Real-Time Convex Optimization in Signal Processing

Jacob Mattingley and Stephen Boyd
Information Systems Laboratory, Electrical Engineering Department, Stanford University

Working draft.
Comments to `jacobm` or `boyd@stanford.edu` are welcome.

***Abstract*—Convex optimization has been used in signal processing for a long time, to choose coefficients for use in fast (linear) algorithms, such as in filter or array design; more recently, it has been used to carry out (nonlinear) processing on the signal itself. Examples of the latter case include total variation de-noising, compressed sensing, fault detection, and image classification. In both scenarios, the optimization is carried out on time scales of seconds or minutes, and without strict time constraints. Convex optimization has traditionally been considered computationally expensive, so its use has been limited to applications where plenty of time is available. Such restrictions are no longer justified. The combination of dramatically increased computational power, modern algorithms, and new coding approaches has delivered an enormous speed increase, which makes it possible to solve modest-sized convex optimization problems on microsecond or millisecond time scales, and with strict deadlines. This enables real-time convex optimization in signal processing.**

I. Introduction

Convex optimization [1] refers to a broad class of optimization problems, which includes, for example, least-squares, linear programming (LP), quadratic programming (QP), and the more modern second-order cone programming (SOCP), semidefinite progamming (SDP), and the ℓ_1 minimization at the core of compressed sensing [2], [3]. Unlike many generic optimization problems, convex optimization problems can be efficiently solved, both in theory (*i.e.*, via algorithms with worst-case polynomial complexity) [4] and in practice [1], [5]. It is widely used in application areas like control [6], [7], [8], circuit design [9], [10], [11], economics and finance [12], [13], networking [14], [15], [16], statistics and machine learning [17], [18], quantum information theory [19], [20], and combinatorial optimization [21], to name just a few.

Convex optimization has a long history in signal processing, dating back to the 1960s. The history is described below in a little more detail; for some more recent applications, see for example the special issue of the *IEEE Journal on Selected Topics in Signal Processing* on convex optimization methods for signal processing [22].

Signal processing applications may be split into two categories. In the first, optimization is used for design, *i.e.*, to choose the weights or algorithm parameters for later use in a (typically linear) signal processing algorithm. A classical example is the design of finite impulse response (FIR) filter coefficients via linear programming (LP) [23], [24]. In these design applications, the optimization must merely be fast enough to not slow the designer; thus, optimization times measured in seconds, or even minutes, are usually sufficient. In the second category, convex optimization is used to process the signal itself, which (generally) yields a nonlinear algorithm; an early example is ℓ_1 regularization for sparse reconstruction in geophysics [25], [26]. Most applications in this category are (currently) off-line, as in geophysics reconstruction, so while faster is better, the optimization is not subject to the strict real-time deadlines that would arise in an on-line application.

Recent advances in algorithms for solving convex optimization problems, along with great advances in processor power, have dramatically reduced solution times. Another significant reduction in solution time may be obtained by using a solver customized for a particular problem family. (This is described in §III.) As a result, convex optimization problems that 20 years ago might have taken minutes to solve can now be solved in microseconds.

This opens up several new possibilities. In the design context, algorithm weights can be re-designed or updated on fast time scales (say, kHz). Perhaps more exciting is the possibility that convex optimization can be emdedded directly in signal processing algorithms that run on-line, with strict real-time deadlines, even at rates of tens of kHz. We will see that solving 10000 modest sized convex optimization problems per second is entirely possible on a generic processor. This is quite remarkable, since solving an optimization problem is generally considered a computationally challenging task, and few engineers would consider an on-line algorithm, that requires the solution of an optimization problem at each step, to be feasible for signal rates measured in kHz.

Of course, for high-throughput or fast signal processing (say, an equalizer running at GHz rates) it is not feasible to solve an optimization problem in each step, and it may never be. But a large number of applications are now potentially within reach of new algorithms in which an optimization problem is solved in each step, or every few steps. We imagine that in the future, more and more signal processing algorithms will involve embedded optimization, running at rates up to or exceeding tens of kHz. (We believe the same trend will take place in automatic control; see, *e.g.*, [27], [28].)

In this article, we briefly describe two recent advances that make it easier to design and implement algorithms for such applications. The first, described in §II, is disciplined convex programming, which simplifies problem specification and allows the transformation to a standard form to be automated. This makes it possible to rapidly prototype applications based on convex optimization. The second advance, described in §III, is convex optimization code generation, in which (source code for) a custom solver that runs at the required high speed is automatically generated from a high level description of the problem family.

In the final three sections, we illustrate the idea of real-time embedded convex optimization with three simple examples. In the first example (§IV), we show how to implement a nonlinear pre-equalizer for a system with input saturation. It pre-distorts the input signal so that the output signal approximately matches the output of a reference linear system. Our equalizer is based on a method called model predictive control [29], which has been widely used in the process control industry for more than a decade. It requires the solution of a QP at each step. It would not surprise anyone to know that such an algorithm could be run at, say, 1 Hz (process control applications typically run with sample times measured in minutes); but we will show that it can easily be run at 1 kHz. This example illustrates how our ability to solve QPs with extreme reliability and speed has made new signal processing methods possible.

In the second example (§V), we show how a standard Kalman filter can be modified to handle occasional large sensor noises (such as those due to sensor failure or intentional jamming), using now standard ℓ_1-based methods. Those familiar with the ideas behind compressed sensing (or several other related techniques) will not be surprised at the effectiveness of these methods, which require the solution of a QP at each time step. What is surprising is that such an algorithm can be run at tens of kHz.

Our final example (§VI) is one from the design category: a standard array signal processing weight selection problem, in which, however, the sensor positions drift with time. Here the problem reduces to the solution of an SOCP at each time step; the surprise is that this can be carried out in a few milliseconds, which means that the weights can be re-optimized at hundreds of Hz.

In the next two subsections we describe some previous and current applications of convex optimization, in the two just-described categories of weight design and direct signal processing. Before preceeding we note that the distinction between the two categories—optimization for algorithm weight design versus optimization directly in the algorithm itself—is not sharp. For example, widely used adaptive signal processing techniques [30], [31] adjust parameters in an algorithm (*i.e.*, carry out re-design) on-line, based on the data or signals themselves.

A. Weight design via convex optimization

Convex optimization was first used in signal processing in design, *i.e.*, selecting weights or coefficients for use in simple, fast, typically linear, signal processing algorithms. In 1969, [23] showed how to use LP to design symmetric linear phase FIR filters. This was later extended to the design of weights for 2-D filters [32], and filter banks [33]. Using spectral factorization, LP and SOCP can be used to design filters with magnitude specifications [34], [35].

Weight design via convex optimization can also be carried out for (some) nonlinear signal processing algorithms; for example, in a decision-feedback equalizer [36]. Convex optimization can also be used to choose the weights in array signal processing, in which multiple sensor outputs are combined linearly to form a composite array output. Here the weights are chosen to give a desirable response pattern [37], [38]. More recently, convex optimization has been used to design array weights that are robust to variations in the signal statistics or array response [39], [40].

