Solving Mixed-Integer Linear Programs with MATLAB

Bowen Hua

Department of Electrical and Computer Engineering

The University of Texas at Austin

November 2018

Outline

- Install MATLAB and YALMIP
- Example problem
- Example unit commitment problem

Outline

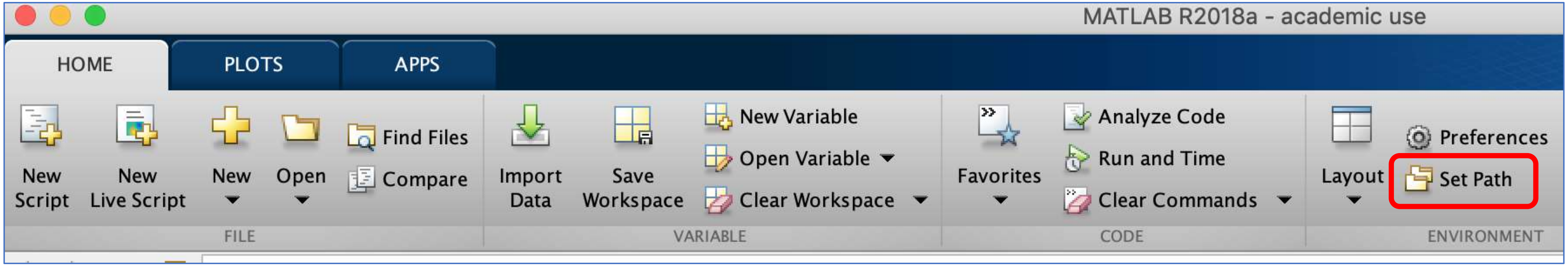
- Install MATLAB and YALMIP
- Example problem
- Example unit commitment problem

Install MATLAB and YALMIP

- Cockrell School provides licenses for MATLAB.
    - http://www.engr.utexas.edu/itg/products/8017-matlab
    - Remember to install the optimization toolbox.
- Download YALMIP and install it.
    - https://yalmip.github.io/download/
    - https://yalmip.github.io/tutorial/installation/
    - Unzip the downloaded file into a folder `~/YALMIP-master`.
    - Add the folder with all its subfolders to your MATLAB path
        - See next slide for detailed instructions

Add YALMIP to MATLAB Path

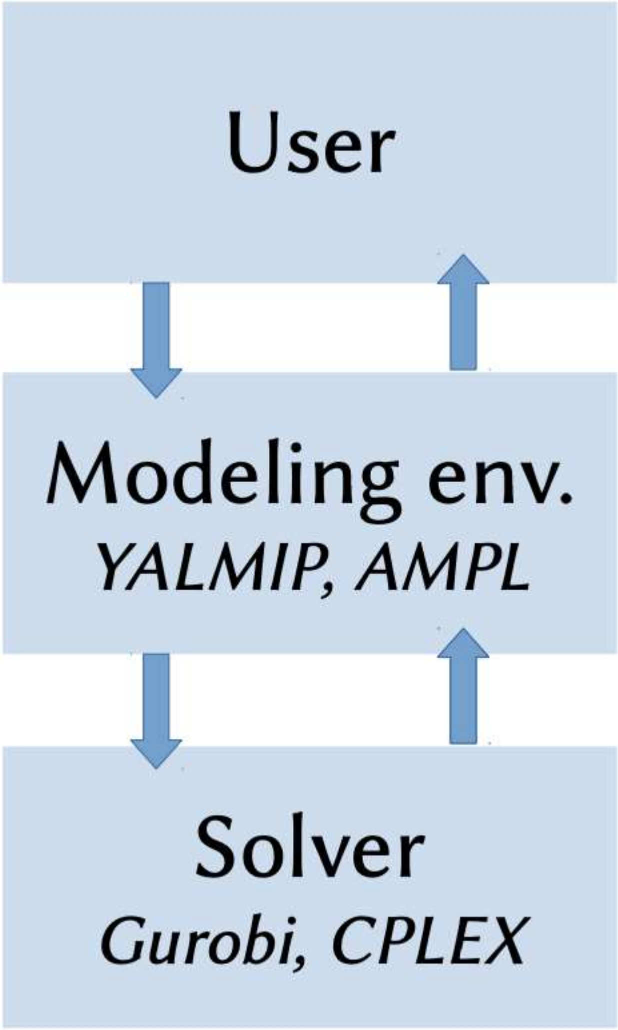
- Click “Set Path” on the MATLAB toolbar



- Click “Add with Subfolders”
- Add the above-mentioned folder
- Run `yalmiptest` in MATLAB to test the installation

What is YALMIP?

