MATLAB-based Optimization: the Optimization Toolbox

Gene Cliff (AOE/ICAM - ecliff@vt.edu)
3:00pm - 4:45pm, Monday, 11 February 2013
..... FDI .....

AOE: Department of Aerospace and Ocean Engineering
ICAM: Interdisciplinary Center for Applied Mathematics

VirginiaTech

MATLAB's Optimization Toolbox

- Classifying Optimization Problems $\Leftarrow$
- A Soup Can Example
- *Intermezzo*
- A Trajectory Example
- 2^{nd} Trajectory Example: `fsolve`

VirginiaTech

Solver Categories

There are four general categories of Optimization Toolbox solvers:

- **Minimizers**

  This group of solvers attempts to find a local minimum of the objective function near a starting point x0. They address problems of unconstrained optimization, linear programming, quadratic programming, and general nonlinear programming.

- **Multiobjective minimizers**

  This group of solvers attempts to either minimize the maximum value of a set of functions (fminimax), or to find a location where a collection of functions is below some prespecified values (fgoalattain).

- **Least-Squares (curve-fitting) solvers**

  This group of solvers attempts to minimize a sum of squares. This type of problem frequently arises in fitting a model to data. The solvers address problems of finding nonnegative solutions, bounded or linearly constrained solutions, and fitting parameterized nonlinear models to data.

- **Equation solvers**

  This group of solvers attempts to find a solution to a scalar- or vector-valued nonlinear equation $f(x) = 0$ near a starting point x0. Equation-solving can be considered a form of optimization because it is equivalent to finding the minimum norm of f(x) near x0.

Generic Optimization Problem

$\min_x f(x_1, x_2, ..., x_n)$, subject to

equality constraints			inequality constraints		
$c\_eq_1(x_1, x_2, ..., x_n)$	$=$	0	$c_1(x_1, x_2, ..., x_n)$	$\leq$	0
$c\_eq_2(x_1, x_2, ..., x_n)$	$=$	0	$c_2(x_1, x_2, ..., x_n)$	$\leq$	0
	$\vdots$			$\vdots$	
$c\_eq_\ell(x_1, x_2, ..., x_n)$	$=$	0	$c_m(x_1, x_2, ..., x_n)$	$\leq$	0

simple bound constraints

$x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2, ..., n$

VirginiaTech

Classifying a Problem

- Identify your objective function as one of five types:
  - Linear
  - Quadratic
  - Sum-of-squares (Least squares)
  - Smooth nonlinear
  - Nonsmooth

- Identify your constraints as one of five types:
  - None (unconstrained)
  - Bound
  - Linear (including bound)
  - General smooth
  - Discrete (binary integer)

VirginiaTech

Problem classification table

Constraint Type	Objective Type				
	Linear	Quadratic	Least Squares	Smooth nonlinear	Nonsmooth
None	n/a (f = const, or min = $-\infty$)	`quadprog`, Theory, Examples	`\`, `lsqcurvefit`, `lsqnonlin`, Theory, Examples	`fminsearch`, `fminunc`, Theory, Examples	`fminsearch`, *
Bound	`linprog`, Theory, Examples	`quadprog`, Theory, Examples	`lsqcurvefit`, `lsqlin`, `lsqnonlin`, `lsqnonneg`, Theory, Examples	`fminbnd`, `fmincon`, `fseminf`, Theory, Examples	`fminbnd`, *
Linear	`linprog`, Theory, Examples	`quadprog`, Theory, Examples	`lsqlin`, Theory, Examples	`fmincon`, `fseminf`, Theory, Examples	*
General smooth	`fmincon`, Theory, Examples	`fmincon`, Theory, Examples	`fmincon`, Theory, Examples	`fmincon`, `fseminf`, Theory, Examples	*
Discrete	`bintprog`, *, Theory, Example	*	*	*	*

We focus on `fmincon`

VirginiaTech

MATLAB's Optimization Toolbox

- Classifying Optimization Problems
- A Soup Can Example $\Leftarrow$
- *Intermezzo*
- A Trajectory Example
- 2^{nd} Trajectory Example: `fsolve`

VirginiaTech

Soup Can Example (from MathWorks Training Docs)

We are to design a soup can in the shape of a right circular cylinder.We are to choose values for:

1. the diameter (d),
2. the height (h)

