QUINOPT Documentation

Release 2.2

Giovanni Fantuzzi

Jun 15, 2022

Contents

CHAPTER 1

What is QUINOPT?

QUINOPT (QUadratic INtegral OPTimisation) is an open-source add-on for YALMIP to compute rigorous upper and lower bounds on the optimal value of optimization problems with infinite-dimensional polynomial quadratic integral inequality constraints. Such problems commonly arise from stability analysis of linear PDEs using the so-called *energy method* (also known as $\mathcal{L}^2$ stability), and in bounding time-average properties of turbulent fluid flows using the so-called *background method*.

In the simplest form, given a bounded interval $[a,b] \subset \mathbb{R}$, a k-times continuously differentiable function $u \in C^k([a,b],\mathbb{R})$, and a vector of optimization variables $\gamma \in \mathbb{R}^s$, QUINOPT computes upper and/or lower bounds on the optimal value of the optimization problem

$$\begin{aligned} \min_{\gamma} \quad & c^T\gamma \\ \text{subject to} \quad & \int_a^b Q_\gamma(x,u(x),u'(x),...,u^{(k)}(x))\,\mathrm{d}x \geq 0 \quad \forall u(x) \in \mathcal{H} \end{aligned}$$

by constructing SDP-representable inner and outer approximations of its feasible set. In the problem above, $Q_\gamma(x,u(x),u'(x),...,u^{(k)}(x))$ is

- a quadratic polynomial in $u(x),u'(x),...,u^{(k)}(x)$;
- a polynomial in x;
- an affine function of the optimization variable γ.

Moreover, $\mathcal{H}$ is the subspace of functions that satisfy m homogeneous boundary conditions, i.e.

$$\mathcal{H} := \left\{ u \in C^k([a,b],\mathbb{R}) \quad a_1u(a) + a_2u(b) + a_3u'(a) + \cdots + a_{2k}u^{(k)}(b) = 0 \right\},$$

where $a_0, \ldots, a_{2k} \in \mathbb{R}^m$ are known vectors.

Note: Inhomogeneous boundary conditions can be "lifted" by changing variables according to $u(x) = v(x) + p(x)$, where $p(x)$ is a polynomial of sufficiently high degree satisfying the inhomogeneous boundary conditions.

A particularly simple example of an optimization problem with an integral inequality is to determine the best Poincaré constant, i.e. the largest $\gamma > 0$ such that

$$\int_0^1 \left[|u'(x)|^2 - \gamma|u(x)|^2\right] \mathrm{d}x \geq 0 \quad \forall u \in C^2([0,1],\mathbb{R}), \quad u(0) = 0 = u(1).$$

Upper and lower bounds on the largest γ are found by QUINOPT upon solving two SDPs.

Note: QUINOPT can also handle problems with more dependent variables, i.e. $u : [a,b] \to \mathbb{R}^q$, and problems in which the boundary values of the dependent variables and their derivatives appear explicitly in the integrand of the inequality constraint. For more details, see our paper or have a look at the *examples*.

- *Back to Table of Contents*

CHAPTER 2

License & System Requirements

2.1 License

QUINOPT is distributed under the Apache 2.0 License

Important: QUINOPT is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under the Apache 2.0 License.

2.2 System requirements

In order to use QUINOPT, you will need:

1. A working version of YALMIP, the MATLAB optimization modelling software by J. Löfberg. See the installation instructions for more details on how to download and install YALMIP.
2. A suitable SDP solver. Choices include SeDuMi, SDPT3, SDPA, Mosek (free for users in academia).

Instructions on how to obtain YALMIP and a suitable SDP solver are given in the *installation guide*.

Warning: QUINOPT has been succesfully tested on MATLAB 7.10 (R2010a) and higher. If you have a different version of MATLAB, use at your own risk!

- *Back to Table of Contents*

CHAPTER 3

Download

3.1 Stable release

QUINOPT's latest stable version is v.|version|.

- Download stable release (.zip)
- See the source code on GitHub

Important: Although the stable releases are tested on a range of examples and on different platforms, QUINOPT is a research code and may contain (sometimes serious) bugs. We recommend that you regularly check QUINOPT's GitHub page for updates.

3.2 Developer version

The developer version of QUINOPT, including all latest features, up-to-date bug fixes, and experimental code, can also be downloaded.

- Download latest developer version (.zip)
- Download latest developer version (.tar.gz)
- See on GitHub

Warning: The developer version is not normally well tested, and might contain serious bugs, incomplete documentation, etc. Use it at your own risk!

3.3 Old versions

You can find old versions here.

- *Back to Table of Contents*

CHAPTER 4

Install QUINOPT

QUINOPT is easily installed by running the installer `installQUINOPT.m` in MATLAB. Below is a detailed installation guide.

4.1 Step 1: Install YALMIP

QUINOPT is an add-on for YALMIP, the optimization modeling software by J. Löfberg. If you already have YALMIP installed, you cak skip this step. Otherwise, download YALMIP and install it by adding the following folders to MATLAB's path:

```
YALMIP-master
YALMIP-master/extras
YALMIP-master/solvers
YALMIP-master/modules
YALMIP-master/modules/parametric
YALMIP-master/modules/moment
YALMIP-master/modules/global
YALMIP-master/modules/sos
YALMIP-master/operators
```

You can test your YALMIP installation by running

```
>> yalmiptest
```

at the MATLAB command prompt. More details on how to install or update YALMIP be found on YALMIP's website.

Note: The folder names above are the default when YALMIP is downloaded from GitHub. Should you wish to use a different folder name, simply replace <YALMIP-master> with the appropriate path.

Warning: YALMIP is regularly updated, and changes in YALMIP may sometimes affect the functionality of QUINOPT. If you have downloaded YALMIP's latest version and are experiencing installation problems, please contact us or file an issue via the GitHub issue tracker.

4.2 Step 2: Install an SDP solver

To be able to use QUINOPT, you need to install a semidefinite programming (SDP) solver compatible with YALMIP. A complete list of YALMIP-compatible SDP solvers can be found here.

Warning: QUINOPT has been tested with SeDuMi, SDPT3, SDPA, and Mosek (free for users in academia). Other suitable YALMIP-compatible SDP solver should work, but use them at your own risk!

4.3 Step 3: Install QUINOPT

If you have successfully installed YALMIP and a compatible SDP solver, you are ready to install QUINOPT. First, download QUINOPT's latest stable version (for the "developer" version or previous versions, visit the Download page).

After unzipping the downloaded folder, navigate to it in MATLAB and simply run the installer:

```
>> installQUINOPT
```

The installer should compile the required files, add the required folders to the MATLAB path, and run some test problems to make sure everything is working. If you experience any installation problems, please contact us or file an issue via the GitHub issue tracker.

Warning: During installation, you may receive the following warning:

```
Warning: Compilation of mex files by installQUINOPT failed.
QUINOPT will still work without compiled mex files, but
it will be slower. To resolve the issue, make sure that
a supported compiler is installed and re-run the installer.
```

QUINOPT should still work, but you may wish to resolve the issue with the mex file compilation. You can find a list of supported compilers for MATLAB's latest version on this webpage; for all other versions of MATLAB please look at this webpage.

- *Back to Table of Contents*

CHAPTER 5

Examples

The best way to get started with QUINOPT is to look at some examples. Below is a list of demo problems of increasing difficulty, each of which introduces a new feature of QUINOPT. The last four examples, in particular, solve non-trivial problems from the field of fluid dynamics.

If you wish to know more about the theory behind QUINOPT, have look at this paper.

5.1 List of examples

5.1.1 Feasibility of an integral inequality

One of the most basic applications of QUINOPT is to determine the value of a parameter such that a homogeneous quadratic integral functional is positive. Here, we demonstrate how to use QUINOPT to find a value γ such that

$$\int_0^1 \left[|u'(x)|^2 + \gamma\, u'(x)\, u(x) + |u(x)|^2\right] \mathrm{d}x \geq 0$$

for all differentiable functions $u(x)$. Clearly, possible choices are $\gamma = 0$, $\gamma = -2$, or $\gamma = 2$ (in the last two cases the integrand is a perfect square).

`Download the MATLAB file for this example`

1. Create the variables

The first step to use QUINOPT is to set up the problem variables. These are the integration variable $x \in [0, 1]$ (the *independent variable*), the unknown function $u(x)$ (the *dependent variable*), and the optimization parameter γ.

First, we create the independent variable $x \in [0, 1]$ using the command `indvar()`, as

```
>> x = indvar(0,1);                    % Create the independent variable with domain␣
↪[0,1]
```

Then, we set up the dependent variable $u(x)$ using the command `depvar()`:

```
>> u = depvar(x);                          % Create the dependent variable u(x)
```

Finally, we set up the optimization parameter γ using the command `parameters`

```
>> parameters gamma;                       % Create the optimization variable gamma
```

Note: The commands `indvar()` and `depvar()` return MATLAB objects of class `@indvar` and `@depvar`, respectively. While the `@indvar` class behaves like a usual YALMIP variable, the `@depvar` class is specific to QUINOPT and does **not** behave like a YALMIP variable. Instead, it is intended to be used only as shown in the following.

2. Set up the inequality

Once the variables have been set up, we can set up the inequality. This is done in QUINOPT by constructing the integrand expression.

```
>> EXPR = u(x,1)^2 + gamma*u(x,1)*u(x) + u(x)^2;          % Create the integrand
```

In the expression above, the syntax `u(x,DER)` is used to specify the derivative of $u(x)$ of order `DER`. In other words, `u(x,1)` is the first derivative of $u(x)$.

Note: The integration interval has already been specified when defining the independent variable.

