Introduction into CVXPY

Roman Schutski

Skoltech

November 16, 2018

Outline

What is CVXPY?

- CVXPY is a modeling system for complex optimization problems.
- Capabilities: LP, QP, SOCP, SDP etc.
- Website: https://www.cvxpy.org/tutorial

Installation

- Linux
  - with Anaconda

    ```
    $ conda install -c conda-forge lapack
    $ conda install -c cvxgrp cvxpy
    ```

  - with pip

    ```
    $ pip install cvxpy
    ```

- Windows
  - Install latest Anaconda
  - Install Visual Studio C++ compiler for Python.
  - install from Anaconda prompt as above

```
$ pip install cvxpy
```

Convex optimization problem

$$
\begin{aligned}
&\text{minimize } f_0(x) \\
&\text{subject to } f_i \leq 0, \quad i = 1 \dots m \\
&Ax = b \\
&x \in R^n
\end{aligned}
\tag{1}
$$

- Objective and equality constraints f_i are convex
- Constraints are linear

Solving a convex problem

- Use someone elseś "standard" solver (LP, QP, SOCP..)
  - Easy, but the problem **must** be in a "standard" form
- Use a convex modeling language
  - transforms user-friendly format to solver-friendly format
  - check/verify problem convexity
  - extends the range of problems that can be solved with the solver

Disciplined convex programming (DCP)

- System for construction of expressions with known convexity
  - constant, affine, non-negative (convex), non-positive (concave), etc
- expressions formed from
  - variables
  - constants
  - library of atoms with known curvature and sign
- more on DCP at dcp.stanford.edu

CVXPY language

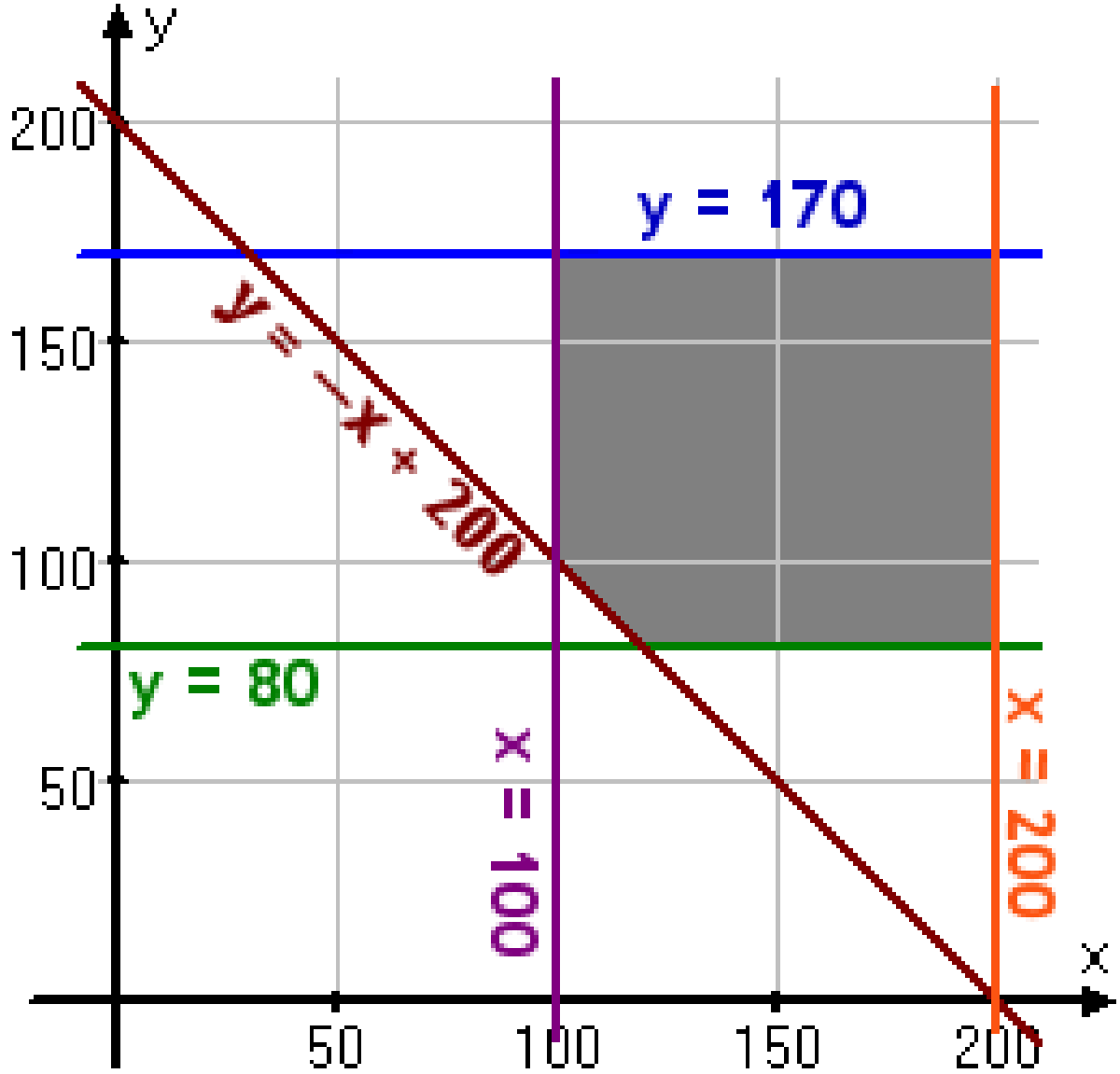
```
import cvxpy as cvx
# Create two scalar optimization variables.
x = cvx.Variable()
y = cvx.Variable()
# Create two constraints.
constraints = [x + y == 1,
               x - y >= 1]
# Form objective.
obj = cvx.Minimize(cvx.square(x - y))
# Form and solve problem.
prob = cvx.Problem(obj, constraints)
prob.solve()  # Returns the optimal value.
print("status:", prob.status)
print("optimal value", prob.value)
print("optimal var", x.value, y.value)
```

