Code Generation for Receding Horizon Control

Jacob Mattingley Yang Wang Stephen Boyd*

August 27, 2010

Abstract

Receding horizon control (RHC), also known as model predictive control (MPC), is a general purpose control scheme that involves repeatedly solving a constrained optimization problem, using predictions of future costs, disturbances, and constraints over a moving time horizon to choose the control action. RHC handles constraints, such as limits on control variables, in a direct and natural way, and generates sophisticated feed-forward actions. The main disadvantage of RHC is that an optimization problem has to be solved at each step, which leads many control engineers to think that it can only be used for systems with slow sampling (say, less than one Hz). Several techniques have recently been developed to get around this problem. In one approach, called explicit MPC, the optimization problem is solved analytically and explicitly, so evaluating the control policy requires only a lookup table search. Another approach, which is our focus here, is to exploit the structure in the optimization problem to solve it efficiently. This approach has previously been applied in several specific cases, using custom, hand written code. However, this requires significant development time, and specialist knowledge of optimization and numerical algorithms. Recent developments in convex optimization code generation have made the task much easier and quicker. With code generation, the RHC policy is specified in a high-level language, then automatically transformed into source code for a custom solver. The custom solver is typically orders of magnitude faster than a generic solver, solving in milliseconds or microseconds on standard processors, making it possible to use RHC policies at kilohertz rates. In this paper we demonstrate code generation with four simple control examples. They show a range of problems that may be handled by RHC. In every case, we show a speedup of several hundred times from generic parser-solvers.

*The authors are with the Information Systems Laboratory, Electrical Engineering Department, Stanford University, CA 94305-9510 USA. Email: {jacobm,yw224,boyd}@stanford.edu.

1 Introduction

Receding horizon control (RHC), also known as *model predictive control* (MPC) [KH05, Whi82, GSD05, Mac02] is a form of feedback control system that first became popular in the 1980s. With RHC, we solve an optimization problem at each time step to determine a plan of action over a fixed time horizon, and then apply the first input from this plan. At the next time step we repeat the planning process, solving a new optimization problem, with the time horizon shifted one step forward. The optimization problem takes into account estimates of future quantities, based on available measurements and data at each time step. Thus the control policy involves *feedback*, where real-time measurements are used to determine the control input.

RHC is a nonlinear control policy that handles input constraints, output constraints, and a variety of control objectives. Using RHC, systems can be controlled near their physical limits, obtaining performance superior to linear control. RHC has been successfully applied in a wide range of practical settings, including industrial and chemical process control [QB03], supply chain management [CTHK03], stochastic control in economics and finance [Her05], and revenue management [TR04].

One drawback of RHC is that an optimization problem must be solved at each time step. Using conventional numerical optimization techniques, the time taken to solve this problem is often much longer compared with, for example, the time taken to compute the control action for a linear controller. This means that applications of RHC have been mostly limited to systems with sample times measured in seconds, minutes or hours.

Many methods can be used to speed up the solution of the optimization problems that arise in RHC. When the numbers of states, inputs, and constraints are small, one approach is *explicit MPC* [BF04, BMDP02], where a closed-form expression for the solution of the optimization problem, as a function of the current state, is computed offline and stored. The online algorithm then reduces to a lookup table search, followed by a linear control law evaluation, which can be computed extremely quickly. Another method, applicable to a problem of any size, is to custom code online optimization solvers that exploit the problem structure in RHC applications [WB08, DFS$^+$02, ÅkerbladH04, RWR04]. These custom solvers can yield computation times that are several orders of magnitude faster than generic solvers, but developing them requires time-consuming hand coding, and significant expertise in optimization algorithms and numerical computation.

In this paper, we describe advances that make it much easier to develop custom RHC solvers. By combining a high-level specification language for optimization and recently-developed code generation tools, a user of RHC can quickly specify and generate fast, reliable custom code. Since a user does not need any expertise in optimization, many more practitioners can use RHC, even in settings involving kilohertz sample rates.

We do not claim that RHC always outperforms traditional control methods. In many cases, a skilled designer can achieve similar performance by carefully tuning a conventional linear controller such as a proportional-integral-derivative (PID) controller, suitably modified to handle constraints (for example by saturating control inputs that are outside their limits).

The main advantage of RHC is the simple and transparent design process, which requires

much less intuition and control expertise. In RHC, the designer specifies the objective and constraints as part of an optimization problem, whereas in a conventional design process, the designer adjusts controller gains and coefficients to indirectly handle constraints, often by trial and error. Thus RHC, combined with automatic code generation, offers an attractive framework for rapid design of sophisticated controllers for a wide range of problems, including those that require fast sample rates.

In the remainder of the paper, we give a high-level overview of RHC, briefly explain code generation for RHC using the software package CVXGEN [MB10b], and illustrate the ideas with four examples. The examples are simple, and chosen to show the variety of problems that can be addressed. In our discussions we avoid unnecessary detail, so we refer the interested reader to [BV04] for a more detailed account of convex optimization, [GBY06] for more on disciplined convex programming, and [MB09, MB10a] for a discussion of code generation for convex optimization.

We restrict attention to systems with linear dynamics, convex objective functions and constraints, for several reasons. First, many real systems can be reasonably modeled in this restricted form. Secondly, standard linearization techniques can be used to extend these methods to many nonlinear systems. (For example, almost all commercial MPC systems for process control rely on linearization around an operating point.) Finally, many of the techniques we discuss could be applied to general nonlinear systems. For nonlinear control, a sequence of linear RHC problems is solved at each time step. For some work in this area, see [RS07], which describes the software package NEWCON; see [OK02] for an example of automatic code generation applied to nonlinear RHC, or [Oht04] for an application to a two-link robot arm. Also see the ACADO toolbox [HF08], which is a software package for solving nonlinear RHC problems.

2 Formulating RHC problems

2.1 System dynamics and control

System dynamics. We consider a discrete-time linear dynamical system of the form

$$x_{t+1} = A_t x_t + B_t u_t + c_t,$$

where $x_t \in \mathbf{R}^n$ is the system state, $u_t \in \mathbf{R}^m$ is the control input, and $c_t \in \mathbf{R}^n$ is an exogenous input. The matrices $A_t \in \mathbf{R}^{n\times n}$ and $B_t \in \mathbf{R}^{n\times m}$ are the dynamics and input matrices, respectively. The subscripts on A_t, B_t, and c_t indicate that they may change with time, but in many applications these matrices are constant and known.

Constraints and objective. The state and input must satisfy some constraints, expressed abstractly as

$$(x_t, u_t) \in \mathcal{C}_t,$$

where $\mathcal{C}_t \subseteq \mathbf{R}^n \times \mathbf{R}^m$ is the constraint set. The instantaneous cost depends on both the current state and control action, and is denoted $\ell_t(x_t, u_t)$. We judge the quality of control

by the average cost,

$$J = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \ell_t(x_t, u_t),$$

where we assume the limit exists. If $\ell_t(x_t, u_t)$ is a random variable, we replace $\ell_t(x_t, u_t)$ with its expected value $\mathbf{E}\,\ell_t(x_t, u_t)$. As with the dynamics data, we subscript the constraint set and objective function with the time t, to handle the case when they vary with time.

Information available for control. The control input u_t is determined using the information available to the controller at time t, including *estimates* of quantities that are not known, based on information that is known. We denote these estimates as

$$\hat{A}_{\tau|t}, \quad \hat{B}_{\tau|t}, \quad \hat{c}_{\tau|t}, \quad \hat{\mathcal{C}}_{\tau|t}, \quad \hat{\ell}_{\tau|t}, \quad \hat{x}_{t|t},$$

where the notation $\hat{z}_{\tau|t}$ denotes the estimate of z_τ, based on information available at time t, where $\tau \geq t$. 'Information available at time t' includes conventional data in a control system, such as those available from sensor measurements, or known quantities and functions. The available information can also include other relevant information that is not typically used in a traditional control system, such as historical usage patterns, weather, price trends, and expert forecasts.

These estimates can be obtained in many ways. In the simplest case, we *know* the quantity being estimated, in which case we replace the estimates with the known value. For example, if the system dynamics matrices A_t and B_t have known and constant values A and B, we take $\hat{A}_{\tau|t} = A$ and $\hat{B}_{\tau|t} = B$. If the controller has access to the current state x_t, we take $\hat{x}_{t|t} = x_t$.

A traditional method for obtaining the estimates is from a statistical model of the unknown data, in which case the estimates can be conditional expectations or other statistical estimates, based on the data available at time t. For example, the additive terms c_t are often modeled as independent zero mean random variables, with natural estimate $\hat{c}_{\tau|t} = 0$.

However, the estimates need not be derived from statistical models; for example, future prices could be obtained from a futures market, or from analysts who predict trends. Another example arises when the system to be controller is nonlinear. In this case $\hat{A}_{\tau|t}$, $\hat{B}_{\tau|t}$, and $\hat{c}_{\tau|t}$ can be a linearization of the nonlinear dynamics, along a trajectory.

Controller design problem. The controller design problem is to find a control policy that chooses the input u_t as a function of the quantities known at time t, so that the constraints are satisfied, and the average cost J is made small.

We have not fully specified the uncertainty model, so this description of the control problem is informal, and we cannot really define an optimal control policy. But when we give a full mathematical description of the uncertainty, for example as a statistical model, we can define the optimal control policy, which is the policy that minimizes J, among all policies that map the information available into a control action while respecting the constraints.

2.2 Receding horizon control

The basic RHC policy works as follows. At time t, we consider a time interval extending T steps into the future: $t, t+1, \ldots, t+T$. We then carry out the following steps:

1. *Form a predictive model.* Replace all unknown quantities over the time interval with their current estimates, using data available at time t.

2. *Optimize.* Solve the problem of minimizing the objective, subject to the dynamics and constraints. Here, the objective, dynamics and constraints are estimates, based on information available at time t.

3. *Execute.* Choose u_t to be the value obtained in the optimization problem of step 2.

