Lecture 4
Scientific Computing:
Optimization Toolbox
Nonlinear Equations, Numerical Optimization

Matthew J. Zahr

CME 292
Advanced MATLAB for Scientific Computing
Stanford University

14th April 2015

Announcement

- Lecture 4 of 8
  - Numerical Optimization (Optimization Toolbox)
- Homework 1 due today
- Homework 2 posted
  - Combined with Homework 3
  - Need to do 2 out of 6 on the combined assignment
  - Due in 2 weeks (April 28)

1. Definitions

2. Nonlinear System of Equations

3. Numerical Optimization
   - Optimization Solvers
   - Optimization Problems
   - Optimization Toolbox

Outline

1. Definitions
2. Nonlinear System of Equations
3. Numerical Optimization
   - Optimization Solvers
   - Optimization Problems
   - Optimization Toolbox

Scalar-valued function derivatives

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a sufficiently smooth function of n variables.

- Jacobian of f:
$$\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_1}(\mathbf{x}) & \cdots & \frac{\partial f}{\partial \mathbf{x}_n}(\mathbf{x}) \end{bmatrix}$$
- Gradient of f:
$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x})^T = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_1}(\mathbf{x}) & \cdots & \frac{\partial f}{\partial \mathbf{x}_n}(\mathbf{x}) \end{bmatrix}^T$$
- Hessian of f:
$$\left[\nabla^2 f(\mathbf{x})\right]_{ij} = \frac{\partial f^2}{\partial \mathbf{x}_i \partial \mathbf{x}_j}(\mathbf{x})$$

Vector-valued function derivatives

Let $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m$ be a sufficiently smooth function of n variables.

- Jacobian of $\mathbf{F}$:

$$\left[\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x})\right]_{ij} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{x}_j}(\mathbf{x})$$

- Gradient of $\mathbf{F}$:

$$\nabla \mathbf{F}(\mathbf{x}) = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x})\right)^T$$

- Hessian of $\mathbf{F}$:

$$\left[\nabla^2 \mathbf{F}(\mathbf{x})\right]_{ijk} = \frac{\partial \mathbf{F}_i^2}{\partial \mathbf{x}_j \partial \mathbf{x}_k}(\mathbf{x})$$

Outline

Problem Definition

Find $\mathbf{x} \in \mathbb{R}^n$ such that

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{1}$$

where $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ continuously differentiable, nonlinear function.

- Solution methods for (1) iterative, in general
  - Require initial guess and convergence criteria
- Solution for (1) not guaranteed to exist
- If solution of (1) exists, not necessarily unique
  - The solution found depends heavily on the initial guess

Example

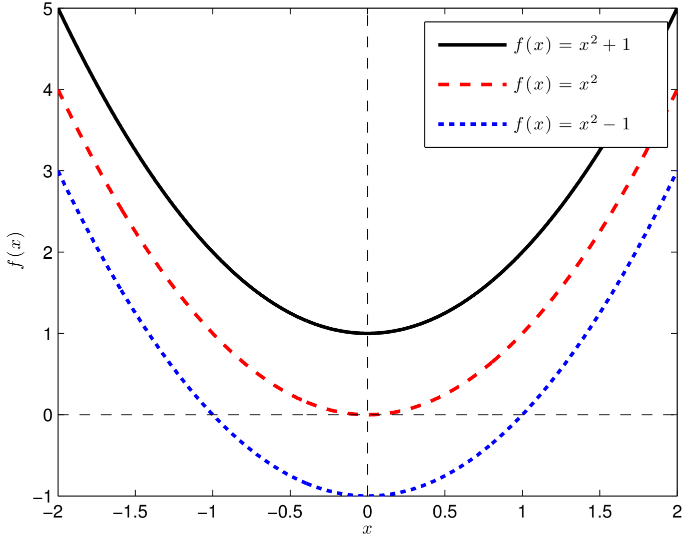


Figure: Non-existence and non-uniqueness of nonlinear equations

Scalar-Valued, Univariate Nonlinear Equation

- Equation (1) with $m = n = 1$ reduces to $f(x) = 0$
- Solution methods
  - Derivative-free methods
    - Only require evaluations of $f(x)$
    - Bisection, Fixed point iteration, etc
    - Demo: `bisection_demo.m`
  - Gradient-based methods
    - Requires function, $f(x)$, and gradient, $f'(x)$, evaluations
    - Newton's method
    - Secant method: use finite differences to approximate gradients instead of analytic gradients (only requires function evaluations)
- `fzero`
  - Derivative-free
  - Combines bisection, secant, and interpolation methods

