MOOSE: Model Based Optimal Input Design Toolbox for Matlab

Technical Report

Mariette Annergren
mariette.annergren@ee.kth.se

Christian Larsson
christian.larsson@ee.kth.se

September 23, 2011

Contents

1 Introduction

1.1 MOOSE

MOOSE is a model based optimal input design toolbox developed for Matlab. The goal of the toolbox is to simplify the implementation of optimal input design problems. It provides an extra layer between the user and a convex optimization environment. The interface to the user is very much inspired by CVX [8].

This user guide requires some knowledge about optimal input design. For a theoretical background and introduction of the notation used, see Section 5.

1.2 Optimal input design formulation

MOOSE is designed to handle input design problems of the form

$$\begin{aligned} \underset{\Phi_u}{\text{minimize}} \quad & \texttt{objective} \\ \text{subject to} \quad & \mathscr{E}_{SI}(\alpha) \subseteq \Theta_{app}(\gamma) \\ & \beta(\omega) \leq \Phi_u(\omega) \leq \delta(\omega), \quad \forall \omega. \end{aligned}$$

The optimization problem is solved through the ellipsoidal relaxation or the scenario optimization approach. Both the input design problem and the methods for solving it are described in detail in Section 5.6. The problem is set up in Matlab by using a series of `keywords` in a MOOSE declaration block. The quick tutorial gives a short introduction to using MOOSE by a simple example. Further details on keywords, lower level implementation in MOOSE and theoretical background of optimal input design are presented in the following sections.

1.3 Quick tutorial

This tutorial presents the process of declaring and solving an optimal input design problem in MOOSE.

1.3.1 Setting up the problem

Consider input design for the system

$$G(\theta, q) = \theta_1 u(t-1) + \theta_2 u(t-2) + e(t),$$

with true parameter values $\theta = [10 \ \ -9]$ and noise variance $\text{var}\{e\} = \lambda = 1$. The objective is to solve the optimization problem

$$\begin{aligned} \underset{\Phi_u(\omega)}{\text{minimize}} \quad & \text{E}\{u^2\} \\ \text{subject to} \quad & \mathscr{E}_{SI}(0.95) \subseteq \Theta_{app}(100) \\ & \Phi_u(\omega) \geq 0, \quad \forall \omega \end{aligned}$$

using an FIR-input spectrum with 20 lags, i.e. coefficients in the spectral density function defined in Section 5.3, 100 samples of data and the ellipsoidal relaxation. A Matlab implementation of the problem is presented below.

```
% Setup system and model
theta  = [10 -9];
G      = tf([0 theta],1,1,'variable','z^-1');
H      = tf(1,1,1,'variable','z^-1');
Lambda = 1;
% Moose declaration block
mooseBegin
    objective minimize(inputPower)
    model G H Lambda
    identification constraints
        spectrum phiU = FIR(20)
        probability 0.95
        numSamples 100
    application constraints
        ellipsoid(@Vapp,100)
mooseEnd
optimalFilter = mooseProblem.spectralFactor;

% Example application cost where the estimate
% of theta2 is important
function V = Vapp(theta)
    theta0 = [10 -9];
    V = norm(theta(2)-theta0(2),2);
end
```

The first three lines setup the model for the nominal system, the noise transfer functions and the noise variance.

The MOOSE declaration block begins with the `mooseBegin` command. This creates the `mooseProblem` variable in the workspace where the input design is stored.

The eight basic keywords are then used to setup the problem:

- `objective` sets the optimization objective function, in the example the input power is minimized.
- `model` sets the nominal system, noise model and noise variance.
- `identification constraints` has no formal action but can be used to make the declaration clearer.
- `spectrum` sets the spectrum type. The non-negativity constraint is handled automatically.
- `probability` is the level of the confidence ellipsoid.
- `numSamples` sets the number of samples used in the identification experiment.

- `application constraints` has no formal action but can be used to make the declaration clearer.

- `ellipsoid` adds the ellipsoid defined by the Hessian of `Vapp` to the set Θ_{app}.

The MOOSE declaration block ends with the `mooseEnd` command. In addition to closing the block, it also calls an optimization problem solver to find the optimal design.

The last line gets the optimal spectral factor which can be used to realize an input signal.

In the example some keywords have been assigned variable names. This makes the keyword available inside the MOOSE declaration block so that it can be used in, for example, the objective function. When the MOOSE declaration block is closed, named variables remain available in the Matlab workspace.