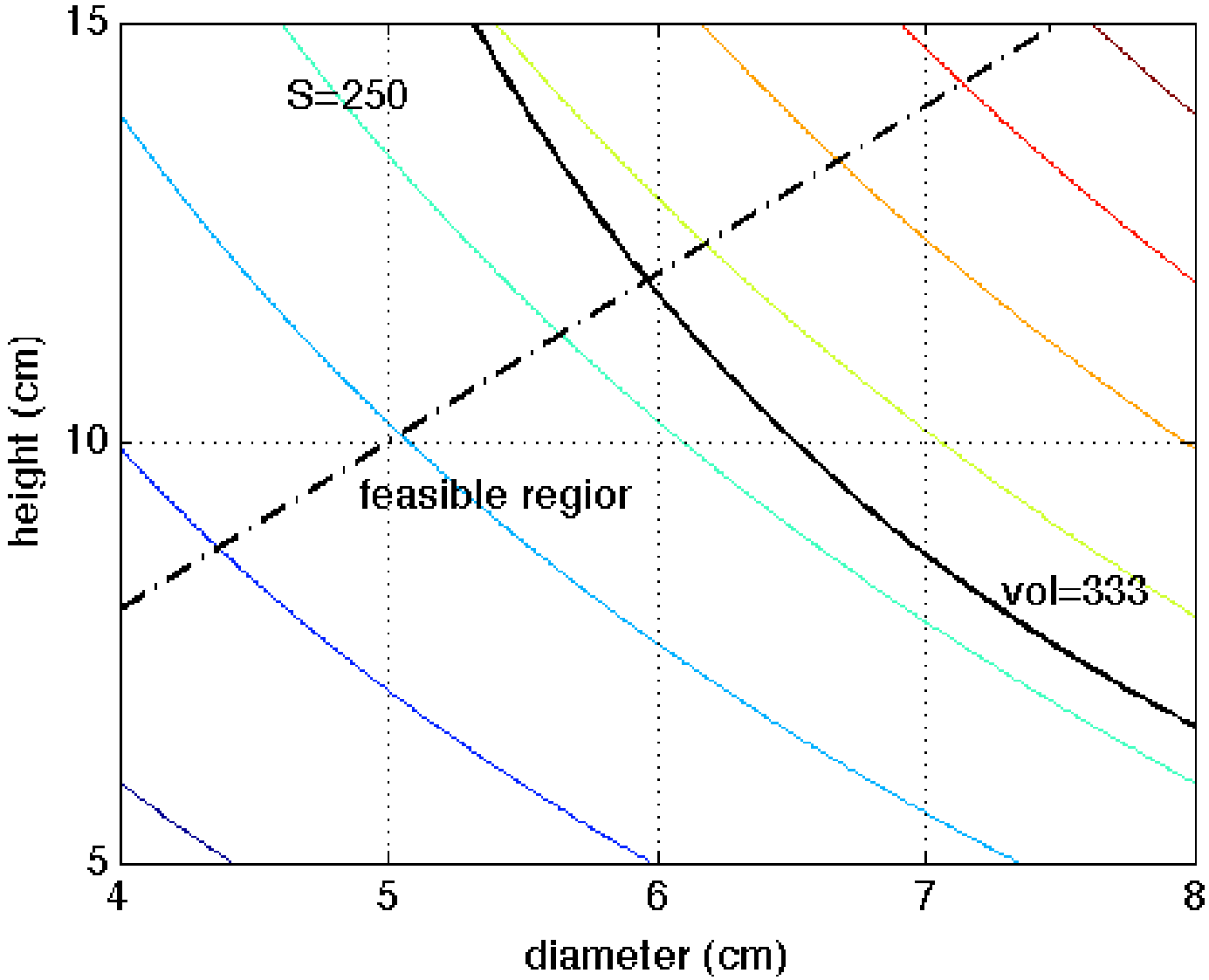
Requirements are:

- the volume ($\frac{\pi d^2}{4} h$) must be 333 cm^3
- the height can be no more than twice the diameter
- the cost is proportional to the surface area ($\frac{\pi d^2}{2} + \pi d h$), and should be minimized

Since the cost function and the volume constraint are nonlinear, we select `fmincon`.