3. Solve the problem with QUINOPT

Once the variables and the integrand of the inequality have been set up, a value of γ for which the integral functional is positive semidefinite can be found using the command `quinopt()`, together with YALMIP's command `value()`

```
>> quinopt(EXPR);          % Solve the problem
>> value(gamma)            % Extract the value of gamma
```

4. Summary

In summary, a feasible value γ such that the integral inequality at the top of the page holds can be found using the following simple commands:

```
>> x = indvar(0,1);                                    % Create the independent␣
↪variable with domain [0,1]
>> u = depvar(x);                                      % Create the dependent␣
↪variable u(x)
>> parameters gamma;                                   % Create the optimization␣
↪variable gamma
>> EXPR = u(x,1)^2 + gamma*u(x,1)*u(x) + u(x)^2;       % Create the integrand
>> quinopt(EXPR);                                      % Solve the problem
>> value(gamma)                                        % Extract the value of gamma
```

- *Back to Table of Contents*

5.1.2 Feasibility of an integral inequality with boundary conditions

We now revisit the *previous example* to show how to specify boundary conditions on the dependent variables. Specifically, we demonstrate how to use QUINOPT to find the minimum value γ such that

$$\int_0^1 \left[|u'(x)|^2 + \gamma\, u'(x)\, u(x) + |u(x)|^2\right] \mathrm{d}x \geq 0$$

for all differentiable functions $u(x)$ that satisfy the boundary conditions

$$u'(0) = 0, \quad \text{and} \quad u(1) = 0.$$

`Download the MATLAB file for this example`

1. Clear the workspace

Before we solve a new example, it is good practice to clear the variables from the workspace using MATLAB's `clear` command. Moreover, to prevent the build-up of unused internal variables in QUINOPT, it is useful to clear QUINOPT's internal variables using the command `quinopt clear`:

```
>> clear
>> quinopt clear
```

2. Create the variables

As in the *previous example*, we create the independent variable $x \in [0, 1]$, the dependent variable $u(x)$, and the optimization parameter γ:

```
>> x = indvar(0,1);
>> u = depvar(x);
>> parameters gamma;
```

3. Set up the inequality

As in the *previous example*, we begin by constructing the integrand expression.

```
>> EXPR = u(x,1)^2 + gamma*u(x,1)*u(x) + u(x)^2;          % Create the integrand
```

The boundary condition can be specified through a vector `BC`, which is interpreted internally as the element-wise condition `BC=0`:

```
>> BC(1) = [u(0,1)];     % Create the boundary condition u'(0)=0
>> BC(2) = [u(1)];       % Create the boundary condition u(1)=0
```

Note: The syntax `u(POINT,DER)` is used to specify the derivative of $u(x)$ of order `DER`, evaluated at the point `POINT`. Possible values of `POINT` are the independent variable of integration `x`, or the extrema of the domain of integration, in this case 0 and 1.

---

4. Solve the problem with QUINOPT

Once the variables and the integrand of the inequality have been set up, a value of γ for which the integral functional is positive semidefinite can be found using the command `quinopt()` with two inputs:

```
>> quinopt(EXPR,BC);      % Solve the problem
>> value(gamma)           % Extract the value of gamma
```

5. Summary

In summary, a feasible value γ such that the integral inequality at the top of the page holds can be found using the following simple commands:

```
>> clear                                                    % Clear workspace
>> quinopt clear                                            % Clear QUINOPT internals
>> x = indvar(0,1);                                         % Create the independent variable␣
↪with domain [0,1]
>> u = depvar(x);                                           % Create the dependent variable␣
↪u(x)
>> parameters gamma;                                        % Create the optimization variable␣
↪gamma
>> EXPR = u(x,1)^2 + gamma*u(x,1)*u(x) + u(x)^2;            % Create the integrand
>> BC(1) = [u(0,1)];                                        % Create the boundary condition u
↪'(0)=0
>> BC(2) = [u(1)];                                          % Create the boundary condition␣
↪u(1)=0
>> quinopt(EXPR,BC);                                        % Solve the problem
>> value(gamma)                                             % Extract the value of gamma
```

- *Back to Table of Contents*

5.1.3 Poincaré's inequality with Dirichlet boundary conditions

Poincaré's inequality for functions $u : [0, 1] \to \mathbb{R}$ that satify the Dirichlet boundary conditions

$$u(0) = 0 = u(1)$$

states that

$$\int_0^1 |u'(x)|^2 \mathrm{d}x \geq \pi^2 \int_0^1 |u(x)|^2 \mathrm{d}x.$$

In this example, we verify that the constant π^2 on the right-hand side is optimal, in the sense that it solve the optimization problem

$$\begin{aligned} &\max_{\gamma} \quad \gamma \\ &\text{subject to} \quad \int_0^1 \left[|u'(x)|^2 - \gamma |u(x)|^2\right] \, \mathrm{d}x \geq 0, \quad u \in C^1([0,1], \mathbb{R}), \quad u(0) = 0 = u(1). \end{aligned}$$

`Download the MATLAB file for this example`

1. Create the variables

As usual, we start by clearing the model and creating the variables:

```
>> clear
>> quinopt clear
>> x = indvar(0,1);
>> u = depvar(x);
>> parameters gamma;
```

2. Set up the inequality

To set up Poincaré's inequality constraint, first we specify the integrand:

```
>> EXPR = u(x,1)^2 - gamma*u(x)^2;
```

and then we set up the vector of boundary conditions (this can be a row vector as in the previous example, or a column vector):

```
>> BC = [u(0); u(1)];
```

3. Solve the problem

To solve the problem and maximize γ, we use once again the command `quinopt()`, this time with three arguments: `EXPR`, `BC` and the objective function.

Note: When calling `quinopt(EXPR,BC,OBJECTIVE)`, QUINOPT **minimizes** the specified objective function.

Since QUINOPT minimizes the specified objective function, instead of maximizing γ we minimize $-\gamma$:

```
>> quinopt(EXPR,BC,-gamma);              % Maximize gamma (by minimizing -gamma)
>> value(gamma)/pi^2                     % Get the optimal value (in units of pi^2)
```

With the default parameters in QUINOPT, we obtain $\gamma_{\text{opt}} = 0.9994\,\pi^2$, i.e. the optimal solution returned by QUINOPT is within 99.9% of true optimum $\gamma_{\text{exact}} = \pi^2$.

4. Summary

In summary, the optimal constant for Poincaré's inequality can be determined with the following simple lines of code:

```
>> clear
>> quinopt clear
>> x = indvar(0,1);
>> u = depvar(x);
>> parameters gamma;
>> EXPR = u(x,1)^2 - gamma*u(x)^2;
>> BC = [u(0); u(1)];
>> quinopt(EXPR,BC,-gamma);
>> value(gamma)/pi^2
```

- *Back to Table of Contents*

5.1.4 Wirtinger's inequality

Wirtinger's inequality states that for a 2π-periodic function v that satisfies

$$\int_0^{2\pi} v(x)\mathrm{d}x = 0$$

there exists a constant C such that

$$\int_0^{2\pi} \left[C|v'(x)|^2 - |v(x)|^2\right] \mathrm{d}x \geq 0.$$

In this example, we verify that the smallest possible constant is $C = 1$. We do so by computing a sequence of convergent upper and lower bounds on the smallest C with QUINOPT. The objective of this example is to demonstrate how to override the default options in QUINOPT.

`Download the MATLAB file for this example`

1. Reformulate the problem

Before inputting the integral inequality into QUINOPT, we need to remove the integral constraint on $v(x)$. This can be done by defining

$$u(x) = \int_0^x v(t)\,\mathrm{d}t$$

so the zero-integral and periodicity conditions on v become

$$u'(0) - u'(2\pi) = 0, \quad u(2\pi) = 0.$$

Moreover, by definition of $u(x)$ we have the additional boundary condition $u(0) = 0$. With this change of variables, Wirtinger's inequality becomes

$$\int_0^{2\pi} \left[C|u''(x)|^2 - |u'(x)|^2\right] \mathrm{d}x \geq 0.$$

2. Set up the problem

As usual, we start by clearing the model and creating the variables:

```
>> clear;
>> yalmip clear;
>> quinopt clear;
>> x = indvar(0,2*pi);
>> u = depvar(x);
>> parameters C;
```

To set up the integral inequality constraint, we specify the integrand and the vector of boundary conditions:

```
>> expr = C*u(x,2)^2 - u(x,1)^2;
>> bc = [u(0); u(2*pi); u(0,1)-u(2*pi,1)];
```

3. Solve the problem

We seek upper and lower bounds on the lowest possible C. QUINOPT's default behaviour is to compute upper bounds on the optimal objective value by formulating and inner approximation of the feasible set of the integral inequality constraints. A lower bound is found by overriding the default method with an `options` structure:

```
>> options.method = 'outer';
```

This makes QUINOPT formulate an outer approximation of the feasible set of the integral inequality constraints. We can also tell QUINOPT to run quietly by setting YALMIP's "verbose" option to 0, and cache YALMIP's available solvers to improve YALMIP's performance:

```
>> options.YALMIP = sdpsettings('verbose',0,'cachesolvers',1);
```

To compute a lower bound on the optimal C we can then simply run

```
>> quinopt(expr,bc,C,options);
>> LB = value(C);                    % extract the lower bound on the optimal C
```

To compute an upper bound, we need to reset QUINOPT's default behaviour:

```
>> options.method = 'inner';          % reset the default behaviour: inner␣
↪approximation
>> quinopt(expr,bc,C,options);
>> UB = value(C);                    % extract the upper bound on the optimal C
```

Note: The commands above return an upper bound‘‘UB = 1.000618‘‘ , but a lower bound `LB = NaN`. This is because QUINOPT's default outer approximation is always feasible, and so the optimization problem that is solved has an unbounded objective value. This issue is resolved in the next section.

4. Improve the results

As we have seen, the lower bound obtained with QUINOPT's default outer approximation is not good. This issue can be resolved by refining the approximation that QUINOPT builds. Roughly speaking, QUINOPT builds such approximations by expanding the dependent variables as polynomials of degree N. By default, QUINOPT determines N based on the problem (see our paper for details): for Wirtinger's inequality, the default value is $N = 2$. Fortunately, we can tell QUINOPT to use a larger value by specifying the option `options.N`:

```
>> options.N = 3;                    % use polynomial expansions of degree N=3
>> options.method = 'outer';         % use outer approximation
>> quinopt(expr,bc,C,options);
>> LB = value(C);                    % extract the improved lower bound on the␣
↪optimal C
>> options.method = 'inner';         % use inner approximation
>> quinopt(expr,bc,C,options);
>> UB = value(C);                    % extract the improved upper bound on the␣
↪optimal C
```

The lower bound obtained with these settings is `LB = 0.657974`, and the upper bound improves to `UB=1.000034`. Increasing N further, we see that the two values converge to 1:

N	Lower bound	Upper bound	Difference
2 (default)	NaN	1.000618	NaN
3	0.657974	1.000034	3.42e-01
4	0.939960	1.000001	6.00e-02
5	0.992796	1.000000	7.20e-03
6	0.999413	1.000000	5.87e-04
7	0.999966	1.000000	3.35e-05
8	0.999999	1.000000	1.17e-06
9	1.000000	1.000000	1.97e-08

Note: If `options.N` is lower than the minimum value (again, see our paper for details), QUINOPT issues a warning and uses the minimum value of N instead.

