TAO

Toolkit for Advanced Optimization

Jason Sarich, Todd Munson, Stefan Wild
Argonne National Laboratory
June 15, 2015

Algorithms

$$\begin{array}{rcl} \min & f(x) & \text{(objective function)} \\ \text{subject to} & x_l \leq x \leq x_u & \text{(optional box constraints)} \end{array}$$

Nonlinear optimization algorithms are iterative processes. In many cases, each iteration involve calculating a **search direction**, then function values and gradients along that direction are calculated until certain conditions are met.

- Conjugate Gradient
- Newton's Method
- Quasi-Newton Methods

Algorithms

Conjugate Gradient Algorithms

These algorithms are an extension of the conjugate gradient methods for solving linear systems. The search direction is computed with

$$d_{k+1} = -g_k + \beta_k d_k \quad (g_k = \nabla f(x_k)\ )$$

then a line search is conducted to find α_{k+1} that satisfies sufficient decrease and curvature conditions.

$$x_{k+1} = x_k + \alpha_{k+1} d_{k+1}$$

$$\beta_k^{FR} = \left(\frac{\|g_{k+1}\|}{\|g_k\|}\right)^2, \qquad \text{Fletcher-Reeves}$$

$$\beta_k^{PR} = \frac{\langle g_{k+1}, g_{k+1} - g_k\rangle}{\|g_k\|^2}, \qquad \text{Polak-Ribière}$$

$$\beta_k^{PR+} = \max\left\{\beta_k^{PR}, 0\right\}, \qquad \text{PR-plus}$$

Algorithms

Newton's Method

- Step 0 Choose initial vector x_0
- Step 1 Compute gradient $\nabla f(x_k)$ and Hessian $\nabla^2 f(x_k)$
- Step 2 Calculate the direction d_{k+1} by solving the system:

$$\nabla^2 f(x_k) d_{k+1} = -\nabla f(x_k)$$

- Step 3 Apply line search algorithm to obtain "acceptable" new vector:
- Step 2-3 Trust Region Alternative Calculate d_{k+1} by solving the system:

$$\begin{array}{cc} \min\limits_{d} & g_k^T d_{k+1} + \frac{1}{2} d_{k+1}^T \nabla^2 f(x_k) d_{k+1} \\ s.t. & \|d_{k+1}\|_2 \leq \Delta_k \end{array}$$

- Return to Step 1

Algorithms

Some notes about Newton's Method

- Newton's method converges quadratically when close to the solution (good!)

Algorithms

Some notes about Newton's Method

- Newton's method converges quadratically when close to the solution (good!)
- Hessian must be derived, computed, and stored (bad?)

Algorithms

Some notes about Newton's Method

- Newton's method converges quadratically when close to the solution (good!)
- Hessian must be derived, computed, and stored (bad?)
- Linear solve must be performed on Hessian (bad!)

Algorithms

Quasi-Newton Methods

Use approximate Hessian $B_k \approx \nabla^2 f(x_k)$. Choose a formula for B_k so that:

- B_k relies on first derivative information only
- B_k can be easily stored
- $B_k d_{k+1} = -\nabla f(x_k)$ can be easily solved

Algorithms

Limited Memory Variable Metric (LMVM)

Quasi-Newton method using L-BFGS update and Moré-Thuente line search.

$$\begin{aligned} s_k &= x_k - x_{k-1} \\ y_k &= g_k - g_{k-1} \\ B_{k+1} &= B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \end{aligned}$$

Using this update and a diagonal initial B_0, the sytem

$$B_{k+1}^{-1} x = -g$$

can be solved directly using only AXPYs and Dot products.

