Gradient descent based optimization

Describtion and setting guidelines

The goal of this package is to optimize the control parameters (e.g. R_E, P_{amb}, P_A, freq...) in order to minimize to_optimize. The argument to_optimize can be any key in the post processing data dictionary, by default it is 'energy_demand'.

- Setting ranges

The argument ranges is a regular dictionary, with 1 key corresponding to each control parameters. The value is a list with 1 or 2 elements: 1, if the parameter is not to be optimized and is kept a constant; 2 if the parameter should be optimized between a lower and upper limit.

Example:

```
ranges = dict(
    R_E = [10e-6, 100e-6],
    ratio = [1.0, 5.0],
    P_amb = [1e5],
    ...
)
```

R_E is optimized between 10 µm and 100 µm

ratio (R_0/R_E) is optimized between 1 and 5

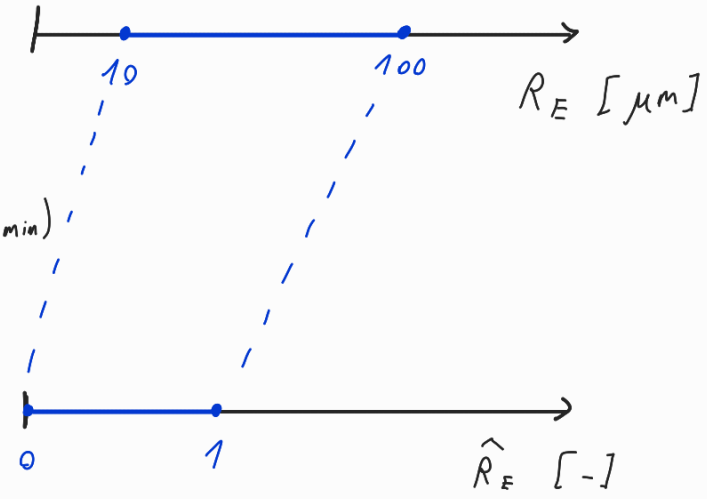
P_{amb} is kept constant 10^5 Pa

Note, that internally, the control parameters are transformed, to be between 0 and 1. In the transformed system, ranges is a multi dimensional unit cube.

$R_E \in [R_{min},\ R_{max}]$

$$\hat{R}_E = \frac{R_E - R_{min}}{R_{max} - R_{min}} \qquad R_E = R_{min} + \hat{R}_E \cdot (R_{max} - R_{min})$$

$\hat{R}_E \in [0,\ 1]$



This way a small change, e.g. $\delta \hat{R}_E = 0.001$ is dependent on ranges (e.g. $R_E \in [10\ \mu m,\ 100\ \mu m] \rightarrow \delta R_E = 0.09\ \mu m$

$R_E \in [10\ \mu m,\ 1000\ \mu m] \rightarrow \delta R_E = 0.99\ \mu m$)

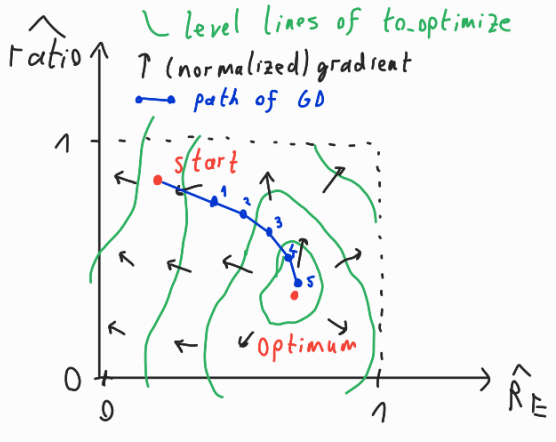
The step size (first_step, min_step, and delta) is also defined with these transformed coordinates:

$$0 < \text{step size} = \sqrt{\Delta \hat{R}_E^2 + \Delta \widehat{ratio}^2 + \Delta \hat{P}_{amb}^2 + \dots} < 1,$$

where Δ denotes the step along one control parameter.

When defining first_step and min_step, keep in mind, that if $\Delta = min\_step$, then the largest possible step is $\Delta R_E \leq \Delta \cdot (R_{max} - R_{min})$.