Steps 1 and 2. The RHC optimization problem in step 2 takes the form

$$\begin{array}{ll} \text{minimize} & \frac{1}{T+1}\sum_{\tau=t}^{t+T} \hat{\ell}_{\tau|t}(\hat{x}_\tau, \hat{u}_\tau) \\ \text{subject to} & \hat{x}_{\tau+1} = \hat{A}_{\tau|t}\hat{x}_\tau + \hat{B}_{\tau|t}\hat{u}_\tau + \hat{c}_{\tau|t}, \\ & (\hat{x}_\tau, \hat{u}_\tau) \in \hat{\mathcal{C}}_{\tau|t}, \quad \tau = t, \ldots, t+T \\ & \hat{x}_t = \hat{x}_{t|t}, \end{array} \tag{1}$$

with variables $\hat{x}_t, \ldots, \hat{x}_{t+T+1}$ and $\hat{u}_t, \ldots, \hat{u}_{t+T}$. The data in this RHC optimization problem are the estimates

$$\hat{A}_{\tau|t}, \quad \hat{B}_{\tau|t}, \quad \hat{c}_{\tau|t}, \quad \hat{\mathcal{C}}_{\tau|t}, \quad \hat{\ell}_{\tau|t},$$

for $\tau = t, \ldots, t+T$, and the current state estimate, $\hat{x}_{t|t}$. (In many applications, we can use known, exact values for many of the parameters.) We can interpret $\hat{u}_t^\star, \ldots, \hat{u}_{t+T}^\star$, the optimal values from the RHC optimization problem (1), as a *plan of action* for the next T steps.

Step 3. We then choose $u_t = \hat{u}_t^\star$ to be the RHC action. At the next time step, the process is repeated, with (possibly) updated estimates of the current state and future quantities.

Comments. We make a few comments about the RHC policy. First, it is common to add a final state constraint or a final state cost, to the RHC problem. In the former case, we add an equality constraint of the form $x_{T+1} = x^{\text{final}}$, or a final constraint set condition $x_{T+1} \in \mathcal{C}^{\text{final}}$. In the latter case, we add $V(x_{T+1})$ to the objective, where V is a cost function for the final state. This can allow simpler, shorter-horizon controllers to approximate the behavior of controllers with longer horizons. Indeed, one can often obtain very good control performance with horizon $T = 0$, with a carefully chosen terminal constraint function [WB09]. In this case the control policy is also known by other names, such as control-Lyapunov policy or approximate dynamic programming (ADP) policy, and can be evaluated on timescales measured in tens of microseconds, allowing control at rates exceeding tens of kilohertz [WB10].

We assume that $\hat{\mathcal{C}}_t$ and $\hat{\ell}_t$ are convex, which means that the RHC problem (1) is a convex optimization problem. We can solve these optimization problems efficiently using standard

convex optimization tools [BV04]. Problems with nonconvex objective and constraints can be handled using sequential convex optimization, where a sequence of convex problems is solved to find good local solutions of the nonconvex problem.

The RHC policy is generally not an optimal policy, even when we have a formal model of the uncertainty. Instead, RHC is merely a sophisticated heuristic, which works very well in many applications.

2.3 Comparison with traditional feedback controller

The RHC controller is very similar to a classical feedback controller, which processes sensor signals and commands to produce the control input at each step. The RHC controller processes sensor signals, commands, and also possibly other nontraditional signals, such as future prices or weather predictions, which are used to form estimates. One major difference is that the RHC policy requires solving an optimization problem in each time step, and is therefore considerably more complex than a traditional feedback control law.

Several other aspects of the RHC controller are reminiscent of standard linear feedback controllers. The RHC controller can be conceptually divided into two parts: An *estimator*, that makes predictions about the future based on the information available, and an *optimizer*, which calculates an optimal plan of action assuming the estimates are correct. This architecture is just like the classical linear-quadratic-Gaussian (LQG) estimated-state feedback controller. In the case of LQG the celebrated *separation principle* tells us the architecture is actually optimal [AM71]. In the general RHC application, however, the separation principle does not hold; instead, the RHC controller is simply a very sophisticated suboptimal controller.

The RHC controller has a natural interpretation in cases when the RHC controller problem is a quadratic program (QP). In this case, the RHC control policy can be expressed as a piecewise affine (linear plus constant) function of the state. (Indeed, this is the basic principle behind explicit MPC [BMDP02, BF04].) Thus the RHC controller can be thought of as a collection of traditional linear controllers; the RHC controller switches between them depending on the current state.

3 Designing and implementing RHC controllers

3.1 Designing an RHC controller

To specify an RHC policy, we must describe three things: the method for estimating unknown quantities from current data, including, in particular, the system model; the horizon T; and the terminal costs and constraints, if any. For a given choice of these, the closed-loop system is simulated to judge the performance, and if needed, some of the design choices can be modified. Simulation of an RHC system requires solving an optimization problem in each time step. Even a basic simulation will require the solution of many optimization problems.

Many convenient software tools [L̈04, GB08, MB08] are available for rapidly formulating and solving convex problems. These *parser-solvers* take a high-level specification, and perform the necessary transformations for solution by a standard convex optimization solver, for example, [TTT99, TTT03, Stu99]. This allows the designer to easily change the objective or constraints in the RHC optimization problem and immediately see the effects in simulation. Parser-solvers are very convenient, but can lead to slow solve times, since each problem instance is parsed and transformed separately. During RHC controller development, however, solver speed is generally not a critical issue; if necessary, simulations can run slower than the required sample rate.

3.2 Implementing an RHC controller

The RHC solver implementation must, of course, run faster than the sample rate. If the solver speed is much faster than the sample rate, we can use a less powerful processor, or run the optimization on a processor performing other tasks. For applications with a slow sample time (measured, for example, in minutes), a parser-solver may be fast enough. However, for many applications a much faster solver is required. Even when a parser-solver is fast enough for real-time implementation, we may prefer a simpler solver, involving few external libraries, simpler memory management (for example, no dynamic memory allocation), and a known maximum execution time.

The traditional route to develop such a solver is custom code development, either using a toolbox (see [WB08], or the references in [Bem06]), or from scratch. This process is very difficult for many users, especially those without a numerical optimization background. In addition if the problem statement changes, for example by adding a new objective term, all code modifications must be made laboriously, by hand. Even small changes in the formulation can lead to dramatic changes in the code, which must be re-written and re-tested.

Another approach is automatic generation of custom solver code, directly from a high level description of the problem family. The user describes the problem to be solved in a convenient high level format; a solver *code generator* then generates *source code* for a fast custom solver of problem instances from the given family. This source code is then compiled, yielding a custom solver.

Automatic code generation typically yields a solver that is much faster and simpler than a parser-solver, for several reasons. First, many algorithm choices such as the transformation to cannonical form or elimination ordering in the linear equation solver can be made at code generation time. Second, the generated code can be split into an initialization part, where memory is allocated, and a real-time part, which involves no further memory allocation. The generated code has few branches, so the compiler can perform extensive code optimization. Finally, a hard limit on the number of iterations can be imposed, which translates into a known maximum execution time.

In the examples presented in this paper we use the automatic code generation software CVXGEN [MB10a], which incorporates all the above features of automatic code generation. CVXGEN handles problems that can be transformed to convex quadratic programs (QPs). The software generates custom code for transforming the original problem data to the QP

format, solving the QP using a primal-dual interior-point method, and transforming the solution of the QP back to the solution of the original problem.

3.3 Outline

In the remainder of the paper, we describe four typical RHC application examples. For each one we describe the model, our method for predicting future quantities, the objective and constraints, and the horizon choice. We give CVXGEN code for each one. The speed of the solvers generated by CVXGEN are collected together and reported in Table 2.

4 Examples

4.1 Pre-ordering

Problem statement. We consider the problem of meeting a fluctuating demand for a perishable commodity, by pre-ordering it with different lead times and also purchasing it on the spot market, all at (possibly) different prices. When we place an order, we will specify delivery for between 1 and n periods in the future. Faster delivery typically incurs a higher unit cost. Let $u_t \in \mathbf{R}^n_+$ represent new orders, where $(u_t)_i$ is the amount, ordered in period t, to be delivered in period $t+i$. Our state will be the order book $x_t \in \mathbf{R}^n_+$, where $(x_t)_i$ is the quantity scheduled to arrive in period $t+i-1$; in particular, $(x_t)_1$ is the stock at hand. The system dynamics are $x_{t+1} = Ax_t + Bu_t$, where A is a matrix with ones on the upper diagonal and zeros everywhere else, and $B = I$. The constraint has the form $u_t \geq 0$, which is convex. (In our general model, we take $\mathcal{C}_t = \mathbf{R}^n \times \mathbf{R}^n_+$.) Thus, in this example, there is no uncertainty in the dynamics or constraints.

Our stage cost has two terms: The cost of placing orders for future delivery (which we recognize immediately), and the cost of making up any unmet demand by purchasing on the spot market. The first term has the form $p_t^T u_t$, where $(p_t)_i \geq 0$ is the price of ordering one unit of the commodity for delivery in period $t+i$. The unmet demand is $(d_t - (x_t)_1)_+$, where $d_t \geq 0$ is the demand in period t, and $(\cdot)_+$ denotes the positive part. The cost of meeting the excess demand on the spot market is $p_t^{\text{spot}}(d_t - (x_t)_1)_+$, where $p_t^{\text{spot}} \geq 0$ is the spot market price at time t. Thus the overall stage cost is

$$\ell_t(x_t, u_t) = p_t^T u_t + p_t^{\text{spot}}(d_t - (x_t)_1)_+,$$

which is a convex function of x_t and u_t. Typically the prices satisfy $p_t^{\text{spot}} > (p_t)_1 > \cdots > (p_t)_n$, *i.e.*, there is a discount for longer lead time.

We consider the simple case in which the pre-order and spot market prices are known and do not vary with time (*i.e.*, $p_t = p \in \mathbf{R}^n_+$, $p_t^{\text{spot}} = p^{\text{spot}} \geq 0$), and demand is modeled as a stochastic process. We assume that demand is a stationary log-normal process, *i.e.*, $\log d_t$ is a stationary Gaussian process with

$$\mathbf{E} \log d_t = \mu, \qquad \mathbf{E}((\log d_t - \mu)(\log d_{t+\tau} - \mu)) = r_\tau,$$

so the mean demand is $\mathbf{E}\, d_t = \exp(\mu + r_0/2)$.

