An Example of Running Constrained Optimization Codes

multivarious.opt: ors, nms, sqp

Civil and Environmental Engineering
Duke University

Henri P Gavin
Spring 2026

Design variables, objective function and constraint inequalities

Design optimizations problems are conventionally defined in terms of n bounded design variables $\boldsymbol{v} = [v_1, \ ..., \ v_n]$, a scalar design objective, $f(\boldsymbol{v})$, and m design constraints $\boldsymbol{g}(\boldsymbol{v}) = [g_1(\boldsymbol{v}), ..., g_m(\boldsymbol{v})]$.

$$\underset{v_1,v_2,...v_n}{\text{minimize}} \ J = f(v_1, v_2, ..., v_n) \text{ such that } \begin{array}{rcl} g_1(v_1, v_2, \ldots, v_n) & \le & 0 \\ \vdots & & \vdots \\ g_m(v_1, v_2, \ldots, v_n) & \le & 0 \end{array} \text{ and } \begin{array}{ccccc} v_{\text{lb},1} & \le & v_1 & \le & v_{\text{ub},1} \\ \vdots & & \vdots & & \vdots \\ v_{\text{lb},n} & \le & v_n & \le & v_{\text{ub},n} \end{array}$$

where $[v_1, \ldots, v_n]$ is a vector of n *design variables*, $f(v_1, \ldots, v_n)$ is the scalar-valued *design objective* to be minimized, $g_i(v_1, \ldots, v_n) \le 0$ is the i-th out of m *design constraints* to be satisfied, and $v_{\text{lb},i} \le v_i \le v_{\text{ub},i}$ provide lower bounds (lb) and upper bounds (ub) on each of the design variables individually.

Convention for inequality constraints and constraint scaling

By convention, all constraints are expressed as an inequality compared to zero. And by convention, positive constraint values are "not ok." A standard (conventional) constraint function $g(\boldsymbol{v})$ for an inequality

$$p(\boldsymbol{v}) \le q(\boldsymbol{v})$$

is an inequality compared to zero

$$g(\boldsymbol{v}) = p(\boldsymbol{v}) - q(\boldsymbol{v}) \le 0$$

If the numerical values of one constraint equation are much much larger or smaller than numerical values of other constraint equations, it is helpful to scale the constraints so that they all have values roughly around the range (-1,1). To do so ...

If p is a positive constant, use	$g(\boldsymbol{v}) = 1 - q(\boldsymbol{v})/p \le 0.$	In code this is written	`g = 1 - q/p;`
If q is a positive constant, use	$g(\boldsymbol{v}) = p(\boldsymbol{v})/q - 1 \le 0.$	In code this is written	`g = p/q - 1;`
If p is a negative constant, use	$g(\boldsymbol{v}) = q(\boldsymbol{v})/p - 1 \le 0.$	In code this is written	`g = q/p - 1;`
If q is a negative constant, use	$g(\boldsymbol{v}) = 1 - p(\boldsymbol{v})/q \le 0.$	In code this is written	`g = 1 - p/q;`

An example optimization problem

This document demonstrates the use of three numerical algorithms for constrained optimization to solve the following constrianed optimization problem:

$$\min_{v_1,v_2} J = f(v_1,v_2) = (v_1-c_1)^2 + (v_2-c_2)^2 + c_3 N$$

$$\begin{aligned} \text{such that:} \quad & g_1(v_1,v_2) = \quad a_1((v_1-a_2)^2 + (v_2-a_3)^2) - a_4 \le 0 \quad , \\ & g_2(v_1,v_2) = \quad b_1((v_1-b_2)^2 \ + (v_2-b_3)^2) - b_4 \le 0 \quad , \\ \text{and} \quad & 0 \le v_i \le 1 \quad i \in (1,2) \end{aligned}$$

where $[a_1,\ a_2,\ a_3,\ a_4]$, $[b_1,\ b_2,\ b_3,\ b_4]$, and $[c_1,\ c_2,\ c_3]$, are numerical constants and N is an uncertain value modeled by a standard-normal distribution. The coefficient c_3 determines the level of uncertainty in the objective. In this example, the unconstrained minimum of $f(\boldsymbol{v})$ is at $(v_1^*, v_2^*) = (c_1, c_2)$.