- YALMIP is a modeling environment for optimization problems.
- It allows a user to describe an optimization problem by writing algebraic equations.
- It then translate the optimization problem into a form that is recognizable by a solver.
- The solver then finds the solution to the problem.



Outline

- Install MATLAB and YALMIP
- Example problem
- Example unit commitment problem

Describe the problem with YALMIP

- Declare variables
- Define constraints
- Define the objective function
- Solve

Example Problem

- Problem (4.45) in Section 4.8.3 on page 143 of Section 4.
- Mathematical formulation of the problem:

$$\min_{z\in\mathbb{Z},x\in\mathbb{R}} \{4z+x \mid -x=-3, 0\leq z\leq 1, 2z\leq x\leq 4z\}.$$

Declare variables

- Code:

```
z = binvar(1,1);
x = sdpvar(1,1);
```

- `binvar(1,1)` defines a binary variable.
- `sdpvar(1,1)` defines a continuous variable.

Define constraints

- Put all constraints in a list:

```
constr = [-x == -3];
constr = [constr, 2*z <= x <= 4 * z];
```

- Double-sided inequality constraints are supported.

Define objective function

```
Objective = 4*z + x;
```

Solve

```
options = sdpsettings('verbose',1,'solver','INTLINPROG');
sol = optimize(constr,Objective,options);
```

- We use the built-in mixed-integer linear program solve of MATLAB, `intlinprog`.
- To see the optimal objective function value, we can use:
  - `value(Objective)`
- To see the optimal value of the decision variables, we can use:
  - `value(x)`
  - `value(z)`

Outline

- Install MATLAB and YALMIP
- Example problem
- Example unit commitment problem

Example unit commitment problem

- Unit Commitment Example in Section 10.8.1 on page 126 of Section 10.
- Mathematical formulation of the problem:

$$
\begin{aligned}
\min_{u,z,P} \quad & \sum_{t=1}^{2} 1000(u_{1t} + u_{2t}) + 25P_{1t} + 35P_{2t} \\
\text{s.t.} \quad & 0 \leq P_{1t} \leq 100z_{1t}, \quad \forall t \\
& 0 \leq P_{2t} \leq 50z_{2t}, \quad \forall t \\
& u_{11} = z_{11} - 1 \\
& u_{12} = z_{21} - z_{11} \\
& u_{21} = z_{21} \\
& u_{22} = z_{22} - z_{21} \\
& P_{11} + P_{21} = 110 \\
& P_{12} + P_{22} = 125 \\
& z_{1t} \in \{0,1\}, \quad \forall t \\
& z_{2t} \in \{0,1\}, \quad \forall t \\
& u_{1t} \in \{0,1\}, \quad \forall t \\
& u_{2t} \in \{0,1\}, \quad \forall t
\end{aligned}
$$

Declare variables

- Code:

```
z1 = binvar(2,1);
z2 = binvar(2,1);
u1 = binvar(2,1);
u2 = binvar(2,1);
P1 = sdpvar(2,1);
P2 = sdpvar(2,1);
```

- `binvar(2,1)` defines a 2-column-vector of binary variables.
- `sdpvar(2,1)` defines a 2-column-vector of continuous variables.
- `z1, u1, P1` are variables for generator 1.

Define constraints

- Bounds for power outputs. These constraints are defined in vector form:

```
private_constr = [0 <= P1 <= 100 * z1];
private_constr = [private_constr, 0 <= P2 <= 50 * z1];
```

- Logical constraints between startup and on/off variables:

```
private_constr = [private_constr, u1(1) == z1(1) - 1];
private_constr = [private_constr, u1(2) == z1(2) - z1(1)];
private_constr = [private_constr, u2(1) == z2(1)];
private_constr = [private_constr, u2(2) == z2(2) - z2(1)];
```

- Power balance constraints:

```
power_balance = [P1(1) + P2(1) == 110];
power_balance = [power_balance, P1(2) + P2(2) == 125];
```

Define objective function

```
Objective = 1000 * (sum(u1) + sum(u2)) + 25 * sum(P1) + 35 * sum(P2);
```

Solve

```
options = sdpsettings('verbose',1,'solver','INTLINPROG');
sol = optimize([private_constr, power_balance],Objective,options);
```

- To see the optimal objective function value, we can use:
  - `value(Objective)`
- To see the optimal value of the decision variables, we can use:
  - `value(P1)`