Algorithms

Box-constrained algorithms

$$\begin{array}{rcl} \min & f(x) & \text{(objective function)} \\ \text{subject to} & x_l \leq x \leq x_u & \text{(optional box constraints)} \end{array}$$

- GPCG – Gradient Projection Conjugate Gradient (quadratic problems only)
- TRON – Newton trust region algorithm on free variables
- BLMVM – Bounded Quasi-Newton algorithm

Algorithms

Complementarity

Mixed complementarity problems, or box-constrained variational inequalities.

$$\begin{array}{ll} F_i(x^*) \geq 0 & \text{if } x_i^* = \ell_i \\ F_i(x^*) = 0 & \text{if } \ell_i < x_i^* < u_i \\ F_i(x^*) \leq 0 & \text{if } x_i^* = u_i. \end{array}$$

- Semismooth Solvers (SSILS, SSFLS)
- Active Set Solvers (ASILS, ASFLS)

Algorithms

PDE-constrained systems

TAO solves PDE-constrained optimization problems of the form

$$\begin{array}{ll} \min_{u,v} & f(u,v) \\ \text{subject to} & g(u,v) = 0, \end{array}$$

where the state variable u is the solution to the discretized partial differential equation defined by g and parametrized by the design variable v, and f is an objective function.

Algorithms

PDE-constrained systems

TAO solves PDE-constrained optimization problems of the form

$$\begin{array}{ll} \min\limits_{u,v} & f(u,v) \\ \text{subject to} & g(u,v) = 0, \end{array}$$

where the state variable u is the solution to the discretized partial differential equation defined by g and parametrized by the design variable v, and f is an objective function.

- Linearly-Constrained Augmented Lagrangian Method (LCL)

Algorithms

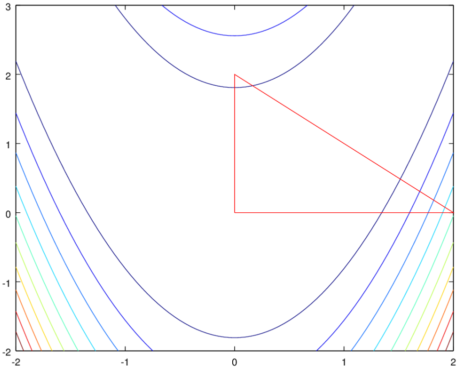
Derivate Free Algorithms

There are some applications for which it is not feasible to find the derivative of the objective function. There are some algorithms available that can solve these applications, but they can be very slow to converge.

- Model-based methods
- Use finite differences
- Nelder-Mead Simplex
- Automatic Differentiation

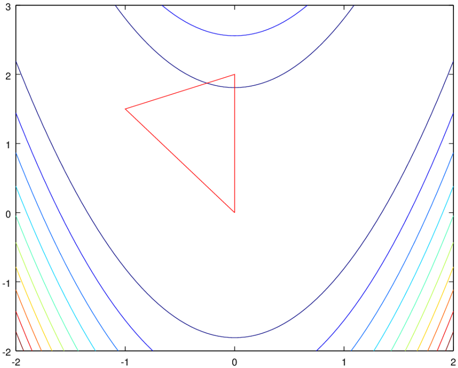
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



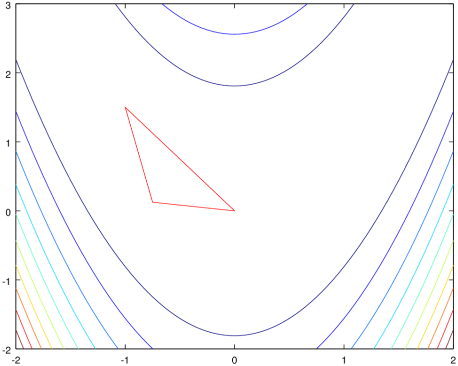
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



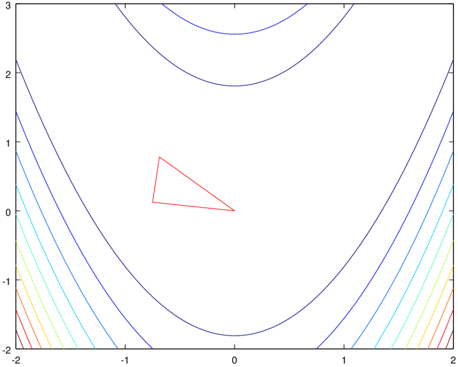
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



