Convex optimisation in communications with cvxpy

Robert P. Gowers,[1] Sami C. Al-Izzi,[1] Timothy M. Pollington,[1] Roger J. W. Hill,[1] and Keith Briggs[2]
[1]*Department of Mathematics, University of Warwick*
[2]*BT, Adastral Park*

"Nothing takes place in the world whose meaning is not that of some maximum or minimum."

LEONARD EULER (1707-1783)

INTRODUCTION

Convexity is an important property in optimisation. This is because if a problem is convex then the task of finding a global minimum is reduced to that of finding a local minimum. The importance of finding these minima efficiently in science and engineering has driven the development of software packages, such as `cvxpy`.

Here we show that many interesting problems in communications not only have a convex formulation (as shown in [1]), but can be easily implemented computationally using the `cvxpy` programming language. The ease of implementation of such problems makes `cvxpy` an invaluable tool for both students and researchers in many areas of science and engineering.

Convex Sets and Functions

If a set C is *convex*, then for any two points $x, y \in C$, any point z along the x, y line must also $\in C$ [1, p.23]. More formally, for $x, y \in C$ and $\theta \in [0, 1]$:

$$\theta x + (1 - \theta)y \in C. \tag{1}$$

In terms of a function f, where $f : \mathbb{R}^n \to \mathbb{R}$. For f to be *convex* then the domain of f, $\mathbf{dom} f$, must be a convex set and $\forall x, y \in \mathbf{dom} f$ [1, p.67]:

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y). \tag{2}$$

This means that the function is less than or equal to linear. This can be seen graphically in Fig 1. A concave function can be made convex by the operation $f \to -f$.

Convex optimisation

A *convex optimisation problem* has three components:

- a convex *objective function* $f_0(x)$,
- m convex *inequality constraint functions* $f_i(x)$,
- k convex *equality constraint functions* $g_j(x)$,

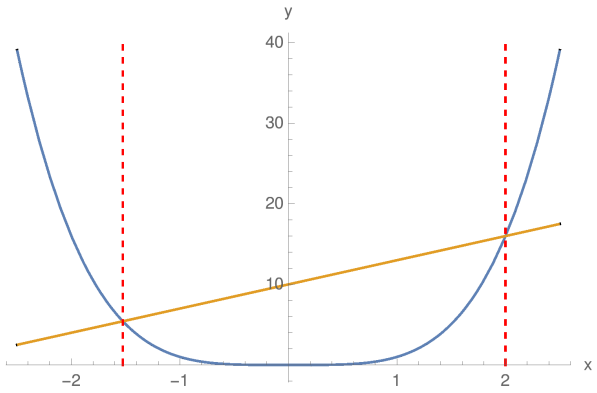

Figure 1. A chord showing convexity of the function x^4.

where $f, g : \mathbb{R}^n \to \mathbb{R}$ [1, p.141]. We seek an optimum $x^* \in \mathbb{R}^n$, where $f_0(x^*)$ is a minimum. Formally the problem is defined as:

$$\begin{array}{lll} \text{minimise} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i \in \{1, .., m\} \\ & g_j(x) = 0, & j \in \{1, ..., k\}. \end{array} \tag{3}$$

Any local optimum x^* for a convex optimisation problem is also a global optimum [1, pp.138-139]. An optimum of a convex optimisation problem may not be unique: consider the convex function $f(x) = \text{const}$, which has line of optima x^*.

Disciplined convex programming

`cvxpy` is a symbolic programming language for *Python* developed by Diamond and Boyd at Stanford[3]. It uses a set of rules, called *disciplined convex programming* (DCP), to determine whether a function is convex. If `cvxpy` finds that the function is convex, then interior point methods can be safely applied, which are guaranteed to find the optimal solution.

To implement DCP, the `cvxpy` package has a set of predefined functions in classes containing their curvature and sign. By only allowing certain operations with these known functions, general composition theorems from convex analysis can be applied to determine the convexity of the more complex expression. See Fig. 2.

The set of available DCP functions do not define the complete set of convex functions. However, as we will show later, a large class of interesting convex problems can be written in DCP format.

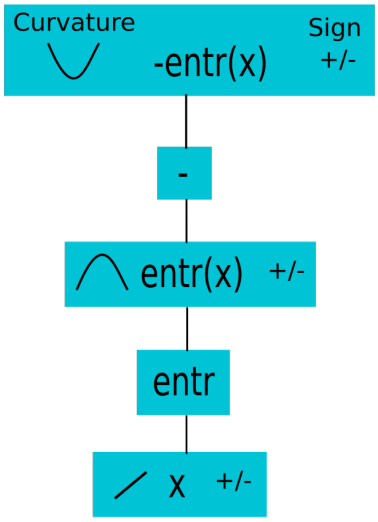

Figure 2. Application of DCP rules to a set of `cvxpy` functions, where $\text{entr}(x) = -x\log(x)$. Figure adapted from the Stanford DCP Analyzer [2].

