Exploiting problem structure in optimization with GeNIOS.jl

(GEneralized Newtown Inexact Operator Splitting solver)

Theo Diamandis, Zachary Frangella, Bartolomeo Stellato, Madeleine Udell

MIT JuliaLab · **Juliacon 2023**

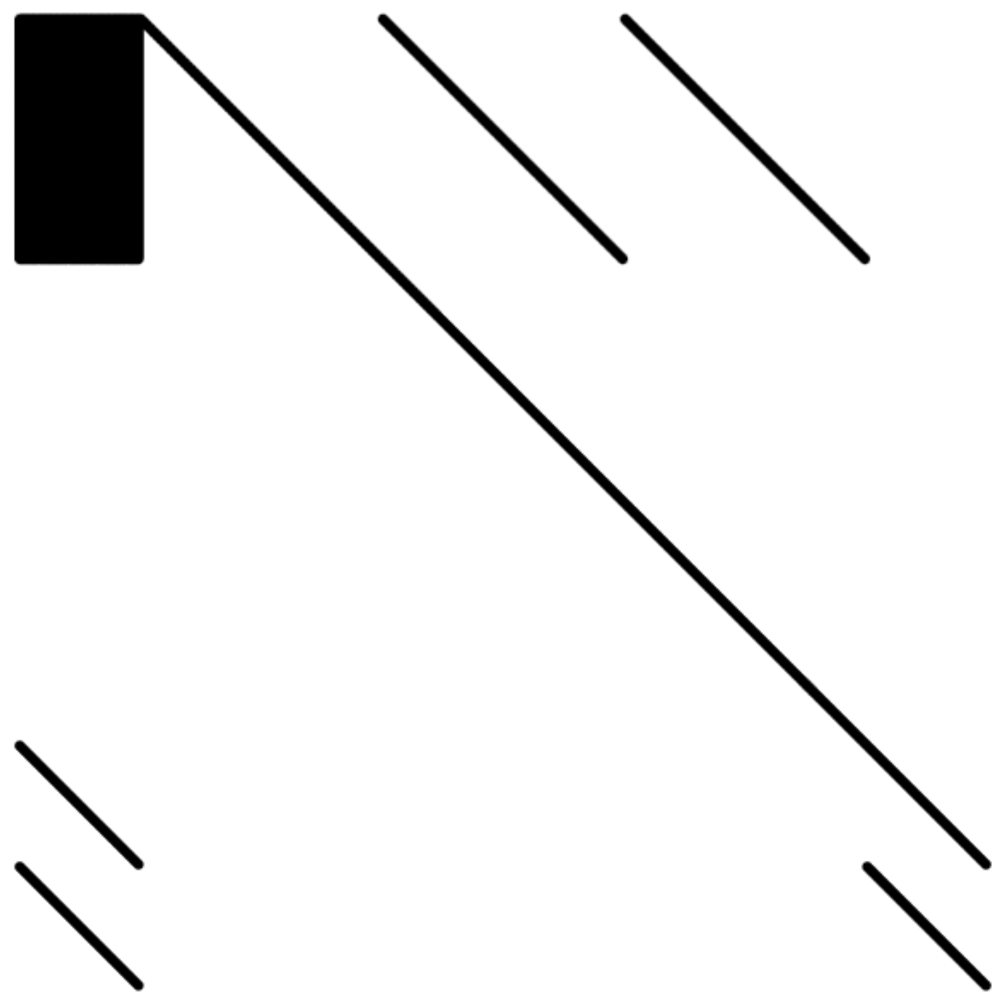
Exploiting structure yields huge speedups

(More than simply using SparseArrays)

- Most optimization problems have a lot of problem structure

Exploiting structure yields huge speedups
(More than simply using SparseArrays)

- Most optimization problems have a lot of problem structure

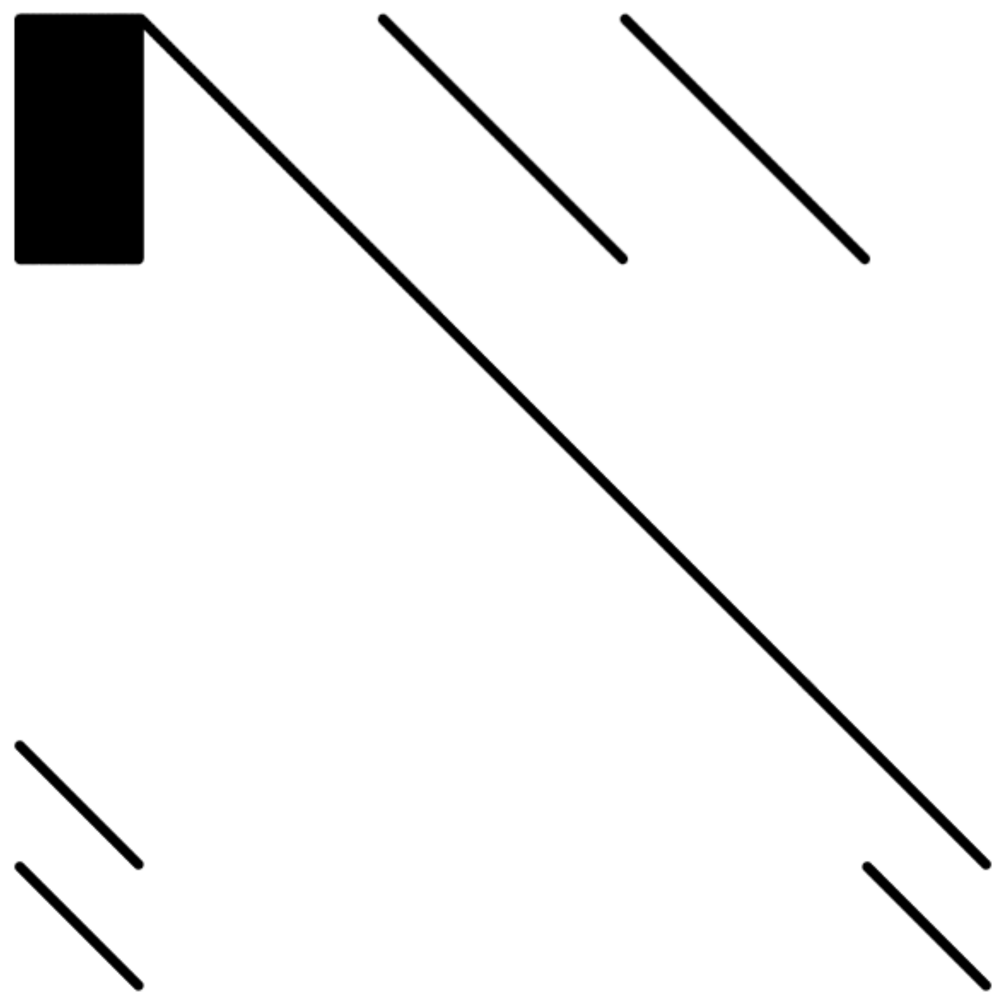


Huber fitting (as a quadratic program)