Newton's Method

- Iterative method to solve $f(x) = 0$
- Expand $f(x)$ in a truncated Taylor series about current iteration

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \mathcal{O}((x - x_k)^2) = 0$$

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

- Demo: `newton_demo.m`

fzero

MATLAB builtin command for finding a root of a continuous, scalar-valued, univariate function

- `[x,fval,exitflag,output] = fzero(fun,x0,options)`
  - Inputs
    - `fun` - function handle containing scalar-valued function
    - `x0` - initial guess for root
    - `options` - structure containing options
  - Outputs
    - `x` - root found
    - `fval` - value of `fun` at `x`
    - `exitflag` - convergence flag
    - `output` - structure containing information about solver

Assignment

- Use `fzero` to solve $f(x) = x^2 - 1$
  - Start from any point away from ± 1
- Use `fzero` to solve $f(x) = x^2$
  - Start from any point away from 0
- Use `fzero` to solve $f(x) = x^2 + 1$
  - Start from any point away from 0

Vector-Valued, Multivariate Nonlinear Equations

- Solution methods for (1) in general case
  - Derivative-free methods
    - Requires function, $\mathbf{F}(\mathbf{x})$, evaluations
    - Fixed point iteration, Secant method, etc
  - Gradient-based methods
    - Requires function, $\mathbf{F}(\mathbf{x})$, and Jacobian, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x})$, evaluations
    - Newton-Raphson method
    - Gauss-Newton and Levenberg-Marquardt (nonlinear least squares: $\min ||\mathbf{F}(\mathbf{x})||$)
    - Can use finite difference approximations to gradients instead of analytic gradients (only requires function evaluations)
- `fsolve`
  - Gradient-based

fsolve

MATLAB builtin command for finding a root of a continuous, vector-valued, multivariate function

- `[x,fval,exitflag,output,jac] = fzero(fun,x0,options)`
    - Inputs/Ouputs - same as `fzero`
        - `fun` - function handle containing *vector*-valued function
- Algorithms
    - Standard trust region (default)
        - Requires square system $m = n$
    - Trust region reflexive, Gauss-Newton, Levenberg-Marquardt
        - Nonlinear least squares (use if f may not have root)
- By default uses finite differences to compute Jacobian
- To supply analytic Jacobian
    - `fun` return Jacobian as *second* output
    - `options.Jacobian = 'on'`

Example

- Derive *Jacobian* of

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_1 - 4x_1^2 - x_1x_2 \\ 2x_2 - x_2^2 - 3x_1x_2 \end{bmatrix}$$

- Solve $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ using `fsolve`
  - First *without* using Jacobian information
  - Then *with* Jacobian information
    `(options = optimset('Jacobian','on'))`
- How do the number of function/Jacobian evaluations compare in the two cases?
  - `output.funcCount`

Outline

Problem Definition

Consider the *general* optimization problem

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^{n_v}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{c}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{c}_{eq}(\mathbf{x}) = \mathbf{0} \\ & \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \tag{2}$$

- $\boldsymbol{\ell}, \mathbf{u} \in \mathbb{R}^{n_v}$ are box constraints
- $\mathbf{c} : \mathbb{R}^{n_v} \to \mathbb{R}^{n_{ineq}^{nl}}$ are nonlinear inequality constraints
- $\mathbf{c}_{eq} : \mathbb{R}^{n_v} \to \mathbb{R}^{n_{eq}^{nl}}$ are nonlinear equality constraints
- $\mathbf{A} \in \mathbb{R}^{n_{ineq}^{lin} \times n_v}$ define linear inequality constraints
- $\mathbf{A}_{eq} \in \mathbb{R}^{n_{eq}^{lin} \times n_v}$ define linear equality constraints

