1 Steepest Descent for Quadratic Case

I've written a code in `python` to generate a quadratic optimization problem at random and then solve it using *steepest descent* algorithm. It can be run using the following command:

```
$ python3 steepest_descent.py -n 100 -eps 1e-3 -seed 150
```

Parameters:

- `n`: the n in $\mathbb{R}^n \ni x$. This is optional, default value is 100. Type: integer.
- `eps`: the ε used for stopping criteria. Note: the stopping criteria is $\|\nabla f\| < \varepsilon$. This is also optional, if not provided the default value is taken to be 10^{-3}. Type: float.
- `seed`: all random number generators are *pseudo-random* sequence generators. Given a `seed` value, the random sequence is determined. This is good for repeatability. This is optional, if not set a new problem will be generated in each run. Type: integer.

I also wrote another code which is specialized for $n = 2$ and visualized the results in fig. 1. The code is for a particular problem (value of Q and b) only, since I had to fine tune the level set plots. Plotting level sets at a particular value of the function in python is not convenient. Run the code as (to check repeatability, and also you can zoom in on the tiny parts of the figure using the magnifying glass tool):

```
$ python3 steepest_descent_plot.py
```

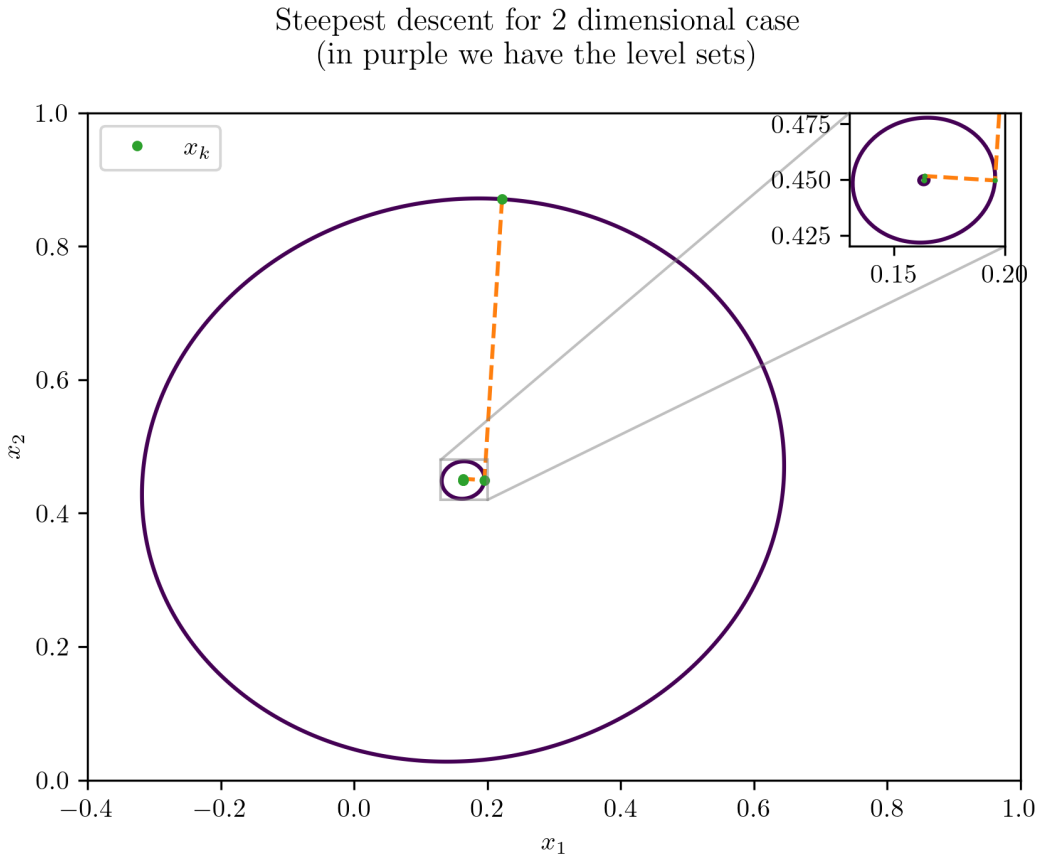


Figure 1: Movement of x^k towards the optimum value

I then ran this code for $n \in \{2, \ldots, 100\}$ and plotted the number of iterations it takes for the method to converge (fig. 2). The number fluctuates a lot (because each problem is different), but it sometimes takes steepest descent more than 6000 steps to converge where Newton's method would have taken just 2.