In period t, the controller has access to the current order book x_t, and the current and last N values of demand, $d_t, d_{t-1}, \ldots, d_{t-N}$, in addition to the various constants: the prices p and p^{spot}, the log demand mean μ, and the log demand autocovariances r_τ. The orders made in period t are based on this information.

Receding horizon policy. Our RHC policy requires estimates of future stage cost, which depends on the (unknown) future demand. We will take

$$\hat{d}_{\tau|t} = \exp \mathbf{E}(\log d_\tau | d_t, \ldots, d_{t-N}),$$

i.e., the exponential of the conditional mean of log demand, given the previous N demand values. (We know the current demand, so we can take $\hat{d}_{t|t} = d_t$.) Since we have assumed the demand is a stationary log-normal process, the conditional expectation of $\log d_\tau$ is an affine (linear plus constant) function of $\log d_t, \ldots, \log d_{t-N}$:

$$\hat{d}_{\tau|t} = \exp\left(a_{\tau-t}^T(\log d_t, \ldots, \log d_{t-N}) + b\right), \ldots,$$

for $\tau = t+1, \ldots, t+T$, where $a_j \in \mathbf{R}^{N+1}$ and $b \in \mathbf{R}$ can be found from the data μ and $r_0, \ldots, r_{N+T+1}$.

For this example we will also add a terminal constraint, $\mathbf{1}^T \hat{x}_{t+T+1} = n\,\mathbf{E}\,d_t$, where $\mathbf{E}\,d_t = \exp(\mu + r_0/2)$. This ensures we do not myopically reduce cost by exhausting inventory at the end of the horizon. The RHC optimization problem (1) becomes

$$\begin{array}{ll} \text{minimize} & \frac{1}{T+1}\sum_{\tau=t}^{t+T} p^T \hat{u}_\tau + p^{\text{spot}}(\hat{d}_{\tau|t} - (\hat{x}_\tau)_1)_+ \\ \text{subject to} & \hat{x}_{\tau+1} = A\hat{x}_\tau + \hat{u}_\tau, \quad \tau = t, \ldots, t+T \\ & \hat{u}_\tau \geq 0, \quad \tau = t, \ldots, t+T \\ & \mathbf{1}^T \hat{x}_{t+T+1} = n\,\mathbf{E}\,d_t, \quad \hat{x}_t = x_t, \end{array}$$

with variables $\hat{x}_t, \ldots, \hat{x}_{t+T+1}$ and $\hat{u}_t, \ldots, \hat{u}_{t+T}$. This is a convex optimization problem, and can be reduced to a linear program (LP).

Constant order policy. We will compare the RHC policy with a simple policy: At each time t, we let $u_t = (0, \ldots, 0, \bar{u})$, *i.e.*, we order a constant amount with the maximum delivery time. We will use $\bar{u} = \mathbf{E}\,d_t = \exp(\mu + r_0/2)$, *i.e.*, we order with maximum lead-time (presumably, at the lowest price) an amount equal to the average demand.

4.1.1 Related work

Much work has been done on supply chain planning. For an overview of the field, though without the optimization component, see [Mil02]. For the application of RHC to the supply chain, see [SWR06], or [CTHK03] which covers multi-factory supply chains. In [BP00], the authors use extensive simulation of MPC to test the sensitivity of various policies, while [MTA06] explores various levels of decentralization. Finally, for supply chain optimization with mixed-integer constraints see [PLYG03], and for planning under uncertainty see [GM03].

4.1.2 Numerical example

Our example has $n = 5$ order lead times, with prices

$$p_{\text{spot}} = 1, \qquad p = (\gamma, \gamma^2, \gamma^3, \gamma^4, \gamma^5),$$

with $\gamma = 0.7$. (Thus, we get a constant 30% discount for each period of lead time.) The demand process data are $\mu = 0$, and $r_\tau = 0.1(0.95^\tau)$. Our RHC controller will use horizon $T = 30$, and we estimate future demand using the last $N = 100$ demands.

```
dimensions
  T = 30; n = 5
end

parameters
  A (n,n); p (n,1)
  d[t], t=0..T
  pspot positive; ubar
  x[0] (n)
end

variables
  x[t] (n), t=1..T+1
  u[t] (n), t=0..T
end

minimize
  (1/(T+1))*sum[t=0..T](p'*u[t]
   + pspot*pos(d[t] - x[t][1]))
subject to
  x[t+1] == A*x[t] + u[t], t=0..T
  u[t] >= 0, t=0..T
  sum(x[T+1]) == n*ubar
end
```

Figure 1: CVXGEN code segment for the pre-order example.

Results. We simulate both the RHC and constant ordering policies for 1000 steps (with the same demand realization). The constant order policy incurs an average cost $J = 0.37$, while, as expected, the RHC policy performs considerably better, with an average cost $J = 0.28$. Some example trajectories are shown in Figure 2. We compare the costs incurred by the RHC policy (blue) and constant policy (red), over 500 time steps. The plots show demand, pre-order cost and spot market costs, and overall stage cost.

In Figure 3 we show the log-demand trajectories for a selected time frame. The vertical lines show $\exp(\log \hat{d}_{t|220} \pm \sigma_t)$, where $\sigma_t = (\mathbf{E}(\log d_t - \log \hat{d}_{t|220}))^{1/2}$. We see that while the predicted trajectory captures the general trend, the prediction error is large. Nevertheless, RHC performs very well, even with inaccurate predictions.

The CVXGEN code takes up to 250 μs to solve at each time step, which is 4000$\times$ faster than with plain CVX. Although this speed is much faster than needed for many applications, the extra speed is useful for testing different scenarios and ordering strategies, which require extensive Monte-Carlo simulation. Computational performance details are collected in Table 2.

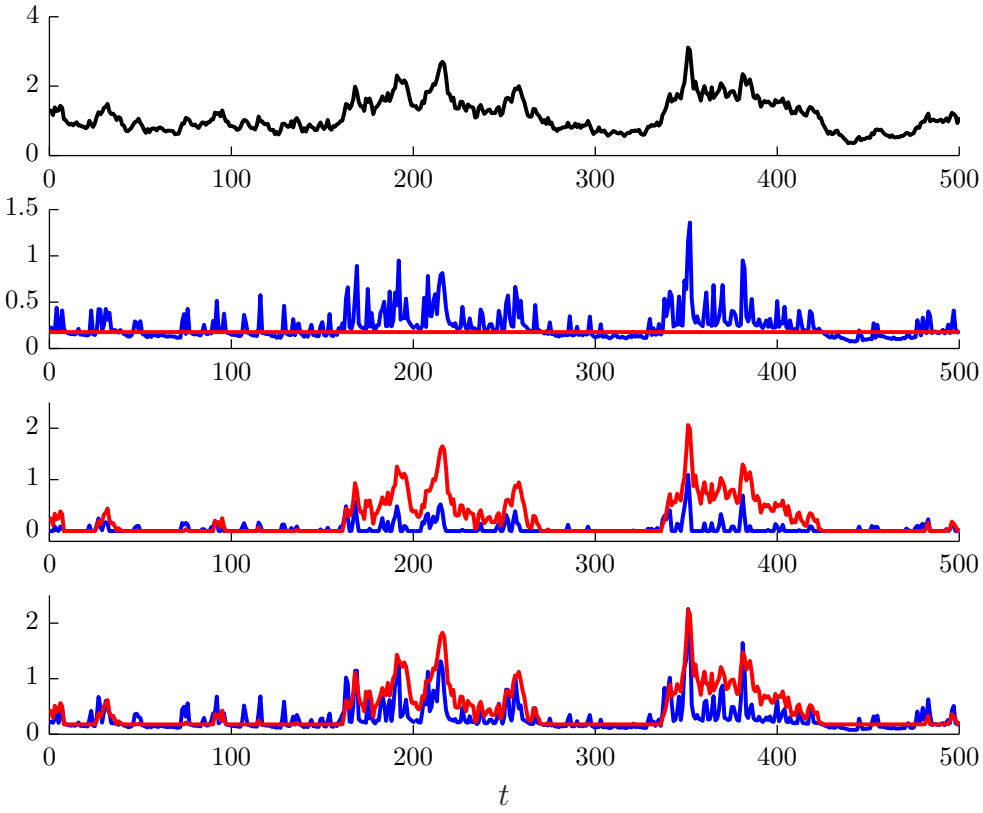

Figure 2: Comparison of RHC policy (blue) and constant order policy (red) for the pre-order example. From top to bottom: demand (d_t), pre-order cost ($p^T u_t$), spot market cost ($p^{\text{spot}}(d_t - (x_t)_1)_+$), and stage cost ($\ell(x_t, u_t)$).