Feasbile Set

For (2), $\mathbf{x} \in \mathbb{R}^{n_v}$ is *feasbile* if $\mathbf{x}$ satisfies the constraints

$$\begin{aligned} \mathbf{A}\mathbf{x} &\leq \mathbf{b} \\ \mathbf{A}_{eq}\mathbf{x} &= \mathbf{b}_{eq} \\ \mathbf{c}(\mathbf{x}) &\leq \mathbf{0} \\ \mathbf{c}_{eq}(\mathbf{x}) &= \mathbf{0} \\ \boldsymbol{\ell} \leq \mathbf{x} &\leq \mathbf{u} \end{aligned} \tag{3}$$

- Define the *feasbile set*, $\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^{n_v} \mid \mathbf{x} \text{ satisfies } (3)\}$
- $\mathbf{x}$ is feasible if $\mathbf{x} \in \mathcal{F}$
- If there is no $\mathbf{x}$ such that (3) is satisfied, $\mathcal{F} = \emptyset$ and the problem is said to be *infeasible*
- If $f(\mathbf{x})$ is independent of $\mathbf{x}$, (2) is said to be a *feasibility* problem

Lagrangian

Write (2) concisely as

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^{nv}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}) \geq 0 \\ & \mathbf{h}(\mathbf{x}) = 0 \end{aligned} \tag{4}$$

Define the Lagrangian of (4) as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{x}) - \boldsymbol{\mu}^T \mathbf{h}(\mathbf{x}) \tag{5}$$

where $\boldsymbol{\lambda}$ are the Lagrange mulitpliers for the inequality constraints and $\boldsymbol{\mu}$ are the Lagrange multipliers for the equality constraints.

- Notice there is a Lagrange multiplier *for each constraint*, whether the constraint is a simple bound, general linear, nonlinear, equality, or inequality.

Optimality Conditions

- First-order necessary condition
  - Suppose $\mathbf{x}^*$ is a local solution to (4) plus additional assumptions. Then there are Lagrange multipliers, $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$, such that the following *Karush-Kuhn-Tucker* (KKT) conditions are satisfied

$$\begin{aligned} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= 0 \\ \mathbf{g}(\mathbf{x}^*) &\geq 0 \\ \mathbf{h}(\mathbf{x}^*) &= 0 \\ \boldsymbol{\mu}^* &\geq 0 \\ {\boldsymbol{\lambda}^*}^T \mathbf{g}(\mathbf{x}^*) + {\boldsymbol{\mu}^*}^T \mathbf{h}(\mathbf{x}^*) &= 0 \end{aligned} \qquad (6)$$

- Second-order necessary condition: $\nabla^2_{\mathbf{xx}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \succeq 0$
- Second-order sufficient condition: $\nabla^2_{\mathbf{xx}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \succ 0$

Types of Optimization Solvers: Function Information

- Derivative-free
  - Only function evaluations used
    - Brute force
    - Genetic algorithms
    - Finite difference computations of gradients/Hessians
  - Usually require *very large* number of function evaluations
- Gradient-based
  - Most popular class of techniques
  - Function and gradient evaluations used
    - Finite difference computations of Hessians
    - SPD updates to build Hessian approximations (BFGS)
- Hessian-based
  - Hessians usually difficult/expensive to compute, although often very sparse
  - Second-order optimality conditions
  - Function, gradient, and Hessian evaluations used

Types of Optimization Solvers

- Interior point methods
  - Iterates always strictly feasible
  - Use barrier functions to keep iterates away from boundaries
  - Sequence of optimization problems
- Active set methods
  - Active set refers to the inequality constraints *active* at the solution
  - Possibly infeasible iterates when constraints nonlinear
  - Minimize problem for given active set, then update active set based on Lagrange multipliers

Types of Optimization Solvers: Globalization

- Globalization: techniques to make optimization solver *globally convergent* (convergent to some *local* minima from *any* starting point)
  - Trust region methods
    - Determine step length
    - Define model problem whose solution will be the *step direction*
  - Line search methods
    - Determine search direction (Newton, Quasi-Newton, etc)
    - Find step length based on sufficient decrease criteria

Linear Program

Linear program ($\mathbf{f} \in \mathbb{R}^{n_v}$)

$$\begin{aligned} \underset{\mathbf{x} \in \mathbb{R}^{n_v}}{\text{minimize}} \quad & \mathbf{f}^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \tag{7}$$