Many classification algorithms from machine learning involve what is essentially weight design via convex optimization [41]. For example, objects x (say, images or email messages) might be classified into two groups by first computing a vector of features $\phi(x) \in \mathbf{R}^n$, then, in real-time, using a simple linear threshold to classify the objects: we assign x to one group if $w^T\phi(x) \geq v$, and to the other group if not. Here $w \in \mathbf{R}^n$ and $v \in \mathbf{R}$ are weights, chosen by training from objects whose true classification is known. This off-line training step often involves convex optimization. One widely used method is the support vector machine (SVM), in which the weights are found by solving a large QP [17], [18]. While this involves solving a (possibly large) optimization problem to determine the weights, only minimal computation is required at run-time to compute the features and form the inner product that classifies any given object.

B. Signal processing via convex optimization

Recently introduced applications use convex optimization to carry out (nonlinear) processing of the signal itself. The crucial difference from the previous category is that speed is now of critical importance. Convex optimization problems are now solved in the main loop of the processing algorithm, and the total processing time depends on how fast these problems can be solved.

With some of these applications, processing time again matters only in the sense that ‘faster is better’. These are off-line applications where data is being analyzed without strict time constraints. More challenging applications involve on-line solution, with strict real-time deadlines. Only recently has the last category become possible, with the development of reliable, efficient solvers and the recent increase in computing power.

One of the first applications where convex optimization was used directly on the signal is in geophysics [25], [26], where ℓ_1 minimization was used for sparse reconstruction of signals. Similar ℓ_1-techniques are widely used in total variation noise removal in image processing [42], [43], [44]. Other image processing applications include deblurring [45] and, recently, automatic face recognition [46]. Other signal identification algorithms use ℓ_1 minimization or regularization to recover signals from incomplete or noisy measurements [47], [48], [2]. Within statistics, feature selection via the Lasso algorithm

[49] uses similar techniques. The same ideas are applied to reconstructing signals with sparse derivative (or gradient, more generally) in total variation de-noising, and in signals with sparse second derivative (or Laplacian) [50]. A related problem is parameter estimation where we fit a model to data. One example of this is fitting MA or ARMA models; here parameter estimation can be carried out with convex optimization [51], [52], [53].

Convex optimization is also used as a relaxation technique for problems that are essentially Boolean, as in the detection of faults [54], [55] or in decoding a bit string from a received noisy signal. In these applications a convex problem is solved, after which some kind of rounding takes place to guess the fault pattern or transmitted bit string [56], [57], [58].

Many methods of state estimation can be interpreted as involving convex optimization. (Basic Kalman filtering and least-squares fall in this category, but since the objectives are quadratic, the optimization problems can be solved analytically using linear algebra.) In the 1970s, ellipsoidal calculus was used to develop a state estimator less sensitive to statistical assumptions than the Kalman filter, by propagating ellipsoids that contain the state [59], [60]. The standard approach here is to work out a conservative update for the ellipsoid; but the most sophisticated methods for ellipsoidal approximation rely on convex optimization [1, §8.4]. Another recently developed estimation method is minimax regret estimation [61], which relies on convex optimization.

Convex optimization algorithms have also been used in wireless systems. Some examples here include on-line pulse shape design for reducing the peak or average power of a signal [62], receive antenna selection in MIMO systems [63], and performing demodulation by solving convex problems [64].

II. Disciplined convex programming

A standard trick in convex optimization, used since the origins of LP [65], is to transform the problem that must be solved into an equivalent problem, which is in a standard form that can be solved by a generic solver. A good example of this is the reduction of an ℓ_1 minimization problem to an LP; see [1, Chap. 4] for many more examples. Recently developed *parser-solvers*, such as YALMIP [66], CVX [67], CVXMOD [68], and Pyomo [69] automate this reduction process. The user specifies the problem in a natural form by declaring optimization variables, defining an objective, and specifying constraints. A general approach called *disciplined convex programming* (DCP) [70], [71] has emerged as an effective methodology for organizing and implementing parser-solvers for convex optimization. In DCP, the user combines built-in functions in specific, convexity-preserving ways. The constraints and objective must also follow certain rules. As long as the user conforms to these requirements, the parser can easily verify convexity of the problem and automatically transform it to a standard form, for transfer to the solver. The parser-solvers CVX (which runs in Matlab) and CVXMOD (Python) use the DCP approach.

A very simple example of such a scheme is the CVX code shown in Figure 1, which shows the required CVX code for specifying the convex optimization problem

```
1 A = [...]; b = [...]; Q = [...];
2 cvx_begin
3   variable x(5)
4   minimize(quad_form(x, Q))
5   subject to
6     abs(x) <= 1; sum(x) == 10; A*x >= 0
7 cvx_end
8 cvx_status
```

1) Problem data is specified within Matlab as ordinary matrices and vectors. Here A is a 3×5 matrix, b is a 3-vector, and Q is a 5×5 matrix.
2) Changes from ordinary Matlab mode to CVX model specification mode.
3) $x \in \mathbf{R}^5$ is an optimization variable object. After solution, x is replaced with a solution (numerical vector).
4) Recognized as convex objective $x^T Q x$ (provided $Q \geq 0$).
5) Does nothing, but enhances readability.
6) In CVX model specification mode, equalities and inequalities specify constraints.
7) Completes model specification, initiates transformation to standard form, and calls solver; solution is written to x.
8) Reports status, *e.g.*, Solved or Infeasible.

Fig. 1: CVX code segment (above) and explanations (below).

$$\begin{array}{ll} \text{minimize} & x^T Q x \\ \text{subject to} & |x| \leq 1, \quad \sum_i x_i = 10, \quad Ax \geq 0, \end{array} \tag{1}$$

with variable $x \in \mathbf{R}^5$, where $Q \in \mathbf{R}^{5\times 5}$ satisfies $Q = Q^T \geq 0$ (*i.e.*, is symmetric positive semidefinite) and $A \in \mathbf{R}^{3\times 5}$. Here both inequalities are elementwise, so the problem requires that $|x_i| \leq 1$, and $(Ax)_i \geq 0$. This simple problem could be transformed to standard QP form by hand; CVX and CVXMOD do it automatically. The advantage of a parser-solver like CVX would be much clearer for a larger more complicated problem. To add further (convex) constraints to this problem, or additional (convex) terms to the objective, is easy in CVX; but quite a task when the reduction to standard form is done by hand.

III. Code generation

Designing and prototyping a convex optimization-based algorithm requires choosing a suitable problem format, then testing and adjusting it for good application performance. In this prototyping stage, the speed of the solver is often nearly irrelevant; simulations can usually take place at significantly reduced speeds. In prototyping and algorithm design, the key is the ability to rapidly change the problem formulation and test the application performance, often using real data. The parser-solvers described in the previous section are ideal for such use, and reduce development time by freeing the user from the need to translate their problem into the restricted standard form required by the solver.

