Fmincon

Spencer Ashley, Julie DiNitto, Jennifer Huling, Nichole Smith
Optimization Theory

April 9, 2013

Set Up:

As the name suggests, the command *fmincon* in MATLAB is used to find the minimum of a constrained nonlinear multivariable function:

$$\min f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

where b and beq are vectors, A and Aeq are matrices, $c(x)$ and $ceq(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. As stated, $f(x)$ is typically nonlinear, but $c(x)$ and $ceq(x)$ may also be nonlinear functions.

You can use *fmincon* through the Optimizaton Toolbox as well as in the command window. When using the command window, you have the following options,

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

where:

- f is the objective function,
- x_0 is the initial point (scalar, vector, or matrix valued),
- A, b are such that $A * x \leq b$,
- Aeq, beq are such that $Aeq * x = beq$,
- and lb, ub are the lower and upper bounds on x.

MATLAB suggests you use *fmincon* for general smooth constraints and one of either a linear, quadratic, or least squares objective function. You may also use *fmincon* for a smooth nonlinear objective function with either general smooth, linear, or bound constraints, but you may also use *fseminf* as well.

Application:

In designing structural columns which are subjected to exceptionally large loads (as on a jib crane), it is optimum (and cheaper) to build such a structure with minimum mass that can still support heavy loads (See **Figure 1** below). In one particular example, data was gathered and the objective function, as well as the constraints were set:

Objective function:

$$f(x) = (\text{Mass Density})(\text{Length})(\text{Cross-sectional Area}) = (7850)(5)(2\pi Rt)$$

where R is the mean radius of the tube, and t is the wall thickness.

Constraints:

- Stress Constraint: $\sigma \leq \sigma_a$ where σ is the equation describing the normal stress and σ_a is the allowable stress.
- Buckling load constraint: $P \leq P_{cr}$ where P is the actual load and P_{cr} is the equation describing the buckling load.
- Deflection (bending) constraint: $\delta \leq \Delta$ where δ is the equation describing the point of deflection and Δ is the allowable amount of deflection.
- Radius/thickness constraint: $\frac{R}{t} \leq 50$.
- Bounds on the variables: $0.01 \leq R \leq 1$ and $0.005 \leq t \leq 0.2$.

After coding m-files for invoking *fmincon*, defining the objective function, and defining the constraints, the output gives $R = 0.0537$ and $t = 0.005$, suggesting the minimum wall width and an appropriate width for the inner tube in order to minimize the mass of the column, but still satisfy and constraints dictated by the mass the column will be bearing.(Arora)

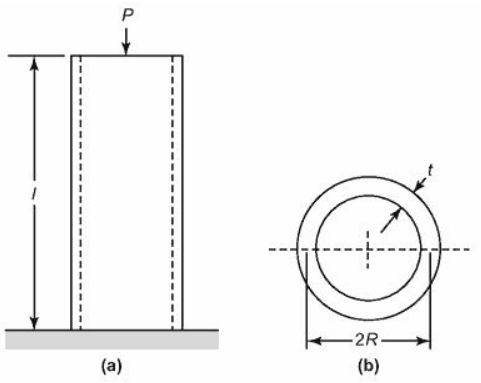


Figure 1: Configuration of vertical column.

Algorithms:

MATLAB is capable of using four types of *fmincon* algorithms: Trust-Region Reflective, Interior Point, Active Set, and SQP. Below is a table depicting the capabilities of each algorithm:

	Interior Point	SQP	Active Set	Trust-Region
Solves non-linear constraints*	X	X	X	
Satisfies bounds at iterations	X	X		
Recovers from NaN or Inf results	X	X		
Fast Speeds			X	
Allows for non-smooth constraints			X	
Requires user-supplied gradient				X
Cheap on memory		X		X

*It is important to note the Trust-Region method can ONLY use linear constraints OR bounds.

Trust Region Reflective:

The Trust Region Reflective method is currently the default algorithm MATLAB implements when running *fmincon*. It is best used for solving non-convex problems with variable bounds or linear equality constraints.

This method begins by formulating a two-dimensional trust region sub-problem. After defining a neighborhood, N, around an initial point, x_0, the algorithm approximates the objective function, f, with a more simple function q, which behaves similarly to f within N.

Now, solving the subproblem

$$\min_s \{\frac{1}{2} s^T H s + s^T g\}$$

(where H is the hessian, D is a diagonal scaling matrix, Δ is a positive scalar, and g is the user-supplied gradient of f evaluated at the current iterate point) such that $||Ds|| \leq \Delta$, yields the trial step-size of s. If $f(x+s) < f(x)$, then $x = x+s$ for the next step. Otherwise, the subproblem is solved with a smaller value of Δ.

