Workshop on the ACTS Toolkit

October 10–13, 2001

National Energy Research Scientific Computing Center

TAO – Toolkit for Advanced Optimization

Steve Benson, Lois Curfman McInnes, and Jorge J. Moré

http://www.mcs.anl.gov/tao

Mathematics and Computer Science Division
Argonne National Laboratory

Outline

- Optimization background
- TAO
  - Algorithms
  - Interface
  - Usage

What is Nonlinearly Constrained Optimization?

$$\min \{ f(x) : x_l \le x \le x_u ,\ c_l \le c(x) \le c_u \}$$

- Systems of nonlinear equations

$$\min \left\{ \tfrac{1}{2} \| r(x) \|^2 : x_l \le x \le x_u \right\} , \qquad r : \mathbb{R}^n \mapsto \mathbb{R}^n$$

- Nonlinear least squares

$$\min \left\{ \tfrac{1}{2} \| r(x) \|^2 : x_l \le x \le x_u \right\} , \qquad r : \mathbb{R}^n \mapsto \mathbb{R}^m , \quad m \ge n$$

- Bound-constrained optimization

$$\min \{ f(x) : x_l \le x \le x_u \}$$

The Ginzburg-Landau Model for Superconductivity

Minimize the Gibbs free energy for a homogeneous superconductor with a vector potential perpendicular to the superconductor.

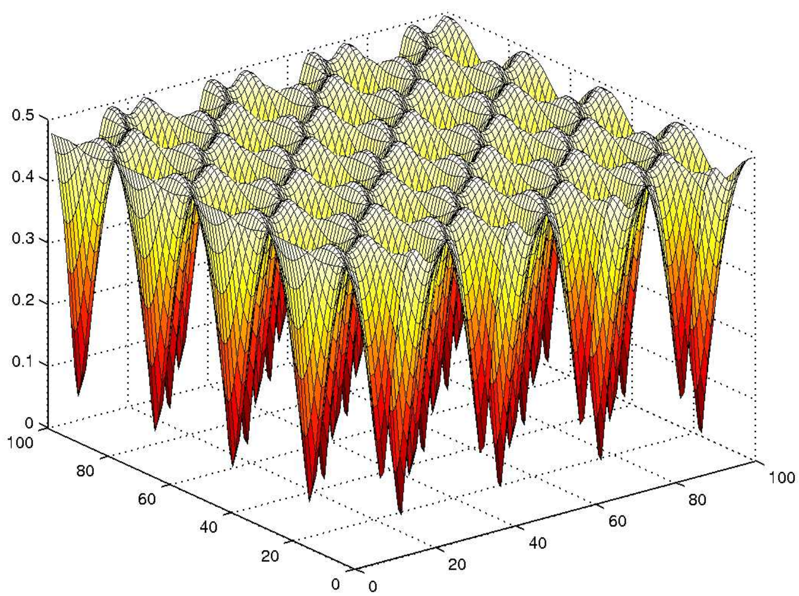
$$\int_{\mathcal{D}} \left\{ -|v(x)|^2 + \tfrac{1}{2}|v(x)|^4 + \|[\nabla - iA(x)]\, v(x)\|^2 + \kappa^2 \, \|(\nabla \times A)(x)\|^2 \right\} dx$$

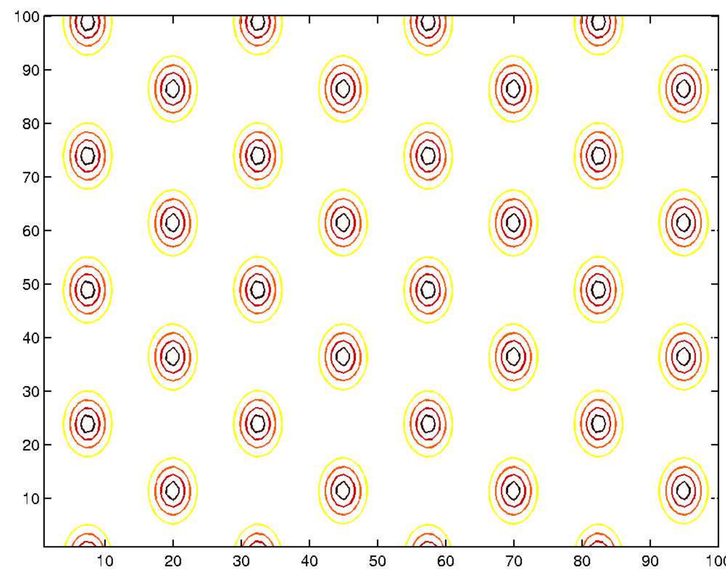
$v : \mathbb{R}^2 \to \mathbb{C}$ is the order parameter

$A : \mathbb{R}^2 \to \mathbb{R}^2$ is the vector potential

The Ginzburg-Landau Model for Superconductivity

Unconstrained problem. Non-convex function. Hessian is singular. Unique minimizer, but there is a saddle point.





