Support for complex numbers within Convex.jl

Artem Oboturov

May 3, 2016

1 Introduction[1]

Convex.jl is a package for Disciplined Convex Programming. Convex.jl makes it easy to describe optimization problems in a natural, mathematical syntax, and to solve those problems using a variety of different (commercial and open-source) solvers, through the MathProgBase interface. This project would add support for solving complex semidefinite programs (SDP) to Convex.jl.

Many problems in applied mathematics, engineering, and physics are most naturally posed as convex optimization problems over complex valued variables and with complex valued data. These include:

1. Phase retrieval from sparse measurements
2. Optimization problems in AC power systems
3. Frequency domain analysis in signal processing and control theory

While optimization over complex numbers can always be encoded as optimization over real variables through transformations, this often results in significant overhead (both in user effort and computation time) in many applications. Support for complex convex optimization in Convex.jl would boost the usage of Julia as a language of choice for users working on these and other applications.

This work entails writing functions to transform complex SDPs into equivalent real valued SDPs, and to transform the solutions back from real to complex variables.

Students with further background and motivation could continue to improve the SDP solver itself. In particular, the transformations used by Convex.jl to write a problem as an SDP often introduce many extra variables and constraints than are necessary, and may result in poor conditioning. A presolve routine, eliminating redundant variables and constraints and improving conditioning before passing the problem to a solver, would be a welcome addition to the Convex.jl library. While many tricks for presolving LPs are well known, there is significant room for imagination in writing a presolve for SDP; the project might well lead to a publication were the student so inclined.

2 Discussion

As was suggested in the Convex.jl issue tracker [2] one could write constraints and goal function via real and imaginary parts of complex variables. An example of such approach is given in [3]. The disadvantage - it is too verbose:

```
function dnorm(L)
    J = involution(L)

    dx = size(J,1) |> sqrt |> x -> round(Int,x)
    dy = dx

    if prev_dx != dx
        M = E_(dy,dx)
        prev_dx = dx
    end

    Jr = real(J)
    Ji = imag(J)

    Xr  = Variable(dy*dx, dy*dx)
    Xi  = Variable(dy*dx, dy*dx)
    ρ0r = Variable(dx, dx)
    ρ0i = Variable(dx, dx)
    ρ1r = Variable(dx, dx)
    ρ1i = Variable(dx, dx)

    prob = maximize( retrϕ( ϕ(Jr,Ji)'*ϕ(Xr,Xi) ) )

    prob.constraints += trace(ρ0r) == 1
    prob.constraints += trace(ρ0i) == 0
    prob.constraints += trace(ρ1r) == 1
    prob.constraints += trace(ρ1i) == 0

    Mρ0r = reshape(M * vec(ρ0r), dy*dx, dy*dx)
    Mρ0i = reshape(M * vec(ρ0i), dy*dx, dy*dx)
    Mρ1r = reshape(M * vec(ρ1r), dy*dx, dy*dx)
    Mρ1i = reshape(M * vec(ρ1i), dy*dx, dy*dx)

    prob.constraints += isposdef( ϕ(ρ0r,ρ0i) )

    prob.constraints += isposdef( ϕ(ρ1r,ρ1i) )

    prob.constraints += isposdef( ϕ( [ Mρ0r Xr ; Xr' Mρ1r ], [ Mρ0i Xi ; -X

    solve!(prob)

    if prob.status != :Optimal
        println("DNORM error.")
        println("Input: $(L)")
        println("Input's Choi spectrum: $(eigvals(liou2choi(L)))")
        error("Could not compute the diamond norm.")
    end

    return prob.optval
end
```

```
end
```

All variables with a subindex r and i are excessive an should be removed in favor of simple definition via complex variables. Functions ϕ and $retr\phi$ should also be removed in this case.

The same problem expressed via complex variables is given below:

```
function dnorm(L)
    J = involution(L)

    dx = size(J,1) |> sqrt |> x -> round(Int,x)
    dy = dx

    if prev_dx != dx
        M = E_(dy,dx)
        prev_dx = dx
    end

    X = Variable(dy*dx, dy*dx)
    ρ0 = Variable(dx, dx)
    ρ1 = Variable(dx, dx)

    prob = maximize( trace(J'*X) )

    prob.constraints += trace(ρ0) == 1
    prob.constraints += trace(ρ1) == 1

    Mρ0 = reshape(M * vec(ρ0), dy*dx, dy*dx)
    Mρ1 = reshape(M * vec(ρ1), dy*dx, dy*dx)

    prob.constraints += isposdef( ρ0 )

    prob.constraints += isposdef( ρ1 )

    prob.constraints += isposdef( [ Mρ0 X ; X' Mρ1] )

    solve!(prob)

    if prob.status != :Optimal
        println("DNORM error.")
        println("Input: $(L)")
        println("Input's Choi spectrum: $(eigvals(liou2choi(L)))")
        error("Could not compute the diamond norm.")
    end

    return prob.optval
end
end
```

It is more than 30% shorter than the original and much more concise. One could also check that new representation works for real domain without any changes whatsoever!

3 Practical approaches

Explanation of transformation from a complex SDP to an SDP over $\mathbb{R}^N$ is given in [4, Section 3].

As is specified in [5, Section 5.3]:

> One or both sides of an equality constraint may be complex; inequality constraints, on the other hand, must be real. A complex equality constraint is equivalent to two real equality constraints, one for the real part and one for the imaginary part. An equality constraint with a real side and a complex side has the effect of constraining the imaginary part of the complex side to be zero.

4 Proposed solution

Write a macro for equality and inequality constraints.

Provide a custom form for trace $(Q * X)$ optimization problem.

References

[1] "Support for complex numbers within convex.jl." `http://julialang.org/soc/ideas-page.html#support-for-complex-numbers-within-convexjl`. Accessed: 2016-04-30.

[2] "Support for complex variables?." `https://github.com/JuliaOpt/Convex.jl/issues/103`. Accessed: 2016-05-02.

[3] "dnorm.jl." `https://github.com/BBN-Q/SchattenNorms.jl/blob/master/src/dnorm.jl`. Accessed: 2016-05-02.

[4] M. X. Goemans and D. Williamson, "Approximation algorithms for max-3-cut and other problems via complex semidefinite programming," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pp. 443–452, ACM, 2001.

[5] "The cvx users' guide, release 2.1." `http://cvxr.com/cvx/doc/CVX.pdf`. Accessed: 2016-05-03.