VirginiaTech

Soup can design space



VirginiaTech

Soup can: cost function

```
function [val val_x] = cost_soup_can( x )
% Evaluate the cost function for the soup-can example
%
% x(1) - diameter of the can
% x(2) - height of the can
%
% area = 2*(pi*d^2)/4 + pi*d*h

    val =  pi*x(1)*(x(2) + x(1)/2);

% Evaluate the gradient
    if nargout > 1
        val_x =  pi*[x(1)+x(2); x(1)];
    end

end
```

VirginiaTech

Soup can: volume constraint

```
function [c ceq c_x ceq_x]= con_soup_can( x, volume)
%Evaluate the constraint for the soup-can example
% x(1) - diameter of the can
% x(2) - height of the can
%

   c = [];  % no nonlinear inequalities

   ceq = volume - (pi/4)*x(2)*x(1)^2; % volume = pi*d^2*h/4

% compute the Jacobians
   if nargout > 2
        c_x   = [];
        ceq_x = -(pi/4)*x(1)*[2*x(2); x(1)];
   end

end
```

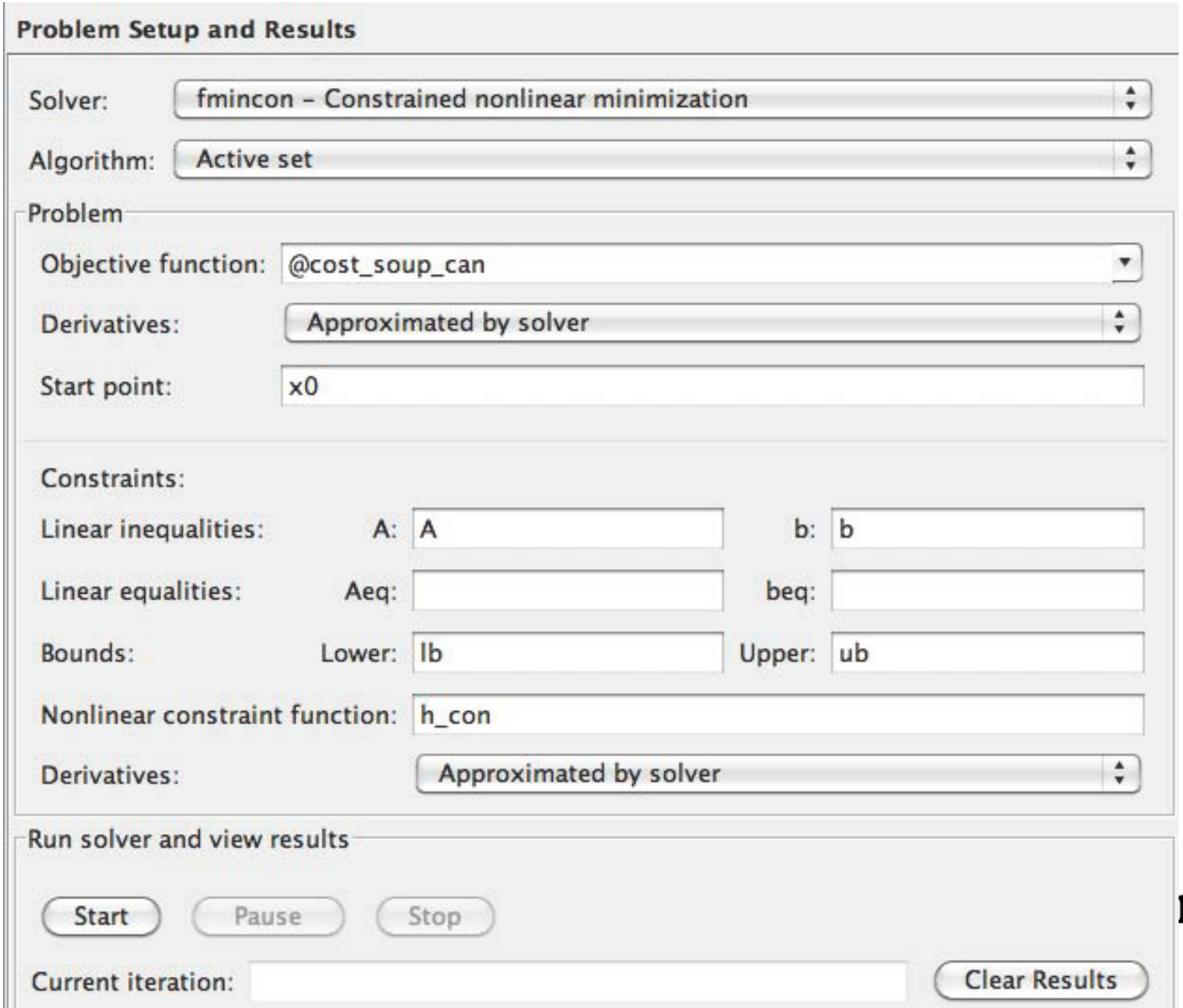
VirginiaTech

Soup can: set-up script

```
% Script to set up soup-can example
% We are to design a right-cylindrical (circular) can of a given volume
% and with minimum surface area (material cost). The height can be no more
% than twice the diameter


% volume = pi*d^2*h/4
% area   = 2*(pi*d^2)/4 + pi*d*h
%  h \le 2*d ==> -2*d + h \le 0

% In our optimization problem we have
%   x = [d ; h];
% The specified volume is 333 cm^3

% We have external function files
% cost_soup_can.m
% con_soup_can.m

%% define handle to the constraint function with the specified volume value
    volume = 333;
    h_con  = @(x) con_soup_can(x, volume);

% Arrays for the linear inequality
    A = [-2 1]; b = 0;

% lower/upper bounds
    lb = [ 4;  5];
    ub = [ 8; 15];

% initial guess
    x0 = [ 6; 10];
```

VirginiaTech

optimtool: soup can example



Command Window: soup can example

```
>> soup_can_2

 Iter F-count            f(x)  Max constraint  Line search steplength  Directional derivative  First-order optimality  Procedure
    0       3         245.044           50.26                                                                 Infeasibl
    1       6         247.138           35.41                       1                   3.93                     1.2
    2       9         265.113           1.713                       1                   17.3                    2.68
    3      12         265.948         0.05285                       1                   6.92                   0.798
    4      15          265.92         0.06939                       1                -0.0716                  0.0899
    5      18         265.956       0.0001174                       1                   6.49                 0.00326
    6      21         265.957       4.871e-08                       1                   0.53                6.88e-05  Hessian m

Local minimum possible. Constraints satisfied.