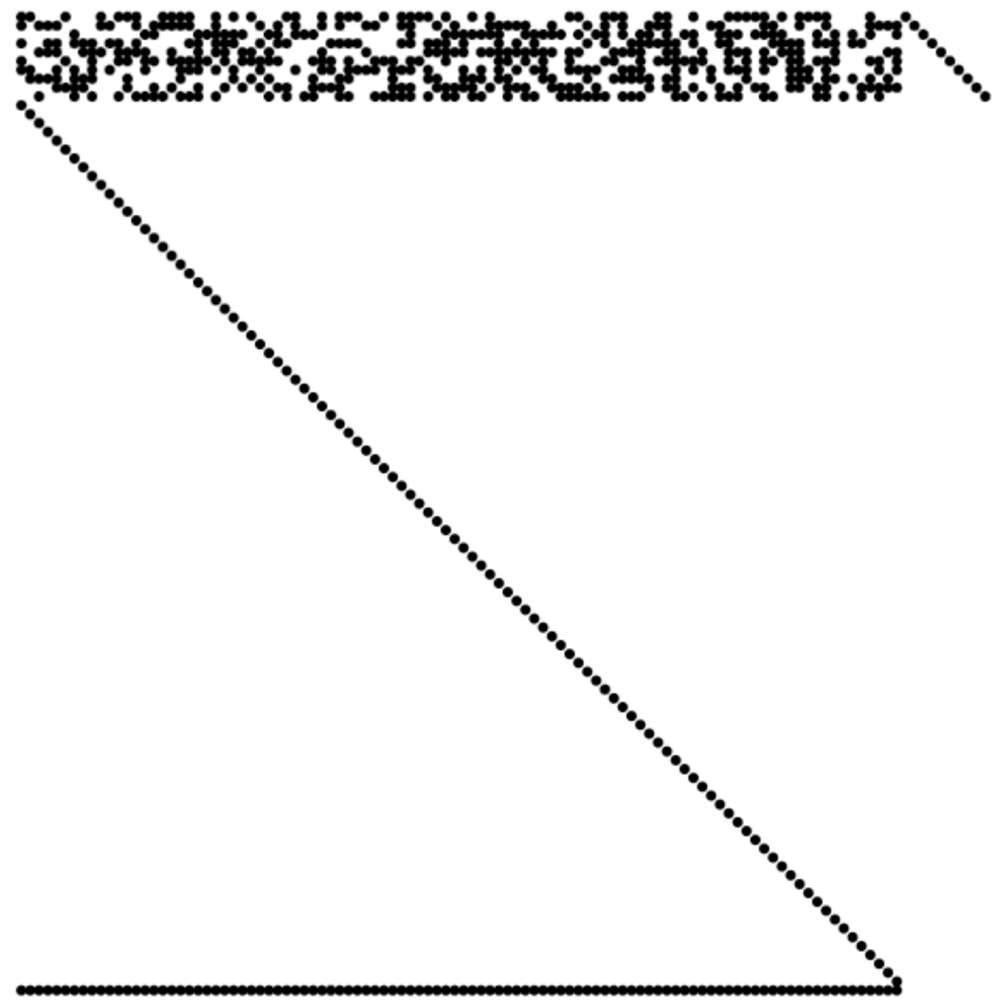
Exploiting structure yields huge speedups

(More than simply using SparseArrays)

- Most optimization problems have a lot of problem structure



Huber fitting (as a quadratic program)



Markowitz Portfolio Optimization

Exploiting structure yields huge speedups

(More than simply using SparseArrays)

- Most optimization problems have a lot of problem structure

Exploiting structure yields huge speedups
(More than simply using SparseArrays)

- Most optimization problems have a lot of problem structure

- Many solvers force problems into a standard *conic form*
    - Num variables $n \to 4n$: solve time can increase 2-3 orders of magnitude

Exploiting structure yields huge speedups

(More than simply using SparseArrays)

- Most optimization problems have a lot of problem structure
- Many solvers force problems into a standard *conic form*
    - Num variables $n \to 4n$: solve time can increase 2-3 orders of magnitude
- Harder to take advantage of problem data structure in this form

Example: robust (Huber) regression

Quadratic program formulation has ~5x the variables!

$$\text{minimize} \quad \sum_{i=1}^{N} \ell^{\text{hub}}(a_i^T x - b_i) + \lambda_1 \|x\|_1,$$

$$\ell^{\text{hub}}(w) = \begin{cases} w^2 & |w| \leq 1 \\ 2|w| - 1 & |w| > 1. \end{cases}$$

Huber robust regression

$$\begin{array}{ll} \text{minimize} & r^T r + 2\mathbf{1}^T(s+t) + \lambda_1 q \\ \text{subject to} & Ax - r - s + t = b \\ & -q \leq x \leq q \\ & 0 \leq s, t \end{array}$$

Equivalent quadratic program

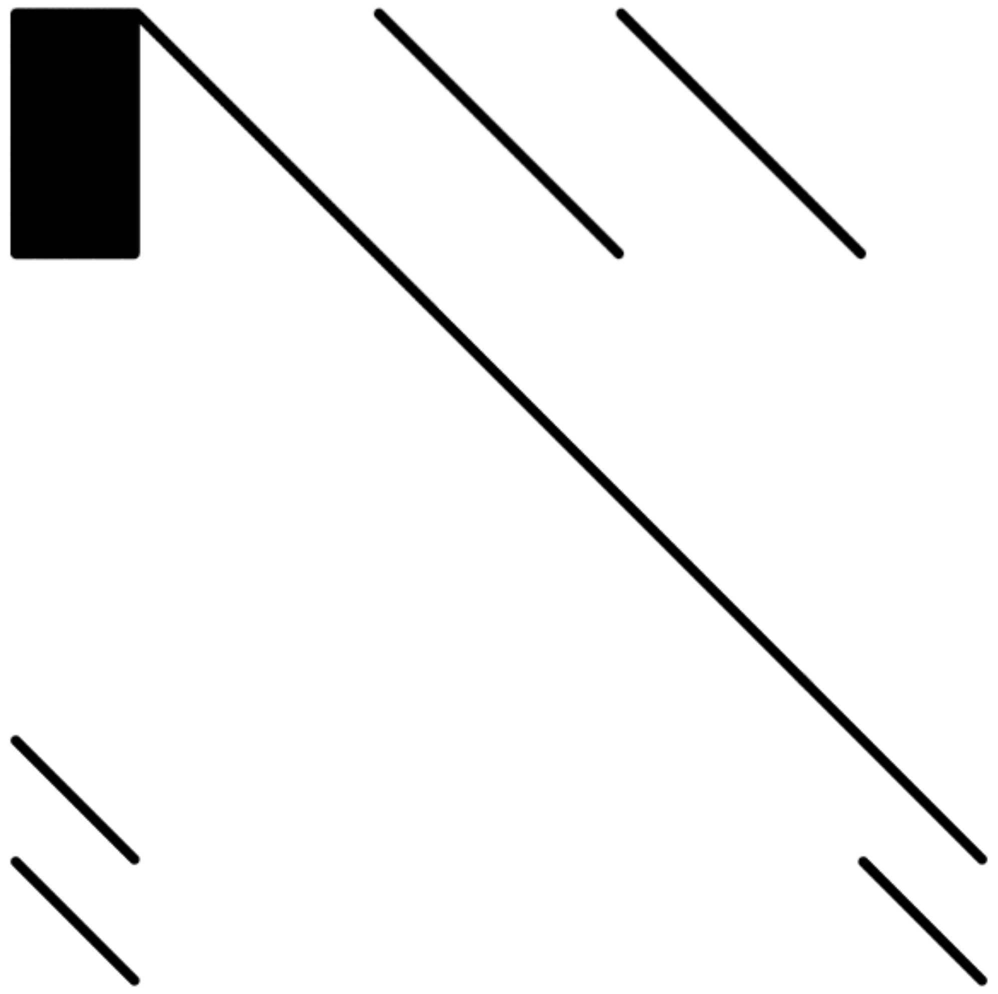
Example: robust (Huber) regression

Quadratic program formulation has ~5x the variables!

