Examples of Running Constrained Optimization Codes

CEE 201L. Uncertainty, Design, and Optimization
Department of Civil and Environmental Engineering
Duke University
Henri P. Gavin
Spring, 2022

Optimized Step-Size Random Search \textit{ORSopt}
Nelder-Mead with inequality constraints \textit{NMAopt}
Sequential Quadratic Programming \textit{SQPopt}

These three optimization functions all make use of the following five helper functions:
- set values for optimization algorithm parameters \textit{optim_options.m}
- average and coefficient of variation of augmented objective \textit{avg_cov_func.m}
- constrain a vector to a box of specified size \textit{box_constraint.m}
- plot the objective as a surface over two parameter values \textit{plot_surface.m}
- plot the convergence history of variables and max constraint \textit{plot_cvg_hst.m}

These .m-files and more may be found at \url{www.duke.edu/~hpgavin/m-files}.

Objective function

minimize $J = f(p, q) = (p - u)^2 + (q - v)^2 + wN$ ,

where $u$, $v$, and $w$ are constants, and $N$ is an uncertain value modeled by a standard-normal distribution. The coefficient $w$ determines the level of uncertainty in the objective. The unconstrained minimum of $F$ is at $(p^*, q^*) = (a, b)$

Constraint Inequalities

such that:  
\begin{align*}
g_1(p, q) &= ((p - 0.2)^2 + (q - 0.5)^2) - 0.3 \leq 0 \\
g_2(p, q) &= -((p + 0.5)^2 + (q - 0.5)^2)2.0 + 1.5 \leq 0 \\
\end{align*}

and $0 \leq (p, q) \leq 1$

![Figure 1. surfaces and contours of the objective, $f$ (red), and the constraints $g_1$ (green) and $g_2$ (blue).](image-url)
function [f, g] = optim_example_analysis(x, c)

% [f, g] = optim_example_analysis(x, c)

% INPUT DESCRIPTION DIMENSION
% x a vector of design variables n x 1
% c a vector or cell array of problem constants

% OUTPUT DESCRIPTION DIMENSION
% f the objective function corresponding to x 1 x 1
% g the constraint vector corresponding to x m x 1

% example of how to set up an optimization-oriented analysis

p = x(1); % description of design variable number 1, units
q = x(2); % description of design variable number 2, units
u = c(1); % description of constant number 1, units
v = c(2); % description of constant number 2, units
w = c(3); % description of constant number 3, units

f = (p - u)^2 + (q - v)^2 + w*randn; % the objective function

g = [( (p-0.2)^2 + (q-0.5)^2 )*1.0 - 0.3; % constraint g1
     -((p+0.5)^2 + (q-0.5)^2 )*2.0 + 1.5]; % constraint g2

% optim_example_opt.m
% script for carrying out a numerical optimization

x_lb = [0; 0]; % lower bound on permissible parameter values
x_ub = [1; 1]; % upper bound on permissible parameter values
x_init = x_lb + rand(2,1).*(x_ub - x_lb); % random initial guess, or ...
    % a specified initial guess

% some constants used in the analysis

% algorithmic parameters ...

% msglevel tolX tolF tolG MaxEvals Penalty Exponent nMax errF
options = [ 3 0.001 0.001 0.001 5e2 1.0 0.5 50 0.01 ];

% ... ORSopt or NMAopt or SQPopt ...
[x_opt, f_opt, g_opt, cvg_hst] = NMAopt('optim_example_analysis', x_init, x_lb, x_ub, options, c);

plot_cvg_hst( cvg_hst, x_opt, 100); % plot the convergence history

Convention for writing inequality constraints

By convention, all constraints are written as expressions compared to zero. And by convention, positive constraint values are “not ok”. So a constraint expressed as

\[ a \leq b \]

would be expressed in code in any number of possible comparisons to zero. The following three inequalities are mathematically equivalent to each other and to the inequality above.

\[ g = a - b \leq 0 \]
\[ g = a/b - 1 \leq 0 \]
\[ g = 1 - b/a \leq 0 \]

But each of these three choices can lead to different results when applied to numerical optimization methods. In general the second or third expressions are preferred, since they express the inequality in dimensionless form and are both compared to 1.