5. Summary

In summary, the optimal constant for Wirtinger's inequality can be determined with the following simple lines of code:

```
>> % Clean up
>> clear;
>> yalmip clear;
>> quinopt clear;
>> % Set up the problem
>> x = indvar(0,2*pi);
>> u = depvar(x);
>> parameters C;
>> expr = C*u(x,2)^2 - u(x,1)^2;
>> bc = [u(0); u(2*pi); u(0,1)-u(2*pi,1)];
>> % Set options for YALMIP
>> options.YALMIP = sdpsettings('verbose',0,'cachesolvers',1);
>> % Compute a lower bound on the optimal C
>> options.method = 'outer';
>> quinopt(expr,bc,C,options);
>> LB = value(C);
>> % Compute an upper bound on the optimal C
>> options.method = 'inner';
>> quinopt(expr,bc,C,options);
>> UB = value(C);
>> % Improve the solution by setting options.N
>> options.N = 9;
>> options.method = 'outer';
>> quinopt(expr,bc,C,options);
>> LB = value(C);
>> options.method = 'inner';
>> quinopt(expr,bc,C,options);
>> UB = value(C);
```

- *Back to Table of Contents*

5.1.5 Poincaré's inequality for odd and periodic functions

Poincaré's inequality for *odd* functions $u : [-1, 1] \to \mathbb{R}$ that satify the periodicity condition

$$u(-1) = u(1)$$

states that

$$\int_{-1}^{1} |u'(x)|^2 \mathrm{d}x \geq \pi^2 \int_{0}^{1} |u(x)|^2 \mathrm{d}x.$$

In this example, we verify that the constant π^2 on the right-hand side is optimal, in the sense that it is the optimal value for the optimization problem

$$\begin{aligned} \max_{\nu} \quad & \nu \\ \text{subject to} \quad & \int_{-1}^{1} \left[|u'(x)|^2 - \nu |u(x)|^2\right] \, \mathrm{d}x \geq 0, \quad u \in C^1([0,1], \mathbb{R}), \begin{cases} u(-1) = u(1), \\ u(-x) = -u(x). \end{cases} \end{aligned}$$

The aim of this example is to show how QUINOPT can handle symmetry constraints on the dependent variables.

`Download the MATLAB file for this example`

1. Create the variables

As usual, we start by clearing the workspace, YALMIP's and QUINOPT's internal variables, and creating the variables for the problem:

```
>> clear
>> yalmip clear
>> quinopt clear
>> x = indvar(-1,1);      % the integration variable with domain [-1,1]
>> u = depvar(x);         % the dependent variable u(x)
>> parameters nu;         % the optimization variable nu
```

2. Set up the inequality

To set up Poincaré's inequality constraint, first we specify the integrand:

```
>> EXPR = u(x,1)^2 - nu*u(x)^2;
```

Then, we set the boundary and symmetry conditions on $u(x)$. The periodic boundary conditions is enforced as $u(-1) - u(1) = 0$, while the symmetry condition can be enforced using the command `assume()`:

```
>> BC = [ u(-1)-u(1) ];
>> assume(u,'odd')
```

Note: Other valid assumptions are `assume(u,'even')` to assume that $u(x)$ is even, and `assume(u,'none')` to remove any previous assumption. Moreover, when the domain of the independent variable used to construct the dependent variable `u` is a generic interval $[a, b]$ rather than a symmetric interval $[-a, a]$, the symmetry condition set with the command `assume()` is relative to the midpoint of the domain, $(a + b)/2$.

---

3. Solve the problem

To solve the problem and maximize ν, we use the command `quinopt()` with three arguments: `EXPR`, `BC` and the objective function. Since QUINOPT minimizes the specified objective function, instead of maximizing ν we minimize $-\nu$.

```
>> quinopt(EXPR,BC,-nu);        % Maximize nu (by minimizing -nu)
>> value(nu)/pi^2               % Get the optimal value (in units of pi^2)
```

With the default parameters in QUINOPT, we obtain $\nu_{\text{opt}} = 0.8184\,\pi^2$, i.e. the optimal solution returned by QUINOPT is within 81.8% of true optimum $\nu_{\text{exact}} = \pi^2$. In fact, we can refine the approximation of the integral inequalities by increasing the number of Legendre coefficients used by QUINOPT to expand them. We do this by setting the option `options.N`, as in *the previous example*:

```
>> options.N = 5;               % Use N=5 expansion coefficients
>> quinopt(EXPR,BC,-nu);        % Maximize nu (by minimizing -nu)
>> value(nu)/pi^2               % Get the optimal value (in units of pi^2)
```

The optimal value of ν returned by QUINOPT in this case is $\nu_{\text{opt}} = 0.999965\,\pi^2$, meaning that the numerical optimum is essentially indistinguishable from the true optimum $\nu_{\text{exact}} = \pi^2$. Setting `options.N` to larger values further improves the numerical optimum (note that roundoff errors might result in a numerical optimum that is slightly larger than the exact solution).

4. Summary

In summary, the optimal constant for Poincaré's inequality for odd, periodic functions can be determined with the following simple lines of code:

```
>> % Set up the variables
>> clear
>> quinopt clear
>> x = indvar(-1,1);
>> u = depvar(x);
>> parameters nu;
>> % Build the inequality
>> EXPR = u(x,1)^2 - nu*u(x)^2;
>> BC = [ u(-1)-u(1) ];
>> assume(u,'odd')
>> % Solve with the default parameters
>> quinopt(EXPR,BC,-nu);
>> value(nu)/pi^2
>> % Refine the solution: solve with N=5 expansion coefficients
>> options.N = 5;
>> quinopt(EXPR,BC,-nu,options);
>> value(nu)/pi^2
```

- *Back to Table of Contents*

5.1.6 Lyapunov stability of a linear PDE

Consider the partial differential equation (PDE)

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + (k - 24\,x + 24\,x^2)u,$$

subject to the boundary conditions $u(0) = 0 = u(1)$.

In this example we use QUINOPT to find a polynomial $p(x)$ of degree 4 such that the functional

$$\mathcal{V}(u) = \int_0^1 p(x)\,|u(x)|^2\,\mathrm{d}x$$

is a Lyapunov function proving the global stability of the zero solution $u(x) = 0$ when $k = 15$. This means that we must find $p(x)$ such that, for some constant $c > 0$,

$$\mathcal{V}(u) = \int_0^1 p(x)\,|u(x)|^2\,\mathrm{d}x \geq c\int_0^1 |u(x)|^2,$$
$$-\frac{\mathrm{d}\mathcal{V}}{\mathrm{d}t} = \int_0^1 -2\,p(x)\,u(x)\,\frac{\partial u}{\partial t}\,\mathrm{d}x \geq 0.$$

Note that since both inequalities are homogeneous in $p(x)$, we may without loss of generality take $c = 1$.

`Download the MATLAB file for this example`

1. Set up the variables

As usual, we start by cleaning up from previous work and defining the basic problem variables:

```
>> clear
>> yalmip clear
>> quinopt clear
>> x = indvar(0,1);
>> u = depvar(x);
```

Then, we set up the PDE:

```
>> k = 15;
>> u_t = u(x,2) + (k-24*x+24*x^2)*u(x);      % the right-hand side of the PDE
>> bc = [u(0), u(1)];                        % the boundary conditions
```

Finally, we define the polynomial p(x), the coefficients of which are the optimization variable. We use the command `legpoly()` to define a polynomial with variable coefficients in the Legendre basis (this is convenient because QUINOPT represents the variables with Legendre series expansions internally):

```
>> p = legpoly(x,4);                    % Create the variable polynomial p(x)
```

Note:

The coefficients of $p(x)$ can be obtained using the command `coefficients()`:

```
>> c = coefficients(p); % c is a vector containing the variable Legendre coefficients␣
↪of p
```

or, more simply, by using two outputs when creating $p(x)$ with the command `legpoly()`

```
>> [p,c] = legpoly(x,4); % c is a vector containing the variable Legendre␣
↪coefficients of p
```

---

2. Set up & solve the integral inequalities

To set up the inequalities in QUINOPT, simply specify their integrands as the elements of a vector `EXPR`:

```
>> EXPR(1) = (p-1)*u(x)^2;              % \int p(x)*u(x)^2 >= \int u(x)^2
>> EXPR(2) = -2*p*u(x)*u_t;             % -V_t(u) >=0.
```

To solve for p(x) and obtain some information about the solution, we use the command `quinopt()` with one output argument:

```
>> SOL = quinopt(EXPR,bc);
```

The output `SOL` contains information about the solution, such as the CPU time taken to set up and solve the problem. In particular,the field `SOL.FeasCode` indicates whether the problem was solved successfully (in this case, `SOL.FeasCode==0`). For more information on QUINOPT's outputs and the meaning of feasibility codes, check

```
>> help quinopt
>> quinoptFeasCode
```

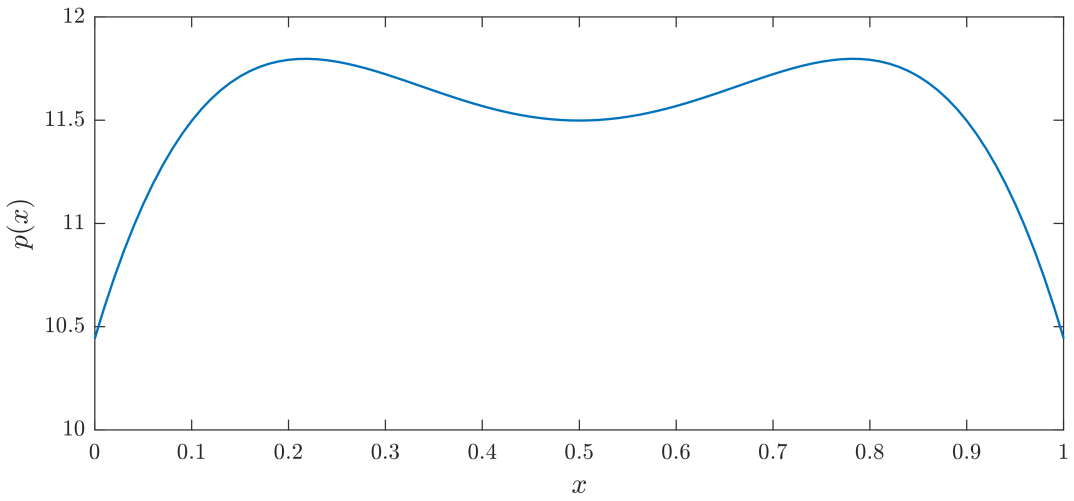
Note: It may be shown that the PDE we are analysing is unstable if $k = 16$. In this case, no suitable polynomial $p(x)$ exists, and the feasibility problem solved by QUINOPT is infeasible. This can be verified by checking the value of `SOL.FeasCode`.