$$\text{minimize} \quad \sum_{i=1}^{N} \ell^{\text{hub}}(a_i^T x - b_i) + \lambda_1 \|x\|_1,$$

$$\ell^{\text{hub}}(w) = \begin{cases} w^2 & |w| \le 1 \\ 2|w| - 1 & |w| > 1. \end{cases}$$

Huber robust regression



$$\begin{array}{ll} \text{minimize} & r^T r + 2\mathbf{1}^T(s+t) + \lambda_1 q \\ \text{subject to} & Ax - r - s + t = b \\ & -q \le x \le q \\ & 0 \le s, t \end{array}$$

alent quadratic program

Example: robust (Huber) regression

Quadratic program formulation has ~5x the variables!

We'd prefer to solve this

$$\text{minimize} \quad \textstyle\sum_{i=1}^{N} \ell^{\text{hub}}(a_i^T x - b_i) + \lambda_1 \|x\|_1,$$

$$\ell^{\text{hub}}(w) = \begin{cases} w^2 & |w| \le 1 \\ 2|w| - 1 & |w| > 1. \end{cases}$$

Huber robust regression

$$\begin{array}{ll} \text{minimize} & r^T r + 2\mathbf{1}^T(s+t) + \lambda_1 q \\ \text{subject to} & Ax - r - s + t = b \\ & -q \le x \le q \\ & 0 \le s, t \end{array}$$

alent quadratic program

GeNIOS.jl solves convex problems…

$$\begin{array}{ll}\text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx - z = c,\end{array}$$

GeNIOS.jl solves convex problems…

$f(x), \quad \nabla f(x), \quad v \mapsto \hat{\nabla}^2 f(x) v$

$$
\begin{array}{ll}
\text{minimize} & f(x) + g(z) \\
\text{subject to} & Mx - z = c,
\end{array}
$$

GeNIOS.jl solves convex problems…

$f(x), \ \ \nabla f(x), \ \ v \mapsto \hat{\nabla}^2 f(x) v$ $\qquad$ $g(z), \ \ \hat{\mathbf{prox}}_{g/\rho}(z)$

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx - z = c, \end{array}$$

GeNIOS.jl solves convex problems…

$f(x), \quad \nabla f(x), \quad v \mapsto \hat{\nabla}^2 f(x) v$ $\qquad$ $g(z), \quad \hat{\mathbf{prox}}_{g/\rho}(z)$

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx - z = c, \end{array}$$

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u, \end{array}$$

Quadratic Programs

GeNIOS.jl solves convex problems…

$f(x), \quad \nabla f(x), \quad v \mapsto \hat{\nabla}^2 f(x) v$ $\quad\quad$ $g(z), \quad \hat{\mathbf{prox}}_{g/\rho}(z)$

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx - z = c, \end{array}$$

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & l \le Ax \le u, \end{array}$$

Quadratic Programs

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax - z = c \\ & z \in \mathcal{K}, \end{array}$$

Conic Programs

GeNIOS.jl solves convex problems…

$f(x), \quad \nabla f(x), \quad v \mapsto \hat{\nabla}^2 f(x) v$

$g(z), \quad \hat{\mathbf{prox}}_{g/\rho}(z)$

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx - z = c, \end{array}$$

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & l \le Ax \le u, \end{array}$$

Quadratic Programs

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \ell(a_i^T x - b_i) + (1/2)\lambda_2 \|x\|_2^2 + \lambda_1 \|z\|_1 \\ \text{subject to} & x - z = 0 \end{array}$$

Machine Learning Problems

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax - z = c \\ & z \in \mathcal{K}, \end{array}$$

Conic Programs

…using inexact ADMM

…using inexact ADMM

- ADMM involves solving two subproblems at each iteration

…using inexact ADMM

- ADMM involves solving two subproblems at each iteration
- We **approximate** these ADMM subproblems

…using inexact ADMM

- ADMM involves solving two subproblems at each iteration
- We **approximate** these ADMM subproblems
    - 2nd order approximation to 1st subproblem $\Longrightarrow$ **linear system solve**

…using inexact ADMM

- ADMM involves solving two subproblems at each iteration
- We **approximate** these ADMM subproblems
  - 2nd order approximation to 1st subproblem $\Longrightarrow$ **linear system solve**
    - Only need function, gradient, HVP (auto-diff is useful here)

…using inexact ADMM

- ADMM involves solving two subproblems at each iteration
- We **approximate** these ADMM subproblems
  - 2nd order approximation to 1st subproblem $\implies$ **linear system solve**
    - Only need function, gradient, HVP (auto-diff is useful here)
  - Approximate prox, $\mathbf{aprox}_g(v)$, operator for 2nd subproblem

…using inexact ADMM

- ADMM involves solving two subproblems at each iteration
- We **approximate** these ADMM subproblems
  - 2nd order approximation to 1st subproblem $\implies$ **linear system solve**
    - Only need function, gradient, HVP (auto-diff is useful here)
  - Approximate prox, $\mathbf{aprox}_g(v)$, operator for 2nd subproblem
- Then we **inexactly solve** these approximate subproblems

Back to robust (Huber) regression

Quadratic program formulation has ~5x the variables!

$$\text{minimize} \quad \textstyle\sum_{i=1}^{N} \ell^{\text{hub}}(a_i^T x - b_i) + \lambda_1 \|x\|_1,$$

$$\ell^{\text{hub}}(w) = \begin{cases} w^2 & |w| \leq 1 \\ 2|w| - 1 & |w| > 1. \end{cases}$$

Huber robust regression

$$\begin{array}{ll} \text{minimize} & r^T r + 2\mathbf{1}^T (s + t) + \lambda_1 q \\ \text{subject to} & Ax - r - s + t = b \\ & -q \leq x \leq q \\ & 0 \leq s, t \end{array}$$

Equivalent quadratic program

- Can try solving QP with/without inexact subproblem solves

MLSolver only needs loss & regularization

$$\text{minimize} \quad \sum_{i=1}^{m} \ell(a_i^T x - b_i) + \lambda_1 \|x\|_1 + (1/2)\lambda_2 \|x\|_2^2,$$

```julia
## Huber problem: min ∑ fʰᵘᵇ(aᵢᵀx - bᵢ) + λ||x||₁
f(x) = abs(x) <= 1 ? 0.5*x^2 : abs(x) - 0.5
λ1 = λ
λ2 = 0.0
solver = GeNIOS.MLSolver(f, λ1, λ2, A, b)
res = solve!(solver; options=GeNIOS.SolverOptions(use_dual_gap=false))
```

