University of Konstanz
Department of Mathematics and Statistics
Summer Term 2012

Sebastian Gruler
María López Quijorna
Markus Schweighofer

Polynomial Optimization – Computer Project 1

The aim of this project is to solve the polynomial optimization problem

$$
\begin{aligned}
(P)\qquad \text{minimize } & 2x_1^4 + 10x_1^3 + x_1^2x_2 + 19x_1^2 - x_1x_2^2 + 4x_1x_2 + \\
& 14x_1 + 2x_2^4 - 10x_2^3 + 19x_2^2 - 14x_2 + 11 \\
\text{over } & x_1, x_2 \in \mathbb{R} \\
\text{subject to } & x_1^2 + 3x_1 - x_2^2 + x_2 + 3 \geq 0 \\
& x_1^2 + 2x_1 - x_2^2 + 2x_2 + 1 \geq 0 \\
& x_1^3 + 3x_1^2 + 2x_1 - x_2^3 + 3x_2^2 - 2x_2 \geq 0
\end{aligned}
$$

using semidefinite programming. You will need a working version of the commercial system MATLAB[1] including the Symbolic Math Toolbox MuPAD[2] (e.g., the student version[3] or the version installed in the PhyMa-Pool[4]). You need to have installed[5] the following MATLAB-packages (cf. Problem Set 1): the modelling language YALMIP[6] and at least one of the SDP-solvers SeDuMi[7] and SDPT3[8].

In this project, you have to construct four files where `narendra`[9] must be replaced by your given name in lowercase letters:

(1) a MuPAD notebook `pop1narendra.mn`

(2) a MuPAD program `pop1narendra.mu`

(3) a MATLAB script `pop1narendra_constraints.m`

(4) a MATLAB script `pop1narendra.m`

You will first create (1) following our instructions. This will be your potentially first MuPAD notebook where you can take your first steps with the very nice and elegant MuPAD language. In the end (1) should be a readable MuPAD notebook that creates

[1] `http://en.wikipedia.org/wiki/MATLAB`
[2] `http://en.wikipedia.org/wiki/MuPAD`
[3] `http://www.mathworks.com/academia/student_version/`
[4] `http://www.math.uni-konstanz.de/fb_seiten/contrib/studium/phyma/phyma_en.php?menu_var=Studies`
[5] and added to the MATLAB path
[6] `http://users.isy.liu.se/johanl/yalmip/`
[7] `http://sedumi.ie.lehigh.edu/`
[8] `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`
[9] `http://en.wikipedia.org/wiki/Karmarkar%27s_algorithm`

the constraints of a semidefinite program (P_1) relaxing (P) and writes them in YALMIP format to the file (3). In other words, file (3) will be created automatically once your MuPAD notebook works correctly and is evaluated. MuPAD notebooks can only be opened but cannot be called from MATLAB scripts. What we finally need is therefore a MuPAD program. So you will create file (2) by just copying all essential commands from (1) into (2). The "master" file (3) will therefore first execute (2) which produces a semidefinite relaxation (P_1) of (P) and writes it into (2), then (3) will execute (2) to define the constraints in the YALMIP format, and finally (3) will tell YALMIP to solve (P_1) using an SDP-solver.

(a) Start MATLAB and add YALMIP and at least one of SeDuMi and SDPT3 to the MATLAB path.

(b) Set the working directory ("Current Folder") to the folder where you want to store your project in.

(c) Create a blank MuPAD notebook by typing `mupad` in the MATLAB command window, put a comment with your name in the first line of this notebook. Save the notebook in your working directory under the name of `pop1narendra.mn`.

(d) Open MuPAD's help window and work through the topics

Getting Started → First steps in MuPAD
Getting Started → Accessing MuPAD Help
Getting Started → Working with Data Structures → Sequences
Getting Started → Working with Data Structures → Lists
Getting Started → Working with Data Structures → Vectors and Matrices

while experimenting in the notebook.

(e) Now search in the MuPAD help browser for the commands `:=`, =, [], \$, `->`, `_plus`, `map`, `expand`, `monomials`, `lterm`, `op`, `nops`, `matrix`, `transpose`, `max`, `degree`, `contains`, `subs` and whatever other command you are interested in.

(f) Define the goal function `f` of (P) and the three polynomials `p[1]`, `p[2]` and `p[3]` defining the constraints of (P).

(g) Define a function `mon` (using `->`) that yields a sequence of all monomials of degree exactly k when called with a nonnegative integer k.

(h) Define a function `vec` yielding a column vector of all monomials of degree at most k when called with a nonnegative integer k.

(i) Define symmetric matrix polynomials $P_0 \in S\mathbb{R}[X_1, X_2]^{6\times 6}$, $P_1 \in S\mathbb{R}[X_1, X_2]^{3\times 3}$, $P_2 \in S\mathbb{R}[X_1, X_2]^{3\times 3}$ and $P_3 \in S\mathbb{R}[X_1, X_2]^{1\times 1}$ whose positive semidefiniteness expresses the validity of the (mostly redundant) constraints

$$
\begin{aligned}
(a_1 + a_2x_1 + a_3x_2 + a_4x_1^2 + a_5x_1x_2 + a_6x_2^2)^2 &\geq 0 \\
(a_1 + a_2x_1 + a_3x_2)^2(x_1^2 + 3x_1 - x_2^2 + x_2 + 3) &\geq 0 \\
(a_1 + a_2x_1 + a_3x_2)^2(x_1^2 + 2x_1 - x_2^2 + 2x_2 + 1) &\geq 0 \\
a_1^2(x_1^3 + 3x_1^2 + 2x_1 - x_2^3 + 3x_2^2 - 2x_2) &\geq 0,
\end{aligned}
$$

($a_1, \dots, a_6 \in \mathbb{R}$) confer the lecture and Exercise Sheet 1.

(j) Define the list of monomials from $\mathbb{R}[X_1, X_2]$ appearing somewhere now and another list of the same length s of variables $y_i =$ `y[i]` ($i \in \{1, \dots, s\}$).

(k) Define a function that *linearizes* polynomials (cf. Exercise Sheet 1 and the lecture) from $\mathbb{R}[X_1, X_2]$ in the sense that it replaces the i-th monomial in our list by the corresponding y_i (except for the constant monomial 1). Use the `map` function to do this.

(l) Again using the `map` function extend this linearization to matrix polynomials.

(m) Store the linearized goal function f in `l` and the linearized matrix polynomials $P_0, \dots, P_3$ in `M0`,...,`M3`.

(n) Add and understand the following lines to your notebook:

```
use(generate,MATLAB) // load MATLAB function form library generate
MATLAB(l)
MATLAB(M0)
yalmipspec:="y=sdpvar(".nops(monolist).",1); ":
yalmipspec:=yalmipspec.MATLAB(l).MATLAB(M0).MATLAB(M1).MATLAB(M2).MATLAB(M3):
yalmipspec:=stringlib::subs(yalmipspec,"t0"="l"):
yalmipspec:=stringlib::subs(yalmipspec,"zeros"="sdpvar"):
fprint(Unquoted,Text,"pop1narendra_constraints.m", yalmipspec)
```

(o) Create (2) with your favorite editor (or with the MuPAD or the MATLAB editor). Copy and paste everything from (1) which is really needed to fulfill the task of producing (3). Do not copy the things which are nice in a notebook to understand what is going on (like examples). This helps us to see if you have understood what is really necessary. Be careful that you have to slightly adapt your syntax to a program file, e.g. you always have to separate two commands by a semicolon.

(p) Create (4) with your favorite editor and write the following into this file while trying to understand it using the MATLAB help and the YALMIP wiki.

```
% Narendra Karmarkar
read(symengine,'pop1narendra.mu')
pop1narendra_constraints
```

```
constraints=set(M0>=0)+set(M1>=0)+set(M2>=0)+set(M3>=0)
solvesdp(constraints,l)
double(l)
double(y(2))
double(y(3))
evalin(symengine,'subs(f,x[1]=-1,x[2]=1)')
```

(q) Type `pop1narendra` in the MATLAB command window! Why can you conclude that the SDP-relaxation (P_1) has solved (at least up to numerical errors) the POP (P)?

Due by Monday, May 6th, 2012, 5:00 am. The four files (1), (2), (3) and (4) must be sent attached to an electronic mail to `leonid.chatschijan@uni-konstanz.de` where `leonid.chatschijan` must be replaced by your tutor `sebastian.gruler` or `maria.lopez-quijorna`. Files (1), (2) and (4) must be executable without producing errors. Note that this must work in any directory so please avoid using pathnames when specifying filenames. File (3) must be identical with both the file that is produced by executing (1) and (2), respectively. It is perfectly allowed to collaborate with other students. However, the finalization, annotation and submission of the project has to be done by each participant individually. The only file which must be well documented is (1). The other files need not necessarily contain comments. Comments should be short but pregnant and in English language. In files (1), (2) and (4) you must specify your name by a comment which is of the form `\\ Narandra Karmarkar` and `% Narandra Karmarkar`, respectively.

Where it is little effort, you should keep the code sufficiently general so that it can be easily adapted for future projects, e.g., you should use `n` instead of 2 by including a line like

```
n:=2 // number of variables
```

in (1) to allow for later increase of the number of variables. Also you are encouraged to put redundant "pedagogical" code which motivates or explains subsequent more sophisticated code even if it is not necessary for carrying out the project. For example, you could add the line

```
x[i]$i=1..n // the variables
```

which is just there to increase the readibility.

Your project submissions will usually not be discussed during the exercise groups but individually in the offices of the lecturer and the tutors. You have to be able to explain your code.