the observations to be modeled by. The objective function f_t for dataset sequence curve fitting is given by:

$$f_t(\vec{\theta}_t) \equiv \frac{1}{k}\sum_{i=1}^{k}\left(F(q_i, \vec{\theta}_t) - D_{t,i}\right)^2,$$

where $\vec{\theta}_t \equiv X_{t,*}$. This essentially defines the Mean Squared Error (MSE) of the estimate from the observation. Note that the division by k can be omitted for the case of minimization.

3. The discrete sequence optimization algorithm

In this section, we present the novel Discrete Sequence Optimization (DSO) algorithm in detail, elaborating on its core components and heuristics. Additionally, we analyze its convergence properties and present two distributed variants of the algorithm.

The DSO algorithm, depicted in Algorithm 1, consists of two phases: initialization and propagation. The initialization phase (line 1), shown in Section 3.1, assigns initial values to parameters and performs randomized optimization. The propagation phase (line 6), detailed in Section 3.2, uses parameter continuity to disseminate results from a source point to the entire sequence, using spline-based heuristics to extrapolate the parameters. During both phases, the algorithm enforces local and global constraints. The optimization process continues until it fails to find a better parameter matrix for a specified number of consecutive iterations (line 14).

To optimize each of the individual functions, DSO uses existing nonlinear optimization algorithms (ϕ_i in the algorithm), such as L-BFGS [9] and Levenberg–Marquardt [29,32]. This modularity enables the user to choose optimization algorithms that work best for each function, and, if the algorithms are locally convergent, ensures that the DSO algorithm will converge as well.

DSO contains three layers of concurrency: data parallelism, task parallelism and distributed computing. The first two layers are incorporated within the DSO optimizer, which attempts to minimize a given sequence of functions, whereas the third layer distributes several instances of the algorithm, using principles from stochastic optimization and genetic algorithms to achieve global convergence.

Due to the high degree of independent computations, the algorithm is ideal for heterogeneous computing, especially on massively parallel accelerators. For example, the evaluation of model functions $F(q, \vec{\theta})$ when fitting dataset sequences (see Section 2.3) performs identical computations on different values of q, exhibiting data parallelism. Thus, when applicable (e.g., for large datasets), the core optimizer computes model functions and numerical derivatives concurrently.

3.1. Initialization

The initialization phase, shown in Algorithm 2, assigns initial estimates to the matrix X. These estimates can either be provided by the user or undefined, causing them to be randomly chosen from a per-parameter input range using the uniform distribution (line 2 in the algorithm).

Algorithm 1: Discrete Sequence Optimization

```
Input: $\vec{f}$, $\vec{\phi}$, $\lambda$, $conv$, initial estimate matrix $X_0$, bounds $\vec{\ell}$, $\vec{L}$.
Optional Input: $C_\ell$, $C_g$ (local and global constraints)
Output: Final parameter matrix, $\vec{f}$ values.

1  $X$ = Initialize($\vec{f}, \vec{\phi}, C_\ell, C_g, X_0, \vec{\ell}, \vec{L}$);
2  Compute $g(\vec{f}, X, \lambda)$;
3  $X_{old} = X$;
4  repeat
5      Enforce $X \in C_g$, if specified;
6      $X$ = Propagate($\vec{f}, \vec{\phi}, X, C_\ell, C_g, \lambda$);
7      if $g(\vec{f}, X, \lambda) \geq g(\vec{f}, X_{old}, \lambda)$ then
8          $conv = conv - 1$;
9          $X = X_{old}$;
10     else
11         $conv = conv + 1$;
12         $X_{old} = X$;
13     end
14 until $conv = 0$ or for a specified number of iterations;
15 return $X$;
```

Formally, the initialization step is the randomized process of choosing estimates for X from user-input bounds ℓ_i and L_i, such that for any estimate x, $\ell_i \leq x_i \leq L_i$. The matrix produced by the initialization phase is given by:

$$X_1 = \begin{pmatrix} \phi_1(\vec{x}_1^0) \\ \vdots \\ \phi_m(\vec{x}_m^0) \end{pmatrix},$$

where $\vec{x}_t^0 \equiv (X_0)_{t,*}$ and $\forall_{i\in[d],t\in[m]} : (X_0)_{t,i} \sim U(\ell_i, L_i)$.

Each function in the sequence is optimized independently and in parallel, using ϕ_t as the local optimizers (lines 3–5). If any of the optimized estimates do not satisfy the global constraints (line 6), the matrix is uniformly re-randomized in the range $[\ell_i, L_i]$ until achieving feasibility. Local constraints are preserved during randomization and independent optimization.

3.2. Propagation

The propagation phase is the primary iterative portion of DSO. In each iteration of this phase, depicted in Algorithm 3, a "source point" function is selected, individually optimized (using ϕ_i), and the results are then sequentially propagated to the neighboring functions in both directions of the sequence. During this process, the algorithm enforces local and global constraints, if specified. This phase attempts to locally minimize the smoothness term of Eq. (2) while optimizing the sequence functions.

Algorithm 2: Initialization Phase

```
Input: $\vec{f}, \vec{\phi}, C_\ell, C_g, X_0, \vec{\ell}, \vec{L}$
Output: $X$

1  repeat
2      $X$ = Randomize undefined parameters uniformly in
       $[\ell_i, L_i]$;
3      for each function $f_t$ in $\vec{f}$ do in parallel
4          $X_{t,*}$ = Optimize $f_t$ with $\phi_t$, $X_{t,*}$ and $C_\ell$;
5      end
6  until $X \in C_g$;
7  return $X$;
```

Table 1
Parameter extrapolation heuristics.

Extrapolation heuristic	Number of neighboring vectors	Description
Perturbation	0	Random modification of the destination parameter vector.
Copy	1	Copying the vector from the neighboring source. (1-point cubic spline)
Linear	2	Linear extrapolation. (2-point cubic spline)
Spline (3 points)	3	Using 2 vectors from the direction of the source and 1 following the destination.
Spline (5 points)	5	Using 3 vectors from the direction of the source and 2 following the destination.

Algorithm 3: Propagation Phase

```
Input: $\vec{f}, \vec{\phi}, X, C_\ell, C_g, \lambda$
Output: $X$

1 for $i \in [N]$ do in parallel
2     $\sigma^{(i)}$ = Index of minimal value from a random $\sqrt{m}$ subset of $\vec{f}$;
3     $X^{(i)} = X$;
4     $X^{(i)}_{\sigma^{(i)},*}$ = Optimize $f_{\sigma^{(i)}}$ with $\phi_{\sigma^{(i)}}, X^{(i)}_{\sigma^{(i)},*}$ and $C_\ell$;
5     for each $dst \in \{1, m\}$ do in parallel
6         $X^{(i)} = \texttt{Extrapolate}(\sigma^{(i)}, dst, \vec{f}, \vec{\phi}, X^{(i)}, C_\ell, C_g, \lambda)$;
7     end
8 end
9 return $\arg\min_i g(\vec{f}, X^{(i)}, \lambda)$;
```

To increase the rate of convergence, this procedure is carried out in parallel multiple times (line 1 in the algorithm), starting from several source functions concurrently in order to generate different parameter matrices. Out of these matrices, the one that resulted in the minimal g value is chosen (line 9).

In particular, the source function selection process is defined by choosing an index σ (line 2) that represents a function with the minimal optimization error from a random subset R of the sequence, as follows:

$$\sigma = \arg\min_{r \in R} f_r(X_{r,*}).$$

This step is performed to maximize the variation of the initial source point, which can greatly affect the results, in each iteration.

After optimizing the source function independently (line 4), the resulting parameter vector is simultaneously propagated (lines 5–7) in two independent directions: backward (to f_1) and forward (to f_m). This process is detailed in Algorithm 4.

During propagation in one of the directions, the parameter vector (e.g. f_σ) is extrapolated to the neighboring function ($f_{\sigma+1}$), using one of the available extrapolation methods (Algorithm 4, line 4). The resulting parameter vector is then used as an initial estimate for optimization (line 5). When the optimization process is complete, global constraints are enforced (lines 7–9) and the resulting parameter vector is propagated in the same direction (line 2).

To choose the best of the propagation heuristics, the algorithm optimizes the results of all methods (line 3, excluding inapplicable extrapolations due to sequence boundaries), and chooses the one that yields a minimal g value (line 11).

When the propagation reaches the edges of the sequence, the obtained parameters are propagated back to the source in the same manner (lines 13–23).

3.3. Extrapolation heuristics

To propagate parameter values, the DSO algorithm utilizes several extrapolation heuristics ($\mathrm{Ex}^{(i)}$ in Algorithm 4) to estimate parameter values. These heuristics are listed in Table 1.

