This course considers engineering design as an application of engineering analysis in which the design specification (i.e., design plan) is represented by a set of design variables \( \mathbf{x} = [x_1 \ x_2 \ x_3 \ \cdots \ x_n]^T \) and through which performance, utility, safety, equity, and sustainability can be quantified and optimized.

A design that meets performance, utility, safety, equity, and sustainability requirements can be called functional. In the approach adopted for this course one of these design requirements is selected to be the primary design objective. This is quantified by the objective function, which is a function of the design variables.

\[
f(x_1, x_2, \cdots, x_n) \quad \text{or, more compactly,} \quad f(\mathbf{x}) .
\]

A common objective is to simply minimize the total cost of the design, possibly including externalities. In other design problems the objective could be to maximize an aspect of performance.

The functionality of a design depends on more than the primary objective (performance, profits, etc.). Most designs must also meet a range of other criteria (e.g., reliable enough, safe enough, stable enough, strong enough, equitable enough, sustainable enough). These criteria are expressed as inequalities, and also depend upon the values of the design variables. In general, a set of \( m \) inequality constraints can be written as

\[
g_1(x_1, x_2, x_3, \cdots, x_n) \leq 0 \\
g_2(x_1, x_2, x_3, \cdots, x_n) \leq 0 \\
\vdots \\
g_m(x_1, x_2, x_3, \cdots, x_n) \leq 0
\]

or as \( \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \), by collecting the \( m \) individual inequalities into a single vector inequality. Design constraints confine the solution to ranges (or sub-spaces) of admissible possibilities.

In this framework, the primary design objective is to be optimized (minimized or maximized) subject to satisfying the design constraints:

\[
\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{such that} \quad \mathbf{g}(\mathbf{x}) \leq \mathbf{0} .
\]

In certain rare cases, we can write actual equations for \( f(\mathbf{x}) \) and \( \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \) and use calculus to find an equation for the constrained optimum. In the vast majority of practical problems, however, the equations are simply much too complicated to be solved with pencil and paper. In such cases computer-aided analysis can automate the evaluation of the objective and admissibility of a particular trial design. Further, computer-aided optimization allows designers to rapidly iterate on candidate designs in order to converge on an admissible and possibly “optimal” design. Finally, be aware that in challenging design problems it is simply too hard to converge to a design that is both optimal and admissible or feasible.

Sometimes feasibility suffices.
1. Download the CEE 201 optimization codes (0 points)
   - ORSopt.m
   - NMAopt.m
   - SQPopt.m
   - avg_cov_func.m
   - box_constraint.m
   - optim_options.m
   - plot_surface.m
   - plot_cvg_hst.m

and move them into your path/to/cee201/m-files/ folder.

2. The Nelder-Mead method (20 points):

   Consider iterations of the Nelder-Mead algorithm for minimizing a function of several variables.

   (a) Using the figure below, trace out five iterations of the Nelder-Mead algorithm. Each iteration starts with a reflection. Use a straight-edge to draw your triangle-shaped simplexes. Within each triangle write “R” for reflection; “RE” for reflection-extension; “CI” for inside contraction; “CO” for outside contraction; and “S” for shrink.

   (b) Within an iteration, is the best point of a “simplex” ever updated? (yes/no)

   (c) If a “reflection” point is selected, does the “volume” of the “simplex” remain unchanged? (In 2D, the “simplex” is the triangle and the “volume” of the “simplex” is the area of the triangle.) (yes/no)

   (d) Suppose that after several consecutive contraction steps of the Nelder-Mead method, the three vertices of a 2D simplex become nearly co-linear (or the four vertices of a 3D simplex become nearly co-planar). What implication would this have for subsequent steps?

   (e) When enforcing constraint equations via a penalty function, if the optimization algorithm converges to an infeasible point, should you re-try the optimization with a larger or a smaller penalty factor?