fmincon stopped because the predicted change in the objective function
is less than the selected value of the function tolerance and constraints
were satisfied to within the selected value of the constraint tolerance.

<stopping criteria details>

No active inequalities.
>>
```

VirginiaTech

MATLAB's Optimization Toolbox

- Classifying Optimization Problems
- A Soup Can Example
- *Intermezzo* $\Leftarrow$
- A Trajectory Example
- 2^{nd} Trajectory Example: `fsolve`

VirginiaTech

fmincon: choice of algorithms

- *'trust-region reflective'* requires you to provide a gradient, and allows only bounds or linear equality constraints, but not both. Within these limitations, the algorithm handles both large sparse problems and small dense problems efficiently. It is a large-scale algorithm, and can use special techniques to save memory usage, such as a Hessian multiply function. For details, see Trust-Region-Reflective Algorithm.
- *'active-set'* can take large steps, which adds speed. The algorithm is effective on some problems with nonsmooth constraints. It is not a large-scale algorithm.
- *'sqp'* satisfies bounds at all iterations. The algorithm can recover from NaN or Inf results. It is not a large-scale algorithm.
- *'Interior-point'* handles large, sparse problems, as well as small dense problems. The algorithm satisfies bounds at all iterations, and can recover from NaN or Inf results. It is a large-scale algorithm, and can use special techniques for large-scale problems.

VirginiaTech

Large-Scale *vs* Medium-Scale

An optimization algorithm is large scale when it uses linear algebra that does not need to store, nor operate on, full matrices. This may be done internally by storing sparse matrices, and by using sparse linear algebra for computations whenever possible. Furthermore, the internal algorithms either preserve sparsity, such as a sparse Cholesky decomposition, or do not generate matrices, such as a conjugate gradient method. Large-scale algorithms are accessed by setting the LargeScale option to on, or setting the Algorithm option appropriately (this is solver-dependent).

In contrast, medium-scale methods internally create full matrices and use dense linear algebra. If a problem is sufficiently large, full matrices take up a significant amount of memory, and the dense linear algebra may require a long time to execute. Medium-scale algorithms are accessed by setting the LargeScale option to off, or setting the Algorithm option appropriately (this is solver-dependent).

Don't let the name "large-scale" mislead you; you can use a large-scale algorithm on a small problem. Furthermore, you do not need to specify any sparse matrices to use a large-scale algorithm. Choose a medium-scale algorithm to access extra functionality, such as additional constraint types, or possibly for better performance.

