Introduction to Parsers and Numerical Solvers

4DM20 Engineering Optimization

Contents

1 Parsers **1**
- 1.1 YALMIP 1
- 1.2 cvxpy 2

2 Numerical Solvers **2**
- 2.1 How to install Gurobi 3
- 2.2 How to install MOSEK 4

3 How to use YALMIP **5**

4 How to use cvxpy **6**

1 Parsers

Parsers help us users translate optimization problems to the canonical form needed by solvers. Solvers usually required their own syntax and a specific problem formulation, as you have noticed in last week's exercise session, so parsers are commonly used by practitioners to make their life easier. In this course, depending on your preferred programming language, we will be using one of two parsers; **YALMIP** for MATLAB and **cvxpy** for python. Below you find instructions on how to install them.

1.1 YALMIP

YALMIP (Yet Another LMI Parser) is a free MATLAB toolbox for rapid prototyping of optimization problems. Although it is free of charge to use and openly distributed, note that its copyright is owned by Johan Löfberg, Linköping University, and it must be referenced [1] when used in your research endeavors.
YALMIP is a parser and is not shipped with any low-level solvers you use. They should be installed as described in Section 2 and in the solver manuals. Make sure to add the required paths in the environment variables window. Your MATLAB installation might already have solvers available that YALMIP can interface to, but make sure you understand which solvers you are using. The solvers supported by YALMIP can be found at `https://yalmip.github.io/allsolvers/`

Installation

YALMIP is very easy to install, and it is entirely based on MATLAB code. Navigate to `https://yalmip.github.io/` and click on the "Download" tab. Scroll down to "Last release" and click

on the green box labeled "Latest official release". The download of a zip folder named "YALMIP-master" should begin. Once the download is completed, extract the contents of the compressed folder and open the README.txt file. This file contains all the steps to install YALMIP on your computer. If you have any trouble, open the YALMIP main page and click on the second tab "Tutorials". You will find a vast amount of material covering every aspect of the tool. Click on "Installation" (`https://yalmip.github.io/tutorial/installation/`) and check the steps if you have any problems with the installation.

1.2 cvxpy

cvxpy is a python-based parser for convex optimization problems. Just like YALMIP, it should be referenced when used in your research projects [2]. cvxpy comes with some pre-installed open source solvers, but you can install more as described in Section 2.

Installation

Installation instructions can be found on `https://www.cvxpy.org/install/`.

2 Numerical Solvers

As you have learned during lectures, optimization problems can be divided into several categories. Sometimes one category can be regarded as a special case of a broader set. Linear programming can be solved by quadratic programming which can be solved by second-order cone programming which can be solved by semi-definite programming. Hence, in theory, you only need a semi-definite programming solver for all the problems above. In practice, dedicated solvers are recommended.

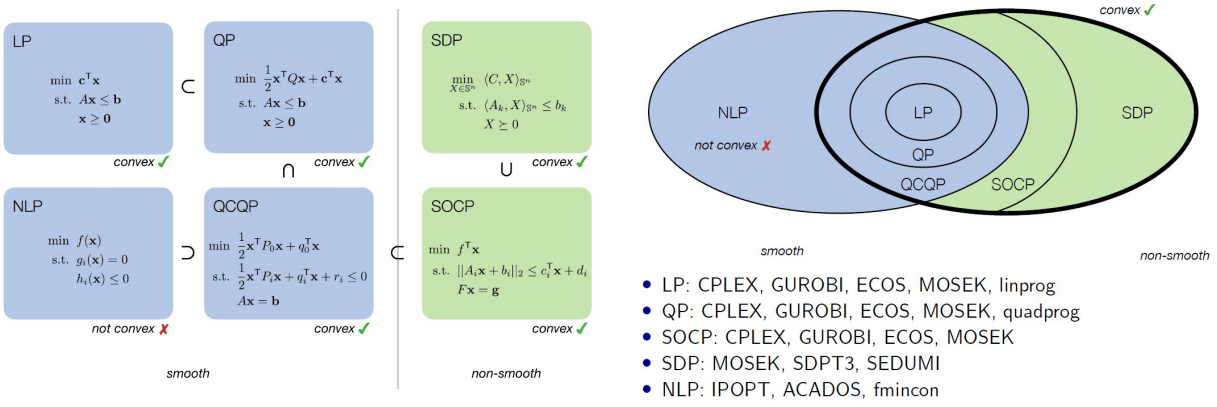


Figure 1: Optimization Programs and Numerical Solvers [3].