I also plot the distribution of the number of iterations it takes steepest descent to converge for $n = 30$ from 500 runs of SD (fig. 3).

I then run the steepest descent algorithm for different n but keep the condition number fixed (and $= 1000$ which is very large), we see that ill conditioning results in increase of iterations (fig 4).

Naturally, I also ran the algorithm for varying values of n when the condition number is good. See fig. 5. The number of iterations are less by a factor of 1000.

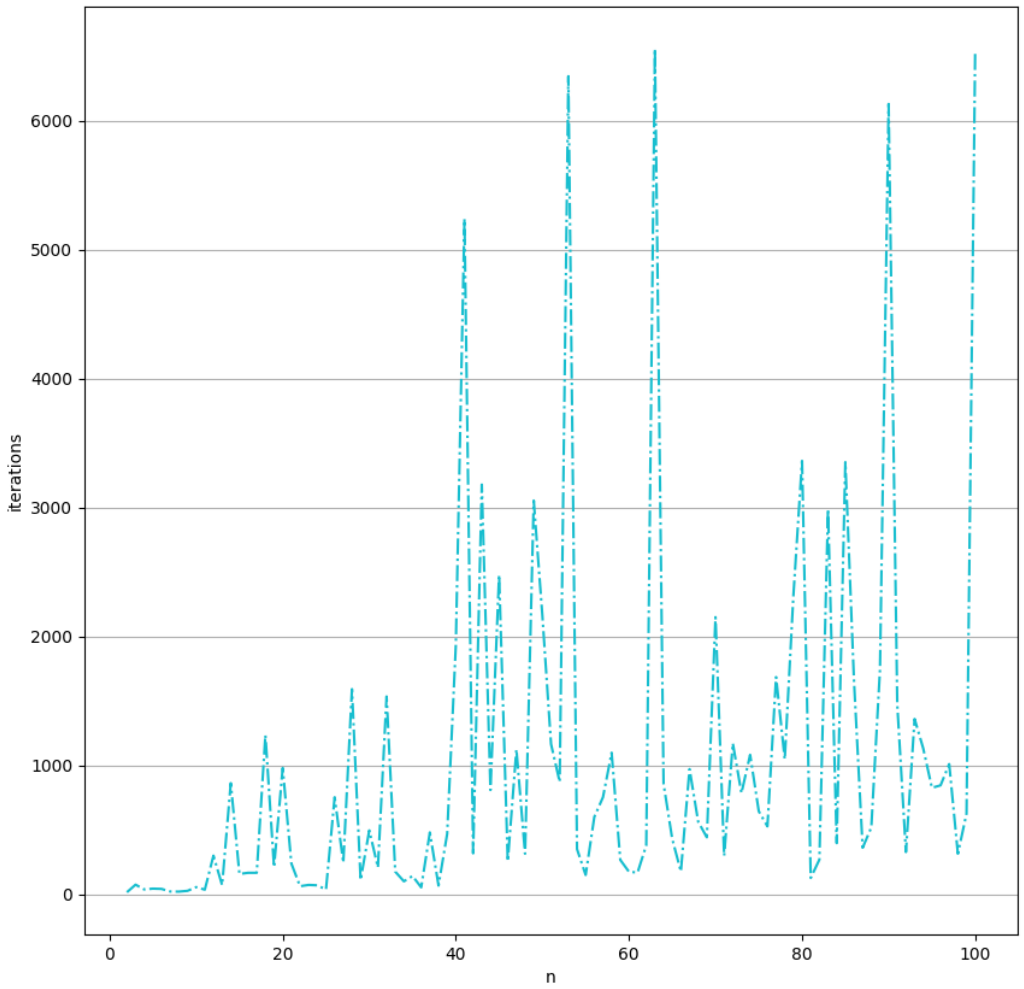


Figure 2: Number of iterations for convergence with varying values of n

Figure 6 shows how the function value evolves with each iteration. We can see that the function value drops quickly in the first few iterations but the progress quickly gets stagnant. The function value almost stays around the same region for thousands of iterations before convergence (and note, the converged value is still not exact.)

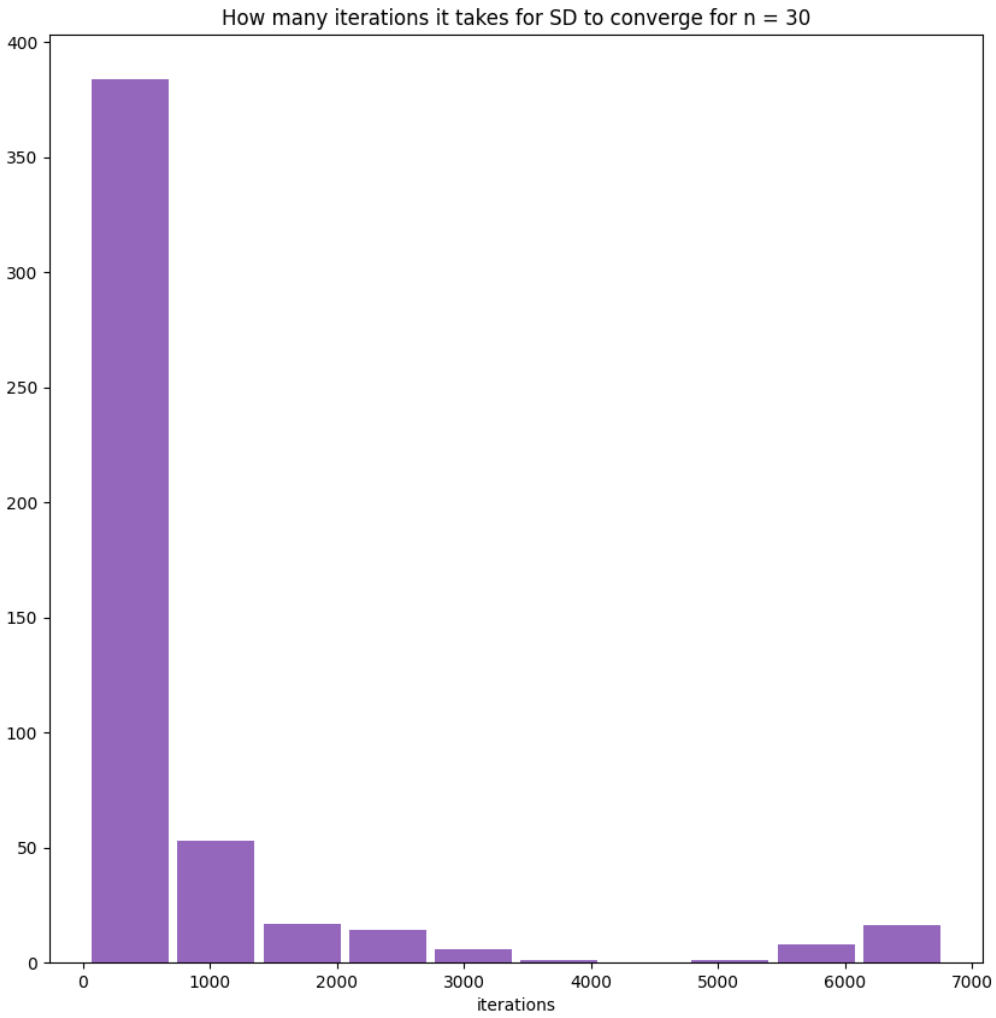


Figure 3: Number of iteration distribution for $n = 30$

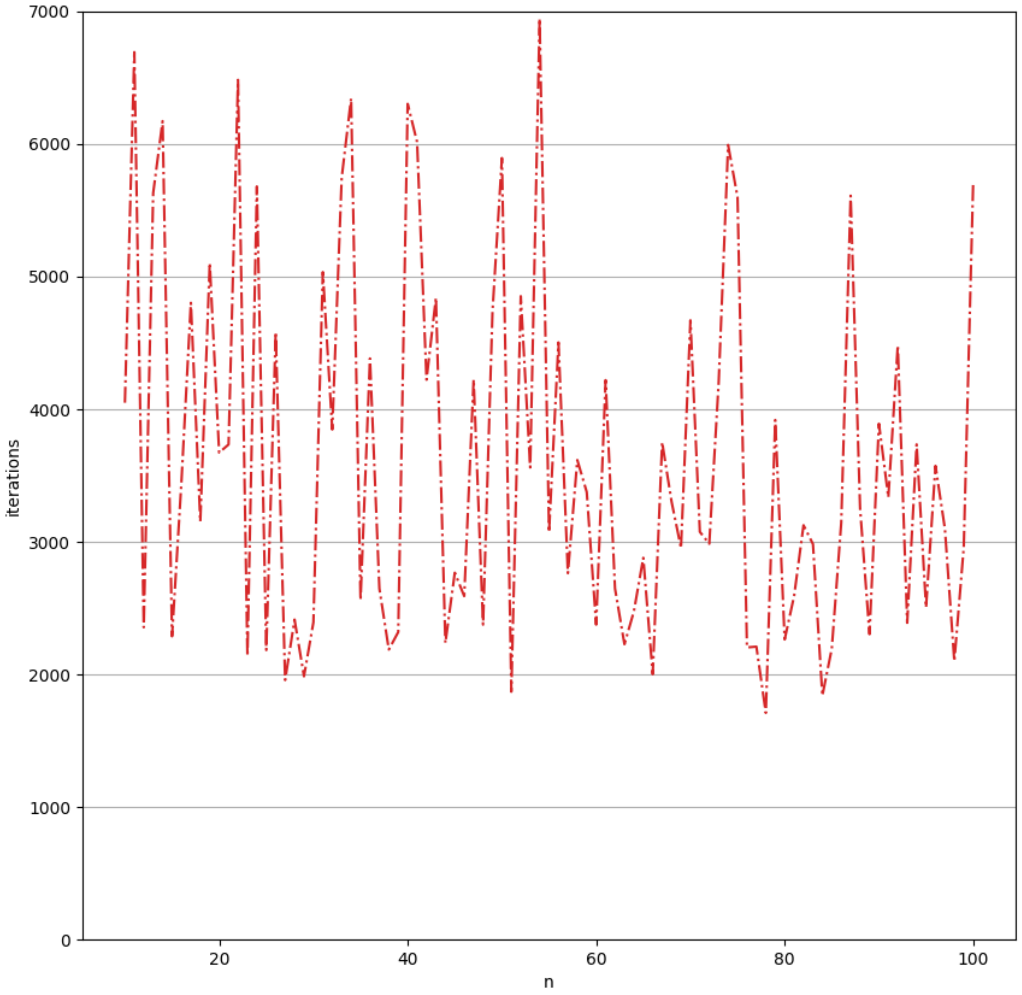


Figure 4: Number of iterations for convergence with varying values of n with a fixed value of condition number $r = 1000$

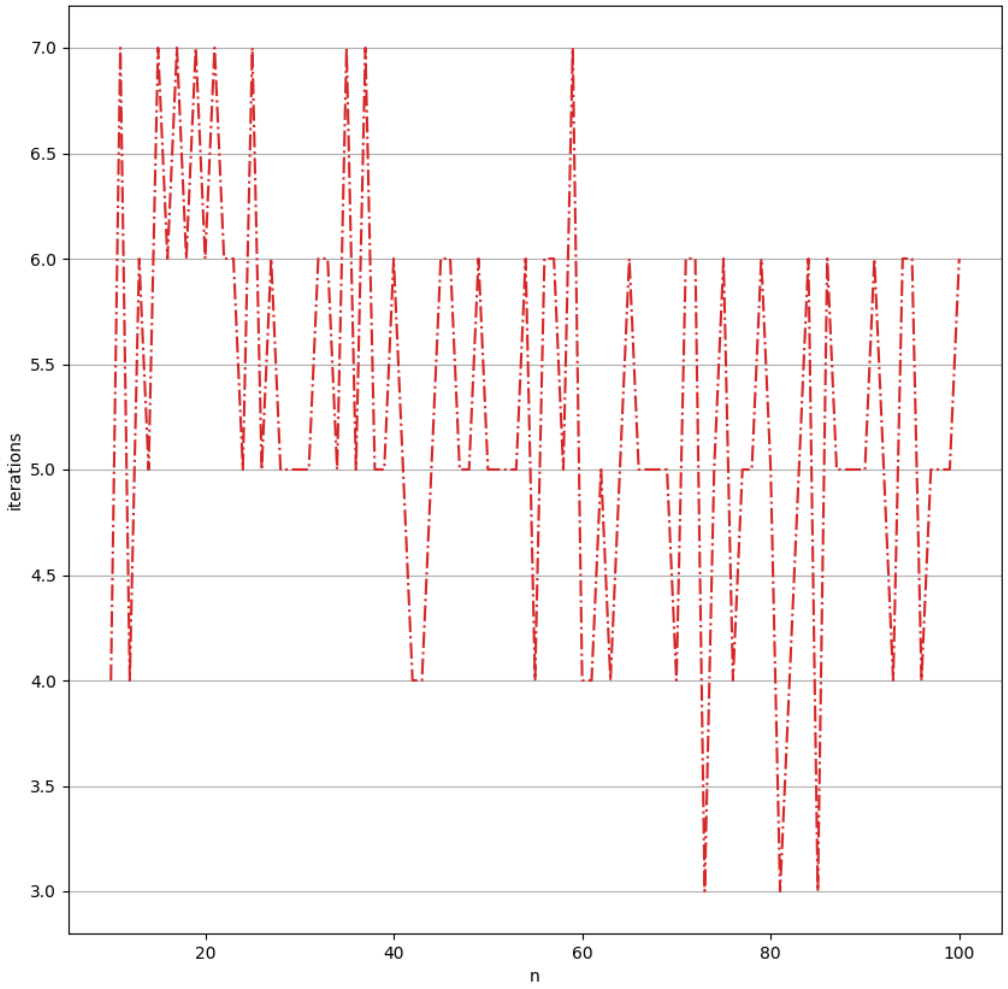


Figure 5: Number of iterations for convergence with varying values of n with a fixed value of condition number $r = 1.2$