   (f) If the optimization routine reaches its maximum iteration limit before converging, what changes to the optimization algorithm options:

   ```
   % display tolX tolF tolG MaxEvals Penalty Exponent
   options = [ 2 0.01 0.01 0.01 500 1.0 1.0 ];
   ```

   would you try, and why? The meaning of the terms in this vector are explained in the document “Examples of Running Constrained Optimization Codes.”
3. Convergence criteria (10 points)

In an iteration of the Nelder-Mead method for $n = 5$ has a simplex $x$ represented by

\[
\begin{array}{cccccc}
84.80 & 84.93 & 84.88 & 84.92 & 85.01 & 85.10 \\
14.89 & 15.05 & 14.98 & 15.12 & 14.95 & 15.07 \\
9.97 & 10.00 & 9.99 & 10.09 & 9.84 & 9.94 \\
55.91 & 55.91 & 56.28 & 56.16 & 56.02 & 56.04 \\
72.98 & 72.79 & 72.89 & 72.95 & 72.86 & 72.96 \\
\end{array}
\]

with corresponding objective function values $f$

\[
\begin{array}{cccccc}
99.81 & 99.93 & 100.04 & 100.06 & 100.10 & 100.18 \\
\end{array}
\]

Consider convergence criteria values of $\epsilon_x = 0.01$ and $\epsilon_f = 0.01$.

(a) Does this simplex represent a solution that is converged in the design variables, $x$?
(b) Does this simplex represent a solution that is converged in the design objective, $f$?
4. Efficiency of numerical optimization methods as compared to gridded search

**Gridded Search**

A continuous parameter space \((x_1, x_2)\) can be sampled into a finite number of points (shown as blue circles in the figure below) on a grid from \(x_{1,lb}\) to \(x_{1,ub}\) and from \(x_{2,lb}\) to \(x_{2,ub}\). The cost \(f(x_1, x_2)\) and constraints \(g(x_1, x_2)\) can be evaluated at every point on the grid. By keeping track of the best feasible point as each point is evaluated, the optimal solution from within the set of grid points can be identified (the green star). In most cases, the true optimal value (black star) is off of the grid. A finer grid would contain a point closer to the true optimal, but would also involve more evaluations of \(f\) and \(g\). This idea raises the idea of a method to adaptively refine the grid around (one or more) approximate optimal solutions.

The figure below is representative of problems that have discontinuous feasible regions. A local minimizing routine starting within region A might not move into region B as this would require moving through the infeasible space (hashed). Numerical optimization methods could move from one feasible region into another, if the step size of the update of the design variables is larger than the distance between adjacent feasible regions.

In 3D, the gridded space would be a cube divided into little blocks. The feasible regions would be 3D objects within the 3D grid. The grid shown below is a regular grid in which all values of \(x_i\) are uniformly spaced with the same increment.
(a) Constrained Optimization using a Gridded Parameter Search (35 points):
The objective function in this optimization problem is called the Rastrigin function. With the constraints below, it features the possibility of disconnected feasible regions.

\[
\begin{align*}
\text{minimize} & \quad f(x_1, x_2, x_3) = 3a + \sum_{i=1}^{3} \left( x_i^2 - a \cos(\sqrt{2} \pi x_i) \right) \\
\text{such that} & \quad g_1(x_1, x_2, x_3) = 100 - (c_1 x_1 - b_1)^2 - (c_2 x_2 - b_2)^2 - (c_3 x_3 - b_3)^2 \leq 0 \\
& \quad g_2(x_1, x_2, x_3) = 200 - (b_1 x_1 - c_1)^2 - (b_2 x_2 - c_2)^2 - (b_3 x_3 - c_3)^2 \leq 0
\end{align*}
\]

with constants \( a = 10 \), \( b = [1, 2, 3] \) and \( c = [3, 2, 1] \). To further constrain the problem consider design variables within the bounds: \(-5 \leq x_i \leq 5 \) for \( i \in \{1, 2, 3\} \)

Write a \texttt{matlab} function and a \texttt{matlab} script.

i. The \texttt{matlab function} uses a trial set of values for \( x_1, x_2, \) and \( x_3 \) (along with values for the constants, \( a, b, \) and \( c \) — see lines 1-4 on page 7) to calculate the objective \( f(x_1, x_2, x_3) \), and the constraints, \( g_1(x_1, x_2, x_3) \) and \( g_2(x_1, x_2, x_3) \). This file name must be \texttt{udo_HW2_2024_analysis.m} ,

and the first line of this function file must be:

\[
\text{function } [f,g] = \text{udo_HW2_2024_analysis}( x , \text{constant } );
\]

This function uses values in a three-element vector \( x \) to compute a value of the objective function (the scalar \( f \)), and values of the constraints (the \texttt{column} vector \( g \)). In this problem, the \texttt{matlab} (column) vector \( g \) has two elements: \([g(1) ; g(2)] \). This \texttt{.m}-function can be as short as six lines of code.