If \( a \) is a constant, use \( g = b/a - 1 \leq 0 \). In code this is written \( g = b/a - 1; \)

If \( b \) is a constant, use \( g = 1 - a/b \leq 0 \). In code this is written \( g = 1 - a/b; \)
Figure 2. ORSopt convergence on penalized objective function surface.

\[ f_{\text{opt}} = 1.0154 \times 10^0 \quad \max(g_{\text{opt}}) = -6.8954 \times 10^{-5} \]

Figure 3. ORSopt convergence histories.
Figure 4. NMAopt convergence on penalized objective function surface with no uncertainty in $f(x)$.

$$f_{\text{opt}} = 1.0166e+00 \quad \max(g_{\text{opt}}) = -5.9375e-03$$

Figure 5. NMAopt convergence histories.
Figure 6. SQPopt convergence on constrained objective function surface.

\[ f_{\text{opt}} = 1.0151 \times 10^0 \quad \text{max}(g_{\text{opt}}) = 2.3184 \times 10^{-4} \]

Figure 7. SQPopt convergence histories.
Optimization of Uncertain Systems

The design-based analysis of many systems usually involves imprecise or uncertain information. These can include uncertainties in the system’s operating environment (loading such as the monthly rainfall into a watershed, the peak hurricane wind loading on a skyscraper, or the impact velocity of a car crash), and uncertainties in the system’s intrinsic attributes (behavior such as watershed dynamics or material strength).

Probability distributions (e.g., the mean and standard deviation of a normal distribution), provide a quantification of these intrinsic and environmental uncertainties. A distribution of a system’s performance can be estimated by computing a statistical sample of the performance from samples of intrinsic and environmental variables, e.g., a sample of material strength values, and sample of peak wind speed values. This is done through the process of multiple re-analyses of a candidate design using different, but statistically representative, values of the uncertain quantities in each analysis. The mean and variability of the performance of a particular design (in the context of intrinsic and environmental uncertainties) can be estimated from a statistical sample of the performance metric (i.e., the objective function). Note that confidence in the statistical estimates improves with the number of re-analyses. In other words, estimates of statistics like the mean or the standard deviation improve with larger sample sizes. System analyses can take considerable computational time, and so it is desirable to limit the number of re-analyses, even at the cost of poorer estimates of the mean and standard deviation of the performance.

Now, comparing two candidate designs in terms of poor estimates of the mean and the standard deviation of the performance, could be misleading. The comparative assessment based on statistics estimated from a small sample of performance metrics depends entirely on the values in the sample, which might not be truly statistically representative if the sample size is small. Design A could appear to be better than design B based on a small sample, while a very large sample would indicate the opposite.

To optimize designs with uncertain performance, search methods (such as the Nelder-Mead method) can be more robust (less sensitive to sampling variability) than gradient-based methods (such as SQP).

The implementation of the Optimized Step Size Random Search in ORSopt and the Nelder-Mead method in NMAopt.m can handle problems with uncertain performance metrics. The use of these optimization methods for optimization with uncertainty can be tailored (or tuned) by changing three values in the set of options:

- Uncertain objective functions can be assessed in terms of their statistical properties (for example, their mean (average) and their standard deviation (or coefficient of variation)). Estimating the mean performance \( M_F(\mathbf{x}) \) and coefficient of variation of the performance \( C_F(\mathbf{x}) \) requires a sample of values of the performance metric \( f \) for the same set of design variables \( \mathbf{x} \). Given a sample \([f_1, f_2, ..., f_m]\), from \( m \) repeated evaluations of the objective function, the estimates of the mean and coefficient of variation can be computed as follows:

\[
M_F = \frac{1}{m} \sum_{i=1}^{m} f_i
\]

and

\[
C_F = \frac{1}{M_F} \left[ \frac{1}{m-1} \sum_{i=1}^{m} (f_i - M_F)^2 \right]^{1/2}
\]

Because the estimate of the mean and c.o.v. of \( F \) requires a sample of \( m \) evaluations, the total number of evaluations for the optimization will be larger than what is used in deterministic optimization (without uncertainty). To allow for this the maximum total number of function evaluations, \( \text{MaxEvals} \) (the fifth value in options) should be set larger.

- How many evaluations should be used in computing \( M_F \) and \( C_F \)? That is, what value for \( m \) should be specified? The answer to this question depends on the level of inherent variability in \( f(\mathbf{x}) \), on the desired statistical error \( e_F \) in the estimate of \( M_F \), and the level of confidence we require of our estimate. The inherent variability of \( f(\mathbf{x}) \) is estimated as \( C_F \).
The greater this inherent variability, the larger the sample size should be.
The smaller the desired error, the larger the sample size should be.
The larger the level of confidence in the estimate, the larger the sample size should be.