Example for the audience

$$
\begin{aligned}
&\text{minimize } -2x + 5y \\
&\text{subject to} \\
&100 < x < 200 \\
&80 < y < 170 \\
&y > -x + 200
\end{aligned}
\tag{2}
$$

- What are x and y?
- What is the maximal value of $x + y$ such that the problem is still feasible?

hint



CVXPY solvers

	LP	QP	SOCP	SDP	EXP	MIP
CBC	X					X
GLPK	X					
GLPK_MI	X					X
OSQP	X	X				
CPLEX	X	X	X			X
Elemental	X	X	X			
ECOS	X	X	X		X	
ECOS_BB	X	X	X		X	X
GUROBI	X	X	X			X
MOSEK	X	X	X	X		
CVXOPT	X	X	X	X	X	
SCS	X	X	X	X	X	

SOCP - second order cone programming, EXP - exponential cone constraints, MIP - mixed integer programming.

- Each solver has its own interface and "standard" form. We consider ECOS solver.

Arguments of the ECOS solver

- Internal "standard form" of the ECOS solver

$$\begin{aligned} &\text{minimize} \quad c^T x \quad \text{subject to} \\ &G \cdot x \leq h \\ &A \cdot x = b \\ &x_i \geq 0 \end{aligned} \tag{3}$$

```
# Setup the problem
x = cvx.Variable(2)
obj = cvx.Minimize(x[0] + cvx.norm(x, 1))
constraints = [x >= 2]
prob = cvx.Problem(obj, constraints)
# Get ECOS arguments.
data = prob.get_problem_data(cvx.ECOS)
print('Raw arguments of the solver:')
print(data[0].keys(), '\n')
```

```
$ Raw arguments of the solver:
$ dict_keys(['c', 'offset', 'dims', 'A', 'b', 'G', 'h'])
```