MATLAB solver (`linprog`)

- Medium-scale
  - Simplex method
  - `options = optimset('LargeScale', 'off', 'Simplex', ... 'on')`
- Large-scale
  - Primal-dual interior point method
  - Default

Binary Integer Linear Program

Binary integer linear program ($\mathbf{f} \in \mathbb{R}^{n_v}$)

$$
\begin{aligned}
\underset{\mathbf{x} \in \mathbb{R}^{n_v}}{\text{minimize}} \quad & \mathbf{f}^T \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\
& \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\
& \mathbf{x}_j \in \{0, 1\} \text{ for } j \in \{0, \ldots, n_v\}
\end{aligned} \tag{8}
$$

MATLAB solver (`binitprog`)

- Linear program branch-and-bound
  - Sequence of LP-relaxation problems

Quadratic Program

Quadratic program ($\mathbf{H} \in \mathbb{R}^{n_v \times n_v}$, $\mathbf{f} \in \mathbb{R}^{n_v}$)

$$\begin{aligned} \underset{\mathbf{x} \in \mathbb{R}^{n_v}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{f}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \tag{9}$$

MATLAB solver (`quadprog`)

- Medium-scale
  - Active-set method with linear subproblems
- Large-scale
  - Trust region reflexive method
    - Newton-type method
    - Preconditioned CG to solve linear system

Unconstrained Optimization

$$\underset{\mathbf{x}\in\mathbb{R}^{nv}}{\text{minimize}} \quad f(\mathbf{x}) \tag{10}$$

MATLAB solvers

- MATLAB solvers (`fminunc`, `fminsearch`)
  - `fminsearch`
    - Derivative-free
    - Simplex search
  - `fminunc`
    - Quasi-Newton with linesearch (medium scale)
    - Trust region reflexive (large scale)

Linearly Constrained Optimization

- General

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^{nv}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \tag{11}$$

- Bounds only

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^{nv}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \tag{12}$$

MATLAB solvers (`fminbnd`, `fmincon`)

- `fminbnd`
  - Single variable optimization
- `fmincon`
  - Trust region reflexive method
  - Active-set method

Nonlinearly Constrained Optimization

$$
\begin{aligned}
\underset{\mathbf{x}\in\mathbb{R}^{nv}}{\text{minimize}} \quad & f(\mathbf{x}) \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\
& \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\
& \mathbf{c}(\mathbf{x}) \leq \mathbf{0} \\
& \mathbf{c}_{eq}(\mathbf{x}) = \mathbf{0} \\
& \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}
\end{aligned} \tag{13}
$$

- Solvers
  - Trust region reflexive
  - Sequential Quadratic Programming (SQP)
  - Interior Point (IP)

Optimization Toolbox Syntax

- Syntax for call to Optimization Toolbox
  - `[x,fval,exitflag,out,lam,grad,hess] = ...`
    `solver(f,x0,A,b,Aeq,beq,lb,ub,nlcon,opt)`
  - `[x,fval,exitflag,out,lam,grad,hess] = solver(problem)`
- Outputs
  - `x` - minimum found
  - `fval` - value of objective function at `x`
  - `exitflag` - describes exit condition of solver
  - `out` - structure containing output information
  - `lam` - structure containing Lagrange multipliers at `x`
  - `grad` - gradient of objective function at `x`
  - `hess` - Hessian of objective function at `x`

Optimization Toolbox Syntax

- Syntax for call to Optimization Toolbox
  - `[x,fval,exitflag,out,lam,grad,hess] = ...`
    `solver(f,x0,A,b,Aeq,beq,lb,ub,nlcon,opt)`
  - `[x,fval,exitflag,out,lam,grad,hess] = solver(problem)`
- Inputs
  - `f` - function handle (or vector for LP) defining objective function (and gradient)
  - `x0` - vector defining initial guess
  - `A,b` - matrix, RHS defining linear inequality constraints
  - `Aeq, beq` - matrix, RHS defining linear equality constraints
  - `lb,ub` - vectors defining lower, upper bounds
  - `nlcon` - function handle defining nonlinear contraints (and Jacobians)
  - `opt` - optimization options structure
  - `problem` - structure containing above information