The above algorithm is varied slightly for various types of constraints. If the problem involves linear equality constraints, an initial point x_0 is computed using a sparse least-squares step so that $Aeq * x_0 = beq$ and reduced preconditioned conjugate gradients are used to solve the trust region sub-problem in place of the typical PCG's. If the problem calls for least and upper bound constraints on x, these bounds are added by using a scalar

modified Newton step when solving for the step-size. Note that these methods do not allow for both an equality constraint and bounds.

Interior Point:

The interior point method, also called the barrier method, is used for minimizing linear and non-linear convex functions. Instead of moving along the function until finding a minimum, the interior point method starts inside the feasible set of the function and moves toward the boundary of the function as defined by the barrier.

The goal of the interior point method is to replace inequality constraints with easier to solve equality constraints. The function to be minimized, f, with an equality constraint, $h(x) = 0$, and inequality constraint, $g(x) \leq 0$, is changed to the following minimization problem:

$$\min_{x,s} f_\mu(x,s) = \min_{x,s} f(x) - \mu \sum_i \ln(s_i) \quad \text{subject to } h(x) = 0 \text{ and } g(x) + s = 0.$$

The term, $\mu \sum_i \ln(s_i)$, is known as the barrier function, and μ is a positive scaling factor. As $\mu \to 0$, the minimum of $f_\mu(x,s)$ approaches the minimum of $f(x)$. The term s_i is the slack variable.

At each iteration the method tries to take a direct step based on the Newton method and if that is not possible, it uses the conjugate gradient method. Each step must also reduce the merit function, $f_\mu(x,s) + v||(h(x), g(x) + s)||$, or another step is tried.

Active Set:

As mentioned previously, the Trust Region method is currently the MATLAB default. If all conditions for the trust-region-reflective algorithm are not satisfied, the active set algorithm is implemented. Because the Trust Region method is so limited in the types and number of bounds it can satisfy, the newest version MATLAB will automatically run Active Set as its default algorithm.

The active set algorithm is a medium-scale algorithm. Medium-scale algorithms internally create full matrices and use dense linear algebra. For large problems, this can take up a significant amount of memory, and may take a long time to execute.

The KKT equations are necessary conditions for optimality in a constrained optimization problem. The solution of the KKT equations form the basis to many nonlinear programming algorithms.

The active set algorithm uses Sequential Quadratic Programming (SQP) methods to guarantee super-linear convergence by accumulating second-order information regarding the KKT equations.

There are three main stages of the SQP implementation.

- Updating the Hessian Matrix: At each major iteration, a positive definite quasi-Newton approximation of the Hessian of the Lagrangian function is calculated using the BFGS method.
- Quadratic Programming Solution: At each major iteration, the inequality constraints are written as equality constraints, and this sub-problem is solved using a quadratic programming method. The solution to the QP subproblem is used to form a search direction for a line search procedure. This procedure has two phases. The first phase calculates a feasible point (if one exists). The second phase generates an iterative sequence of feasible points that converge to the solution.
- Line Search and Merit Function: The step length parameter defined by the steps above is determined so that a sufficient decrease in a merit function is obtained.

This algorithm requires a feasible point to start. If the current point from the SQP method is not feasible, you can find a feasible point by setting x to a value that satisfies the equality constraints. This value can be determined by solving an under or overdetermined set of linear equations formed from the set of equality constraints. The QP algorithm can be modified for LP problems by setting the search direction to the steepest descent direction at each iteration.

A nonlinearly constrained problem can often be solved in fewer iterations than an unconstrained problem using SQP. One of the reasons for this is that, because of the limits on the feasible area, the optimizer can make informed decisions regarding directions of search and step length.

SQP:

The Sequential Quadratic Programming algorithm uses the same SQP iterations used above in the Active Set algorithm, except SQP has certain restrictions at each iteration which slows it down comparatively.

Firstly, the SQP algorithm is different because each iterative step is taken within the region constrained by the bounds. This is helpful if you have a function which behaves poorly, is complex-valued, or is undefined outside the boundary. However, these bounds are not strict, so the iterative step can land on the boundary.

Secondly, unlike the Active Set algorithm, the SQP can recover and take a smaller step in the case that the value initially returned is undefined (NaN) or infinite (Inf).

Thirdly, the SQP algorithm uses a different set of equations to solve the subproblem needed to create each new iterate.

Lastly, while the Active Set Method finds a sufficient decrease in the merit function, the SQP algorithm tries to minimize the merit function in order to find the most feasible solution, these extra variables slow down the process, but allow the command to run in fewer iterations.