2 Using the toolbox

2.1 Installation of MOOSE

To install MOOSE, download the toolbox catalog and then run the command `mooseSetup`. The toolbox is based on `CVX`, a package for specifying and solving convex programs [8]. Thus, you also need to have `CVX` installed to be able to use MOOSE.

2.2 `mooseBegin` and `mooseEnd`

An input design problem is defined in MOOSE in a MOOSE declaration block.

2.2.1 The `mooseBegin` command

The `mooseBegin` command declares the beginning of a MOOSE declaration block. When the command is called, two things happen. First, the other keywords needed in the problem declaration are activated. Second, a variable `mooseProblem` is created in the workspace to handle the input design declaration.

The MOOSE keywords are not available outside of a MOOSE declaration block. Thus, any conflict with other Matlab functions is minimized.

2.2.2 The `mooseEnd` command

The `mooseEnd` command declares the end of a MOOSE declaration block. When the command is called the input design problem is passed to `CVX` and solved. The solver print out and optimal value are shown in the command window.

2.3 Models

Two types of models are supported by MOOSE. They are transfer function and state space models. The transfer function model is defined as

$$\mathcal{M}_{tf}: \; y(t) = G(q^{-1},\theta)u(t) + H(q^{-1},\theta)e(t), \tag{1}$$

where G and H are transfer functions and q^{-1} is the backward shift operator. The state space model is defined as

$$\begin{aligned} \mathcal{M}_{ss}: \; x(t+1) &= F(\theta)x(t) + G(\theta)u(t) + K(\theta)e(t), \\ y(t) &= H(\theta)x(t) + e(t), \end{aligned} \tag{2}$$

where F, G, K and H are matrices. The noise signal $e(t)$ is, for both models, a white Gaussian process with covariance matrix Λ. The unknown parameter vector is denoted θ.

To declare a $\mathcal{M}_{tf}$ in MOOSE you would type

```
mooseBegin
    model G H Lambda
mooseEnd
```

where `G` and `H` are the discrete transfer functions in model (1). They are declared using Matlab's `tf`-command. The third argument, `Lambda`, is the covariance matrix of the noise.

To declare a $\mathcal{M}_{ss}$, you would instead type

```
mooseBegin
    model F G H K Lambda Ts
mooseEnd
```

where `F`, `G`, `K`, `H` are the matrices in model (2). The fifth argument, `Lambda`, is the covariance matrix of the noise and the sixth argument, `Ts`, is the sampling time. An initial estimate of the unknown parameters θ is used when declaring both $\mathscr{M}_{tf}$ and $\mathscr{M}_{ss}$.

MOOSE assumes that the models are completely parameterized, all coefficients in the model are considered as separate parameters. When this is not the case, you can declare which of the parameters in the model are known and which you need to estimate. For instance an $\mathscr{M}_{tf}$ can be declared in the following way:

```
G      = tf({[10 -9]},{[1 0 0]},1);
H      = tf({1},{1},1);
Gindex = '{[1 2]},{[0 0 0]}';
Hindex = '{0},{0}';

mooseBegin
    model G(Gindex) H(Hindex) Lambda
mooseEnd
```

The unknown parameters are numbered $1,2,\ldots$ and the known parameters are numbered 0. Each number is assigned to the parameter situated at the same position in the `tf`-declaration of `G` and `H`. If you know that two or more coefficients are alike, you assign them the same number.

An $\mathscr{M}_{ss}$ can be declared in the following way:

```
mooseBegin
    model F(Findex) G(Gindex) H(Hindex) K(Kindex) Lambda Ts
mooseEnd
```

The numbering syntax is the same as for $\mathscr{M}_{tf}$ but the arguments, `Findex, Gindex, Kindex` and `Hindex` are matrices instead.

It is possible to name your models. This is done simply by writing

```
mooseBegin
    model modelName = G H Lambda
mooseEnd
```

and equivalently for state space models.

2.4 Objectives

There are two default objectives supported by MOOSE. They are D-optimality [1] and minimization of input power. These objectives are declared as

```
mooseBegin
    objective dOptimality
mooseEnd
```

and

```
mooseBegin
    objective minimize(inputPower)
mooseEnd
```

NB! When the objective is D-optimality, the maximum input power needs to be set to obtain a well posed optimization problem, see Section **??**.

Sometimes it is desirable to express the objective using the spectrum coefficients, number of samples and so forth. As long as the objective function remains convex in the decision variables, the design problem will remain convex. This is not fully implemented in the current version and is therefore not documented.