Problem Structure

Instead of specifying many different inputs in a call to the optimizer, store inputs in `problem` structure

- `problem`
  - `objective`
  - `x0`
  - `Aineq`, `bineq`
  - `Aeq`, `beq`
  - `lb`, `ub`
  - `nonlcon`
  - `solver`
  - `options`
- Helpful in interacting with optimization GUI

Options Structure

- Each optimization solver has tunable options that can be specified with an `options` structure
  - `optimset <solver>` to return options available for particular solver, along with default values
  - Gives fine-grained control over optimization solver
    - Objective/Constraint functions
    - Maximum iterations
    - Tolerances
    - Difference intervals
    - Gradient, Hessian computations
    - Preconditioning
    - Scaling
    - Parallelization
    - Output and plotting functions

Optimization Functions

- Linear Programming
  - `linprog`
- Binary Integer Programming
  - `bintprog`
- Quadratic Programming
  - `quadprog`
- Unconstrained Optimization
  - `fminsearch`, `fminunc`
- Constrained Optimization
  - General Nonlinear
    - `fminbnd`, `fmincon`, `ktrlink`
    - TOMLAB (commercial software)
  - Least Squares
    - Linear: `lsqlin`, `lsqnonneg`
    - Nonlinear: `lsqcurvefit`, `lsqnonlin`
- Multiobjective Optimization
  - `fgoalattain`, `fminimax`

Objective Function

- Most optimization solvers require *scalar-valued* objective function
  - Exceptions: `fgoalattain`, `fminimax`, `fsolve`, `lsqcurvefit`, `lsqnonlin`
  - Must take optimization variables, $\mathbf{x}$, as input
  - Must return value of objective function $f(\mathbf{x})$
  - May return gradient, $\nabla f(\mathbf{x})$, and Hessian, $\nabla^2 f(\mathbf{x})$
    - User-supplied gradient only used if: `optimset('GradObj','on')`
    - User-supplied Hessian only used if: `optimset('Hessian','on')`
- For vector-valued objective functions, $\mathbf{F}(\mathbf{x})$
  - Input syntax identical to scalar-valued case
  - Instead of gradient, return *Jacobian* matrix, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$

Nonlinear Constraint Functions

- Nonlinear constraints are of the form $\mathbf{c}(\mathbf{x}) \leq 0$, $\mathbf{c}_{eq}(\mathbf{x}) = 0$
- Nonlinear constraint function must return $\mathbf{c}$ and $\mathbf{c}_{eq}$, even if they both do not exist (use `[]`)
- Call syntax
  - No derivatives: `[c,ceq] = constr_func(x)`
  - Derivatives: `[c,ceq,grad_c,grad_ceq] = constr_func(x)`
    - Derivatives must be in the form of *gradients*: grad_c(i,j) $= \frac{\partial \mathbf{c}_j}{\partial \mathbf{x}_i}$, grad_ceq(i,j) $= \frac{\partial \mathbf{c}_{eq\,j}}{\partial \mathbf{x}_i}$

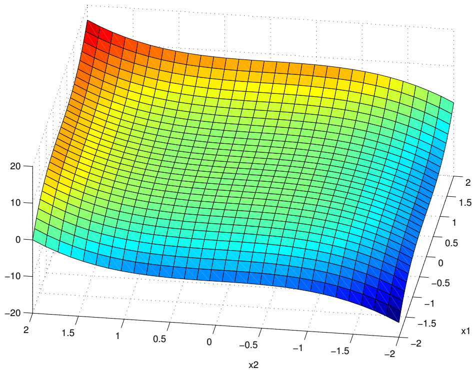
Assignment

Consider the following *nonlinearly constrained* optimization problem

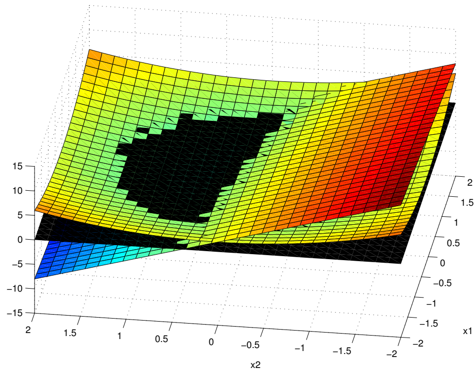
$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^2}{\text{minimize}} \quad & \mathbf{x}_1^3 + \mathbf{x}_2^3 \\ \text{subject to} \quad & \mathbf{x}_1 + 5\mathbf{x}_2 \geq 0 \\ & \mathbf{x}_1^2 + \mathbf{x}_2^2 \leq 2 \\ & -2 \leq \mathbf{x} \leq 2 \end{aligned} \tag{14}$$