Solver	LP	MILP	QP	QCQP	MIQP	SOCP	MISOCP	SDP	NLP
MOSEK	✓	✓	✓	✓	✓	✓	✓	✓	
GUROBI	✓	✓	✓	✓	✓	✓	✓		
ECOS	✓	✓	✓	✓	✓	✓	✓		
MATLAB	linprog	intlinprog	quadprog						fmincon

Table 1: Comparison of MOSEK, GUROBI, ECOS and the corresponding built-in solver in MATLAB.

MATLAB offers some default solvers in its Optimization Toolbox, but for solving large scale complex optimization problems they often fall short. Luckily, as academics we have free access to some very powerful commercial solvers that have been developed to work efficiently and are improved constantly. These solvers are used by large international companies to route large aircraft fleets, find optimal schedules for sports leagues, design high-performance machinery, optimize financial portfolios and much more. They employ a mix of very sophisticated solving techniques and leverage a rich background of optimization theory.

We recommend installing at least two different solvers. Sometimes a solver could have trouble solving your problem and especially during debugging it is useful to check the feasibility with more than one numerical solver.
Below we outline the steps for the installation of Gurobi and MOSEK on Windows, for other operating systems you can consult the documentation of the respective solvers, which we recommend in any case to avoid problems.

2.1 How to install Gurobi

Gurobi is a powerful general purpose commercial solver that has become somewhat of an industry standard. It offers a free academic license for which an account is required. Navigate to `https://www.gurobi.com/features/academic-named-user-license/` and follow the instructions to create a free account as an academic user. You will have to request a personalized license key as "Named-User Academic", which will generate a snippet with the structure `grbgetkey <key>`. Copy said code and paste it into your computer's command window, which will trigger a prompt for where to store the key and suggest a default location, which we recommend using.

To install Gurobi, navigate to `https://www.gurobi.com/downloads/gurobi-software/` and select the appropriate installer for your operation system. Once downloaded, run the installer and select a directory of your choice. For simplicity we recommend to use the default directory suggested by the installer.

In MATLAB

Once the key has been stored, you can open MATLAB and type in the command window

```
cd c:\gurobi\win64\matlab
```

or wherever else you have saved Gurobi. MATLAB will open your Gurobi path and you can type

```
gurobi_setup
```

which will install Gurobi's MATLAB interface. If you want to keep using gurobi regularly, we recommend typing

```
savepath
```

to save the path definition.
Gurobi is now ready to be used in MATLAB. To get started we recommend reading Section 3.

In Python

Type

```
pip install gurobipy
```

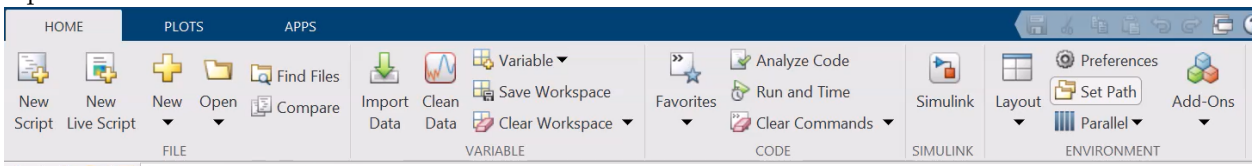
in your currently active python environment (or use the equivalent syntax in your package manager of choice). If your license file stored in the default location as noted above, you should be good to go.

