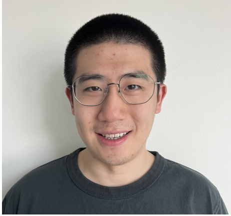
GPU-Accelerated Linear Programming and Beyond

Haihao Lu

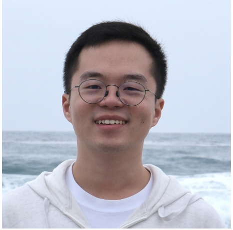
MIT

MIP Workshop, University of Minnesota

June, 2025



Jinwen Yang



Zedong Peng

Acknowledgment of my early collaboration with Google

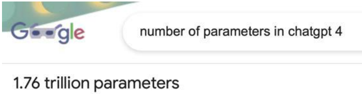
Mostly based on a series of papers:

- H Lu, J Yang (2023a) "cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia".
- H Lu, J Yang (2023b) "A Practical and Optimal First-Order Method for Large-Scale Convex Quadratic Programming".
- H Lu, J Yang (2023c) "On the Geometry and Refined Rate of Primal-Dual Hybrid Gradient for Linear Programming".
- H Lu, J Yang (2023d) "On a unified and simplified proof for the ergodic convergence rates of PPM, PDHG and ADMM".
- H Lu, J Yang (2024) "Restarted Halpern PDHG for Linear Programming".
- H Lu, Z Peng, J Yang (2024) "MPAX: Mathematical programming in JAX".
- ......

Motivation

Machine learning infrastructure has grown like crazy in the last 10 years

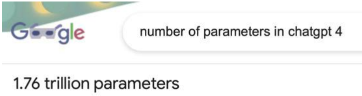
- Hardware: GPUs and TPUs
- Software: first-order method, Tensorflow and PyTorch



Motivation

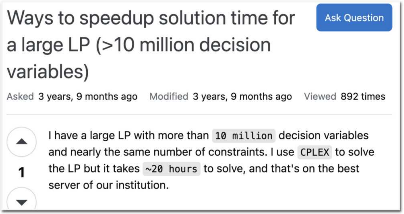
Machine learning infrastructure has grown like crazy in the last 10 years

- Hardware: GPUs and TPUs
- Software: first-order method, Tensorflow and PyTorch



The scale of mathematical programming we can handle is arguably stuck

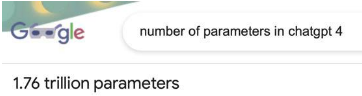
- Hardware: shared memory CPU
- Software: simplex/barrier method, Gurobi



Motivation

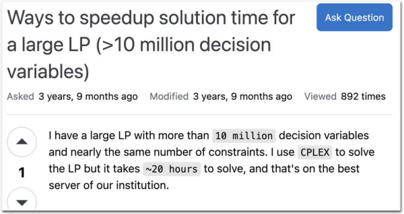
Machine learning infrastructure has grown like crazy in the last 10 years

- Hardware: GPUs and TPUs
- Software: first-order method, Tensorflow and PyTorch



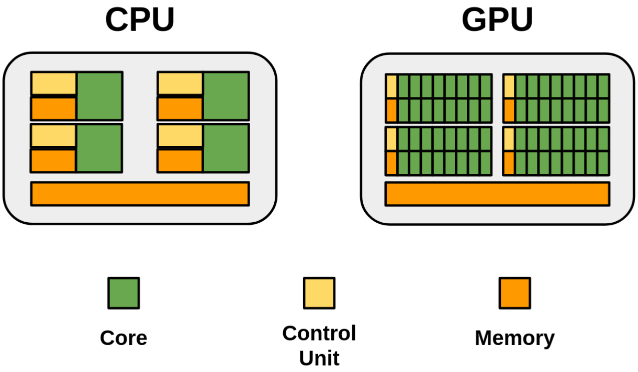
The scale of mathematical programming we can handle is arguably stuck

- Hardware: shared memory CPU
- Software: simplex/barrier method, Gurobi



Motivation: Can we use GPU and FOMs to speed and scale up mathematical programming?

Comparison of CPU and GPU



- CPU commonly has two to 64 cores, while GPU commonly has thousands or more cores
- CPU is better at serial tasks, and GPU is better at parallel tasks

What numerical operations are GPU good at?

For large-scale mathematical programming problems:

	Sparse linear system solve	Sparse matrix-vector multiplication
CPU		
GPU		
Methods	**Active-set / IPMs**	**FOMs**

- Traditionally, it was believed that GPU is not suitable for solving sparse linear systems.
- NVIDIA released cuDSS, which makes sparse solving possible on GPUs, but the speedup is not the scale of SpMV.

Algorithm and Implementation

Primal-Dual Hybrid Gradient (PDHG)

LP (in standard form):

$$\begin{aligned} &\min\ c^\top x \\ &\text{s.t.} Ax = b, x \geq 0 \end{aligned}$$

LP (primal-dual form):

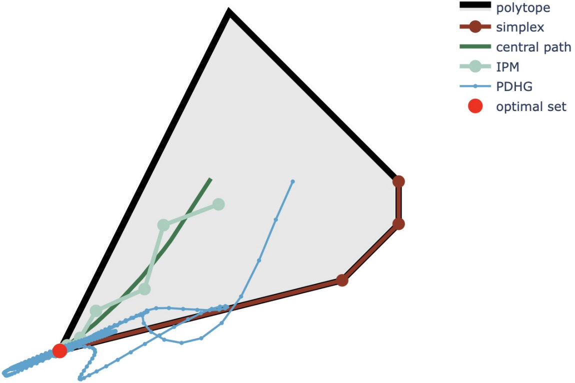
$$\min_{x\geq 0} \max_{y}\ c^\top x + y^\top b - y^\top Ax$$

Primal-Dual Hybrid Gradient (PDHG) [Chambolle and Pock 2011]

$$\begin{aligned} x^{k+1} &= \mathbf{proj}_{R^n_+}(x^k + \eta(A^\top y - c)) \\ y^{k+1} &= y^k - \tau(A(2x^{k+1} - x^k) - b) \end{aligned}$$

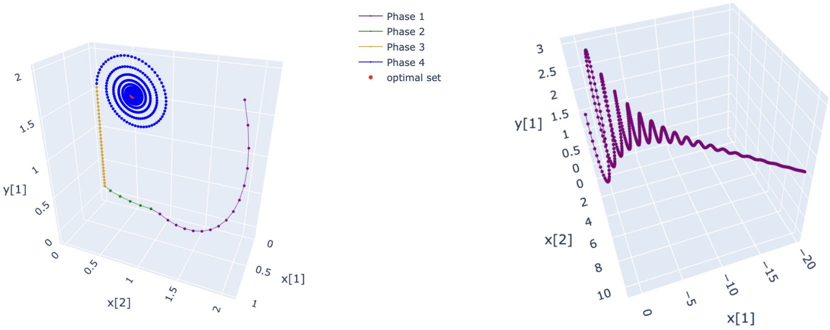
- η and τ are the primal and dual step-size, respectively
- Cost per iteration is matrix vector multiplication

