

Disciplined Nonlinear Programming

Stephen Boyd Maximilian Schaller Daniel Cederberg

Stanford University

Based on joint work with William Zhang and Parth Nobel

Outline

Nonlinear programming

Disciplined nonlinear programming

DNLP examples

Conclusions

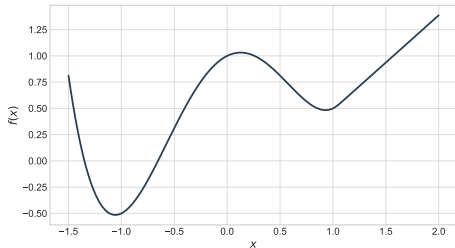
Nonlinear programming (NLP)

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & c(x) = 0 \\ & \ell \leq x \leq u\end{array}$$

- ▶ $f : \mathbf{R}^n \rightarrow \mathbf{R}$, $c : \mathbf{R}^n \rightarrow \mathbf{R}^p$ differentiable, not necessarily convex
- ▶ we seek a **local solution**

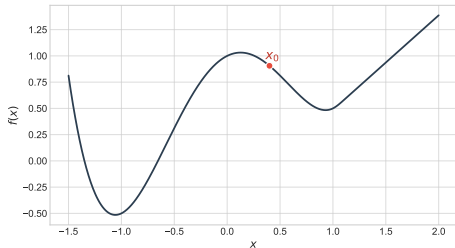
Misconception 1 about NLP

NLP solvers always compute global minimizers



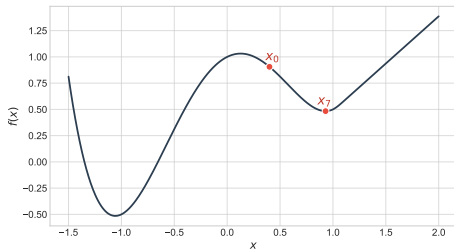
Misconception 1 about NLP

NLP solvers always compute global minimizers



Misconception 1 about NLP

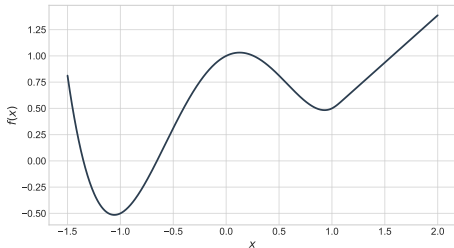
NLP solvers always compute global minimizers



- Ipopt [Wächter and Biegler, 2006] converges to a local (but not global) minimizer

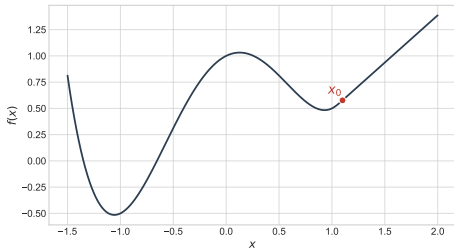
Misconception 2 about NLP

NLP solvers always converge to the local minimizer closest to the initial point



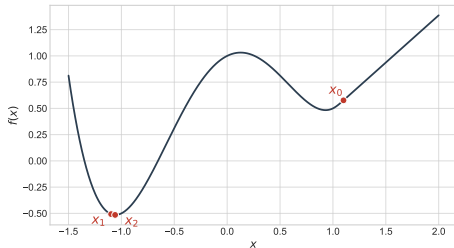
Misconception 2 about NLP

NLP solvers always converge to the local minimizer closest to the initial point



Misconception 2 about NLP

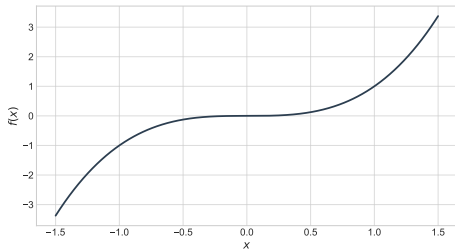
NLP solvers always converge to the local minimizer closest to the initial point



- Ipopt converges to a local minimizer, but not the one closest to the initial point