Algorithm 4: Parameter Extrapolation

```
Input: $src, dst, \vec{f}, \vec{\phi}, X, C_\ell, C_g, \lambda$
Output: $X$

1  $\vec{x} = \{\}$;
2  for $t = src$ to $dst$ do
3      for each extrapolation method $\mathcal{Ex}^{(i)}$ do in parallel
4          $x_i = \mathrm{Ex}^{(i)}(t, dst, X_{t-3,*}, \cdots, X_{t+3,*})$;
5          $x_i$ = Optimize $f_t$ with $\phi_t, x_i$ and $C_\ell$;
6          $X'$ = $X$ with $x_i$ as row $t$;
7          if $X' \notin C_g$ then
8              Mark $x_i$ as invalid;
9          end
10     end
11     $X_{t,*} = \arg\min_i g(\vec{f}, x_i, \lambda)$;
12 end
13 for $t = dst$ to $src$ do
14     for each extrapolation method $\mathcal{Ex}^{(i)}$ do in parallel
15         $x_i = \mathrm{Ex}^{(i)}(t, src, X_{t-3,*}, \cdots, X_{t+3,*})$;
16         $x_i$ = Optimize $f_t$ with $\phi_t, x_i$ and $C_\ell$;
17         $X'$ = $X$ with $x_i$ as row $t$;
18         if $X' \notin C_g$ then
19             Mark $x_i$ as invalid;
20         end
21     end
22     $X_{t,*} = \arg\min_i g(\vec{f}, x_i, \lambda)$;
23 end
24 return $X$;
```

In the table, it is apparent that all heuristics are based on the spline model, which is general and problem-independent. The use of splines allows the algorithm to accurately predict parameter values, while preserving the first and second derivatives of the parameters across the sequence without generating oscillations. Specifically, the extrapolation heuristics are based on natural cubic splines. Note that any nonlinear extrapolation heuristic can potentially be used in the algorithm, including Extended Kalman Filters with arbitrary transition models (when applicable).

The extrapolation methods must be asymmetric, in order to allow variation between the two propagation directions; and bounded by the values of the neighboring parameter vectors. The latter is necessary in order to enforce sequence-wise regularization on the results.

The motivation behind each of the heuristics in Table 1 is to minimize the smoothness term of Eq. (2). While "Perturbation" tries to avoid convergence to local minima, "Copy" minimizes the first-order difference $|\dot{X}_t|$ by setting it to 0. Similarly, "Linear" minimizes $|\ddot{X}_{t-1}|$, and the 3 and 5-point cubic splines ("Spline") generate parameter values with sequential continuity, as well as continuous first and second-order differences.

3.4. Convergence analysis

According to the definition of convergent optimization methods (Appendix A.2), the DSO algorithm converges locally with respect to g. This is due to the convergence test in line 7 of Algorithm 1, which ensures the minimization of g with each iteration.

The rate of convergence, as well as global convergence properties, depends on the characteristics of the optimized functions. While DSO's rate of convergence is subject to ϕ_t, global convergence can be achieved by applying stochastic optimization heuristics on the core optimizer, as described in the following section. This increases the probability of finding a global minimum as a function of the number of evaluations.

3.5. *Distributed DSO*

In addition to the sequential DSO optimizer, we present two approaches for distributing the execution of the algorithm to multiple nodes. The first, Branched DSO, relies on stochastic optimization; whereas the second approach, Memetic DSO, is based on a sub-class of genetic algorithms.

In Branched DSO, the computation is broken down to P branches, each using an independent instance of the algorithm with different, randomized estimates. For each branch, an iteration is composed of M optimizer propagations. At the end of every iteration, the L branches that achieved the lowest sequence scalarization (g from Eq. (2)) values are kept for the next iteration, whereas the remaining branches are discarded and restarted with new estimates. After a pre-specified number of iterations, the branch that achieved the best scalarization value is chosen. As the number of iterations increases, global convergence follows stochastic optimization principles. In each iteration, the best branches are preserved, and new random estimates are continually generated for the others.

Branched DSO is centralized, using the master node to communicate. At the end of each iteration, the slave nodes send their g values to the master node, which in turn transmits back one of two commands: continue or re-randomize. After the final iteration, the master node requests the parameter matrix with the lowest g value and it is selected as the result. Essentially, this process requires all nodes to synchronize when the master node compares the g values.

Memetic DSO implements a Memetic Algorithm [34] (MA), a sub-class of Genetic Algorithms (GA) that performs local search (candidate refinement) for each of the population members during an iteration. GAs and MAs have traditionally been used for various nonlinear optimization problems [17]. Utilizing both the distributed characteristics of MAs and sequence scalarization as a suitable objective function, we implemented a variation of MA using the DSO optimizer as the local search method. Memetic DSO follows the global convergence principles of genetic and memetic algorithms.

Specifically, the population consists of parameter matrices (encoded as chromosome vectors) that are randomly initialized. With each iteration of the GA (generation), parts of the population are mutated (parameter values are perturbed) and recombined (creating a new parameter matrix from two existing matrices). The MA adds a per-generation candidate refinement (local search) step for each parameter matrix, which is implemented as one iteration of DSO. The overall fitness function is g.

Memetic DSO is distributed to nodes by assigning multiple candidates (portions of the population) to each node. Between generations, the nodes randomly exchange information for recombination, thereby resulting in an asynchronous distributed model. After the final generation, the best (lowest g) candidate is chosen out of the entire population.

4. Case studies

This section presents example applications of Discrete Sequence Optimization (DSO). First, we present synthetic problems in nonlinear optimization and curve fitting of dataset sequences, followed by a brief overview of case studies from audio signal decomposition, X-ray scattering analysis and video tracking. In Appendices B–E, we provide details about each problem, present the corresponding objective function, model function and constraints, followed by additional results of DSO.

The DSO algorithm was implemented in C++, OpenMP [38] and CUDA [14] as a portable task-parallel library, which was then used for each of the above applications. The optimization engine used in the examples is the Ceres Solver library [2], utilizing the L-BFGS [9] and Levenberg–Marquardt [29,32] algorithms as the underlying optimizers (ϕ_t).

4.1. *Benchmark examples*

These examples test DSO on two non-trivial functions with multiple parameters and nonlinear partial derivatives. The results are then compared to traditional independent optimization (fitting), where each function (dataset) is optimized separately.

We use the Ackley Function [1] (f_A, see Appendix B) as the benchmark for optimization. In the sequence, an offset is applied to the function, moving its global minimum. The continuous version is defined by $f_A(x - \cos t, y - \sin t)$ for $t \in [0, 2\pi]$. The minimum of f_A is obtained at $(0, 0)$ for $x, y \in [-5, 5]$. A comparison between traditional independent optimization with DSO, where the segment $[0, 2\pi]$ was divided to 200 discrete points, is shown in Fig. 1(a).

For curve fitting, the model function $F\left(q, \vec{\theta}\right) = aq^{\sin\left(\frac{bq}{10}\right)}$ was fit to synthetically-generated datasets, with relative SSE as the objective function (see Appendix D for details). Fig. 1(b) shows the results of performing DSO vs. independent curve fitting.

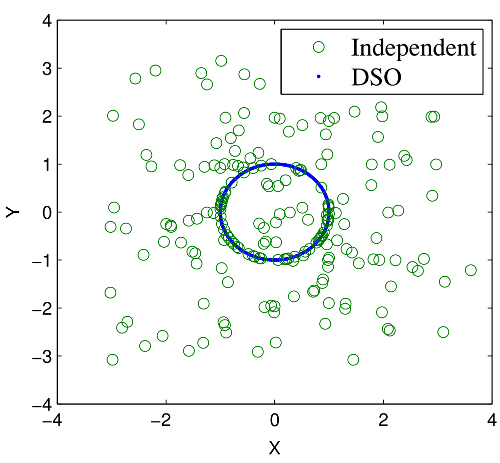
Both parts of Fig. 1 show that while some functions yield correct results, most did not globally converge in the independent case. On the other hand, parameter continuity results in global convergence for most of the functions in DSO. In particular, the scalarization (g from Eq. (2)) values of the results shown in Fig. 1(a) are 1700.85 for independent optimization and 0.19 for DSO, where $\lambda = 1$. The corresponding values for the results in Fig. 1(b) are $3, 975, 712.26$ and 32.57. We note that running DSO with $\lambda = 0$ resulted in g values of 0 in both cases.

4.2. *Audio signal decomposition and transcription*

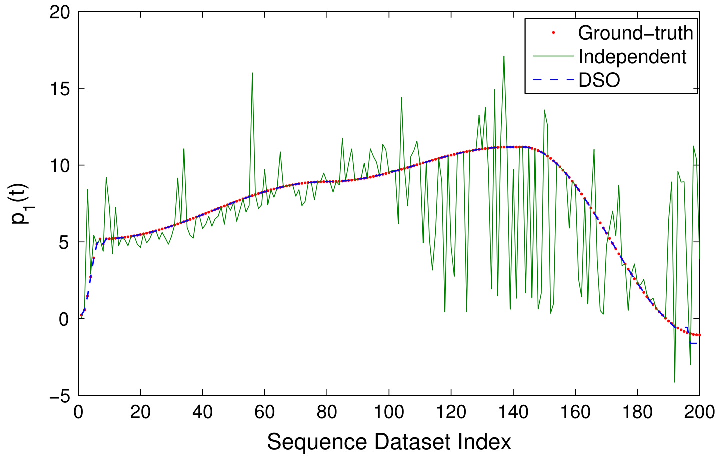
Audio signal decomposition is a thoroughly researched problem, used for a wide range of applications. The idea is to estimate various audio sources from a single mixture signal in order to provide a more accurate starting point for other audio signal processing algorithms. Current methods for audio source separation include Time–frequency Masking [54], Independent Component Analysis [21], Matching Pursuit [18] and other methods for automatic music transcription [7,46].

We propose a forward approach to the decomposition and transcription problems using DSO. To provide comparable results, this example focuses on single-channel (monaural) signals consisting of a mixture of synthetic instruments that do not include speech. However, the algorithm can be used with arbitrary models, including multiple channels (e.g. stereo) and other instruments.

