

A Flexible Solver for PDE-Constrained Optimization

Dominique Orban

dominique.orban@gerad.ca

joint work with

Nick Gould and Sue Thorne (Rutherford Appleton Lab)

SIAM CSE 2011, Reno NV

Topics

- Equality-Constrained Optimization
- Python, optimization and NLPy
- Python, PDEs and FEniCS
- PDE-Constrained Optimization

Equality-Constrained Optimization

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad c(x) = 0$$

$$\nabla f(x^*) + J(x^*)^T y^* = 0 \quad c(x^*) = 0$$

$$Z(x^*)^T \nabla^2 L(x^*, y^*) Z(x^*) \succeq 0$$

- Constraint gradients linearly independent at solution

Typical Step Computation

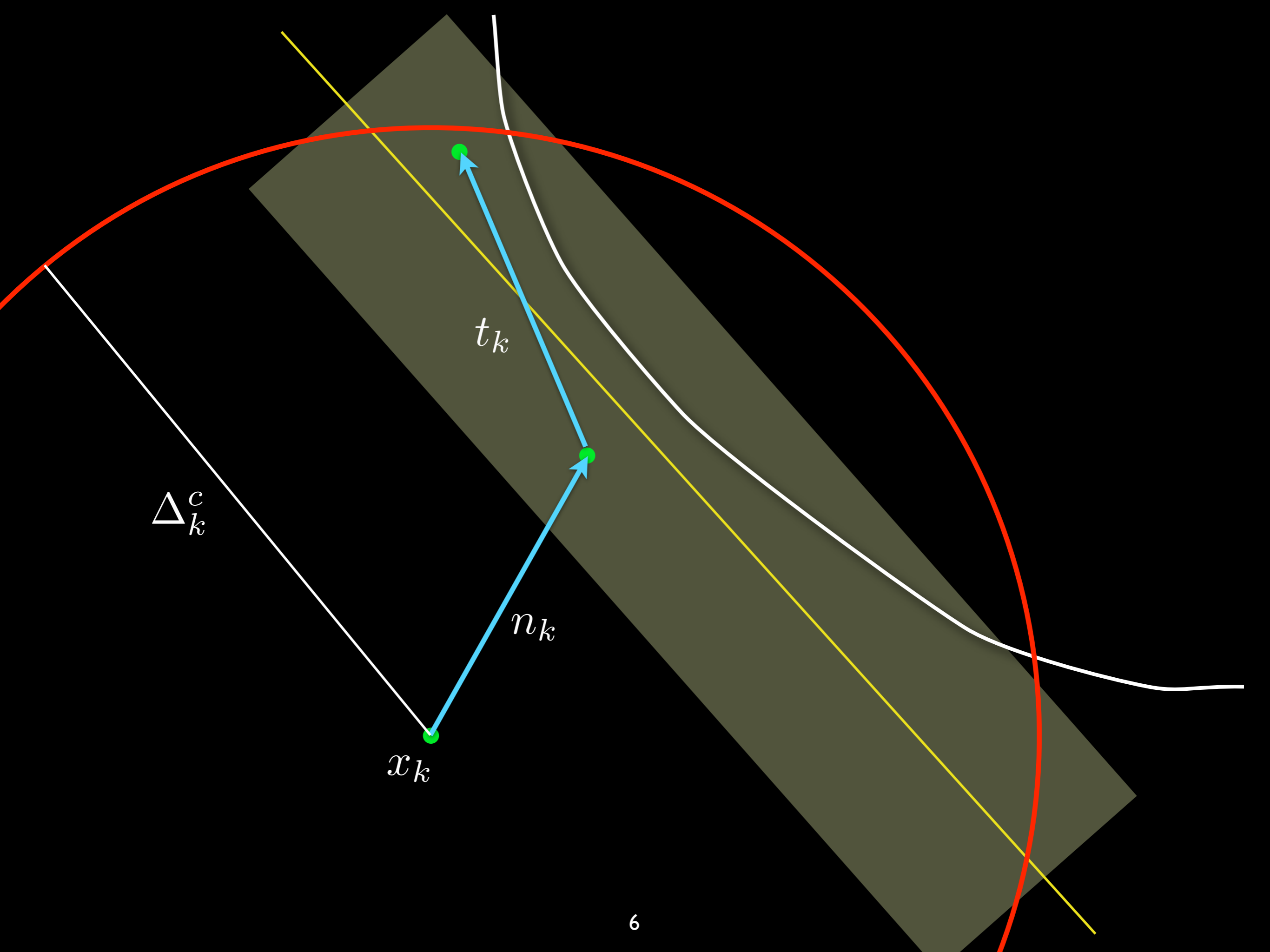
$$\begin{bmatrix} H(x, y) & A(x)^T \\ A(x) & -C \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} g(x, y) \\ c(x) \end{bmatrix}$$