Minimal Surface with Obstacles

Determine the surface of minimal area and given boundary data that lies above an obstacle.

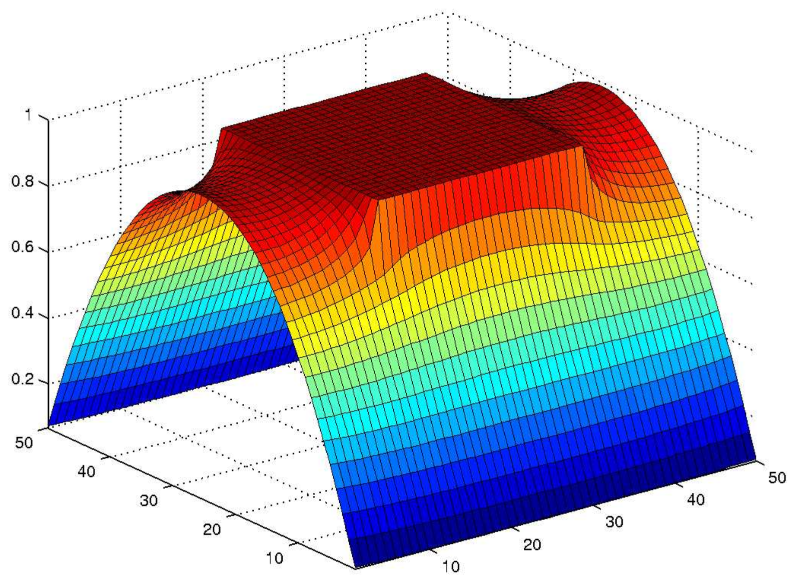
$$\min \{ f(v) : v \in K \}$$

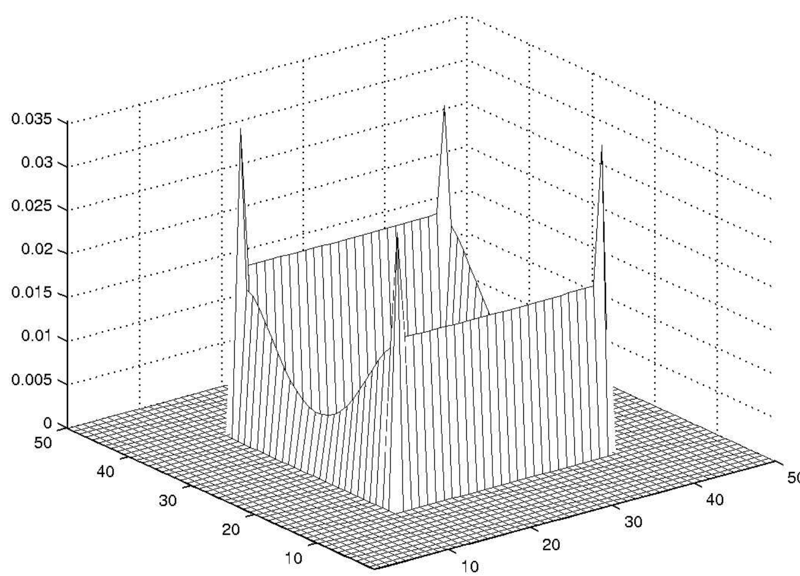
$$f(v) = \int_{\mathcal{D}} \sqrt{1 + \|\nabla v(x)\|^2} \, dx$$

$$K = \left\{ v \in H^1 : v(x) = v_D(x),\ x \in \partial D,\ v(x) \geq v_L(x),\ x \in \mathcal{D} \right\}$$

Minimal Surface with Obstacles

Bound constrained problem. Number of active constraints depends on the height of the obstacle. Almost all multipliers are zero.