Fig. 2 compares the accuracy of DSO vs. Non-negative Matrix Factorization (NMF) for blind audio source separation [40] with three independent instruments on a short 8-bit ("chiptune") audio signal [20]. In the figure, the top row shows the complete signal (linear combination of all the instruments), whereas the bottom three rows show the signals for each instrument. Fig. 2(a) (left

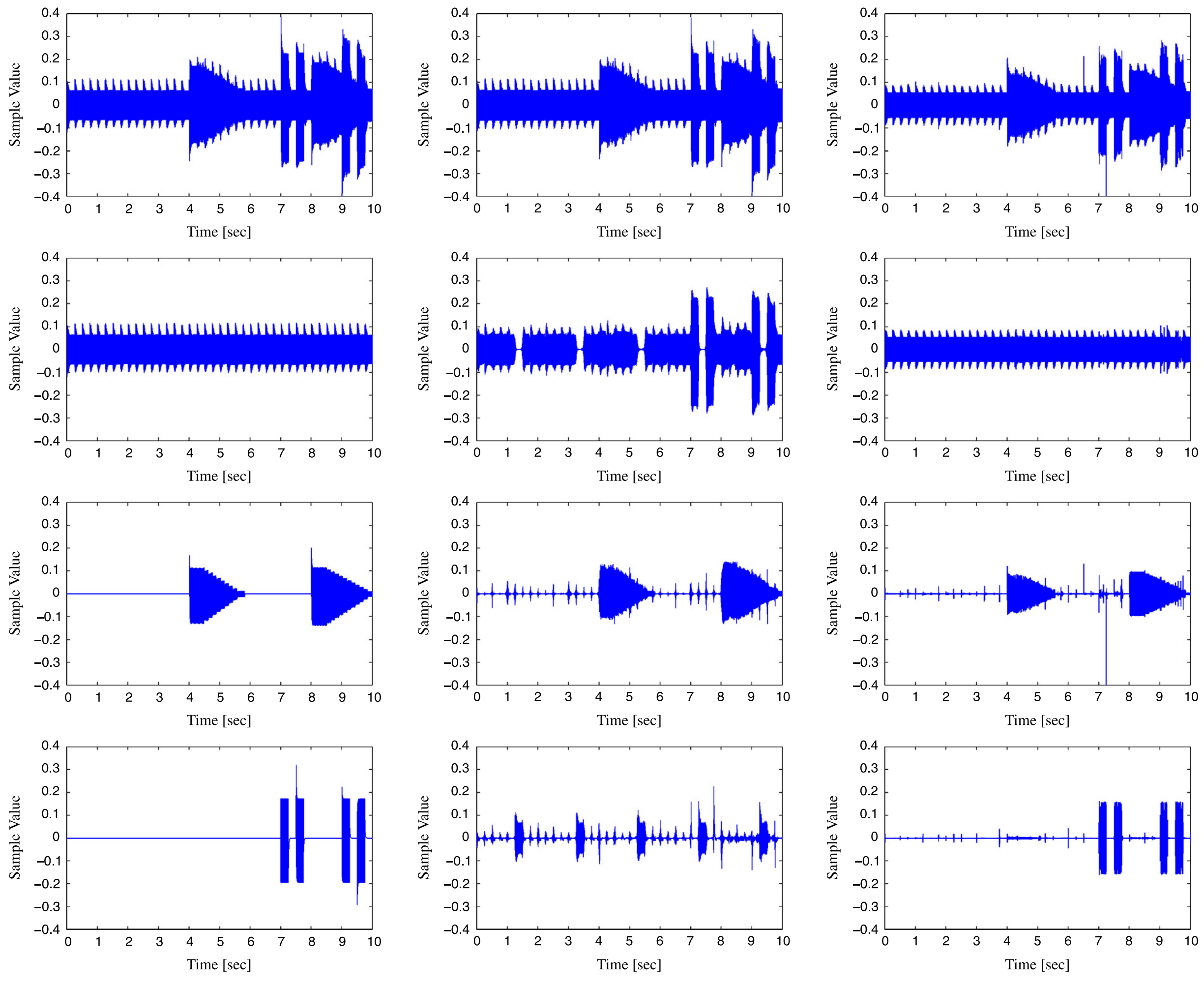


(a) Discrete sequence optimization.



(b) Dataset sequence fitting.

Fig. 1. Comparison between independent optimization and DSO.



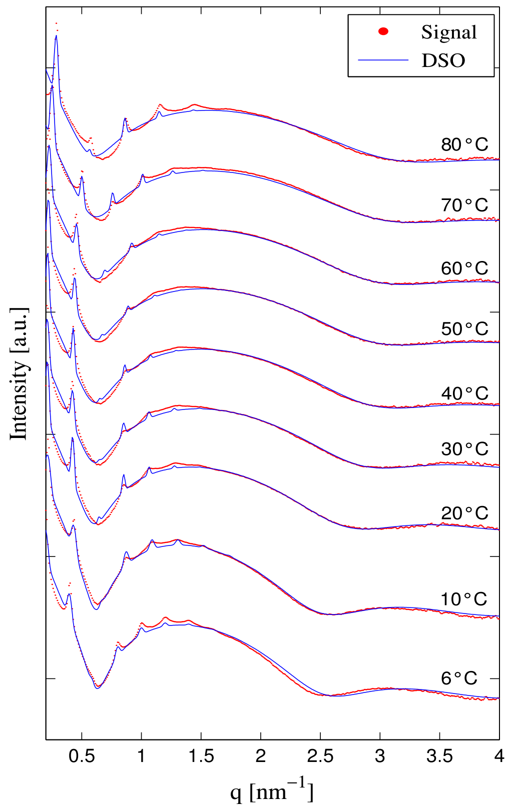
(a) Ground-truth tracks. (b) NMF results. (c) DSO results.

Fig. 2. Comparison of audio signal decomposition methods.

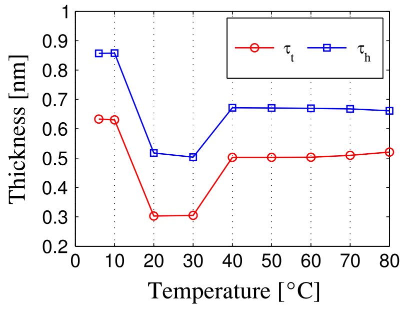
column) presents the original waveform and its ground-truth tracks; Fig. 2(b) (center column) depicts the results of NMF; and Fig. 2(c) (right column) shows the results of DSO. Observe that the results of DSO (right column) closely match their respective ground-truth tracks (left column). For an in-depth analysis of the results, refer to Appendix C.

4.3. *Temperature-resolved X-ray scattering analysis*

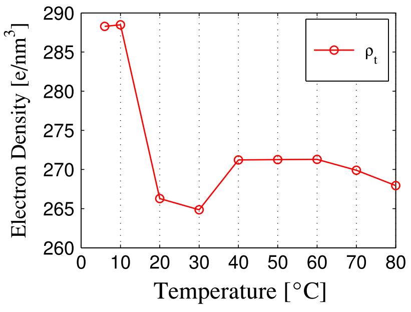
X-ray scattering from complex fluids is used to measure objects in length scales ranging between 0.1 and 100 nm. Specifically, solution Small Angle X-ray Scattering (SAXS) is valuable for researching macro-molecular self-assembled structures [49]. Due



(a) DSO results.



(b) Parameters τ_t and τ_h.



(c) Parameter ρ_t.

Fig. 3. Temperature-resolved solution X-ray scattering analysis using DSO. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to advances in X-ray scattering data acquisition, researchers are now able to study the ways in which structures are assembled, disassembled and change over time.

In temperature-resolved measurements, the parameters that model each component are initially unknown and may change with temperature, while the results are likely to show continuous, physically sound dynamics.

Fig. 3 shows the results of applying the DSO algorithm on SAXS data for temperatures varying between 6 and 80 °C. In Fig. 3(a), the dotted (red) curves depict the obtained signals, whereas the solid (blue) curves show the fitted models. These results, which accurately describe the underlying model, could not be trivially achieved without DSO.

As a result of the optimization process, the variation of the obtained parameter values with temperature is presented in Fig. 3(b) and (c). The results indicate two interesting points, in which the parameters change drastically, identified at 10–20 °C and 30–40 °C. Observe that these critical points cannot be trivially distinguished in the original signals. In reality, the first critical point corresponds to a phase transition that occurs when the measured membrane undergoes a change from a gel (at lower temperatures) to liquid phase. Note that measurements such as these currently have to be analyzed manually, and typically include hundreds to thousands of datasets.

4.4. Video tracking of arbitrary objects

Video tracking algorithms in computer vision follow moving objects, usually obtained from a stationary camera. This problem has many applications ranging from surveillance, through video editing, to behavioral analysis. Current algorithms, such as Particle Filters [43] and Mean-shift Tracking [13], use the sequential properties of the video frames to estimate object motion. One disadvantage of such methods is their use of the two-dimensional shapes as they appear in the frame, instead of optimizing the three-dimensional projections on the image plane. This causes the traditional methods to be less effective when several objects overlap, or when some of them move too fast. Furthermore, it is not trivially possible to parameterize the tracked objects (e.g., determining the position and angle of a human hand) without using additional heuristics.