- Always symmetric, typically indefinite
- Step computed “exactly” or “inexactly”
- Line search or trust region
- Step acceptance based on some merit function or filter
- Inexact scheme: [Byrd, Curtis, Nocedal, 2010]

The Funnel Method

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ f(x) \quad \text{subject to} \ c(x) = 0$$

- Matrix-free and inexact by construction
- No filter or parameter-dependent merit function
- Dissociate objective and constraints using two models
- No need to satisfy linearized constraints accurately
- Normal/tangential steps computed only if relevant
- Globalized via two trust-region mechanisms
- [Gould and Toint, 2008]



Intentionally Vague Outline of the Funnel Method

1. Compute a “normal” step if infeasibility is significant

$$\underset{n}{\text{minimize}} \quad \frac{1}{2} \|c_k + J_k n\|^2 \quad \text{subject to} \quad \|n\| \leq \Delta_k^c, \quad \|n\| \leq \kappa_n \|c_k\|$$

2. Compute updated multiplier estimates

$$\underset{y}{\text{minimize}} \quad \frac{1}{2} \|\bar{g}_k + J_k^T y\|^2$$

$$\text{Residual: } r_k := \bar{g}_k + J_k^T y_k \approx \text{Proj}(\bar{g}_k; \text{Null}(J_k))$$

3. Compute a “tangential” step if criticality residual lags behind infeasibility

$$\begin{aligned} & \underset{t}{\text{minimize}} \quad f_k + \bar{g}_k^T t + \frac{1}{2} t^T G_k t \\ & \text{subject to} \quad \|c_k + J_k(n_k + t)\|^2 \leq \bar{\theta}_k, \\ & \quad \quad \quad \|t\| \leq \Delta_k \end{aligned}$$

G_k is an approximate Lagrangian Hessian that incorporates the updated multipliers. G_k need not be definite.

Criticality measure: $\pi_k := \frac{\bar{g}_k^T r_k}{\|r_k\|^2}$

Essential Convergence Properties

Assumption: $J_k := J(x_k)$ is uniformly nonsingular over all iterations when close to feasibility

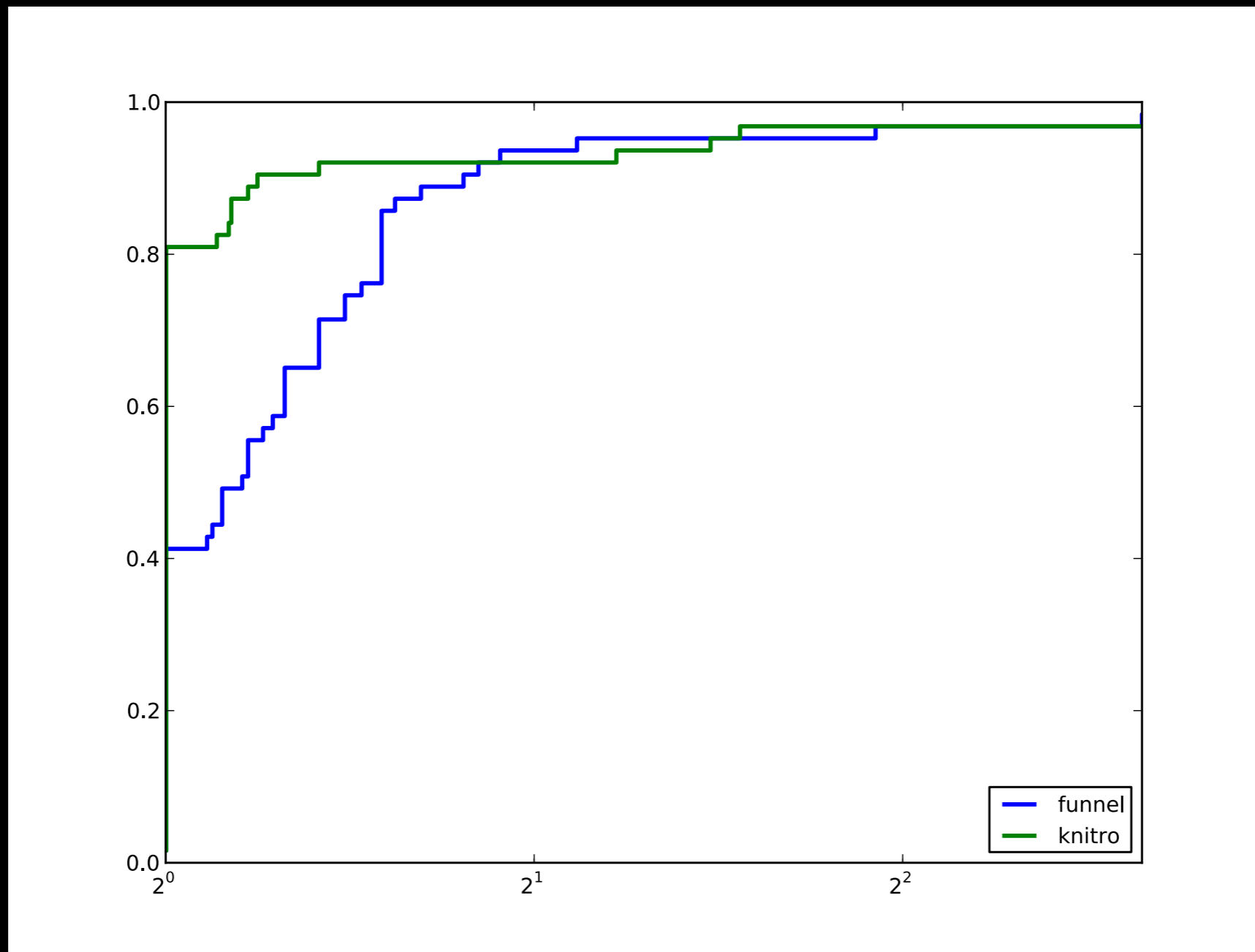
THEOREM

Either:

1. A subsequence approaches an infeasible stationary point for the infeasibility measure, or
2. all limit points are first-order critical.

Funnel vs. Knitro 6.0

- # objective evaluations profile
- 63 equality-constrained problems from CUTEr



PDE Modeling Environments

- Open or commercial
- Often specific to a (family of) PDE system(s) or to an application
- In a variety of languages
- FreeFEM++, IFISS, COMSOL, Sundance, deal.II, SfePy, and many more
- FEniCS!

FEniCS and DOLFIN

- Specify function spaces, functionals, finite-element bases and weak form of PDE systems symbolically
- Handles discretization on a given mesh
- Several linear algebra backends (PETSc, Trilinos, uBLAS, MTL4, ...)
- AD on functionals and weak forms
- Python bindings: Dolfin [Logg, Wells, 2010]
- <http://www.fenicsproject.org>

Example

$$\Omega = [0, 1] \times [0, 1]$$

$$-\nabla \cdot ((1 + u^2)\nabla u) = f \text{ in } \Omega$$

$$u = 1 \text{ on } \partial\Omega \text{ where } x = 1,$$

$$\frac{du}{d\vec{n}} = 0 \text{ on } \partial\Omega \text{ where } x \neq 1$$

```
from dolfin import *

# Sub domain for Dirichlet boundary condition
class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return abs(x[0] - 1.0) < DOLFIN_EPS and on_boundary

# Create mesh and define function space
mesh = UnitSquare(16, 16)
V = FunctionSpace(mesh, "CG", 1)

# Define boundary condition
g = Constant(1.0)
bc = DirichletBC(V, g, DirichletBoundary())

# Define source and solution functions
f = Expression("x[0]*sin(x[1])")
u = Function(V)

# Define variational problem
v = TestFunction(V)
du = TrialFunction(V)
L = dot(grad(v), (1 + u**2)*grad(u))*dx - v*f*dx
a = derivative(L, u, du)

# Solve nonlinear variational problem
problem = VariationalProblem(a, L, bc, nonlinear=True)
problem.solve(u)
```