4. Plot $p(x)$

Once a feasible polynomial $p(x)$ is found, as is the case when $k = 15$, one may wish to see what it looks like. MATLAB's usual `plot()` function is overloaded on polynomials defined with the command `legpoly()`, making it really easy to plot $p(x)$. Juse type:

```
>> xval = 0:0.01:1;          % the values of x at which p is plotted
>> plot(xval,p)
```

An example of what $p(x)$ might look like is shown below.



Important: When using `plot()` on a Legendre polynomial variable, the values `xval` must within the domain of the

independent variable `x` originally used to define the polynomial. Otherwise, `plot()` throws an error. The function `getDomain()` can be used to check the domain over which a Legendre polynomial is defined.

- *Back to Table of Contents*

5.1.7 Energy stability of a stress-driven shear flow

Consider an idealized two-dimensional layer of fluid, periodic in the horizontal (x) direction with period Λ, bounded below ($y = 0$) by a solid wall, and driven at the surface ($y = 1$) by a shear stress of non-dimensional magnitude G (known as the *Grashoff number*).

The linear velocity profile $Gy\,\hat{\boldsymbol{\imath}}$, where $\hat{\boldsymbol{\imath}}$ is the unit vector along the x direction, is globally stable when G is sufficiently small. In particular it can be shown (see e.g. Fantuzzi & Wynn, Phys. Rev. E 93(4), 043308 (2016)) that a velocity perturbation

$$\boldsymbol{u} = u(z)\,\mathrm{e}^{ik_n x}\,\hat{\boldsymbol{\imath}} + v(z)\,\mathrm{e}^{ik_n x}\,\hat{\boldsymbol{k}} + \text{complex conjugate}$$

with horizontal wavenumber $k_n = 2\pi\,n/\Lambda$ decays in time if

$$\int_0^1 \left\{ \frac{1}{k_n^2}\left[|u''(y)|^2 + |v''(y)|^2\right] + 2\left[|u'(y)|^2 + |v'(y)|^2\right] \right.$$
$$\left. + k_n^2\left[|u(y)|^2 + |w(y)|^2\right] - \frac{G}{k_n}\left[u(y)\,v'(y) - v(y)\,u'(y)\right] \right\} \mathrm{d}y \geq 0$$

for all functions $u(v)$ and $v(y)$ that satisfy the boundary conditions

$$u(0) = 0, \quad u(1) = 0, \quad u'(0) = 0, \quad u''(1) = 0,$$
$$v(0) = 0, \quad v(1) = 0, \quad v'(0) = 0, \quad v''(1) = 0.$$

This example shows how QUINOPT can be used to determine the maximum Grashoff number G such that, for a given value of k_n, the above stability condition is satisfied. The aim of the example is to illustrate how to define and use multiple dependent variables, and some good practices to solve optimization problems with QUINOPT using a loop.

`Download the MATLAB file for this example`

1. Define some problem parameters and options

As usual, we start by cleaning the workspace and the internal variables in YALMIP and QUINOPT

```
>> clear
>> yalmip clear
>> quinopt clear
```

We then define the horizontal period Λ; in this example, we use $\Lambda = 6\pi$.

```
>> lambda = 6*pi;
```

Furthermore, to run in silent mode we set YALMIP's option 'verbose' to 0, and we set YALMIP's 'cachesolvers' option to 1 to improve YALMIP's performance. Finally, we increase the degree of the Legendre expansion used internally by QUINOPT to 10.

```
>> opts.YALMIP = sdpsettings('verbose',0,'cachesolvers',1);
>> opts.N = 10;
```

2. Maximize G for a given k_n

Let us first consider the problem of finding the maximum Grashoff number that satisfies the integral inequality at the top of the page for the first wavenumber $k_1 = 2\pi/\Lambda$. The variable of integration y, the dependent variables $u(y)$ and $v(y)$, and the Grashoff number G are defined using the commands

```
>> y = indvar(0,1);
>> [u,v] = depvar(y);
>> parameters G;
```

Note: The syntax `[u1,u2,...uN] = depvar(x)` creates N dependent variables `u1`, `u2`, ..., `uN` that depend on the same independent variable `x`.

The integrand of the integral inequality at the top of the page and the boundary conditions on the dependent variables can then be set up in the usual way:

```
>> k = 2*pi/lambda;
>> expr = ( u(y,2)^2+v(y,2)^2 )/k^2 + 2*( u(y,1)^2+v(y,1)^2 ) + k^2*( u(y)^2+v(y)^2 )␣
↪- G/k*( u(y)*v(y,1) - u(y,1)*v(y) );
>> bc = [u(0); u(1); u(0,1); u(1,2)];              % boundary conditions on u
>> bc = [bc; v(0); v(1); v(0,1); v(1,2)];          % boundary conditions on v
```

Finally, the maximum G for which the stability condition is satisfied is computed by calling

```
>> quinopt(expr,bc,-G,opts);
>> LB = value(G);
```

Note that the commands above maximize G using an inner approximation of the integral inequality (the default in QUINOPT) so the optimal value `LB` represents a lower bound on the "true" optimal G. An upper bound can be computed by asking QUINOPT to use an outer approximation:

```
>> opts.method = 'outer';
>> quinopt(expr,bc,-G,opts);
>> UB = value(G);
```

3. Maximize G for multiple wavenumbers: using QUINOPT in a loop

We now turn our attention to computing the maximum Grashoff number that satisfies the integral inequality at the top of the page not for a single wavenumber, but for all wavenumbers up to the maximum value $k_{\max}$. Since the variables and the boundary conditions are the same for all values of the wavenumber k_n, this could be achieved with the following `while` loop (we take $k_{\max} = 5$):

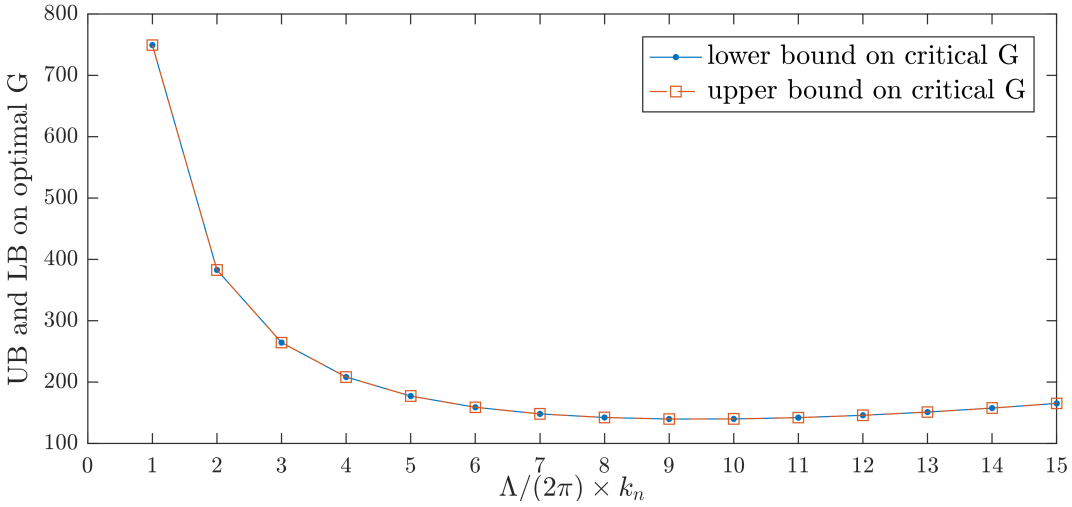
```
>> k = 0;                                          % initial dummy value for k
>> k_max = 5;                                      % maximum wavenumber to solve for
>> n = 1;                                          % start from n=1
>> while k<=k_max
>>     % Set the wavenumber
>>     k = 2*pi*n/lambda;
```

(continues on next page)

(continued from previous page)

```
>>      % Set up and solve the problem
>>      expr = ( u(y,2)^2+v(y,2)^2 )/k^2 + 2*( u(y,1)^2+v(y,1)^2 ) + k^2*( u(y)^2+v(y)^
↪2 ) - G/k*( u(y)*v(y,1) - u(y,1)*v(y) );
>>
>>      opts.method = 'inner';
>>      quinopt(expr,bc,-G,opts);
>>      LB(n) = value(G);
>>      opts.method = 'outer';
>>      quinopt(expr,bc,-G,opts);
>>      UB(n) = value(G);
>>      % update n for the next iteration
>>      n = n+1;
>> end
```

The upper and lower bounds obtained with QUINOPT using the lines of code above are plotted below.



Important: When the number of iterations in the loop is large the build-up of internal variables in YALMIP and QUINOPT due to repeated calls to `quinopt()` could result in significant loss of computational performance. To avoid this, it may be better to clear YALMIP's and QUINOPT's variables after each iteration, and re-initialize them every time. For example, the `while` loop above could be replaced by:

```
>> k = 0;                                    % initial dummy value for k
>> k_max = 5;                                % maximum wavenumber to solve for
>> n = 1;                                    % start from n=1
>> while k<=k_max
>>      % Set the wavenumber
>>      k = 2*pi*n/lambda;
>>      % Define the problem variables at the start of each iteration
>>      y = indvar(0,1);
>>      [u,v] = depvar(y);
>>      parameters G;
>>      % Set up the problem, including the boundary conditions
>>      expr = ( u(y,2)^2+v(y,2)^2 )/k^2 + 2*( u(y,1)^2+v(y,1)^2 ) + k^2*( u(y)^2+v(y)^
↪2 ) - G/k*( u(y)*v(y,1) - u(y,1)*v(y) );
>>      bc = [u(0); u(1); u(0,1); u(1,2)];          % boundary conditions on u
>>      bc = [bc; v(0); v(1); v(0,1); v(1,2)];      % boundary conditions on v
>>      % Solve using inner and outer approximations
```

(continues on next page)

(continued from previous page)

```
>>     opts.method = 'inner';
>>     quinopt(expr,bc,-G,opts);
>>     LB(n) = value(G);
>>     opts.method = 'outer';
>>     quinopt(expr,bc,-G,opts);
>>     UB(n) = value(G);
>>     % Clear YALMIP's and QUINOPT's internal variables
>>     yalmip clear
>>     quinopt clear
>>     % update n for the next iteration
>>     n = n+1;
>> end
```

• *Back to Table of Contents*

5.1.8 Bounds on energy dissipation for a stress-driven shear flow

As in the previous example, consider an idealized two-dimensional layer of fluid, periodic in the horizontal (x) direction with period Λ, bounded below ($y = 0$) by a solid wall, and driven at the surface ($y = 1$) by a shear stress of non-dimensional magnitude G (known as the *Grashoff number*).