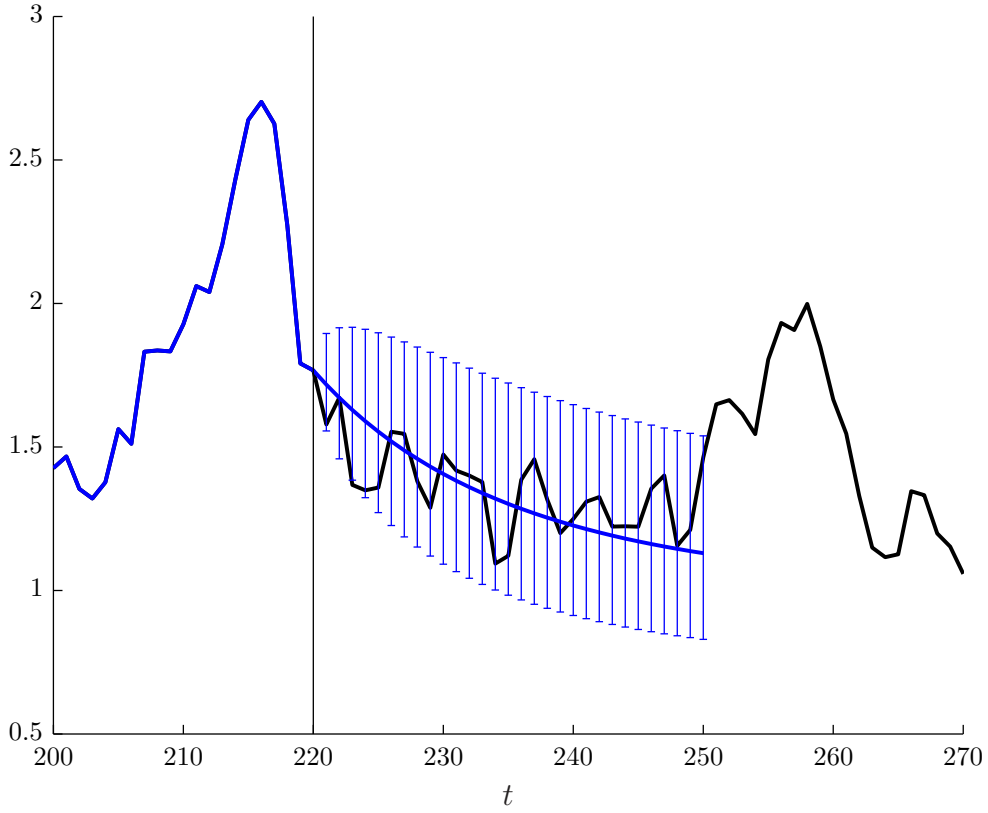

Figure 3: Black: $\log d_t$; Blue: $\log \hat{d}_{t|220}$ for the pre-order example. Vertical lines show prediction error.

4.2 Active suspension control

Problem statement. We consider an active vehicle suspension system, where the control system applies force to a suspension system in real-time, using a preview of the terrain height profile obtained from sensors or maps. For simplicity, we consider a scalar problem, with a single, scalar vehicle height, and one actuator. (A more accurate model would incorporate different heights and actuator forces for each wheel, and more complicated dynamic coupling between them.)

Let $h_t \in \mathbf{R}$ denote the terrain height, $y_t \in \mathbf{R}$ the vehicle height, and $e_t = y_t - h_t$ the suspension extension, at discrete time period t. The extension is offset, with the suspension at rest when $e_t = 0$. The suspension has minimum extension $E^{\min} < 0$ and maximum extension $E^{\max} > 0$, so we always have

$$E^{\min} \leq e_t \leq E^{\max},$$

The vehicle dynamics are

$$x_{t+1} = Ax_t + bu_t + v_t,$$

where $A \in \mathbf{R}^{n\times n}$ and $b \in \mathbf{R}^n$ are known dynamic model coefficients, $x_t \in \mathbf{R}^n$ is the vehicle dynamic state, $v_t \in \mathbf{R}^n$ is the exogeneous force applied to the vehicle by the changing terrain, and $u_t \in \mathbf{R}$ is the active suspension actuator force, which must satisfy

$$F^{\min} \leq u_t \leq F^{\max},$$

where $F^{\min} < 0$ and $F^{\max} > 0$ are given minimum and maximum active suspension forces. The vehicle's height y_t and vertical acceleration a_t are given by

$$y_t = c^T x_t, \quad a_t = d^T x_t + g^T u_t + w_t,$$

where c, d and $g \in \mathbf{R}^n$ are known model coefficients, and $w_t \in \mathbf{R}$ is a linear function of the terrain height and gradient at time t.

Our goal is to minimize a weighted sum of the squared acceleration, the squared active suspension force, and a penalty term that returns the suspension to equilibrium. This can be thought of as a measure of the 'ride roughness', penalized to avoid excessive suspension effort. Our cost function is

$$\ell(x_t, u_t) = a_t^2 + \rho u_t^2 + \mu e_t^2,$$

where the parameters $\rho > 0$ and $\mu > 0$ control the effort and extension penalties.

Receding horizon policy. At time t, the controller has access to an estimate of the current vehicle state $\hat{x}_{t|t}$, and a preview of upcoming terrain, *i.e.*, $\hat{h}_{\tau|t}$ for $\tau = t, \ldots, t+L$, where L is the look-ahead time. It also has access to exogenous input estimates $\hat{v}_{\tau|t}$ and $\hat{w}_{\tau|t}$, which are formed using the terrain height estimates, along with estimates of the terrain's gradient. The actuator force to apply at the current time step, u_t, is determined from these data. In our RHC formulation, we add the terminal constraint $x_{t+T+1} = 0$, which means

our suspension returns to a neutral position at the end of the look-ahead interval. The RHC optimization problem (1) becomes

$$\begin{array}{ll}\text{minimize} & \frac{1}{T+1}\sum_{\tau=t}^{t+T}\left(a_\tau^2+\rho u_\tau^2+\mu(y_\tau-\hat{h}_{\tau|t})^2\right)\\ \text{subject to} & x_{\tau+1}=Ax_\tau+bu_\tau+\hat{v}_{\tau|t},\quad y_\tau=c^Tx_\tau,\quad a_\tau=d^Tx_\tau+gu_\tau+\hat{w}_{\tau|t},\\ & E^{\min}\le y_\tau-\hat{h}_{\tau|t}\le E^{\max},\quad F^{\min}\le u_\tau\le F^{\max},\quad \tau=t,\ldots t+T\\ & x_{t+T+1}=0,\quad x_t=\hat{x}_{t|t},\end{array}$$

with variables $x_\tau,\ldots,x_{\tau+T}\in\mathbf{R}^n$, $y_\tau,\ldots,y_{\tau+T}\in\mathbf{R}$, $a_\tau,\ldots,a_{\tau+T}\in\mathbf{R}$, and $u_\tau,\ldots,u_{\tau+T}\in\mathbf{R}$. This is a convex optimization problem, and can be solved as a QP.

Uncontrolled policy. We will compare the RHC policy to the performance of the uncontrolled mass-spring-damper system.

4.2.1 Related work

Active suspension control is used in a range of production vehicles, although with less sophisticated control algorithms [Wil97]. In [MAH+97], a similar model that also incorporates an unsprung mass is used, and the problem solved as a QP. A number of authors consider semi-active suspension to reduce cost, including [GBTH05] and [CMN06], which use MPC to control passive damping, and [DSL05], which uses H_∞ control.

4.2.2 Numerical example

Our example has a sampling rate of 20 Hz, and a horizon $T=20$, which corresponds to a one-second lookahead. To determine A, b, c, d and g, we use a quarter-vehicle model, with mass 2000 kg, damping coefficient 5 kNs/m and spring constant 30 kN/m. This gives a natural frequency of 0.6 Hz, and a light damping coefficient of 0.3. We assume forces are held constant during each time interval, and use an exact discretization via the matrix exponential. We set the negative and positive extension limits to $E^{\min}=-0.1$ and $E^{\max}=0.1$ m, and restrict our maximum actuator force to 3 kN. We set $\rho=10^{-4}$, and $\mu=5$. Figure 4 shows CVXGEN code for this example.

Results. Figure 5 shows vehicle and suspension behavior for a 5-second journey over terrain with a single, large bump. The RHC control system reduces the the vehicle's root-mean-square acceleration value by approximately 70%. Note that the uncontrolled system exceeds the extension limits twice, which is avoided by the RHC controller.

The CVXGEN code always takes less than 1 ms to solve at each time step, even on a low-power (2 W) processor. This is much faster than required for the 20 Hz sample rate. The extra speed means that RHC could be one of multiple jobs handled on a single computer with a suitable real-time operating system. Further performance details are collected in Table 2.

```
dimensions
  n = 2
  T = 20
end

parameters
  A (n,n); B (n,1)
  C (1,n); D

  h[t], t=0..T+1
  v[t] (n), t=0..T
  w[t], t=0..T

  rho positive; mu positive
  Fmin; Fmax;  Emin; Emax
  x[0] (n)
end

variables
  u[t], t=0..T
  x[t] (2), t=1..T+1
  a[t], t=0..T
end

minimize
  sum[t=0..T](square(a[t]) + rho*square(u[t]) + mu*square(x[t][1] - h[t]))
subject to
  x[t+1] == A*x[t] + B*u[t] + v[t], t=0..T
  a[t] == C*x[t] + D*u[t] + w[t], t=0..T
  Fmin <= u[t] <= Fmax, t=0..T
  Emin <= x[t][1] - h[t] <= Emax, t=1..T
end
```

Figure 4: CVXGEN code segment for the suspension example.