MLSolver only needs loss & regularization

$$\text{minimize} \quad \sum_{i=1}^{m} \ell(a_i^T x - b_i) + \lambda_1 \|x\|_1 + (1/2)\lambda_2 \|x\|_2^2,$$

- Supply conjugate -> can use duality gap convergence criterion

```julia
## Huber problem: min ∑ fʰᵘᵇ(aᵢᵀx - bᵢ) + λ||x||₁
f(x) = abs(x) <= 1 ? 0.5*x^2 : abs(x) - 0.5
λ1 = λ
λ2 = 0.0
solver = GeNIOS.MLSolver(f, λ1, λ2, A, b)
res = solve!(solver; options=GeNIOS.SolverOptions(use_dual_gap=false))
```

QPSolver just needs problem data mats/vecs

(Can be supplied through JuMP)

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & l \leq Ax \leq u,
\end{array}
$$

```
solver = GeNIOS.QPSolver(P, q, A, l, u)
res = solve!(solver)
```

QPSolver just needs problem data mats/vecs

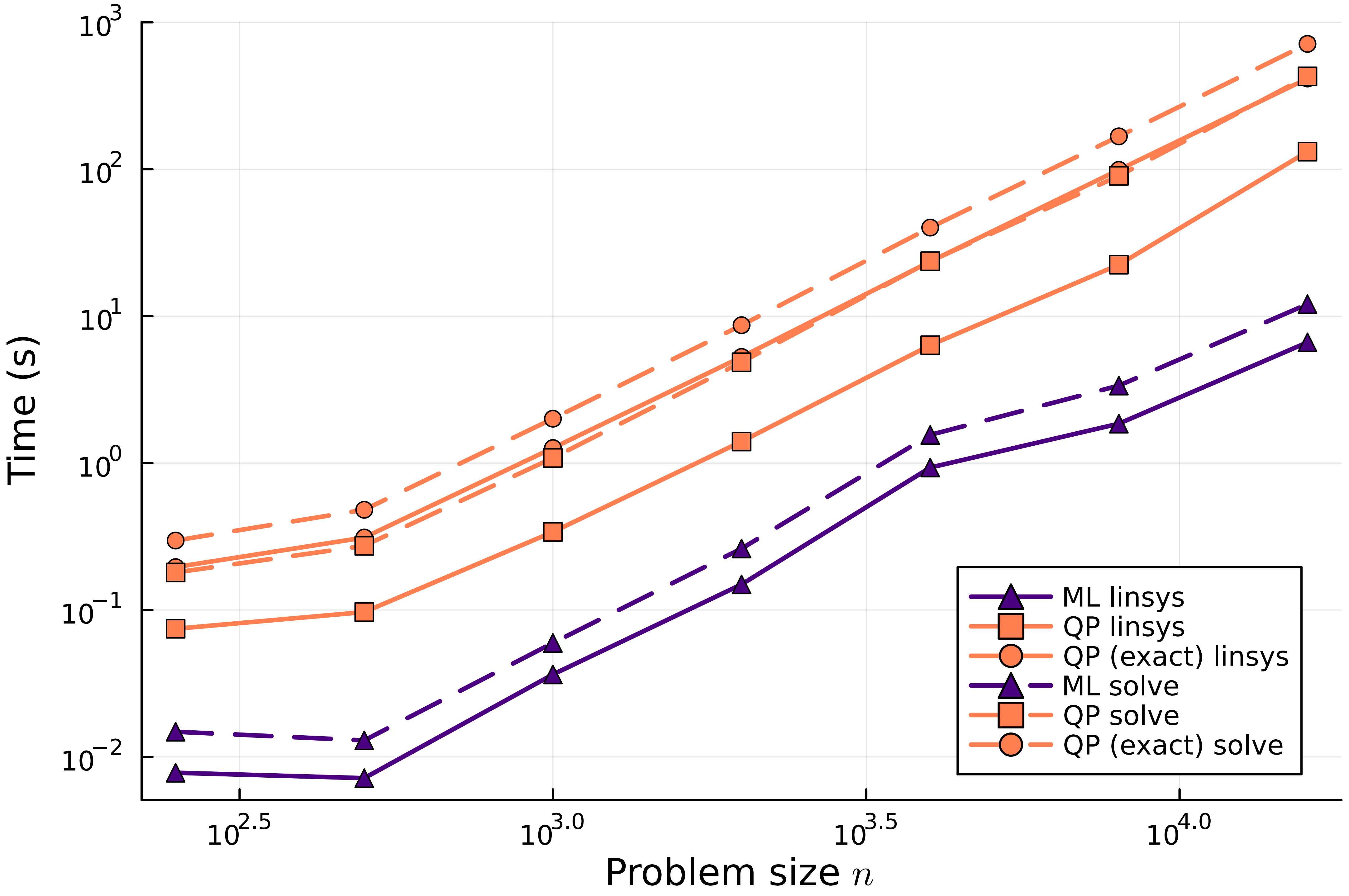
(Can be supplied through JuMP)