2.5 Identification constraints

The keyword `identification constraints` does nothing. It is only provided to make the problem declaration more readable. The `spectrum`, `probability` and `numSamples` keywords can be viewed as identification constraints. That is, constraints regarding the identification procedure of the system.

2.5.1 Spectrum

One type of input spectrum is supported by MOOSE. It is the FIR spectrum, see Section 5.3. To declare an FIR spectrum in MOOSE you type

```
mooseBegin
    identification constraints
        spectrum spectrumName = FIR(50)
mooseEnd
```

where `nameSpectrum` is the name of the spectrum. The name can be omitted. The argument, in this example 50, is the number of coefficients in the FIR spectrum. It is possible to set a maximum input power by declaring

```
spectrumName.maxPower = maximumPower;
```

2.5.2 Probability

To declare the probability that you want to have in your optimal input design you type

```
mooseBegin
    identification constraints
        probability probabilityName = 0.95
mooseEnd
```

where the probability is given in decimal form.

2.5.3 Number of samples

The number of samples that is to be used in the identification experiment is declared in the following way:

```
mooseBegin
    identification constraints
        numSamples numSamplesName = 100
mooseEnd
```

2.6 Application constraints

The keyword `application constraints` does nothing. It is, as for `identification constraints`, only provided to make the problem declaration more readable. Currently `ellipsoid` and `scenarios` are available to define the application constraints. That is, constraints regarding the required performance of the model.

2.6.1 Ellipsoid

The keyword adds an ellipsoid to the application constraints. The syntax is

```
mooseBegin
    application constraints
        ellipsoid(fun,gamma)
mooseEnd
```

The argument `fun` can be a function handle to an application cost or the Hessian of the application cost evaluated at the true parameter values. The format of the application cost must be

```
function V = Vapp(theta)
```

The argument `gamma` is the accuracy which sets the value for the level curve that gives the ellipsoid. For a theoretical description of the application cost and the accuracy γ, see Section 5.5.

2.6.2 Scenarios

The keyword adds scenario constraints to the application constraints. The syntax is

```
mooseBegin
    application constraints
        scenarios(mat,gamma)
mooseEnd
```

The argument `mat` is a fat matrix where the first rows contain the scenarios and the last row the application cost values at each scenario. The argument `gamma` is the accuracy which sets the maximum application cost value.

2.7 Spectral factorization

Once a MOOSE problem has been solved you can get the stable minimum phase spectral factor of the optimal input spectrum. The syntax is

```
optH = mooseProblem.spectralFactor;
```

2.8 Extra functions

Other functions which are included in the toolbox are explained in the following sections.

2.8.1 `firreal()`

The Matlab function `firreal()` yields a filter for the optimal input signal given the auto-correlation sequence of the optimal input signal and the sampling time used in the design.

2.8.2 `ellipse()`

The Matlab function `ellipse()` plots an ellipse and its center point given a matrix, a point and the desired color of the plot.

2.8.3 `hessian()`

The Matlab function `hessian()` calculates the Hessian of a function at a particular point given the function and the point. It is part of a Matlab package named Adaptive Robust Numerical Differentiation created by John D'Errico [6].

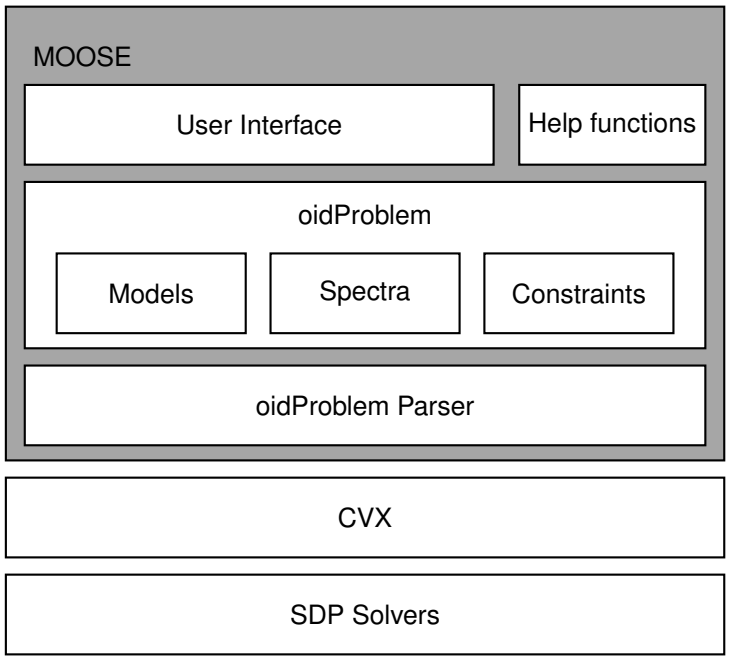

Figure 1: The structure of the MOOSE implementation. The user interacts with the top layer through the user interface and the help functions. The lower layers of MOOSE are used to define and store the optimal input design problem. The last layer of MOOSE is a parser that converts an instance of oidProblem to a `CVX` problem. MOOSE relies on `CVX` and SDP solvers to solve the optimization problem.

