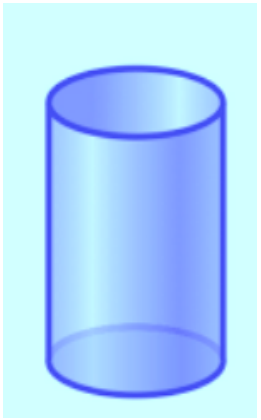
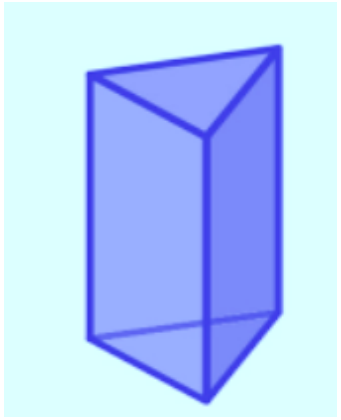
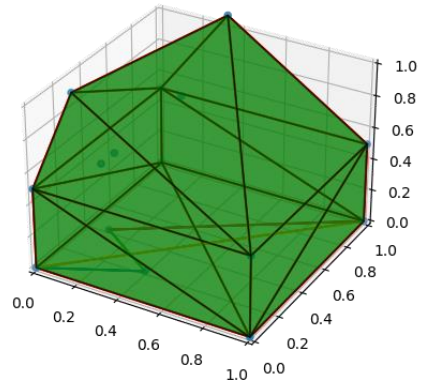
MODA Assignment 2 - 2022:
Design Optimization