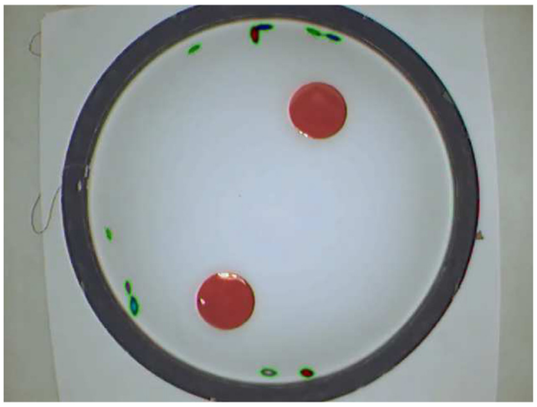
The proposed method uses DSO to minimize the difference between background-subtracted frames (see Appendix E) and all tracked objects. The object projections on the frame are formulated as a sum of components, where each component represents a single object, its location and parameters.

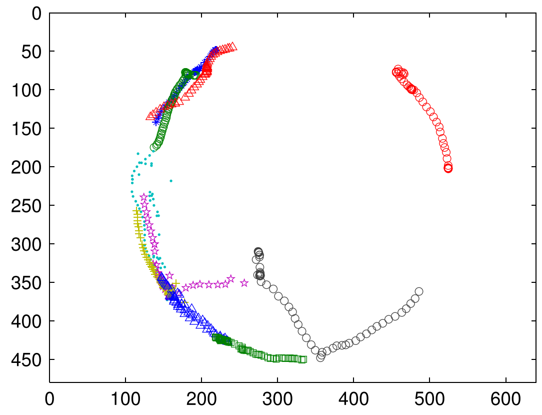
This method tracks objects of arbitrary shapes, which can either be defined by two-dimensional or three-dimensional models, and compounds thereof, enabling the choice of any desired free-form shape.

The results of DSO for tracking bugs in a video captured with a static camera, obtained from the LEURRE project [28,31], are shown in Fig. 4. In Fig. 4(a), the overlay of the results on the original frame (green aura) shows that DSO successfully detects all objects, their location, size and angle. The functions used to model the bugs and their motion constraints are described in Appendix E. Fig. 4(b) presents the obtained parameters in form of a motion graph, depicting the locations of all objects in a segment of the video.

5. Performance evaluation

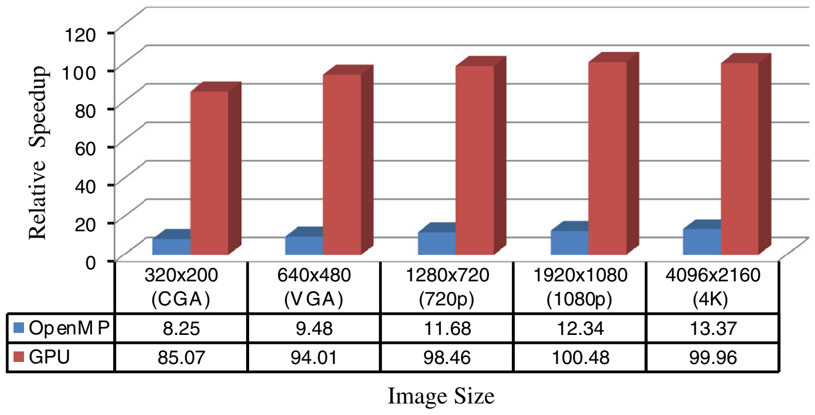
This section analyzes various performance and convergence properties of the DSO algorithm by evaluating the different case studies from Section 4.


(a) Overlay of DSO results on video frame.



(b) Resulting motion graph for objects.

Fig. 4. Video tracking of arbitrary objects using DSO. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



	320x200 (CGA)	640x480 (VGA)	1280x720 (720p)	1920x1080 (1080p)	4096x2160 (4K)
🟦 OpenMP	8.25	9.48	11.68	12.34	13.37
🟥 GPU	85.07	94.01	98.46	100.48	99.96

Fig. 5. Data parallelism performance of video tracking. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The experimental setup used for the tests is composed of a Intel Xeon E5-2650 CPU, with a clock rate of 2.60 GHz and 16 cores; an NVIDIA GeForce GTX TITAN Black GPU, with a clock rate of 0.98 GHz and 2880 cores; and 128 GiB of RAM.

The first two evaluated properties, data and task parallelism, demonstrate the efficiency of DSO on a single node by showing that it can fully utilize a multi-core node with multiple GPUs. The third property, memory complexity, shows that the memory use of DSO is nearly independent of the length of the sequence, allowing optimization of arbitrarily long sequences in limited-memory environments. The fourth measured property is the convergence rate of DSO and its distributed variants, demonstrating that using multiple nodes can speed up local convergence and improve global convergence.

5.1. *Data parallelism*

In this test, we evaluate the objective functions (f_t) of the video tracking case study from Section 4.4 on different architectures. We use typical image resolutions, measuring only the evaluation time. This includes computation of the model function for each pixel, and computing the sum of squared residuals from the background-subtracted video frame. This reduction-based parallel algorithm was tested on both GPU and multi-core CPU versions, implemented over CUDA and OpenMP respectively. During the runtime of DSO, this process is used both for function evaluation and derivative computation, the two major building blocks in nonlinear optimization of a single function.

In Fig. 5, we see that the relative speedup of both the multi-core (blue bars) and GPU (red bars) versions increases with the size of the image. Specifically, for the multi-core version, where the upper limit is 16x performance, the speedup reaches a close 13.37x speedup. The GPU, however, maintains a relatively constant gain, averaging at approximately 95.6x.

Note that using the GPU for evaluation has the additional advantage of freeing the CPU to run other computations (e.g., linear solvers), while the evaluation is offloaded to the GPU. Furthermore, multiple GPUs can be used on the same node in order to evaluate several estimates in parallel.

5.2. *Task parallelism*

In this test, we run the benchmark optimization example from Section 4.1 on a varying number of CPU cores. Fig. 6 demonstrates the task parallelism capabilities of DSO, running with different number of source points per iteration (see Section 3.2). The underlying algorithm used for nonlinear optimization of each function is L-BFGS.

Task parallelism manifests in DSO in several places. First, each source point propagates independently from the other source points. Second, for each source point, the propagation process continues in two separate directions. Third, during propagation of a function to its neighbor, the various extrapolation heuristics are performed concurrently.

Using multiple source points increases the probability of global convergence by evaluating several estimates in parallel,

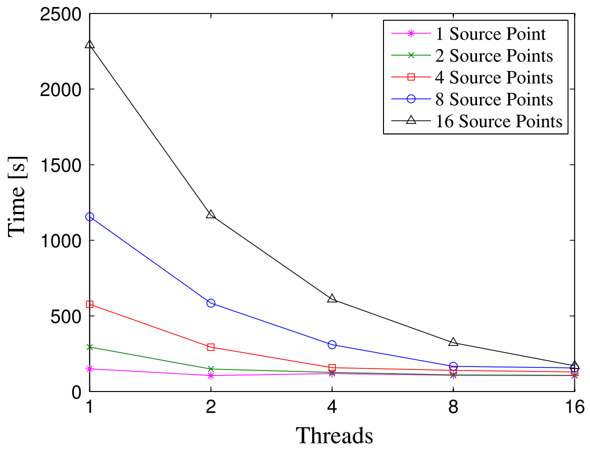


Fig. 6. Task parallelism capabilities of DSO.

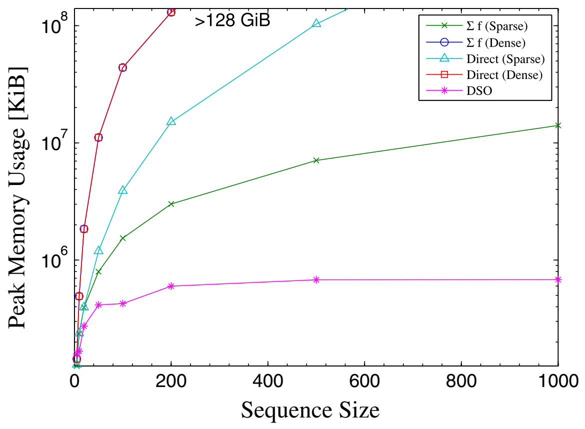


Fig. 7. Memory usage of DSO.

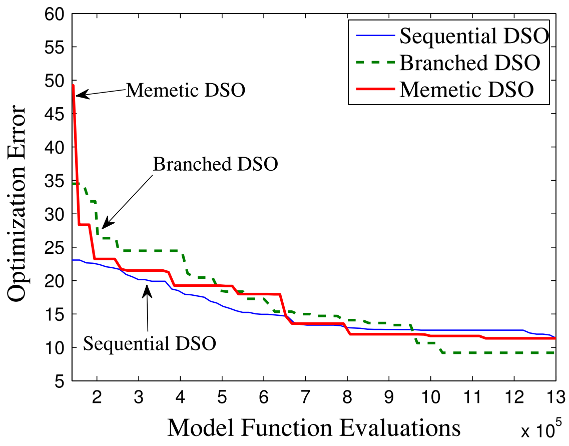


Fig. 8. Convergence of DSO and its distributed variants. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

propagating from different points in the sequence. However, there is a trade-off: choosing too few source points results in the CPU not being fully utilized, whereas too many causes redundant computations, due to the propagation process starting from close points. Therefore, the number of source points should be adapted to the number of available cores as well as to the length of the sequence.