3 Implementation

The implementation of MOOSE uses the object-oriented programming capabilities of Matlab. The structure of the implementation is presented in Figure 1. The design is built around a predefined set of interfaces for the necessary component of the optimal input design problem.

User interaction with MOOSE is typically done through the user interface and in some cases through the help functions. The user interface is based on a set of `keywords` that define an optimal input design problem.

The central object is the class `oidProblem` where the input design problem is stored. Every instance of `oidProblem` contains a model- and a spectrum-instance and one or more constraint-instances.

Abstract classes are used for the model, spectrum and constraint classes to define interfaces. This allows for easy implementation of new models, spectra and constraint classes. An abstract class is also defined for the parser.

There is no optimization implemented in MOOSE. Instead the toolbox relies on external SDP solvers for solving the defined optimization problem. An abstract parser class defines the interface of the parsers used to construct the SDP from an `oidProblem`. Currently the only implemented parser is to `CVX`.

3.1 Interfaces

Abstract classes are used to define interfaces for the subclasses of the `oidProblem` and the parser-class. The current structure of the interfaces is not expected to be used in future versions of MOOSE and are therefore not presented here.

4 Future versions

At the time of writing, the MOOSE project is in its initial stages and the first version of the toolbox is just released. The project is however very active with many planned extensions in future versions of the toolbox. The features with highest priority for implementation are presented here. If any other features are requested, contacting the creators is most appreciated. Any other comments are also welcome.

A complete API for the toolbox is planned to be released in the near future. Users are encouraged to implement their own classes as they are needed. Well implemented contributions may be included in future versions of the toolbox.

4.1 Planned features

The most highly prioritized features for coming version of MOOSE are:

- Spectrum types, in particular discrete spectra, from signals that are sums of sinusoids, and spectra that use partial correlation parameterization [10].
- Physically parameterized state space models.
- Optimization objectives, in particular output variance, joint input and output variance.
- Multiple application costs.

5 Theoretical background: Optimal input design

The objective with optimal input design is to deliver a model that, when used in the intended application, results in an acceptable performance. This is achieved by constructing a particular input signal to be used in the system identification experiment. The obtained model is highly dependent on the input signal used. Thus, by designing the input we are in a way designing the model obtained from the identification experiment.

5.1 Dynamic system and model

We consider a multivariate linear time invariant discrete time dynamic system. It is asymptotically stable. The system has sequences corresponding to an impulse response $\{g_k\}$, input signal $\{u(t)\}$, output signal $\{y(t)\}$, noise filter impulse response $\{h_k\}$ and additive zero mean white Gaussian noise $\{e(t)\}$ with a known variance Λ. The output response can be expressed as

$$y(t) = \sum_{k=1}^{\infty} g_k u(t-k) + h_k e(t-k), \quad t = 1,2,\ldots \tag{3}$$

The dynamic system can be approximated by a parametrized model. Given a structure, the model response is expressed as

$$\mathscr{M}(\theta): \quad y(t) = G(q,\theta)u(t) + v(t), \tag{4a}$$
$$v(t) = H(q,\theta)e(t), \tag{4b}$$

where $v(t)$ is the filtered noise signal. The transfer functions G and H are parameterized by $\theta \in \mathbb{R}^n$ and q is the forward-shift operator.

The true system is assumed to be parametrized by the same structure as the model. Thus, there exist parameters θ^0 such that the output response (3) can be written as

$$\mathscr{S}: \quad y(t) = G(q,\theta^0)u(t) + v_0(t),$$
$$v_0(t) = H(q,\theta^0)e(t).$$

The unknown parameters are θ, the true values of the parameters are θ^0 and the estimated parameters based on N observations are $\hat{\theta}_N$. The observations consist of observed output and input signal sequences, $Z^N = \{y(t), u(t)\}_{t=1}^N$.