Once prototyping is complete, however, the final code must often run much faster. Thus, a serious challenge in using real-time convex optimization is the creation of a fast, reliable solver for a particular application. It is possible to hand-code solvers that take advantage of the special structure of a problem family, but such work is tedious and difficult to get exactly right. Given the success of parser-solvers for off-line applications, one option is to try a similar approach to the problem of generating fast custom solvers.

It is sometimes possible to use the (slow) code from the prototyping stage in the final algorithm. For example, the acceptable time frame for a fault detection algorithm may be measured in minutes, in which case the above prototype is likely adequate. Often, though, there are still advantages in having code that is independent of the particular modeling framework like CVX or CVXMOD. On the other hand (and as previously mentioned), some applications may require time scales that are faster than those achievable even with a very good generic solver; here explicit methods may be the only option. We are left with a large category of problems where a fast, automatically-generated solver would be extremely useful.

This introduces automatic code generation, where a user who is not necessarily an expert in algorithms for convex optimization can formulate and test a convex optimization problem within a familiar high level environment, and then request a custom solver. An automatic code generation system analyzes and processes the problem, (possibly) spending a significant amount of time testing or analyzing various methods. Then it produces code highly optimized for the particular problem family, including auxiliary code and files. This code may then be embedded in the user's signal processing algorithm.

We have developed an early, preliminary version of an automatic code generator. It is built on top of CVXMOD, a convex optimization modeling layer written in Python. After defining a problem (family), CVXMOD analyzes the problem's structure, and creates C code for a fast solver. Figure 2 shows how the problem family (1) can be specified in CVXMOD. Note, in particular, that no parameter values are given at this time; they are specified at solve time, when the problem family has been instantiated and a particular problem instance is available.

CVXMOD produces a variety of output files. These include `solver.h`, which includes prototypes for all necessary functions; `initsolver.c`, which allocates memory and initializes variables, and `solver.c`, which actually solves an instance of the problem. Figure 3 shows some of the key lines of code which would be used within a user's signal processing algorithm. In an embedded application, the initializations (lines 2–4) are called when the application is starting up; the solver (line 5) is called each time a problem instance is to be solved, for example in a real-time loop.

Generating and then compiling code for a modest sized convex optimization problem can take far longer than it would take the solve a problem instance using a parser-solver. But once we have the compiled code, we can solve instances of this specific problem at extremely high speeds. This compiled code is perfectly suited for inclusion in a real-time signal processing algorithm.

```
1  A = param('A', 3, 5)
2  b = param('b', 3, 1)
3  Q = param('Q', 5, 5, psd=True)
4  x = optvar('x', 5, 1)
5  objv = quadform(x, Q)
6  constr = [abs(x) <= 1, sum(x) == 10,
             A*x >= 0]
7  prob = problem(minimize(objv), constr)
8  codegen(prob).gen()
```

1) A is specified in CVXMOD as a 3×5 parameter. No values are (typically) assigned at problem specification time; here A is acting as a placeholder for later replacement with problem instance data.
2) b is specified as a 3-vector parameter.
3) Q is specified as a symmetric, positive-definite 5×5 parameter.
4) $x \in \mathbf{R}^5$ is an optimization variable.
5) Recognized as a convex objective, since CVXMOD has been told that $Q \geq 0$ in line 3.
6) Saves the affine equalities and convex inequalities to a list.
7) Builds the (convex) minimization problem from the convex objective and list of convex constraints.
8) Creates a code generator object based on the given problem, which then generates code.

Fig. 2: CVXMOD code segment (above) and explanations (below).

IV. LINEARIZING PRE-EQUALIZATION

Many types of nonlinear pre- and post-equalizers can be implemented using convex optimization. In this section we focus on one example, a nonlinear pre-equalizer for a nonlinear system with Hammerstein [72] structure, a unit saturation nonlinearity followed by a stable linear time-invariant system, shown in Figure 4. Our equalizer, shown in Figure 5, will have access to the scalar input signal u, with a lookahead of T samples (or, equivalently, with an additional delay of T samples), and will generate the equalized input signal v, that is applied to the system, resulting in output signal y. The goal is to choose v so that the actual output signal y matches the reference output y^{ref}, which is the output signal that would have resulted without the saturation nonlinearity. This is shown in the block diagram in Figure 6, which includes the error signal $e = y - y^{\mathrm{ref}}$. If the error signal is small, then our pre-equalizer followed by the system gives nearly the same output as the reference system, which is linear; thus, our pre-equalizer has linearized the system.

When the input peak does not exceed the saturation level 1, the error signal is zero; our equalizer will come into play only when the input signal peak exceeds one. A baseline choice of pre-equalizer is none: We simply take $v = u$. We will use this simple equalizer as a basis for comparison with the nonlinear

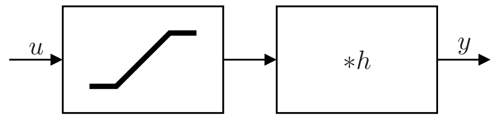

Fig. 4: Nonlinear system consisting of unit saturation followed by linear time-invariant system.

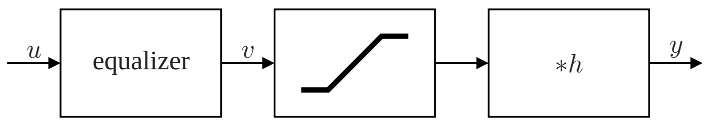

Fig. 5: Our pre-equalizer processes the incoming signal u (with a look-ahead of T samples) to produce the input v applied to the system.

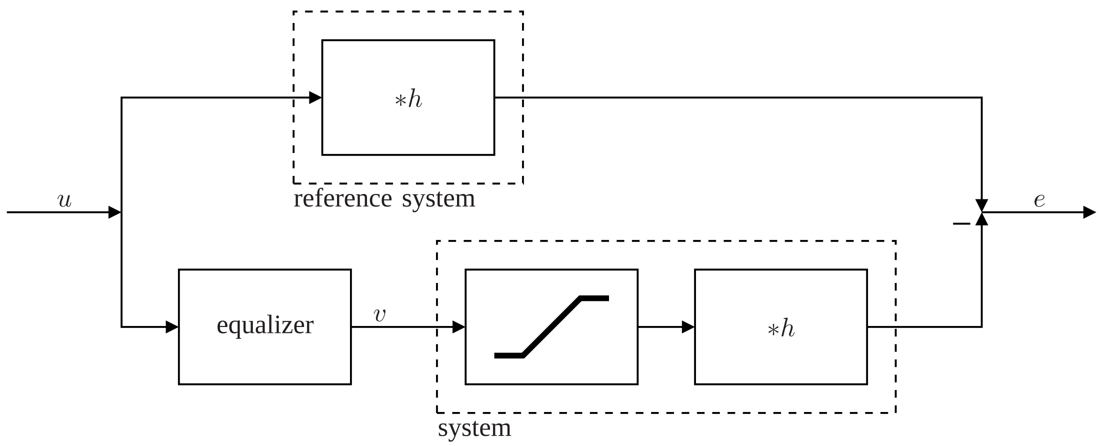

Fig. 6: The top signal path is the reference system, a copy of the system but without the saturation. The bottom signal path is the equalized system, the pre-equalizer followed by the system, shown in the dashed box. Our goal is to make the error e small.

equalizer we describe here. We'll refer to the output produced without pre-equalization as $y^{\rm none}$, and the corresponding error as $e^{\rm none}$.