VirginiaTech

fmincon: command line inputs

`x = fmincon(fun, x0, A, b, Aeq, beq, lb, ub, nonlcon, options)`

- `fun` - function handle for the cost function
- `x0` - initial guess for solution
- `A, b` - matrix, rhs vector for inequality constraints ($Ax \leq b$)
- `Aeq, beq` - matrix, rhs vector for equality constraints
- `lb, ub` - lower,upper bounds for solution vector
- `nonlincon` - function handle for the nonlinear inequality and equality constraints; `[c, ceq] = nonlincon(x)`
- `options` - structure of options for the algorithm

VirginiaTech

fmincon: additional outputs

`[x,fval,exitflag,output,lambda,grad,hessian]`

- `exitflag`
  - 1: First-order optimality measure was less than options.TolFun, and maximum constraint violation was less than .TolCon.
  - 0: Number of iterations exceeded options.MaxIter or number of function evaluations exceeded options.FunEv
- `output` -structure of data about performance of the algorithm
- `lambda` - structure of the *Lagrange multipliers*
- `grad` -gradient of the *Lagrangian*
- `Hessian` -Hessian of the *Lagrangian*

VirginiaTech

MATLAB's Optimization Toolbox

- Classifying Optimization Problems
- A Soup Can Example
- *Intermezzo*
- A Trajectory Example $\Leftarrow$
- 2^{nd} Trajectory Example: `fsolve`

VirginiaTech

Trajectory Example

We are to launch an object at speed v_0; we seek an initial elevation angle for maximum range. In the classical case with no drag, the best elevation is $\frac{\pi}{4}$. Suppose we have a simple drag force; $b\,v^2$?

- formulate an initial-value problem for the projectile motion
- the initial position and speed are given, the initial elevation angle ($\gamma(0)$) is unknown
- the final range ($x(t_\mathrm{f})$ to be maximized) occurs when the height returns to its initial value (final time (t_f) is unknown)

Since the cost function and the final height constraint are nonlinear functions of the unknowns, we select `fmincon`.

VirginiaTech

Trajectory: Setup and solve an IVP

```
function [ range, altitude] = trajectory(gam_0, t_f, param )
% Solve an IVP for the ballistic trajectory
% Evaluate the final altitude and range
%
% gam_0 is the initial flight-path angle (radians)
% t_f   is the final time (s)
%
% range/altitude are the final values
%
% param is a data structure
% param.b_coef is the drag coefficient
% param.grav   is the gravitational acceleration (m/s^2)
% param.vel_0  is the initial speed (m/s)

% anonymous function handle with specified parameters
    h_rhs    = @(t, z) ballistic_rhs(t, z, param.b_coef, param.grav);
    z_0      = [ 0 ; 0; param.vel_0; gam_0]; % set the initial state
    [~, Z]   = ode23(h_rhs, [0 t_f], z_0);   % solve the IVP
    range    = Z(end, 1);
    altitude = Z(end, 2);

end
```

Note that to evaluate the cost we need the range, and to evaluate the constraint we need the altitude.

Do we really have to solve the IVP twice to evaluate both ?

VirginiaTech

Trajectory: the RHS of the ODE system

```
function z_dot = ballistic_rhs(~, z, b_coef, grav)
% Evaluate rhs of eq. of motion for a ballistic object
%  z = [x, h, v, gamma]
%  x     - range
%  h     - altitude
%  v     - speed
%  gamma - flight-path angle

   sin_g = sin(z(4)); cos_g = cos(z(4));
   v     = max(z(3), 0.1); % guard against zero divisor
   z_dot = [ z(3)*cos_g; z(3)*sin_g;
             -b_coef*z(3)*z(3) - grav*sin_g;
             -grav*cos_g/v ];
end
```

VirginiaTech

MATLAB: ObjectiveandConstraints

```
function [ cost, nonlincon] = ObjectiveandConstraints(param)
% Encapsulates cost and constraint functions for fmincon
% cost and nonlincon are function handles
% param is a structure that encodes parameters for the cost/constraint fcns
% Initialize variables and make them available to the nested functions
  range      = []; altitude   = []; LastZ      = []; % initialize
  cost       = @objective;  nonlincon = @constraints;

% Nested functions
    function [val, val_Z] = objective(z)
      if ~isequal(z, LastZ) % update for this value
% Solve the IVP
        [range, altitude] = trajectory(z(1), z(2), param);
        LastZ = z;
      end
%   Evaluate cost
       val   = -range; % minimize the negative range
       val_Z = [];     % gradient not computed in this version
    end
%
    function [c, ceq  c_Z, ceq_Z] = constraints(z)
      if ~isequal(z, LastZ) % update for this value
% Solve the IVP
        [range, altitude] = trajectory(z(1), z(2), param);
        LastZ = z;
      end
%  Evaluate constraints
     c     = []; % no inequality constraints
     ceq   = altitude;
     c_Z   = []; % Jacobians not computed
     ceq_Z = [];
  end
end
```

ObjectiveandConstraints: insights

- Invoking `ObjectiveandConstraints` defines the function handles `cost` and `nonlincon`.
- Since the variables: `param, range, altitude, lastZ` are defined at the high-level, they are available to the *nested functions* `objective` and `constraints`.
- If $z \neq LastZ$ we solve the IVP and return `range` and `altitude`.
- If $z == LastZ$ we use the stored values of `range` and `altitude`.
- This approach is useful in cases wherein evaluating the cost/constraint functions requires an expensive calculation, such as the solution of an ODE/IVP or a PDE/BVP.
- Future documentation of the Optimization Toolbox will include this description.