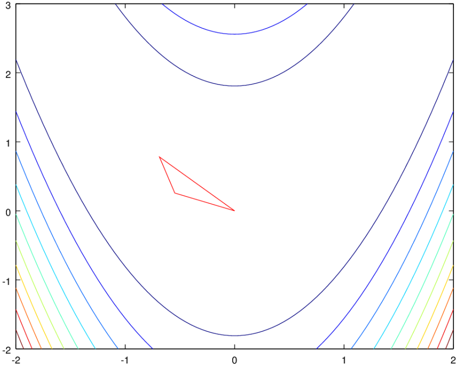
Nelder-Mead

Nelder-Mead algorithms forms an $n + 1$-dimensional simplex and moves one vertex at a time



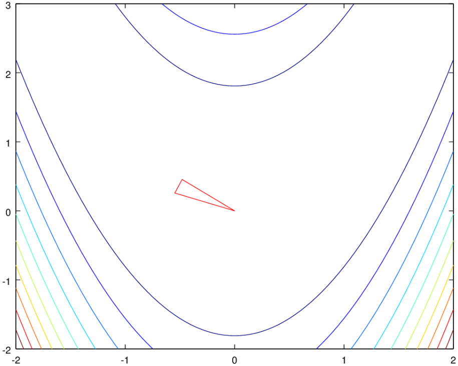
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



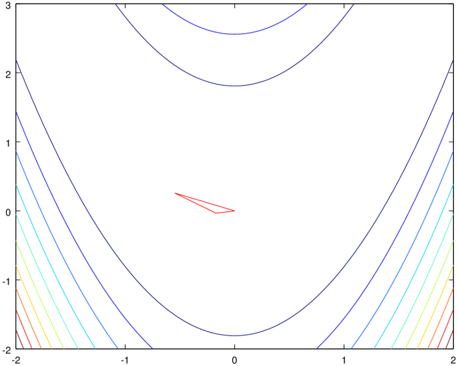
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



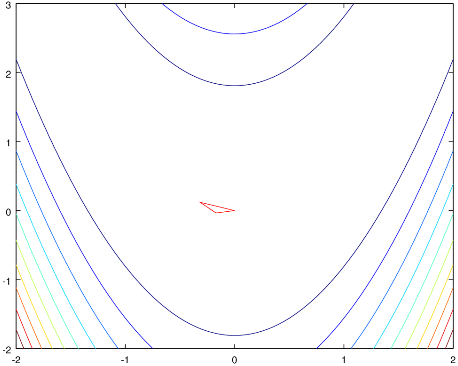
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



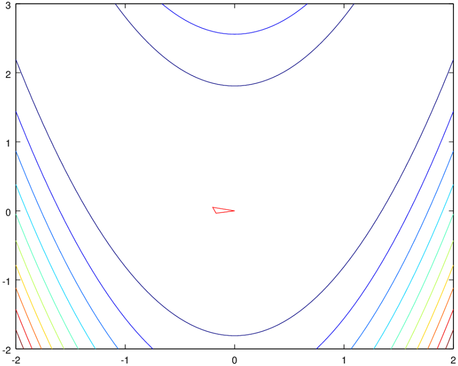
Nelder-Mead

Nelder-Mead algorithms forms an $n + 1$-dimensional simplex and moves one vertex at a time



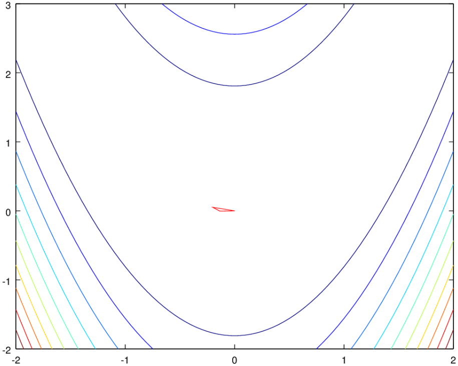
Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time

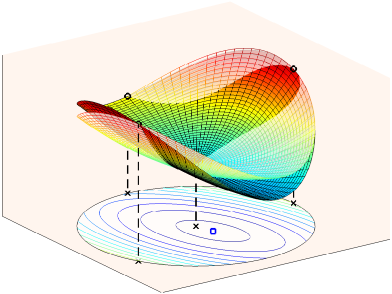


Nelder-Mead

Nelder-Mead algorithms forms an $n+1$-dimensional simplex and moves one vertex at a time