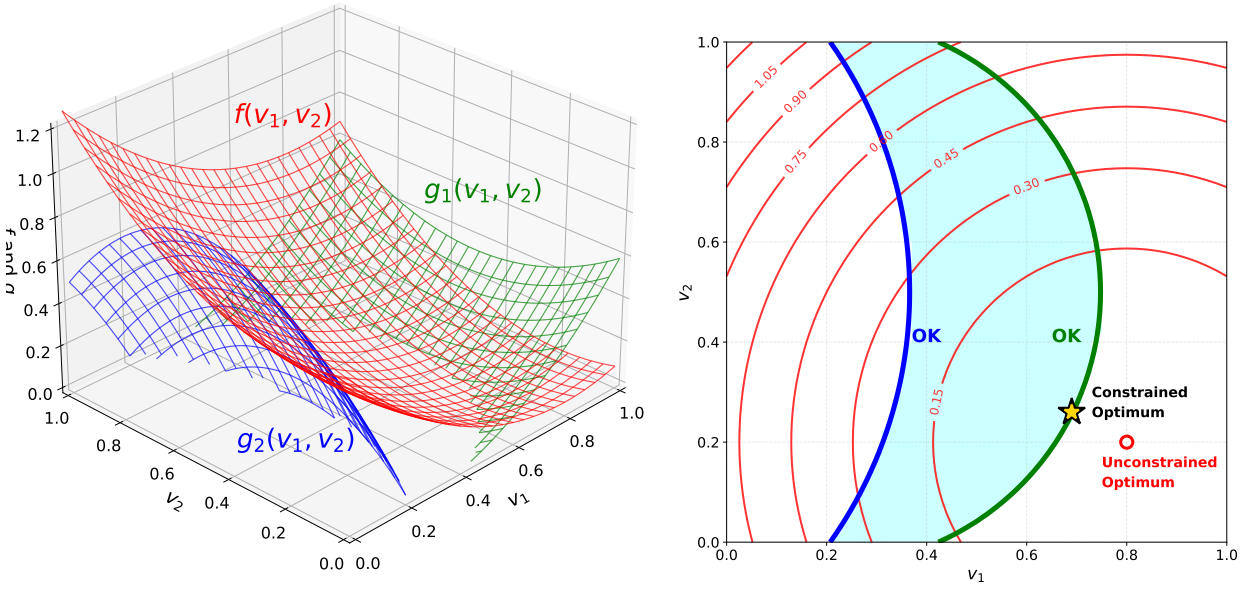


Figure 1.
Surfaces and contours of the objective, $f(\boldsymbol{v})$ (red), and the constraints $g_1(\boldsymbol{v})$ (green) and $g_2(\boldsymbol{v})$ (blue).
Constant values:
$\boldsymbol{a} = [1.0,\ 0.2,\ 0.5,\ 0.3]$,
$\boldsymbol{b} = [-2.0,\ -0.5,\ 0.5,\ -1.5]$ and
$\boldsymbol{c} = [0.8,\ 0.2,\ 0.0]$
(max headroom)

Three algorithms for constrained optimization

The numerical solution to an optimization problem depends on the algorithm used to obtain the solution. For some optimization problems, the choice of the algorithm can make a significant difference.

The constrained optimization algorithms provided by the `multivarious` library are:

Optimized Random Search	`ors.py`
Nelder-Mead Simplex	`nms.py`
Sequential Quadratic Programming	`sqp.py`

The three optimization methods are used with the similar function calls.

```
v_opt, f_opt, g_opt, cvg_hst, _, _     = ors( func, v_init, v_lb, v_ub, opts, C )
v_opt, f_opt, g_opt, cvg_hst, _, _     = nms( func, v_init, v_lb, v_ub, opts, C )
v_opt, f_opt, g_opt, cvg_hst, lmda, H = sqp( func, v_init, v_lb, v_ub, opts, C )
```

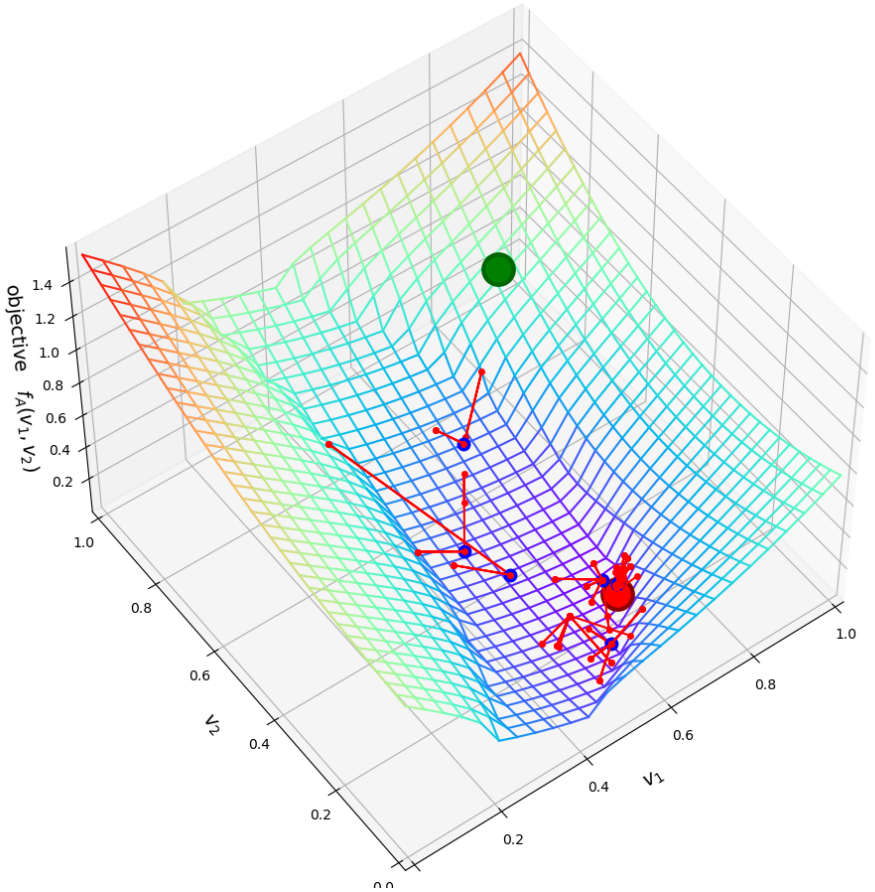
INPUT	
`func`	the name of the function to be optimized in the form `[objective, constraints] = func(v,C)`
`v_init`	the initial guess for the design variable values, $\boldsymbol{v}$
`v_lb`	lower bound on the design variables, $\boldsymbol{v}$
`v_ub`	upper bound on the design variables, $\boldsymbol{v}$
`opts`	an optional array or list of algorithmic options
	`opts[0]` = `msg` level of displayed intermediate information
	`opts[0] = 0`: display no intermediate results
	`opts[0] = 1`: display a synopsis of results at each iteration
	`opts[0] = 2`: display comprehensive results at each iteration
	`opts[0] = 3`: plot the convergence with respect to variables `opt[10]` and `opt[11]`
	`opts[1]` = `tol_x` tolerance on convergence of design variables
	`opts[2]` = `tol_f` tolerance on convergence of the design objective
	`opts[3]` = `tol_g` tolerance on convergence of constraints
	`opts[4]` = `max_evals` limit on number of function evaluations
	`opts[5]` = `penalty` on constraint violations
	`opts[6]` = `exponent` on constraint violations
	`opts[7]` = `m_max` max number of function evaluations to estimate the mean of $f(\boldsymbol{v}, \boldsymbol{C})$
	`opts[8]` = `cov_F` desired accuracy of the estimate of the mean of $f(\boldsymbol{v}, \boldsymbol{C})$ (as a c.o.v.)
	`opts[9]` = `1`: stop when the solution is feasible j
`C`	an optional array, list, set, simple name space, tuple or named tuple of constants used by `func(v,C`
OUTPUT	
`v_opt`	a set of design variables at or near the optimal value
`f_opt`	the objective associated with the optimal design variables
`g_opt`	the constraints associated with the optimal design variables
`cvg_hst`	record of v, f, g, `function_count`, and convergence criteria
`lmbda`	the set of Lagrange multipliers at the active constraints
`Hess`	the Hessian of the objective function at the optimal point