Visualizing PDHG



- PDHG iterates restricted in the primal (or dual) space look mysterious

Visualizing PDHG



- PDHG iterates, in the primal-dual space, follow with "spiral rays", till the active basis changes
- The spiral improves feasibility, and the ray improves the primal-dual gap

PDLP(=Primal Dual Algorithms for LP)

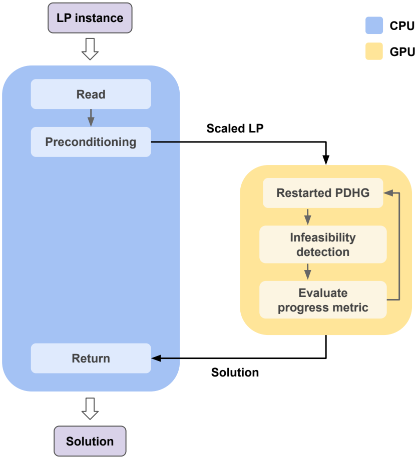
On a benchmark set with 383 instances and 1 hour time limit

Method	Solved to 10^{-4} (rela. err.)	Solved to 10^{-8} (rela. err.)
PDHG (CPU)	40%	19%
PDLP (CPU)	97%	85%

Indeed PDHG itself does not work well. We propose many enhancements

- Adaptive step sizes
- Diagonal preconditioning
- Infeasibility detection
- Primal weight updates
- Halpern iteration
- Reflection
- Restarts
- Feasibility polishing

cuPDLP and r^2HPDLP



cuPDLP ($\approx$ GPU-version of PDLP)

- Avoid all serial steps of PDLP
- All major steps are done on GPU
- Only two rounds of CPU-GPU communication

Recently, we propose $\mathrm{r}^2\mathrm{HPDLP}$ (restarted reflected Halpern version of PDLP) with better theory and practice

Computation

Mostly based on

- H Lu, J Yang (2023a) "cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia".
- H Lu, J Yang (2024) "Restarted Halpern PDHG for Linear Programming".

Major Message

Major Message:

- cuPDLP (GPU) is “on par” with state-of-the-art LP solvers
- r^2HPDLP overall has superior performance than cuPDLP

Datasets

- MIPLIB Relaxations (383 instances)

	Small	**Medium**	**Large**
Number of nonzeros	100K - 1M	1M - 10M	>10M
Number of instances	269	94	20

Table: Scales of instances in MIPLIB Relaxations

- Experiment details
  - Gurobi runs on CPU with 16 cores and 160GB of memory, crossover disabled
  - cuPDLP.jl/r^2HPDLP runs on H100 GPU with 80GB memory

cuPDLP/r^2HPDLP versus Gurobi, without presolve

	Small (269) (1-hour limit)		Medium (94) (1-hour limit)		Large (20) (5-hour limit)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**
Primal simplex (Gurobi)	268	7.81	73	140.18	13	1180.42
Dual simplex (Gurobi)	267	5.75	87	45.49	13	973.96
Barrier (Gurobi)	268	2.91	86	37.95	13	576.57
cuPDLP	266	8.61	92	14.80	19	111.19
r^2HPDHG	267	6.61	93	7.84	19	90.81

Table: Moderate accuracy Tol 10^{-4}

	Small (269) (1-hour limit)		Medium (94) (1-hour limit)		Large (20) (5-hour limit)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**
Primal simplex (Gurobi)	266	9.06	68	166.03	12	1578.04
Dual simplex (Gurobi)	265	7.14	84	60.97	11	1438.33
Barrier (Gurobi)	268	3.38	82	46.13	13	630.21
cuPDLP	261	23.47	86	40.69	16	421.40
r^2HPDHG	260	19.13	87	28.35	16	229.47

Table: High accuracy Tol 10^{-8}

cuPDLP/r^2HPDLP versus Gurobi, with presole

	Small (269) (1-hour limit)		Medium (94) (1-hour limit)		Large (20) (5-hour limit)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**
Primal simplex (Gurobi)	269	5.67	71	121.23	19	297.59
Dual simplex (Gurobi)	268	4.17	86	37.56	19	179.49
Barrier (Gurobi)	269	1.21	94	15.32	20	30.70
cuPDLP	269	5.35	93	10.31	19	33.93
r^2HPDHG	267	3.95	94	6.45	19	17.13

Table: Moderate accuracy Tol 10^{-4}

	Small (269) (1-hour limit)		Medium (94) (1-hour limit)		Large (20) (5-hour limit)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**
Primal simplex (Gurobi)	269	5.19	75	100.03	18	171.72
Dual simplex (Gurobi)	268	3.53	89	27.17	19	121.94
Barrier (Gurobi)	269	1.34	94	16.85	20	33.48
cuPDLP	264	17.53	90	30.05	19	81.07
r^2HPDHG	261	15.24	90	21.67	19	56.19

Table: High accuracy Tol 10^{-8}

cuPDLP versus PDLP, MIPLIB

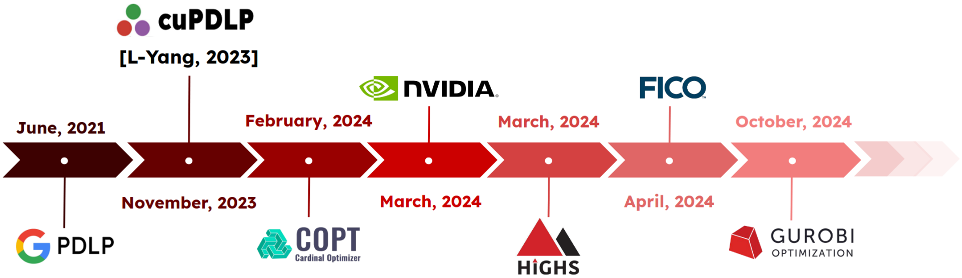
	Small (269) (1-hour limit)		Medium (94) (1-hour limit)		Large (20) (5-hour limit)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**
FirstOrderLp.jl	253	35.94	82	155.67	12	2002.21
PDLP (1 thread)	256	22.69	85	98.38	15	1622.91
PDLP (4 threads)	260	24.03	91	42.94	15	736.20
PDLP (16 threads)	238	104.72	84	142.79	15	946.24
cuPDLP	266	8.61	92	14.80	19	111.19
r^{2}HPDHG	267	6.61	93	7.84	19	90.81