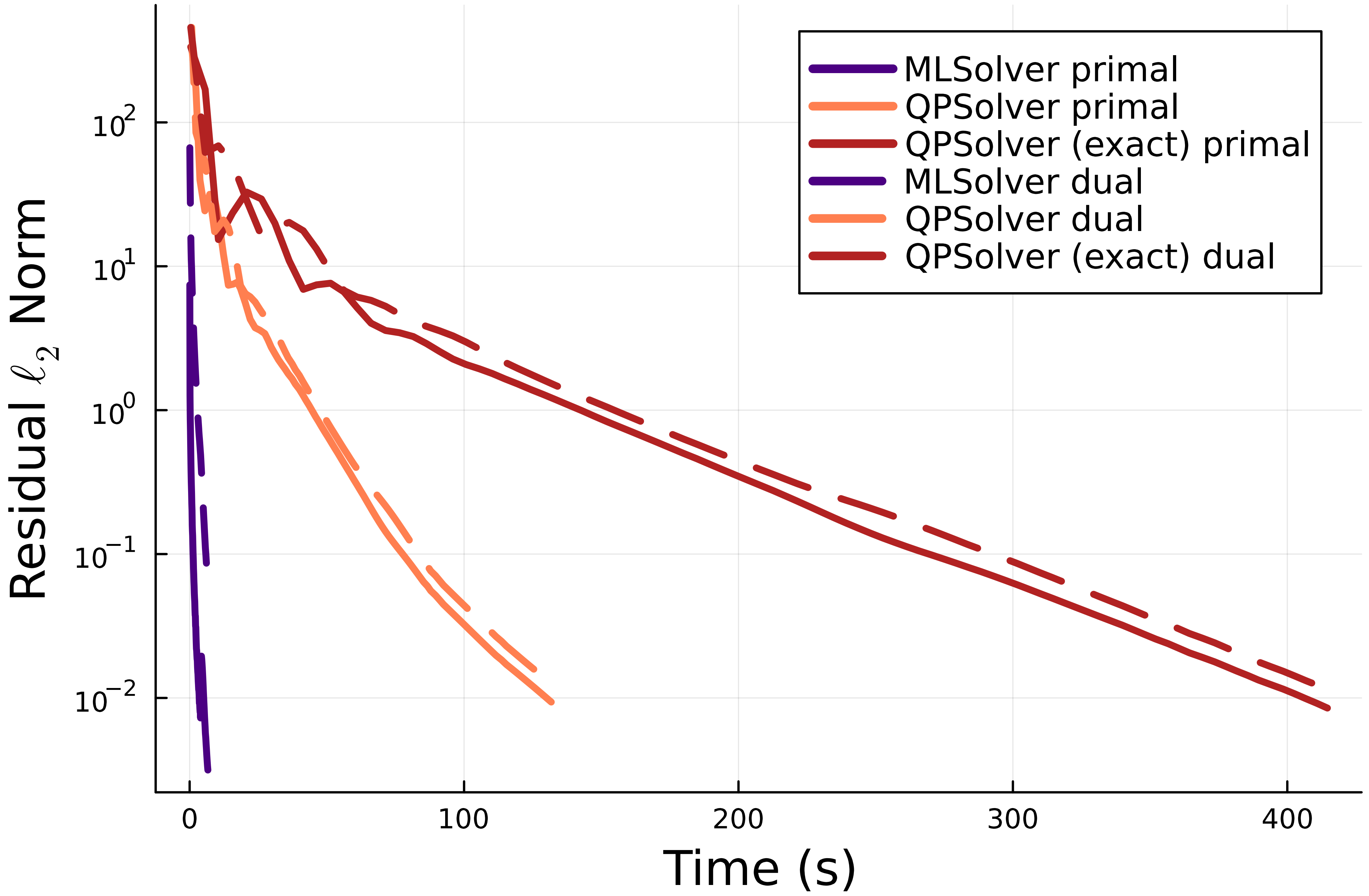
$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & l \leq Ax \leq u,
\end{array}
$$

```
solver = GeNIOS.QP[illegible], q, A, l, u)
res = solve[illegible]ver)
```