5.2 Prediction error method

The prediction error method (PEM) [13] is a method of identifying the unknown parameters θ in the model (4). The parameter estimates are found by minimizing a criterion function of the prediction error with respect to θ. The prediction error is defined as the difference between the output of the true system and the output predicted by the model. Based on the model structure (4), the predicted output of the system is

$$\hat{y}(t|\theta) = H^{-1}(q,\theta)G(q,\theta)u(t) + \left[I - H^{-1}(q,\theta)\right] y(t).$$

Consequently, the prediction error becomes

$$\varepsilon(t,\theta) = y(t) - \hat{y}(t|\theta) = H^{-1}(q,\theta)\left[y(t) - G(q,\theta)u(t)\right].$$

The criterion function to be minimized is denoted $V_N(\theta, Z^N)$. The estimates are defined as

$$\hat{\theta}_N = \arg\min_{\theta} V_N(\theta, Z^N). \tag{5}$$

In MOOSE the criterion function is set to the quadratic criterion. That is,

$$V_N(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N} \varepsilon(t,\theta)^T \Lambda \varepsilon(t,\theta). \tag{6}$$

It holds under mild conditions that the estimated parameters converge to the true values with probability one as the number of observations tends to infinity [13]. We also have, under the same conditions, that the sequence of random variables

$$N(\hat{\theta}_N - \theta^0)^T V_N''(\theta^0, Z^N)(\hat{\theta}_N - \theta^0)$$

converges in distribution to the χ^2 distribution with n degrees of freedom [13]. Thus, for a sufficiently large N, the estimates $\hat{\theta}_N$ are with a probability α contained inside the ellipsoid

$$\mathscr{E}_{SI} = \left\{ \theta \mid (\theta - \theta^0)^{\mathrm{T}} V_N''(\theta^0, Z^N)(\theta - \theta^0) \leq \frac{\chi^2_\alpha(n)}{N} \right\}, \tag{7}$$

where $\chi^2_\alpha(n)$ is the α-percentile of the χ^2 distribution with n degrees of freedom. We call $\mathscr{E}_{SI}$ the system identification set.

The Hessian of the quadratic criterion function in (6) is

$$V_N''(\theta^0, Z^N) = \frac{1}{N}\sum_{t=1}^{N} \hat{y}'(t|\theta^0)\Lambda^{-1}\hat{y}'(t|\theta^0)^T. \tag{8}$$

We can rewrite expression (8) in the frequency domain using Parseval's relation. We get different expressions depending on if we perform open or closed loop identification. Open loop identification is when there is no feedback control of the system during the identification experiment. Closed loop identification is when there is feedback control. MOOSE only handles open loop identification. Thus, we have

$$\begin{aligned} V_N''(\theta^0, Z^N) = &\frac{1}{2\pi}\int_{-\pi}^{\pi} \Gamma_u(e^{i\omega}), \theta)(\Lambda^{-1} \otimes \Phi_u(e^{i\omega}))\Gamma_u^T(e^{-i\omega}), \theta) d\omega + \\ &\frac{1}{2\pi}\int_{-\pi}^{\pi} \Gamma_e(e^{i\omega}), \theta)(\Lambda^{-1} \otimes \Lambda(e^{i\omega}))\Gamma_e^T(e^{-i\omega}), \theta) d\omega, \end{aligned} \tag{9a}$$

where

$$\Gamma_u = \begin{bmatrix} vec[F_u^1] \\ \vdots \\ vec[F_u^n] \end{bmatrix}, \ \Gamma_e = \begin{bmatrix} vec[F_e^1] \\ \vdots \\ vec[F_e^n] \end{bmatrix}, \tag{9b}$$

$$F_u^i = H^{-1}\frac{dG(\theta)}{d\theta_i}, \ F_e^i = H^{-1}\frac{dH(\theta)}{d\theta_i}, \text{ for all } i = 1 \ldots n. \tag{9c}$$

Here θ_i denotes the i:th component of the vector θ. Furthermore, $vec[X]$ denotes a row vector which contains the rows of the matrix X stacked adjacent to each other. For

details, see [2]. From expression (9), we can see that the Hessian is an affine function of the input spectrum $\phi_u(\omega)$. Thus, we can directly affect the shape of the system identification set and, consequently, the estimates by designing a particular spectrum of the input signal.