We now describe the system, and the pre-equalizer, in more detail. We use a state-space model for the linear system,

$$x_{t+1} = Ax_t + B\text{sat}(v_t), \quad y_t = Cx_t,$$

with state $x_t \in \mathbf{R}^n$, where the unit saturation function is given by $\text{sat}(z) = z$ for $|z| \leq 1$, $\text{sat}(z) = 1$ for $z > 1$, and $\text{sat}(z) = -1$ for $z < -1$. The reference system is then

$$x^{\rm ref}_{t+1} = Ax^{\rm ref}_t + Bu_t, \quad y^{\rm ref}_t = Cx^{\rm ref}_t,$$

with state $x^{\rm ref}_t \in \mathbf{R}^n$. Subtracting these two equations we can express the error signal $e = y - y^{\rm ref}$ via the system

$$\tilde{x}_{t+1} = A\tilde{x}_t + B(\text{sat}(v_t) - u_t), \quad e_t = C\tilde{x}_t,$$

where $\tilde{x}_t = x_t - x^{\rm ref}_t \in \mathbf{R}^n$ is the state tracking error.

We now come to the main (and simple) trick: We will assume that (or more accurately, our pre-equalizer will guarantee that) $|v_t| \leq 1$. In this case $\text{sat}(v_t)$ can be replaced by v_t above, and we have

$$\tilde{x}_{t+1} = A\tilde{x}_t + B(v_t - u_t), \quad e_t = C\tilde{x}_t.$$

We can assume that $\tilde{x}_t$ is available to the equalizer; indeed, by stability of A, the simple estimator

$$\hat{x}_{t+1} = A\hat{x}_t + B(v_t - u_t)$$

will satisfy $\hat{x}_t \to \tilde{x}_t$ as $t \to \infty$, so we can use $\hat{x}_t$ in place of $\tilde{x}_t$. In addition to $\tilde{x}_t$, our equalizer will use a look-ahead of T samples on the input signal, *i.e.*, v_t will be formed with knowledge of $u_t, \ldots, u_{t+T}$.

We will use a standard technique from control, called model predictive control [29], in which at time t we solve an optimization problem to 'plan' our input signal over the next T steps, and use only the first sample of our plan as the actual equalizer output. At time t we solve the optimization problem

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{t+T} e_\tau^2 + \tilde{x}_{t+T+1}^T P \tilde{x}_{t+T+1} \\ \text{subject to} & \tilde{x}_{\tau+1} = A\tilde{x}_\tau + B(v_\tau - u_\tau), \quad e_\tau = C\tilde{x}_\tau \\ & \tau = t, \ldots, t+T \\ & |v_\tau| \leq 1, \quad \tau = t, \ldots, t+T, \end{array} \tag{2}$$

with variables $v_t, \ldots, v_{t+T} \in \mathbf{R}$ and $\tilde{x}_{t+1}, \ldots, \tilde{x}_{t+T+1} \in \mathbf{R}^n$. The initial (error) state in this planning problem, $\tilde{x}_t$, is known. The matrix P, which is a parameter, is symmetric and positive semdefinite.

The first term in the objective is the sum of squares of the tracking errors over the time horizon $t, \ldots, t+T$; the

```
1  #include "solver.h"
2  int main(int argc, char **argv) {
3    Params params = init_params();
4    Vars vars = init_vars();
5    Workspace work = init_work(vars);
6    for (;;) {
7      update_params(params);
8      status = solve(params, vars, work);
9      export_vars(vars);   }}
```

1) The automatically generated data structures are loaded.
2) CVXMOD generates standard C code for use on a range of platforms.
3) The `params` structure is used to set problem parameters.
4) After solution, the `vars` structure will provide access to optimal values for each of the original optimization variables.
5) An additional `work` structure is used for working memory. Its size is fixed, and known at compilation time. This means that *all* memory requirements and structures are known at compile time.
6) Once the initialization is complete, we enter the real-time loop.
7) Updated parameter values are retrieved from the signal processing system.
8) Actual solution requires just one command. This command executes in a bounded amount of time.
9) After solution, the resulting variable values are used in the signal processing system.

Fig. 3: C code generated by CVXMOD.

second term is a penalty for the final state error; it serves as a surrogate for the tracking error past our horizon, which we cannot know since we do not know the input beyond the horizon. One reasonable choice for P is the output Grammian of the linear system,

$$P = \sum_{i=0}^{\infty} \left(A^i\right)^T C^T C A^i,$$

in which case we have

$$\tilde{x}_{t+T+1}^T P x_{t+T+1} = \sum_{\tau=t+T+1}^{\infty} e_\tau^2,$$

provided $v_\tau = u_\tau$ for $\tau \geq t+T+1$.

The problem above is a QP. It can be modified in several ways; for example, we can add a (regularization) term such as

$$\rho \sum_{\tau=t+1}^{T+1} (v_{\tau+1} - v_\tau)^2,$$

where $\rho > 0$ is a parameter, to give a smoother post-equalized signal.

Our pre-equalizer works as follows. At time step t, we solve the QP above. We then use v_t, which is one of the variables from the QP, as our pre-equalizer output. We then update the error state as $\tilde{x}_{t+1} = A\tilde{x}_t + B(v_t - u_t)$.

A. Example

We illustrate the linearizing pre-equalization method with an example, in which the linear system is a third-order lowpass system with bandwidth 0.1π, with impulse response that lasts for (about) 35 samples. Our pre-equalizer uses a look-ahead horizon $T = 10$ samples, and we choose P as the output Gramian. We use smoothing regularization with $\rho = 0.01$. The input u a is lowpass filtered random signal, which saturates (*i.e.*, has $|u_t| > 1$) around 20% of the time.

The unequalized and equalized inputs are shown in Figure 7. We can see that the pre-equalized input signal is quite similar to the unequalized input when there is no saturation, but differs considerably when there is. The corresponding outputs, including the reference output, are shown in Figure 8, along with the associated output tracking errors.

The QP (2), after transformation, has 96 variables, 63 equality constraints, and 48 inequality constraints. Using Linux on an Intel Core Duo 1.7 GHz, it takes approximately 500 μs to solve using CVXMOD-generated code, which compares well with the standard SOCP solver SDPT3 [73], [74], whose solve time is approximately 300 ms.

V. Robust Kalman filtering

Kalman filtering is a well known and widely used method for estimating the state of a linear dynamical system driven by noise. When the process and measurement noises are independent identically distributed (IID) Gaussian, the Kalman filter recursively computes the posterior distribution of the state, given the measurements.

In this section we consider a variation on the Kalman filter, designed to handle an additional measurement noise term that is sparse, *i.e.*, whose components are often zero. This term can be used to model (unknown) sensor failures, measurement outliers, or even intentional jamming. The robust Kalman filter is obtained by replacing the standard measurement update, which can be interpreted as solving a quadratic minimization problem, with the solution of a similar convex minimization problem, that includes an ℓ_1 term to handle the sparse noise. Thus the robust Kalman filter requires the solution of a convex optimization problem in each time step. (The standard Kalman filter requires the solution of a quadratic optimization problem at each step, which has an analytical solution expressible using basic linear algebra operations.)