POUNDERS - Model-based Derivate-free optimization



$n = 2$, $|\mathcal{Y}\_k| = 4$

using an interpolating quadratic,

$$q_k(x_k + y_i) = f(x_k + y_i), \quad \forall y_i \in \mathcal{Y}_k.$$

$\rightarrow$ Function values are all you have

- Other models possible
- Only provide local approximation
- Coarse models $\leftrightarrow$ smooth noise

POUNDERS - Nonlinear Least Squares

$$f(x) = \frac{1}{2}\sum_{i=1}^{p}(S_i(x) - d_i)^2$$

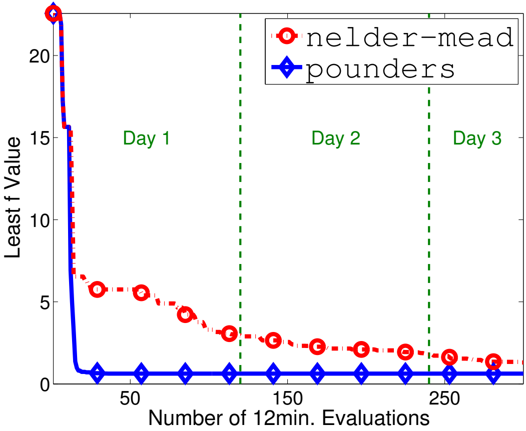
- Obtain a vector of output $S_1(x), \ldots, S_p(x)$ with each simulation
- Approximate:

$$\begin{aligned} \nabla f(x) &= \sum_i \nabla \mathbf{S_i(x)}(S_i(x) - d_i) \\ &\to \sum_i \nabla \mathbf{m_i(x)}(S_i(x) - d_i) \end{aligned}$$

$$\begin{aligned} \nabla^2 f(x) &= \sum_i \nabla \mathbf{S_i(x)} \nabla \mathbf{S_i(x)}^T + \sum_i (S_i(x) - d_i) \nabla^2 \mathbf{S_i(x)} \\ &\to \sum_i \nabla \mathbf{m_i(x)} \nabla \mathbf{m_i(x)}^T + \sum_i (S_i(x) - d_i) \nabla^2 \mathbf{m_i(x)} \end{aligned}$$

- Model f via Gauss-Newton or similar

POUNDERS for hfbtho



- ○ 72 cores on Jazz
- ○ 12 wall-clock minutes per $f(\mathbf{x})$

- POUNDERS: acceptable $\mathbf{x}$ in 3.2 hours
- Nelder-Mead: no acceptable $\mathbf{x}$ in 60 hours

Finite Differences

It is possible (though highly unrecommended) to use finite differences to approximate the gradient (and/or Hessian). It is recommended to test

the accuracy of hand-coded gradients and Hessians using finite differences. This can be with command line options using the special TAO solver “test”:

```
-tao_type test -tao_test_hessian
```

TAO Solvers

Solvers available in TAO

	handles constraints	requires gradient	requires Hessian
Quasi-Newton (lmvm)	no	yes	no
Newton Line Search (nls)	no	yes	yes
Newton Trust Region (ntr)	no	yes	yes
Newton Trust with Line Search (ntl)	no	yes	yes
Conjugate Gradient (cg)	no	yes	no
Nelder-Mead (nm)	no	no	no
Quasi-Newton (blmvm)	bounds	yes	no
Newton Trust Region (tron)	bounds	yes	yes
Conjugate Gradient (gpcg) (Quadratic objective only)	bounds	yes	no
Model-based derivative free nonlinear least-squares (pounders)	yes	no	no
Semismooth – Feasibility-enforced (SSFLS)	complementarity	yes	yes
Semismooth – Feasibility not enforced (SSILS)	complementarity	yes	yes
Active-Set Semismooth – Feasibility-enforced (ASFLS)	complementarity	yes	yes
Active-Set Semismooth – Feasibility not enforced (ASILS)	complementarity	yes	yes
Linearly Constrained Lagrangian	pde		
Interior Point Method (ipm)	general	yes	yes

TAO Applications

What do you need to do for the User Routines?