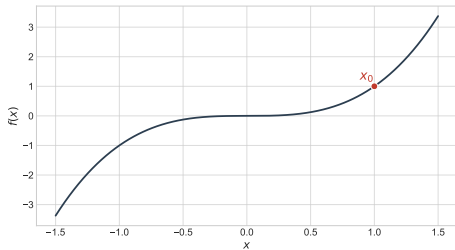
Misconception 3 about NLP

NLP solvers always converge to a local minimizer



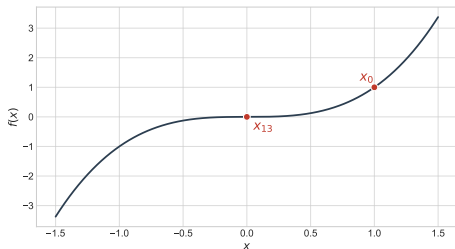
Misconception 3 about NLP

NLP solvers always converge to a local minimizer



Misconception 3 about NLP

NLP solvers always converge to a local minimizer



- Ipopt converges to a saddle point when minimizing $f(x) = x^3$ initialized at $x_0 = 1$

Misconception 4 about NLP

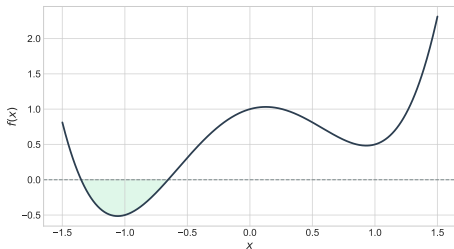
NLP solvers always converge to a 'good' feasible point (KKT point)

Misconception 4 about NLP

$$\begin{array}{ll}\text{minimize} & f(x) = x^4 - 2x^2 + 0.5x + 1 \\ \text{subject to} & f(x) \leq 0\end{array}$$

(Also see Wächter & Bowly, Gurobi, 2026)

NLP solvers always converge to a 'good' feasible point (KKT point)

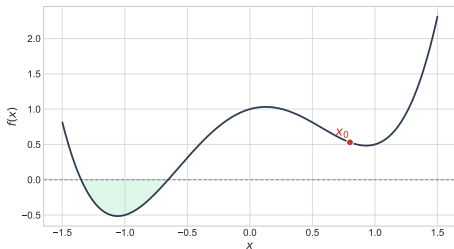


Misconception 4 about NLP

$$\begin{array}{ll}\text{minimize} & f(x) = x^4 - 2x^2 + 0.5x + 1 \\ \text{subject to} & f(x) \leq 0\end{array}$$

(Also see Wächter & Bowly, Gurobi, 2026)

NLP solvers always converge to a 'good' feasible point (KKT point)

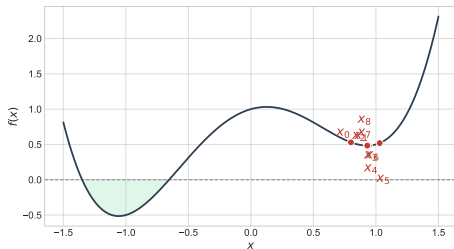


Misconception 4 about NLP

$$\begin{array}{ll}\text{minimize} & f(x) = x^4 - 2x^2 + 0.5x + 1 \\ \text{subject to} & f(x) \leq 0\end{array}$$

(Also see Wächter & Bowly, Gurobi, 2026)

NLP solvers always converge to a 'good' feasible point (KKT point)



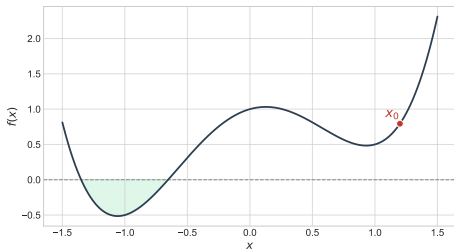
- Ipopt converges to an infeasible point when initialized at $x_0 = 0.8$

Misconception 4 about NLP

$$\begin{array}{ll}\text{minimize} & f(x) = x^4 - 2x^2 + 0.5x + 1 \\ \text{subject to} & f(x) \leq 0\end{array}$$

(Also see Wächter & Bowly, Gurobi, 2026)

NLP solvers always converge to a 'good' feasible point (KKT point)

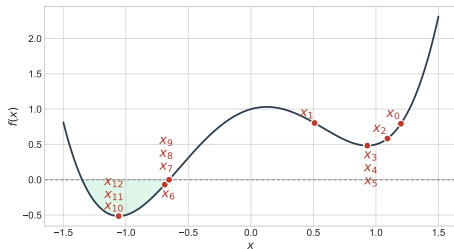


Misconception 4 about NLP

$$\begin{array}{ll}\text{minimize} & f(x) = x^4 - 2x^2 + 0.5x + 1 \\ \text{subject to} & f(x) \leq 0\end{array}$$

(Also see Wächter & Bowly, Gurobi, 2026)

NLP solvers always converge to a 'good' feasible point (KKT point)