The system identification set in (7) can be expressed using the Fisher information matrix instead of the Hessian. The information matrix is defined as

$$\mathbf{I}_F = \frac{1}{N}\sum_{t=1}^{N} \hat{y}'(t|\theta^0)\Lambda\hat{y}'(t|\theta^0)^T, \tag{10}$$

[13]. Thus, we can write

$$\mathscr{E}_{SI} = \left\{ \theta \mid (\theta-\theta^0)^{\mathrm{T}}\mathbf{I}_F(\theta^0)(\theta-\theta^0) \leq \frac{\chi^2_\alpha(n)}{N} \right\}.$$

PEM can also be used with state space formulations of the system and model. For details, see [11].

5.3 Spectrum of input signal

We saw in the previous section that the spectrum of the input signal used in the identification experiment affect the estimates. Thus, input design can be performed in terms of its frequency characteristics by choosing the spectrum of the signal. The spectral density of a stationary signal $u(t)$ can be written as

$$\Phi_u(\omega) = \sum_{k=-\infty}^{\infty} c_k \mathscr{B}_k(e^{i\omega}), \tag{11}$$

where the scalar basis functions $\{\mathscr{B}_k(e^{i\omega})\}_{k=0}^{\infty}$ are proper, stable, and rational such that $\mathscr{B}_{-k}(e^{i\omega}) = \mathscr{B}_k(e^{-i\omega})$ and the real coefficients $c_{-k} = c_k$. MOOSE only handles spectrum that are shaped as an FIR filter. That is, the basis functions are exponentials, $\mathscr{B}_k(e^{i\omega}) = e^{-i\omega k}$. Consequently, the coefficients become the autocorrelation sequence of the input signal. That is, $c_k = \mathrm{E}\{u(t)u(t-k)\}$. See for example [14].

Some optimal input design problems can be formulated as convex optimization problems with decision variables c_k. The design is then a matter of finding the coefficients c_k. There are two main difficulties with choosing them. First, the spectral density of a stationary process is a non-negative entity. Therefore, the coefficients must be chosen such that

$$\Phi_u(\omega) \succeq 0, \text{ for all } \omega, \tag{12}$$

for (11) to define a spectral density. Second, the constraint (12) is infinite dimensional making it computationally impractical to work with. To simplify the problem, we consider the partial expansion

$$\Phi_u(\omega) = \sum_{k=-(m-1)}^{m-1} c_k \mathscr{B}_k(e^{i\omega}). \tag{13}$$

Hence, only the first m coefficients of (11) are used to define the spectrum. Two approaches for choosing the coefficients c_k are partial correlation parameterization [10] and finite dimensional parameterization. MOOSE uses the latter approach.

5.3.1 Finite dimensional parameterization

Finite dimensional parameterization requires that $\{c_k\}_{k=0}^{m-1}$ is chosen such that (13) is a spectrum. It means that condition (12) must hold for the truncated sum (13). This can be achieved in various ways, the most frequently used technique is an application of the positive real lemma which springs from the Kalman-Yakubovich-Popov lemma.

Lemma 1: If $\{A,\ B,\ C,\ D\}$ is a controllable state-space realization of $\Phi_u^+(\omega) = \sum_{k=0}^{m-1} c_k \mathscr{B}_k(e^{i\omega})$. Then there exists a matrix $Q = Q^T$ such that

$$K(Q,\{A,B,C,D\}) \triangleq \begin{bmatrix} Q - A^T QA & -A^T QB \\ -B^T QA & -B^T QB \end{bmatrix} + \begin{bmatrix} 0 & C^T \\ C & D + D^T \end{bmatrix} \succeq 0, \qquad (14)$$

if and only if $\Phi_u(\omega) = \sum_{k=-(m-1)}^{m-1} c_k \mathscr{B}_k(e^{i\omega}) \geq 0,$ for all ω.

Thus, the necessary and sufficient condition for (12) to hold for the truncated sequence is the matrix inequality (14), assuming a matrix Q exists. The matrix inequality becomes a linear matrix inequality (LMI) in c_k and Q if the only matrices that are linearly dependent on the coefficients c_k are C and D.

5.4 Input generation

When the input spectrum is found, a corresponding time realization of the signal has to be generated. The realization is then used to excite the system in the identification experiment. One possible input generation is to let the input signal be white Gaussian noise filtered through a transfer function matrix. The matrix is chosen such that the obtained signal has the required spectrum. The matrix design is a problem of minimum phase spectral factorization, as such, it has many known solutions, see for example [14]. MOOSE do not provide tools for input generation.