We will work with the system

$$x_{t+1} = Ax_t + w_t, \quad y_t = Cx_t + v_t + z_t,$$

where $x_t \in \mathbf{R}^n$ is the state (to be estimated) and $y_t \in \mathbf{R}^m$ is the measurement available to us at time step t. As in the standard setup for Kalman filtering, the process noise w_t is IID $\mathcal{N}(0, W)$, and the measurement noise term v_t is IID $\mathcal{N}(0, V)$. The term z_t is an additional noise term, which we assume is sparse (meaning, most of its entries are zero) and centered around zero. Without the additional sparse noise term

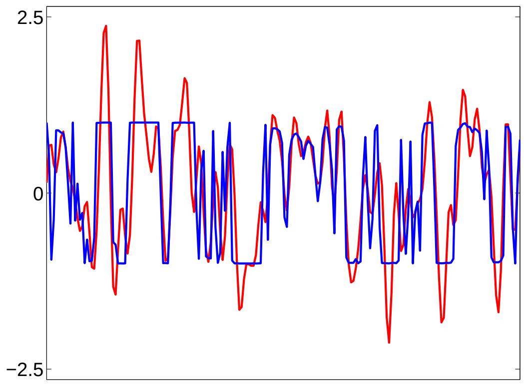

Fig. 7: Input without pre-equalization (red, u_t), and with linearizing pre-equalization (blue, v_t).

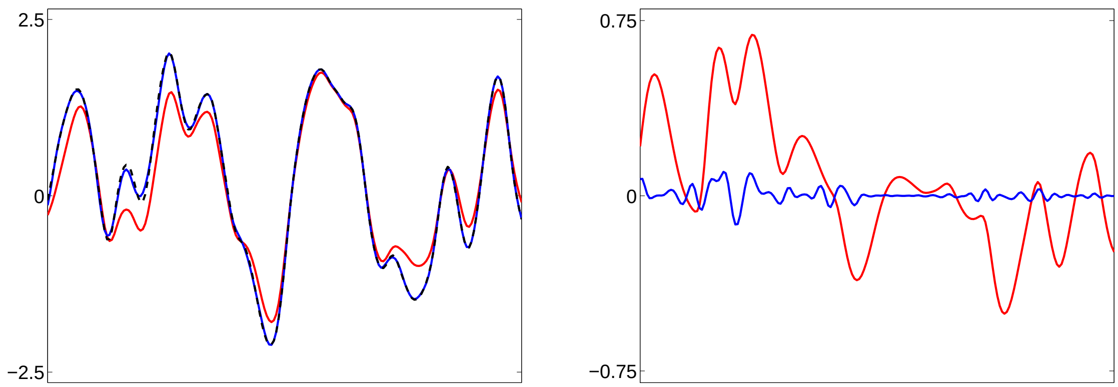

Fig. 8: Left: Output y without pre-equalization (red), and with nonlinear pre-equalization (blue). The reference output $y^{\rm ref}$ is shown as the dashed curve (black). **Right:** Tracking error e with no pre-equalization (red) and with nonlinear pre-equalization (blue).

z_t, our system is identical to the standard one used in Kalman filtering.

We will use the standard notation from the Kalman filter: $\hat{x}_{t|t}$ and $\hat{x}_{t|t-1}$ denote the estimates of the state x_t, given the measurements up to y_t, and Σ denotes the steady-state error covariance associated with predicting the next state. In the standard Kalman filter (*i.e.*, without the additional noise term z_t), all variables are jointly Gaussian, so the (conditional) mean and covariance specify the conditional distributions of x_t, conditioned on the measurements up to y_t and y_{t-1}, respectively.

The standard Kalman filter consists of alternating time and measurement updates. The time update,

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1}, \tag{3}$$

propagates forward the state estimate at time $t-1$, after the measurement $y_{t=1}$, to the state estimate at time t, but before the measurement y_t is known. The measurement update,

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + \Sigma C^T (C\Sigma C^T + V)^{-1}(y_t - C\hat{x}_{t|t-1}), \tag{4}$$

then gives the state estimate at time t, given the measurement y_t, starting from the state estimate at time t, before the measurement is known. In the standard Kalman filter, $\hat{x}_{t|t-1}$ and $\hat{x}_{t|t}$ are the conditional means, and so can be interpreted as the minimum mean-square error estimates of x_t, given the measurements up to y_{t-1} and y_t, respectively.

To (approximately) handle the additional sparse noise term z_t, we will modify the Kalman filter measurement update (4). To motivate the modification, we first note that $\hat{x}_{t|t}$ can be expressed as the solution of a quadratic optimization problem,

$$\begin{array}{ll} \mbox{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ \mbox{subject to} & y_t = Cx + v_t, \end{array}$$

with variables x and v_t. We can interpret v_t as our estimate of the sensor noise; the first term in the objective is a loss term corresponding to the sensor noise, and the second is a loss term associated with our estimate deviating from the prior.

In the robust Kalman filter, we take $\hat{x}_{t|t}$ to be the solution of the convex optimization problem

$$\begin{array}{ll} \mbox{minimize} & v_t^T V^{-1} v_t + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ & \quad +\lambda \|z_t\|_1 \\ \mbox{subject to} & y_t = Cx + v_t + z_t, \end{array} \tag{5}$$

with variables x, v_t, and z_t. (Standard methods can be used to transform this problem into an equivalent QP.) Here we interpret v_t and z_t as our estimates of the Gaussian and the sparse measurement noises, respectively. The parameter $\lambda \geq 0$ is adjusted so that the sparsity of our estimate coincides with our assumed sparsity of z_t. For λ large enough, the solution of this optimization problem has $z_t = 0$, and so is *exactly* the same as the solution of the quadratic problem above; in this case, the robust Kalman filter measurement update coincides with the standard Kalman filter measurement update.

In the robust Kalman filter, we use the standard time update (3), and the modified measurement update (5), which requires solving a (nonquadratic) convex optimization problem. With this time update, the estimation error is not Gaussian, so the estimates $\hat{x}_{t|t}$ and $\hat{x}_{t|t-1}$ are no longer conditional means (and Σ is not the steady-state state estimation error covariance). Instead we interpret them as merely (robust) state estimates.

A. Example

For this example, we randomly generate matrices $A \in \mathbf{R}^{50\times 50}$ and $C \in \mathbf{R}^{15\times 50}$. We scale A so its spectral radius is 0.98. We generate a random matrix $B \in \mathbf{R}^{50\times 5}$ with entries $\sim \mathcal{N}(0,1)$, and use $W = BB^T$ and $V = I$. The sparse noise z_t was generated as follows: with probability 0.05, component $(y_t)_i$ is set to $(v_t)_i$; *i.e.*, the signal component is removed. This means that $z \neq 0$ with probability 0.54, or, roughly, one in two measurement vectors contains at least one bogus element. We compare the performance of a traditional Kalman filter tuned to W and V with the robust Kalman filter described above.