- Ipopt converges to global minimizer when initialized at $x_0 = 1.2$

Summary of misconceptions

NLP solvers are **not** guaranteed to

- ▶ compute global minimizers
- ▶ compute the local minimizer closest to the initial point
- ▶ compute local minimizers at all (might converge to saddle points)
- ▶ compute a feasible point at all (might converge to an infeasible point)

Summary of misconceptions

NLP solvers are **not** guaranteed to

- ▶ compute global minimizers
- ▶ compute the local minimizer closest to the initial point
- ▶ compute local minimizers at all (might converge to saddle points)
- ▶ compute a feasible point at all (might converge to an infeasible point)

but in practice, NLP solvers often work well (perhaps with decent initialization)

Constraint qualifications (CQs)

- ▶ NLP solvers attempt to compute **KKT points**
- ▶ KKT conditions rely on a local linear approximation of the feasible set
 - KKT is necessary for local optimality only when this approximation is “good enough”
 - (formally: the tangent cone equals the linearized tangent cone)
- ▶ **constraint qualifications** are conditions that guarantee this
 - most common CQ: LICQ (gradients of active constraints are linearly independent)

Constraint qualifications (CQs)

- ▶ NLP solvers attempt to compute **KKT points**
- ▶ KKT conditions rely on a local linear approximation of the feasible set
 - KKT is necessary for local optimality only when this approximation is “good enough”
 - (formally: the tangent cone equals the linearized tangent cone)
- ▶ **constraint qualifications** are conditions that guarantee this
 - most common CQ: LICQ (gradients of active constraints are linearly independent)
 - we found LICQ to be important in practice

Modeling languages for NLP

- ▶ YALMIP (MATLAB), JuMP (Julia), Pyomo (Python), and CasADi (C++)
- ▶ commercial software like AMPL, GAMS, AIMMS

Modeling languages for NLP

- ▶ YALMIP (MATLAB), JuMP (Julia), Pyomo (Python), and CasADi (C++)
- ▶ commercial software like AMPL, GAMS, AIMMS
- ▶ main task of NLP modeling languages:
 - provide solvers with derivatives
- ▶ communication between modeling language and solver in every iteration
 - contrast with DCP-based languages, where the solver only ever sees the canonicalized problem data

When many existing NLP modeling languages do a bad job

$$\text{minimize } \|Ax - b\|_2^2 + \lambda \|x\|_1$$

- ▶ variable x , regularization parameter $\lambda > 0$
- ▶ convex problem (lasso) so should be solved gracefully
- ▶ the solution occurs at a point of nondifferentiability
- ▶ objective is treated as a black box and “derivatives” are supplied to the solver
- ▶ ignoring nondifferentiability might cause solvers to struggle

When existing NLP modeling languages do a bad job

output from Ipopt when interfacing it using Casadi:

```
This is Ipopt version 3.14.11, running with linear solver MUMPS 5.4.1.

...

iter    objective    inf_pr  inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  1.1126107e+01  0.00e+00  1.71e+00  -1.7  1.29e+08  -7.8  1.00e+00  1.32e-23f 77
  11  1.3152054e+10  0.00e+00  1.71e+00  -1.7  4.09e+08  -8.3  1.00e+00  1.00e+00w  1
  12  3.5423855e+10  0.00e+00  1.66e+00  -1.7  1.13e+09  -8.8  1.00e+00  1.00e+00w  1
  13  1.4112259e+11  0.00e+00  1.58e+00  -1.7  3.39e+09  -9.2  1.00e+00  1.00e+00w  1
Cannot call restoration phase at point that is almost feasible (violation 0.0).
Abort in line search due to no other fall back.

Number of Iterations.....: 13

EXIT: Error in step computation!
```

Outline

Nonlinear programming

Disciplined nonlinear programming

DNLP examples

Conclusions

Disciplined nonlinear programming (DNLP)

- ▶ a new grammar for specifying nonlinear programs
- ▶ smooth functions can be freely mixed with nonsmooth functions
- ▶ rules governing how nonsmooth functions can appear
- ▶ smooth = twice continuously differentiable in the interior of their domain

Disciplined nonlinear programming (DNLP)

- ▶ a new grammar for specifying nonlinear programs
- ▶ smooth functions can be freely mixed with nonsmooth functions
- ▶ rules governing how nonsmooth functions can appear
- ▶ smooth = twice continuously differentiable in the interior of their domain

a DNLP-compliant problem can be transformed to a form that

- ▶ only involves smooth functions
- ▶ does not introduce LICQ violations in the transformation