VirginiaTech

fmincon:trajectory example

```
% Script to set parameters for and then run the max-range trajectory problem
%
% param is a structure of data for the problem
% param.b_coef is the drag coefficient
% param.grav   is the gravitational acceleration (m/s^2)
% param.vel_0  is the initial speed (m/s)

   param.b_coef =  0.1;
   param.grav   =  9.8;
   param.vel_0  = 25.0;

% define handles for functions evaluating the cost/contraints

    [ cost, nonlcon] = ObjectiveandConstraints(param);

% lower/upper bounds
     lb = [ 0    ;  0.5*param.vel_0/param.grav];
     ub = [ pi/4;  5*lb(2)];

% initial guess
     x0 = 0.5*(lb+ub);

%% set parameters and invoke fmincon
     OPT = optimset('fmincon');
     OPT = optimset(OPT, 'Algorithm'   , 'active-set', ...
                         'Display'     , 'iter', ...
                         'UseParallel', 'always');

%    x_star = fmincon(fun , x0, A , b ,Aeq,beq, lb, ub, nonlcon, options)
     x_star = fmincon(cost, x0, [], [], [], [], lb, ub, nonlcon, OPT);
```

MATLAB's Optimization Toolbox

- Classifying Optimization Problems
- A Soup Can Example
- *Intermezzo*
- A Trajectory Example
- 2^{nd} Trajectory Example: `fsolve` $\Leftarrow$

VirginiaTech

2nd Trajectory Example: `fsolve`

With the same dynamics as earlier, we now seek an initial elevation angle (γ_0)) and a final time (t_f) so that the trajectory ends at a specified point in the vertical plane ($x_\mathrm{f}, h_\mathrm{f}$).

- since the IVP solution depends on time, as well as on the initial elevation angle, we write the range and height functions as $x(t;\gamma_0)$ and $h(t;\gamma_0)$, respectively.
- we want to find values of t_f and γ_0 that lead to zero for the vector-valued function:

$$\begin{aligned} f_1(\gamma_0, t_\mathrm{f}) &\triangleq x(t_\mathrm{f}, \gamma_0) - x_\mathrm{f} \\ f_2(\gamma_0, t_\mathrm{f}) &\triangleq h(t_\mathrm{f}, \gamma_0) - h_\mathrm{f} \end{aligned}$$

- we use the the function `fsolve` from the Optimization Toolbox

VirginiaTech

Modified `trajectory` code

This version can return the time/state history `[T, Z]`

```
function [residual, T, Z] = trajectory(gam_0, t_f, param)
% Solve an IVP for the ballistic trajectory
% Evaluate the final altitude and range
%
% gam_0 is the initial flight-path angle (radians)
% t_f   is the final time (s)
%
% range/altitude are the final values
%
% param is a data structure
% param.b_coef is the drag coefficient
% param.grav   is the gravitational acceleration (m/s^2)
% param.vel_0  is the initial speed (m/s)
% param.x_f    is the specified target range  (m)
% param.h_f    is the specified target altitude (m)

% anonymous function handle with specified parameters
    h_rhs    = @(t, z) ballistic_rhs(t, z, param.b_coef, param.grav);
    z_0      = [ 0 ; 0; param.vel_0; gam_0]; % set the initial state
    if nargout == 1
       [~, Z]   = ode23(h_rhs, [0 t_f], z_0);   % solve the IVP
       residual = Z(end,1:2)' - [param.x_f; param.h_f];
    else
       [T, Z]   = ode23(h_rhs, [0 t_f], z_0);   % solve the IVP
       residual = Z(end,1:2)' - [param.x_f; param.h_f];
    end
end
% 'local' ballistic function goes here
```