- How gradient descent (GD) works?



The basic idea behind GD is that the gradient of a function is always perpendicular to to level lines, and points in the direction of the greatest growth.

Thus, by always moving opposite to the local gradient, eventually a local minimum is reached. In practise, the gradient is calculated numerically, and discreete steps are taken.

- gradient with finite differences (delta)

$g(\underline{y}): \mathbb{R}^n \to \mathbb{R}$ function that returns the value of to_optimize (e.g. 'energy_demand' in MJ/kg of NH_3)

$\underline{y} = [\hat{R}_E^{\,1}, \text{ratio}^{\,2}, \hat{P}_{amb}^{\,3}, \dots^{n}]^T$ vector of transformed control parameters

y_i i^{th} element of $\underline{y}$ (e.g. here $y_1 = \hat{R}_e$), $i \in \{1, 2, \dots n\}$

δ small number (e.g. delta = 10^{-6}) alias difference

$$\underline{grad}\,(g(\underline{y})) = \begin{bmatrix} \frac{\partial g(\underline{y})}{\partial y_1} \\ \vdots \\ \frac{\partial g(\underline{y})}{\partial y_n} \end{bmatrix}$$

forward difference: $\frac{\partial g(\underline{y})}{\partial y_i} \approx \frac{g(y_1, \dots y_i + \delta, \dots y_n) - g(\underline{y})}{\delta}$

- n+1 evaluations of g() for gradient
- bigger numeric error
- default

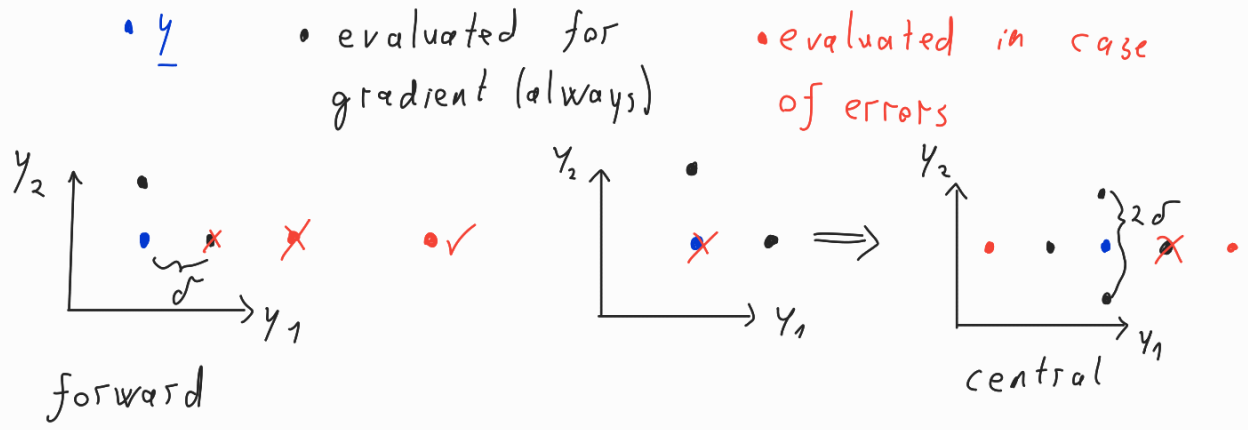
central difference: $\frac{\partial g(\underline{y})}{\partial y_i} \approx \frac{g(y_1, \dots y_i + \delta, \dots y_n) - g(y_1, \dots y_i - \delta, \dots y_n)}{2\delta}$

- 2·n evaluation of g() for gradient
- smaller numeric error

error management:

(sometimes $g(\underline{y})$ fails due to timeout or convergence failiure)

- If a forward point fails, then δ is doubled for the current finite difference. (maximum 2 times)
- If the central point fails, then the central difference is used. (δ doubling max 2 times)
- If everything fails, the last step is repeated. (maximum 2 times, otherwise GD terminates)



– step size control (first_step, min_step)

A step is always opposite to the local gradient:

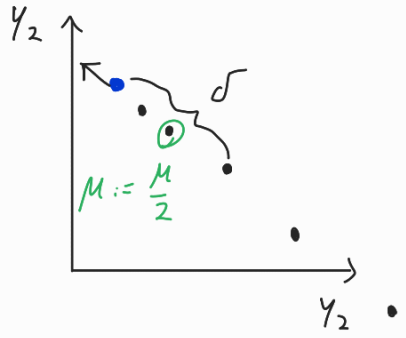
μ: step size, $\mu \in (0, 1)$

$\underline{y}[k]$: k^{th} step

$$\underline{y}[k+1] = \underline{y}[k] - \mu \cdot \frac{\underline{\text{grad}}\,(g(\underline{y}[k]))}{|\underline{\text{grad}}\,(g(\underline{y}[k]))|}$$

gradient is normalized

Initially, μ = first_step. During each time, 5 trial steps are taken: $\frac{1}{4}\mu$, $\frac{1}{2}\mu$, μ, 2μ, 4μ. Function $g()$ is evaluated in each of the trial points, and the step size corresponding to the smallest value is used:

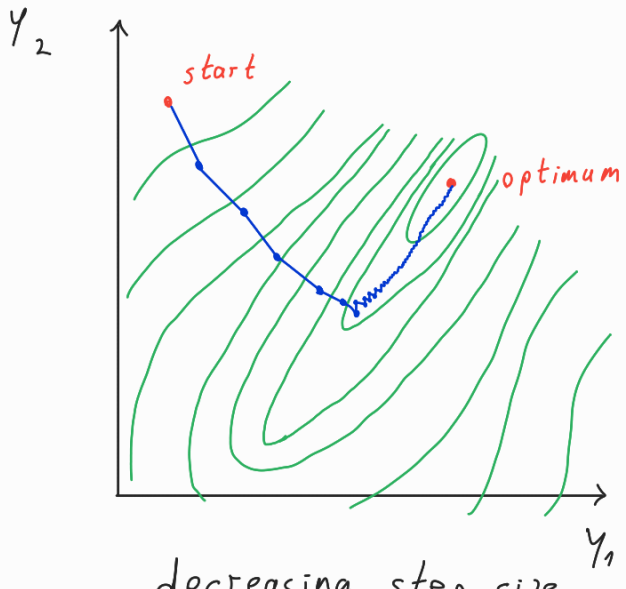


- current point
- gradient
- trial points
- best trial point

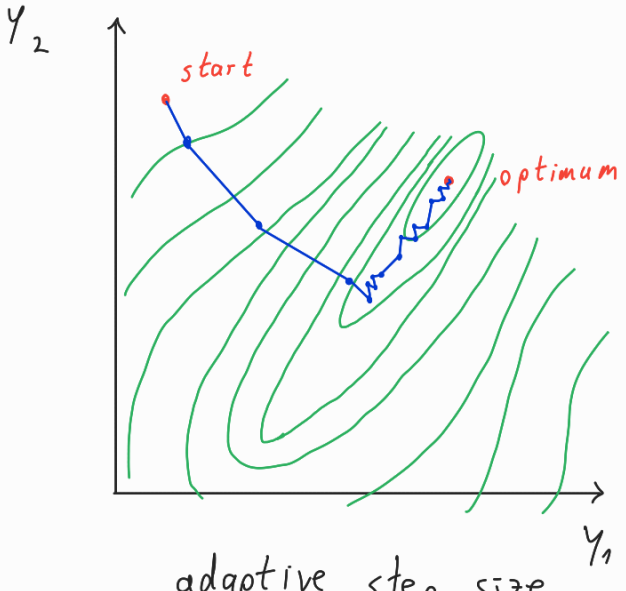
This way, the step size can both decrease for finer tuning (e.g. near the optimum) and increase for faster movement. In narrow elongated walleys, the GD quickly moves into the walley, but doing so the step size is greatly reduced. Thus, a simpler reducing step size control would barely move forward, and even this adaptive step size control requires several steps:

GD path

level lines of GD



decreasing step size control



adaptive step size control

– why norm the gradient?

In the original GD (e.g. stochastic gradient descent in machine learning) there is no step size control, the size of the gradient is used instead.