The DNLP grammar

in DNLP, problems are constructed using **atoms** with known attributes

- ▶ atoms are divided into three categories:
 - **smooth**
 - **nonsmooth convex**
 - **nonsmooth concave**
- ▶ DNLP enforces simple rules on how atoms can be composed

The DNLP grammar

in DNLP, problems are constructed using **atoms** with known attributes

- ▶ atoms are divided into three categories:
 - **smooth**
 - **nonsmooth convex**
 - **nonsmooth concave**
- ▶ DNLP enforces simple rules on how atoms can be composed
- ▶ the DNLP rules are very simple if you know the DCP rules:

a problem is DNLP if it is DCP when smooth atoms are linearized

Some new DNLP atoms

Atom	Definition
multiply	$\phi(x, y) = xy$
quad_form	$\phi(x) = x^T Q x$ where $Q \in \mathbf{S}^n$
sin	$\phi(x) = \sin x$
sigmoid	$\phi(x) = \frac{1}{1+e^{-x}}$
normal_cdf	$\phi(x) = (1/\sqrt{2\pi}) \int_{-\infty}^x e^{-t^2/2} dt$

- ▶ these atoms are not convex or concave, but they are smooth

Dealing with nonsmooth atoms

- ▶ a DNLP problem is equivalent to a problem where any
 - nonsmooth convex atom ϕ appears in a constraint of the form $\phi(x) \leq t$
 - nonsmooth concave atom ϕ appears in a constraint of the form $\phi(x) \geq t$

Dealing with nonsmooth atoms

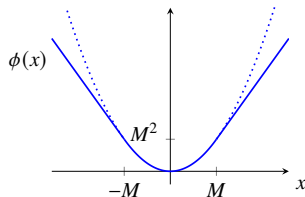
- ▶ a DNLP problem is equivalent to a problem where any
 - nonsmooth convex atom ϕ appears in a constraint of the form $\phi(x) \leq t$
 - nonsmooth concave atom ϕ appears in a constraint of the form $\phi(x) \geq t$
- ▶ these constraints can be formulated using smooth functions [Grant and Boyd, 2008]
 - the reformulations do not introduce LICQ violations
- ▶ **example:** $|x| \leq t$ can be reformulated as $-t \leq x \leq t$

Dealing with nonsmooth atoms

- ▶ a DNLP problem is equivalent to a problem where any
 - nonsmooth convex atom ϕ appears in a constraint of the form $\phi(x) \leq t$
 - nonsmooth concave atom ϕ appears in a constraint of the form $\phi(x) \geq t$
- ▶ these constraints can be formulated using smooth functions [Grant and Boyd, 2008]
 - the reformulations do not introduce LICQ violations
- ▶ **example:** $|x| \leq t$ can be reformulated as $-t \leq x \leq t$
- ▶ other nonsmooth atoms, such as the Huber atom

$$\phi(x) = \begin{cases} x^2 & |x| \leq M \\ 2M|x| - M^2 & |x| > M \end{cases}$$

have more complex reformulations



Outline

Nonlinear programming

Disciplined nonlinear programming

DNLP examples

Conclusions

A DNLP extension of CVXPY

- ▶ DNLP is implemented in CVXPY version 1.9
- ▶ supported solvers:
 - IPOPT (open-source, interior-point method)
 - Uno [Vanaret and Leyffer, 2025] (open-source, IPM and SQP)
 - KNITRO (commercial, many algorithms)
 - COPT (commercial, interior-point method)
- ▶ computes sparse Jacobians and Hessians via C backend

Example: Circle packing

$$\begin{array}{ll}\text{minimize} & \max_{i=1,\dots,n} (\|c_i\|_\infty + r_i) \\ \text{subject to} & \|c_i - c_j\|_2^2 \geq (r_i + r_j)^2, \quad 1 \leq i < j \leq n,\end{array}$$

- ▶ goal is to pack n circles with given radii r_i into smallest square
- ▶ variables are the circle centers $c_i \in \mathbf{R}^2$
- ▶ becomes DCP when smooth atoms are linearized, so DNLP-compliant

Example: Circle packing

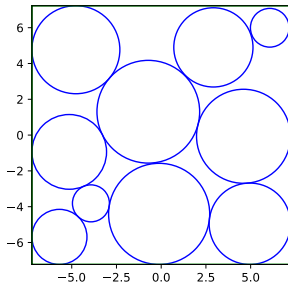
$$\begin{array}{ll}\text{minimize} & \max_{i=1,\dots,n} (\|c_i\|_\infty + r_i) \\ \text{subject to} & \|c_i - c_j\|_2^2 \geq (r_i + r_j)^2, \quad 1 \leq i < j \leq n,\end{array}$$