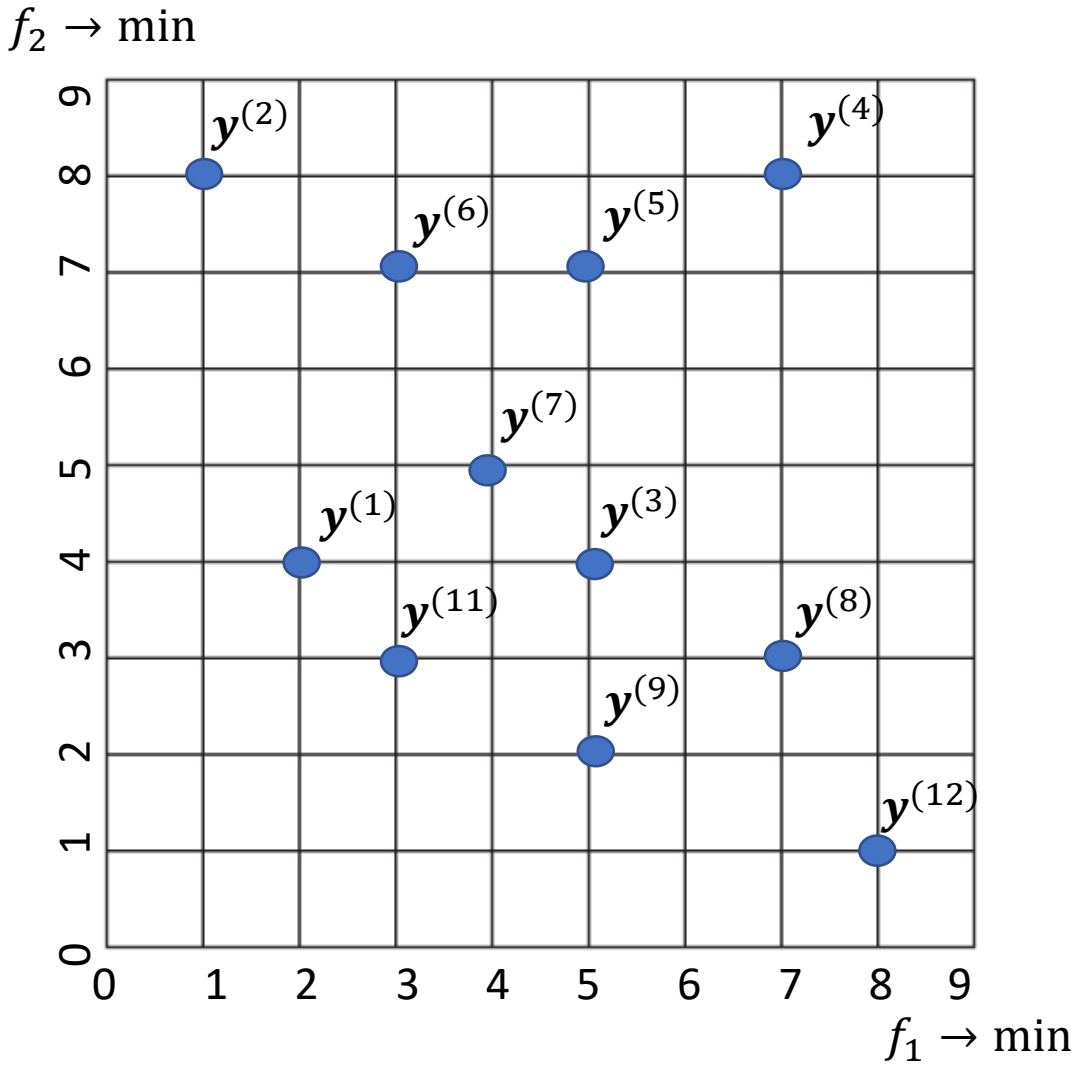


Emmerich (m.t.m.Emmerich (a sign) liacs.leidenuniv.nl),
Kamand Hajiaghapour (TA, k.hajiaghapour (a sign) umail.leidenuniv.nl)
LIACS 2022

For technical questions related to python please ask TA Kamand Hajiaghapour (please use emails above or, preferably, the brightspace forum, for questions).

Task 1: Non-dominated sorting and hypervolume (20%)

- Describe how the following population will be sorted when using the non-dominated sorting algorithm as in NSGA II. What will be the layers of equal rank. What are the crowding distances in the first ranked layer?
- What is the hypervolume indicator of this population when the reference point is (9,9)
- What are the hypervolume contributions of the points in the best ranked layer (only consider points in that layer).



Task 2: Single Point Method – Tchebycheff DRP (or Achievement scalarization), python(40%)

Use 2-D local search method – see next two slides - for distance to a reference point method $r = (0, M), M = \frac{1}{2}5^2 10$=250 with at least 5 different Tchebycheff weight combinations* (say $w_1 = 0.0, w_1 = 0.1, w_1 = 0.5, w_1 = 0.9\ w_1 = 1.0$) to compute an approximation of the Pareto front and efficient set of the prism (see MODA Assignment 1).

Volume: $f_1(x_1, x_2) = \frac{1}{2}x_1^2 x_2$, Area: $f_2(x_1, x_2) = x_1^2 + 3x_2x_1$

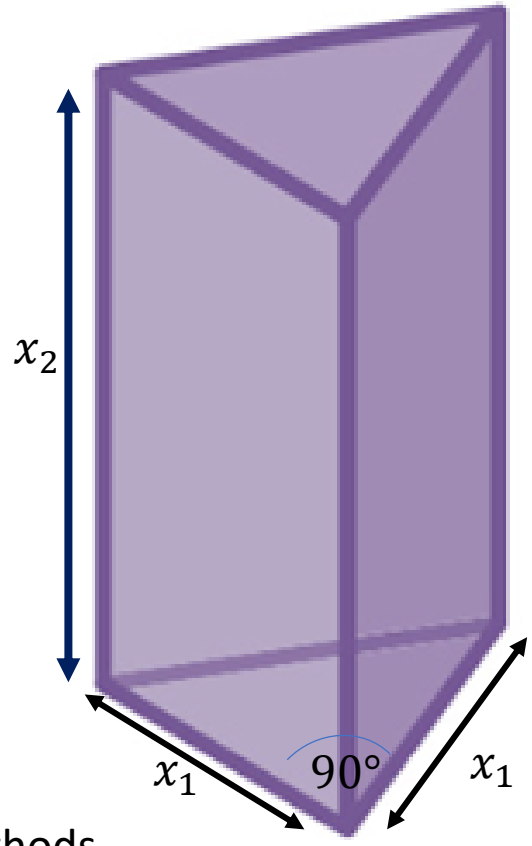
Adjust the stepsize to an appropriate value and choose intervals for x_1 and x_2 as [0,5], [0,10].