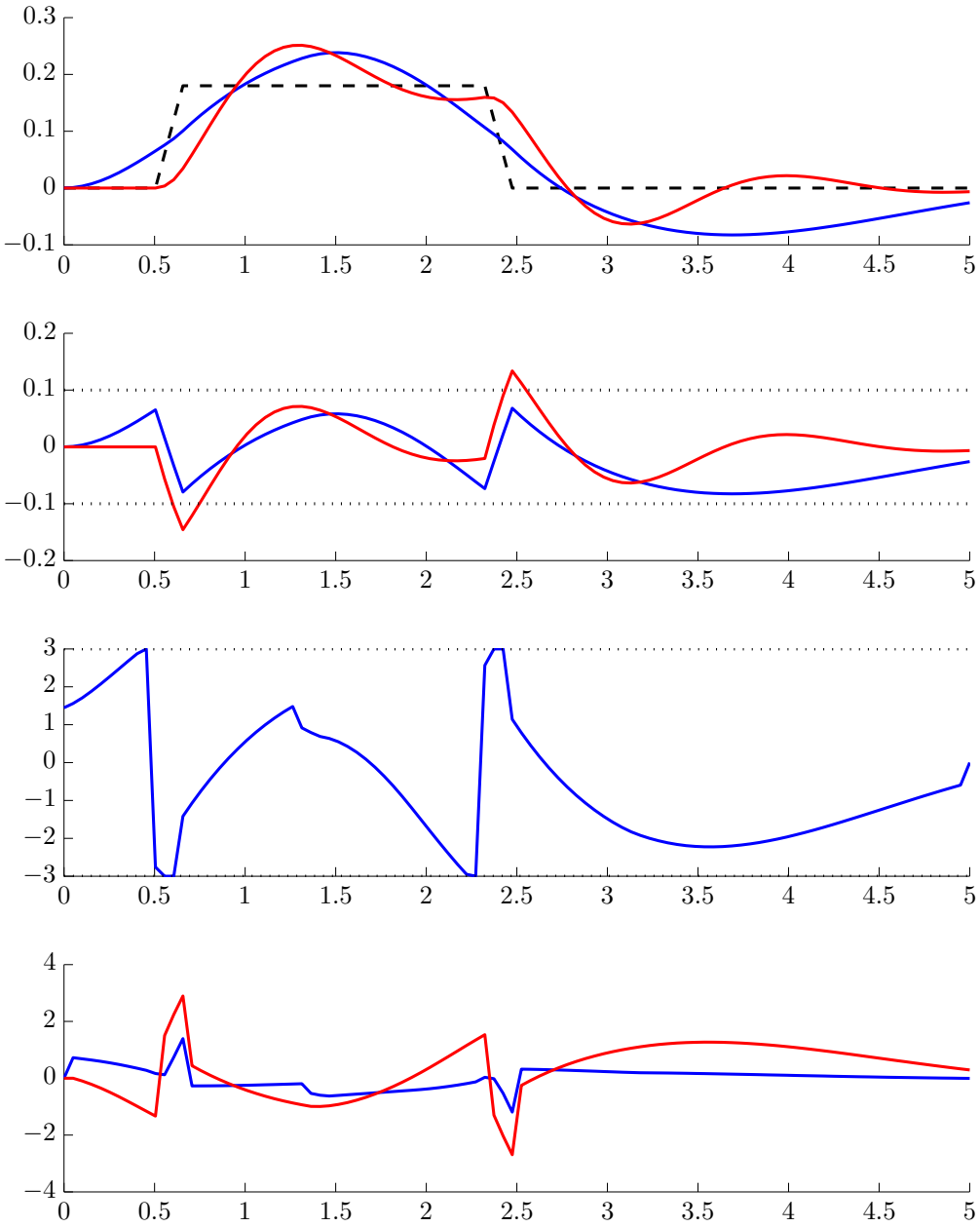

Figure 5: (From top) Vertical position (m), suspension extension (m), suspension force (kN) and acceleration (ms^{-2}). RHC (blue), open loop (red). Dashed lines indicate the terrain profile (top) and constraints.

4.3 Processor speed control

Problem statement. In this example, a single processor handles jobs from a set of n queues. At each time step the processor adjusts the work rate for each queue. The total work rate determines the processor (clock) speed, which in turn determines the power dissipated by the processor. The goal is to adjust the rates to optimally balance average processor power dissipation and queue length.

We use a discrete-time formulation, with state $x_t \in \mathbf{R}^n_+$ and input $u_t \in \mathbf{R}^n_+$, where $(x_t)_i$ is the amount of work to be done in queue i, and $(u_t)_i$ is the work rate (expressed as effective speed) for queue i, at time t. The dynamics are $x_{t+1} = x_t - u_t + a_t$, where $a_t \in \mathbf{R}^n_+$ denotes the new work arriving in each queue between time periods t and $t+1$. At each time we cannot process more than the available work in each queue, so we must have $u_t \leq x_t$. The total work rate of the processor, over all queues, is $\mathbf{1}^T u_t$.

The processor speed at time t is a function of the work rate vector u_t:

$$s_t = \max\{S^{\min}, \mathbf{1}^T u_t\},$$

where $S^{\min}$ is the minimum allowed processor speed. The processor has a maximum allowed processor speed, $s_t \leq S^{\max}$, which translates to the constraint $\mathbf{1}^T u_t \leq S^{\max}$. The processor power dissipation is modeled as αs_t^2, where $\alpha > 0$.

With each queue we associate a quadratic-plus-linear cost $c_i(x_t)_i + d_i(x_t)_i^2$, where c_i and d_i are positive weights. We can interpret c_i as relative queue priorities, when the queues are small, and c_i/d_i as the queue length at which the cost is twice the linear cost alone. When the queue lengths have some probability distribution, the expected value of the queue cost is c_i times the mean queue length, plus d_i times the mean square queue length.

The overall stage cost is

$$\ell(x_t, u_t) = \alpha \max\{S^{\min}, \mathbf{1}^T u_t\}^2 + c^T x_t + d^T x_t^2,$$

where x_t^2 is interpreted elementwise. This is a convex function.

For this example the dynamics matrices, constraints, and stage costs are all known. The only uncertainty is the arrivals a_t, which we assume has the form

$$(a_t)_i = \exp(\lambda_i \sin(2\pi t/M - \theta_i) + (w_t)_i), \quad i = 1, \ldots, n,$$

where M is the period, λ_i, θ_i are known constants, and w_t is IID gaussian with mean μ and covariance Σ.

In period t, the controller chooses the work rates u_t based on knowledge of the current state x_t, as well as the data $S^{\min}$, $S^{\max}$, α, a, b, λ, θ, μ Σ, M.

Receding horizon policy. In the RHC policy, our estimates of the arrivals are

$$(\hat{a}_{\tau|t})_i = \mathbf{E}(a_t)_i = \exp(\lambda_i \sin(2\pi t/M - \theta_i) + \mu_i + 0.5\Sigma_{ii}), \quad i = 1, \ldots, n, \quad \tau = t, \ldots, t+T.$$

The RHC optimization problem becomes

$$\begin{array}{ll} \text{minimize} & \frac{1}{T+1}\sum_{\tau=t}^{t+T} \alpha \max\{S^{\min}, \mathbf{1}^T\hat{u}_\tau\}^2 + c^T\hat{x}_t + d^T\hat{x}_t^2 \\ \text{subject to} & \hat{x}_{\tau+1} = \hat{x}_\tau - \hat{u}_\tau + \hat{a}_{\tau|t}, \quad \tau = t, \ldots, t+T \\ & 0 \le \hat{u}_\tau \le \hat{x}_\tau, \quad \mathbf{1}^T\hat{u}_\tau \le S^{\max}, \quad \tau = t, \ldots, t+T \\ & \hat{x}_t = x_t, \end{array} \tag{2}$$

where the variables are $\hat{x}_t, \ldots, \hat{x}_{t+T+1}$, $\hat{u}_t, \ldots, \hat{u}_{t+T}$. This is a convex optimization problem, which can be transformed into a QP.

Proportional policy. A simple policy is to set the work rates to be proportional to the amount of work left in each queue. Specifically, we take

$$(u_t)_i = \min\{(x_t)_i, ((x_t)_i/\mathbf{1}^T x_t)S^{\max}\}.$$

Here we take the minimum to ensure $u_t \le x_t$ is satisfied. The constraint $s_t \le S^{\max}$ is also satisfied by this policy.

4.3.1 Related work

For an overview of power-aware design, see one of the survey papers [IP05, SSH+03, DM06]. Closely related work appears in [WAT09], which uses a dynamic speed scaling scheme, motivated by queueing theory, to balance energy consumption and mean response time in a multi-processor system. The problem is formulated as a stochastic dynamic program, with an upper bound used instead of an exact solution. In [MBM+09] the authors consider a related problem, where the goal is to maximize processing speed while respecting system temperature limits.

4.3.2 Numerical instance

We consider a simple numerical example with $n = 3$ queues, and problem data

$$S^{\min} = 1, \quad S^{\max} = 5, \quad \alpha = 2, \quad c = (1, 1, 1), \quad d = (0.5, 0.5, 0.5),$$

and

$$\lambda = (3, 3.5, 3.2), \quad \theta = (0, 1, 2), \quad \mu = (-2, -2, -2), \quad \Sigma = \mathbf{diag}((0.04, 0.04, 0.04)).$$

Typical arrivals trajectories are shown in Figure 6. For the RHC policy we will use horizon $T = 30$.

Results. We simulate both policies for 1000 time steps (with the same arrival realization). The RHC policy incurs an average cost of $J = 71.3$, while the proportional policy achieves $J = 95.3$, which is around 34% worse. Figure 8 shows some sample trajectories. We compare

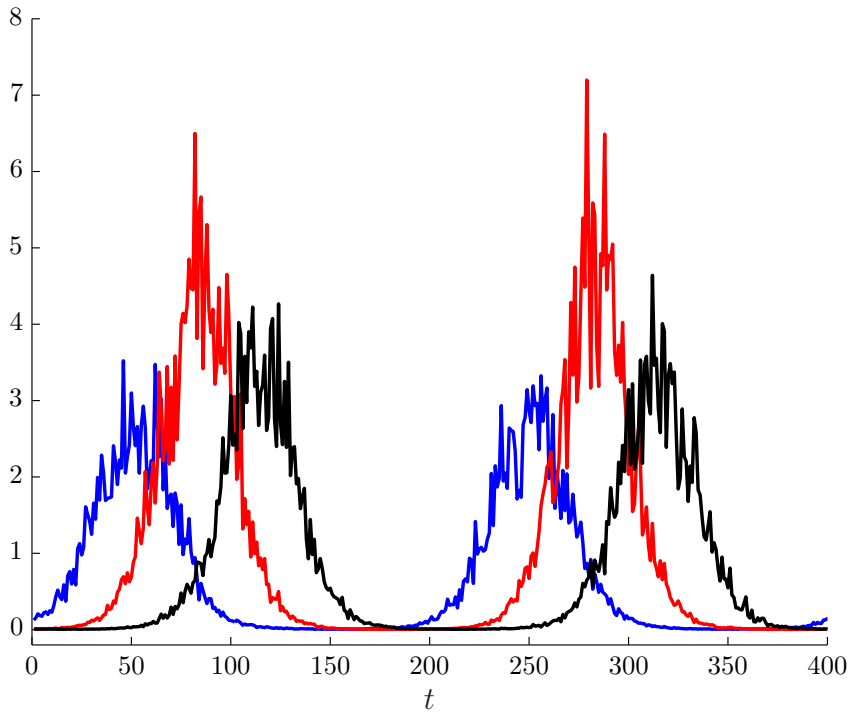

Figure 6: Sample trajectories for $(a_t)_1$ (blue), $(a_t)_2$ (red), and $(a_t)_3$ (black).

the RHC policy (blue) with the simple proportional policy (red). The plots are (from top to bottom): $(x_t)_1$, $(x_t)_2$, $(x_t)_3$, and stage cost $\ell(x_t, u_t)$.

The CVXGEN code takes at most 0.9 ms to solve at each time step, which is 5000× faster than with CVX. This means that a dedicated processor can adjust work rates at 1 kHz using RHC. Alternatively, the same processor could use 1% of its processing power to adjust its own rates at 10 Hz. Further computational performance details are collected in Table 2.

```
dimensions
  n = 3; T = 30
end

parameters
  ahat[t] (n), t=0..T
  alpha positive
  lambda positive
  c (n)
  D (n,n) psd diagonal
  x[0] (n)
  Smin positive
  Smax positive
end

variables
  x[t] (n), t=1..T+1
  u[t] (n), t=0..T
end

minimize
  alpha*sum[t=0..T](sum(square( pos(max(Smin, sum(u[t])))  )))
   + sum[t=0..T+1](c'*x[t]) + sum[t=0..T+1](quad(x[t], D))
   + lambda*sum[t=0..T](sum(square(u[t])))
subject to
  x[t+1] == x[t] - u[t] + ahat[t], t=0..T
  u[t] >= 0, t=0..T
  u[t] <= x[t], t=0..T
  sum(u[t]) <= Smax, t=0..T
end
```

Figure 7: CVXGEN code segment for the processor speed control example.

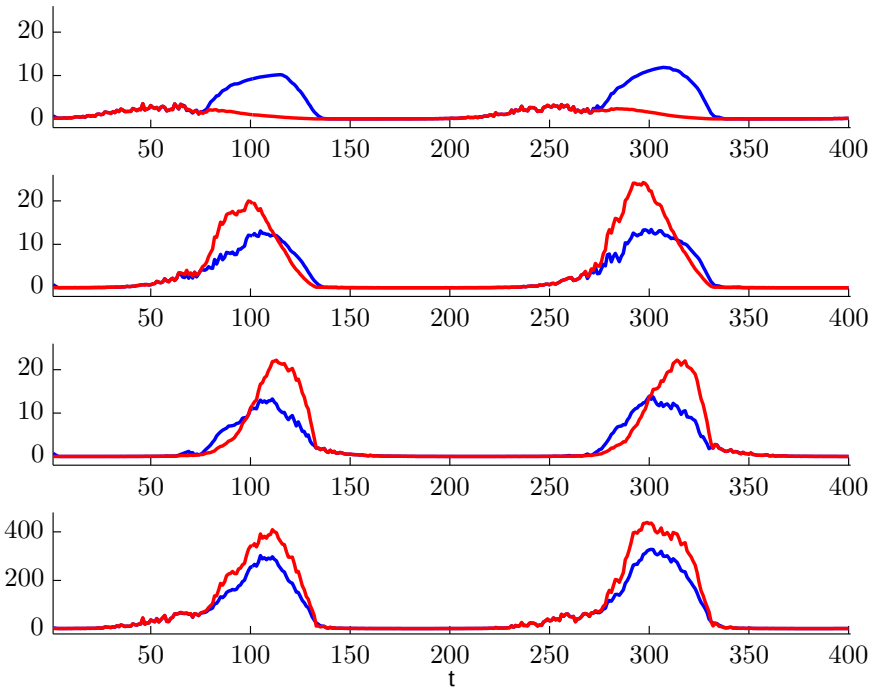

Figure 8: Sample trajectories for processor speed control example. From top to bottom: $(x_t)_1$, $(x_t)_2$, $(x_t)_3$, and stage cost $\ell(x_t, u_t)$, for RHC policy (blue) and proportional policy (red).

4.4 Energy storage

Problem statement. We consider an energy storage system that can be charged or discharged from a source with varying energy price. A simple example is a battery connected to a power grid. The goal is to alternate between charging and discharging in order to maximize the average revenue.

Let $q_t \geq 0$ denote the charge in the energy store at time period t. The energy store has capacity C, so we must have $q_t \leq C$. We let $u_t^c \geq 0$ denote the amount of energy taken from the source in period t to charge the energy store, and we let $u_t^d \geq 0$ denote the amount of energy discharged into the source from our energy store. (For the problem we consider, at most one of these will be positive; that is, we will never charge and discharge the store simultaneously.) The charging and discharging rates must satisfy

$$u_t^c \leq C^{\max}, \quad u_t^d \leq D^{\max},$$

where $C^{\max}$ and $D^{\max}$ are the maximum charge/discharge rates.

Charging increases the energy in our store by $\kappa^c u_t$, where $\kappa^c \in (0,1)$ is the charge efficiency; discharging decreases the energy in our store by u_t/κ^d, where $\kappa^d \in (0,1)$ is the discharge efficiency. In each time period the energy store leaks, losing energy proportional to its charge, with leakage coefficient $\eta \in (0,1)$. Incorporating all these effects, the system dynamics are

$$q_{t+1} = \eta q_t + \kappa^c u_t^c - u_t^d/\kappa^d.$$

In the context of our general framework, the dynamics matrices are $A = \eta$ and $B = (\kappa^c, 1/\kappa^d)^T$, with $u_t = (u_t^c, u_t^d)$.

The revenue in period t is given by $p_t(u_t^d - u_t^c)$, where p_t is the energy price at time t. To discourage excessive charging and discharging, we add a penalty of the form $\gamma(u_t^c + u_t^d)$, where $\gamma \geq 0$ is a parameter. (An alternative interpretation of this term is a transaction cost, with bid-ask spread γ: We buy energy at price $p_t + \gamma$, and sell energy back at price $p_t - \gamma$.) Our stage cost (*i.e.*, negative revenue, to be minimized) is thus

$$\ell_t(q_t, u_t) = p_t(u_t^c - u_t^d) + \gamma(u_t^c + u_t^d) = (p_t + \gamma)u_t^c - (p_t - \gamma)u_t^d,$$

which can be interpreted as the profit, at time t.

We will model the energy price as a stationary log-normal process with

$$\mathbf{E} \log p_t = \mu, \quad \mathbf{E}(\log p_t - \mu)(\log p_{t+\tau} - \mu) = r_\tau.$$

At time period t the controller has access to the current charge level q_t, the data C, $C^{\max}$, $D^{\max}$, κ^c, κ^d, η, γ, the current and last N prices $p_t, p_{t-1}, \ldots, p_{t-N}$, as well as the mean and autocovariance, μ and r_τ. The future prices are not known.

Receding horizon policy. To implement the receding horizon policy, we take our estimates of the future prices to be

$$\hat{p}_{\tau|t} = \exp \mathbf{E}(\log p_\tau | p_t, \ldots, p_{t-N}), \quad \tau = t+1, \ldots, t+T,$$

which is an affine function of $\log p_t, \ldots, \log p_{t-N}$. (Note that this is not the same as $\mathbf{E}(p_\tau | p_t, \ldots, p_{t-N})$, which can also be computed and used as estimates of future prices.) Our estimates of the stage costs are

$$\hat{\ell}_t(\hat{q}_\tau, \hat{u}_\tau) = (\hat{p}_{\tau|t} + \gamma)\hat{u}^c_\tau - (\hat{p}_{\tau|t} - \gamma)\hat{u}^d_\tau.$$

Thus, the RHC optimization problem becomes

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{t+T} \hat{\ell}_t(\hat{q}_\tau, \hat{u}_t) \\ \text{subject to} & \hat{q}_{\tau+1} = \eta\hat{q}_\tau + \kappa^c\hat{u}^c_\tau - \hat{u}^d_\tau/\kappa^d, \\ & 0 \le \hat{u}^c_\tau \le C^{\max}, \quad 0 \le \hat{u}^d_\tau \le D^{\max}, \\ & \quad \tau = t, \ldots, t+T \\ & 0 \le \hat{q}_\tau \le C, \quad \tau = t, \ldots, t+T+1 \\ & \hat{q}_\tau = q_t, \end{array} \tag{3}$$

with variables $\hat{q}_t, \ldots, \hat{q}_{t+T+1}$, $\hat{u}^c_t, \ldots, \hat{u}^c_{t+T}$, $\hat{u}^d_t, \ldots, \hat{u}^d_{t+T}$. This is a convex optimization problem, and can be written as an LP.

Thresholding policy. We will compare the receding horizon policy with a simple thresholding policy, which works as follows:

$$u^c_t = \begin{cases} \min(C^{\max}, C - q) & p_t \le p_{\text{thc}} \\ 0 & \text{otherwise,} \end{cases} \qquad u^d_t = \begin{cases} \min(D^{\max}, q) & p_t \ge p_{\text{thd}} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, we charge at the maximum rate if the price is below a threshold p_{thc}, and we discharge at the maximum rate if the price is above a threshold p_{thd}. If the price is in between we do nothing. We take the minimum to ensure we do not charge above the capacity or discharge below zero.

4.4.1 Related work

In [HNPWHH07], the authors consider a distributed energy system where individual grid-connected households use an RHC controller to control micro combined heat and power plants. For more on distributed generation and variable pricing, see, respectively, [HNNS06] and [Bra05]. On the generation side, [KS09] applies MPC to wind turbinees with batteries in order to smooth the power produced. The paper includes a case study with real data. A related application is to hybrid vehicles, where multiple power sources are available. See [KY06] or [PBK$^+$07], or for a vehicle with multiple different energy storage units see [WBS03].

4.4.2 Numerical example

We look at a particular numerical instance with $\eta = 0.98$, $\kappa^c = 0.98$, $\kappa^d = 0.98$, $C^{\max} = 10$, $D^{\max} = 10$, $C = 50$, $\gamma = 0.02$, $q_0 = 0$, $\mu = 0$, $r_\tau = 0.1(0.99^\tau \cos(0.1\tau))$. For the receding horizon policy we used a time horizon of $T = 50$ steps, and $N = 100$ previous prices to estimate future prices.

```
dimensions
  T = 50
end

parameters
  eta; kappac; kappad; Cmax; Dmax
  gamma; C; p[t], t=0..T; q[0]
end

variables
  q[t], t=1..T+1
  uc[t], t=0..T
  ud[t], t=0..T
end

minimize
  sum[t=0..T]((p[t] + gamma)*uc[t] - (p[t]
   - gamma)*ud[t])
subject to
  q[t+1] == eta*q[t] + kappac*uc[t]
   - (1/kappad)*ud[t], t=0..T
  0 <= q[t] <= C, t=1..T+1
  0 <= uc[t] <= Cmax, t=0..T
  0 <= ud[t] <= Dmax, t=0..T
end
```

Figure 9: CVXGEN code segment for the storage example.