Suppose that we can find a *background field* $\varphi(y)$ that satisfies the boundary conditions

$$\varphi(0) = 0, \quad \varphi'(1) = G.$$

and such that the integral constraint

$$\int_0^1 \left\{ \frac{1}{k_n^2} \left[|u''(y)|^2 + |v''(y)|^2 \right] + 2 \left[|u'(y)|^2 + |v'(y)|^2 \right] \right.$$
$$\left. + k_n^2 \left[|u(y)|^2 + |w(y)|^2 \right] - \frac{2}{k_n} \varphi'(y) \left[u(y)\, v'(y) - v(y)\, u'(y) \right] \right\} \mathrm{d}y \geq 0$$

holds for all wavenumbers k_n and all functions $u(v)$ and $v(y)$ that satisfy the boundary conditions

$$u(0) = 0, \quad u(1) = 0, \quad u'(0) = 0, \quad u''(1) = 0,$$
$$v(0) = 0, \quad v(1) = 0, \quad v'(0) = 0, \quad v''(1) = 0.$$

Then, the time-averaged bulk energy dissipation coefficient C_ε can be bounded according to

$$C_\varepsilon \leq G \times \left[2\varphi(1) - \int_0^1 |\varphi'(y)|^2 \, \mathrm{d}y \right]^{-2}.$$

For more details, see e.g. Fantuzzi & Wynn, Phys. Rev. E 93(4), 043308 (2016).

This example shows how QUINOPT can be used to construt a polynomial background field $\varphi(y)$ to maximize

$$\mathcal{B}(\varphi) = 2\varphi(1) - \int_0^1 |\varphi'(y)|^2 \, \mathrm{d}y,$$

and thereby minimize the bound on the time-averaged bulk energy dissipation, when $G = 1000$ and $\Lambda = 2$. For simplicity, we assume that it suffices to check the integral inequality constraint for $k_1 = 2\pi/\Lambda = \pi$. The aim of the example is to illustrate how constraints on the optimization variables can be specified in addition to integral inequality constraints.

Download the MATLAB file for this example

1. Define the problem variables

We begin by clearing the workspace and defining the problem parameters.

```
>> clear
>> yalmip clear
>> quinopt clear
>> lambda = 2;
>> G = 1000;
```

Then, we define the independent and dependent variables with the commands

```
>> y = indvar(0,1);
>> [u,v] = depvar(y);
```

Then, we construct the polynomial background field $\varphi(y)$. We will take $\varphi(y)$ to have degree 20. Since QUINOPT represents polynomials in the Legendre basis internally, we define $\varphi(y)$ in the Legendre basis directly using the command `legpoly()`.

```
>> phi = legpoly(y,20);
```

Moreover, the integral inequality constraint depends on the first derivative of $\varphi(y)$. This is easily found using the function `jacobian()`:

```
>> D1phi = jacobian(phi,y);
```

Finally, we need to specify the boundary conditions on the background field, $\varphi(0) = 0$ and $\varphi'(1) = G$. These can be specified in a vector of constraints like any standard YALMIP constraint, and the boundary values $\varphi(0)$ and $\varphi'(1)$ can be accessed using the function `legpolyval()`:

```
>> CNSTR(1) = legpolyval(phi,0)==0;              % \phi(0)=0
>> CNSTR(2) = legpolyval(D1phi,1)==G;            % \phi'(1)=G
```

2. Set up the optimization problem

The integral inequality constraint can be set up, as usual, by defining its integrand and the boundary conditions on the dependent variables:

```
>> k = pi;
>> EXPR = ( u(y,2)^2+v(y,2)^2 )/k^2 + 2*( u(y,1)^2+v(y,1)^2 ) + k^2*( u(y)^2+v(y)^2 )␣
↪- 2*D1phi/k*( u(y)*v(y,1) - u(y,1)*v(y) );
>> BC = [u(0); u(1); u(0,1); u(1,2)];          % boundary conditions on u
>> BC = [BC; v(0); v(1); v(0,1); v(1,2)];      % boundary conditions on v
```

The objective function $\mathcal{B}(\varphi)$, to be maximised, can be set up using the command `legpolyint()` to compute the boundary value $\varphi(1)$, and the command `int()` to integrate the square of $\varphi'(y)$ over $[0, 1]$:

```
>> OBJ = 2*legpolyval(phi,1) - int(D1phi^2,y,0,1)/G;
```

3. Solve and plot the optimal $\varphi(y)$

Having defined all variables and constraints, we can maximize the objective function `OBJ` using the syntax

```
>> quinopt(EXPR,BC,-OBJ,[],CNSTR)
```

Note: The first two arguments specify the integral inequality constraint, while the additional constraints are specified in the fifth argument `CNSTR`. The fourth argument specifies QUINOPT's options, and here it is left empty to use the default options. Finally, note the minus sign in the objective function, which is needed because QUINOPT minimizes the specified objective function by default.

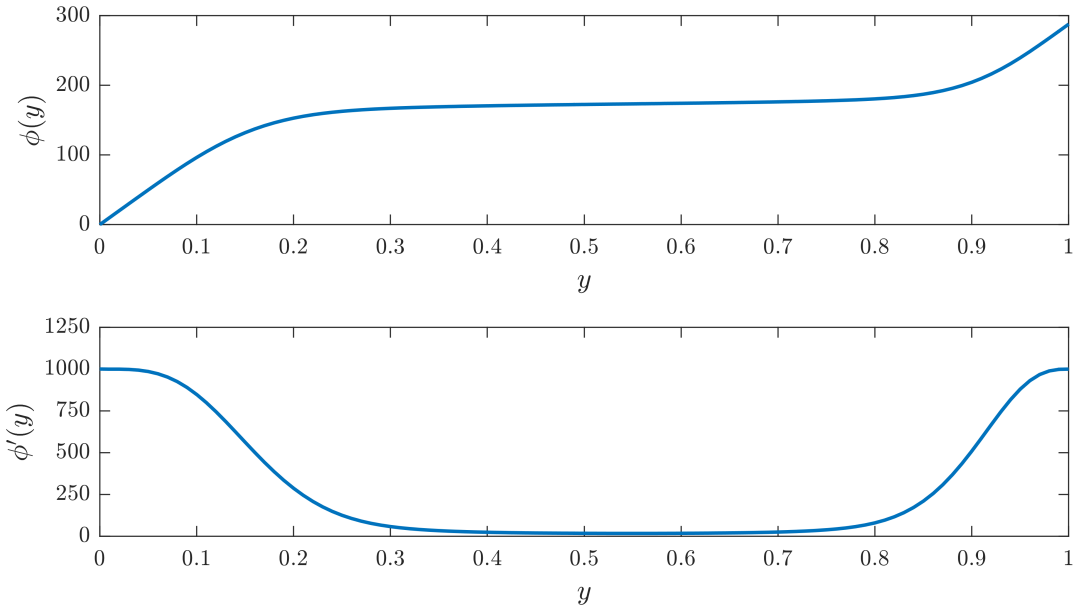
Once the problem is (successfully) solved, we can compute the upper bound on the dissipation coefficient C_ε with

```
>> UB = G/(value(OBJ))^2;
```

to find $C_\varepsilon \leq 7.48 \times 10^{-3}$ approximately. Finally, we can plot the optimal background field $\varphi(y)$ and its first derivative using the command `plot()`, which is overloaded on polynomials defined using the function `legpoly()`:

```
>> subplot(2,1,1)
>> plot(0:0.01:1,phi,'-','LineWidth',1.5);
>> subplot(2,1,2)
>> plot(0:0.01:1,D1phi,'-','LineWidth',1.5);
```

This produces the figure below; note that the boundary conditions $\varphi(0) = 0$ and $\varphi'(1) = G\,(= 1000)$ are indeed satisfied.



- *Back to Table of Contents*

5.1.9 Bounds on energy dissipation for plane Couette flow

In this example we will compute bounds on the infinite-time-and-volume-averaged energy dissipation for plane Couette flow using the *auxiliary functional method* (see Chernyshenko et al., Philos. Trans. R. Soc. A 372, 20130350 (2014) and Chernyshenko, arXiv:1704.02475 [physics.flu-dyn] (2017)) with a quadratic storage functional.

`Download the MATLAB file for this example`

Warning: This example may take a few minutes to run: it took 53 seconds on a linux machine with a 3.40GHz Intel Core i7-4770 CPU, running MATLAB2016b and using MOSEK as the SDP solver.

1. Description of the flow

Plane Couette flow describes the motion of an incompressible fluid between two horizontal parallel plates at a vertical distance of 1 unit, the top one of which moves along the x direction with unit velocity. The system admits the steady (i.e., time independent) flow velocity field $\boldsymbol{U}_l = z\,\hat{\boldsymbol{\imath}}$, and velocity perturbations $\boldsymbol{v} = (u, v, w)$ with respect to this state obey the governing equations

$$\frac{\partial \boldsymbol{v}}{\partial t} = Re^{-1}\nabla^2\boldsymbol{v} - \nabla p - (\boldsymbol{v}\cdot\nabla)\boldsymbol{v} - (\boldsymbol{v}\cdot\nabla)\boldsymbol{U}_l - (\boldsymbol{U}_l\cdot\nabla)\boldsymbol{v},$$
$$\nabla\cdot\boldsymbol{v} = 0.$$

In these equations, Re is the Reynolds number and p is the pressure. The boundary conditions are periodic in the horizontal directions (x and y) with periods Λ_x and Λ_y, plus

$$\boldsymbol{v}|_{z=0} = 0, \qquad \boldsymbol{v}|_{z=1} = 0.$$

If $\Omega = [0, \Lambda_x] \times [0, \Lambda_y] \times [0, 1]$ is the fluid's domain, the volume-averaged energy dissipation is defined at any instant in time as

$$\begin{aligned}\varepsilon(t) &= \frac{1}{\Lambda_x\Lambda_y}\int_\Omega |\nabla(\boldsymbol{v} + \boldsymbol{U}_l)|^2 d\Omega. \\ &= 1 + \frac{1}{\Lambda_x\Lambda_y}\int_\Omega |\nabla\boldsymbol{v}|^2 d\Omega.\end{aligned}$$

The second equality can be easily proven using integration by parts, the boundary conditions on $\boldsymbol{v}$, and the definition of $\boldsymbol{U}_l$.

