REPORT 2

Mathieu Gaillard, Yichen Sheng

April 7, 2019

What has been done

We changed the code to run examples from the matrix repository. We wasted some time because of a bug in the MatrixDepot module. The issue is described here: https://github.com/JuliaMatrices/MatrixDepot.jl/issues/34 A possible workaround on Windows is to use this line of code after mdopen:

```
md = mdopen(name)
# Workaround
MatrixDepot.addmetadata!(md.data)
```

We changed the code so that we do not rely on another LP solver to get a feasible starting point. Instead we start from the point $x = 1$, $\lambda = 0$, $s = 1$ and it works because we use the path-following infeasible-interior-point method. It may be slower and not optimal, but it works. We will read chapter 5 and 6 of the book to improve our implementation and to answer some questions we have. How to choose the starting point? How to pick the values of σ and α.

We can now solve 3 problems from the list on the project description

- lp_afiro => OK
- lp_agg => OK
- lp_stocfor1 => OK
- lp_brandy => Singular exception
- lp_fit1d => Singular exception (Cause: conversion to standard form)
- lp_ganges => Singular exception (Cause: conversion to standard form)
- lpi_chemcom => Singular exception (Cause: conversion to standard form)
- lp_adlittle => Singular exception (Cause: Numerical divergence, works with a small tolerance)
- lp_25fv47 => Singular exception

The main problem we are facing is when we convert the problems to standard form. Here is out current code:

```
cs = [problem.c; zeros(n)]
As = [problem.A zeros(m, n);
            Matrix{Float64}(I,n,n) Matrix{Float64}(I,n,n)]
bs = [problem.b - problem.A * problem.lo;
            problem.hi - problem.lo]
```

We introduce a lot of numerical imprecisions when we compute $problem.hi - problem.lo$ because most of the time, $hi = 1e308$ and $lo = 0.0$.

Future work

On the long term, we still need to do the following things:

Our algorithm to find the value of alpha could be perfected. Right now, we use the same principle as backtracking. However, the way we pick alpha depends on the algorithm we use to find the descent direction. We need to be certain that it is the most appropriate way. This is not critical as the only benefit we would get is speed and we prefer to focus on robustness.

The top priority is to improve the numerical robustness of our implementation. There is a section about this problem in the introduction. We could also implement the Mehrotra's Predictor-Corrector Algorithm as it seems to be a well-established method.

Right now, we do not check if the problem is unbound in the opposite of the gradient direction. When that happens, our algorithm will run optimization process in max allowed steps and return a value that does not satisfy KKT condition. We don't know yet a method to test that.