The NLPy Platform

- An OO toolbox for optimization in Python
- AMPLModel or NLPModel
- Common linear algebra tools
- Common subproblem solvers
- Several complete solvers
- <https://github.com/dpo/nlpy>

Building Blocks

Line search ([csrch](#), [mcsrch](#))

Sparse matrix data structures (via [pysparse.sf.net](#))

Krylov-type methods (native)

Trust-region subproblem ([GLTR](#), native truncated CG)

Factorizations ([HSL MA57](#), [ICFS](#), [SuperLU](#), [UMFPACK](#))

Generic Preconditioners (Band, L-BFGS, [ICFS](#))

Modeling language ([AMPL](#), also native without AD... yet!)

Linear operators (native)

...

Problems Adressed

Linear Programs

Quadratic Programs

Linear and Nonlinear Least-Squares

Unconstrained Nonlinear Programs

Constrained Nonlinear Programs

Convex or Non-convex Nonlinear Programs

Degenerate Problems

```

from nlp.model import AmplModel

nlp = AmplModel('someNlFile') # Created by AMPL

x0 = nlp.x0 # Initial point
pi0 = nlp.pi0 # Initial multipliers
print '%d variables, %d constraints' % (nlp.n, nlp.m)

print 'Lower bounds on x: ', nlp.Lvar # Numpy array
print 'Upper bounds on x: ', nlp.Uvar # Numpy array
print 'f(x0) = ', nlp.obj(x0)
print 'grad f(x0) = ', nlp.grad(x0) # Numpy array

if nlp.m > 0:
    print 'Lower constraint bounds: ', nlp.Lcon
    print 'Upper constraint bounds: ', nlp.Ucon
    c = nlp.cons(x0)
    J = nlp.jac(x0) # Pysparse sparse matrix

H = nlp.hess(x0, pi0) # Pysparse sparse matrix

print ' nnzJ = ', J.nnz
print ' nnzH = ', H.nnz

```

Anatomy of a PDE-Constrained Optimization Problem

- Subclass NLPy's `NLPModel` to store mesh and function space information: `PDENLPModel`
- User subclasses `PDENLPModel` to define objective and constraints.
- `PDENLPModel`'s structure ensures that expressions are assembled when needed and derivatives are computed via FEniCS when required.

Example

$$\begin{aligned} \min \quad & \frac{1}{2} \int_{\Omega} |u|^2 \\ \text{s.t.} \quad & \int_{\Omega} \nabla u \cdot \nabla v = \int_{\Omega} f v \\ & u(\partial\Omega) = u_0 \end{aligned}$$

```
class ExamplePDENLPMoDel(PDENLPMoDel):  
  
    def register_objective_functional(self):  
  
        u = self.u  
        self.objective_functional = 0.5 * dot(u,u)*dx  
        return  
  
    def register_variational_problem(self):  
  
        u = self.u  
        v = TestFunction(self.function_space)  
  
        # Define and register boundary conditions.  
        allbndry = AllBoundary()  
        u0 = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]')  
        bc = DirichletBC(self.function_space, u0, allbndry)  
        self.bcs.append(bc)  
  
        # Define linear form.  
        f = Expression('-8*x[0] - 10*x[1]')  
        self.L = f*v*dx  
  
        # Define bilinear form.  
        self.a = dot(grad(u), grad(v))*dx  
        return
```