Note: We consider the equations of motion of the flow relative to the laminar flow to so the dynamic variable $\boldsymbol{v}$ satisfies homogeneous boundary conditions. This is done because currently QUINOPT only allows one to specify homogeneous boundary conditions. Inhomogeneous boundary conditions should be "lifted" at the modelling stage in order to use QUINOPT.

2. The auxiliary functional method

According to the auxiliary functional method, U is an upper bound on the infinite-time average of $\varepsilon(t)$ if there exists a functional $\mathcal{V}(t) = \mathcal{V}[\boldsymbol{v}(t,\cdot)]$ that satisfies the bounding inequality

$$\frac{d\mathcal{V}}{dt} + \varepsilon \le U.$$

Here, we choose

$$\mathcal{V}(t) = \frac{1}{\Lambda_x\Lambda_y}\int_\Omega \left[\frac{a\,Re}{2}|\boldsymbol{v}|^2 - Re\,\varphi(z)\,\hat{\boldsymbol{\imath}}\cdot\boldsymbol{v}\right] d\Omega.$$

with $a \in \mathbb{R}$ and the function $\varphi(z)$ to be determined such that the bounding inequality is satisfied. We also assume that $\varphi(0) = \varphi(1) = 0$. The bounds obtained with the auxiliary functional method are then the same as those obtained with the *background method* (see e.g. Plasting & Kerswell, J. Fluid Mech. 477, 363–379 (2003)).

With these choices, the bounding inequality can be rearranged into

$$\frac{1}{\Lambda_x \Lambda_y} \int_\Omega \left[U - a\, Re\, \boldsymbol{v} \cdot \frac{\partial \boldsymbol{v}}{\partial t} + Re\, \varphi(z)\, \hat{\boldsymbol{\imath}} \cdot \frac{\partial \boldsymbol{v}}{\partial t} - 1 - |\nabla \boldsymbol{v}|^2 \right] d\Omega \geq 0.$$

Upon

1. Substituting the equations of motions, integrating by parts using the boundary and incompressibility conditions, and
2. Fourier-transforming in the horizontal directions assuming that the critical velocity perturbations $\boldsymbol{v}$ are *streamwise invariant*, i.e. independent of x,

it can be shown that the last inequality above is equivalent to the infinite set of conditions

$$\int_0^1 \left[(a-1)|\hat{u}_0'(z)|^2 + \varphi''(z)\, \hat{u}_0(z) + U - 1 \right] dz \geq 0,$$

$$\int_0^1 \Big\{ (a-1) \left[|\hat{u}_k'(z)|^2 + k^2 |\hat{u}_k(z)|^2 \right] + (a-1) \left[\frac{1}{k^2} |\hat{w}_k''(z)|^2 + 2|\hat{w}_k'(z)|^2 + k^2 |\hat{w}_k(z)|^2 \right] + Re[a + \varphi'(z)]\, \hat{u}_k(z)\, \hat{w}_k(z) \Big\}\, dz \geq 0, \qquad \forall k = \frac{2\pi \mathbb{N}_+}{\Lambda_y}.$$

In these equations primes denote differentiation with respect to z, while $\hat{u}_k$ and $\hat{w}_k$ denote the k-th Fourier amplitudes of the velocity perturbation components u and w, respectively. It can also be shown that they may be assumed to be real without loss of generality, and that for all $k \geq 0$ they must satisfy the boundary conditions

$$\hat{u}_k(0) = \hat{u}_k(1) = 0, \quad \hat{w}_k(0) = \hat{w}_k(1) = 0, \quad \hat{w}_k'(0) = \hat{w}_k'(1) = 0.$$

3. Optimization of the bound U with QUINOPT

To compute the optimal bound on the infinite-time average of the bulk energy dissipation $\varepsilon(t)$ for plane Couette flow, we use QUINOPT to minimize U subject to the set of inequalities derived in the previous section.

As usual, we begin by clearing the workspace and defining some of the problem parameters. For illustration, we consider $Re = 500$ and $\Lambda_y = 4\,\pi$.

```
>> clear;                  % clear the workspace
>> yalmip clear;           % clear YALMIP's internal variables
>> quinopt clear;          % clear QUINOPT's internal variables
>> Re = 500;
>> Lambda_y = 4*pi;
```

We then proceed to define the independent variable of integration z, the dependent variables $\hat{u}_k$ and $\hat{w}_k$ (we will drop the subscript k and the hats in the code), and the boundary conditions.

```
>> z = indvar(0,1);
>> [u,w] = depvar(z);
>> BC = [u(0); u(1); w(0); w(1); w(0,1); w(1,1)];
```

Note: When a problem has multiple integral inequality constraints with the same integration domain, there is no need to define different independent and dependend variables for each. Since the dependent variables are simply symbolic variables in MATLAB, they can be re-used when defining multiple inequalities (provided the integration domain is the same).

We now need to define the optimization variables of the problem, i.e. the scalars a and U, and the function $\varphi(z)$. We take $\varphi(z)$ to be a polynomial of degree 25 in the Legendre basis, constructed using the command `legpoly()`. Moreover, we enforce the boundary conditions $\varphi(0) = 0 = \varphi(1)$ using the command `legpolyval()` to evaluate Legendre polynomials. The code reads:

```
>> parameters a U
>> PHI = legpoly(z,25);
>> PHI_BC = [legpolyval(PHI,0)==0; legpolyval(PHI,1)==0];
```

In order to define the integral inequality constraints, we need the first and second derivatives of $\varphi(z)$. These are easily obtained with the command `jacobian()`:

```
>> D1PHI = jacobian(PHI,z);
>> D2PHI = jacobian(D1PHI,z);
```

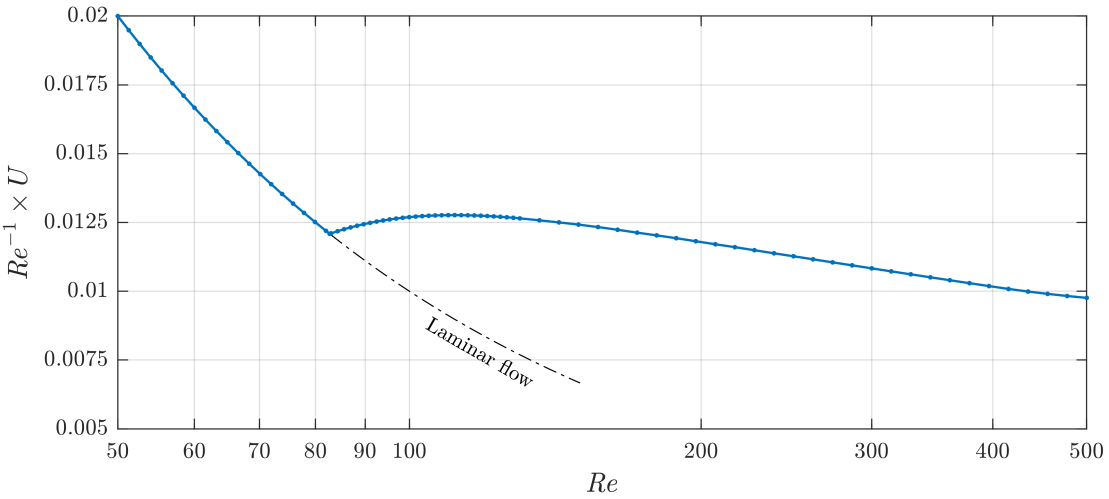
We are now in a position to define the integral inequalities. Of course, only a finite number of wavenumbers can be implemented in QUINOPT; in this example, we consider $k \leq 10$ only.

```
>> EXPR =  (a-1)*u(z,1)^2 + D2PHI*u(z) + U-1;            % the inequality for k=0
>> n = 0; k = 0;
>> while k < 10
>>       n = n+1;
>>       k = 2*pi*n/Lambda_y;
>>       EXPR(end+1) = (a-1)*( u(z,1)^2 + k^2*u(z)^2 ) +(a-1)*( w(z,2)^2/k^2 + 2*w(z,
↪1)^2 + k^2*w(z)^2 ) + Re*( a+D1PHI )*u(z)*w(z);
>> end
```

Finally, we can solve the problem using the command `quinopt()`. We will use the default options, but we need to pass the boundary condition on $\varphi(z)$ as additional constraint, so we call

```
>> quinopt(EXPR,BC,U,[],PHI_BC);          % solve with additional constraints
>> value(U)                               % extract the optimal value
```

We find $U = 4.8818$. The figure below illustrates how the bound, plotted in terms of the usual friction coefficient $Re^{-1} \times U$, varies with the reynolds number Re. The blue curve replicates the results presented in Figure 2 of Plasting & Kerswell, J. Fluid Mech. 477, 363–379 (2003). Note also that the bound coincides with the laminar dissipation value (indicated by the dot-dashed line in the figure below) up to the well-known energy stability boundary $Re \approx 82.7$.



Note: The results obtained in this example are, strictly speaking, upper bounds on the optimal U. This is because QUINOPT strengthens the integral inequality constraints by default to obtain a finite- dimensional *inner* approximation of their feasible set. Lower bounds on the optimal U can also be found with QUINOPT, by defining a structure `OPTIONS` with a field called `method` set to `'outer'`. Then, QUINOPT will relax the integral inequality constraints to obtain an *outer* approximation of their feasible set. The following snippet of code demonstrate how to do this in practice:

```
>> OPTIONS.method = 'outer';
>> quinopt(EXPR,BC,U,OPTIONS,PHI_BC);        % Call quinopt() with user-defined options
```

Computing both upper and lower bounds is useful to assess how far from "true optimality" the answer returned by QUINOPT is. If needed, the quality of QUINOPT's approximation can be improved as described *in this previous example*. More details regarding inner and outer approximations of the feasible set of integral inequalities can be found in our paper.