Refer to Examples of Running Constrained Optimization Codes.

To check if your function \texttt{udo_HW2_2024_analysis} is working correctly, here is an example of a correct calculation. After assigning values to the constants and to the set \texttt{constant}, (by typing lines 1-4 on page 7 into the \texttt{Command} window) type the following into the \texttt{Command} window ...

\[
>> [f,g] = \text{udo_HW2_2024_analysis}( [2 ; 3 ; 4 ], \text{constant } )
\]

... which should give you ...

\[
f = 55.619 \quad \text{and} \quad g = [58 ; 62 ].
\]
ii. The **matlab script**, called `udo_HW2_2024_opt.m`, is provided for you on the next page. (You’re welcome.) It is a `.m`-file containing **matlab** commands that use the analysis function to find the optimal value of the design variables. It implements a method that is terribly inefficient, but is easy-to-understand, easy-to-program, and is guaranteed to find a value that is close to the optimal value. Lines 1-3 of the `.m`-script assign values to the constants. Line 4 puts the constants `{a, b, c}` into a set called `constant`. Lines 6-7 assign lower and upper bounds for $x_1$, $x_2$, and $x_3$, and lines 9-12 assign vectors of several values of $x_1$, $x_2$, and $x_3$ that are separated by an **increment** and are between their lower and upper bounds. Lines 14-16 then determine the number of values in each vector with the **length** command. The total number of analyses is the product of the length of each parameter-set vector (line 18).

Given vectors of values for $x_1$, $x_2$, and $x_3$, the script uses a set of “nested for-loops” to evaluate your function

$$[f, g] = \text{udo_HW2_2024_analysis}(x, \text{constant});$$

for each and every combination of design variables (lines 25-52). For each evaluation, the script computes the objective $f$ and constraints $g$ by evaluating the **analysis** function (line 31) and checks to see if all elements of $g$ are less than zero and if $f$ is less than the smallest value of $f$ found so far (line 33). If all elements of $g$ are less than zero (feasible) and $f$ is less than the best value computed so far, the script updates $f_{\text{opt gs}} = f$; and $x_{\text{opt gs}} = x$; (lines 35-37). If not, the script simply goes on to the next combination of design variables. After all combinations have been tried, the optimal values for the design variables will be $x_{\text{opt gs}}$ and the associated value of the objective function will be $f_{\text{opt gs}}$ (lines 56-58).

Since this method evaluates lots and lots of alternative designs, you may expect the script to take some time to run. The script keeps track of how much time it takes using the commands **tic** and **toc** (lines 24 and 55). When the script completes, the **Command** window will display the **compute_time**, which is the number of seconds elapsed between **tic** and **toc**.
a = 10;
b = [1; 2; 3];
c = [3; 2; 1];
constant = {a, b, c}; % the set of constants
x_lb = [-5.0 -5.0 -5.0]; % lower bounds for each design variable
x_ub = [5.0 5.0 5.0]; % upper bounds for each design variable
increment = 0.1; % smaller increment, more values to try
x1_set = [x_lb(1) : increment : x_ub(1)]; % the set of values for x(1)
x2_set = [x_lb(2) : increment : x_ub(2)]; % the set of values for x(2)
x3_set = [x_lb(3) : increment : x_ub(3)]; % the set of values for x(3)
N_x1 = length(x1_set); % the number of x(1) values in x1_set
N_x2 = length(x2_set); % the number of x(2) values in x2_set
N_x3 = length(x3_set); % the number of x(3) values in x3_set
NumberOfAnalyses = N_x1 * N_x2 * N_x3
f_opt_gs = 1e16; % initialize a value for objective f ... some big number
more off; % show results as they are computed
count = 0;
tic
for i1=1:N_x1 % loop over values of x(1)
for i2=1:N_x2 % loop over values of x(2)
for i3=1:N_x3 % loop over values of x(3)

x = [x1_set(i1) x2_set(i2) x3_set(i3)]; % trial values for x1, x2, x3
[f,g] = udo_HW2_2024_analysis(x, constant); % run the analysis
if all(g<=0) && f < f_opt_gs % best solution, so far
f_opt_gs = f % update objective
x_opt_gs = x % update variables
g_opt_gs = g % update constraints
end