- Derive derivative information for objective and *nonlinear* constraints
- Convert optimization problem into MATLAB compatible form
  - Linear inequality constraints: $\mathbf{A}\mathbf{x} \leq \mathbf{b}$
  - Nonlinear inequality constraints: $\mathbf{c}(\mathbf{x}) \leq \mathbf{0}$
- Solve using `fmincon`

Assignment



(a) Objective Functions



(b) Constraints

Figure: Assignment

Assignment

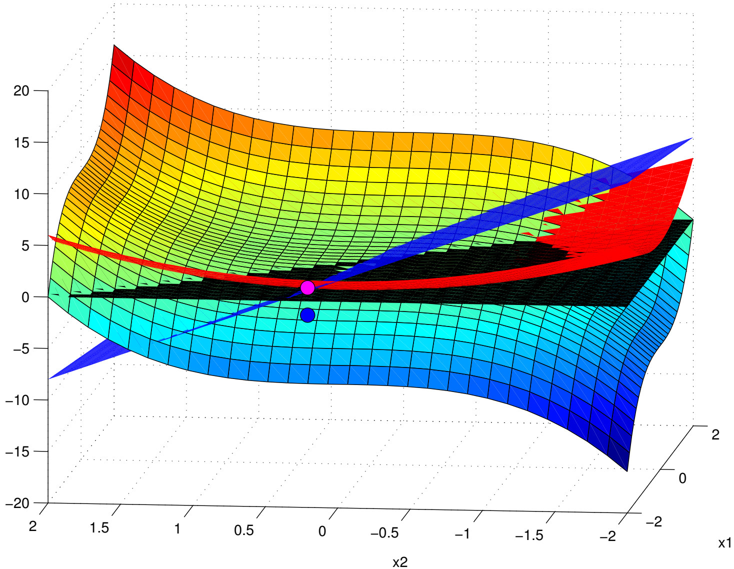


Figure: Assignment with solution

Optimization Toolbox GUI (`optimtool`)

- MATLAB's Optimization Toolbox comes equipped with GUI
  - `optimtool` to launch
  - User-friendly
  - Import `problem` or `options` from workspace
  - Generate code
  - Export optimization problem to workspace
- Brief Demo
- Power of MATLAB is scripting capabilities

Large-scale vs. Medium-scale

- *Large-scale* optimization algorithm
  - Linear algebra does not store or operate on *full* matrices
  - Sparse matrices and sparse linear algebra
  - Internal algorithms perserve sparsity or do not generate matrices (iterative solvers)
  - May be used for small problems
- *Medium-scale* optimization algorithms
  - Full matrices and dense linear algebra used
  - May require significant amount of memory and CPU time for large problems
  - Use to access functionality not implemented in *large-scale* case

Choosing an Algorithm

http://www.mathworks.com/help/optim/ug/choosing-a-solver.html

Parallel Computing and the Optimization Toolbox

- Parallel Computations available with commands `fmincon`, `fminattain`, `fminimax`
  - Start MATLAB pool of workers
  - Set `UseParallel` option to `'always'`
  - `parfor` to loop over multiple initial conditions
    - Attempt at global optimization
    - Embarrassingly parallel

```
>>  matlabpool open 2
>>  options = optimset('UseParallel','always');
```

Passing Additional Arguments to Functions

- An objective function or constraint may require additional arguments besides the optimization variables ($\mathbf{x}$)
- MATLAB's optimizers only pass optimization variables, $\mathbf{x}$, into functions
- Options for passing additional arguments to functions
    - Store variables in anonymous functions
    - Nested functions (variable sharing between parent/child)
    - Global variables