Output the approximation set of the Pareto front and efficient set numerically or graphically (both possible).

Add plots of the history of the method on the contour of the Tchebycheff DRP utility (5 plots). (as in the example in the appendix)

Hint: Consider two slides in appendix source code; find all source code files in localsearch.py in the Assignment on brightspace.

(stand-alone python; use in python terminal: pip install numpy; pip install matplotlib).



*max($w_1\ (|f_1(x_1, x_2) - r_1|) + w_2\ (|f_2(x_1, x_2) - r_2|) \rightarrow$ min

$w_1 \in [0,1], w_2 = 1 - w_1$, see also lecture on Single Point Methods

Task 3: Desdeo Framework for Design (40%)

Next, download the desdeo framework and run it. Find all source code files in desdeo-example.py in the Assignment on brightspace.

Change the problem to the **elliptic cylinder optimization** problem. Here the ground area and the to area are ellipses with radii r_1 and r_2.

Use the same boundaries for the decision variables than in the cylinder example in the shared code. Report the changes to your source code.

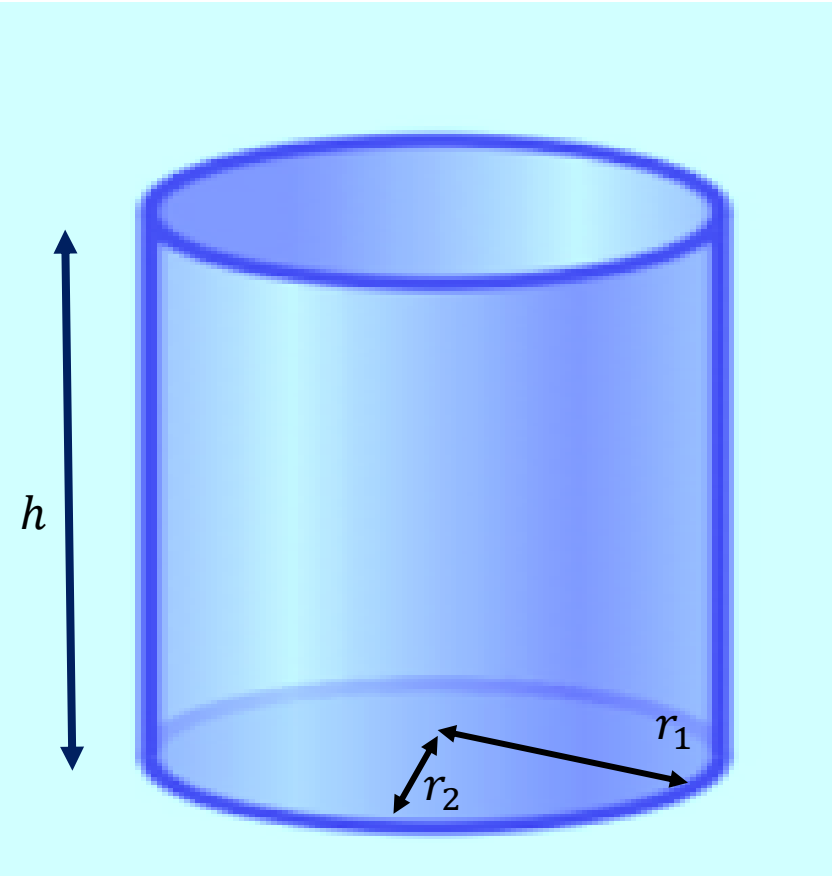
Change the objective function in the source code of Task 1 (desdeo example). Report the changes to your source code by adding the source code to your report. The volume is to be maximized and the surface to be minimized.

$$f_1(r_1, r_2, h) = \pi h\, r_1 r_2 \rightarrow \max$$

$$f_2(r_1, r_2, h) = \pi\left(3(r_1 + r_2) - \sqrt{(3r_1 + r_2)(r_1 + 3r_2)}\right)h + 2\pi r_1 r_2 \rightarrow \min^*$$

Run the optimization with NSGA-III and output the Pareto front. Report the Pareto front and visualize the efficient set in a Parallel coordinate diagram.

Discuss briefly the result. Do you notice a significant pattern that is preserved among Pareto optimal points?

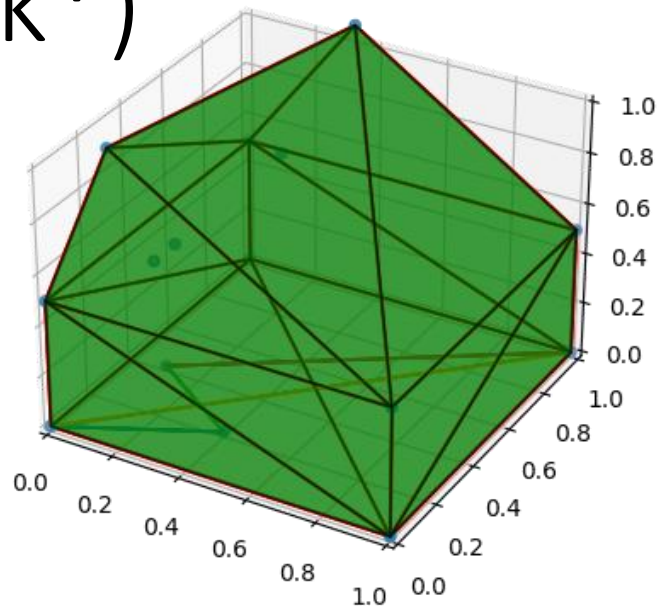


*We use here the formula by Ramnujan which approximates the perimeter of the ellipse.
https://www.mathsisfun.com/geometry/ellipse-perimeter.html

Task 4: Tent example (Bonus Task*)

1. Implement the tent example with four points and visualize some solutions on the Pareto front using the plot visualization tool. (surface(P) →min vs. volume(P) → max). Please fix the base structure. and only update the free points.
2. Visualize one tent in the knee point region and two in the optimal regions of f_1 and f_2 of the Pareto front (for the visualization and details on its visualization see the lecture on design optimization).
3. Experiment with different Pareto fronts for different number of design points and objectives (height, floor area, surface, volume) and report your results.

Hint: To make it easier to start, we will put a working example in the course materials that you can modify. Find all source code files in the Assignment menu on brightspace.




```
random_points = np.array([
    [0,0,0], [0,1,0],[1,1,0], [1,0,0], # floor
    [0, 0, 0.5], [0, 1, 0.5], [1, 1, 0.5],[1, 0, 0.5],  # floor+0.5
    [0, 0.3, 0.4+0.5], [0.4, 0.2, 0.6+0.5], [0.1, 0.4, 0.8+0.5],
    [0.4, 0.5, 0.4+0.5], [.5, 0.7, 0.8+0.5],
    [.2, 0.3, 0.1+0.5], [.2, 0.2, 0.1+0.5]
])
box3 = Tent(random_points)
box3.plot()
```

*Meaning of 'Bonus': not mandatory. When you complete this question, your grade for MODA Assignment 2 will be increased by maximally 2pt (20% of total grade). If your final grade would be or instance 10 it would however not increase. You can therefore compensate for points you might have lost in other parts of assignment 2.

Some example of how to visualize a tent using the program provided. See lecture May 18th for more details on the parameterization of tents.