From the figure, it can be seen that even with a single source point, there is a certain degree of parallelism. This is due to the aforementioned inherent concurrency of the algorithm. Furthermore, as the number of source points increases, the algorithm becomes more scalable. For instance, with a single source point, the speedup of 16 threads over 1 is ∼1.43x, whereas with 16 source points it is ∼13.46x.

5.3. Memory complexity

The DSO algorithm optimizes a subset of the sequence at a time, thereby retaining a constant memory complexity with respect to the length of the sequence. The following test demonstrates this property of DSO on the audio signal decomposition case study (see Section 4.2) by measuring the peak memory usage of the application with different sequence sizes and comparing it with traditional nonlinear optimization methods. The evaluated sequence contains fixed-length segments of 100 ms, using the Levenberg–Marquardt algorithm for nonlinear optimization of each function.

Fig. 7 compares the memory usage (in kilobytes) of DSO with nonlinear optimization of all functions ($\sum f$), and direct optimization of g from Eq. (2), using dense and sparse Jacobian matrices for the underlying linear solvers. From the figure, it can be seen that both dense methods (g and $\sum f$) consume approximately the same amount of memory, which grows quadratically with the size of the sequence. This means that most of the memory usage originates in the derivative matrices. Additionally, we can see that the sparse version of $\sum f$ optimization grows linearly with the size of the sequence, whereas the sparse version of direct optimization grows super-linearly. Furthermore, both the dense versions of traditional optimization, as well as the sparse version of g, passed the maximal amount of resident memory of 128 GiB. The DSO version, however, exhibits lower consumption that scales well with the size of the sequence. This is due to the size-independent characteristic of the propagation process.

It is worth noting that the figure shows the memory usage of DSO slightly increasing with the length of the sequence. This is due to the memory requirements for storing the parameter matrix and audio sequence on the RAM. In limited-memory systems, consumption can be dramatically reduced by fetching the audio segments directly from files.

5.4. Distributed variants

In this test, we measure and analyze the convergence of the DSO algorithm and its distributed counterparts (see Section 3.5). The algorithms were run on a simulated distributed environment, where each node is a sub-process that communicates with the others using IPC mechanisms (e.g., shared memory). Running the distributed variants on a cluster would produce similar results and exhibit the same convergence properties.

To implement genetic and memetic algorithms, the Dolphin Memetic Algorithm Package [36,37] was used. To optimize each of the functions in the sequence, we use the Levenberg–Marquardt method.

Fig. 8 depicts the convergence of the DSO algorithm and the distributed variants for the X-ray scattering analysis case study, described in Section 4.3. The sequential algorithm (solid blue line) is compared with the two distributed variants: Branched DSO (dashed green line) and Memetic DSO (solid red line).

The parameters used for Branched DSO are $P = 4, M = 10$, and $L = 2$. As for Memetic DSO, a population of size 50 is used, preserving 5 of the best candidates (elitists) with each generation [5]. Additionally, the Memetic Algorithm utilizes Lamarckian learning [26] for candidate refinement.

From the figure, it can be seen that the sequential DSO is relatively efficient during the first few iterations, but later converges to a local minimum. The distributed versions, however, keep decreasing and converge to better minima. Furthermore, due to the size of the population, Memetic DSO reaches lower sequence scalarization values with fewer evaluations, but as the generations evolve, the population converges to several local minima, creating local "clusters", as opposed to Branched DSO, which is slower at first, but continues to improve throughout the optimization process.

6. Related work

Recent developments in parallel and distributed optimization are divided into two categories: convex optimization [8,42] and general nonlinear optimization [12,53,10]. While DSO generally belongs to the latter, its concurrency is a result of the multitude of functions, rather than the underlying optimization process.

Another important aspect of DSO is its memory complexity. In general, large-scale nonlinear optimization methods are known to not be scalable with respect to memory consumption. A few solutions have been suggested, including sacrificing accuracy for memory usage reduction [9]. Another solution [51] constructs specialized data structures instead of the derivative matrices when information about the specific problem is known.

The DSO algorithm is closely related to Multi-objective Optimization [33] (MO) in the sense that multiple functions must be optimized simultaneously. However, in MO the functions do not have to be sequentially related. On the contrary, MO often optimizes functions that contradict each other, forming trade-offs (e.g., risk vs. profit).

Online parameter estimation methods, such as the Kalman [22] and Particle filters [11] (Sequential Monte-Carlo method), have been applied to sequences of points (specifically, time series) to estimate and predict parameters based on past values with potential noise. The underlying process is assumed to be a linear dynamical system. Extended Kalman Filters [19] (EKF) generalize this solution for nonlinear dynamical systems. However, none of the existing methods provide a generalized solution for offline optimization of nonlinear dynamical systems. It is worth noting that Particle Filters have also been used for video tracking [43].

In contrast to the online "predict-correct" scheme of EKF, offline sequence optimization can be utilized to gain more precise information on the underlying dynamical system. This can be attributed to both asymmetric extrapolation, which can be used to skip noisy and incorrect values; and the generation of additional estimates via multiple passes on the sequence.

Due to their continuous characteristics, splines have been traditionally used for time series modeling and smoothing [15,30]. The DSO algorithm presented in the paper uses this model as the basis for all extrapolation methods. In practice, the tested real-world applications showed that the resulting estimates were actually close to the optimal solutions. Due to the ill-posed nature of most models, there is a heavy dependence between initial estimates and the global optimality of the results. Therefore, optimizing the same objective function (Eq. (2)) directly did not achieve similar results.

As for inference of critical points from the resulting dynamical systems, automatic time-series segmentation [24,47,4] can be performed. Hidden Markov Models [6] provide a general infrastructure for time-series segmentation, finding critical points and states in latent variables using probabilistic methods.

Often with curve fitting of dataset sequences, a weighted sum of components is fitted to the datasets. Optimizing the parameters and weights of each component simultaneously results in source separation, similar to algorithms such as Independent Component Analysis [21] (ICA). Two of the three real-world applications described in Section 4 determine weights of components, as well as the properties of each component during the optimization process. There are two main differences between DSO and ICA. First, by utilizing sequence continuity, DSO can provide results that are more coherent than ICA, in which the order of the datasets does not matter. The second difference is that DSO assumes a-priori knowledge on the data, which is not required in ICA.

7. Conclusions

This paper introduced a novel, generalized algorithm for optimizing function sequences, called Discrete Sequence Optimization (DSO). The DSO algorithm can simultaneously optimize nonlinear function sequences, relying on the continuous characteristic of the functions to provide results that are equal to or more accurate than traditional independent optimization. To define the algorithm and extend the concept of optimization to sequences, we presented a scalarization function for sequences.

DSO can utilize different per-function optimization methods and run efficiently on parallel and distributed systems. We showed that the algorithm locally converges with respect to the objective function. DSO enforces regularization by its extrapolation heuristics and global constraints, and inherently bounds parameter differences using the sequence scalarization function. Two distributed versions of DSO were presented, the first relies on the global convergence of stochastic optimization and the second on a sub-class of genetic algorithms.

To test the accuracy of the algorithm, we studied two synthetic problems and three real-world problems related to audio signal decomposition, X-ray scattering analysis and video tracking. We showed that the results produced by DSO are at least on par with existing problem-specific solutions, and in some cases even surpass such solutions.

In particular, traditional methods for X-ray scattering analysis require manual dataset fitting, as opposed to our automatically generated results. The efficiency of DSO in this application creates opportunities for massive data analysis of phase diagrams, time-resolved data, or other dataset sequences that vary with parameters in a continuous manner. Such data are obtained when studying the dynamics of structures in chemistry, biology, drug design, drug delivery and soft matter.

The performance of DSO was tested on massively parallel systems (using GPUs), multi-core environments, and simulated distributed environments. The results showed that the algorithm can offload a considerable portion of the computations to heterogeneous architectures to achieve, in some cases, speedup of two orders of magnitude. The results also showed that the algorithm is scalable with respect to the number of processors, and that its memory consumption is independent of the length of the sequence. Furthermore, by distributing the algorithm over multiple nodes, the overall rate of convergence can be increased.

Further research of scalarization functions and extrapolation heuristics (e.g., employing Extended Kalman Filters) may yield better and more coherent results with fewer iterations. Additionally, the close relation to Multi-objective Optimization (MO) can be tested by using standard MO implementations on function sequences. For this case, two objective functions should be used: one that minimizes the function values and another that regularizes parameter vectors along the sequence. The definition of a sufficiently smooth discrete parameter function (Appendix A.1) is also subject to future modifications.

The propagation phase of the DSO algorithm could also be augmented in future research. One possible modification is using backtracking instead of rejecting sub-optimal extrapolations. Additionally, it is not uncommon that the parameter vector $X_{t,*}$ consists of several independent subsets that represent multiple components (e.g., tracked objects). Thus, the parameters can potentially be segmented into disjoint subsets and extrapolated separately.

Acknowledgments

This research was supported by the Ministry of Science and Technology, Israel; the Israel Science Foundation (Grant #1372/13); and the US-Israel Binational Science Foundation (Grant #2009271). The authors wish to thank the anonymous reviewers for their constructive comments and suggestions.

Appendix A. Definitions

A.1. Sequential continuity

Since continuity is not properly defined for a function sequence f_t and a discrete parameter matrix X as defined in Section 2, we define sequential continuity using principles from the definition of continuous functions in calculus.

We require continuous sequences to be sufficiently smooth, i.e., exhibiting a low rate of change and a minimal amount of extrema, which can be caused by noise and inconsistencies. For a continuous twice-differentiable function, this would require low absolute second derivative values. For sequences, we can estimate derivatives using finite difference operators.