count = count + 1;
if rem(count,10000) % when will this ever finish?
secs = toc;
secs_left = round((NumberOfAnalyses-count)*secs/count);
etta = datestr(now+secs_left/3600/24);
fprintf(' %8d (%4.1f %); %5.2f secs / count; eta: %s (%4d s)\n', ...
count,100*count/NumberOfAnalyses,secs/count,eta,secs_left);
end
end % x(3) loop
end % x(2) loop
end % x(1) loop
NumberOfAnalyses % display the total number of analyses
compute_time = toc % display the total time to complete the analysis
x_opt_gs % display the best design variables found
f_opt_gs % display the associate objective function value
g_opt_gs % display the associate constraint values

“Consider everything. Keep the good. Avoid evil whenever you notice it.”
(b) Constrained Optimization using ORSopt, NMAopt, and SQPopt (20 points)

Solve this problem using the `matlab` functions provided on the course web-page. The methods implemented in these functions keep track of the design variables, the objective function, and the constraints. Each method implements a different approach in determining what the next guess of the optimal value of the design variables should be, based on the information it has developed so far by trying out several different values of the design variables. As information about the problem accumulates with each successive iteration, the methods converge toward an optimal solution. To use these optimization functions, use the same function `udo_HW2_2024_analysis.m` that you wrote for part (a). This will be passed to the optimization routines in single quotes, to indicate that the variable is a function name. These methods, like most optimization methods, require a good initial guess at the design variables. The initial guess of the optimal design variables is specified in the `matlab` vector `x_init`. The lower and upper bounds of the design variables are given in `matlab` vectors `x_lb` and `x_ub`.

To apply the constrained optimization methods in your `.m`-script file, add the following lines at the end of `udo_HW2_2024_opt.m`:

```matlab
x_init = [0 0 1]; % some initial guess for x(1), x(2), and x(3)
options = [1 increment/(x_ub(1)-x_lb(1)) 1e-2 1e-4 1000 100];
tic
[x_opt, f_opt, g_opt, cvg_hst] = ... NMAopt ('udo_HW2_2024_analysis', x_init, x_lb, x_ub, options, constant);
compute_time = toc % display the total time to complete the analysis
plot_cvg_hst (cvg_hst, x_opt, 11);
```

Note here that the value of `tolX` is set to the value of `increment / (x_ub(1)-x_lb(1))` used in the gridded search. This value is the precision of the gridded search relative to the span of the search domain. Setting `tolX` in this way makes for a level comparison between the efficiency of the gridded search and more advanced optimization methods. Normally, smaller values of `tolX` would be used. Try each of the three optimization functions. Try various values in the initial design variable vector `x_init` to see if the computed solution depends on the initial design variables. Keep track of the total number of function evaluations made in each of the three methods. (This information is displayed in the `Command` window.) Which (if any) of the constraint inequalities are binding the optimum point?
To hand in:

(a) your well-commented .m-function udo_HW2_2024_analysis.m

Every MATLAB file that you write should be well-commented. In the first lines of the .m-file, write comments that tell a user about your .m-file ... what it does, how to use it, your name, your e-mail, and the date.

(b) The tables below, filled in. Run two analyses, first with increment = 1.0 and again with increment = 0.1. In each row, circle the value of the constraint(s) (g1, g2) that bind(s) the optimum point.

Gridded Search Increment Value = 1.0

<table>
<thead>
<tr>
<th>f*</th>
<th>g1*</th>
<th>g2*</th>
<th>x1*</th>
<th>x2*</th>
<th>x3*</th>
<th>compute</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gridded Search Increment Value = 0.1

<table>
<thead>
<tr>
<th>f*</th>
<th>g1*</th>
<th>g2*</th>
<th>x1*</th>
<th>x2*</th>
<th>x3*</th>
<th>compute</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To complete the table below, run three analyses using each of the three methods ORSopt, NMAopt and SQPopt, using different initial values of the design variables each time. In each row, circle the value of the constraint(s) \((g_1, g_2)\) that bind(s) the optimum point.

<table>
<thead>
<tr>
<th>method</th>
<th>initial values</th>
<th>optimal values</th>
<th>number of analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>ORSopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORSopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORSopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMAopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMAopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMAopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMAopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQPopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQPopt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQPopt</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>