Isomerization of α-pinene

Determine the reaction coefficients in the thermal isometrization of α-pinene from measurements by minimizing

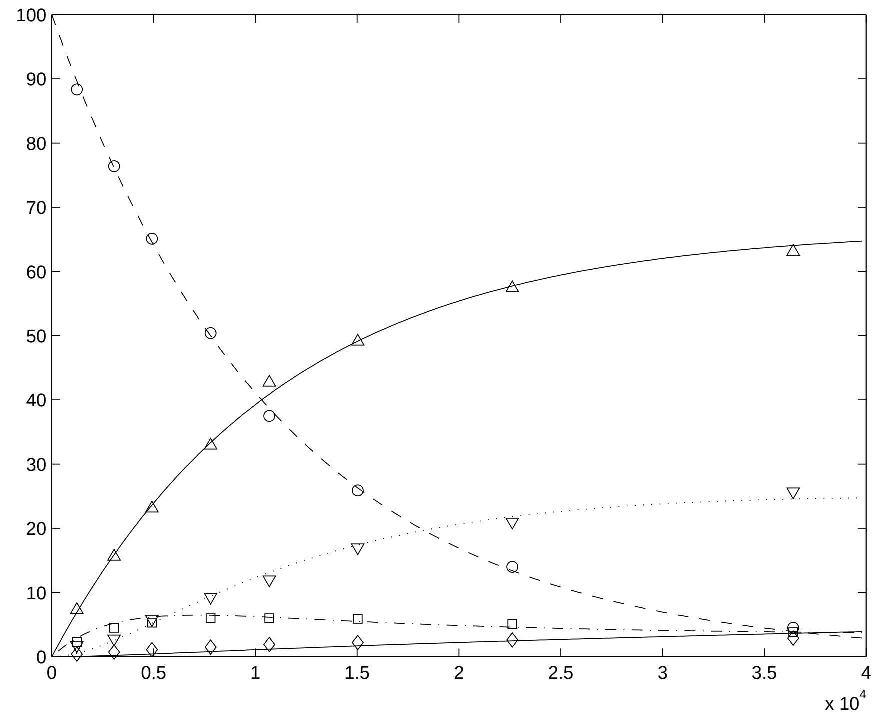
$$\sum_{j=1}^{8} \|y(\tau_j;\theta) - z_j\|^2,$$

where z_j are the measurements and

$$\begin{aligned}
y_1' &= -(\theta_1 + \theta_2) y_1 \\
y_2' &= \theta_1 y_1 \\
y_3' &= \theta_2 y_1 - (\theta_3 + \theta_4) y_3 + \theta_5 y_5 \\
y_4' &= \theta_3 y_3 \\
y_5' &= \theta_4 y_3 - \theta_5 y_5
\end{aligned}$$

Isomerization of α-pinene

Only equality constraints. Typical parameter estimation problem.



Optimization Toolkits

State-of-the-art in optimization software:

- ◇ Scattered support for parallel computations
- ◇ Little reuse of linear algebra software
- ◇ Minimal use of automatic differentiation software
- ◇ Few object-oriented optimization codes
- ◇ Nonlinear optimization problems with more than $10,000$ variables are considered large.

TAO

The process of nature by which all things change and which is to be followed for a life of harmony.

The Right Way

Toolkit for advanced optimization

- Object-oriented techniques
- Component-based interaction
- Leverage of existing parallel computing infrastructure
- Reuse of external toolkits

TAO Goals

- Portability
- Performance
- Scalable parallelism
- An interface independent of architecture

TAO Algorithms for Bound-Constrained Optimization

$$\min \{f(x) : x_l \le x \le x_u\}$$

- Conjugate gradient algorithms
- Limited-memory variable-metric algorithms
- Newton algorithms

You must supply the function $f : \mathbb{R}^n \mapsto \mathbb{R}$ and the gradient

$$\nabla f(x) = (\partial_i f(x))$$

For Newton methods you also need to supply the Hessian matrix.

$$\nabla^2 f(x) = (\partial_{i,j} f(x))$$

Conjugate Gradient Algorithms

$$x_{k+1} = x_k + \alpha_k p_k$$

$$p_{k+1} = -\nabla f(x_k) + \beta_k p_k$$

where α_k is determined by a line search.

Three choices of β_k are possible ($g_k = \nabla f(x_k)$):

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

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

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

Limited-Memory Variable-Metric Algorithms

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$$

where α_k is determined by a line search.