Arguments of the ECOS solver

- It is possible to extract standard parameters from any LP formulation

```
# Get ECOS arguments.
data = prob.get_problem_data(cvx.ECOS)
print('Raw arguments of the solver:')
print(data[0].keys(), '\n')
```

```
$ Raw arguments of the solver:
$ dict_keys(['c', 'offset', 'dims', 'A', 'b', 'G', 'h'])
```

- It is possible to extract duals for each constraint

```
# Form and solve problem.
prob = cvx.Problem(obj, constraints)
prob.solve()
# Extract dual values
print("optimal dual for constraint 0",
    constraints[0].dual_value)
```

Parametric problems

(LASSO regularization)

$$\begin{aligned} &\text{minimize } |X \cdot \beta - Y|_2^2 + \lambda \cdot |\beta|_1 \\ &\beta \in R^n \end{aligned} \tag{4}$$

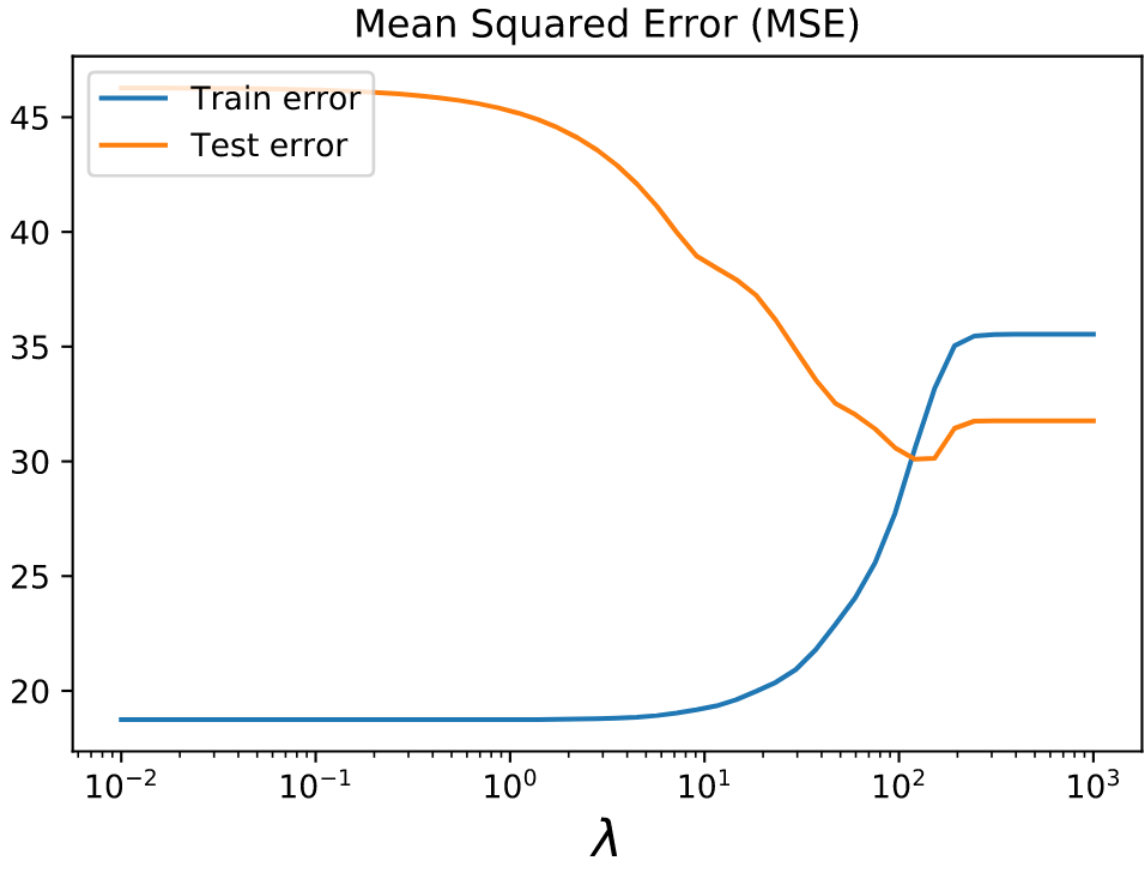
```
import cvxpy as  cvx
beta = cvx.Variable(n)
lambd = cvx.Parameter(nonneg=True)
error = cvx.norm2(cvx.matmul(X, beta) - Y)**2
regularizer = cvx.norm1(beta)
problem = cvx.Problem(cvx.Minimize(error+regularizer))
```

For-loop style train/test curve

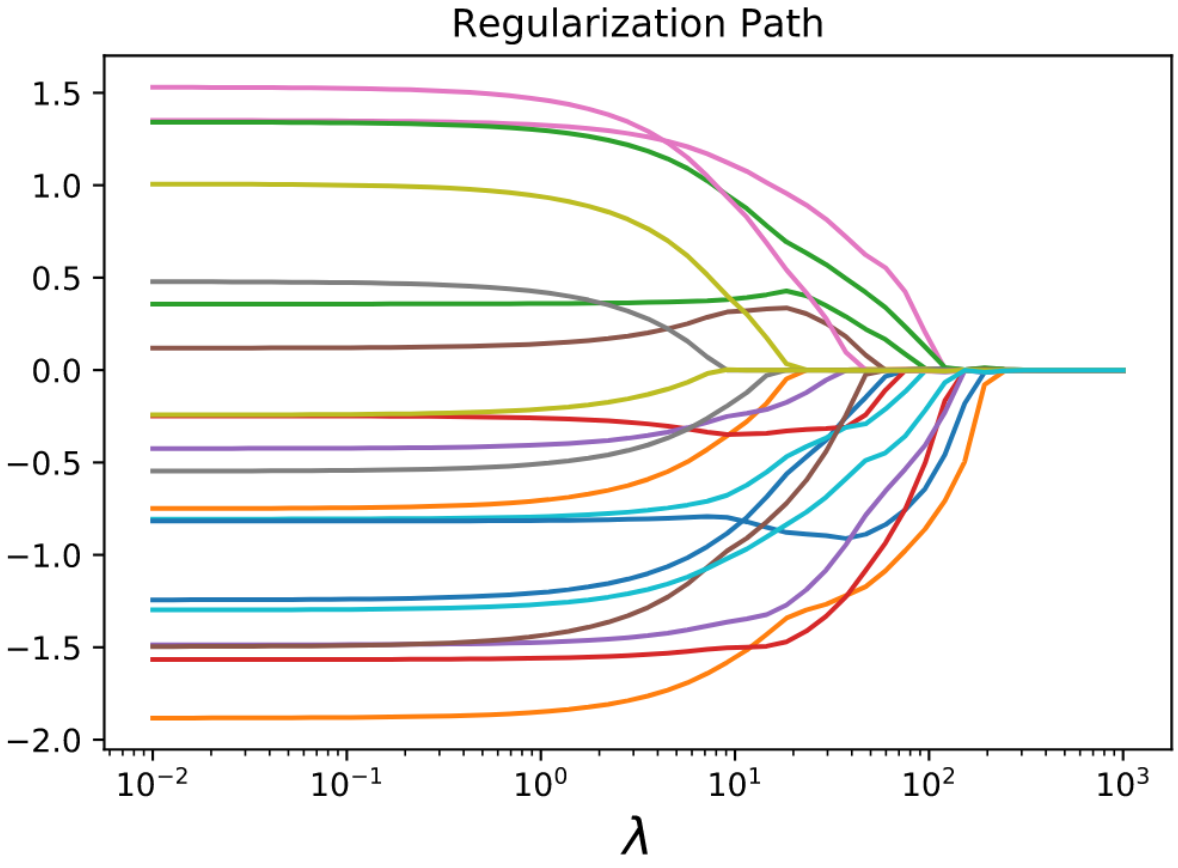
- We can compute the train/test accuracy by updating the value of the parameter

```
lambd_values = np.logspace(-2, 3, 50)
for v in lambd_values:
    lambd.value = v
    problem.solve()
    beta_values.append(beta.value)
```