VirginiaTech

fsolve: 2nd trajectory example

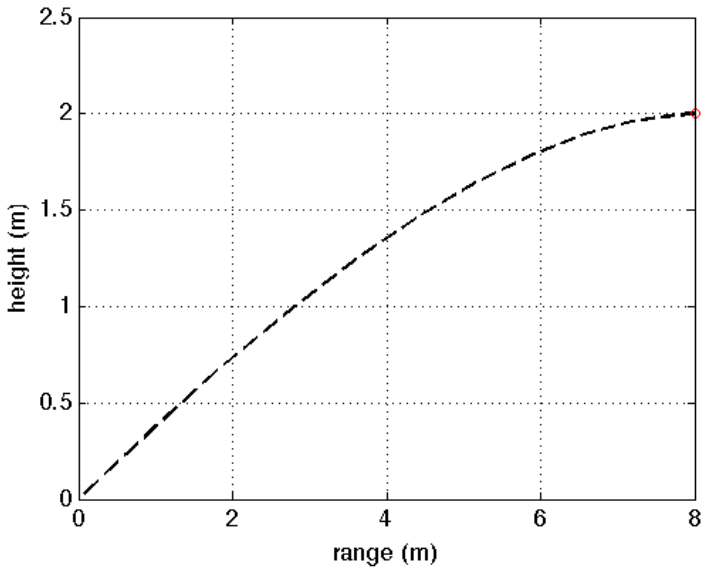
```
% Script to set parameters for and then run a trajectory target problem
%
% param is a structure of data for the problem
% param.b_coef is the drag coefficient
% param.grav   is the gravitational acceleration (m/s^2)
% param.vel_0  is the initial speed (m/s)
% param.x_f    is the specified target range  (m)
% param.h_f    is the specified target altitude (m)

  param.b_coef =  0.1;
  param.grav   =  9.8;
  param.vel_0  = 25.0;

  param.x_f    =  8.0;
  param.h_f    =  2.0;

% define handles for functions evaluating the cost/contraints

  f_hndl = @(x) trajectory(x(1), x(2), param);

% initial guess
    x0 = [pi/4; 0.5*param.vel_0/param.grav];

%% set parameters and invoke fsolve
    OPT = optimset('fsolve');
    OPT = optimset(OPT,   'Display'     , 'iter', ...
                          'UseParallel', 'always');

%   [x_star, f_val, exit_flag] = fsolve( FUN   ,  X0, OPTIONS)
    [x_star,   ~   ,   flag   ] = fsolve(f_hndl,  x0,  OPT);
```

fsolve: 2nd trajectory example

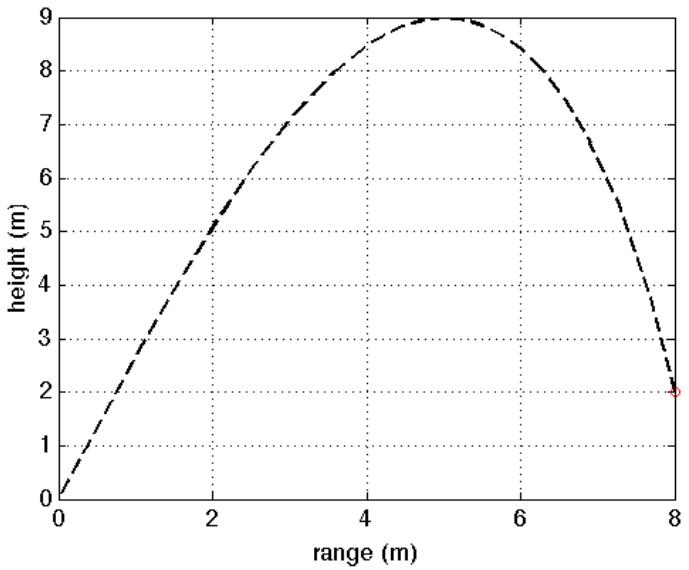
```
%   [x_star, f_val, exit_flag] = fsolve( FUN   ,  X0, OPTIONS)
    [x_star, ~, flag] = fsolve(f_hndl,  x0,  OPT);

    if flag == 1
        [ ~ , T, Z] = trajectory(x_star(1), x_star(2), param);
        figure
        plot(Z(:,1), Z(:,2), '--k', 'LineWidth', 2);
        hold on; grid on
        plot(param.x_f, param.h_f, 'ro')
        xlabel('range (m)'); ylabel('height (m)')
    else
        fprintf( 1, '\n flag = %02i \n', flag);
    end
```