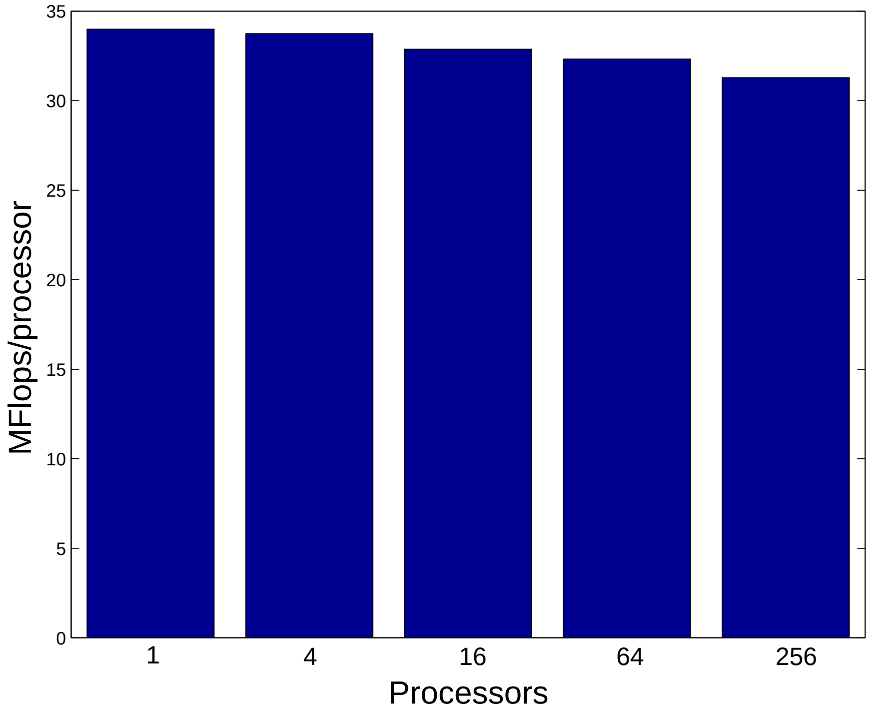
The matrix H_k is defined in terms of information gathered during the previous m iterations.

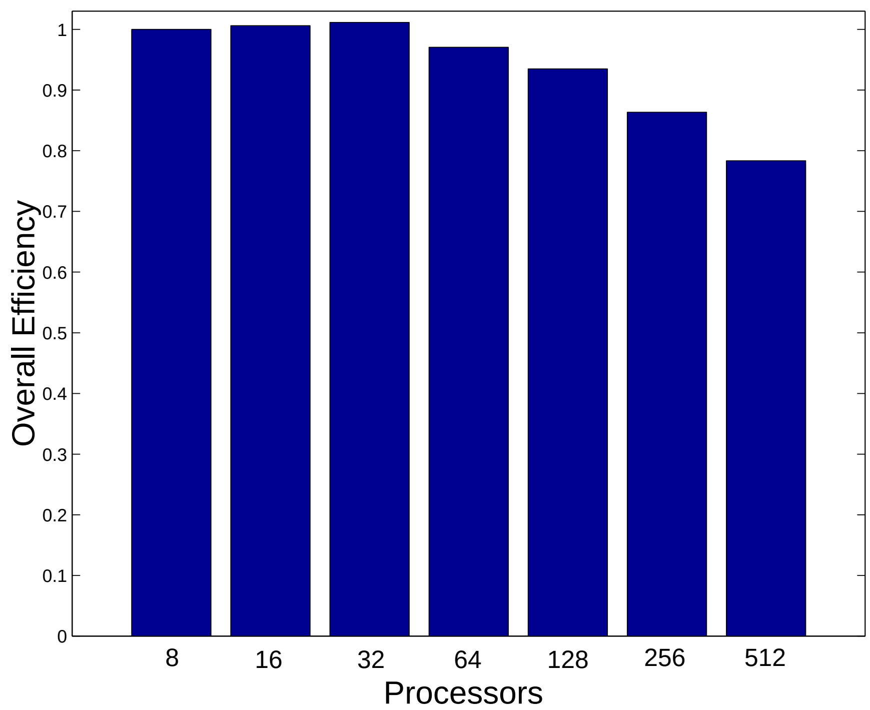
- H_k is positive definite.
- Storage of H_k requires $2mn$ locations.
- Computation of $H_k \nabla f(x_k)$ costs $(8m + 1)n$ flops.