Examples:

In class, we minimized the unconstrained Rosenbrock function, but what happens when it is constrained to the unit disk: $x^2 + y^2 \leq 1$? Because the global minimum $(1, 1)$ is so difficult to converge to, the point $(0.7864, 0.6177)$ is given as a possible minimum when running *fmincon*. The Trust Region method cannot be used since the unit disk constraint is an inequality constraint. As seen in the figures below, the SQP algorithm uses the fewest iterations to converge.

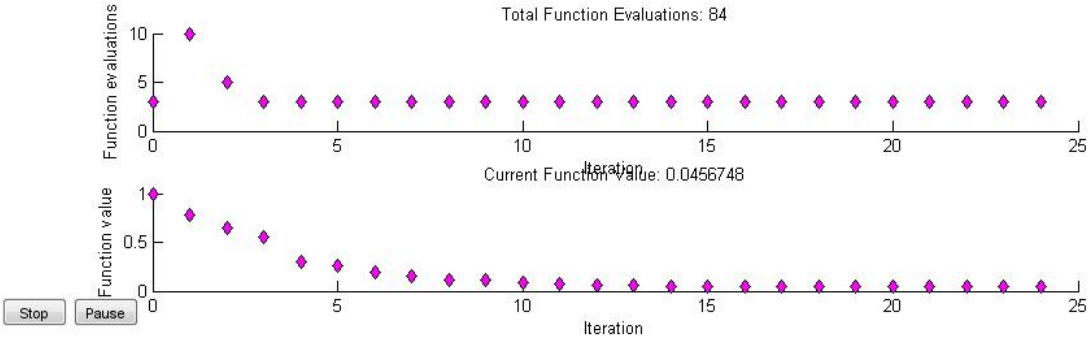


Figure 2: Interior Point Method- 84 Iterations

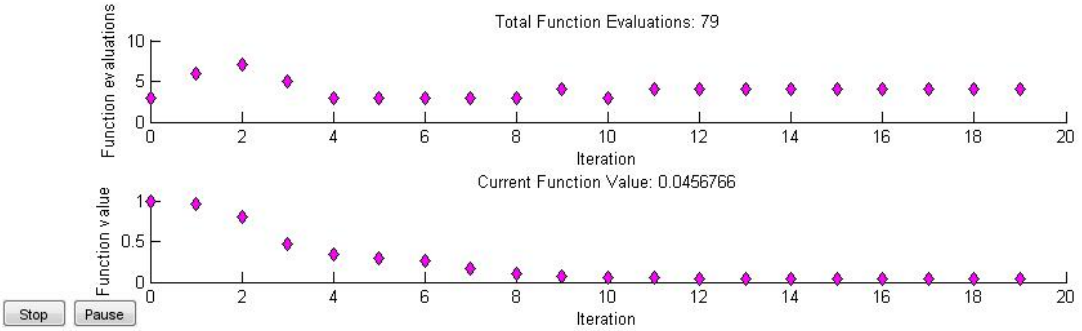


Figure 3: Active Set Method- 79 Iterations

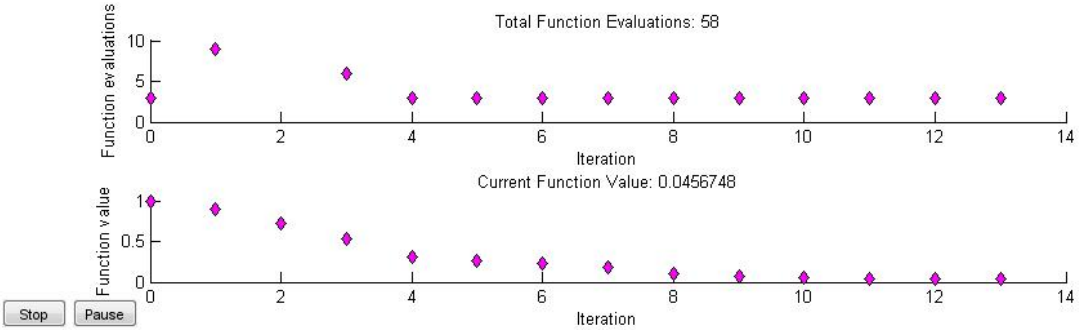


Figure 4: SQP Method- 58 Iterations

These algorithms are easily implemented in the Optimization Toolbox within MATLAB since each the Rosenbrock function and the unit disk constraints are pre-written m-files. They may also be written into the command window with the following code:

Interior Point Method:

```
function [c, ceq] = unitdisk(x)
c = x(1)^2 + x(2)^2 - 1;
ceq = [ ];

options = optimset('Display','iter','Algorithm','interior-point','PlotFcns',
@optimplotx);
[x,fval] = fmincon(@rosenbrock,[0 0],[],[],[],[],[],[],@unitdisk,options)
```

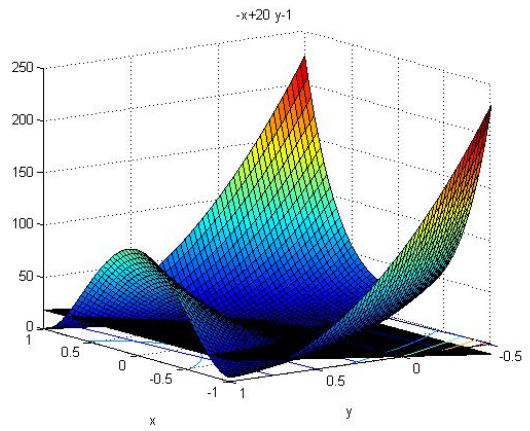
To implement either the active set algorithm or the SQP method, the only alteration needed to the above code is to replace "interior-point" with "active-set" or "sqp" respectively.

Trust Region Method:
To solve a constrained Rosenbrock function using the Trust Region Method, it is necessary to define a linear equality constraint. Consider the following equality constraint: $h(x) = -x + 20y - 1$. To write this constraint in terms MATLAB can configure, we define $A = [-1, 20]$ and $b = 1$. And run this in the Command Window with the following code:

```
f = @(x,y)100*(y - x^2)^2 + (1-x)^2;
Aeq = [-1,20];
beq = 1;
Options = optimset('Display','iter','GradObj','on','Algorithm','trust-region-reflective','T
[x,fval] = fmincon(@rosenbrock,[-1.3,0.2],[],[],Aeq,beq,[],[],[],Options)
```

Output after 7 iterations:

```
x = 0.2795    0.0640
fval = 0.5391
```



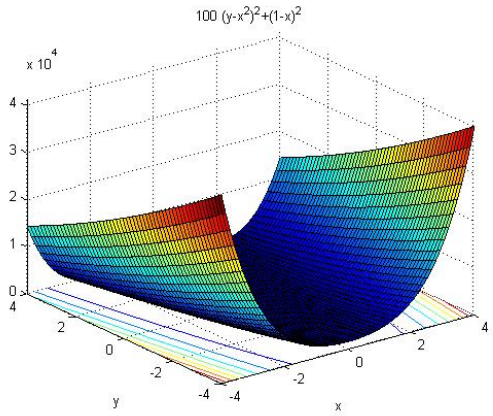
Plot of Rosenbrock function and equality constraint

It is also possible to solve a constrained Rosenbrock function using the Trust Region Method with variable bounds. Consider the bounds $-4 \leq x, y \leq 4$. The additional Command Window code is as follows:

```
lb = [-4,-4];
ub = [4,4];
[x,fval] = fmincon(@rosenbrock,[0,0],[],[],[],[],lb,ub,[],Options)
```

Output after 16 iterations:

```
x =1.0000    1.0000
fval =4.8482e-23
```



Plot of Rosenbrock function with variable bounds

Limitations:

- The command *fmincon* relies on the condition that your objective function and constraint functions are all continuous, as well as their first derivatives.
- The Trust-Region Reflective method does not allow for equal lower and upper bounds and suggest using the medium-scale method instead.
- Each of the Trust-Region Reflective and Interior Point methods require a different syntax for passing a Hessian.
- "For trust-region-reflective, the Hessian of the Lagrangian is the same as the Hessian of the objective function. You pass that Hessian as the third output of the objective function."
- "For interior-point, the Hessian of the Lagrangian involves the Lagrange multipliers and the Hessians of the nonlinear constraint functions. You pass the Hessian as a separate function that takes into account both the position x and the Lagrange multiplier structure lambda."

References

Arora, J.S. "Introduction to Optimum Design 3rd Ed.," Waltham, MA: Academic Press, 2012.

Coleman, T.F. and Li, Y. "A Reflecting Newton Method for Minimizaing a Quadratic Function Subject to Bounds on Some of the Variables," Advanced Computing Research Institute, Cornell University, 1992.

MATLAB R2013a Help Documents: fmincon, Choosing a Solver, Constrained Nonlinear Optimization Algorithms

Mesgari, F.C. "Application of MAG Index for Optimal Grasp Planning," 2011 IEEE International Conference on Mechatronics and Automation, Aug 2011.