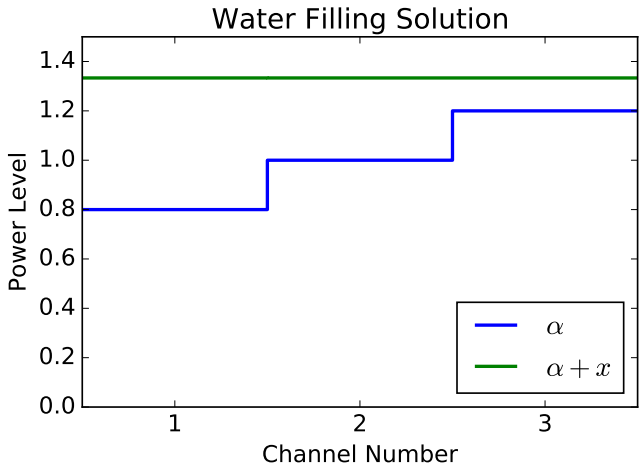

Figure 3. Water filling solution for three channels.

We will show how `cvxpy` can be used simply to test a variety of optimisation problems, as a powerful easy-to-use tool for gaining insight into real-world problems.

REAL WORLD APPLICATIONS

Water filling

Convex optimisation can be used to solve the classic water filling problem. In communications, this is where a total power P must be assigned to n different communication channels, with the objective of maximising the total communication rate. The communication rate of the the ith channel, r_i, is given by [1, p.245],

$$r_i = \log(x_i + \alpha_i), \tag{4}$$

where x_i is the power allocated to channel i and α_i represents the floor above the baseline at which power can be added to the channel. From Fig. 3, we can see how this is analogous to filling an uneven basin to a constant level.

The optimisation problem is then [1]:

$$\begin{aligned} &\text{minimise} && f_0(\mathbf{x}) = -\sum_{i=1}^{n}\log(\alpha_i + x_i), \quad \boldsymbol{\alpha} \succ \mathbf{0} \\ &\text{subject to} && \mathbf{x} \succeq \mathbf{0}, \\ & && \sum_{i=1}^{n} x_i = \mathbb{1}^T\mathbf{x} = P \end{aligned} \tag{5}$$

where symbols $\succ$, $\succeq$ denote elementwise inequalities between vectors or matrices.

This is a convex problem since $f_0(\mathbf{x})$ has positive curvature within its domain, and the inequality and equality constraints are convex.

Example `cvxpy` code in Python

To indicate the brevity of code implementation in `cvxpy` we show an abridged version below:

```
# Define variable (x) & parameter (a) with total
    power to allocate (sum_x) and number of
    channels (n).
x = cvx.Variable(n)
alpha = cvx.Parameter(n,sign='positive')
alpha.value = a

# Define objective function
obj = cvx.Maximize(cvx.sum_entries
                   (cvx.log(Îś + x))
                   )

# Define constraints
constraints = [x >= 0,
               cvx.sum_entries(x) - sum_x ==0]

# Solve problem
prob = cvx.Problem(obj, constraints)
prob.solve()
```

Computational complexity

`cvxpy` employs different solvers for each problem and so they have different computational complexities; these were assessed by averaging the runtime over many runs. In the case of water filling, for problem sizes up to 1000 channels, the computational time scaled linearly, Fig. 4.Given that we have n items in this problem and

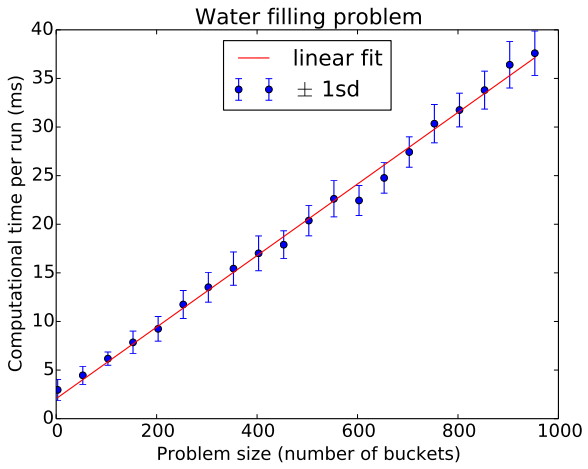

Figure 4. Linear plot of computation time (wall time) of water filling against problem size, for 1000 runs.

we apply the objective function to each channel independently, a computational complexity of $O(n)$ should be expected.