The equation for $m$ is:

$$m = \left[ \frac{z_{\alpha/2}}{e_F} \right]^2 C_F$$

where $z_{\alpha/2} = 1.645$ for a 90% confidence level on the estimate of $M_F$.

The ninth value in options, $mMax$, sets a limit on the sample size $m$ in order to restrict $m$ from becoming too large, (in cases with very large $C_F$ or very small desired $e_F$).

The tenth value in options, $errF$, sets the value of $e_F$, the desired estimation error for the mean, $M_F$.

Note that optimizing with small values of $errF$ and large values of $mMax$ could require many (many) function evaluations, but will represent the statistics of the objective function very well.

- For optimization problems with uncertain objective functions, it is sometimes desirable to recognize the uncertainty of the objective in the cost function. Optimization cost functions for uncertain objective functions are called risk measures.

The .m-file `avg_cov_func.m` takes care of computing $M_F$, $C_F$, and the risk measure to be optimized. A number of risk measures for stochastic optimization problems can be selected within `avg_cov_func.m`: the sample max, the sample average, the sample average plus the sample standard deviation (the 84th percentile of the objective function), or the sample average plus the sample standard deviation divided by $\sqrt{m}$ (the 84th percentile of the mean estimate).

```matlab
if m > 1
    F_risk = max(F); % worst-of-m values
    end
end
```

In the example script below, the uncertainty-level in this example problem is set by the coefficient $c_3$, so in this example we know in advance that the standard deviation of $F$ is equal to the value we use for $c_3$ (0.10). Setting $errF$ to 0.05 means that we desire an estimate for the mean of $F$ that is accurate to within ±5%, with a 90% confidence level. Using this information along with the equation for $m$, above, we will need a sample size of $m = (1.645 \times 0.10/0.05)^2 \approx 11$. We have set the maximum sample size, $mMax$, equal to 11.

Overall, the goal in setting values of $mMax$ and $errF$ is to use values that get the overall optimization to consistently converge to good solutions with the smallest number of function evaluations.

Note that:

- The risk measure used in this example is $F_{\text{risk}} = M_F(1 + C_F)$.
- The values of the optimized objective functions $f_{opt}$ shown in the figures are all very close to one another, even for the problem with added uncertainty.
- The uncertain objective function takes about eleven times as many function evaluations to converge as does the problem without uncertainty. This is related to the value of $mMax$ which is set to 11 in the uncertain case and is set to 1 in the case without uncertainty.
% optim_example_optU.m — Optimization with Uncertainty
% H.P. Gavin, Dept. Civil & Environ. Eng’g, Duke Univ., 2011–1–25

x_lb = [ 0 ; 0 ];               % lower bound on permissible parameter values
x_ub = [ 1 ; 1 ];               % upper bound on permissible parameter values

x_init = x_lb + rand(2,1).*(x_ub - x_lb); % random initial guess, or ...
x_init = [ 0.8 ; 0.8 ];         % a specified initial guess

cstnts = [ 0.8 ; 0.2 ; 0.10 ];  % other constants used in the cost function

% algorithmic parameters ...
options = [ 4 0.05 0.05 0.001 5e3 1.0 0.5 11.0 0.05 ];

% ... ORSopt or NMAopt or SQPopt ...
[x_opt, f_opt, g_opt, cvg_hst] = NMAopt('optim_example_analysis', x_init, x_lb, x_ub, options, cstnts);

plot_cvg_hst( cvg_hst, x_opt, 100);  % plot the convergence history
Figure 8. NMAopt convergence on penalized objective function surface with 15% uncertainty in $f(x)$.

Figure 9. NMAopt convergence histories.
ORSopt.m

% \[x_\text{opt}, f_\text{opt}, g_\text{opt}, \text{cvg\_hst}] = \text{ORSopt}(\text{func, } x_\text{init}, x_\text{lb}, x_\text{ub}, \text{options}, \text{consts})