Train/test curve



Sparsity of the solution/feature selection



Standard Python parallelism is supported by CVXPY

```
# Assign a value to lambda and find an optimal beta
def get_beta(lambda_value):
    lambd.value = lambda_value
    problem.solve()
    return beta.value

# Use a standard futures framework for
# parallel execution
import concurrent.futures
beta_values = []
with concurrent.futures.ProcessPoolExecutor(
            max_workers=nproc) as executor:
     for beta_value in executor.map(
            get_beta, np.logspace(-2, 3, 50))
        beta_values.append(beta_value)
```

Summary

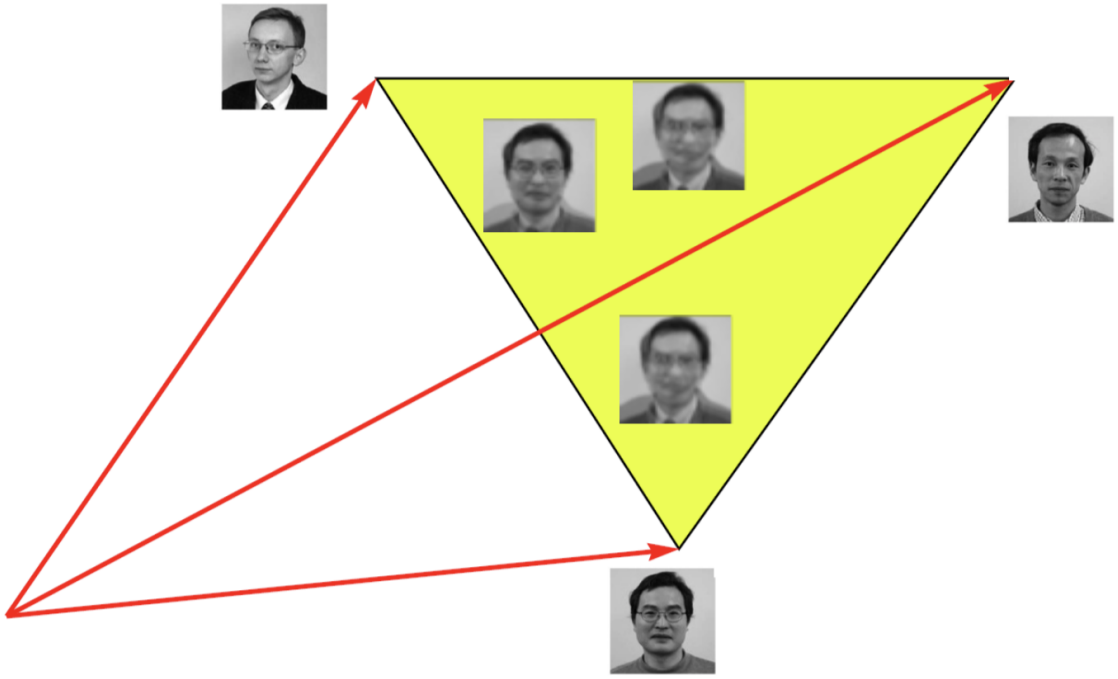
- Convex optimization is easy with CVXPY
- Embedded language mixes well with standard Python
  - Parallelism
  - Object oriented design of the problems (not covered)

Task for the brave!

- Convex Analysis of Mixture of Non-Negative Sources (CAMNS)

$$X_i = \sum_{j}^{M} A_{ij} \cdot S_j \tag{5}$$

$$j = 1 \ldots M, \; i = 1 \ldots N, \; N < M$$



Task for the brave!

- Task: Port to Python the solution to the CAMNS problem
- Source code: https://github.com/qbit-/CAMNS_program
- You need to: complete the port, make a runnable notebook and send it by email to r.schutski@skoltech.ru
- First 5 solutions will get +5% to the final grade.