2nd trajectory example: `fsolve`

Note that as in the zero-drag case, the problem has two solutions



Low trajectory



High trajectory

VirginiaTech

THE END

Please complete the evaluation form
`http://www.fdi.vt.edu/training/evals/`

Thanks

VirginiaTech

Backup - underlying ideas - problem w/o constraints

- Problem $\mathcal{P}_0$: Find $x^* \in \mathbb{R}^n$ to minimize a smooth function $f : \mathbb{R}^n \to \mathbb{R}$.
- We assume that f is twice continuously differentiable in the neighborhood of a solution.
- If x^* a minimizer for $\mathcal{P}_0$, then x^* is a stationary point for f, so that $(\nabla f)_{x^*} = 0 \in \mathbb{R}^n$, furthermore the Hessian of f, is positive semi-definite, $\left(\nabla^2 f\right)_{x^*} \geq 0$.
- Applying Newton's method to $\nabla f = 0$ we get the update $p_k = -\left(\nabla^2 f\right)^{-1}_{x_k} (\nabla f)_{x_k}, \quad x_{k+1} = x_k + p_k$
- Algorithms for $\mathcal{P}_0$ generate estimates for $\left(\nabla^2 f\right)$ based on computes changes in (∇f)
- The update is commonly generalized to $x_{k+1} = x_k + \alpha p_k$, where $\alpha > 0$ is a step-size.
- *Trust-region* methods minimize a quadratic approximation to f near x_k subject to a step size (trust-region radius).

VirginiaTech

Backup- underlying ideas -Newton update

- Write $(\nabla F)_{x+p} \approx (\nabla F)_x + (\nabla^2 F)_x \; p$
- Newton step - compute p so that $(\nabla F)_{x_k+p} = 0$
  $(\nabla^2 F)_{x_k} \; p = -(\nabla F)_{x_k}$
- Quasi-Newton update for $(\nabla^2 F)_{k+1}$
  $\underbrace{(\nabla^2 F)_{k+1}}_{\text{update this}} \; \tilde{p} = (\nabla F)_{x_k+\tilde{p}} - (\nabla F)_{x_k}$
- Q-N rules are commonly based on rank-two, least-change ideas
  $(\nabla^2 F)_{k+1} = (\nabla^2 F)_k + \underbrace{\Delta}_{\text{rank two}}$
- In the period 1960 - 1980 a great deal of work was done
  Davidon, Fletcher, Powell, Broyden, Goldfarb, Shanno

VirginiaTech

Backup- underlying ideas - problem w equality constraints

- Problem $\mathcal{P}_c$: Find $x^* \in \mathbb{R}^n$ to minimize a smooth function $f : \mathbb{R}^n \to \mathbb{R}$; subject to $g(x) = 0 \in \mathbb{R}^m$ where $g : \mathbb{R}^n \to \mathbb{R}^m$
- We assume that f, g are twice continuously differentiable in the neighborhood of a solution.
- If x^* is a minimizer for $\mathcal{P}_c$ and the Jacobian $J = \nabla g$ has full rank at x^* then there exists a vector $\hat{\lambda} \in \mathbb{R}^m$ such that x^* is a stationary point for the Lagrange function $\mathcal{L}(x) = f(x) + \langle \hat{\lambda}, g(x) \rangle$. Furthermore, x^* is a local minimizer for $\mathcal{L}$ in the null space of $J(x^*)$.
- The latter condition implies that the projected Hessian of $\mathcal{L}$ is positive semi-definite $Z^T \left(\nabla^2 \mathcal{L}\right)_{x^*} Z \geq 0$ where the columns of Z span the null-space of $J(x^*)$.

VirginiaTech

Backup- underlying ideas - problem w inequality constraints

- Problem $\mathcal{P}_{\mathrm{i}}$ - the constraints $_{\mathcal{J}} = k+1, ..., m$ are inequalities, $g_{\mathcal{J}} \leq 0$.
- Karush-Kuhn-Tucker theory implies that $\lambda_{\mathcal{J}} \geq 0$ (NB - in some formulations $\lambda_{\mathcal{J}} \leq 0$.)
- Many algorithms are based on an active-set strategy. Some set $\mathcal{A} \subset \{k+1, ..., m\}$ of inequalities are treated as equalities in a version of problem $\mathcal{P}_{\mathrm{c}}$
- At each (major) iteration the set $\mathcal{A}$ is adjusted:
  1. if $g_\ell > 0$ for some $\ell \in \mathcal{A}^{\mathrm{c}}$, then add ℓ to the active-set
  2. If $\lambda_\ell < 0$ for some $\ell \in \mathcal{A}$, then remove ℓ from the active-set

VirginiaTech