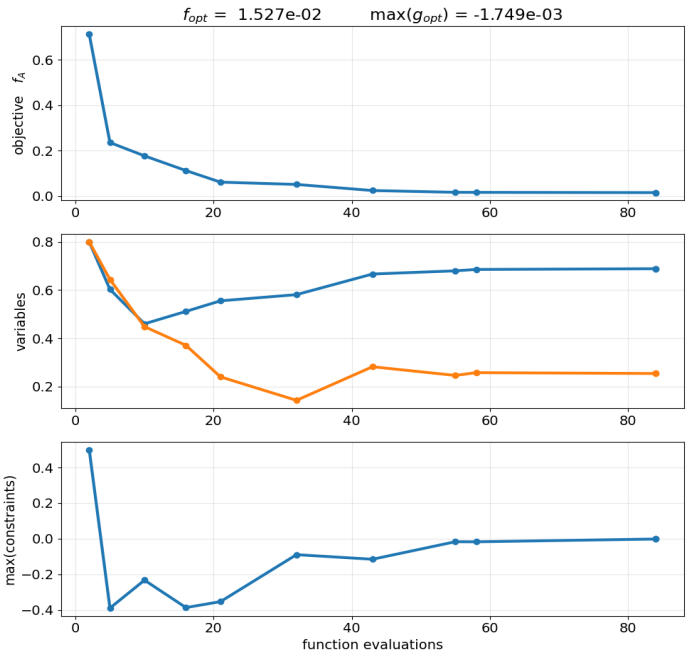
1. To install the multivarious library: `git clone https://github.com/hpgavin/multivarious`
2. SQP computes Lagrange multipliers (`lmbda`) and a Hessian matrix (`H`). (ORS and NMS do not.)

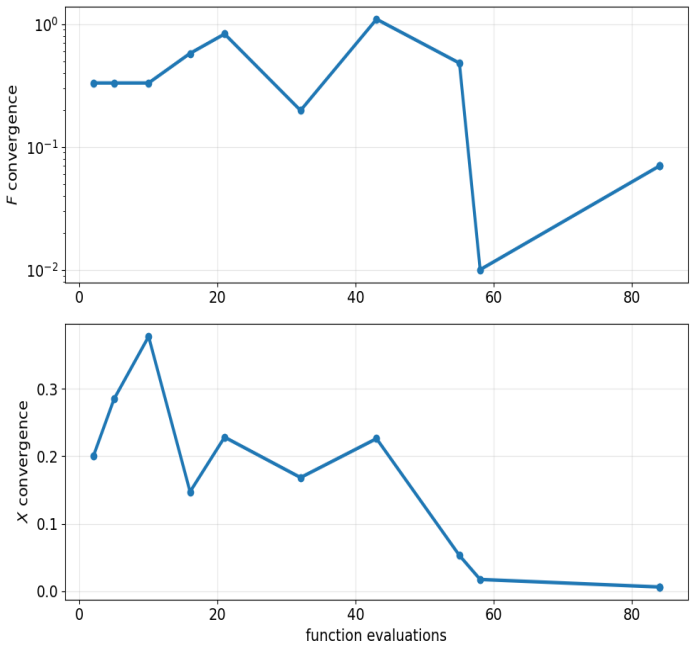
Code to define and solve the example optimization problem (*with comments*)

```
#! /usr/bin/python3 -i

import numpy as np
from types import SimpleNamespace

from multivarious.opt import ors
from multivarious.opt import nms
from multivarious.opt import sqp
from multivarious.utl import plot_cvg_hst

# Define the optimization problem. ========================================
def opt_example_analysis( v, C ):
    """
    Relate the design objective, f, and the design constraints, g, to
    the design variables, v, and constants, C.
    """

    v1 = v[0]                    # description of design variable "v1", units
    v2 = v[1]                    # description of design variable "v2", units

    a = C.a                      # description of constant a, units
    b = C.b                      # description of constant b, units
    c = C.c                      # description of constant c, units

    # the design objective
    f = ( v1 - c[0] )**2 + ( v2 - c[1] )**2 +  c[2]*np.random.randn(1)

    # the array of design constraints
    g = np.array([
        a[0] * ( (v1 - a[1])**2 + (v2 - a[2])**2 )/a[3] - 1,  # "g1"
        1 - b[0]*( (v1 - b[1])**2 + (v2 - b[2])**2 )/b[3]     # "g2"
    ])

    return f[0], g               # end of opt_example_analysis()

# Set-up and Solve the optimization problem. ==============================

# Constants used within the optimization analysis ...
C = SimpleNamespace()
C.a = np.array([  1.0,  0.2,  0.5,  0.3 ])
C.b = np.array([ -2.0, -0.5,  0.5, -1.5 ])
C.c = np.array([  0.8,  0.2,  0.00 ])

v_lb = np.array([ 0.0,  0.0])        # lower bound on the design variables
v_ub = np.array([ 1.0,  1.0])        # upper bound on the design variables

n = len(v_lb)                        # the number of design variables

#v_init = v_lb + np.random.randn(n)*(v_ub - v_lb) # a  random   initial guess
v_init = np.array([ 0.8 , 0.8 ])                  # a specified initial guess

# optimization options ...
#        0      1        2        3         4          5      6      7      8
#      msg    tol_v    tol_f    tol_g    max_evals  pnlty  expn  m_max  cov_F
opts = [ 3 ,   1e-2 ,   1e-2 ,   1e-3 ,    50*n**3 ,  0.5 ,  0.5 ,   1 ,  0.05 ]

# Solve the optimization problem using one of ... ors , nms , sqp
v_opt, f_opt, g_opt, cvg_hst, _,_  = sqp(opt_example_analysis, v_init, v_lb, v_ub, opts, C)

# plot the convergence history
plot_cvg_hst( cvg_hst , v_opt )
```

Optimized Random Search (`ors`) Results

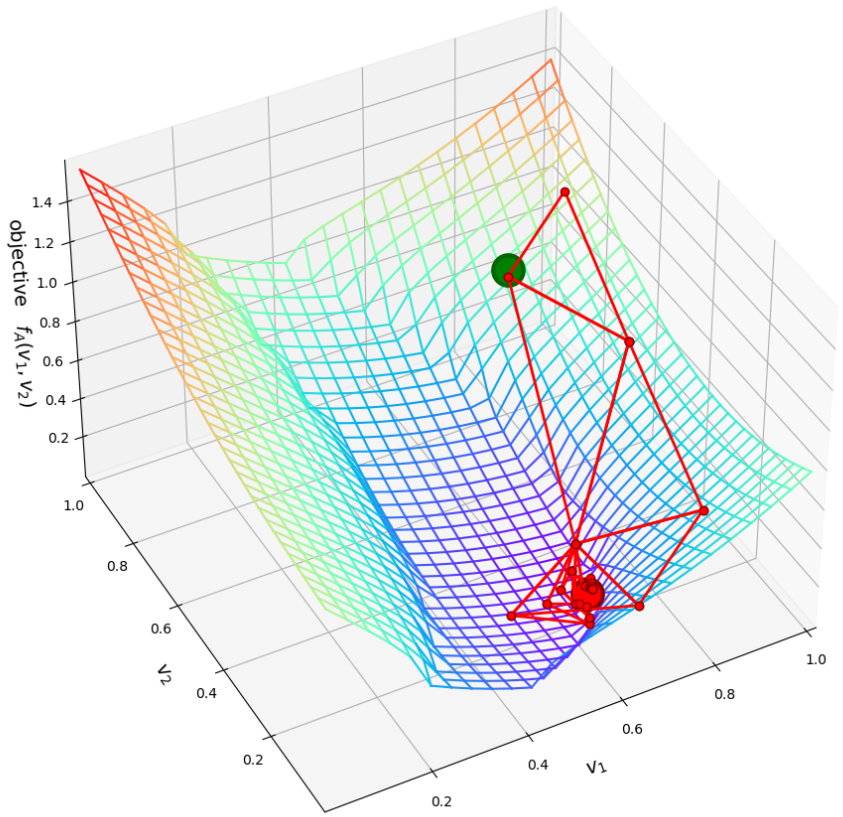


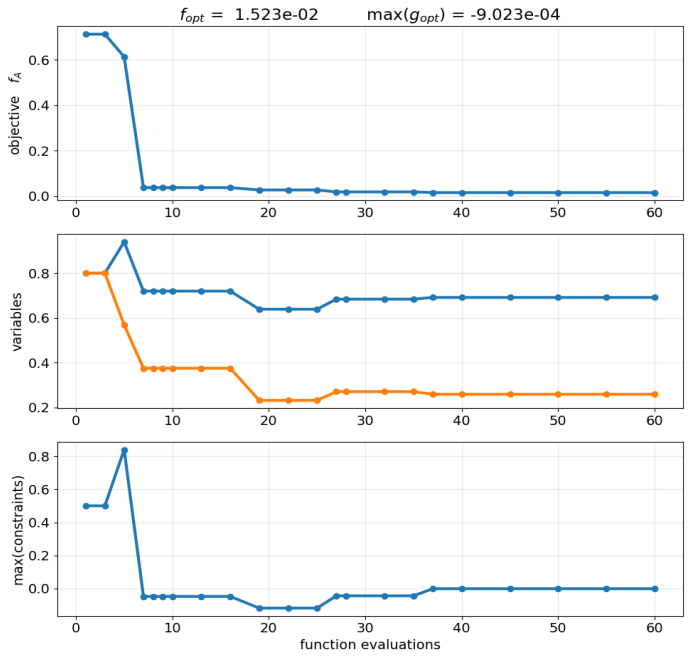


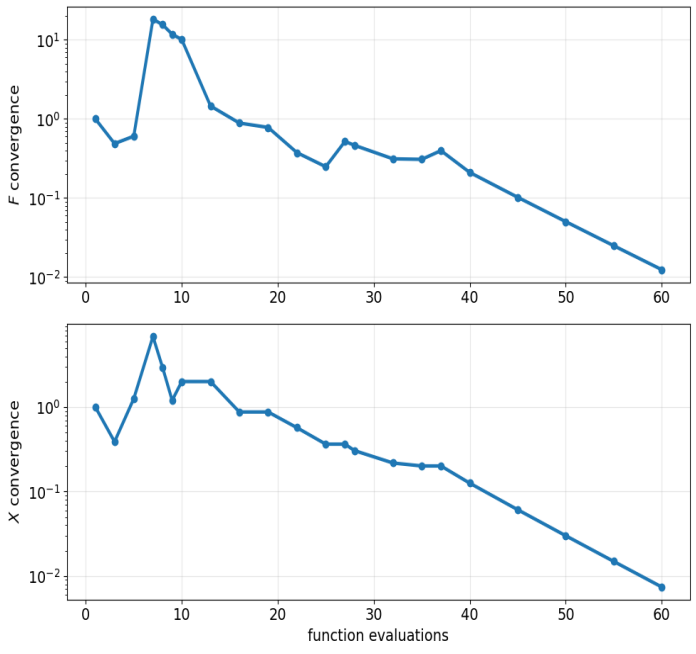


```
-+-+-+-+-+-+-+-+-+-+- ORS -+-+-+-+-+-+-+-+-+-+-+-+-+-+
iteration               =     8     ***  feasible  ***
function evaluations    =    62  of    400  (15.5%)
e.t.a.                  = 12:09:44
objective               =   2.424e-02
variables               =   7.014e-01  3.205e-01
max constraint          =  -5.470e-02 (0)
objective convergence   =  7.5190e-03    tol_f = 0.010000
variable  convergence   =  9.0573e-03    tol_v = 0.010000
c.o.v. of F_A           =   0.000e+00
step std.dev (sigma)    = 0.058
-+-+-+-+-+-+-+-+-+-+- ORS -+-+-+-+-+-+-+-+-+-+-+-+-+-+
line quadratic update successful

* Woo-Hoo! Converged solution found!
*           convergence in design variables
*           convergence in design objective
* Woo-Hoo! Converged solution is feasible!
* Objective   :   2.424e-02
* ------------------------------------------------------------
*             v_init         v_lb     <     v_opt    <     v_ub
* ------------------------------------------------------------
* v[  0]      0.8000         0.0000         0.70136        1.0000
* v[  1]      0.8000         0.0000         0.32048        1.0000
* Constraints :
*     g(  0) =     -0.05470
*     g(  1) =     -0.96733
*
* ------------------------------------------------------------
* Completion  : 12:09:35 (0:00:01)
```

Nelder Mead Simplex (nms) Results

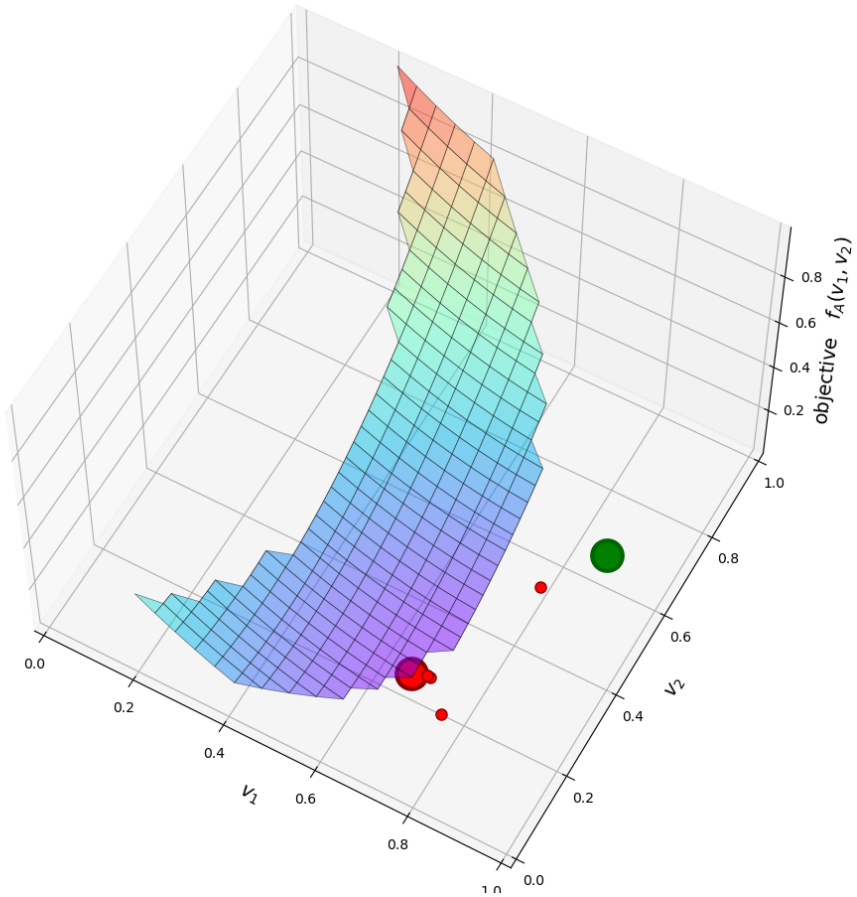


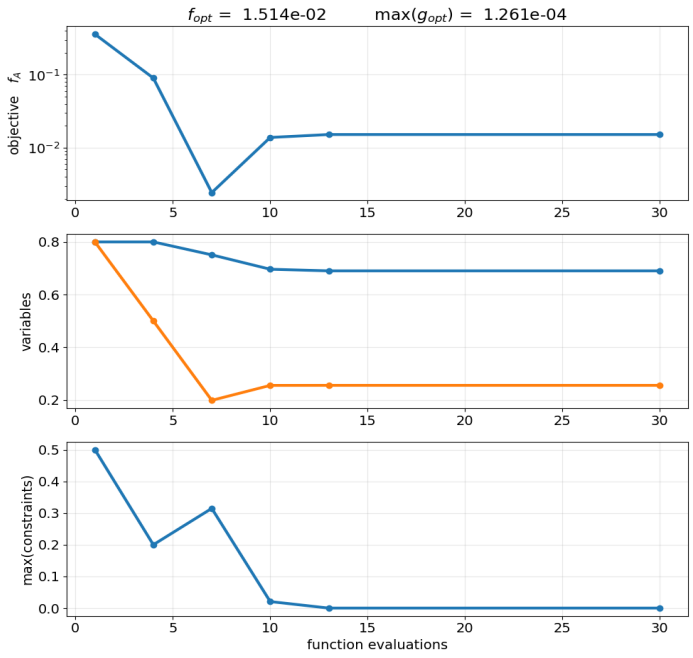


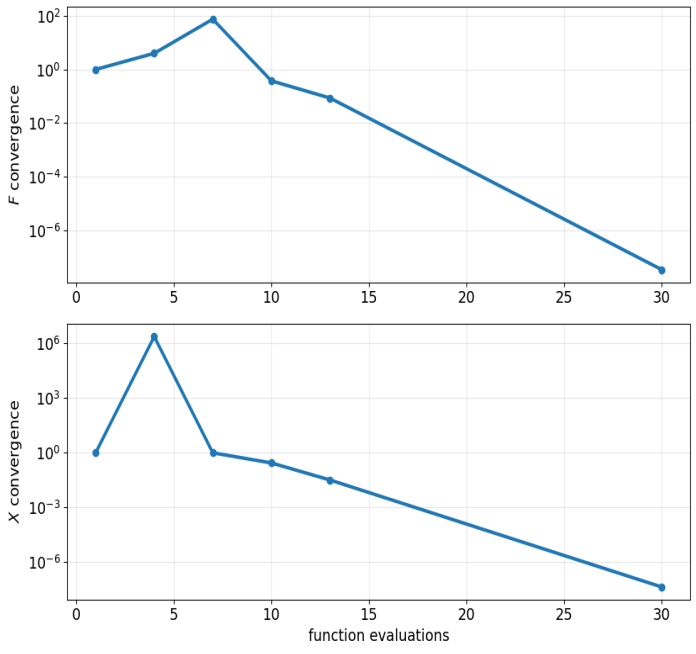


```
====================== NMS ============================
iteration                =    22   *** feasible ***
                                          shrink
function evaluations     =    60 of   400 (15.0%)
e.t.a.                   = 11:59:28
objective                =   1.523e-02
simplex :    vertex 1    vertex 2    vertex 3
           6.915e-01   6.923e-01   6.910e-01
           2.588e-01   2.606e-01   2.595e-01
f_A        1.523e-02   1.528e-02   1.542e-02
max(g)   = -9.023e-04  -1.229e-03  -3.622e-03
cov(F_A) =  0.000e+00   0.000e+00   0.000e+00
objective convergence    =  1.2412e-02   tol_f = 0.010000
variable  convergence    =  7.4241e-03   tol_v = 0.010000
c.o.v. of F_A            =  0.0000e+00
====================== NMS ============================

* Woo Hoo!  Converged solution found!
*           convergence in design variables
* Woo Hoo!  Converged solution is feasible
*          objective =   1.523e-02   evals = 60   time = 1.53s
* ---------------------------------------------------------------------------
*                v_init      v_lb     <    v_opt     <    v_ub
* ---------------------------------------------------------------------------
*  v[  1]       0.8000      0.0001        0.69148        1.0000
*  v[  2]       0.8000      0.0001        0.25881        1.0000
* ---------------------------------------------------------------------------
* Constraints:
*  g[  1] =     -0.00090   ** binding **
*  g[  2] =     -0.97041
*
* ---------------------------------------------------------------------------
* Completion  : 11:59:20 (0:00:01)
```

Sequential Quadratic Programming (sqp) Results







```
********************** SQP ****************************
iteration               =     6   *** feasible ***
function evaluations    =    30 of   400 ( 7.5%)
e.t.a.                  = 12:13:07
objective               =   1.514e-02
variables               =   6.901e-01   2.553e-01
max constraint          =   1.261e-04  (1)
Step Size               =  -6.104e-05
BFGS method             : modify gradients to ensure Hessian > 0, Hessian update
QP method               : ok
objective convergence   =  3.3522e-08   tol_f = 0.010000
variable  convergence   =  4.3403e-08   tol_v = 0.010000
********************** SQP ****************************

* Woo Hoo!  Converged solution found in 6 iterations!
*           convergence in design variables
*           convergence in design objective
* Woo Hoo!  Converged solution is feasible
*          objective =   1.514e-02   evals = 30   time = 0.70s
* ------------------------------------------------------------------------------
*                v_init      v_lb     <    v_opt     <    v_ub      lambda
* ------------------------------------------------------------------------------
*  v[  1]       0.8000      0.0000        0.69008        1.0000
*  v[  2]       0.8000      0.0000        0.25534        1.0000
* ------------------------------------------------------------------------------
* Constraints:
*  g[  1] =      0.00013      lambda[  1] =      0.22479       ** binding **
*  g[  2] =     -0.96821      lambda[  2] =      0.00000
* Active Constraints:  1
* ------------------------------------------------------------------------------
* Completion  : 12:12:59 (0:00:00)
```

Optimization of Design Problems with Uncertainties

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 $M_F(\boldsymbol{v})$ and coefficient of variation $C_F(\boldsymbol{v})$ of the uncertain performance requires a sample of values of the performance metric $f_{\mathsf{A}}(\boldsymbol{v}, \boldsymbol{C})$ for the same set of design variables $\boldsymbol{v}$ and constants $\boldsymbol{C}$, but with different values of the uncertain variables. Given a sample $[f_{\mathsf{A},1}, f_{\mathsf{A},2}, ..., f_{\mathsf{A},m}]$, from m repeated evaluations of the augmented objective function, $f_{\mathsf{A}}(\boldsymbol{v}, \boldsymbol{C})$ the estimates of the mean and coefficient of variation can be computed as follows:

$$M_F = \frac{1}{m} \sum_{i=1}^{m} f_{\mathsf{A},i}(\boldsymbol{v}, \boldsymbol{C})$$

and

$$C_F = \frac{1}{M_F}\left[\frac{1}{m-1}\sum_{i=1}^{m}\left(f_{\mathsf{A,i}}(\boldsymbol{v},\boldsymbol{C}) - M_F\right)^2\right]^{1/2}$$

- 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 $f_{\mathsf{A}}(\boldsymbol{v},\boldsymbol{C})$, 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 $f_{\mathsf{A}}(\boldsymbol{v},\boldsymbol{C})$ is estimated as C_F.

  – Problems with larger inherent variability require larger sample sizes to achieve the same level of confidence.

  – Problems which require higher confidence require larger sample sizes to achieve the same level of inherent variability.

  The equation for m is:

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

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

  The option `m_max` (`opts[7]`) 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 option `cov_F` (`opts[8]`) sets the value of e_F, the desired estimation error for the mean, M_F (as a c.o.v.)

  Note that optimizing with small values of `cov_F` (`opts[8]`) and large values of `m_max` (`opts[7]`) could require many (many) function evaluations, but will represent the statistics of the objective function very well.

- Because the estimate of the mean and c.o.v. of F_{A} requires a sample of m evaluations, the total number of evaluations for the optimization needed for a desired level of confidence in the estimate of M_F may require an undesireably large amount of computational time. To allow for the increased computational burden, the maximum total number of function evaluations, `max_evals` (`opts[4]`) may need to be set larger.