2.2 How to install MOSEK

Navigate to `https://www.mosek.com/downloads/` and select the appropriate installer for your operation system, for most users this will be the **64 bit x86** version. Once downloaded, run the installer and select a directory of your choice. For simplicity we recommend to use the default directory suggested by the installer.
MOSEK offers a free academic license, which you can request at `https://www.mosek.com/products/academic-licenses/` by selecting "Request Personal Academic License". It is recommended to use your `tue.nl` student email in the application form. Upon requesting the license you will quickly receive an email with a `.lic` file and instructions on what to do with it. As before, we highly recommend following those instructions and place the license inside a folder called `mosek` in the appropriate directory.

In MATLAB

Open MATLAB and select `Set Path` in the `HOME` tab.



Select `Add with Subfolders...` and navigate to the directory where you installed MOSEK (usually `C:\ProgramFiles\Mosek`). Here, select the folder named like the version you installed and hit `Save` before you close the `Set Path` window.
Type in the command window

```
mosekdiag
```

which should result in a message displaying your MATLAB version, system architecture, mosekopt path, and other information, but most importantly

```
Test linear solve:  Success
mosekopt works OK. You can use MOSEK in MATLAB.
```

In that case, MOSEK is now ready to be used in MATLAB. To get started we recommend reading Section 3.

In Python

Type

```
pip install mosek
```

in your currently active python environment (or use the equivalent syntax in your package manager of choice). To test if your installation worked properly, you can try

```
import mosek
```

3 How to use YALMIP

Every time you call sdpvar etc., an internal database grows larger. It is good practice to start by clearing YALMIP's internal database using

```
yalmip('clear')
```

The most important command in YALMIP is sdpvar. This command is used to define continuous optimization variables. To define a matrix P with n rows and m columns, we write

```
P = sdpvar(n,m);
```

Note that a square matrix is defined as symmetric by default. To obtain a fully parameterized (i.e., not necessarily symmetric) square matrix, we need to add the argument `'full'`.
To define a collection of constraints, we simply create and concatenate them.

```
constraints = [sum(sum(P)) >= 3];
for i = 1 : 3
     constraints = [constraints, P(i,i) + P(i+1,i+1) <= P(i+2,i+2)];
end
```

Strict inequalities cannot be used (YALMIP will warn you) since they have no meaning when doing optimization with numerical solvers, as these solvers always work with numerical tolerances anyway. Hence, if you need the upper bound to be strict, you have to select a margin. Note that the selection of this margin is a tricky issue. If too small, it will make no difference as it will drown in the general tolerances solvers use, if too large it might reduce your feasible space too much. After defining variables and constraints, we define the objective function and we solve the problem using a numerical solver of our choice. We can give YALMIP many indications in the "options" variable. It is always important to read the text output, as it could show interesting insights into the problem (e.g., when debugging!).

```
objective = norm(P,1);
options = sdpsettings('verbose',1,'solver','quadprog','quadprog.maxiter',100);
sol = optimize(constraints,objective,options);

if sol.problem == 0
solution = value(P)
else
display('Hmm, something went wrong!');
sol.info
yalmiperror(sol.problem)
end
```

For more information go to: `https://yalmip.github.io/tutorial/basics`.

4 How to use cvxpy

Start your code with

```
import cvxpy as cp
```

You define decision variables with

```
x = cp.Variable(n)
```

where `n` is the size of your vector. Define the objective function as

`objective = cp.Minimize(`*your function here*`)`

and add constraints as follows:

```
constraints = [0 <= x, x <= 1]
```

Note, that it makes little sense to define strict inequality constraints in numerical optimization as mentioned in the YALMIP section above. If we need an inequality to be strict, we can add a small margin to the inequality.
Finally, we need to define our problem as a problem instance, this is done with

```
prob = cp.Problem(objective,constraints)
```

To return the solution, we invoke

```
result = prob.solve()
print(x.value)
```

More information and many examples can be found under `cvxpy.org`.

References

[1] J. Löfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB", 2004.

[2] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization", *Journal of Machine Learning Research*, 2016, To appear. [Online]. Available: `https://stanford.edu/~boyd/papers/pdf/cvxpy_paper.pdf`.

[3] M. Salazar, *Advanced electric and hybrid powertrain design, 4at030, lecture 9: Optimization*, 2021.