Assuming that the sequence represents evenly-spaced points in the continuous dimension, the first-order ($\dot{X}$) and second-order ($\ddot{X}$) difference operators are given by:

$$\dot{X}_t \equiv X_{t,*} - X_{t-1,*}; \tag{A.1}$$

$$\ddot{X}_t \equiv \dot{X}_{t+1} - \dot{X}_t = X_{t+1,*} - 2X_{t,*} + X_{t-1,*}, \tag{A.2}$$

where X_t is row t of matrix X and $1 < t < m$.

Based on the above, in order for X to be sequentially continuous, the values of its columns must be sufficiently smooth, having a minimal sum of absolute second-order differences, $\sum_{t=2}^{m-1} \|\ddot{X}_t\|$.

A.2. Convergent optimization methods

Convergent iterative processes are defined using Lyapunov's theory of dynamical system stability [39] and the global convergence theorem, given by Zangwill [55]. The following definition narrows the nonlinear programming theory to locally convergent deterministic optimization methods over $\mathbb{R}^d$.

Definition. A mapping $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is called a *convergent optimization method* with respect to $f : \mathbb{R}^d \to \mathbb{R}$ (a continuous function) and the solution set $\Gamma \subset \mathbb{R}^d$, if for any initial estimate $x_0 \in \mathbb{R}^d$, the sequence $(x_n)_{n=1}^{\infty}$ defined by $x_{n+1} = \phi(x_n)$ satisfies:

1. $\forall_{m>n} : (x_n \notin \Gamma) \Rightarrow f(x_m) < f(x_n)$;
2. $\forall_{m>n} : (x_n \in \Gamma) \Rightarrow f(x_m) \le f(x_n)$.

The solution set Γ defines the equilibrium points of the iterative process. For locally convergent optimization methods, it is defined by:

$$\Gamma = \left\{x^* \in \mathbb{R}^d : \exists_{\varepsilon>0} : \forall_{x\in\mathbb{R}^d} : \|x - x^*\|_2 < \varepsilon \Rightarrow f(x^*) \le f(x)\right\},$$

the set of local minima on f.

Appendix B. Synthetic discrete sequence optimization

The Ackley function for two dimensions [1] is defined by:

$$f_A(x, y) = 20 + e - 20e^{-0.2\sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos(2\pi x)+\cos(2\pi y)}{2}}.$$

It is used as a benchmark for optimization algorithms due to its multitude of local minima, see Fig. B.9 for a plot of f_A in the range $x, y \in [-5, 5]$.

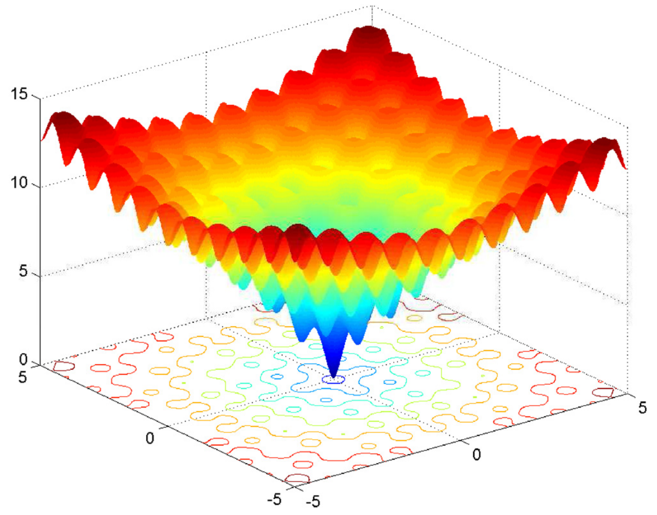


Fig. B.9. The Ackley function (f_A).

Appendix C. Audio signal decomposition and transcription

The optimization process uses the Mean Squared Error (MSE) as the objective function. In the proposed scheme, the single-channel audio signal is first converted to a sequence of short segments (tens of milliseconds in length). The sequence is then fit, using DSO, to a model that comprises the sum of all the instruments played in the signal. This model is defined as follows:

$$F\left(q, \vec{\theta}\right) = \sum_{i=1}^{n_{inst}} I_i\left(q, \lambda_i, v_i, \ell_i, h_i, \vec{ep}_i\right),$$

where n_{inst} denotes the number of instruments, I_i is the instrument model (see below), λ_i is the instrument pitch, v_i the volume, ℓ_i the offset (for maintaining continuity between segments), h_i the vertical offset of the instrument and $\vec{ep}_i$ is a vector containing instrument-specific parameters.

The synthetic instrument models used in the example are sine waves and rectangular (pulse) waves. While the sine model is straightforward, the rectangular wave model is not optimal for fitting because of its discrete nature, which hinders partial derivative computation. The formal definition of a rectangular wave model is:

$$I_{rect}(q, \vec{\theta}) = h + v \cdot \begin{cases} +1 & \text{if } \dfrac{(\lambda q + \ell) \bmod 2\pi}{2\pi} \le r, \\ -1 & \text{otherwise,} \end{cases}$$

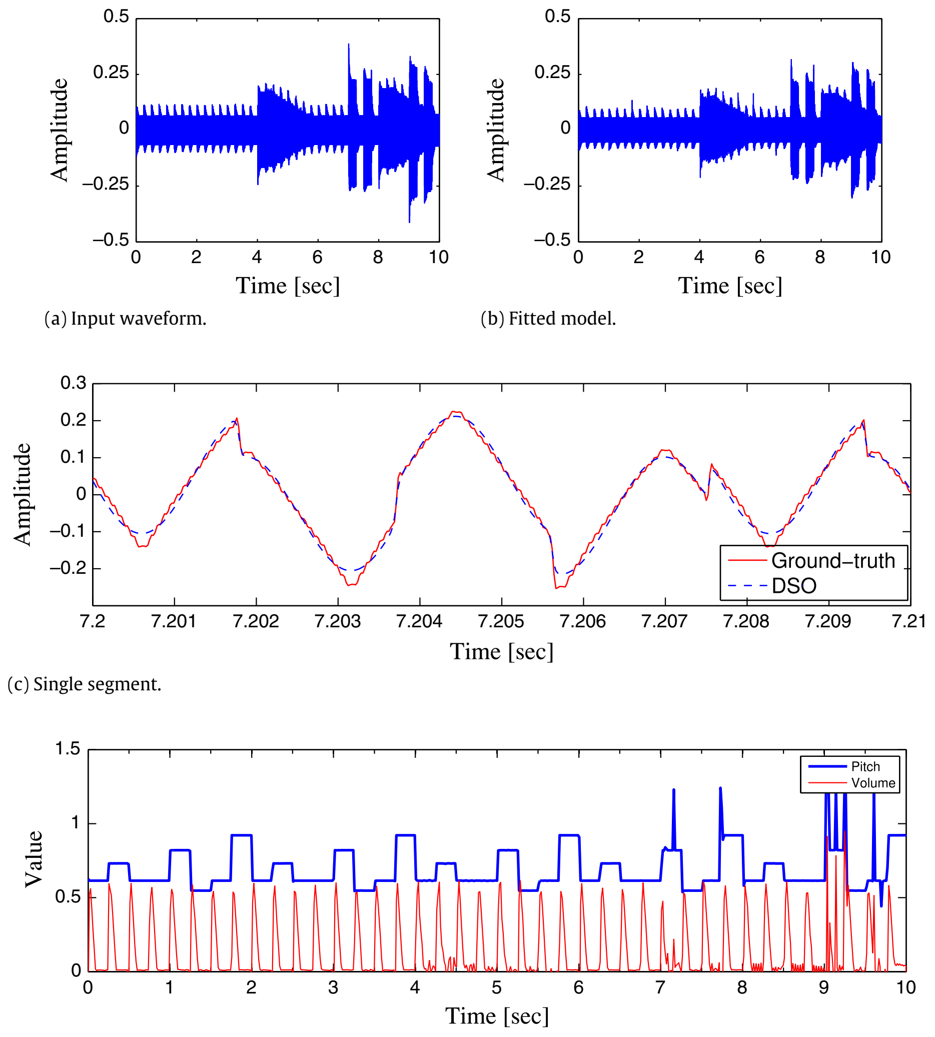
where r defines the ratio between the amount of 1 and -1 values in a period ($r = 0.5$ producing a perfect square wave). Therefore, a continuous equivalent wave model, based on hyperbolic tangents, is defined by:

$$I_{\tanh}(q, \vec{\theta}) = h + \frac{v}{2}\left\{\tanh\left[\mu\left(y + 2r\right)\right] - \tanh\left[\mu\left(y - 2r\right)\right]\right\},$$

where $y \equiv \sin(\lambda q + \ell)$ and μ is a smoothing constant for the hyperbolic tangent function.

The local constraints used in this application are box constraints with reasonable bounds for each parameter, and pitch bounds chosen to limit instruments within their octaves. The global constraints limit the length of each note and bound the difference between the pitch values within consecutive audio segments.

The results of audio signal decomposition and transcription using DSO are shown in Fig. C.10. Fitting was performed on a waveform consisting of 3 synthetic instruments: two square waves and one triangular wave. The fitted model contains two rectangular waves and one sine wave. Fig. C.10(a) depicts the input waveform



(a) Input waveform.

(b) Fitted model.

(c) Single segment.

(d) Pitch graph of a single instrument.