Results. The simulations were carried out for 1000 time steps. Figure 10 shows the cumulative profit,

$$r_t = \sum_{\tau=0}^{t} p_\tau(u_\tau^d - u_\tau^c) - \gamma(u_\tau^d + u_\tau^c),$$

for the RHC policy (blue) and the simple thresholding policy (red), over 500 time steps. For the thresholding policy, we adjusted the charge/discharge thresholds via trial and error to achieve good performance. The final thresholds we used are $p_{\text{thc}} = 0.8$, $p_{\text{thd}} = 1.3$. Clearly, the RHC policy outperforms the thresholding policy. The average profit achieved for the RHC policy is 0.23 per-period, whereas thresholding achieves a profit of 0.029 per-period (averaged over 1000 time steps).

Figure 11 shows the actual (black) and predicted (blue) log-price trajectories starting at $t = 150$. The vertical lines show $\exp(\log \hat{p}_{t|150} \pm \sigma_t)$, where $\sigma_t = (\mathbf{E}(\log p_t - \log \hat{p}_{t|150}))^{1/2}$. The CVXGEN code takes up to 360 μs to solve at each time step, which is 3500$\times$ faster than with CVX. Further computational performance details are collected in Table 2.

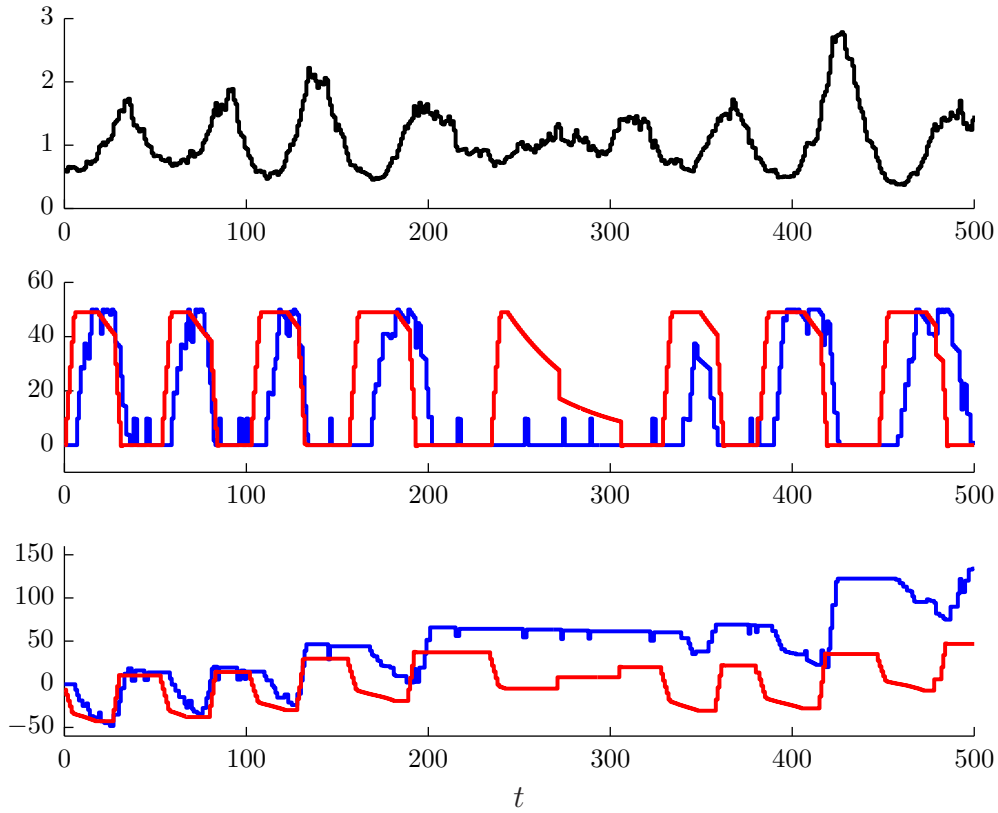

Figure 10: Comparison of RHC policy (blue) and thresholding policy (red) for the storage example. From top to bottom: price (p_t), charge (q_t), cumulative profit (r_t).

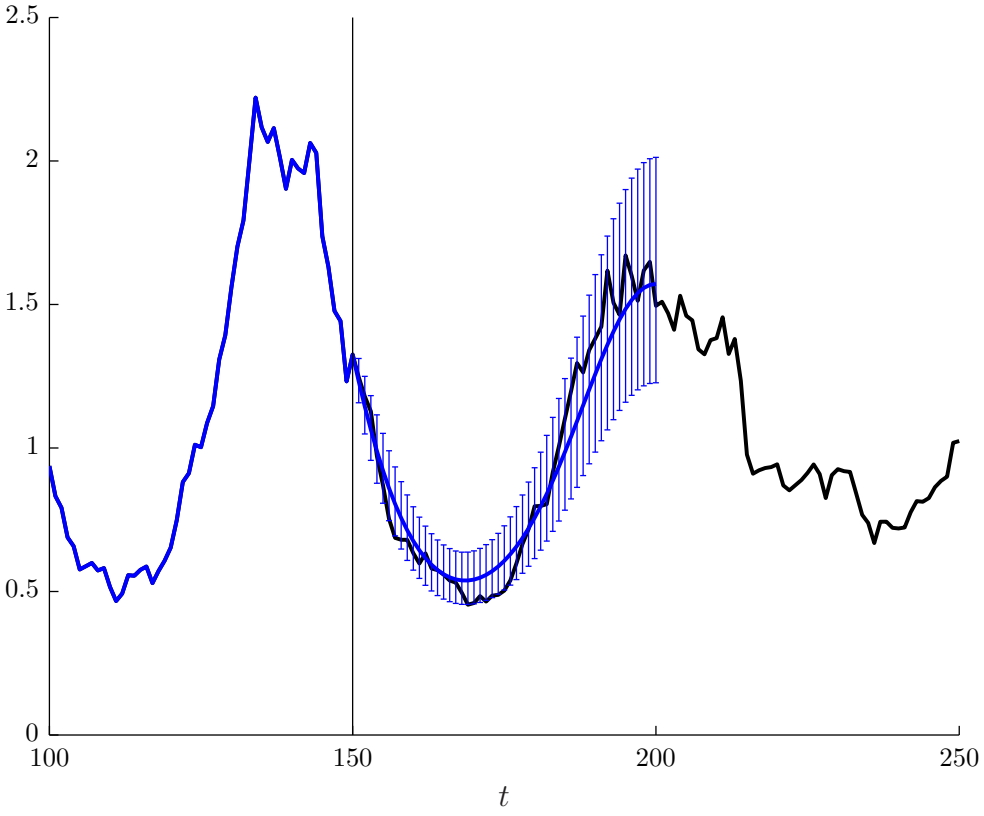

Figure 11: Black: $\log p_t$; Blue: $\log \hat{p}_{t|150}$ for the storage example. Vertical lines show prediction error.

5 CVXGEN performance

To give a rough guide to CVXGEN's performance, we test the CVXGEN code for each example on three different computers. The given timings should not be taken too seriously, since there are many things that could easily improve performance, often reducing speed by an order of magnitude or more. First, single-precision floats could be used in place of double-precision, since the scale of data is known ahead of time. Secondly, the time-horizon selected for the examples is relatively long. With a suitable choice of final state cost (see [WB09]), the horizon could be reduced, giving a speedup proportional to the horizon. Finally, we solve the problems to high accuracy, which requires up to 15-20 steps. With a small amount of tuning, adequate control performance could easily be achieveable using a fixed step limit of (say) 5 steps [WB08]. Thus, all of the numerical results should be taken as preliminary upper bounds on performance.

We summarize computer properties in Table 1. We use `gcc-4.4` on each processor, with the compiler optimization flag `-Os`.

In each case, we ensure the computer is idle, then solve the optimization problem instances continuously for at least one second. We calculate the maximum time taken to solve

	OS	Processor	Cache size	Max speed	Max power
Computer 1	Linux 2.6	Intel Atom Z530	512 kB	1.60 GHz	2 W
Computer 2	Linux 2.6	Intel Core Duo T2300	2 MB	1.66 GHz	31 W
Computer 3	OS X 10.6	Intel Core i7-860	8 MB	3.46 GHz	95 W

Table 1: Computer properties

	`preorder`	`storage`	`suspension`	`proc_speed`
Variables, original	310	153	84	112
Variables, transformed	341	153	104	279
Constraints, transformed	373	357	165	465
KKT matrix nonzeros	1116	1121	636	1960
KKT factor fill-in	1.64	1.45	2.25	1.65
Max steps required	10	16	7	19
CVXGEN, Computer 1 (ms)	2.34	4.01	0.99	7.80
CVXGEN, Computer 2 (ms)	0.96	1.98	0.39	3.64
CVXGEN, Computer 3 (ms)	0.25	0.36	0.11	0.85
CVX (ms)	970	1290	2570	4190

Table 2: CVXGEN performance

any instance, ensuring that each problem is solved to sufficient optimality so that control performance is not affected. To get a rough idea of the speed of a traditional parser solver, we also test the performance of CVX on the fastest computer, Computer 3, using Matlab 7.9 and CVX 1.2. For `preorder` and `storage` we set the solver used by CVX to Sedumi 1.2; for `suspension` and `proc_speed` we select SDPT3 4.0. All results are shown in Table 2.

6 Conclusions

In this paper we have shown that receding horizon control offers a simple and transparent method for designing feedback controllers that deliver good performance while respecting complex constraints. A designer specifies the RHC controller by specifying various components (objective, constraints, prediction method and horizon), each of which has a natural choice suggested directly by the application. In more traditional approaches, such as PID control, a designer tunes the controller coefficients, often using trial and error, to handle the objectives and constraints indirectly. A PID control designer needs much intuition and experience; an RHC designer needs far less.