5.5 Application set

The input signal needs to be designed with the intended application of the model in mind. To enable this, a measure of how well the model performs is defined. The degradation in performance due to a mismatch between the model and the system is specified by an application cost function. The cost emphasizes an important performance quality of the system. Examples of such qualities are the sensitivity function and the closed loop output response. The cost function is denoted $V_{app}(\theta)$. The minimal value of V_{app} is equal to zero and is achieved when the true parameters are used in the function. These conditions are equivalent to the constraints $V_{app}(\theta^0) = 0$, $V'_{app}(\theta^0) = 0$ and $V''_{app}(\theta^0) \succeq 0$.

An increased value of the application cost reflects an increased degradation in performance. The maximal allowed degradation is defined by

$$V_{app}(\theta) \leq \frac{1}{\gamma}, \qquad (15)$$

where γ is a positive scalar. The parameters fulfilling inequality (15) are called acceptable parameters and they belong to the application set. The set is defined as

$$\Theta_{app}(\gamma) = \left\{ \theta \ | V_{app}(\theta) \leq \frac{1}{\gamma} \right\}.$$

The concept of using application sets comes from [9] and [3].

5.6 Optimal input design problem

We want the estimated parameters to give a model with acceptable performance. Thus, one part of the objective of optimal input design is to guarantee, with high probability, that the estimated parameters are acceptable parameters. This condition is equivalent to requiring that $\mathscr{E}_{SI}(\alpha) \subseteq \Theta_{app}(\gamma)$, for specific values of α and γ. The second part of the objective is to minimize the cost related to performing the experiment. To simplify the minimization greatly, the cost is chosen as a convex function with respect to the coefficients in the spectral density function of the input signal. For example $c_0 = \mathrm{E}\left\{u(t)^T u(t)\right\}$, the power of the input signal.

Let f_{cost} denote the cost related to the experiment. The complete objective of optimal input design can then be stated as the optimization problem

$$\underset{c_k}{\text{minimize}}\ f_{cost}(c_k), \tag{16a}$$

$$\text{subject to } \mathscr{E}_{SI}(\alpha) \subseteq \Theta_{app}(\gamma), \tag{16b}$$

$$\Phi_u(\omega) \geq 0, \text{ for all } \omega. \tag{16c}$$

The optimization problem (16) can be approximated by a convex formulation. The benefit is that a convex optimization problem can be solved accurately and efficiently [4]. The objective function (16a) is convex. The second constraint (16c) can be replaced by an LMI using finite dimensional parametrization. The first constraint (16b) needs to be relaxed into a convex constraint. Two methods of doing this are the scenario approach and the ellipsoidal approximation. MOOSE supports both methods.

5.6.1 Scenario approach

The requirement that the system identification set lies inside the application set is relaxed. It is enough that a finite number of the estimated parameters are contained inside Θ_{app}. These parameters are chosen from $\mathscr{E}_{SI}$ according to a given probability distribution. It is shown in [5] that if

$$(\theta_i - \theta^0)^{\mathrm{T}} \mathbf{I}_F (\theta_i - \theta^0) \geq \frac{\chi^2_\alpha(n)\gamma}{N} V_{app}(\theta_i) \text{ for } i = 1 \ldots M < \infty,$$

where $\theta_i \in \mathscr{E}_{SI}$, then $\mathscr{E}_{SI}$ lies inside Θ_{app} with a high probability. Thus, optimization problem (16) can be approximated as

$$\underset{c_k}{\text{minimize}}\ f_{cost}(c_k), \tag{17a}$$

$$\text{subject to } (\theta_i - \theta^0)^{\mathrm{T}} \mathbf{I}_F (\theta_i - \theta^0) \geq \frac{\chi^2_\alpha(n)\gamma}{N} V_{app}(\theta_i),\ i = 1 \ldots M, \tag{17b}$$

$$K(Q, \{A, B, C, D\}) \succeq 0. \tag{17c}$$

Here we use finite dimensional parametrization on the second constraint (16c). Formulation (17) is a convex optimization problem.

5.6.2 Ellipsoidal approximation

The application cost is approximated by its second order Taylor expansion centered around the true parameters. The corresponding application set becomes an ellipsoidal region. Thus, $\Theta_{app} \approx \mathscr{E}_{app}$ for θ close to θ^0, where

$$\mathscr{E}_{app}(\gamma) = \{\theta \mid (\theta - \theta^0)^{\mathrm{T}} V''_{app}(\theta^0)(\theta - \theta^0) \leq 1/\gamma\}. \tag{18}$$

It is shown in [9] that $\mathscr{E}_{SI}$ lies inside $\mathscr{E}_{app}$ if and only if $\mathbf{I}_F \succeq \chi^2_\alpha(n)\gamma V''_{app}(\theta^0)/N$. Thus, optimization problem (16) can be approximated as

$$\underset{c_k}{\text{minimize}}\ f_{cost}(c_k), \tag{19a}$$

$$\text{subject to } \mathbf{I}_F \succeq \chi^2_\alpha(n)\gamma V''_{app}(\theta^0)/N, \tag{19b}$$

$$K(Q,\{A,B,C,D\}) \succeq 0. \tag{19c}$$

Here we also use finite dimensional parametrization. Formulation (19) is a convex optimization problem.