Fig. C.10. Audio signal decomposition and music transcription using DSO. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and Fig. C.10(b) the output of the algorithm. Fig. C.10(c) shows the fitting results of a single segment of 10 ms, containing multiple instruments. Fig. C.10(d) is the pitch graph of one of the resulting square waves, presenting both the pitch (top, blue) and volume (bottom, red) of the instrument, essentially reproducing its notes. Observe that the noisy behavior toward the end of the pitch graph can be automatically detected and corrected by post-processing. Inputting the corrected pitch values to DSO can yield accurate results for the rest of the parameters.

Appendix D. Temperature-resolved X-ray scattering analysis

For any object of volume V, the X-ray scattering amplitude $F\left(\vec{q}\right)$, is defined by the Fourier Transform of its electron density contrast, $\Delta\rho(\vec{r})$, with respect to a given solvent [3]:

$$F\left(\vec{q}\right) = \int_V \Delta\rho\left(\vec{r}\right)\exp\left(i\vec{q}\cdot\vec{r}\right)d\vec{r},$$

where $\vec{q}$ is the scattering vector and $q \equiv \left|\vec{q}\right|$ is expressed in units of nm^{-1}.

The electron density contrast originates in the different molecular composition of the scattering objects with respect to the surrounding solvent. The measured form factor intensity is $\left|F\left(\vec{q}\right)\right|^2$. In solution X-ray scattering, the intensity is averaged over all the possible orientations in q-space. The model function used for X-ray scattering analysis is composed of three components: form factor, structure factor and background. Each of the components define a different aspect of the molecular structure of the object.

In this example, we analyze a measurement of a water-solution of 100 mg/ml 1,2-dilauroyl-sn-glycero-3-phospho-L-serine (DLPS), forming lipid membranes. According to Székely et al. [48], using the Gaussian electron density profile, the model for the form factor is defined by:

$$FF\left(q,\vec{\theta}\right) = \frac{\pi}{(2\ln 2)\,q^2}\left(\Delta\rho_t\tau_t e^{\frac{-q^2\tau_t^2}{16\ln 2}} + 2\Delta\rho_h\tau_h e^{\frac{-q^2\tau_h^2}{16\ln 2}}\right)^2,$$

where τ_t, τ_h are the membrane's tail and head thicknesses respectively and $\Delta\rho_t, \Delta\rho_h$ are the electron density contrasts (e/nm^3), with respect to the electron density of the solvent. The structure factor is modeled as follows:

$$SF\left(q,\vec{\theta}\right) = B_p + N^2A_p\cdot e^{-q^2\langle u\rangle^2/2}\sum_{i=1}^{\left\lfloor\frac{q_{\max}}{2\pi d}\right\rfloor} e^{-\left[Nd\left(q-\frac{2\pi i}{d}\right)\right]^2/4\pi},$$

where N designates the average number of membranes in the multi-layered (lamellar) structure, d the distance between adjacent membranes in the structure, $\langle u\rangle^2$ the mean squared displacement of a membrane (contributing to the Debye–Waller factor [3]), A_p the structure factor amplitude, B_p is an additive coefficient for the structure factor, representing the Fourier transform of the fluctuations of the local density from the average density, and $q_{\max}$ is a constant defining the maximal value of q in the datasets.

The complete model function is defined by:

$$F\left(q,\vec{\theta}\right) = A_m \cdot FF\left(q,\vec{\theta}\right) \cdot SF\left(q,\vec{\theta}\right) + BG_{amp} \cdot q^{-BG_{power}},$$

where an empirical power law is used as the background for the model with BG_{amp} and BG_{power}.

Due to the varying scale of the signal and the importance of features in all scales, the objective function chosen for this application is relative SSE, a variation on the sum of squared errors, defined by:

$$\mathrm{RelSSE}\,(D, M) = \sum_{i=1}^{n} \begin{cases} \left[(D_i - M_i)/D_i\right]^2 & \text{if } D_i \neq 0, \\ (D_i - M_i)^2 & \text{if } D_i = 0, \end{cases} \tag{D.1}$$

where D and M are the data and the model values respectively.

Since the parameter space is large, a rough estimate was given for one signal (at 40 °C) and feasible lower and upper bounds were input to DSO. The estimate and bounds were approximated from prior knowledge of the measured membrane. Implementing the bounds as local box constraints sufficed to obtain the results in this application, in addition to an inequality constraint enforcing $\Delta\rho_t \leq \Delta\rho_h$, as observed in DLPS membranes.

Appendix E. Video tracking of arbitrary objects

In order to fit a model function to a video, it is necessary to subtract the background from the original frames. Since the camera is assumed to be stationary and the background is static, we removed the background by pixel color modeling, using median color subtraction. As a result, the objective function chosen for this application is the MSE of the background-subtracted frame from the generated model image.

Given a video (as a sequence of two-dimensional frames) with moving objects over static background, the model is defined as follows:

$$F\left(q_x, q_y, \vec{\theta}\right) = \sum_{i=1}^{n_{obj}} I_i\left(q_x, q_y, \ell_i^x, \ell_i^y, \overrightarrow{op}_i\right),$$

where n_{obj} is the number of objects in the frame, I_i is the model function for each object, $\left(\ell_i^x, \ell_i^y\right)$ is the (x, y) coordinates of each object and $\overrightarrow{op}_i$ contains object-specific parameters.

The model chosen to represent the tracked objects (bugs in the video) is an elliptic two-dimensional Gaussian, defined by:

$$I_{gauss}\left(x, y, \vec{\theta}\right) = s \cdot e^{-a(x-\ell_x)^2 - 2b(x-\ell_x)(y-\ell_y) - c(y-\ell_y)^2},$$

where α is the object orientation, w and h are the width and height of the object, s is its intensity, $a \equiv \frac{\cos^2\alpha}{2w} + \frac{\sin^2\alpha}{2h}$, $b \equiv -\frac{\sin 2\alpha}{4w} + \frac{\sin 2\alpha}{4h}$, and $c \equiv \frac{\sin^2\alpha}{2w} + \frac{\cos^2\alpha}{2h}$.

The initial estimates in this application are the x and y coordinates of the objects in the first frame, allowing the width, height and angle parameters to be automatically optimized.

As for local constraints, box constraints were defined for all the parameters. Since the direction of the object can be inferred at a later stage, the object orientation α is limited to the range of $[0, \pi)$. Another constraint maintains the inequality $h \geq w$.

The global constraints in this application are used to enforce the logical movement of the objects. In the definition of the constraints, two parameters were defined: the maximal distance that an object can move between two frames, and the maximal degrees that an object can rotate between two frames. These two parameters form a perimeter around the object, as illustrated in Fig. E.11.

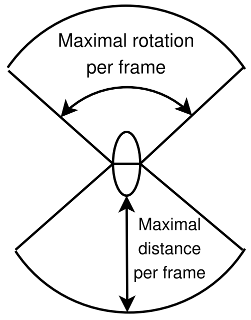


Fig. E.11. Global constraints for video tracking with DSO.

References

[1] D.H. Ackley, A connectionist machine for genetic hillclimbing, in: Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, 1987.
[2] S. Agarwal, K. Mierle, et al. Ceres solver. URL: http://ceres-solver.org.
[3] J. Als-Nielsen, D. McMorrow, Elements of Modern X-ray Physics, J. Wiley & Sons, 2001.
[4] U. Appel, A.V. Brandt, Adaptive sequential segmentation of piecewise stationary time series, Inform. Sci. 29 (1) (1983) 27–56. http://dx.doi.org/10.1016/0020-0255(83)90008-7. Institute of Electrical and Electronics Engineers Workshop "Applied Time Series Analysis".
[5] S. Baluja, R. Caruana, Removing the genetics from the standard genetic algorithm, in: The Proceedings of the 12th Annual Conference on Machine Learning, 1995, pp. 38–46.
[6] L.E. Baum, T. Petrie, Statistical inference for probabilistic functions of finite state Markov chains, Ann. Math. Statist. 37 (6) (1966) 1554–1563. http://dx.doi.org/10.2307/2238772.
[7] J.P. Bello, G. Monti, M. Sandler, Techniques for automatic music transcription, in: International Symposium on Music Information Retrieval, 2000, pp. 23–25.
[8] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Mach. Learn. 3 (1) (2011) 1–122.
[9] R. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Comput. 16 (5) (1995) 1190–1208. http://dx.doi.org/10.1137/0916069.
[10] J. Cao, K.A. Novstrup, A. Goyal, S.P. Midkiff, J.M. Caruthers, A parallel Levenberg–Marquardt algorithm, in: Proceedings of the 23rd International Conference on Supercomputing, ICS'09, ACM, New York, NY, USA, 2009, pp. 450–459. http://dx.doi.org/10.1145/1542275.1542338.
[11] O. Cappe, S.J. Godsill, E. Moulines, An overview of existing methods and recent advances in sequential Monte Carlo, Proc. IEEE 95 (5) (2007) 899–924. http://dx.doi.org/10.1109/JPROC.2007.893250.
[12] Y. Censor, S.A. Zenios, Parallel Optimization: Theory, Algorithms and Applications, Oxford University Press, 1997.
[13] D. Comaniciu, V. Ramesh, P. Meer, Real-time tracking of non-rigid objects using mean shift, in: Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on, Vol. 2, 2000, pp. 142–149, http://dx.doi.org/10.1109/CVPR.2000.854761.
[14] NVIDIA CUDA SDK, 2015. URL: http://www.nvidia.com/cuda/.
[15] J. Fan, Q. Yao, Nonlinear Time Series: Nonparametric and Parametric Methods, Springer, New York, 2003, http://dx.doi.org/10.1007/b97702.
[16] C. Fox, An Introduction to the Calculus of Variations, in: Dover Books on Mathematics Series, Dover Publications, Incorporated, 1950.
[17] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Pub., 1989.
[18] R. Gribonval, E. Bacry, Harmonic decomposition of audio signals with matching pursuit, IEEE Trans. Signal Process. 51 (1) (2003) 101–111. http://dx.doi.org/10.1109/TSP.2002.806592.
[19] S. Haykin, Kalman Filtering and Neural Networks, John Wiley & Sons, Inc., 2002, http://dx.doi.org/10.1002/0471221546.
[20] S. Hulick, Uncharted Worlds, Mass Effect Original Soundtrack, November 2007.
[21] A. Hyvärinen, E. Oja, Independent component analysis: algorithms and applications, Neural Netw. 13 (4–5) (2000) 411–430. http://dx.doi.org/10.1016/S0893-6080(00)00026-5.
[22] R.E. Kalman, A new approach to linear filtering and prediction problems, Trans. ASME Ser. D 82 (1960) 35–45.
[23] H. Kawamura, Dynamical simulation of spin-glass and chiral-glass orderings in three-dimensional heisenberg spin glasses, Phys. Rev. Lett. 80 (1998) 5421–5424. http://dx.doi.org/10.1103/PhysRevLett.80.5421.
[24] E. Keogh, S. Chu, D. Hart, M. Pazzani, An online algorithm for segmenting time series, in: Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, 2001, pp. 289–296. http://dx.doi.org/10.1109/ICDM.2001.989531.
[25] O.A. Ladyzhenskaya, A dynamical system generated by the Navier–Stokes equations, J. Soviet Math. 3 (4) (1975) 458–479. http://dx.doi.org/10.1007/BF01084684.