In addition to the straightforward design process, we have seen that RHC controllers can be implemented in real-time at kilohertz sampling rates. These speeds are useful for both real-time implementation of the controller, as well as rapid Monte-Carlo simulation for

design and testing purposes. Thus, receding horizon control should no longer be considered a slow, computationally intensive policy. Indeed, RHC can be applied to a wide range of control problems, including applications involving fast dynamics.

With recent advances in automatic code generation, RHC controllers can now be rapidly designed and implemented. The RHC optimization problem can be specified in a high-level description language, and custom solvers for the problem family can be automatically generated by a software tool, such as CVXGEN. The generated code is optimized for the specific problem family, and is often orders of magnitude faster than a general optimization solver (such as sedumi or SDPT3). In addition, the generated code has few external library dependencies, which facilitates implementation on different real-time platforms.

We believe that receding horizon control, combined with automatic code generation, should form a new framework for designing and implementing feedback controllers. This framework will allow designers with little optimization expertise to rapidly design and implement sophisticated high-performance controllers for a wide range of real-time applications.

References

[ÅkerbladH04] M. Åkerblad and A. Hansson. Efficient solution of second order cone program for model predictive control. *International Journal of Control*, 77(1):55–77, January 2004.

[AM71] B. Anderson and J. Moore. *Linear optimal control*. Prentice Hall, 1971.

[Bem06] A. Bemporad. Model predictive control design: New trends and tools. In *Proceedings of 45th IEEE Conference on Decision and Control*, pages 6678–6683, San Diego, California, 2006.

[BF04] A. Bemporad and C. Filippi. Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming. *Journal of Optimization Theory and Applications*, 117(1):9–38, November 2004.

[BMDP02] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[BP00] S. Bose and J. F. Pekny. A model predictive framework for planning and scheduling problems: A case study of consumer goods supply chain. *Computers and Chemical Engineering*, 24(2-7):329–335, 2000.

[Bra05] S. D. Braithwait. Real-time pricing and demand response can work within limits. *Natural Gas and Electricity*, 21(11):1–9, 2005.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CMN06] M. Canale, M. Milanese, and C. Novara. Semi-active suspension control using fast model-predictive techniques. *IEEE Transactions on control systems technology*, 14(6):1034–1046, 2006.

[CTHK03] E. G. Cho, K. A. Thoney, T. J. Hodgson, and R. E. King. Supply chain planning: Rolling horizon scheduling of multi-factory supply chains. In *Proceedings of the 35th conference on Winter simulation: driving innovation*, pages 1409–1416, New Orleans, Louisiana, 2003.

[DFS+02] M. Diehl, R. Findeisen, S. Schwarzkopf, I. Uslu, F. Allgöwer, H. G. Bock, E. D. Gilles, and J. P. Schlöder. An efficient algorithm for nonlinear model predictive control of large-scale systems. Part I: Description of the method. *Automatisierungstechnik*, 50(12):557–567, 2002.

[DM06] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA'06)*, pages 78–88, 2006.

[DSL05] H. Du, K. Y. Sze, and J. Lam. Semi-active $\mathbf{H}_\infty$ control of vehicle suspension with magneto-rheological dampers. *Journal of Sound and Vibration*, 283(3-5):981–996, 2005.

[GB08] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming (web page and software). `http://www.stanford.edu/~boyd/cvx/`, July 2008.

[GBTH05] N. Giorgetti, A. Bemporad, H. E. Tseng, and D. Hrovat. Hybrid model predictive control application towards optimal semi-active suspension. In *IEEE International Symposium on Industrial Electronics*, volume 1, pages 391–398, 2005.

[GBY06] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In L. Liberti and N. Maculan, editors, *Global Optimization: from Theory to Implementation*, Nonconvex Optimization and Its Applications, pages 155–210. Springer Science & Business Media, Inc., New York, 2006.

[GM03] A. Gupta and C. D. Maranas. Managing demand uncertainty in supply chain planning. *Computers & Chemical Engineering*, 27(8-9):1219–1227, 2003.

[GSD05] G. C. Goodwin, M. M. Seron, and J. A. De Doná. *Constrained control and estimation*. Springer, 2005.

[Her05] F. Herzog. *Strategic Portfolio Management for Long-Term Investments: An Optimal Control Approach*. PhD thesis, ETH, Zurich, 2005.

[HF08] B. Houska and H. J. Ferreau. ACADO toolkit: Automatic control and dynamic optimization (web page and software). `http://www.acadotoolkit.org/`, August 2008.

[HNNS06] E. Handschin, F. Neise, H. Neumann, and R. Schultz. Optimal operation of dispersed generation under uncertainty using mathematical programming. *International Journal of Electrical Power & Energy Systems*, 28(9):618–626, 2006.

[HNPWHH07] M. Houwing, R. R. Negenborn, B. De Schutter P. W. Heijnen, and H. Hellendoorn. Least-cost model predictive control of residential energy resources when applying μCHP. *Proceedings of Power Tech*, 291:425–430, July 2007.

[IP05] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[KH05] W. H. Kwon and S. Han. *Receding Horizon Control*. Springer-Verlag, 2005.

[KS09] M. Khalid and A. V. Savkin. Model predictive control for wind power generation smoothing with controlled battery storage. In *Joint IEEE Conference on Decision and Control and Chinese Control Conference*, pages 7849–7853, Shanghai, China, 2009.

[KY06] R. Kumar and B. Yao. Model based power-split and control for electric energy system in a hybrid electric vehicle. In *Proceedings of IMECE 2006*, pages 335–341, Chicago, Illinois, 2006.

[LÖ4] 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`.

[Mac02] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[MAH+97] R. K. Mehra, J. N. Amin, K. J. Hedrick, C. Osorio, and S. Gopalasamy. Active suspension using preview information and model predictive control. In *Proceedings of the 1997 IEEE International Conference on Control Applications*, pages 860 –865, Hartford, Connecticut, oct. 1997.

[MB08] J. E. Mattingley and S. Boyd. CVXMOD: Convex optimization software in Python (web page and software). `http://cvxmod.net/`, August 2008.

[MB09] J. E. Mattingley and S. Boyd. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 23(3):50–61, 2009.

[MB10a] J. E. Mattingley and S. Boyd. Automatic code generation for real-time convex optimization. In D. P. Palomar and Y. C. Eldar, editors, *Convex optimization in signal processing and communications*, pages 1–41. Cambridge University Press, 2010.

[MB10b] J. E. Mattingley and S. Boyd. CVXGEN: Automatic convex optimization code generation (web page and software). `http://cvxgen.com/`, April 2010.

[MBM+09] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta. Processor speed control with thermal constraints. *IEEE Transactions on Circuits and Systems*, 56(9):1994–2007, 2009.

[Mil02] T. C. Miller. *Hierarchical operations and supply chain planning*. Springer Verlag, 2002.

[MTA06] E. Mestan, M. Turkay, and Y. Arkun. Optimization of operations in supply chain systems using hybrid systems approach and model predictive control. *Ind. Eng. Chem. Res*, 45(19):6493–6503, 2006.

[Oht04] T. Ohtsuka. A continuation/gmres method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563–574, April 2004.

[OK02] T. Ohtsuka and A. Kodama. Automatic code generation system for nonlinear receding horizon control. *Transactions of the Society of Instrument and Control Engineers*, 38(7):617–623, July 2002.

[PBK+07] Z. Preitl, P. Bauer, B. Kulcsar, G. Rizzo, and J. Bokor. Control solutions for hybrid solar vehicle fuel consumption minimization. In *2007 IEEE Intelligent Vehicles Symposium*, pages 767–772, Istanbul, Turkey, 2007.

[PLYG03] E. Perea-Lopez, B. E. Ydstie, and I. E. Grossmann. A model predictive control strategy for supply chain optimization. *Computers and Chemical Engineering*, 27(8-9):1201–1218, 2003.

[QB03] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003.

[RS07] A. Romanenko and L. O. Santos. A nonlinear model predictive control framework as free software: outlook and progress report. *Assessment and Future Directions of Nonlinear Model Predictive Control*, pages 229–238, 2007.

[RWR04] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of interior point methods to model predictive control. *Journal of optimization theory and applications*, 99(3):723–757, November 2004.

[SSH+03] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware computer systems: Opportunities and challenges. *IEEE Micro*, 23(6):52–61, 2003.

[Stu99] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11:625–653, 1999. Software available at `http://sedumi.ie.lehigh.edu/`.

[SWR06] J. D. Schwartz, W. Wang, and D. E. Rivera. Simulation-based optimization of process control policies for inventory management in supply chains. *Automatica*, 42(8):1311–1320, 2006.

[TR04] K. T. Talluri and G. J. Van Ryzin. *The Theory and Practice of Revenue Management*. Springer, 2004.

[TTT99] 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*, 11(1):545–581, 1999.

[TTT03] R. Tütüncü, K. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming*, 95(2):189–217, 2003.

[WAT09] A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *Proc. IEEE INFOCOM*, pages 2007–2015, Rio de Janeiro, Brazil, 2009.

[WB08] Y. Wang and S. Boyd. Fast model predictive control using online optimization. In *Proceedings IFAC World Congress*, pages 6974–6997, Seoul, South Korea, July 2008.

[WB09] Y. Wang and S. Boyd. Performance bounds for linear stochastic control. *System and Control Letters*, 53(3):178–182, March 2009.

[WB10] Y. Wang and S. Boyd. Fast evaluation of quadratic control-lyapunov policy. *IEEE Transactions on Control Systems Technology*, 2010. To appear. Manuscript available at `http://www.stanford.edu/~boyd/papers/fast_clf.html`.

[WBS03] M. J. West, C. M. Bingham, and N. Schofield. Predictive control for energy management in all/more electric vehicles with multiple energy storage units. In *IEEE International Electric Machines and Drives Conference*, volume 1, pages 222–228, Madison, Wisconsin, 2003.

[Whi82] P. Whittle. *Optimization over Time*. John Wiley & Sons, Inc. New York, NY, USA, 1982.

[Wil97] R. A. Williams. Automotive active suspensions part 1: Basic principles. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 211(6):415–426, 1997.