- ▶ goal is to pack n circles with given radii r_i into smallest square
- ▶ variables are the circle centers $c_i \in \mathbf{R}^2$
- ▶ becomes DCP when smooth atoms are linearized, so DNLP-compliant

```
1 c, constr = Variable((n, 2)), []
2 for i in range(n - 1):
3     constr += [sum((c[i, :] - c[i+1:, :]) ** 2, axis=1) >= (r[i] + r[i+1:]) ** 2]
4 cost = max(norm_inf(c, axis=1) + r)
5 prob = Problem(Minimize(cost), constr)
6 c.value = uniform(-5.0, 5.0, (n, 2))           # random initial point
7 prob.solve(nlp=True)                          # or prob.solve(nlp=True, best_of=500)
```

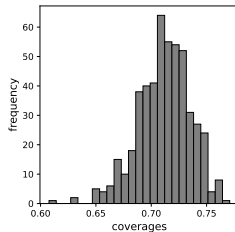
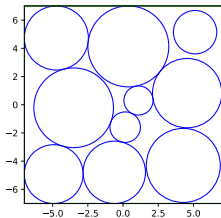
Example: Circle packing

- ▶ result for $n = 10$ circles with random radii, with random initial point
- ▶ the fraction of the square covered by the circles is 0.72



Example: Circle packing

- ▶ result for same problem instance over 500 random initializations using `best_of=500`
- ▶ best coverage found is 0.77



Example: Sparse signal recovery

$$\begin{array}{ll}\text{minimize} & \sum_{i=1}^n \sqrt{|x_i|} \\ \text{subject to} & Ax = y,\end{array}$$

- ▶ goal is to recover a sparse signal $x_0 \in \mathbf{R}^n$ from m given measurements $y = Ax_0$
- ▶ variable is the signal $x \in \mathbf{R}^n$
- ▶ the objective is nonconvex but promotes sparsity better than the ℓ_1 norm
- ▶ becomes DCP when smooth atoms are linearized, so DNLP-compliant

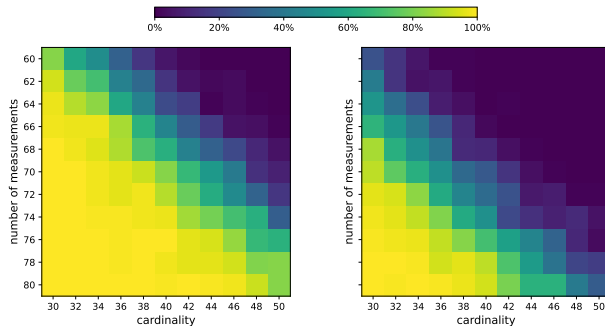
Example: Sparse signal recovery

$$\begin{array}{ll}\text{minimize} & \sum_{i=1}^n \sqrt{|x_i|} \\ \text{subject to} & Ax = y,\end{array}$$

- ▶ goal is to recover a sparse signal $x_0 \in \mathbf{R}^n$ from m given measurements $y = Ax_0$
- ▶ variable is the signal $x \in \mathbf{R}^n$
- ▶ the objective is nonconvex but promotes sparsity better than the ℓ_1 norm
- ▶ becomes DCP when smooth atoms are linearized, so DNLP-compliant

```
1 x = Variable(n)
2 cost, constr = sum(sqrt(abs(x))), [A @ x == y]
3 prob = Problem(Minimize(cost), constr)
4 prob.solve(nlp=True, solver='knitro_ipm')
```

Example: Sparse signal recovery



- probability of successful recovery for signal dimension $n = 100$, nonconvex approach (left) and convex ℓ_1 approach (right)

Many more examples in the paper

- ▶ planning and geometry
- ▶ signal processing
- ▶ finance
- ▶ energy systems
- ▶ statistics

Outline

Nonlinear programming

Disciplined nonlinear programming

DNLP examples

Conclusions

Conclusions

- ▶ nonlinear programming solvers seek local solutions of general nonconvex problems
- ▶ disciplined nonlinear programming is a new grammar for specifying NLPs
 - smooth functions can be freely mixed with nonsmooth functions
 - a problem is DNLP if it is DCP when smooth atoms are linearized
 - implemented in CVXPY
- ▶ **NLP solvers lack optimality guarantees and depend on the initialization**