You need to write C, C++, or Fortran functions that:

- Set the initial variable vector (optional)
- Compute the objective function value at a given vector
- Compute the gradient at a given vector
- Compute the Hessian matrix at a given vector (for Newton methods)
- Set the variable bounds (for bounded optimization)

TAO Applications

Write routines for computing the ojective function, gradient, and (if available) Hessian. An opaque data structure maybe used to store application-specific parameters or data.

```
typedef struct {
    PetscReal      epsilon; /* application parameter */
} Ctx;
```

The evaluation routines should then look like:

```
PetscErrorCode MyFunction(TaoSolver tao, Vec x,
                     PetscReal *fcnval, void *Ctx){
}
PetscErrorCode MyGradient(TaoSolver tao, Vec x, Vec g,
                     void *Ctx){
}
PetscErrorCode MyHessian(TaoSolver tao, Vec x, Mat *H,
                Mat *Hpre, MatStructure *flag, void *Ctx){
}
```

TAO Applications

```
Tao            tao;  /* TAO Optimization solver    */
UserContext    user; /* user-defined structure     */
Vec            x;    /* solution vector            */
Mat            H;    /* Hessian Matrix             */

PetscInitialize(&argc,&argv,0,0);
... Set up vectors, matrices, application data ...
TaoCreate(PETSC_COMM_WORLD,&tao);
TaoSetType(tao,"tao_lmvm");
TaoSetInitialVector(tao,x);
TaoSetObjectiveRoutine(tao,MyFunction,(void *)&user);
TaoSetGradientRoutine(tao,MyGradient,(void *)&user);
TaoSetHessianRoutine(tao,H,H,MyHessian,(void *)&user);
TaoSetFromOptions(tao);
TaoSolve(tao);
```

TAO Examples

TAO has some example applications (in C and Fortran) included in the source distribution for you to test the TAO installation, learn about TAO features, and reference for creating your own applications

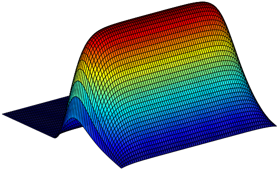
unconstrained	bound	least-squares	pde
eptorsion1.c	jbearing2.c	chwirut1.c	elliptic.c
eptorsion2.c	plate2.c	chwirut2.c	hyperbolic.c
eptorsion2f.F	plate2f.F	chwirut1f.F	parabolic.c
minsurf1.c		chwirut2f.F	
minsurf2.c			
rosenbrock1.c			
rosenbrock1f.F			

TAO Examples

Pressure in a Journal Bearing

$$\min\left\{ \int_{\mathcal{D}} \left\{ \tfrac{1}{2} w_q(x) \|\nabla v(x)\|^2 - w_l(x) v(x) \right\} \, dx : v \geq 0 \right\}$$

$$w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$$
$$w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$$
$$\mathcal{D} = (0, 2\pi) \times (0, 2b)$$

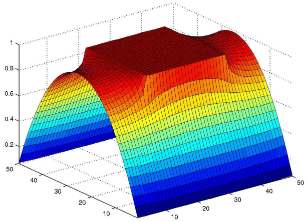


Number of active constraints depends on the choice of ϵ in $(0, 1)$.
Nearly degenerate problem. Solution $v \notin C^2$.

TAO

Minimal Surface with Obstacles

$$\min\left\{\int_{\mathcal{D}}\sqrt{1+\|\nabla v(x)\|^2}\,dx : v \geq v_L\right\}$$



Number of active constraints depends on the height of the obstacle. The solution $v \notin C^1$. Almost all multipliers are zero.

Toolkit for Advanced Optimization

As of PETSc 3.5 (June 30, 2014), TAO is included as part of the PETSc distribution http://www.mcs.anl.gov/petsc

The documention online includes installation instructions, a user's manual and a man page for every TAO subroutine. Please contact us with any questions or comments you have.

- petsc-maint@mcs.anl.gov