Or use JuMP

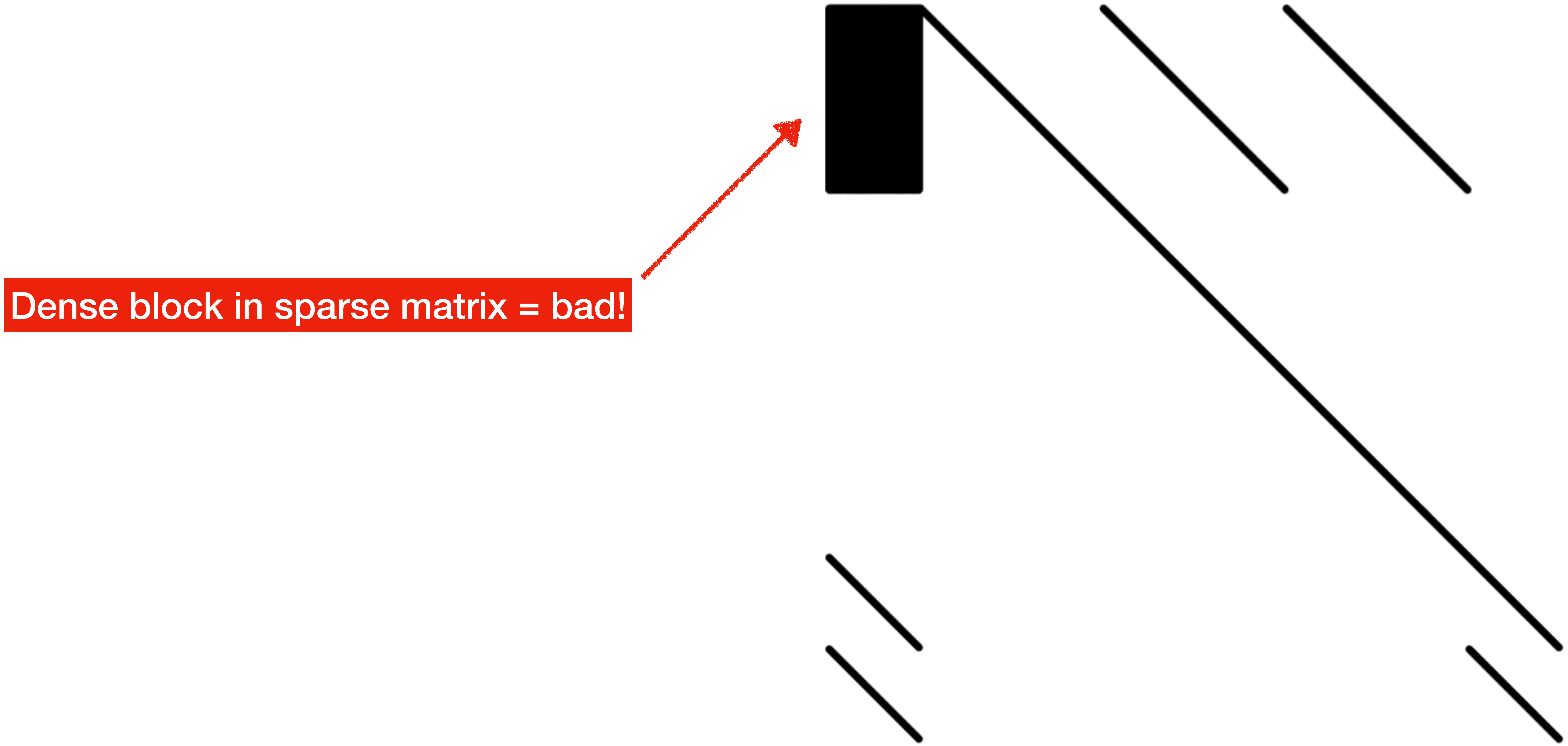
MLSolver gives 35x-60x speedup





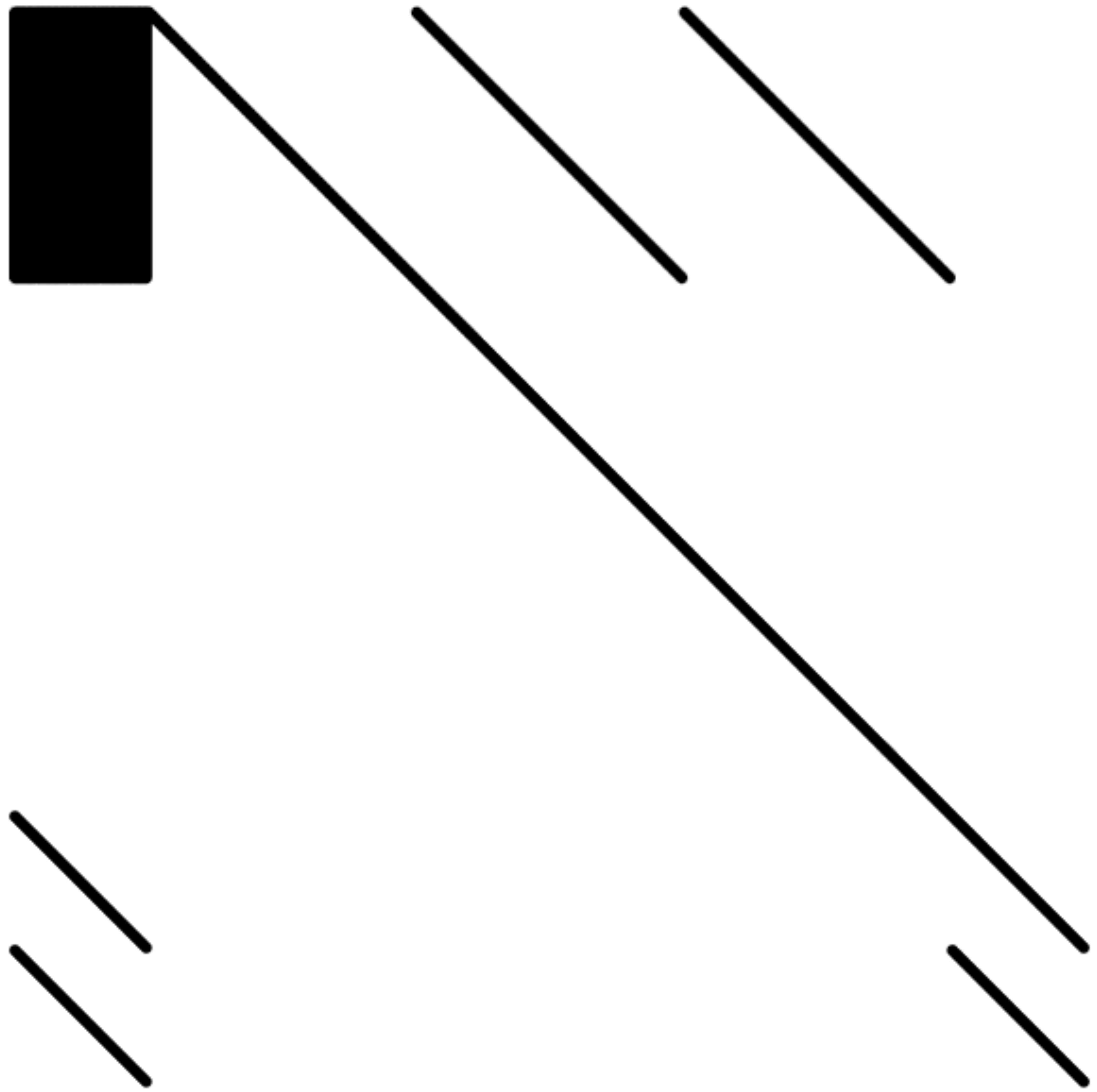
Sparsification slows mat-vec-products

(explains wider gap bt linsys & full solve)



Sparsification slows mat-vec-products

(explains wider gap bt linsys & full solve)



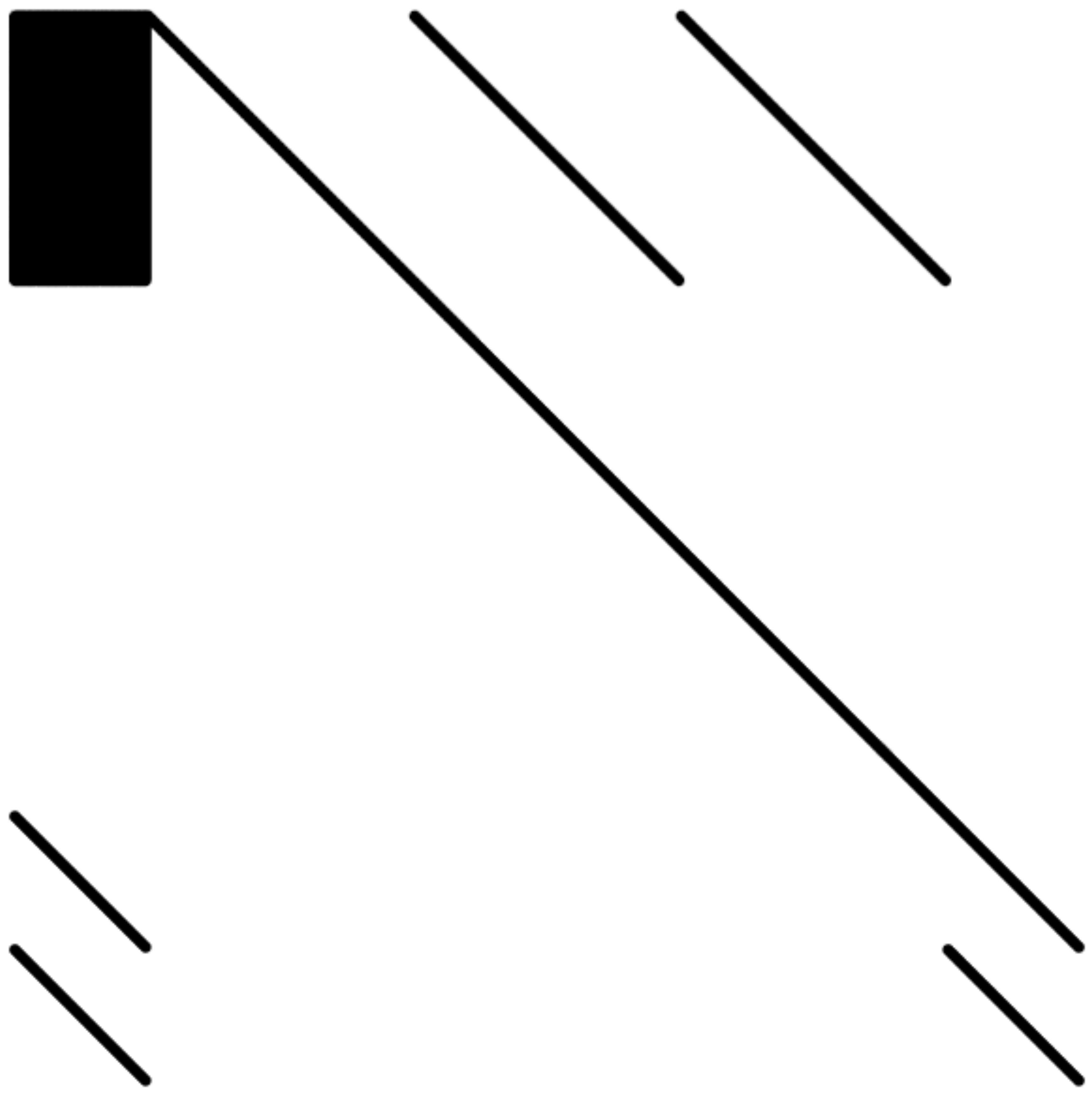
Dense block in sparse matrix = bad!

What about custom matrix type here?

Sparsification slows mat-vec-products

(explains wider gap bt linsys & full solve)

Dense block in sparse matrix = bad!



What about custom matrix type here?

Sounds like a lot of work…

Markowitz Portfolio Optimization

i.e., maximize risk-adjusted return of a long-only portfolio

$$\begin{array}{ll}\text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x \\ \text{subject to} & \mathbf{1}^T x = 1 \\ & x \geq 0,\end{array}$$

Markowitz Portfolio Optimization

i.e., maximize risk-adjusted return of a long-only portfolio

$$
\begin{array}{ll}
\text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x \\
\text{subject to} & \mathbf{1}^T x = 1 \\
& x \geq 0,
\end{array}
$$

- Covariance is **<u>diagonal + low rank</u>**: $\Sigma = D + FF^T$ where $F \in \mathbf{R}^{n \times k}$, $k \ll n$

Markowitz Portfolio Optimization

i.e., maximize risk-adjusted return of a long-only portfolio

$$
\begin{array}{ll}
\text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x \\
\text{subject to} & \mathbf{1}^T x = 1 \\
& x \geq 0,
\end{array}
$$

- Covariance is **<u>diagonal + low rank</u>**: $\Sigma = D + FF^T$ where $F \in \mathbf{R}^{n \times k}$, $k \ll n$
- This is a quadratic program:

$$
\begin{array}{ll}
\text{minimize} & (1/2) x^T P x + q^T x \\
\text{subject to} & l \leq Ax \leq u,
\end{array}
$$

How can we take advantage of structure?

Traditional method: solve *equivalent* QP with n+k variables

- An equivalent QP w new variable $y \in \mathbf{R}^k$:

$$\begin{array}{ll} \text{minimize} & \gamma x^T D x + \gamma y^T y - \mu^T x \\ \text{subject to} & y = F^T x \\ & \mathbf{1}^T x = 1 \\ & x \geq 0. \end{array}$$

- Pro: much faster solve
- Con: burden on user to do reformulation (can be very complex in other cases!)

Julia way: multiple dispatch

We can create fast mat-vec-products for constraint & obj matrices

```
using LinearMaps
## P = γ*(F*F' + Diagonal(d))
F_lm = LinearMap(F)
P = γ*(F_lm*F_lm' + Diagonal(d))

## M = vcat(I, ones(1, n))
M = vcat(LinearMap(I, n), ones(1, n))

solver = GeNIOS.QPSolver(P, q, M, l, u; check_dims=false);
res = solve!(solver; options=GeNIOS.SolverOptions(eps_abs=1e-6));
```

Julia way: multiple dispatch

We can create fast mat-vec-products for constraint & obj matrices

```
using LinearMaps
## P = γ*(F*F' + Diagonal(d))
F_lm = LinearMap(F)
P = γ*(F_lm*F_lm' + Diagonal(d))

## M = vcat(I, ones(1, n))
M = vcat(LinearMap(I, n), ones(1, n))

solver = GeNIOS.QPSolver(P, q, M, l, u; check_dims=false);
res = solve!(solver; options=GeNIOS.SolverOptions(eps_abs=1e-6));
```

- But the solver allows us to solve an even more interesting formulation…

We don’t even need to use a QP solver!

projection onto the simplex is quite fast itself

We don't even need to use a QP solver!

projection onto the simplex is quite fast itself

- Equivalent convex (non-QP) problem:

$$\begin{array}{ll} \text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x + I_S(z) \\ \text{subject to} & x - z = 0, \end{array}$$

- where I_S is an indication function of the set $S = \{z \mid \mathbf{1}^T z = 1$ and $z \geq 0\}$

We don’t even need to use a QP solver!

projection onto the simplex is quite fast itself

- Equivalent convex (non-QP) problem:

$$\begin{array}{ll} \text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x + I_S(z) \\ \text{subject to} & x - z = 0, \end{array}$$

- where I_S is an indication function of the set $S = \{z \mid \mathbf{1}^T z = 1$ and $z \geq 0\}$

- Z-subproblem is a projection onto S

We don't even need to use a QP solver!

projection onto the simplex is quite fast itself

- Equivalent convex (non-QP) problem:

$$\begin{array}{ll} \text{minimize} & -\mu^T x + (\gamma/2) x^T \Sigma x + I_S(z) \\ \text{subject to} & x - z = 0, \end{array}$$

- where I_S is an indication function of the set $S = \{z \mid \mathbf{1}^T z = 1 \text{ and } z \geq 0\}$
- Z-subproblem is a projection onto S
  - can be solved by 1d root finding (read: really really fast)

Generic problem can be specified directly

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx - z = c, \end{array}$$

```
solver = GeNIOS.GenericSolver(
    f, grad_f!, Hf,           # f(x)
    g, prox_g!,               # g(z)
    I, zeros(n)               # M, c: Mx - z = c
)
res = solve!(solver)
```

We saw four ‘equivalent’ ways to solve:

We saw four ‘equivalent’ ways to solve:

- Solve the original QP

We saw four ‘equivalent’ ways to solve:

- Solve the original QP
- Solve the original QP with fast operators (multiple dispatch!)

We saw four ‘equivalent’ ways to solve:

- Solve the original QP
- Solve the original QP with fast operators (multiple dispatch!)
- Solve the reformulated, equivalent QP

We saw four ‘equivalent’ ways to solve:

- Solve the original QP
- Solve the original QP with fast operators (multiple dispatch!)
- Solve the reformulated, equivalent QP
- Solve the equivalent convex problem (not a QP)

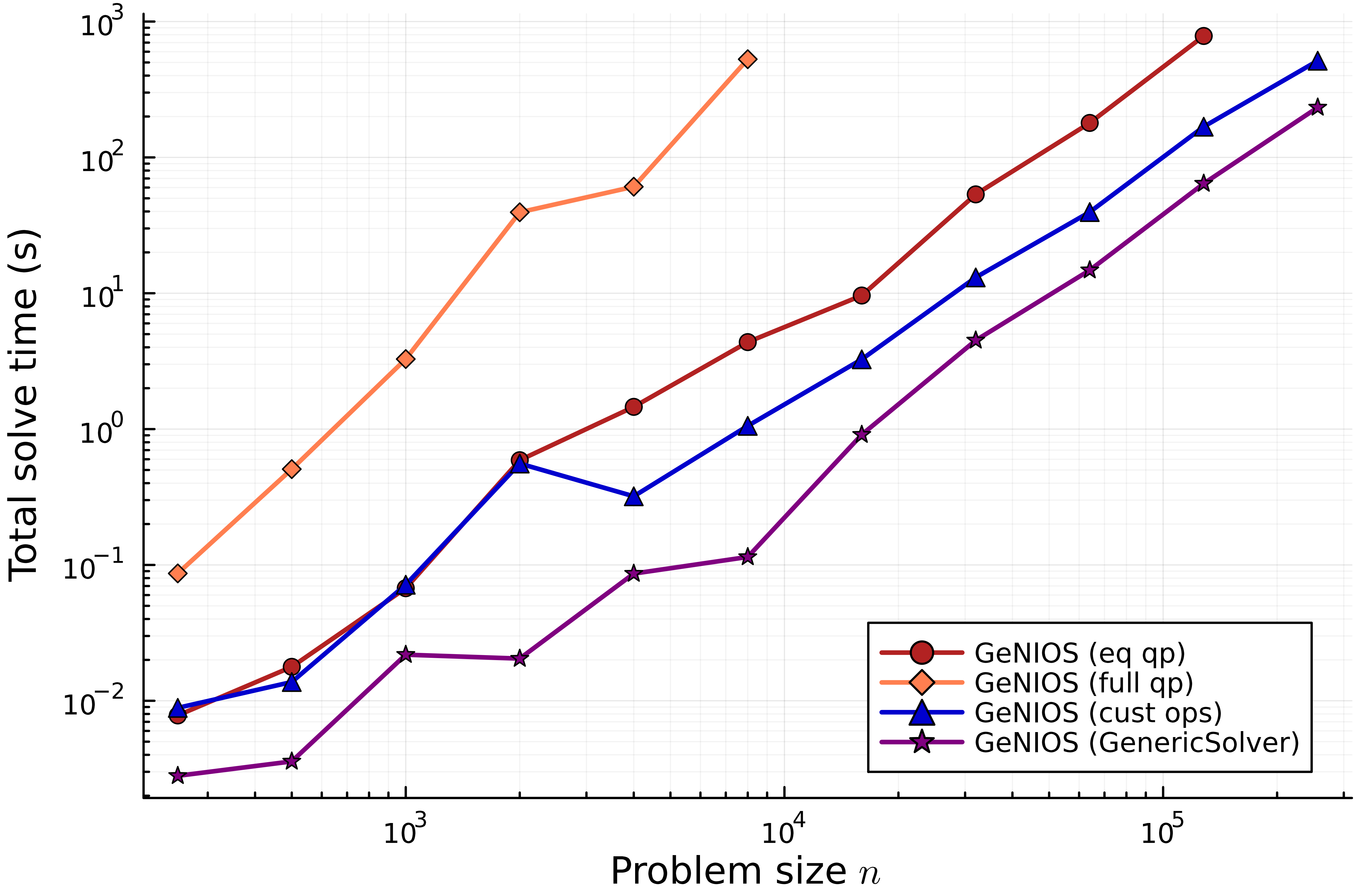
We saw four ‘equivalent’ ways to solve:

- Solve the original QP
- Solve the original QP with fast operators (multiple dispatch!)
- Solve the reformulated, equivalent QP
- Solve the equivalent convex problem (not a QP)

- How do they compare?

In short: the more structure used, the better

And Julia lets us avoid error-prone reformulations



Thank you! Questions?

- **Code & documentation:**

  https://github.com/tjdiamandis/GeNIOS.jl

- **Relevant theory paper:**

  Frangella, Z., Zhao, S., Diamandis, T., Stellato, B., & Udell, M. (2023). On the (linear) convergence of Generalized Newton Inexact ADMM. *arXiv preprint arXiv:2302.03863*.

- **Plan to expose more of the interface in JuMP, extend to more sets**

- **Email: tdiamand@mit.edu** (or open an issue on GitHub)

Appendix

ADMM is a popular first-order method for constrained optimization

ADMM is a popular first-order method for constrained optimization

- Core idea: split original problem (difficult) into 2+ easy problems
  - Repeatedly solve these problems & push solns together

ADMM is a popular first-order method for constrained optimization

- Core idea: split original problem (difficult) into 2+ easy problems
    - Repeatedly solve these problems & push solns together

- *Very* effective for large-scale, data-driven opt problems

Distributed optimization and statistical learning via the **alternating direction method** of **multipliers**

S Boyd, N Parikh, E Chu, B Peleato… - … and Trends® in …, 2011 - nowpublishers.com

… review, we argue that the **alternating direction method** of **multipliers** is well suited to distributed convex optimization, and in particular to large-scale problems arising in statistics, …

Save Cite Cited by 20291 Related articles All 43 versions

We use x and z to “decouple” subproblems

and then solve these subproblems until convergence…

$$
\begin{aligned}
x^{k+1} &= \underset{x}{\operatorname{argmin}} \left(f(x) + (\rho/2)\|Mx - z^k - c + u^k\|_2^2\right) \\
z^{k+1} &= \underset{z}{\operatorname{argmin}} \left(g(z) + (\rho/2)\|Mx^{k+1} - z - c + u^k\|_2^2\right) \\
u^{k+1} &= u^k + Mx^{k+1} - z^{k+1} - c.
\end{aligned}
$$

We use x and z to “decouple” subproblems

and then solve these subproblems until convergence…

$$
\begin{aligned}
x^{k+1} &= \underset{x}{\operatorname{argmin}} \left(f(x) + (\rho/2)\|Mx - z^k - c + u^k\|_2^2 \right) \\
z^{k+1} &= \underset{z}{\operatorname{argmin}} \left(g(z) + (\rho/2)\|Mx^{k+1} - z - c + u^k\|_2^2 \right) \\
u^{k+1} &= u^k + Mx^{k+1} - z^{k+1} - c.
\end{aligned}
$$

- But solving the x-subproblem can still be difficult / slow

We use x and z to “decouple” subproblems

and then solve these subproblems until convergence…

$$
\begin{aligned}
x^{k+1} &= \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|Mx - z^k - c + u^k\|_2^2 \right) \\
z^{k+1} &= \operatorname*{argmin}_z \left(g(z) + (\rho/2)\|Mx^{k+1} - z - c + u^k\|_2^2 \right) \\
u^{k+1} &= u^k + Mx^{k+1} - z^{k+1} - c.
\end{aligned}
$$

- But solving the x-subproblem can still be difficult / slow
  - Generally have to run an algorithm like L-BFGS at each iteration

We use x and z to “decouple” subproblems

and then solve these subproblems until convergence…

$$
\begin{aligned}
x^{k+1} &= \underset{x}{\operatorname{argmin}} \left(f(x) + (\rho/2)\|Mx - z^k - c + u^k\|_2^2 \right) \\
z^{k+1} &= \underset{z}{\operatorname{argmin}} \left(g(z) + (\rho/2)\|Mx^{k+1} - z - c + u^k\|_2^2 \right) \\
u^{k+1} &= u^k + Mx^{k+1} - z^{k+1} - c.
\end{aligned}
$$

- But solving the x-subproblem can still be difficult / slow
    - Generally have to run an algorithm like L-BFGS at each iteration
- A (perhaps silly?) idea: replace $f(x)$ with an easy approximation

In fact, we can get quite sloppy…

(Subject to a condition on subproblem errors)

In fact, we can get quite sloppy…

(Subject to a condition on subproblem errors)

- Idea 1: replace $f(x)$ with the second order Taylor expansion around x^k
  - => The x subproblem is just a linear system solve (!)
  - => Only require function, gradient, and Hessian-vector-product for f (!)
    - Makes interface very easy (& can leverage auto diff)

In fact, we can get quite sloppy…

(Subject to a condition on subproblem errors)

- Idea 1: replace $f(x)$ with the second order Taylor expansion around x^k
  - => The x subproblem is just a linear system solve (!)
  - => Only require function, gradient, and Hessian-vector-product for f (!)
    - Makes interface very easy (& can leverage auto diff)

- Idea 2: solve this linear system inexactly
  - We solve with CG method, decreasing the tolerance at each iteration

What about the z-subproblem?

- Often can put the hard/time-consuming parts of the problem in f
- But theory tells us we can solve z-subproblem inexactly too!

- We won't discuss here
- But interesting future directions…