On the other hand, energy_demand can change several orders of magnitude during one step, while it can improve less than a percent during other steps:

- **wrong places**: energy_demand ~ $10^{10} - \infty$ MJ/kg, the resultant target_specie is zero or numeric noise. The gradient is unreilable if it doesn't fail. Shouldn't start or venture in here.
- **steep places**: energy_demand ~ $10^{3} - 10^{10}$ MJ/kg and it can change an order of magnitude per step. Gradient is huge.
- **walley**: energy_demand ~ $10^{2} - 10^{5}$ MJ/kg. Close the optimum, the step size is small, and a step means less than 1% improvement. Gradient is small.

– STOP conditions (min_step, step_limit)

The GD search will terminate if

- the step size is redured below min_step
  (normal operation, the GD is near an optimum or a very fine detail)
- step count is greater than step_limit
  (search terminates before reaching any optimum)
- calculation of the gradient fails repeatedly
  (fatal error, GD is possibly not even remotely close to an optimum — this actually never happened before)

– Steps are leaving ranges

When GD would leave the boundary, the step is squized to remain within ranges.

$y_i \in [0+10\delta, 1-10\delta]$ is garanteed.

- How to initiate start_point?

A GD search should not start in wrong places, where the gradient cannot be approximated well. Also, GD will move towards the nearest local optimum. To find the global optimum, you should start from several spread out points. These points don't have to be near the optimum, 10^5–10^{10} MJ/kg is also sufficent. You have 2 approaches:

  - Do a coarse brute force parameter study along the more important control parameters. Use the best 5–100 results as start points.
  - (As in example) Evaluate in several random trial point, and choose the best 5–100. I recommend 200–500 trial points in 2–3 dimensions, 1000–10 000 with more dimensions.

- Settings of gradient_descent()

gradient_descent(

ranges, – according to 1st chapter

path, – location to save resulting CSV files. If None, a list of datas is also returned.

to_optimize, – key in data dict to minimize, usually 'energy_demand'

start_point, – according to last chapter

step_limit, – between 50–200, if you want results fast
  300–500, if you want most searches to converge
  even higher, if the majority of searches still don't fully converge

first_step, – you shouldn't step over the optimum. As the step size can grow, you may start small, e.g. 0.001

min_step – determines the final precision of the optimum (see 1st chapter). A small min_step leads to much longer searches, a too large leads to premature termination. (Near the optimum, in the "valley", very small steps are taken.) I recommend 10^{-4} for fast results, 10^{-5} for good convergence.

delta – should be an order of magnitude smaller, than min_step. should be greater than 10^{-6} to avoid numeric noise.

from full_bubble_model.solve():

t_int – simulation time interval, default is [0, 1] sec. the bubble should settle, after that the computational cost is negligible due to adaptive time steps.

LSODA_timeout, **Radau_timeout** – You should have a rough idea about the runtimes. If you observe too much timeouts in Read_CSV_files.ipynb then increase them. Use Radau_timeout=0 for faster results.

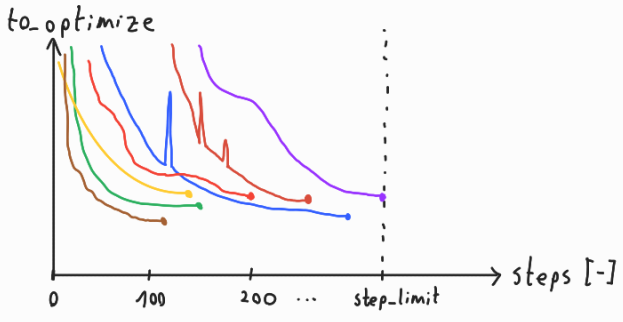
verbose – prints debug data, if True. use only with one GD at a time

- Verify convergece

  - Convergence of to_optimize

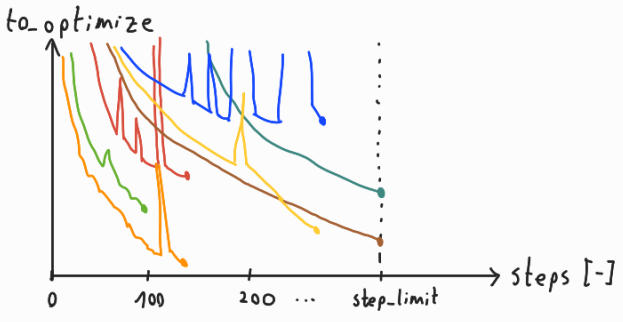
    Included in Gradient_descent.ipynb example. This plot shows, how the value of to_optimize decreases as function of steps:

✓ Good



- most searches terminate before reaching step_limit
- searches seemingly converge (flatten)
- the final values of to_optimize are close to eachother

✗ Bad

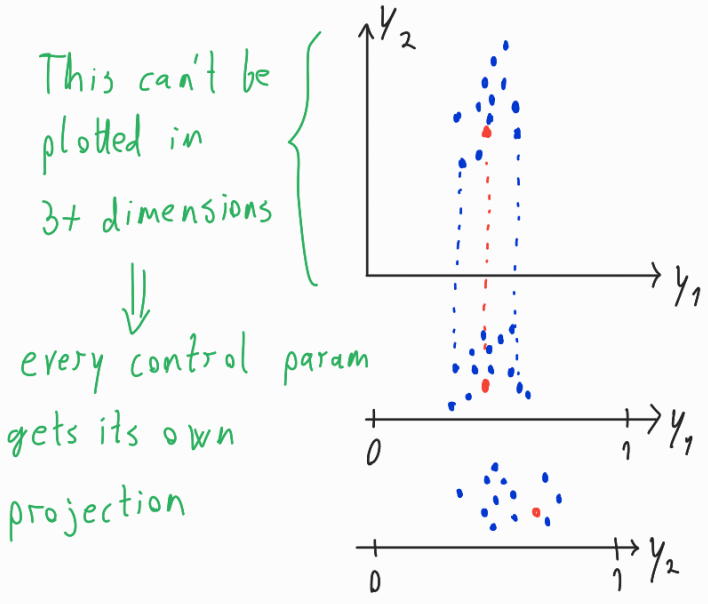


- searches reach step_limit (step_limit is too small)
- searches terminate, while to_optimize is still rapidly decreasing (too large min_step or too small timeouts)
- searches diverge (too large/small

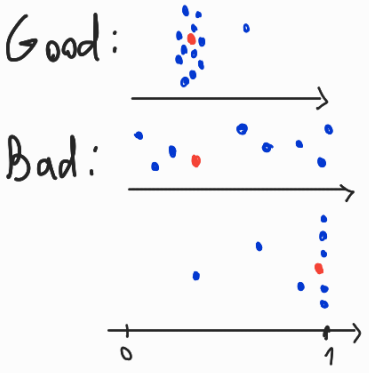
delta or instable/invalid simulations)

- Locations of optimums

Included in Gradient_descent.ipynb example. This plots show the deviation of the found optimums (last steps)



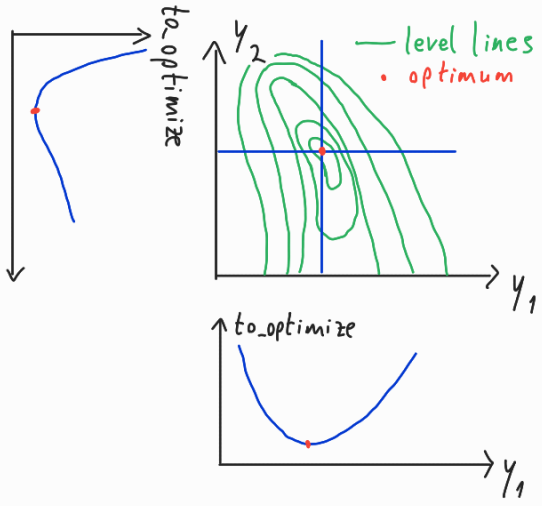
- Found optimums
- Best optimum found



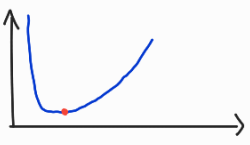
Most points should be close to eachother, and not at the edge of ranges.

- Quality of an optimum

Included in Create_plots.ipynb example. On these plots, all control parameters are fixed (at the optimum), and only one of them is changed at a time.



Good:



Bad: