MadNLP: nonlinear programming on GPUs

François Pacaud
Joint work with: Sungho Shin, Alexis Montoison, and Mihai Anitescu

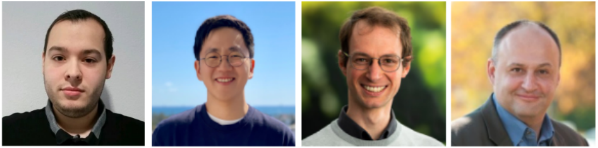
CAS, Mines Paris - PSL

October 29th, 2024
Julia & Optimization Days

MINES PARIS | PSL

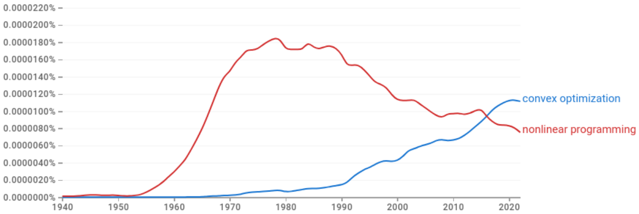
Who are we?

An international team looking at the future of nonlinear programming



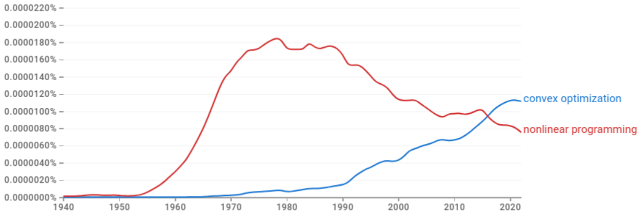
The sad truth...

Nonlinear programming has fallen out of fashion :-(



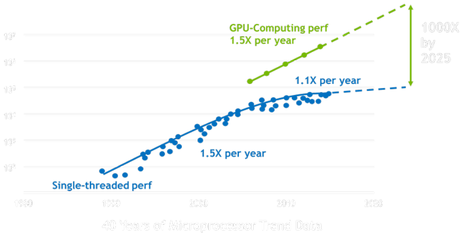
The sad truth...

Nonlinear programming has fallen out of fashion :-(



... but an open-door for new opportunity!

Can we make nonlinear programming great again using modern hardware?



Source of the figure: NVIDIA

MadNLP: a structure exploiting interior-point solver

Winner of the 2023 COIN-OR cup!

MadNLP


```
using MadNLP, MadNLPTests
model = MadNLPTests.HS15Model()
solver = MadNLPSolver(model)
MadNLP.solve!(solver)
```

Fork on github!

https://github.com/MadNLP/MadNLP.jl/

https://github.com/exanauts/ExaModels.jl

MadNLP

- Written in pure Julia
- Filter line-search (ala Ipopt)
- Flexible & Modular

✓ CUDA-compatible

✓ MPI-compatible

✓ Interfaced with the vectorized modeler ExaModels.jl

✓ And now interfaced with Casadi, thanks to Tommaso Sartor!

Building extensively on the Julia ecosystem

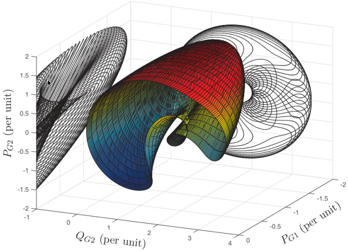
GPU-premium

- CUDA.jl
- CUDSS.jl

Optimization-premium

- JuMP.jl
- NLPModels.jl & JuliaSmoothOptimizers

Nonlinear programming: a reminder



n variables, m inequality constraints, p equality constraints

Continuous nonlinear problems

Objective

Equality cons.

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) \leq 0 \end{cases}$$

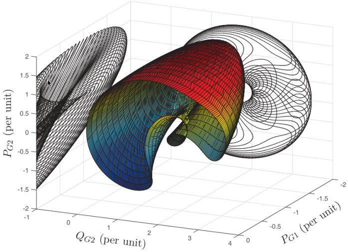
Inequality cons.

The functions f, g, h are smooth, *possibly nonconvex*

- Useful framework to solve practical engineering problems
- Usually, we are interested only at finding a *local optimum*
- Mature solvers exist since the 2000s (Ipopt, Knitro, LOQO)

J. Nocedal, SJ. Wright. Numerical optimization.

Nonlinear programming: a reminder



n variables, m inequality constraints, p equality constraints

Continuous nonlinear problems

Objective → $f(x)$; Equality cons. → $g(x) = 0$; Slack → s, $s \geq 0$

$$\min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) + s = 0\,, \quad s \geq 0 \end{cases}$$

The functions f, g, h are smooth, *possibly nonconvex*

- Useful framework to solve practical engineering problems
- Usually, we are interested only at finding a *local optimum*
- Mature solvers exist since the 2000s (Ipopt, Knitro, LOQO)

J. Nocedal, SJ. Wright. Numerical optimization.

Nonlinear Optimization Software: State-of-the-Art on CPU

Problem Formulation

$$\min_{x \geq 0} f(x)$$

$$s.t.\ c(x) = 0$$

Nonlinear Optimization Software: State-of-the-Art on CPU

Problem Formulation

$$\min_{x \geq 0} f(x)$$
$$s.t.\ c(x) = 0$$

- Classical nonlinear programming
  - the objective and constraints are **smooth**
  - **large number of variables and constraints**
  - the problem is **highly sparse**.

Nonlinear Optimization Software: State-of-the-Art on CPU

Problem Formulation

$$\min_{x \geq 0} f(x)$$
$$s.t.\ c(x) = 0$$

Newton's Step Computation

$$\underbrace{\begin{bmatrix} W & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} p^x \\ p^\lambda \end{bmatrix}}$$

"KKT System" (ill-conditioned)

Line-Search

$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

- Classical nonlinear programming
  - the objective and constraints are **smooth**
  - **large number of variables and constraints**
  - the problem is **highly sparse**.
- Interior-point methods
  - Inequalities $x \geq 0$ replaced by smooth log-barrier functions $f(x) - \mu \sum_i \log(x[i])$.
  - **Newton's Step** is computed by solving a "**KKT system**" (large, sparse, symmetric indefinite, ill-conditioned system).
  - Line-search (along with several additional heuristics) ensures **global convergence**.

Nonlinear Optimization Software: State-of-the-Art on CPU

Problem Formulation

$$\min_{x \geq 0} f(x)$$
$$s.t.\ c(x) = 0$$

Newton's Step Computation

$$\underbrace{\begin{bmatrix} W & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} p^x \\ p^\lambda \end{bmatrix}}$$

"KKT System" (ill-conditioned)

Line-Search

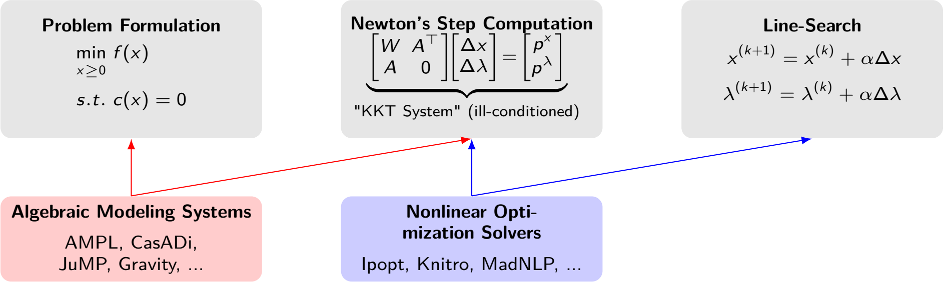
$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

Algebraic Modeling Systems

AMPL, CasADi,
JuMP, Gravity, ...

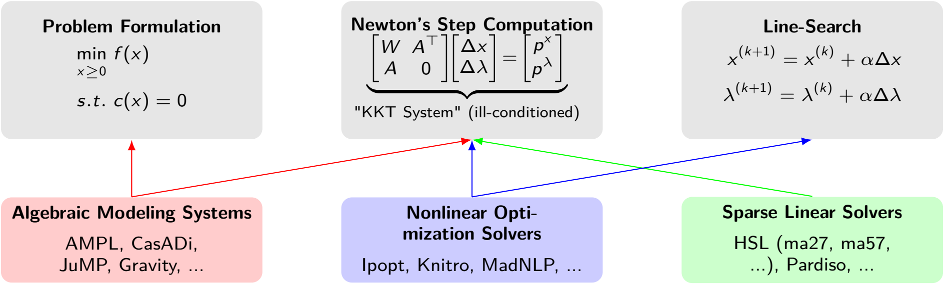
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.

Nonlinear Optimization Software: State-of-the-Art on CPU



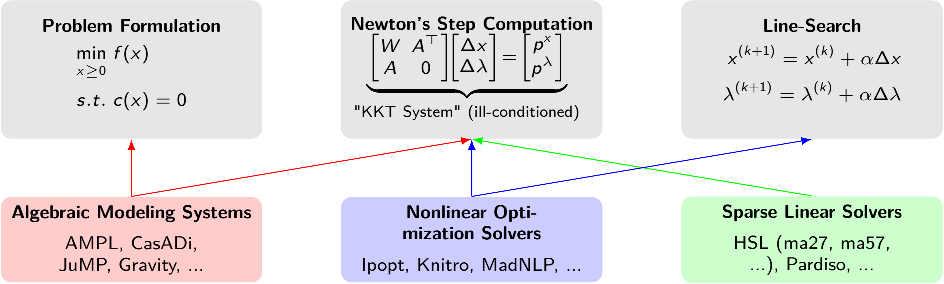
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.
- Nonlinear optimization solvers apply iterations of optimization algorithms.

Nonlinear Optimization Software: State-of-the-Art on CPU



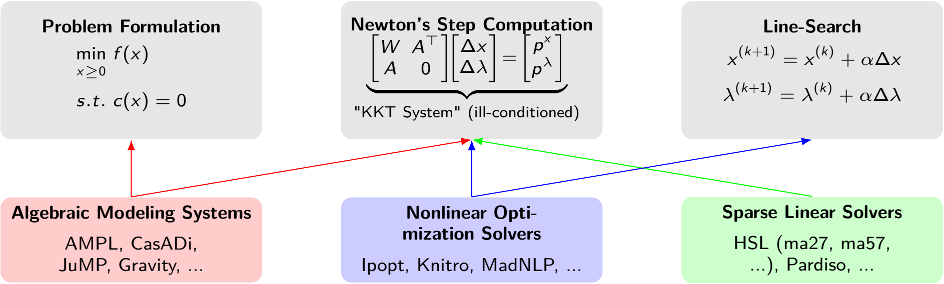
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.
- Nonlinear optimization solvers apply iterations of optimization algorithms.
- Sparse linear solvers solves KKT systems using **sparse matrix factorization**.

Nonlinear Optimization Software: State-of-the-Art on CPU



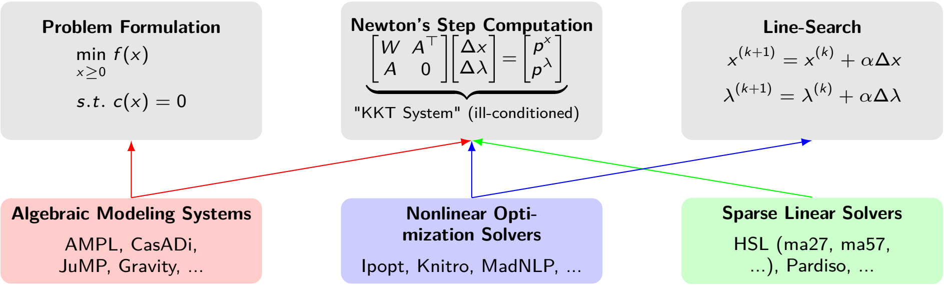
- These software tools have enabled the success of nonlinear optimization on CPUs

Nonlinear Optimization Software: State-of-the-Art on CPU



- These software tools have enabled the success of nonlinear optimization on CPUs
- Many software tools have been developed in 1990s-2000s (**heavily optimized for CPUs**)

Nonlinear Optimization Software: State-of-the-Art on CPU



- These software tools have enabled the success of nonlinear optimization on CPUs
- Many software tools have been developed in 1990s-2000s (**heavily optimized for CPUs**)
- Now we need **GPU-equivalent** of these tools:
  - Algebraic Modeling: **ExaModels.jl**
  - Optimization solver: **MadNLP.jl**
  - Sparse Linear Solvers: **NVIDIA cuDSS** (Cholesky & LDL)

Identifying the computational bottlenecks in IPM

1. **Evaluate derivatives** ∇F_μ
    - **Sparse** Automatic differentiation
    - Algebraic modeling systems (AMPL, JuMP, Casadi,...)
2. **Solve KKT system** $\nabla F_\mu d^k = -F_k$
    - Symmetric indefinite system
    - Efficient sparse linear solvers exist (HSL ma27/ma57, Pardiso, Mumps,...)

First step: Sparse automatic differentiation on GPU with ExaModels.jl

- Large-scale optimization problems **almost always have repetitive patterns**

$$\min_{x^\flat \le x \le x^\sharp} \sum_{l\in[L]} \sum_{i\in[I_l]} f^{(l)}(x; p_i^{(l)}) \qquad \textbf{(SIMD abstraction)}$$

$$\text{subject to } \left[g^{(m)}(x; q_j)\right]_{j\in[J_m]} + \sum_{n\in[N_m]} \sum_{k\in[K_n]} h^{(n)}(x; s_k^{(n)}) = 0, \quad \forall m \in [M]$$

- Repeated patterns are made available by always specifying the models as **iterable objects**

```
constraint(c, 3 * x[i+1]^3 + 2 * sin(x[i+2]) for i = 1:N-2)
```

- **For each repeatitive pattern**, the derivative evaluation kernel is constructed & compiled, and **executed in parallel over multiple data**

Second step: Solving the KKT system on the GPU

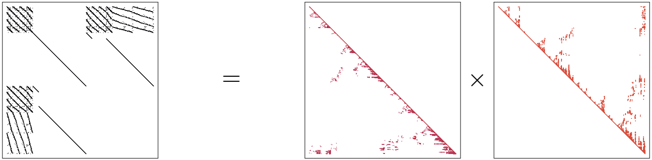


Figure: Matrix factorization using a direct solver

Linear solve: Solve the KKT system $\nabla F_\mu d_k = -F_k$

- Usually require factorizing ∇F_μ (symmetric indefinite: LBL)
- KKT system is highly *ill-conditioned* $\rightarrow$ numerical pivoting

Challenge: solving the sparse linear system on the GPU

- Ill-conditioning of the KKT system
  (= iterative solvers are often not practical)
- Direct solver requires **numerical pivoting** for stability
  (= difficult to parallelize)

B. Tasseff, C. Coffrin, A. Wächter, C. Laird. "Exploring benefits of linear solver parallelism on modern nonlinear optimization applications.", 2019

Solution : Condensation of the linear system

Solution: Condensation

- Reduce the KKT system to a sparse positive definite matrix
- Sparse Cholesky is stable without numerical pivoting
  $\rightarrow$ runs in parallel on the GPU (`cuDSS`)