Maximising channel capacity

Convex optimization can be used to find the channel capacity C of a discrete memoryless channel. Consider a communication channel with input $X(t) \in \{1, 2, ..., n\}$ and output $Y(t) \in \{1, 2, ...m\}$. This means that the random variables X and Y can take n and m different values respectively.

In a discrete memoryless channel, the relation between the input and the output is given by the transition probability,

$$p_{ij} = \mathbb{P}(Y(t) = i | X(t) = j), \tag{6}$$

where these transition probabilities form the channel transition matrix P, with $P \in \mathbb{R}^{m \times n}$. Assume that X has a probability distribution denoted by $x \in \mathbb{R}^n$, meaning that,

$$x_j = \mathbb{P}(X(t) = j), \quad j = \{1, ..., n\}. \tag{7}$$

From Shannon [4], the channel capacity is given by the maximum possible mutual information, I, between X and Y,

$$C = \sup_x I(X;Y), \quad \text{where,} \tag{8}$$

$$I(X;Y) = -\sum_{i=1}^{m} y_i \log_2 y_i + \sum_{j=1}^{n}\sum_{i=1}^{m} x_j p_{ij} \log_2 p_{ij}. \tag{9}$$

Given that $x \log x$ is convex for $x \geq 0$, we can formulate this as a convex optimisation problem,

$$\text{minimise} \quad -I(X;Y), \tag{10}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = 1, \quad x \succeq 0. \tag{11}$$

Here the constraints arise from x describing a probability.

Example `cvxpy` *code*

Due to the built-in entropy function in `cvxpy`, `cvx.entr()`, the channel capacity problem is easy to write in DCP.

```
# n is the number of different input values
# m is the number of different output values
# x is probability distribution of the input signal
    X(t)
x = cvx.Variable(rows=n,cols=1)
# y is the probability distribution of the output
    signal Y(t)
# P is the channel transition matrix
y = P*x
# I is the mutual information between x and y
c = np.sum(P*np.log2(P),axis=0)
I = c*x + cvx.sum_entries(cvx.entr(y))
# Channel capacity maximised by maximising the
    mutual information
obj = cvx.Minimize(-I)
constraints = [cvx.sum_entries(x) == sum_x,x >= 0]
# Form and solve problem
prob = cvx.Problem(obj,constraints)
prob.solve()
```

Computational complexity

The computational complexity of the channel capacity problem follows a power law in `cvxpy`. For n up to 1000, the complexity is of order $\sim O(n^{2.5})$, Fig. 5. Given that we are applying matrix multiplication to the power vector **p**, a complexity of at least n^2 would be expected.

Minimum power for a given SINR

How can we minimise the total power consumption P of n transmitters each with power p_i, yet achieve a minimum SINR γ_0 for all m receivers? This question is relevant to a telecoms company who want to offer a service at a minimum quality standard. We formulate the problem with a given square path gain matrix G, background noise level σ and maximum power value that any transmitter can reach P_{max}:

$$\begin{aligned} &\underset{p}{\text{minimise}} && \sum_j p_j \\ &\text{subject to} && p_j \leq P_{\max} \\ &&& p_j \geq 0 \\ &&& \gamma_i \geq \gamma_0 \end{aligned}$$

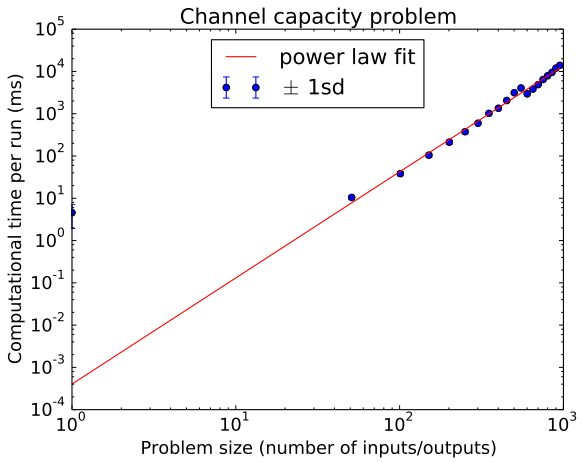

Figure 5. Log-log plot of computation time (wall time) of channel capacity against problem size, for 1000 runs.

The desired signal to receiver i is denoted by $S_i = G_{ik}p_k$, while the interference to i is $I_i = \sum_{j\neq k} G_{ij}p_j$. However DCP does not allow division of the optimisation variable p_i so

$$\gamma_i \geq \gamma_0 \iff \frac{S_i}{\sigma_i + I_i} \geq \gamma_0, \quad \forall i$$

was rearranged to:

$$S_i - \gamma_0(\sigma_i + I_i) \geq 0, \quad \forall i$$

which is DCP.