For this example the time update (5) is transformed into a QP with 95 variables, 15 equality, and 30 inequality constraints. By analytically optimizing over some of the variables that appear only quadratically, the problem can be reduced to a smaller QP. Code generated by `CVXMOD` solves this problem in approximately 120 μs, which allows meausurement updates at rates better than 5 kHz. Solution with SDPT3 takes 120 ms, while a standard Kalman filter update takes 10 μs.

VI. On-line array weight design

In this example, fast optimization is used to adapt the weights to changes in the transmission model, target signal characteristics, or objective. Thus, the optimization is used to adapt or re-configure the array. In traditional adaptive array signal processing [75], the weights are adapted directly from the combined signal output; here we consider the case when this is not possible.

We consider a generic array of n sensors, each of which produces as output a complex number (baseband response) that depends on a parameter $\theta \in \Theta$ (which can be a vector in the general case) that characterizes the signal. In the simplest case, θ is a scalar that specifies the angle of arrival of a signal in 2-D; but it can include other parameters that give the range or position of the signal source, polarization, wavelength, and so on. The sensor outputs are combined linearly with a set of array weights $w \in \mathbf{C}^n$ to produce the (complex scalar) combined output signal

$$y(\theta) = a(\theta)^* w.$$

Here $a : \Theta \to \mathbf{C}^n$ is called the array response function or array manifold.

The weight vector w is to be chosen, subject to some constraints expressed as $w \in \mathcal{W}$, so that the combined signal output signal (also called the array response) has desired characteristics. A generic form for this problem is to guarantee unit array response for some target signal parameter, while giving uniform rejection for signal parameters in some set of values Θ_{rej}. We formulate this as the optimization problem, with variable $w \in \mathbf{C}^n$,

$$\begin{array}{ll} \text{minimize} & \max_{\theta\in\Theta_{\text{rej}}} |a(\theta)^* w| \\ \text{subject to} & a(\theta^{\text{tar}})^* w = 1, \quad w \in \mathcal{W}. \end{array}$$

If the weight constraint set $\mathcal{W}$ is convex, this is a convex optimization problem [38], [76].

In some cases the objective, which involves a maximum over an infinite set of signal parameter values, can be handled exactly; but we will take a simple discretization approach. We find appropriate points $\theta_1, \dots, \theta_N \in \Theta_{\text{rej}}$, and replace the maximum over all values in Θ_{rej} with the maximum over these values to obtain the problem

$$\begin{array}{ll} \text{minimize} & \max_{i=1,\dots,N} |a(\theta_i)^* w| \\ \text{subject to} & a(\theta^{\text{tar}})^* w = 1, \quad w \in \mathcal{W}. \end{array}$$

When $\mathcal{W}$ is convex, this is a (tractable) constrained complex ℓ_∞ norm minimization problem:

$$\begin{array}{ll} \text{minimize} & \|Aw\|_\infty \\ \text{subject to} & a(\theta_{\text{tar}})^* w = 1, \quad w \in \mathcal{W}, \end{array} \qquad (6)$$

where $A \in \mathbf{C}^{N\times n}$, with ith row $a(\theta_i)^*$, and $\|\cdot\|_\infty$ is the complex ℓ_∞ norm. It is common to add some regularization to the weight design problem, by adding $\lambda\|w\|^2$ to the objective, where λ is a (typically small) positive weight. This can be intrepeted as a term related to noise power in the combined array output, or as a regularization term that results keeps the weights small, which makes the combined array response less sensitive to small changes in the array manifold.

With or without regularization, the problem (6) can be transformed to an SOCP. Standard SOCP methods can be used to determine w when the array manifold or target parameter θ_{tar} do not change, or change slowly or infrequently. We are interested here in the case when they change frequently, which requires solving the problem (6) rapidly.

A. Example

We consider an example of an array in 2-D with $n = 15$ sensors with positions $p_1, \dots, p_n \in \mathbf{R}^2$ that change or drift over time. The signal model is a harmonic plane wave with wavelength λ arriving from angle θ, with $\Theta = [-\pi, \pi)$. The array manifold has the simple form

$$a(\theta)_i = \exp\left(-2\pi i (\cos\theta, \sin\theta)^T p_i/\lambda\right).$$

We take $\theta_{\text{tar}} = 0$ as our (constant) target look (or transmit) direction, and the rejection set as

$$\Theta_{\text{rej}} = [-\pi, -\pi/9] \cup [\pi/9, \pi)$$

(which corresponds to a beamwidth of $40°$). We discretize arrival angles uniformly over θ_{rej} with $N = 100$ points.

The initial sensor positions are a 5×3 grid with $\lambda/2$ spacing. Each of these undergoes a random walk, with $p_i(t+1) - p_i(t) \sim \mathcal{N}(0, \lambda/10)$, for $t = 0, 1, \dots, 49$. Tracks of the sensor positions over $t = 0, \dots, 50$ are shown in Figure 10.

For each sensor position we solve the weight design problem, which results in a rejection ratio (relative gain of

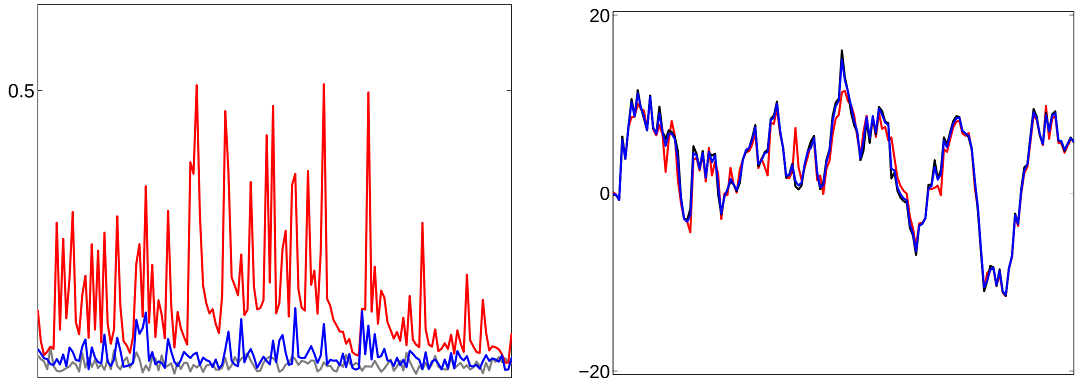

Fig. 9: The robust Kalman filter (blue) has significantly lower error (left) than the standard Kalman filter (red), and tracks $x_t^{(1)}$ (right) more closely.