[26] M.N. Le, Y.S. Ong, Y. Jin, B. Sendhoff, Lamarckian memetic algorithms: local optimum and connectivity structure analysis, Memetic Comput. 1 (3) (2009) 175–190. http://dx.doi.org/10.1007/s12293-009-0016-9.
[27] C. Lemaréchal, Lagrangian relaxation, in: M. Jünger, D. Naddef (Eds.), Computational Combinatorial Optimization, in: Lecture Notes in Computer Science, vol. 2241, Springer Berlin, Heidelberg, 2001, pp. 112–156. http://dx.doi.org/10.1007/3-540-45586-8_4.
[28] The LEURRE Project, ULB, 2005. URL: http://leurre.ulb.ac.be/.
[29] K. Levenberg, A method for the solution of certain non-linear problems in least squares, Quart. Appl. Math. 2 (2) (1944) 164–168.
[30] P.A.W. Lewis, J.G. Stevens, Nonlinear modeling of time series using multivariate adaptive regression splines (mars), J. Amer. Statist. Assoc. 86 (416) (1991) 864–877.
[31] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, A. Martinoli, Swistrack—a flexible open source tracking software for multi-agent systems, in: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, 2008, 4004–4010. http://dx.doi.org/10.1109/IROS.2008.4650937.
[32] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, SIAM J. Appl. Math. 11 (2) (1963) 431–441.
[33] K. Miettinen, Nonlinear Multiobjective Optimization, in: International Series in Operations Research and Management Science, vol. 12, Kluwer Academic Publishers, Dordrecht, 1999.
[34] J.A. Miller, W.D. Potter, R.V. Gandham, C.N. Lapena, An evaluation of local improvement operators for genetic algorithms, IEEE Trans. Syst. Man Cybern. 23 (5) (1993) 1340–1351.
[35] A.H. Mohamed, K.P. Schwarz, Adaptive kalman filtering for INS/GPS, J. Geod. 73 (4) (1999) 193–203. http://dx.doi.org/10.1007/s001900050236.
[36] Y.S. Ong, A.J. Keane, Meta-lamarckian learning in memetic algorithms, IEEE Trans. Evol. Comput. 8 (2) (2004) 99–110. http://dx.doi.org/10.1109/TEVC.2003.819944.
[37] Y.S. Ong, M.H. Lim, N. Zhu, K.W. Wong, Classification of adaptive memetic algorithms: a comparative study, IEEE Trans. Syst. Man Cybern. Part B 36 (1) (2006) 141–152. http://dx.doi.org/10.1109/TSMCB.2005.856143.
[38] OpenMP Architecture Review Board, OpenMP application program interface version 3.0, May 2008. URL: http://www.openmp.org/mp-documents/spec30.pdf.
[39] J.M. Ortega, Stability of difference equations and convergence of iterative processes, SIAM J. Numer. Anal. 10 (2) (1973) 268–282.
[40] A. Ozerov, E. Vincent, F. Bimbot, A general flexible framework for the handling of prior information in audio source separation, IEEE Trans. Audio Speech Lang. Process. 20 (4) (2012) 1118–1133.
[41] S. Pelet, M.J.R. Previte, L.H. Laiho, P.T.C. So, A fast global fitting algorithm for fluorescence lifetime imaging microscopy based on image segmentation, J. Biophys. 84 (4) (2004) 2807–2817. http://dx.doi.org/10.1529/biophysj.104.045492.
[42] Z. Peng, M. Yan, W. Yin, Parallel and distributed sparse optimization, in: Signals, Systems and Computers, 2013 Asilomar Conference on, 2013, pp. 659–646. http://dx.doi.org/10.1109/ACSSC.2013.6810364.
[43] P. Pérez, C. Hue, J. Vermaak, M. Gangnet, Color-based probabilistic tracking, in: In Proc. ECCV, 2002, pp. 661–675.
[44] P. Phua, D. Ming, W. Fan, Y. Zhang, Parallel algorithms for solving large-scale nonlinear optimization problems, in: X. Yang, K. Teo, L. Caccetta (Eds.), Optimization Methods and Applications, in: Applied Optimization, vol. 52, Springer US, 2001, pp. 279–293. http://dx.doi.org/10.1007/978-1-4757-3333-4_15.
[45] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, in: A. Waibel, K. Lee (Eds.), Readings in Speech Recognition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990, pp. 267–296.
[46] M.P. Ryynänen, A.P. Klapuri, Automatic transcription of melody, bass line, and chords in polyphonic music, Comput. Music J. 32 (3) (2008) 72–86. http://dx.doi.org/10.1162/comj.2008.32.3.72.
[47] S.L. Sclove, Time-series segmentation: A model and a method, Inform. Sci. 29 (1) (1983) 7–25. http://dx.doi.org/10.1016/0020-0255(83)90007-5. Institute of Electrical and Electronics Engineers Workshop "Applied Time Series Analysis".
[48] P. Székely, A. Ginsburg, T. Ben-Nun, U. Raviv, Solution X-ray scattering form factors of supramolecular self-assembled structures, Langmuir 26 (16) (2010) 13110–13129. http://dx.doi.org/10.1021/la101433t.
[49] O. Székely, A. Steiner, P. Székely, E. Amit, R. Asor, C. Tamburu, U. Raviv, The structure of ions and zwitterionic lipids regulates the charge of dipolar membranes, Langmuir 27 (12) (2011) 7419–7438.
[50] A.N. Tikhonov, Numerical Methods for the Solution of Ill-posed Problems, Vol. 328, Springer, 1995.
[51] B. Triggs, P.F. McLauchlan, R.I. Hartley, A.W. Fitzgibbon, Bundle adjustment—a modern synthesis, in: Proceedings of the International Workshop on Vision Algorithms: Theory and Practice, ICCV'99, Springer-Verlag, London, UK, UK, 2000, pp. 298–372.
[52] R.S. Tsay, Analysis of Financial Time Series, second ed., in: Series in Probability and Statistics, John Wiley & Sons, 2005.
[53] Y. Xu, Y. Chen, A framework for parallel nonlinear optimization by partitioning localized constraints, in: Proc. International Symposium on Parallel Architectures, Algorithms and Programming, 2008.
[54] O. Yilmaz, S. Rickard, Blind separation of speech mixtures via time–frequency masking, IEEE Trans. Signal Process. 52 (7) (2004) 1830–1847. http://dx.doi.org/10.1109/TSP.2004.828896.
[55] W.I. Zangwill, Nonlinear Programming: A Unified Approach, in: Prentice-Hall International Series in Management, Prentice-Hall, 1969.

Tal Ben-Nun is a Ph.D. candidate in the Department of Computer Science at the Hebrew University of Jerusalem. He received his B.Sc. and M.Sc. at the Hebrew University of Jerusalem. His research interests are in parallel and distributed algorithms, nonlinear optimization, and massively parallel architectures.

Amnon Barak is the director of the Distributed Systems laboratory in the Computer Science Department at Hebrew University. He received a Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign. His research interests are in parallel and distributed systems.

Uri Raviv completed his Ph.D. at the Weizmann Institute of Science in Rehovot in the laboratory of Jacob Klein in 2002, and then moved to the University of California at Santa Barbara to work in the laboratory of Cyrus Safinya as a Human Frontier Science Program (HFSP) postdoctoral fellow. At 2006, he was appointed as a senior lecturer in the Institute of Chemistry at the Hebrew University of Jerusalem and at 2013 was promoted to an associate professor. His research interests are in soft matter, self-assembly of biomolecules, intermolecular forces, and solution x-ray scattering.