5.7 Performing system identification experiments

There is a conflict within optimal input design. The method requires knowledge of the true parameters. However, if these values were known, we would have no reason to construct an identification experiment. To get around the conflict, an initial estimate of the parameters is used instead of the true values. The estimate can be obtained through, for example, an identification experiment or physical insight of the process. A method based on this maneuver is described by the input design algorithm (IDA) in Table 1.

Table 1: Input design algorithm

Step	Action
Step 0	Find an initial estimate of the model parameters.
Step 1	Find the application cost based on simulations of the model with the parameter estimates.
Step 2	Design the optimal input signal based on the application cost and parameter estimates.
Step 3	Find a new estimate of the model parameters using the optimal input signal in the system identification experiment.

IDA can be iterated so that, after implementing the method once, the initial estimates in Step 0 is set to the obtained estimates in Step 3. As more and more data are used in the identification step and if there exist parameters θ^0 such that $\mathscr{S} = \mathscr{M}(\theta^0)$, the estimate converges to the true values. Therefore, we expect the input design based on an initial estimate to converge to the design based on the true values. A discussion on this and a formal proof for autoregressive systems with exogenous inputs are found in [7].

5.7.1 Approximate application cost

It is unlikely that we can evaluate the application cost using outputs from the real process. Such an evaluation requires running the process based on models with more or less arbitrary parameter values. Instead we can perform the evaluation in simulation. For this purpose, we introduce the approximative application cost $\tilde{V}_{app}(\theta)$. The approximation is evaluated using outputs from a linear model of the process where the parameters of the model are set to an initial estimate. To obtain an optimal input design using the approximate application cost, $V_{app}(\theta)$ is replaced by $\tilde{V}_{app}(\theta)$ in the relevant expressions. For a detailed description of this procedure, see [12].

References

[1] A.C. Atkinson and A.N. Doner. *Optimum experiment design*. Clarendon Press, Oxford, 1992.

[2] M. Barenthin Syberg. *Complexity Issues, Validation and Input Design for Control in System Identification*. PhD thesis, Royal Institute of Technology (KTH), December 2008. TRITA-EE 2008:055.

[3] X. Bombois, G. Scorletti, M. Gevers, P. M. J. V. D. Hof, and R. Hildebrand. Least costly identification experiment for control. *Automatica*, 42:1651–1662, 2006.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2003.

[5] G C Calafiore and M C Campi. The Scenario Approach to Robust Control Design. *IEEE Transactions on Automatic Control*, 51(5):742–753, May 2006.

[6] J. D'Errico. Adaptive robust numerical differentiation. http://www.mathworks.com/matlabcentral/fileexchange/13490-adaptive-robust-numerical-differentiation, June 2011.

[7] L. Gerencsér, H. Hjalmarsson, and J. Mårtensson. Identification of ARX systems with non-stationary inputs - asymptotic analysis with application to adaptive input design. *Automatica*, 45(3):623–633, March 2009.

[8] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. http://cvxr.com/cvx, April 2011.

[9] H. Hjalmarsson. System identification of complex and structured systems. In *European Control Conference*, pages 3424–3452, Budapest, Hungary, 2009. Plenary address.

[10] H. Jansson and H. Hjalmarsson. Input Design via LMIs Admitting Frequency-wise Model Specifications in Confidence Regions. *IEEE Transactions on Automatic Control*, 50(10):1534–1549, 2005.

[11] C. A. Larsson. Toward applications oriented optimal input design with focus on model predictive control. Technical report, KTH Royal Institute of Technology, September 2011. Licentiate Thesis.

[12] C. A. Larsson, M. Annergren, and H. Hjalmarsson. On Optimal Input Design for Model Predictive Control. In *Proceedings IEEE Conference on Decision and Control*, December 2011.

[13] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1999.

[14] T. Söderström. *Discrete-Time Stochastic Systems: Estimation and Control*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.