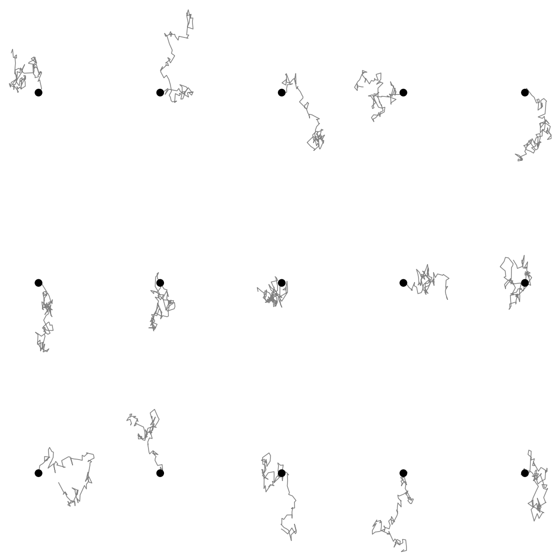

Fig. 10: Tracks of sensor positions, each of which is a random walk.

target direction to rejected directions) ranging from 7.4 dB to 11.9 dB. The resulting array response, *i.e.*, $|y(\theta)|$ versus t, is shown on the left in Figure 11. The same figure shows the array responses obtained using the optimal weights for the initial sensor positions, *i.e.*, without re-designing the weights as the sensor positions drift. In this case the rejection ratio goes up to 1.3 dB, *i.e.*, the gain in a rejection direction is almost the same as the gain in the target direction.

This problem can be transformed to an SOCP with 30 variables and approximately 200 constraints. The current version of `CVXMOD` does not handle SOCPs, but a simple implementation coded by hand solves this problem in approximately 2 ms, which means that we can update our weights at 500 Hz.

References

[1] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[2] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[3] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[4] Y. Nesterov and A. Nemirovskii, *Interior Point Polynomial Algorithms in Convex Programming*. SIAM, 1994, vol. 13.

[5] S. J. Wright, *Primal-Dual Interior-Point Methods*. SIAM, 1997.

[6] S. Boyd and C. Barratt, *Linear Controller Design: Limits of Performance*. Prentice-Hall, 1991.

[7] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics, 1994.

[8] M. A. Dahleh and I. J. Diaz-Bobillo, *Control of Uncertain Systems: A Linear Programming Approach*. Prentice-Hall, 1995.

[9] M. Hershenson, S. Boyd, and T. H. Lee, "Optimal design of a CMOS op-amp via geometric programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 1, pp. 1–21, 2001.

[10] M. Hershenson, S. S. Mohan, S. Boyd, and T. H. Lee, "Optimization of inductor circuits via geometric programming," in *Design Automation Conference*. IEEE Computer Society, 1999, pp. 994–998.

[11] S. Boyd, S.-J. Kim, D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, no. 6, pp. 899–932, 2005.

[12] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[13] G. Cornuejols and R. Tütüncü, *Optimization methods in finance*. Cambridge University Press, 2007.

[14] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, pp. 237–252, 1998.

[15] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[16] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, 2007.

[17] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. Springer, 2000.

[18] N. Cristianini and J. Shawe-Taylor, *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[19] Y. C. Eldar, A. Megretski, and G. C. Verghese, "Designing optimal quantum detectors via semidefinite programming," *IEEE Transactions on Information Theory*, vol. 49, no. 4, pp. 1007–1012, 2003.

[20] Y. C. Eldar, "A semidefinite programming approach to optimal unambiguous discrimination of quantum states," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 446–456, 2003.

[21] R. Graham, M. Grötschel, and L. Lovász, *Handbook of combinatorics*. MIT Press, 1996, vol. 2, ch. 28.

[22] *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, Dec. 2007, special Issue on Convex Optimization Methods for Signal Processing.

[23] R. Calvin, C. Ray, and V. Rhyne, "The design of optimal convolutional filters via linear programming," *IEEE Trans. Geoscience Elec.*, vol. 7, no. 3, pp. 142–145, Jul. 1969.

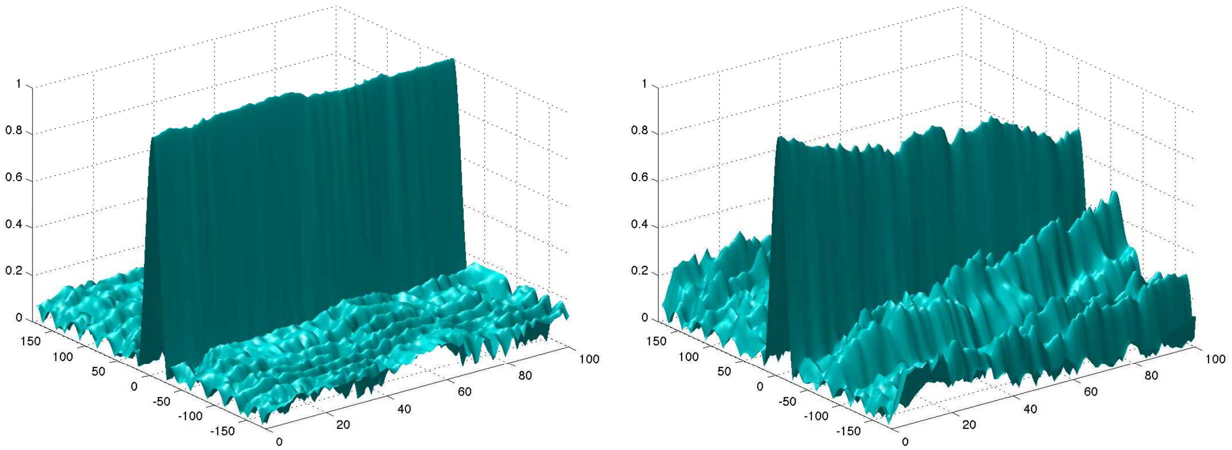

Fig. 11: Array response as sensors move, with optimized weights (left) and using weights for initial sensor positions (right).

[24] L. Rabiner, "Linear program design of finite impulse response (FIR) digital filters," *IEEE Trans. Aud. Electroacoustics*, vol. 20, no. 4, pp. 280–288, Oct. 1972.

[25] J. Claerbout and F. Muir, "Robust modeling with erratic data," *Geophysics*, vol. 38, p. 826, 1973.

[26] H. Taylor, S. Banks, and J. McCoy, "Deconvolution with the ℓ_1 norm," *Geophysics*, vol. 44, no. 1, pp. 39–52, 1979.

[27] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," in *Proceedings IFAC World Congress*, Jul. 2008, pp. 6974–6997.

[28] ——, "Fast evaluation of quadratic control-Lyapunov policy," Working manuscript, Jul. 2009, available online: http://stanford.edu/~boyd/papers/fast_clf.html.

[29] J. M. Maciejowski, *Predictive Control With Constraints*. Prentice Hall, 2002.

[30] A. H. Sayed, *Fundamentals of Adaptive Filtering*. IEEE Press, 2003.

[31] S. Haykin, *Adaptive filter theory*. Prentice Hall, 1996.

[32] X. Lai, "Design of smallest size two-dimensional linear-phase FIR filters with magnitude error constraint," *Multidimensional Systems and Signal Processing*, vol. 18, no. 4, pp. 341–349, 2007.

[33] T. Q. Nguyen and R. D. Koilpillai, "The theory and design of arbitrary-length cosine-modulated filter banks and wavelets, satisfying perfect reconstruction," *IEEE Transactions on Signal Processing*, vol. 44, no. 3, pp. 473–483, 1996.

[34] B. Alkire and L. Vandenberghe, "Interior-point methods for magnitude filter design," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, 2001.

[35] S. P. Wu, S. Boyd, and L. Vandenberghe, "FIR filter design via semidefinite programming and spectral factorization," in *IEEE Conference on Decision and Control*, vol. 1, 1996, pp. 271–276.

[36] R. L. Kosut, C. R. Johnson, and S. Boyd, "On achieving reduced error propagation sensitivity in DFE design via convex optimization (I)," in *IEEE Conference on Decision and Control*, vol. 5, 2000, pp. 4320–4323.

[37] S. A. Vorobyov, A. B. Gershman, and Z. Q. Luo, "Robust adaptive beamforming using worst-case performance optimization via second-order cone programming," in *IEEE Conference on Acoustics, Speech, and Signal Processing*, vol. 3, 2002.

[38] H. Lebret and S. Boyd, "Antenna array pattern synthesis via convex optimization," *IEEE Transactions on Signal Processing*, vol. 45, no. 3, pp. 526–532, 1997.

[39] J. Li, P. Stoica, and Z. Wang, "On robust Capon beamforming and diagonal loading," *IEEE Transactions on Signal Processing*, vol. 51, no. 7, pp. 1702–1715, 2003.

[40] R. G. Lorenz and S. Boyd, "Robust minimum variance beamforming," *IEEE Transactions on Signal Processing*, vol. 53, no. 5, pp. 1684–1696, 2005.

[41] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[42] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, no. 1-4, pp. 259–268, 1992.

[43] E. J. Candès and F. Guo, "New multiscale transforms, minimum total variation synthesis: Applications to edge-preserving image reconstruction," *Signal Processing*, vol. 82, no. 11, pp. 1519–1543, 2002.

[44] A. Chambolle, "An algorithm for total variation minimization and applications," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1, pp. 89–97, 2004.

[45] A. Beck, A. Ben-Tal, and C. Kanzow, "A fast method for finding the global solution of the regularized structured total least squares problem for image deblurring," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 419–443, 2008.

[46] K. L. Kroeker, "Face recognition breakthrough," *Communications of the ACM*, vol. 52, no. 8, pp. 18–19, 2009.

[47] E. Candès, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207–1223, 2005.

[48] J. Tropp, "Just relax: Convex programming methods for identifying sparse signals in noise," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1030–1051, 2006.

[49] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *Journal of the Royal Statistical Society*, vol. 58, no. 1, pp. 267–288, 1996.

[50] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky, "ℓ_1 trend filtering," *SIAM Review*, vol. 51, no. 2, pp. 339–360, 2009.

[51] P. Stoica, T. McKelvey, and J. Mari, "MA estimation in polynomial time," in *Proceedings of the IEEE Conference on Decision and Control*, vol. 4, 1999.

[52] B. Dumitrescu, I. Tabus, and P. Stoica, "On the parameterization of positive real sequences and MA parameter estimation," *IEEE Transactions on Signal Processing*, vol. 49, no. 11, pp. 2630–2639, 2001.

[53] B. Alkire and L. Vandenberghe, "Handling nonnegative constraints in spectral estimation," in *Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, vol. 1, 2000.

[54] A. Zymnis, S. Boyd, and D. Gorinevsky, "Relaxed maximum a posteriori fault identification," *Signal Processing*, vol. 89, no. 6, pp. 989–999, Jun. 2009.

[55] ——, "Mixed state estimation for a linear Gaussian Markov model," *IEEE Conference on Decision and Control*, pp. 3219–3226, Dec. 2008.

[56] J. Feldman, D. R. Karger, and M. J. Wainwright, "LP decoding," in *Proceedings of the annual Allerton conference on communication control and computing*, vol. 41, no. 2, 2003, pp. 951–960.

[57] E. Candes and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[58] E. J. Candes and P. A. Randall, "Highly robust error correction by convex programming," *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 2829–2840, 2008.

[59] F. Schlaepfer and F. Schweppe, "Continuous-time state estimation under disturbances bounded by convex sets," *IEEE Transactions on Automatic Control*, vol. 17, no. 2, pp. 197–205, 1972.

[60] A. B. Kurzhanski and I. Vályi, *Ellipsoidal calculus for estimation and control*. Birkhäuser, 1996.

[61] Y. C. Eldar, A. Ben-Tal, and A. Nemirovski, "Linear minimax regret estimation of deterministic parameters with bounded data uncertainties," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2177–2188, 2004.

[62] T. N. Davidson, Z. Q. Luo, and K. M. Wong, "Design of orthogonal pulse shapes for communications via semidefinite programming," *IEEE Transactions on Signal Processing*, vol. 48, no. 5, pp. 1433–1445, 2000.

[63] A. Dua, K. Medepalli, and A. J. Paulraj, "Receive antenna selection in MIMO systems using convex optimization," *IEEE Transactions on Wireless Communications*, vol. 5, no. 9, pp. 2353–2357, 2006.

[64] G. Sell and M. Slaney, "Solving demodulation as an optimization problem," *To appear, IEEE Transactions on Audio, Speech and Language Processing*, Jul. 2009.

[65] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.

[66] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004, http://control.ee.ethz.ch/~joloef/yalmip.php.

[67] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming (web page and software)," http://www.stanford.edu/~boyd/cvx/, Jul. 2008.

[68] J. Mattingley and S. Boyd, "CVXMOD: Convex optimization software in Python (web page and software)," http://cvxmod.net/, Aug. 2008.

[69] W. E. Hart, "Python optimization modeling objects (Pyomo)," in *Proceedings, INFORMS Computing Society Conference*, 2009.

[70] M. Grant, "Disciplined convex programming," Ph.D. dissertation, Department of Electrical Engineering, Stanford University, Dec. 2004.

[71] M. Grant, S. Boyd, and Y. Ye, "Disciplined convex programming," in *Global Optimization: from Theory to Implementation*, ser. Nonconvex Optimization and Its Applications, L. Liberti and N. Maculan, Eds. New York: Springer Science & Business Media, Inc., 2006, pp. 155–210.

[72] I. W. Hunter and M. J. Korenberg, "The identification of nonlinear biological systems: Wiener and Hammerstein cascade models," *Biological Cybernetics*, vol. 55, no. 2, pp. 135–144, 1986.

[73] K. C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3—a Matlab software package for semidefinite programming, version 1.3," *Optimization Methods and Software*, vol. 11, no. 1, pp. 545–581, 1999.

[74] R. Tütüncü, K. Toh, and M. J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," *Mathematical Programming*, vol. 95, no. 2, pp. 189–217, 2003.

[75] D. H. Johnson and D. E. Dudgeon, *Array signal processing: concepts and techniques*. Simon & Schuster, 1992.

[76] L. El Ghaoui and H. Lebret, "Robust solutions to least-squares problems with uncertain data," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 4, pp. 1035–1064, 1997.