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

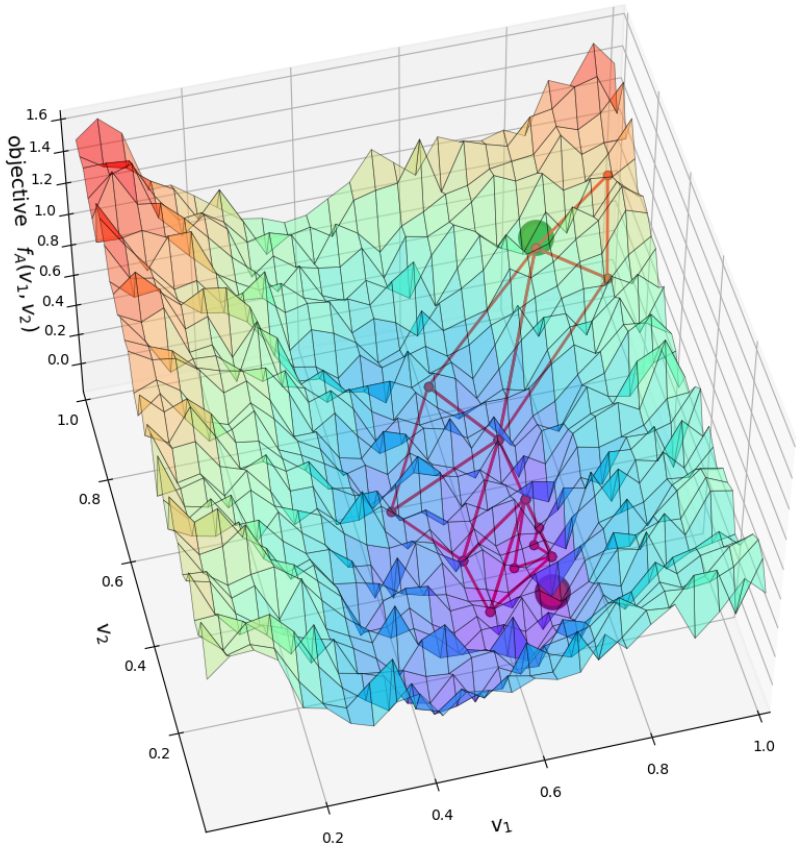
```
#       CHOOSE ONE OF THE FOLLOWING RISK-BASED PERFORMANCE MEASURES ...
#       F_risk = M_F                            # average-of-N values
#       F_risk = M_F * ( 1 + C_F/np.sqrt(m) ) # 84th percentile of the avg. of F
        F_risk = M_F * ( 1 + C_F )              # 84th percentile of F
#       F_risk = max_F;                         # largest-of-N values
```

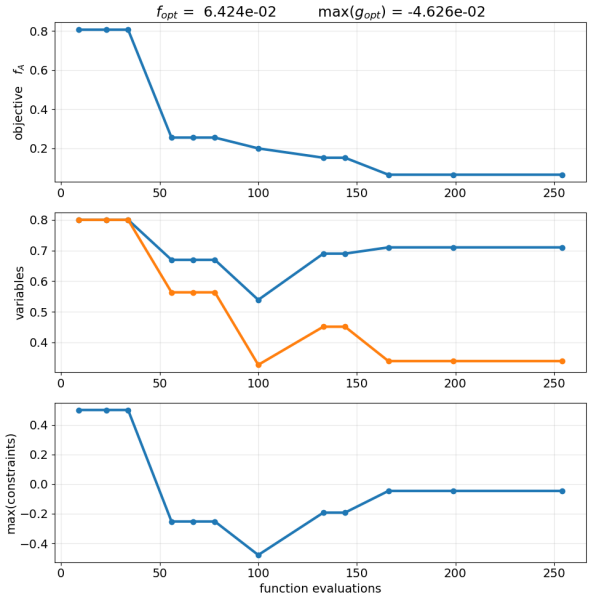
In the example script on page 4, the uncertainty-level in this example problem is set by the coefficient c_3, so in this example we know in advance that the standard deviation of F_{A} is equal to the value we use for c_3 (`0.10`). Setting `cov_F` (`opts[8]`) to `0.05` means that we desire an estimate for the mean of F_{A} that is accurate to within $\pm 5\%$, with a 90% confidence level. Using this information along with the equation for m, above, we will need a sample size of $m = (1.645 \times 0.10/0.05)^2 \approx 11$. So the maximum sample size, `m_max` (`opts[7]`), can be set to `11`.

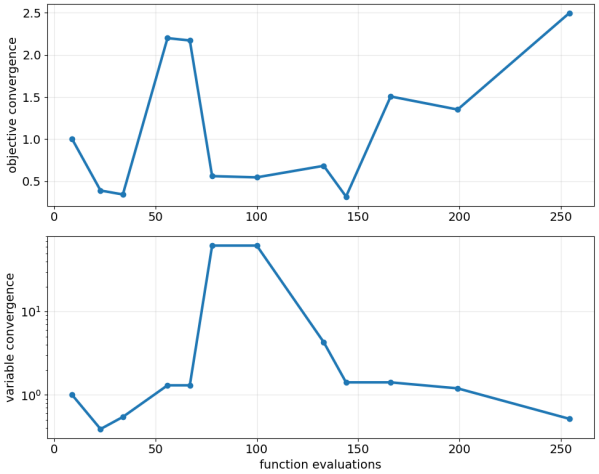
Overall, the goal in setting values of `m_max` (`opts[7]`) and `cov_F` (`opts[8]`) is to balance values that get the overall optimization to consistently converge to sufficiently reliable solutions with a sufficiently small number of function evaluations.

Note that:

- The risk measure used in this example is $F_{\mathsf{A,risk}} = M_F(1 + C_F)$.
- 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.


objective $f_A(v_1, v_2)$
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.8
0.6
v_2
0.4
0.2
0.2
0.4
0.6
0.8
1.0
v_1


f_{opt} = 6.424e-02 max(g_{opt}) = -4.626e-02
objective f_A
variables
max(constraints)
function evaluations


objective convergence
variable convergence
function evaluations