% ORSopt: Optimized Step Size Randomized Search
% Nonlinear optimization with inequality constraints using Random Search
% minimizes \( f(x) \) such that \( g(x) < 0 \) and \( x_\text{lb} <= x_\text{opt} <= x_\text{ub} \).
% \( f \) is a scalar objective function, \( x \) is a vector of parameters, and
% \( g \) is a vector of constraints.
% INPUT
% ==
% func : the name of the matlab function to be optimized in the form
% \[ \text{objective, constraints} ] = \text{func}(x, \text{consts})
% x_\text{init} : the vector of initial parameter values \( (n \times 1) \)
% x_\text{lb} : lower bound on permissible values of the parameters, \( x \) \( (n \times 1) \)
% x_\text{ub} : upper bound on permissible values of the parameters, \( x \) \( (n \times 1) \)
% options : options(1) = 1 means display intermediate results
% 2 means display more intermediate results
% 3, 4, ... even more and more intermediate results
% options(2) = tol_\text{x} tolerance on convergence of parameters
% options(3) = tol_f tolerance on convergence of objective
% options(4) = tol_g tolerance on convergence of constraints
% options(5) = max\_evals limit on number of function evaluations
% options(6) = penalty on constraint violations
% options(7) = exponent on constraint violations
% options(8) = max number of function eval's in est. of mean \( f(x) \)
% options(9) = desired accuracy of mean \( f \) (as a c.o.v.)
% options(10) = 1 means stop when solution is feasible
% consts : an optional vector of constats to be passed to func(x, consts)

% OUTPUT
% ==
% x_\text{opt} : a set of parameters at or near the optimal value
% f_\text{opt} : the objective associated with the near-optimal parameters
% g_\text{opt} : the constraints associated with the near-optimal parameters
% cvg\_hst : record of x, f, g, function_count, and convergence criteria

NMAopt.m

% \[x_\text{opt}, f_\text{opt}, g_\text{opt}, \text{cvg\_hst}] = \text{NMAopt}(\text{func, } x_\text{init}, x_\text{lb}, x_\text{ub}, \text{options}, \text{consts})

% NMAopt: Nelder–Mead method for the nonlinear optimization with inequality constraints
% minimizes \( f(x) \) such that \( g(x) < 0 \) and \( x_\text{lb} <= x_\text{opt} <= x_\text{ub} \).
% \( f \) is a scalar objective function, \( x \) is a vector of parameters, and
% \( g \) is a vector of constraints.
% INPUT
% ==
% func : the name of the matlab function to be optimized in the form
% \[ \text{objective, constraints} ] = \text{func}(x, \text{consts})
% x_\text{init} : the vector of initial parameter values \( (n \times 1) \)
% x_\text{lb} : lower bound on permissible values of the parameters, \( x \) \( (n \times 1) \)
% x_\text{ub} : upper bound on permissible values of the parameters, \( x \) \( (n \times 1) \)
% options : options(1) = 1 means display intermediate results
% options(2) = tol_\text{x} tolerance on convergence of parameters
% options(3) = tol_f tolerance on convergence of objective
% options(4) = tol_g tolerance on convergence of constraints
% options(5) = max\_evals limit on number of function evaluations
% options(6) = penalty on constraint violations
% options(7) = exponent on constraint violations
% options(8) = max number of function eval's in est. of mean \( f(x) \)
% options(9) = desired accuracy of mean \( f \) (as a c.o.v.)
% options(10) = 1 means stop when solution is feasible
% consts : an optional vector of values that are not design variables

% OUTPUT
% ==
% x_\text{opt} : a set of parameters at or near the optimal value
% f_\text{opt} : the objective associated with the near-optimal parameters
% g_\text{opt} : the constraints associated with the near-optimal parameters
% cvg\_hst : record of x, f, g, function_count, and convergence criteria
SQPopt.m

```matlab
% [x_opt, f_opt, g_opt, cvg_hst, lambda, Hess]=SQPopt(func, x_init, x_lb, x_ub, options, c)
% SQPopt: Nonlinear optimization with inequality constraints using
% Sequential Quadratid Programming
% minimizes f(x) such that g(x)<0 and x_lb <= x_opt <= x_ub.
% f is a scalar objective function, p is a vector of parameters, and
% g is a vector of constraints.
% INPUT
% func : the name of the matlab function to be optimized in the form
% [objective, constraints] = func(x, c)
% x_init : the vector of initial parameter values (n x 1)
% x_lb : lower bound on permissible values of the parameters, x (n x 1)
% x_ub : upper bound on permissible values of the parameters, x (n x 1)
% options : options(1) = 1 means display intermediate results
% options(2) = tol_p tolerance on convergence of parameters
% options(3) = tol_f tolerance on convergence of objective
% options(4) = tol_g tolerance on constraint functions
% c : an optional vector of constants used by func(x,c)
% OUTPUT
% x_opt : a set of parameters at or near the optimal value
% f_opt : the objective associated with the optimal parameters
% g_opt : the constraints associated with the optimal parameters
% lambda : the set of Lagrange multipliers at the active constraints
% Hess : the Hessian of the objective function at the optimal point
```