Path gain

The path gain G_{ij} represents the proportion of power that reaches receiver i from transmitter j. Supposing that the desired signal to receiver i is from transmitter k, the power received is,

$$p_{ik}^{\text{rec}} = G_{ik}p_k. \tag{12}$$

The signal-to-interference-plus-noise ratio (SINR) is the strength of the desired signal S_i relative to the interference power I_i plus the background noise σ_i at i. Denoting the SINR at i by γ_i, we have

$$\gamma_i = \frac{G_{ik}p_k}{\sigma_i + \sum_{j\neq k} G_{ij}p_j} = \frac{S_i}{\sigma_i + I_i}. \tag{13}$$

In a physical context, the path gain from transmitter j to receiver i will depend on the distance, d_{ij} between them. Assuming isotropic propagation from each transmitter, the fraction of power from j that reaches i is given by,

$$G_{ij} = k_i/d_{ij}^{\alpha} \tag{14}$$

where α is the pathloss coefficient, k is a proportionality constant specific to the receiver i. For free space $\alpha = 2$, while for an urban environment $\alpha \sim 3.5$. Further physical complexities, such as the stochastic effects from Rayleigh fading can be easily implemented.

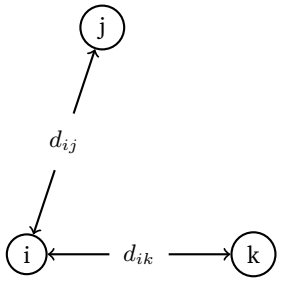

Figure 6. Path lengths between receiver i and transmitters j and k.

Example code

The `cvxpy` implementation is shown below.

```
# Declare variables
I = np.zeros((n,m)) # interference power matrix
S = np.zeros((n,m)) # signal power matrix
delta = np.identity(n)
S = G * delta # using gains matrix G
I = G - S

# Declare optimisation variable
p = cvx.Variable(n)

# Define objective function
obj = cvx.Minimize(
      cvx.sum_entries(p))

# Declare constraints
constraints =    [p >= 0,
                  S*p-alpha*(I*p+sigma)>=0,
                  p <= Pmax]

# Solve problem
prob = cvx.Problem(obj, constraints)
prob.solve()
```

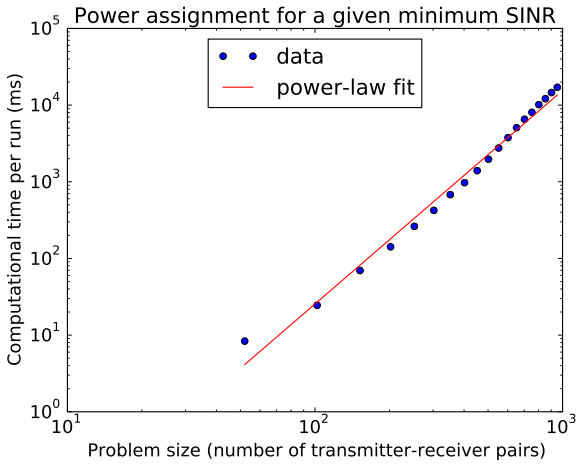

Figure 7. Log-log plot of computation time (wall time) of minimum power problem against the problem size, for 25 runs.

The computational complexity of the power minimisation for n up to 1000 is of order $\sim O(n^{2.8})$.

CONCLUSIONS

This article has shown that many well-known convex problems in communications can be written in a DCP format, meaning that they can be easily implemented in `cvxpy`.

Our example code demonstrates a key strength of `cvxpy`, which is that the translation from the mathematical description to python code is straightforward and clear. Conversely, one can also infer the mathematical problem from reading the code.

The computational complexity of the problems examined had a clear trend which aligned with expectation. This shows that convex optimisation solvers are robust to a variety of problem sizes and are scalable to at least problems of size 1000.

While all problems that can be written in DCP are convex, not all convex problems of interest can be written in DCP. Over time, the library of available DCP functions in `cvxpy` has grown, increasing the scope of convex optimisation problems that `cvxpy` can solve. This expansion of scope of `cvxpy` is ongoing, with version 1.0 under development and expected in the near future.

Acknowledgements

We would like to thank S. Johnson and S. Diamond for helpful discussions.

APPENDIX

Full code can be found at
https://github.com/cvxgrp/cvxpy/tree/master
/examples/communications.

[1] Steven Boyd and L Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
[2] S. Diamond. Stanford dcp analyzer. `http://dcp.stanford.edu/analyzer`, 2013.
[3] S Diamond and S Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016.
[4] Claude E. Shannon and Warren Weaver. *The mathematical theory of communication*. University of Illinois Press, 1949.