TAO Performance: Plate Problem

Cray T3E (NERSC)

$n = 2.56 \cdot 10^6$ variables





TAO Algorithms (partial list)

- Unconstrained optimization
  - Conjugate gradient algorithms PR, FR, PR+
  - Levenberg-Marquardt method (alpha)
- Bound-constrained optimization
  - Limited-memory variable-metric algorithm
  - Trust region Newton method
- Linearly constrained optimization
  - Interior-point quadratic programming method (alpha)
- Nonlinearly constrained optimization
  - Work in progress

TAO Interface

```
TAO_SOLVER tao;                    /* TAO_SOLVER solver context */
Vec        x, g;                   /* solution and gradient vectors */
int        n;                      /* number of variables */
AppCtx     user;                   /* user-defined application context */
TaoVecPetsc *xx,*gg;

VecCreate(MPI_COMM_WORLD,n,&x);
VecDuplicate(x,&g);

TaoWrapPetscVec(x,&xx);
TaoWrapPetscVec(g,&gg);

TaoCreate(xx,''tao_lmvm'',0,MPI_COMM_WORLD,&tao);
TaoSetFunctionGradient(tao,&ff,gg,FunctionGradient,(void *)&user);
TaoSolve(tao);

TaoDestroy(tao);
```

Function Evaluation

```
typedef struct {          /* Used in the minimum surface area problem */
  int         mx, my;            /* discretization in x, y directions */
  Vec         Bottom, Top, Left, Right;            /* boundary values */
} AppCtx;

int FormFunction(TAO_SOLVER tao, TaoVec *xx, double* fcn,void *userCtx){
   AppCtx *user = (AppCtx *)userCtx;
   Vec X;

   TaoVecGetPetscVec(xx,&x);
   ...
   return 0;
}
```

The user sets this routine in the main program via

```
  info = TaoSetFunction(tao,&ff,FormFunction,(void *)&user);
```

Gradient Evaluation

```
  int FormGradient(TAO_SOLVER tao, TaoVec *xx, TaoVec *gg,void *userCtx){
     AppCtx *user = (AppCtx *)userCtx;
     Vec X,G;

     TaoVecGetPetscVec(xx,&x);
     TaoVecGetPetscVec(gg,&g);
     ...
     return 0;
}
```

The user sets this routine in the main program via

```
    info = TaoSetGradient(tao,gg,FormGradient,(void *)&user);
```

Alternatively, the user can supply the function and gradient evaluation in a single routine.

A Hessian evaluation routine can be supplied in a similar manner.

Convergence

Absolute tolerances specify acceptable errors in the optimality of the function and the constraints.

$$f(x) \leq f(x^*) + \epsilon_{fatol}$$

Relative tolerances specify the number of significant digits required in the solution and the constraints.

$$f(x) \leq f(x^*) + \epsilon_{frtol}|f(x^*)|$$

These tolerance can be changed

```
int TaoSetTolerances(TAO_SOLVER solver,double fatol,double frtol,
                                        double catol,double crtol)
```

TAO Basic Facilities

- TaoInitialize
- TaoFinalize
- TaoSetInitialVector
- TaoSetBounds
- TaoGetLinearSolver
- TaoGetIterationData
- TaoView

Parallel Functionality

The TAO interface is the same in a parallel environment, but the user must provide vectors with a parallel structure.

```
VecCreateMPI(MPI_COMM_WORLD,n,PETSC_DECIDE,&x);
VecDuplicate(x,&g);

TaoWrapPetscVec(x,&xx);
TaoWrapPetscVec(g,&gg);
info = TaoCreate(xx,''tao_lmvm'',0,MPI_COMM_WORLD,&tao);
info = TaoSetFunctionGradient(tao,ff,gg,FunctionGradient,(void *)&user);
info = TaoSolve(tao);

info = TaoDestroy(tao);
```

The user still provides the routines that evaluate the function and gradient. These routines do not have to be performed in parallel, but parallel evaluations usually improve performance.

Parallel Function Evaluation

```
typedef struct {                          /* For Minimum Surface Area Problem */
  int         mx, my;                    /* discretization in x, y directions */
  Vec         Bottom, Top, Left, Right;                  /* boundary values */
  DA          da;                        /* distributed array data structure */
} AppCtx;

int FormFunction(TAO_SOLVER tao, TaoVec *xx, double* fcn,void *userCtx){
   AppCtx *user = (AppCtx *)userCtx;
   Vec x;
   double f=0;
   TaoVecGetPetscVec(xx,&x);
   for (i=xs; i<xe; i++){
     for (j=ys; j<ye; j++){
        f += ...
     }
   }
   info = MPI_Allreduce(&f,fcn,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
   return 0;
}
```

TAO

www.mcs.anl.gov/tao

Version 1.2 (June 2001)

- Source Code
- Documentation
- Installation instructions
- Example problems
- Performance results
- Supported architectures