Passing a Problem to Funnel

```
mesh = UnitSquare(5,5)
V = FunctionSpace(mesh, 'CG', 1)

pdenlp = ExamplePDENLPModel(mesh, V, name='Simple problem')
pdenlp.register_objective_functional()
pdenlp.register_variational_problem()

u = Expression('1 + x[0]')
pdenlp.set_initial_guess(u)

funn = Funnel(pdenlp)
funn.solve()
```

iter	NT	f	theta	g+Jy	Delta_f	Delta_c	thetaMax
0		1.17e+00	1.40e+01		1.00e+01	1.00e+01	1.00e+03
1cv	c:	2.07e+00	3.21e-14	7.57e-09	1.00e+01	1.36e+01	9.00e+02

Quadratic Example

$$\underset{u, f}{\text{minimize}} \quad \frac{1}{2} \|u - \hat{u}\|^2 + \frac{1}{2} \beta \|f\|^2$$

$$\text{subject to} \quad -\nabla \cdot \nabla u = f$$

$$u = \hat{u} \text{ on } \partial\Omega$$

$$\Omega = [0, 1]^n \quad \hat{u} = \begin{cases} \prod_i (2x_i - 1)^2 & \text{in } [0, \frac{1}{2}]^n, \\ 0 & \text{otherwise.} \end{cases}$$

- uniform triangle mesh
- Lagrange elements of degree 1 for u and f

nh=16
nvar=578

iter	NT	f	cl	lg+Jyl	Delta_f	Delta_c	thetaMax	CG
0		4.87e-01	7.52e+00	2.17e-03	1.00e+01	1.50e+01	1.00e+03	0
1cv	b:	2.16e-02	1.99e-01	9.27e-03	1.00e+01	3.76e+01	9.00e+02	0
2fv	c-	4.18e-03	7.82e-08	1.55e-03	2.22e+01	3.76e+01	9.00e+02	0
3fv	cc	7.86e-04	3.04e-09	1.27e-03	2.52e+01	3.76e+01	9.00e+02	15
4cv	cc	7.86e-04	3.04e-09	1.07e-08	2.52e+01	3.76e+01	8.10e+02	0

nh=32
nvar=2178

iter	NT	f	cl	lg+Jyl	Delta_f	Delta_c	thetaMax	CG
0		4.87e-01	1.06e+01	7.93e-04	1.00e+01	2.13e+01	1.00e+03	0
1cv	b:	1.51e-01	3.34e-01	1.65e-02	1.00e+01	5.32e+01	9.00e+02	0
2cv	c:	1.38e-02	3.06e-08	2.56e-03	1.00e+01	5.32e+01	8.10e+02	0
3fv	c-	7.51e-03	5.12e-09	6.02e-04	2.50e+01	5.32e+01	8.10e+02	0
4fv	c-	9.40e-04	5.12e-09	9.49e-04	6.25e+01	5.32e+01	8.10e+02	0
5fv	cc	8.23e-04	5.12e-09	4.25e-05	6.25e+01	5.32e+01	8.10e+02	13
6cv	cc	8.23e-04	5.12e-09	1.07e-08	6.25e+01	5.32e+01	7.29e+02	0

nh=64
nvar=8450

iter	NT	f	cl	lg+Jyl	Delta_f	Delta_c	thetaMax	CG
0		4.87e-01	1.50e+01	2.54e-04	1.00e+01	3.01e+01	1.00e+03	0
1cv	b:	3.56e-01	4.53e-01	1.30e-02	1.00e+01	7.52e+01	9.00e+02	0
2cv	b:	2.31e-02	5.55e-02	2.49e-03	1.00e+01	1.13e+02	8.10e+02	0
3fv	c-	1.22e-02	9.84e-09	2.82e-04	1.77e+01	1.13e+02	8.10e+02	0
4fv	c-	6.87e-03	9.66e-09	2.55e-04	4.43e+01	1.13e+02	8.10e+02	0
5fv	c-	9.25e-04	9.66e-09	3.87e-04	1.11e+02	1.13e+02	8.10e+02	0
6fv	cc	8.33e-04	9.67e-09	1.52e-05	1.11e+02	1.13e+02	8.10e+02	12
7cv	cc	8.33e-04	9.67e-09	9.20e-09	1.11e+02	1.13e+02	7.29e+02	0