S. Shin, F. Pacaud, and M. Anitescu. *Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point met*
S. Regev et al., "HyKKT: a hybrid direct-iterative method for solving KKT linear systems." Optimization Methods and Software 38, no. 2 (2023)

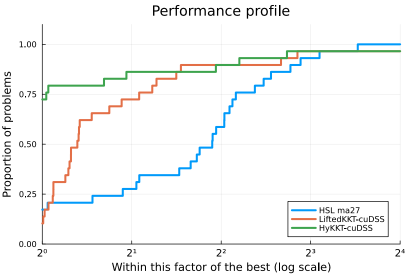
Application: AC-OPF problem

Observations

- We use the newly released `cuDSS` solver (sparse Cholesky and LDL)
- Up to 10x speed-up compared to Ipopt

	HSL MA27				LiftedKKT+cuDSS				HyKKT+cuDSS			
Case	it	init	lin	**total**	it	init	lin	**total**	it	init	lin	**total**
13659_pegase	63	0.45	7.21	**10.14**	75	0.83	1.05	**2.96**	62	0.84	0.93	**2.47**
19402_goc	69	0.63	31.71	**36.92**	73	1.42	2.28	**5.38**	69	1.44	1.93	**4.31**
20758_epigrids	51	0.63	14.27	**18.21**	53	1.34	1.05	**3.57**	51	1.35	1.55	**3.51**
78484_epigrids	102	2.57	179.29	**207.79**	101	5.94	5.62	**18.03**	104	6.29	9.01	**18.90**

Table: OPF benchmark, solved by MadNLP with a tolerance `tol=1e-6`. (A100 GPU)



Application: Nonlinear dynamic optimization

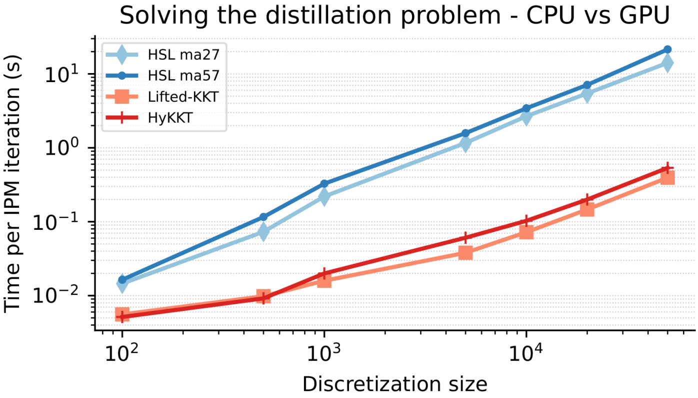


Figure: Time per iteration solve the problem to optimality (in seconds).

How expensive should be your GPU?

Benchmarking different GPUs

• A100 (80GB)	HPC ($10,000)
• A30 (24GB)	workstation ($5,000)
• A1000 (4GB)	laptop

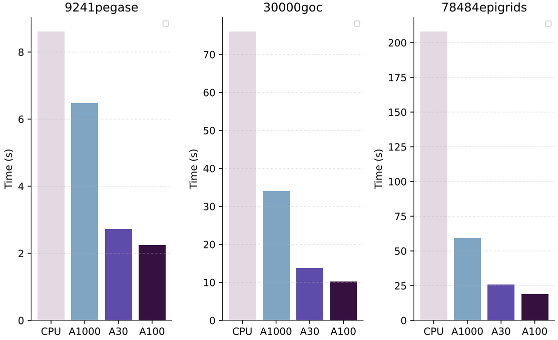


Figure: Time to solve the problem to optimality (in seconds).

What comes next?

Roadmap

- Better accuracy
  - Improve accuracy of condensed-space method
  - Support of multi-precision (`Float128`)
- Better robustness
  - Degenerate problems (e.g. optimal control with state constraints)
  - Complementarity problems (MPEC)

Want (super) fast optimization solvers?

Always looking for new collaborations!

frapac.github.io