- *Back to Table of Contents*

5.1.10 Energy stability of Bénard-Marangoni conduction at infinite Prandtl number

Bénard-Marangoni convection describes the flow of a fluid layer driven by surface tension forces of magnitude described by the *Marangoni number* M. In this example, we consider the idealized case of a unit-depth layer which is infinite in the horizontal directions, and compute the maximum M for which the conductive state (when the fluid is at rest) is stable with respect to sinusoidal temperature perturbations with frequency magnitude k. In particular, the *energy method* shows that conduction is stable if

$$\int_0^1 \left[|T'(z)|^2 + k^2|T(z)|^2 + M\, F_k(z)\, T(z)\, T(1)\right] \mathrm{d}z \geq 0$$

for all functions $T(z)$ that satisfy the boundary conditions $T(0) = 0$ and $T'(1) = 0$, where

$$F_k(z) = \frac{k\, \sinh k}{\sinh(2\,k) - 2\,k}\, \left[k\, z\, \cosh(k\, z) - \sinh(k\, z) + (1 - k\, \coth k)\, z\, \sinh(k\, z)\right].$$

For more details, see e.g. Hagstrom & Doering, Phys. Rev. E 81, 047301 (2010).

The aim of this example is to demonstrate how to solve problems in which unknown boundary values of the dependent variables appear explicitly in the integral inequality constraints. Moreover, we show how QUINOPT can be used to approximate problems with data that is non-polynomial using a truncated Legendre transform.

`Download the MATLAB file for this example`

1. Set up the variables

First, we clear the workspace, as well as QUINOPT's and YALMIP's internal variables:

```
>> clear
>> yalmip clear
>> quinopt clear
```

Then, we define the integration variable $z \in [0, 1]$, the dependent variable $T(z)$, and the Marangoni number M, which is the optimization variable:

```
>> z = indvar(0,1);
>> T = depvar(z);
>> parameters M
```

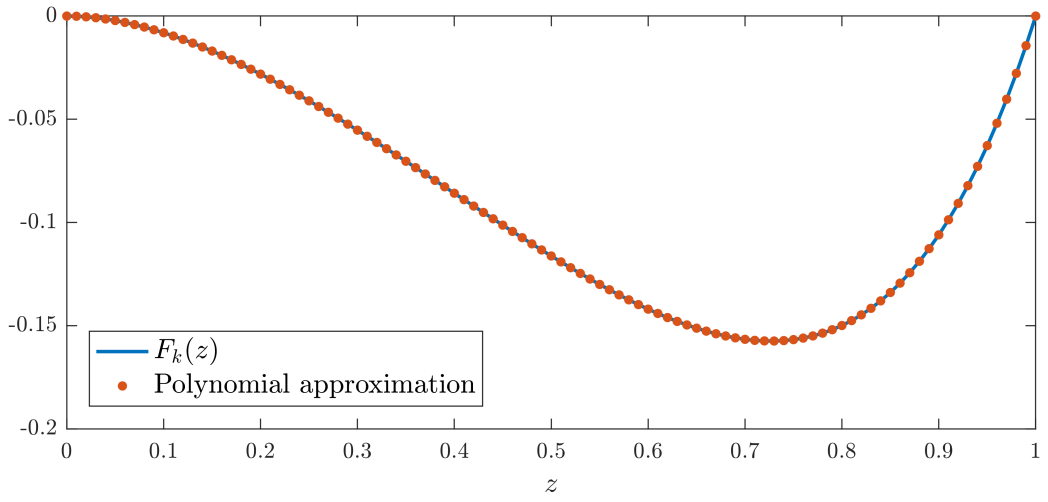
2. Construct a polynomial approximation to $F_k(z)$

QUINOPT can only solve integral inequalities with polynomial data, but the function $F_k(z)$ is clearly not a polynomial. Yet, QUINOPT can be used if we replace $F_k(z)$ with a polynomial approximation of degree d. To do so, we first construct the exact function $F_k(z)$ as a function handle, and we compute its first $d+1$ Legendre expansion coefficients using the *fast Legendre transform* command `flt()`. Finally, we use these coefficients to build a polynomial approximation of degree d using the command `legpoly()`. Below, we set $k = \pi$ and $d = 10$ (this gives an accurate approximation at least up to $k = 5$).

```
>> k = pi;
>> d = 10;
>> Fk = @(z)k*sinh(k)/(sinh(2*k)-2*k).*(k*z.*cosh(k*z)-sinh(k*z)+(1-k*coth(k))*z.
↪*sinh(k*z));
>> leg_coef = flt(Fk,d+1,[0,1]);           % Compute the Legendre expansion␣
↪coefficients
>> FkPoly = legpoly(z,d,leg_coef);         % Build a degree-d legendre polynomial with␣
↪coefficients specified by leg_coef
```

We can easily plot both $F_k(z)$ and its polynomial approximation using the function `plot()`, which is overloaded on polynomials built using `legpoly()`:

```
>> clf;                                                    % clear current figure
>> plot(0:0.01:1,Fk(0:0.01:1),'Linewidth',1.5); hold on;   % plot Fk(z)
>> plot(0:0.01:1,FkPoly,'.','MarkerSize',12); hold off;    % plot the polynomial␣
↪approximation
>> xlabel('$z$','interpreter','latex','fontsize',12);
>> legend('F_k(z)','Polynomial approximation','Location','southwest');
>> axis([0 1 -0.2 0]);
```



Note: The syntax `LCOEF = flt(FUN,NCOEF,DOMAIN)` computes the first `NCOEF` coefficients of the Legendre series expansion of the function specified by the function handle `FUN` over the interval specified by the input `DOMAIN`. These can be used to construct a polynomial approximation to the function specified by `FUN` of degree `NCOEF-1`

(recall that a polynomial of degree d has $d+1$ coefficients). Note that `DOMAIN` should be a bounded interval in the form `[a,b]`, and the function handle `FUN` should only take one input argument with values in the range specified by `DOMAIN`.

3. Maximize M

Once a polynomial approximation of $F_k(z)$ has been constructed, the maximum Marangoni number M satisfying the integral inequality at the top of the page can be computed with QUINOPT. First, we define the integrand of the integral inequality, and the boundary conditions on the dependent variable:

```
>> EXPR = T(z,1)^2 + k^2*T(z)^2 + M*FkPoly*T(z)*T(1);
>> BC = [T(0); T(1,1)];                                % The boundary conditions␣
↪T(0)=0, T'(1)=0
```

Then we maximize M by calling

```
>> quinopt(EXPR,BC,-M)
>> value(M)
```

(note the negative sign in the objective function, which is needed because QUINOPT minimizes the specified objective). The optimal solution is found to be $M \approx 78.55$.

4. Summary

In summary, the maximum Marangoni number M for which a sinusoidal perturbation to the Benard-Marangoni conduction state is stable can be computed with the following lines of code:

```
>> % Clear the workspace, plus YALMIP's and QUINOPT's internal variables
>> clear
>> yalmip clear
>> quinopt clear
>> % Define the problem variables
>> z = indvar(0,1);
>> T = depvar(z);
>> parameters M
>> % Set k and build a polynomial approximation to Fk(z) of degree d=10
>> k = pi;
>> d = 10;
>> Fk = @(z)k*sinh(k)/(sinh(2*k)-2*k).*(k*z.*cosh(k*z)-sinh(k*z)+(1-k*coth(k))*z.
↪*sinh(k*z));
>> leg_coef = flt(Fk,d+1,[0,1]);           % Compute the Legendre expansion␣
↪coefficients
>> FkPoly = legpoly(z,d,leg_coef);         % Build a degree-d legendre polynomial with␣
↪coefficients specified by leg_coef
>> % Set up and solve the optimization problem
>> EXPR = T(z,1)^2 + k^2*T(z)^2 + M*FkPoly*T(z)*T(1);  % the integrand of the␣
↪inequality
>> BC = [T(0); T(1,1)];                                % The boundary conditions␣
↪T(0)=0, T'(1)=0
>> quinopt(EXPR,BC,-M)
>> value(M)
```

- *Back to Table of Contents*

- *Back to Table of Contents*

CHAPTER 6

List of main functions

The main functions needed to work with QUINOPT, which have been used in the examples contained in this documentation, are listed below. Click on a function name to access its documentation.

6.1 `indvar()`

Create an independent variable of integration to define integral inequalities in QUINOPT.

Syntax `x = indvar(a,b)`

Description creates an independent variable of integration with domain $[a, b]$ used to set up an integral inequality constraint with the toolbox QUINOPT.

Warning: An integral inequality constraint defined with QUINOPT can only have one independent variable - multivariable integrals are not allowed. However, one independent variable can be used to define multiple integral inequalities, and different inequalities can have different independent variables (although this is not necessary).

- *Back to the list of main functions*
- *Back to Table of Contents*

6.2 `depvar()`

Define dependent variables to define integral inequalities in QUINOPT.

Syntax `U = depvar(x)`

Description sets up a symbolic variable modelling a generic function $U(x)$, where `x` is a valid independent variable with domain $[a, b]$ created with the command indvar. `U` behaves like a function handle, and is used with the syntax

```
U(POINT)
```

where `POINT` is either the independent variable `x`, the lower extremum a of the domain of `U`, or the upper extremum b of the domain of `U`. Moreover, derivatives of `U` can be created/accessed using the syntax

```
U(POINT,DERIVATIVE)
```

where `POINT` is `x`, a or b and `DERIVATIVE` is the desired derivative order. Note that `U(POINT, 0)` is equivalent to `U(POINT)`.

Syntax `[U1,U2,...Uq] = depvar(x)`

Description sets up multiple dependent variables, `U1`, ..., `Uq`. Each dependent variable depends on the independent variable `x`.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.3 @depvar/assume()

Add assumption on dependent variables in QUINOPT.

Syntax `assume(U,STR)`

Description adds the assumption specified by the character string `STR` on the dependent variable `U` (class `depvar`). Currently, allowed values for `STR` are:

- `'even'`: assume that `U` is symmetric with respect to the midpoint of the domain $[a, b]$ in which the dependent variable `U` is defined.
- `'odd'`: assume that `U` is anty-symmetric with respect to the midpoint of the domain $[a, b]$ in which the dependent variable `U` is defined.
- `'none'`: remove all previous assumptions on the dependent variable `U`.

Syntax `assume(U1,STR1,U2,STR2,...)`

Description adds the assumptions specified by the character strings `STR1`, `STR2`, ..., on the dependent variables `U1`, `U2`, ..., as if set by the commands `assume(U1,STR1)`, `assume(U2,STR2)`, and so on.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.4 parameters()

Create symbolic optimization variables for QUINOPT.

Syntax `P = parameters(m,n)`

Description reates an $m \times n$ matrix of parameters `P` to be used as optimization variables with QUINOPT. The parameters are YALMIP variables (class `sdpvar`).

Syntax `parameters p1 p2 p3 p4`

Description creates multiple scalar optimization parameters, `p1`, ..., `p4`.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.5 quinopt()

Minimize a cost function subject to quadratic integral inequality constraints, or reset QUINOPT.