Nonlinear Example

$$\underset{u, f}{\text{minimize}} \quad \frac{1}{2} \|u - \hat{u}\|^2 + \frac{1}{2} \beta \|f\|^2$$

$$\text{subject to} \quad -\nabla \cdot ((1 + u^2) \nabla u) = f$$

$$u = \hat{u} \text{ on } \partial\Omega$$

$$\hat{u} = \begin{cases} 16\Pi_i(x_i - 1)^2 & \text{in } [0, \frac{1}{2}]^n, \\ 0 & \text{otherwise.} \end{cases}$$

iter	NT	f	c	g+Jy	Delta_f	Delta_c	thetaMax	CG
0		4.87e-01	6.06e-02	4.25e-02	1.00e+00	1.00e+00	1.00e+03	0
1cv	b0	4.86e-01	5.61e-02	5.93e-02	1.00e+00	2.50e+00	9.00e+02	0
2cv	b0	4.83e-01	5.03e-02	5.92e-02	1.00e+00	6.25e+00	8.10e+02	0
3cv	b0	4.79e-01	3.69e-02	5.92e-02	1.00e+00	1.26e+01	7.29e+02	0
4cv	b0	4.77e-01	2.13e-02	5.92e-02	1.00e+00	1.26e+01	6.56e+02	0
5cv	b0	4.77e-01	1.13e-02	5.92e-02	1.00e+00	1.26e+01	5.90e+02	0
6cv	b0	4.77e-01	7.82e-03	5.92e-02	1.00e+00	1.26e+01	5.31e+02	0
7fv	b-	4.64e-01	4.26e-03	5.45e-02	2.00e+00	1.26e+01	5.31e+02	0
8fv	b-	3.80e-01	2.38e-03	4.94e-02	4.03e+00	1.26e+01	5.31e+02	0
9fv	b-	2.12e-01	1.27e-03	3.66e-02	9.50e+00	1.26e+01	5.31e+02	0
10fv	b-	9.12e-03	1.66e-03	6.30e-03	2.34e+01	1.26e+01	5.31e+02	0
11fv	rr	4.35e-03	1.55e-03	1.04e-03	2.34e+01	1.26e+01	5.31e+02	19
12fv	rr	4.26e-03	1.77e-05	7.98e-05	2.34e+01	1.26e+01	5.31e+02	14
13cv	rr	4.26e-03	6.72e-08	9.09e-06	2.34e+01	1.26e+01	4.78e+02	8

Funnel Subproblems

$$\underset{n}{\text{minimize}} \quad \frac{1}{2} \|c_k + J_k n\|^2 \quad \text{subject to} \quad \|n\| \leq \Delta_k^c, \quad \|n\| \leq \kappa_n \|c_k\|$$

- Currently solved with LSQR or truncated CG
- Note that often:

$$J_k = \begin{bmatrix} A_k & B_k^T \\ B_k & C_k \end{bmatrix}$$

Funnel Subproblems

$$\underset{y}{\text{minimize}} \quad \frac{1}{2} \|\bar{g}_k + J_k^T y\|^2$$

- LSQR
- CG
- Any iterative method for

$$\begin{bmatrix} I & J_k^T \\ J_k & 0 \end{bmatrix} \begin{bmatrix} \rho \\ y \end{bmatrix} = \begin{bmatrix} -\bar{g}_k \\ 0 \end{bmatrix}$$

(MINRES, SYMMLQ, ...)

Funnel Subproblems

$$\underset{t}{\text{minimize}} \quad f_k + \bar{g}_k^T t + \frac{1}{2} t^T G_k t$$

$$\text{subject to} \quad \|c_k + J_k(n_k + t)\|^2 \leq \bar{\theta}_k,$$

$$\|t\| \leq \Delta_k$$

- Currently: Truncated Projected CG
- Nested saddle-point-like structure:

$$\begin{bmatrix} G_k & J_k^T \\ J_k & \end{bmatrix}$$

Extensions

- Mixture of differential and algebraic constraints
- Inequality constraints

Final Comments

- Explore alternate subproblem solvers
- “Exact” second derivatives of a weak form in DOLFIN? Hopefully coming up...
- Relevant approximations of second derivatives?
- Mesh refinement?

Thank you!

www.gerad.ca/~orban