Table: Moderate accuracy Tol 10^{-4}

	Small (269) (1-hour limit)		Medium (94) (1-hour limit)		Large (20) (5-hour limit)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**
FirstOrderLp.jl	235	91.14	68	389.34	9	3552.50
PDLP (1 thread)	250	49.31	73	259.04	12	3818.42
PDLP (4 threads)	245	54.19	81	136.16	14	1789.54
PDLP (16 threads)	214	248.34	69	403.17	14	2475.57
cuPDLP	261	23.47	86	40.69	16	421.40
r^{2}HPDHG	260	19.13	87	28.35	16	229.47

Table: High accuracy Tol 10^{-8}

Broader impact



Benchmarks for optimization software (By Hans Mittelmann)

65 probs	1	71.2	3.29	25.0	21.6	16.7	5.77	1.54	2.30	2.36
solved	65	36	59	48	49	50	59	65	57	56
probs	COPT	TULIP	MOSEK	HiGHS	KNITRO	PDLP%	XOPT	OPTV	CUPDL	CUOPT&

Figure: LPfeas Benchmark (find PD feasible point)

65 probs	27.2	1	1.68	7.45	17.0	59.4	93.5	6.60	1.84
solved	40	65	63	52	51	33	32	52	57
probs	CLP	COPT	OPTV	MOSEK	HiGHS	GLOP	SPLX	XOPT	CUPDL

Figure: LPopt Benchmark (find optimal basic solution)

Benchmarks for hard LP instances (By COPT)

On the ZIB03 instance

Solver	**Hardware**	**Time to Optimality**
Barrier Method (2009)	CPU	4 months
COPT (2023)	Modern CPU	16 hours
cuPDLP–C (2023)	NVIDIA H100 GPU	15 minutes

Table: Hard LP instances solved more than 60 times faster with cuPDLP-C.

Performance on large unit commitment instances (By ZIB)

Test Set	Tolerance	PDLP		IPM	
		Avg (s)	Geo mean (s)	Avg (s)	Geo mean (s)
X-Small	1e-4	13.12	13.10	30.27	29.87
	1e-6	25.59	22.66	33.20	32.55
Small	1e-4	9.7	9.18	73.16	72.34
	1e-6	30.68	26.14	89.19	86.74
Medium	1e-4	104.44	104.21	1035.30	1002.34
	1e-6	188.24	166.94	1283.83	1217.09
Large	1e-4	413.63	394.82	4447.56	4354.79
	1e-6	2145.26	1672.49	7014.48	6894.15
X-Large	1e-4	151.867	148.0	11391.09	11296.42
	1e-6	633.62	553.20	15405.88	15193.82
XX-Large	1e-4	480.78	437.52	TIMEOUT	TIMEOUT
	1e-6	3268.83	2181.21	TIMEOUT	TIMEOUT

Table 4: Performance comparison of PDLP and IPM solver across different test sets and tolerances.

Complexity theory

Mostly based on

- H Lu, J Yang (2024) "Restarted Halpern PDHG for Linear Programming".

KKT system of LP

Denote

$$K = \begin{pmatrix} A & 0 \\ -A & 0 \\ 0 & -A^\top \\ -c^\top & b^\top \end{pmatrix} \quad \text{and} \quad h = \begin{pmatrix} b \\ -b \\ -c \\ 0 \end{pmatrix}$$

Then $Kz \geq h$ is the KKT system of LP and solutions $Z^* = \{z \mid Kz \geq h\}$

Progress metric: KKT residual of standard LP

$$\mathrm{KKT}(z) = \|(h - Kz)^+\| = \left\| \begin{pmatrix} Ax - b \\ (A^\top y - c)^+ \\ (c^\top x - b^\top y)^+ \end{pmatrix} \right\|$$

- $z = (x, y)$ with $x \geq 0$

Existing Results for PDHG on LP

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

$\Downarrow$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

$\Downarrow$

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

$\Downarrow$

r^{2}HPDHG

A factor of 2

Traditional convergence results for PDHG on LP are mostly sublinear

- PDHG finds a solution z s.t. $\mathrm{KKT}(z) \leq \epsilon$ within $O(1/\epsilon)$ iterations

Many LP users require high accuracy solutions

- We need linearly convergent algorithms

Linear Convergence of PDHG for LP

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

⇩

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

⇩

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

⇩

r^{2}HPDHG

A factor of 2

Definition: Sharpness of the KKT System

α is the sharpness constant of the KKT system, if for any $z = (x, y), x \geq 0$,

$$\alpha \mathrm{dist}(z, Z^*) \leq \|(h - Kz)^+\| \ .$$

Linear Convergence of PDHG for LP

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

⇓

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log \frac{1}{\epsilon}\right)$

⇓

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log \frac{1}{\epsilon}\right)$

⇓

r²HPDHG

A factor of 2

Definition: Sharpness of the KKT System

α is the sharpness constant of the KKT system, if for any $z = (x, y), x \geq 0$,

$$\alpha \mathrm{dist}(z, Z^*) \leq \|(h - Kz)^+\| \ .$$

Theorem (informal) [Lu-Yang, 2022]: Linear convergence of PDHG

Consider LP in primal-dual form:
$\min_{x \geq 0} \max_y \ c^\top x + y^\top b - y^\top Ax$. Then PDHG finds a solution z such that $\mathrm{KKT}(z) \leq \epsilon$ within

$$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\left(\frac{1}{\epsilon}\right)\right)$$

iterations.

Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

r²HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

↓

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

↓

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

↓

r²HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

z^0

Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

↓

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

↓

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

↓

r²HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

z^0 •

• $\mathrm{PDHG}(z^0)$

Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

⇩

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

⇩

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

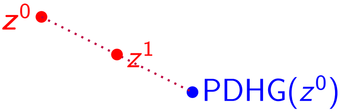
⇩

r²HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

$\Downarrow$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

$\Downarrow$

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

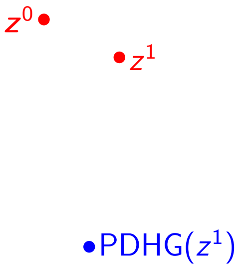
$\Downarrow$

r^{2}HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

rHPDHG

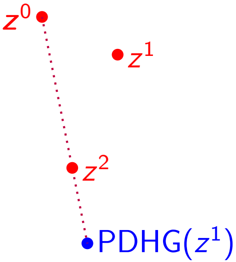
$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

r²HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

rHPDHG

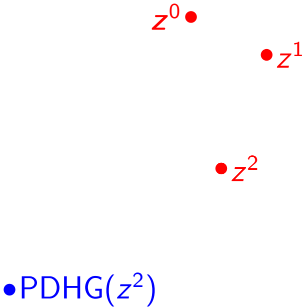
$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

r²HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2} z^0$$



Halpern PDHG (HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

rHPDHG

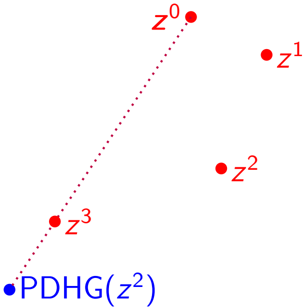
$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

r^2HPDHG

A factor of 2

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



Restarted Halpern PDHG (rHPDHG)

PDHG
$O\left(\frac{\|A\|}{\epsilon}\right)$

$\Downarrow$

PDHG + sharp
$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

$\Downarrow$

rHPDHG
$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

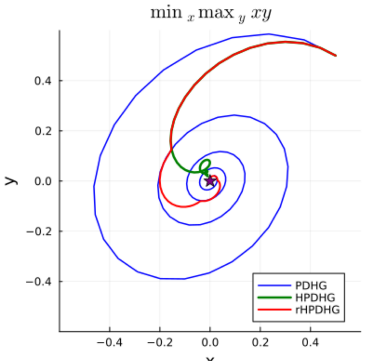
$\Downarrow$

r²HPDHG
A factor of 2

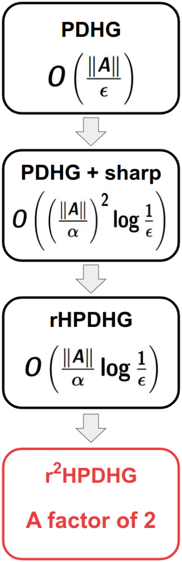
Restarted Halpern PDHG (rHPDHG)

---

initialize the inner loop. inner loop counter $t \leftarrow 0$;
repeat
$\quad z^{n,t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^{n,t}) + \frac{1}{t+2} z^{n,0}$;
until $\|z^{n,t+1} - \mathrm{PDHG}(z^{n,t+1})\| \leq \|z^{n,0} - \mathrm{PDHG}(z^{n,0})\|/2$;
restart the outer loop. $z^{n+1,0} \leftarrow \mathrm{PDHG}(z^{n,t+1})$;



Restarted reflected Halpern PDHG (r^2HPDHG)



Reflected Halpern PDHG

$$z^{k+1} = \frac{k+1}{k+2}(2\mathrm{PDHG}(z^k) - z^k) + \frac{1}{k+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically

Restarted reflected Halpern PDHG ($\mathrm{r^2HPDHG}$)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

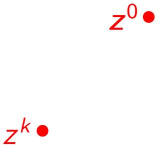
r²HPDHG

A factor of 2

Reflected Halpern PDHG

$$z^{k+1} = \frac{k+1}{k+2}(2\mathrm{PDHG}(z^k) - z^k) + \frac{1}{k+2} z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



Restarted reflected Halpern PDHG (r^{2}HPDHG)

PDHG

$O\left(\frac{\|A\|}{\epsilon}\right)$

⇩

PDHG + sharp

$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

⇩

rHPDHG

$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

⇩

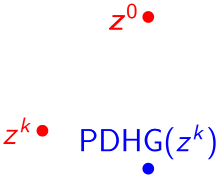
r^{2}HPDHG

A factor of 2

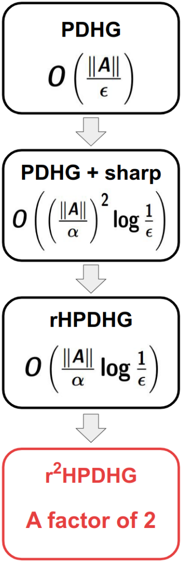
Reflected Halpern PDHG

$$z^{k+1} = \frac{k+1}{k+2}(2\mathrm{PDHG}(z^k) - z^k) + \frac{1}{k+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



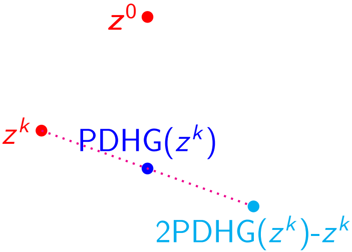
Restarted reflected Halpern PDHG (r^2HPDHG)



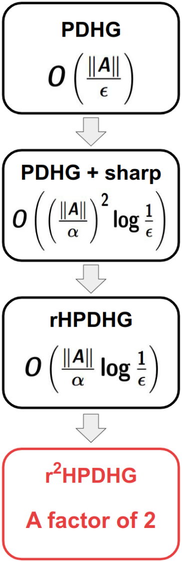
Reflected Halpern PDHG

$$z^{k+1} = \frac{k+1}{k+2}\left(2\mathrm{PDHG}(z^k) - z^k\right) + \frac{1}{k+2} z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



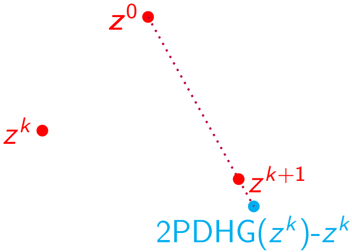
Restarted reflected Halpern PDHG (r^{2}HPDHG)



Reflected Halpern PDHG

$$z^{k+1} = \frac{k+1}{k+2}(2\mathrm{PDHG}(z^k) - z^k) + \frac{1}{k+2} z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



Quadratic Programming

Mostly based on

- H Lu, J Yang (2023b) "A Practical and Optimal First-Order Method for Large-Scale Convex Quadratic Programming".

Quadratic Programming

- QP (primal-dual form):

$$\min_{x} \max_{y \geq 0} L(x, y) := \frac{1}{2} x^\top Qx + c^\top x - y^\top b + y^\top Ax$$

- Major Solvers for QP

Solver	Gurobi	MOSEK	SCS/OSQP	PDLP
Method	Simplex/IPM	IPM	ADMM[1]	PDHG

- Major algorithms:
  - Simplex Methods
  - Interior-Point Methods (IPM)
  - First-Order Methods (FOM)
    - Alternating Direction Method of Multipliers (ADMM)
    - Primal-Dual Hybrid Gradient (PDHG)

[1] Support direct/indirect linear solvers

Two Extremes of QP

Quadratic Programming

$$\min \frac{1}{2}x^\top Qx + c^\top x$$
$$\text{s.t. } Ax \le b$$

Two Extremes of QP

Quadratic Programming

$$\min \frac{1}{2}x^\top Qx + c^\top x \quad \text{s.t. } Ax \le b$$

Linear Programming

$$\min c^\top x \quad \text{s.t. } Ax \le b$$

Unconstrained QP

$$\min \frac{1}{2}x^\top Qx + c^\top x$$

Two Extremes of QP

Quadratic Programming

$$\begin{aligned} \min\ & \frac{1}{2}x^\top Qx + c^\top x \\ \text{s.t.}\ & Ax \leq b \end{aligned}$$

Linear Programming

$$\begin{aligned} \min\ & c^\top x \\ \text{s.t.}\ & Ax \leq b \end{aligned}$$

restart

Unconstrained QP

$$\min\ \frac{1}{2}x^\top Qx + c^\top x$$

momentum

Two Extremes of QP

Quadratic Programming

$$\min \ \frac{1}{2}x^\top Qx + c^\top x$$
$$\text{s.t. } Ax \le b$$

Linear Programming

$$\min \ c^\top x$$
$$\text{s.t. } Ax \le b$$

Unconstrained QP

$$\min \ \frac{1}{2}x^\top Qx + c^\top x$$

restart

momentum

- Optimal FOM should combine restart and momentum

Two-Loop Restarted Accelerated PDHG

Algorithm: Restarted Accelerated PDHG

Input: Initial point $(x^{0,0}, y^{0,0})$, parameters $\{(\beta_t, \theta_t, \eta_t, \tau_t)\}$;
repeat
  initialize the inner loop. inner loop counter $t \leftarrow 0$;
  repeat
    $x_{\mathrm{md}}^{n,t} \leftarrow (1 - \beta_t^{-1})\bar{x}^{n,t} + \beta_t^{-1} x^{n,t}$;
    $y^{n,t+1} \leftarrow \mathrm{Proj}_{\mathbb{R}_+^m} \left\{ y^{n,t} + \tau_t (A(\theta_t(x^{n,t} - x^{n,t-1}) + x^{n,t}) - b) \right\}$;
    $x^{n,t+1} \leftarrow x^{n,t} - \eta_t (Q x_{\mathrm{md}}^{n,t} + c + A^\top y^{n,t+1})$;
    $\bar{x}^{n,t+1} \leftarrow (1 - \beta_t^{-1})\bar{x}^{n,t} + \beta_t^{-1} x^{n,t+1}$;
    $\bar{y}^{n,t+1} \leftarrow (1 - \beta_t^{-1})\bar{y}^{n,t} + \beta_t^{-1} y^{n,t+1}$;
  until a restart condition holds;
  restart the outer loop. $(x^{n+1,0}, y^{n+1,0}) \leftarrow (\bar{x}^{n,t}, \bar{y}^{n,t})$, $n \leftarrow n + 1$;
until $(x^{n,0}, y^{n,0})$ *converges*;
Output: $(x^{n,0}, y^{n,0})$.

- $\beta_t = 1 + t/2$ is the momentum parameter, $\theta_t = t/(t+1)$ is the over-relaxation parameter, η_t and τ_t are the primal and dual step-sizes
- Sublinear rate of accelerated PDHG was studied in [Chen et al., 2014]

Numerical Experiment

Solvers:

- PDQP: CPU / GPU, written in Julia
- SCS: CPU-direct / CPU-indirect / GPU, written in C
- OSQP: CPU, written in C

Datasets:

- Convex QP instances from QPLIB (33 “tiny” instances in total)
- 63 synthetic instances generated from the code of OSQP paper

Termination (1h time limit):

- PDQP has a nearly identical termination criteria as SCS
- OSQP has a much looser criteria by neglecting primal-dual gap and using ℓ_2 norm

Convex QP, Large Instances

- Seven synthetic classes of convex QP problems from OSQP paper
- Small (300k nnz), medium (3m nnz), large (30m nnz)

	Small (21)		Medium (21)		Large (21)		Total (63)	
	Count	Time	Count	Time	Count	Time	Count	Time
PDQP (GPU)	21	1.20	21	1.94	21	6.13	63	2.92
PDQP (CPU)	21	3.01	21	27.53	18	359.31	60	46.49
SCS (GPU)	21	2.47	21	10.02	21	68.54	63	16.97
SCS (CPU-indirect)	21	9.39	21	81.86	15	700.88	57	98.18
SCS (CPU-direct)	21	1.06	21	10.19	21	133.52	63	21.76
OSQP (CPU)	21	1.11	21	11.80	21	170.71	63	25.24

Table 7: Solve time in seconds and SGM10 of different solvers on instances of with tolerance 10^{-3}.

	Small (21)		Medium (21)		Large (21)		Total (63)	
	Count	Time	Count	Time	Count	Time	Count	Time
PDQP (GPU)	21	2.08	21	3.40	21	12.17	63	5.31
PDQP (CPU)	21	5.55	21	54.18	17	647.42	59	76.89
SCS (GPU)	21	5.69	21	24.38	18	196.75	60	38.14
SCS (CPU-indirect)	21	23.90	20	267.27	12	1593.95	53	237.04
SCS (CPU-direct)	21	3.09	21	30.25	20	395.85	62	49.79
OSQP (CPU)	21	3.31	21	30.70	21	375.24	63	49.32

Table 8: Solve time in seconds and SGM10 of different solvers on instances of with tolerance 10^{-6}.

Convergence Guarantee (Upper Bound)

Theorem (informal) [Lu-Yang, 2023b]: Convergence Rate of Restarted Accelerated PDHG

Consider QP in primal-dual form:

$$\min_{x} \max_{y \geq 0} \ L(x, y) := \frac{1}{2} x^\top Q x + c^\top x - y^\top b + y^\top A x \ .$$

Then restarted accelerated PDHG finds a solution z with $\mathrm{dist}(z, \mathcal{Z}^*) \leq \epsilon$ within

$$O\left(\max\left\{ \sqrt{\frac{\|Q\|}{\alpha_\xi}}, \frac{\|A\|}{\alpha_\xi} \right\} \log \frac{1}{\epsilon} \right)$$

iterations.

$\alpha_\xi > 0$ is the quadratic growth constant of the smoothed gap in QP

- LP: α_ξ recovers the sharpness constant of LP
- Unconstrained QP: α_ξ recovers the minimum positive singular value of the quadratic term

Convergence Guarantee (Lower Bound)

Restarted accelerated PDHG achieves optimal linear rate under LP and unconstrained QP [Applegate-Hinder-**L**-Lubin, 2021], [Nesterov, 1983]

Problem:	LP	Unconstrained QP
Upper bound:	$O\left(\frac{\|A\|_2}{\alpha}\log\frac{1}{\epsilon}\right)$	$O\left(\sqrt{\frac{\|Q\|_2}{\sigma_{\min}^{+}(Q)}}\log\frac{1}{\epsilon}\right)$
Lower bound:	$\Omega\left(\frac{\|A\|_2}{\alpha}\log\frac{1}{\epsilon}\right)$	$\Omega\left(\sqrt{\frac{\|Q\|_2}{\sigma_{\min}^{+}(Q)}}\log\frac{1}{\epsilon}\right)$

- α is the sharpness constant of LP
- $\sigma_{\min}^{+}(Q)$ is the minimum positive singular value of Q

Semidefinite Programming

SDP: MaxCut

We are building up a new GPU-based SDP solver

- The methodology is based on an augmented Lagrangian method
- The numerical performance is suprisingly promising (see the next two slides)
- If you have any large-scale SDP problems to be solved, please feel free to contact us!

SDP: MaxCut

instance	dimension	time
NACA0015	1039183	11.485
delaunay_n20	1048576	7.333
kron_g500-logn20	1048576	131.911
rgg_n_2_20_s0	1048576	9.888
belgium_osm	1441295	8.283
delaunay_n21	2097152	15.262
kron_g500-logn21	2097152	335.199
rgg_n_2_21_s0	2097152	21.679
packing-500x100x100-b050	2145852	26.731
netherlands_osm	2216688	12.081
M6	3501776	46.567
333SP	3712815	40.248
AS365	3799275	49.985
venturiLevel3	4026819	41.529
NLR	4163763	51.739
delaunay_n22	4194304	33.287
rgg_n_2_22_s0	4194304	47.507
hugetrace-00000	4588484	50.669
channel-500x100x100-b050	4802000	80.858

instance	dimension	time
hugetric-00000	5824554	77.006
hugetric-00010	6592765	79.448
italy_osm	6686493	40.709
adaptive	6815744	96.847
hugetric-00020	7122792	83.734
great-britain_osm	7733822	61.270
delaunay_n23	8388608	67.985
rgg_n_2_23_s0	8388608	164.510
germany_osm	11548845	94.783
asia_osm	11950757	108.128
hugetrace-00010	12057441	186.480
road_central	14081816	174.989
hugetrace-00020	16002413	314.768
delaunay_n24	16777216	189.024
rgg_n_2_24_s0	16777216	412.733
hugebubbles-00000	18318143	387.761
hugebubbles-00010	19458087	280.580
hugebubbles-00020	21198119	308.113
road_usa	23947347	308.473

Table 1: Performance of ALORA on MaxCut instances. Solve time in seconds.

SDP: matrix completion

$r = 3$			$r = 5$		
n	m	time	n	m	time
10000	828659	0.379	10000	2300917	0.504
20000	1782453	0.726	20000	4952616	0.969
50000	4868248	1.409	50000	13522024	3.025
100000	10361604	2.944	100000	28777073	6.762
200000	21961921	5.511	200000	61013229	14.786
350000	40223331	12.759	350000	111700922	28.282

Table 2: Performance of ALORA on matrix completion with varying r, n, and m. Solve time in seconds.

MPAX: Mathematical Programming in JAX

Mostly based on

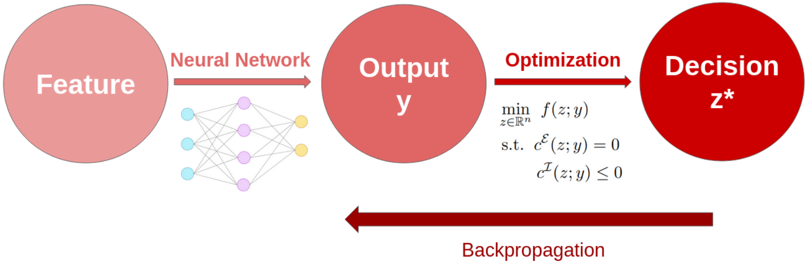
- H Lu, Z Peng, J Yang (2024), “MPAX: Mathematical Programming in JAX”.
- GitHub Repository: `https://github.com/MIT-Lu-Lab/mpax`

MPAX: Math Programming in JAX

MPAX (**M**ath **P**rogramming in J**AX**) is a hardware-accelerated, differentiable, batchable, and distributable solver for mathematical programming in JAX:

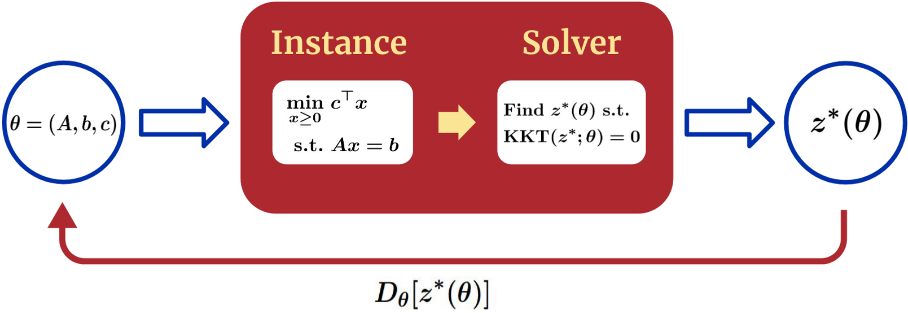
- **Hardware-accelerated**: executes on multiple architectures including CPUs, GPUs and TPUs
- **Differentiable**: easily computes derivatives of solutions with respect to inputs through implicit or unrolled differentiation
- **Batchable**: solves multiple problem instances simultaneously
- **Distributed**: executes distributedly across multiple devices, such as multiple GPUs

Example application: end-to-end decision making



- Rather than the traditional predict-then-optimize paradigm, end-to-end decision making optimizes jointly the prediction and optimization
- This is an actively studied research area [Amos et al., 2017]
- LP layer can serve as loss functions, enforce constraints, make decisions, etc
- Applications in robotics, control, reinforcement learning, video games, AI for science, etc

Differentiable layer

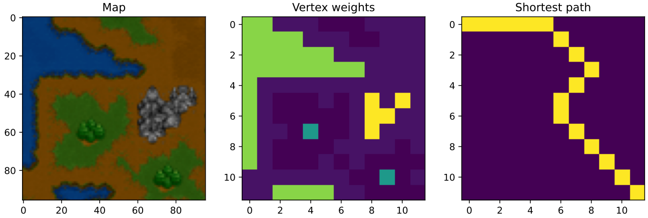


The key is to efficiently compute or approximate $D_\theta[z^*(\theta)]$

- Approximated differentiation
  - Smart Predict-then-Optimize loss
  - Perturbed Fenchel-Young loss
- Auto-differentiation
  - The high-level idea is to unroll the PDLP solver and compute the gradient via the chain rule

Warcraft Shortest Path

Task: find the shortest path between the top left and the bottom right vertices given the Warcraft map.



End-to-end predict-then-optimize

- Use the first five layers of ResNet18 to predict the costs for each vertice.
- Solve the LP to find the shortest path.
- Compute the Smart-Predict-then-Optimize+ loss and backpropagate to update the weights.

MPAX versus PyEPO (Gurobi)

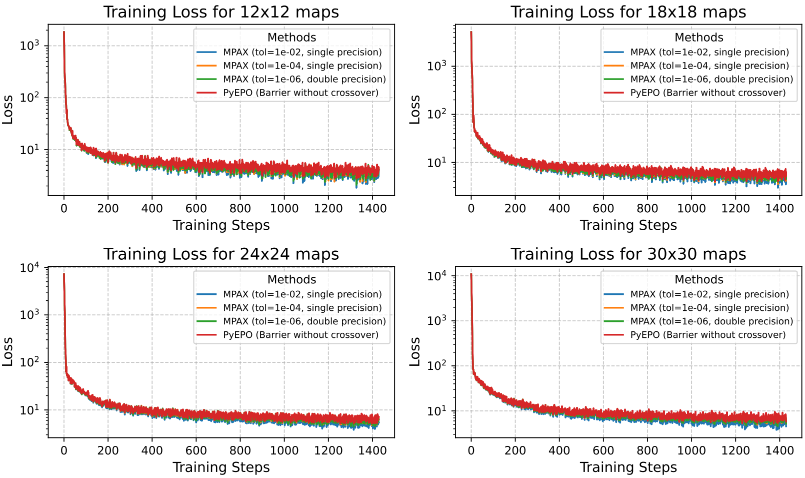
Experiment details

- Training: 10,000 samples, batch size = 70, epochs = 10.
- Gurobi runs on CPU with 16 cores and 256GB of memory.
- PyTorch, FLAX, and MPAX run on A100 GPU with 80GB memory.

Methods	Configuration		Training time per epoch			
			k=12	k=18	k=24	k=30
FLAX+MPAX	single precision	tol=1e-2	17.56	31.86	55	94.39
		tol=1e-3	24.83	44.98	78.72	130.8
		tol=1e-4	33.37	55.85	99.56	170.76
	double precision	tol=1e-6	32.07	71.17	127.99	210.44
PyEPO (PyTorch+Gurobi)	Method=automatic	Process=1	178.08	427.27	792.85	1273.24
		Process=4	80.74	226.18	513.78	1034.65
		Process=8	40.16	108	245.8	480.1
		Process=16	27.21	70.82	159.25	317.86
	Method=barrier (crossover disabled)	Process=1	202.62	474.67	856.1	1383.03
		Process=4	85.48	237.69	538.62	1067.9
		Process=8	42.65	115.89	261.95	506.06
		Process=16	29.97	77.7	170.05	337.95

k denotes the number of rows and columns in the map.

Training loss with different tolerances



- The quality of 10^{-2} accuracy solutions is enough for this application.

Discussions

cuPDLP demonstrates the power of GPU in solving LPs

- It demonstrates an alternative, not a substitute
- Better tuning and implementation led to 3-4 times further speedup

Is PDHG/PDLP the first-order method for LP?

- Unlikely, but it is the best among what we tried

A paradigm shift from CPU to GPU?

- Likely, in the next decade

Many future directions to be explored:

- Other optimization problems, SOCP? NLP? MIP?
- Multiple GPUs implementation
- ...

Thank you!

Additional Slides

Why $2x^{k+1} - x^k$ in PDHG (informal)?

Denote $z = (x, y)$, and $F(z) = [\nabla_x L(x, y), -\nabla_y L(x, y)]$.

Why $2x^{k+1} - x^k$ in PDHG (informal)?

Denote $z = (x, y)$, and $F(z) = [\nabla_x L(x, y), -\nabla_y L(x, y)]$. Then we have

$$(\text{GDA}) : z^{k+1} = z^k - \eta F(z^k)$$

$$(\text{PPM}) : z^{k+1} = z^k - \eta F(z^{k+1})$$

- Implicit algorithm (PPM) is more stable than explicit algorithm (GDA) due to a high-order effect [**L**, 2022]

Why $2x^{k+1} - x^k$ in PDHG (informal)?

Denote $z = (x, y)$, and $F(z) = [\nabla_x L(x, y), -\nabla_y L(x, y)]$. Then we have

$$(\text{GDA}) : z^{k+1} = z^k - \eta F(z^k)$$

$$(\text{PPM}) : z^{k+1} = z^k - \eta F(z^{k+1})$$

- Implicit algorithm (PPM) is more stable than explicit algorithm (GDA) due to a high-order effect [**L**, 2022]

It turns out we can rewrite the update of PDHG as

$$(\text{PDHG}) : z^{k+1} = z^k - P^{-1} F(z^{k+1}) \ ,$$

where $P = \begin{bmatrix} \frac{1}{\eta} I & A^T \\ A & \frac{1}{\eta} I \end{bmatrix}$.

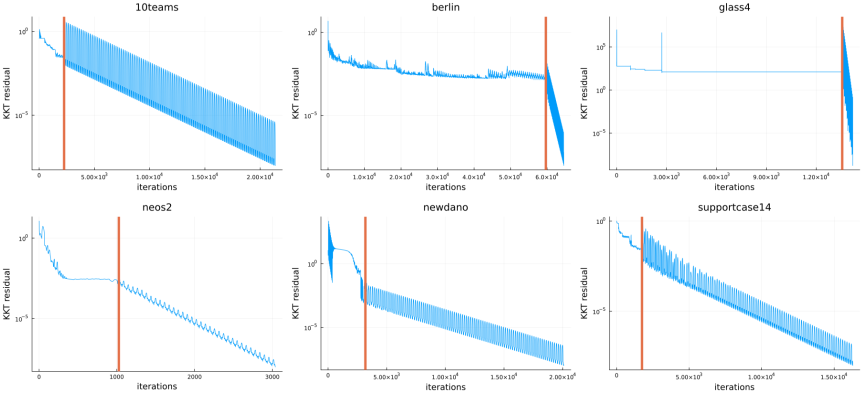
- [Lu-Yang, 2023d] presents a unified and simplified convergence analysis of PPM, PDHG and ADMM

Trajectory-based Analysis

Mostly based on

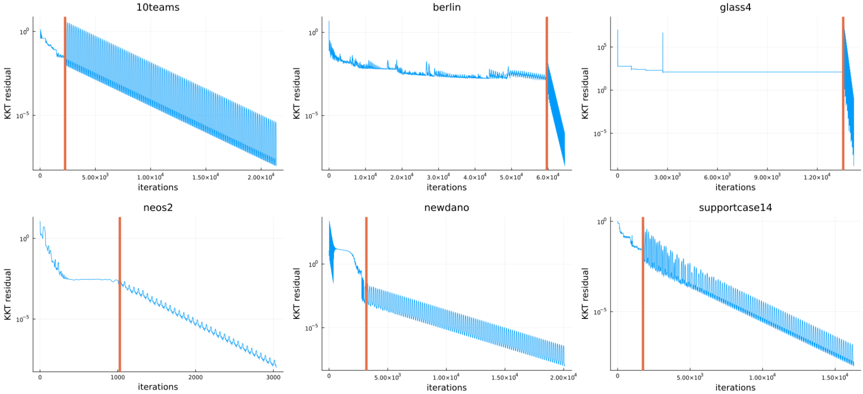
- H Lu, J Yang (2023c) "On the Geometry and Refined Rate of Primal-Dual Hybrid Gradient for Linear Programming".
- H Lu, J Yang (2024a) "Restarted Halpern PDHG for Linear Programming".

Typical Behaviors of PDHG for LP



- There are two stages of convergence
- Slow initial convergence, then fast linear convergence

Two Questions

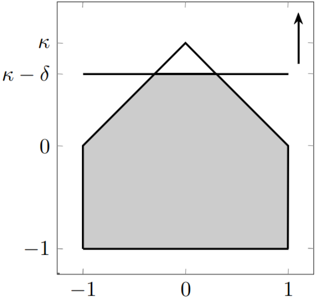


- What are the geometric driving forces of the two stages?
- Is it possible to derive a refined convergence complexity?

A Simple yet Representative Example

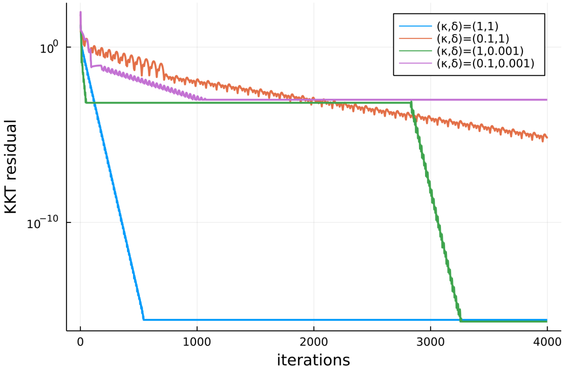
Consider a class of 2-D dual LP with parameter (κ, δ): $\max_y \; b^\top y, \text{ s.t. } A^\top y \le c$

$$
\begin{aligned}
\max \;& y_2 \\
\text{s.t. } & y_1 \ge -1 \\
& y_1 \le 1 \\
& y_1 + \frac{1}{\kappa} y_2 \le 1 \\
& -y_1 + \frac{1}{\kappa} y_2 \le 1 \\
& y_2 \le \kappa - \delta
\end{aligned}
$$



- κ controls the condition number of the matrix A
- δ controls closeness to degeneracy

A Simple yet Representative Example

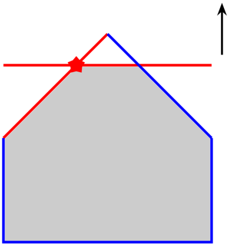


- δ small: slow first stage
- κ small: slow second stage

What is going on in the two stages? (Informal)

Stage 1: Slow initial convergence

- This is the process to identify active basis set S such that $x_S^* > 0$
- Driving force of this stage is closeness to degeneracy (i.e., δ)



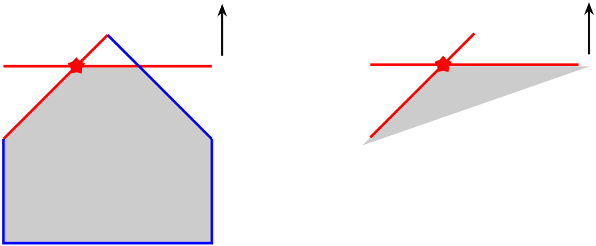
What is going on in the two stages? (Informal)

Stage 1: Slow initial convergence

- This is the process to identify active basis set S such that $x_S^* > 0$
- Driving force of this stage is closeness to degeneracy (i.e., δ)

Stage 2: "Fast" eventual convergence

- Once active set is fixed, the dynamic has faster linear convergence
- Driving force of this stage is condition number of matrix A_S (i.e., $\|A_S\|/\alpha_S$)
- $\alpha_S \approx \sigma^+_{\min}(A_S) \gg \alpha$



Refined Complexity (Informal)

Theorem [Lu-Yang, 2024a]: Two-Stage Convergence of $\mathrm{r^2HPDHG}$ for LP

Consider $\mathrm{r^2HPDHG}$ for solving LP, then we have

- **(Finite time identification)** $\mathrm{r^2HPDHG}$ will identify the non-degenerate active variables in
$$k \geq K := \mathcal{O}\left(\frac{\|A\|/\alpha_S}{\delta}\right)$$
iterations.

Refined Complexity (Informal)

Theorem [Lu-Yang, 2024a]: Two-Stage Convergence of $\mathrm{r^2HPDHG}$ for LP

Consider $\mathrm{r^2HPDHG}$ for solving LP, then we have

- **(Finite time identification)** $\mathrm{r^2HPDHG}$ will identify the non-degenerate active variables in

$$k \geq K := \mathcal{O}\left(\frac{\|A\|/\alpha_S}{\delta}\right)$$

iterations.

- **(Linear convergence after identification)** After identification, $\mathrm{r^2HPDHG}$ will compute a solution z such that $\mathrm{KKT}(z) \leq \epsilon$ in

$$\mathcal{O}\left(\frac{||A_S||}{\alpha_S}\log\left(\frac{1}{\epsilon}\right)\right) .$$

iterations.

Refined Complexity (Informal)

Theorem [Lu-Yang, 2024a]: Two-Stage Convergence of $\mathrm{r^2HPDHG}$ for LP

Consider $\mathrm{r^2HPDHG}$ for solving LP, then we have

- **(Finite time identification)** $\mathrm{r^2HPDHG}$ will identify the non-degenerate active variables in
$$k \geq K := \mathcal{O}\left(\frac{\|A\|/\alpha_S}{\delta}\right)$$
iterations.
- **(Linear convergence after identification)** After identification, $\mathrm{r^2HPDHG}$ will compute a solution z such that $\mathrm{KKT}(z) \leq \epsilon$ in
$$\mathcal{O}\left(\frac{\|A_S\|}{\alpha_S}\log\left(\frac{1}{\epsilon}\right)\right) .$$
iterations.

- The fundamental difficulty of the analysis comes from "degeneracy" of the LP