APPENDIX

Task 1: Plotting the history

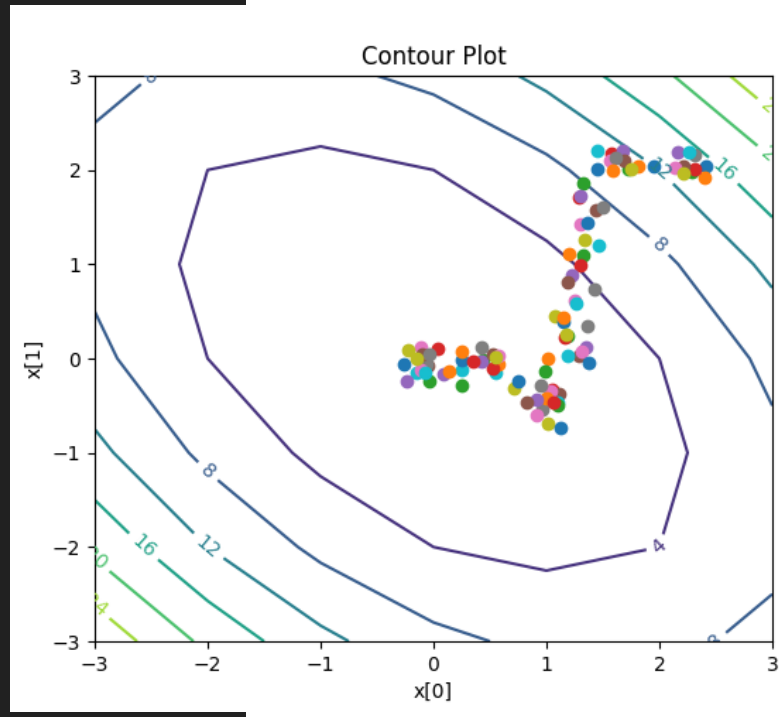
```
bounds=asarray([[-3.0,3.0],[-3.0,3.0]])
step_size=[0.4,0.4]
n_iterations=100
init=[2.4,2.0]
best, score, points, scores,  = local_hillclimber(objective,
                                                  bounds, n_iterations,
                                                  step_size, init)

n, m = 7, 7
start = -3

x_vals = np.arange(start, start+n, 1)
y_vals = np.arange(start, start+m, 1)
X, Y = np.meshgrid(x_vals, y_vals)

print(X)
print(Y)
fig = plt.figure(figsize=(6,5))
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
ax = fig.add_axes([left, bottom, width, height])


Z = (X**2 + Y**2 + X*Y)
cp = ax.contour(X, Y, Z)
ax.clabel(cp, inline=True,
          fontsize=10)
ax.set_title('Contour Plot')
ax.set_xlabel('x[0]')
ax.set_ylabel('x[1]')
for i in range(n_iterations):
    plt.plot(points[i][0],points[i][1],"o")
plt.show()
```



Plotting the history (Task 2) - materials

Task 1 supplementary: Simple 2-D stochastic hillclimber

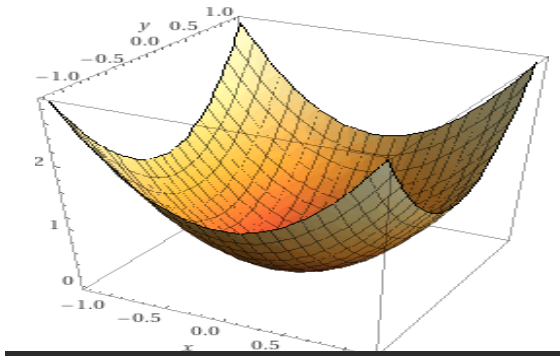
(see also Hillclimbing Unit 11)

```
#  black-box optimization software
def local_hillclimber(objective, bounds, n_iterations, step_size,init):
    # generate an initial point
    best = init
    # evaluate the initial point
    best_eval = objective(best)
    curr, curr_eval = best, best_eval   # current working solution
    scores = list()
    for i in range(n_iterations):
        # take a step
        candidate = [curr[0] +rand()*step_size[0]-step_size[0]/2.0,
                     curr[1]+rand()*step_size[1]-step_size[1]/2.0]
        print('>%d f(%s) = %.5f, %s' % (i, best, best_eval,candidate))
        #+ randn(len(bounds)) * step_size
        # evaluate candidate point
        candidate_eval = objective(candidate)
        # check for new best solution
        if candidate_eval < best_eval:
            # store new best point
            best, best_eval = candidate, candidate_eval
            # keep track of scores
            scores.append(best_eval)
            # report progress
            print('>%d f(%s) = %.5f' % (i, best, best_eval))
            # current best
            curr=candidate
    return [best, best_eval, scores]
```

Candidate solution
Symmetric distribution

Evaluate candidate

Update current best?




```
# objective function
def objective(x):
    return x[0]**2+x[1]**2
```

Define single objective function. Some example.