Syntax `quinopt clear` or `quinopt('clear')`

Description clears QUINOPT's internal variables, to be used in combination with MATLAB's `clear` command.

Syntax `quinopt(EXPR)`

Description tests whether a quadratic integral inequality with integrand specified by `EXPR` is feasible using a finite dimensional relaxation based on semidefinite programming. If multiple integral inequalities must be tested simultaneously, `EXPR` can be a vector such that the i-th entry specifies the integrand of the i-th integral inequality. Each entry of `EXPR` must be a polynomial of the integration variable returned by the command `indvar()`, and a quadratic polynomial of the dependent variables returned by the function `depvar()`.

Syntax `quinopt(EXPR,BC)`

Description determines whether the integral inequalities, with integrand specified by `EXPR`, are feasible for all dependent variables satisfying the homogeneous boundary conditions specified by the vector `BC`. Specifically, `BC` is interpreted as the list of boundary conditions `BC(1)=0`, ..., `BC(end)=0`. Like `EXPR`, `BC` must be created using the variables returned by the commands `indvar()` and `depvar()`.

Syntax `quinopt(EXPR,BC,OBJ)`

Description **minimizes** the objective function `OBJ` constrained by the integral inequalities specified by `EXPR` and `BC`.

Syntax `quinopt(EXPR,BC,OBJ,OPTIONS)`

Description overrides the default options. `OPTIONS` is a structure containing any of the following fields:

- `OPTIONS.YALMIP`: a substructure containing the options for YALMIP, set with YALMIP's command `sdpsettings()`.
- `OPTIONS.N`: an integer specifying the number of Legendre coefficients to use in the expansion of the dependent variable to obtain an SDP-representable relaxation of the quadratic integral inequality.
- `OPTIONS.method`: if set to `'inner'` (default), QUINOPT generates an inner approximation of the feasible set of the integral inequalities specified by `EXPR`, i.e. the quadratic integral inequality is strenghtened. If set to `'outer'`, an outer approximation is constructed, i.e. the integral inequality is relaxed into a weaker constraint.
- `OPTIONS.BCprojectorBasis`: string specifying which basis to use for the projection on the boundary conditions. If set to `'rref'` (default), use a "rational" basis. If set to `'orth'`, use an orthonormal basis. The orthonormal basis may be preferable numerically, but it may destroy sparsity of the data.
- `OPTIONS.sosdeg`: the degree of the sum-of-squares polynomials used in the S-procedure to localize SOS constraints from the integral inequality to the integration domain. Default value: 6.
- `OPTIONS.solve`: if set to `'true'` (default), QUINOPT calls the solver specified by the YALMIP options (or YALMIP's default solver). If set to `'false'`, QUINOPT does not call the solver, but simply sets up the YALMIP problem structure. In this case, additional outputs to QUINOPT are required (see below).

Syntax `quinopt(EXPR,BC,OBJ,OPTIONS,CNSTR)` or `quinopt(EXPR,BC,OBJ,OPTIONS,CNSTR,PARAMETERS)`

Description minimizes the objective function `OBJ` subjet to the integral inequalities specified by `EXPR` and `BC`, and the additional constraints given by `CNSTR`. `CNSTR` is a constraint object built with YALMIP. If `CNSTR` contains sum-of-square constraints, then the variable parameters in the polynomial expressions **must** be specified in the input vector `PARAMETERS`. See YALMIP's function `sos()` and `solvesos()` for more details on specifying sum-of-squares constraints with YALMIP.

Syntax `[SOL,CNSTR,DATA] = quinopt(...)`

Description returns solution informations in the structure `SOL`, the set of YALMIP constraint `CNSTR` used to solve the optimization problem, and a structure `DATA` containing all variables used to set up the constraints in `CNSTR`. The solution structure `SOL` contains the following fields:

- `SOL.setupTime`: the time taken to set up the problem
- `SOL.solutionTime`: the time taken by YALMIP to solve the problem
- `SOL.problem`: code of problem encountered during setup. Values are:
  - `0`: no problem
  - `1`: ill-posed inequality
  - `2`: infeasible relaxation

- `SOL.FeasCode`: code for the feasibility of the solution returned by YALMIP. Run `quinoptFeasCode` at the MATLAB command line to obtain a complete list.
- `SOL.YALMIP`: the solution structure returned by YALMIP. See YALMIP's functions `optimize()` and `solvesos()` for more details.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.6 `legpoly()`

Create polynomial in the Legendre basis to use with QUINOPT.

Syntax `P = legpoly(x,DEG)`

Description creates a polynomial `P` in the independent variable `x` of degree `DEG`, expressed in Legendre basis. That is, the polynomial $P(x)$ is expressed as

$$P(x) = C_1\,\mathcal{L}_0[z(x)] + C_2\,\mathcal{L}_1[z(x)] + \cdots + C_{\mathrm{DEG}+1}\,\mathcal{L}_{\mathrm{DEG}}[z(x)]$$

where $\mathcal{L}_n(z)$ is the Legendre polynomial of degree n. Since Legendre polynomials are defined over the standard domain $[-1, 1]$, the original independent variable x with domain $[a, b]$ is rescaled to

$$z(x) = \frac{2\,x - b - a}{b - a}$$

The input `x` must be a valid independent variable (class `indvar`), and `DEG` should be a non-negative integer. The coefficients of the polynomial are YALMIP variables (class `sdpvar`) and can be recovered with the command `C = coefficients(P)`. Finally, `P` can be displayed symbolically in the standard monomial basis using the command `sdisplay(P)`.

Syntax `[P,C] = legpoly(x,DEG)`

Description also returns the Legendre coefficients of the polynomials in the vector `C`. These are YALMIP variables (class `sdpvar`). The coefficients in `C` are listed in order of increasing degree of the corresponding Legendre polynomial (see above).

Syntax `P = legpoly(x,DEG,COEF)`

Description creates a polynomial `P` expressed in Legendre basis whose coefficients are specified by `COEF`. `COEF` can be a numeric/sdpvar vector, or an $M \times N$ cell array whose entries are numeric/sdpvar vectors. When `COEF` is a cell array, an $M \times N$ matrix of Legendre polynomials is created such that the coefficients of the entry `P(i,j)` are given by `COEF{i,j}`.

Syntax `[P,C] = legpoly(x,DEG,M,N)`

Description creates an $M \times N$ matrix of Legendre polynomials of degree `DEG`. The output `C`, containing the coefficients of each entry of `P`, is optional.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.7 @legpoly/legpolyval()

Evaluate polynomial in Legendre basis (class `legpoly`).

Syntax `F = legpolyval(p,x)`

Description evaluates the polynomial `p` at the points specified by the vector (or matrix) `x`. The points in `x` must be in the interval $[a, b]$ where `p` is defined (this can be recovered by calling `getDomain(p)`).

- *Back to the list of main functions*
- *Back to Table of Contents*

6.8 @legpoly/jacobian()

Differentiate a polynomial in the Legendre basis (class legpoly).

Syntax `J = jacobian(P,x)`

Description computes the derivative of each entry of the matrix `P` of polynomials in the Legendre basis (class `legpoly`) with respect to the independent variable `x` (class `indvar`).

- *Back to the list of main functions*
- *Back to Table of Contents*

6.9 @legpoly/int()

Integrate a polynomial in the Legendre basis (class `legpoly`).

Syntax `P = INT(p)` or `P = INT(p,x)`

Description integrates the polynomial `p` in Legendre basis (class `legpoly`" with respect to its independent variable `x`. Indefinite integration is performed such that $P(0) = 0$. If a different behaviour is required, please use the function `legpolyint()`.

Syntax `P = INT(p,x,a,b)`

Description computes the integral of `p` from `a` to `b`. The integration limits `a` and `b` must be contained within the domain of definition of the polynomial `p`, as returned by calling `getDomain(p)`.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.10 @legpoly/plot()

Short description

Syntax `plot(X,P)`

Description plots the Legendre polynomial `P` at the points specified by the vector `X`. All points in `X` must be within the domain of definition $[a, b]$ of the Legendre polynomial, as returned by calling `getDomain(P)`.

Syntax `PLOT(X,P,LINESPEC)`

Description applies the formatting specified by `LINESPEC`, as in the usual `plot()` command in MATLAB.

- *Back to the list of main functions*
- *Back to Table of Contents*

6.11 flt()

Fast Legendre transform according to the algorithm presented in Iserles, Numer. Math. 117, 529-553 (2010).

Syntax `C = flt(FUN,N,DOMAIN)`

Description computes the first `N` Legendre coefficients of the expansion of the function `FUN`, defined over the domain `DOMAIN`, and returns them in the vector `C`. The input `FUN` must be a handle to the function to be projected onto the first `N` Legendre polynomials, `N` must be a non-negative integer, and `DOMAIN` must be a vector with 2 elements, i.e. `DOMAIN = [a,b]` with `a<b`.

- *Back to the list of main functions*
- *Back to Table of Contents*
- *Back to Table of Contents*

CHAPTER 7

How to cite

If you find QUINOPT useful, please cite the following papers:

[1] G. Fantuzzi, A. Wynn, P. Goulart, A. Papachristodoulou (2017). Optimization with affine homogeneous quadratic integral inequality constraints, *IEEE Transactions on Automatic Control* 62(12), 6221-6236, 2017.
(DOI - arXiv - `BibTex`)

[2] G. Fantuzzi and A. Wynn (2016). Semidefinite relaxation of a class of quadratic integral inequalities. In: *Proceedings of the 55th IEEE Conference on Decision and Control*, Las Vegas (NV), USA, pp. 6192-6197.
(DOI - `BibTex`).

In addition, you are welcome to cite the source code for the latest stable release (version 2.2) as

[3] G. Fantuzzi, A. Wynn, P. Goulart, A. Papachristodoulou, *QUINOPT version* 2.2, 2017. Available from: https://github.com/aeroimperial-optimization/QUINOPT.
(`BibTex`)

Note: A selection of BibTex styles that support arXiv preprints can be found here.

- *Back to Table of Contents*

CHAPTER 8

Support

8.1 Getting help

The easiest way to get help on QUINOPT's functions is to use the `help` command in MATLAB, and look at the source code. If you still need help, you can contact us. If you find a bug, or have an issue, see the next section.

8.2 Bug reports and support

Please report any issues via the Github issue tracker, or contact us. All types of issues are welcome, including bug reports, documentation typos, and requests for new features.

- *Back to Table of Contents*