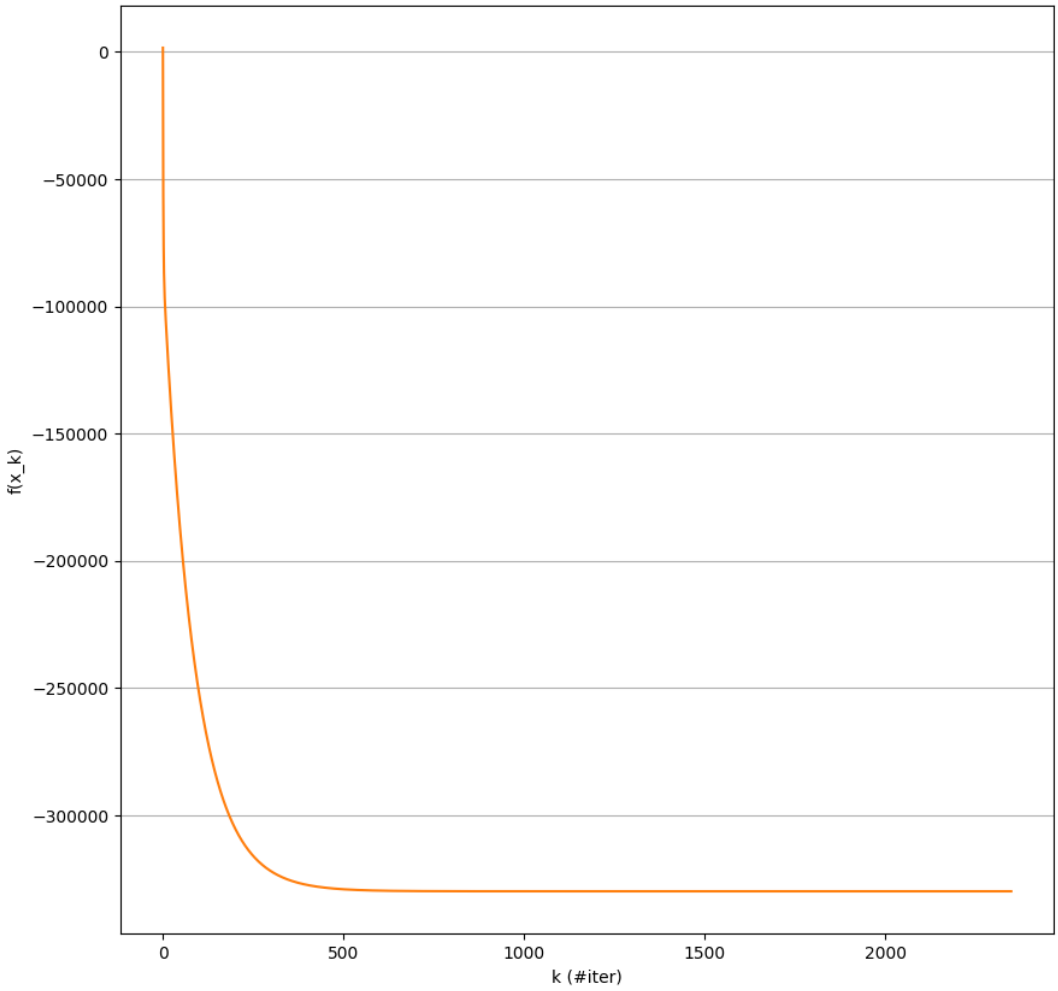
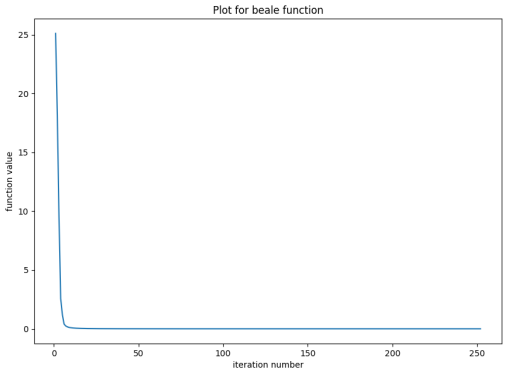
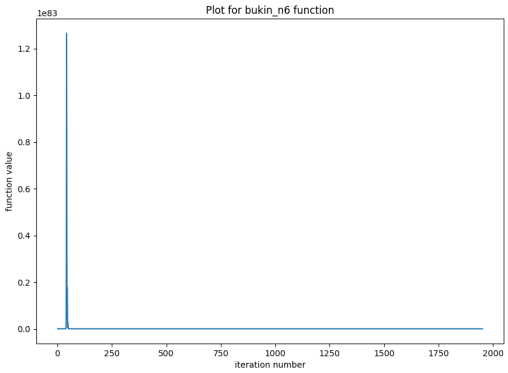


Figure 6: Evolution of the function value with each successive iteration



(a) Minimum value = 0, function is multimodal



(b) Minimum value = 0, function has many local minima

Figure 7: (a) converges while (b) just completely fails

2 Steepest descent for general convex functions

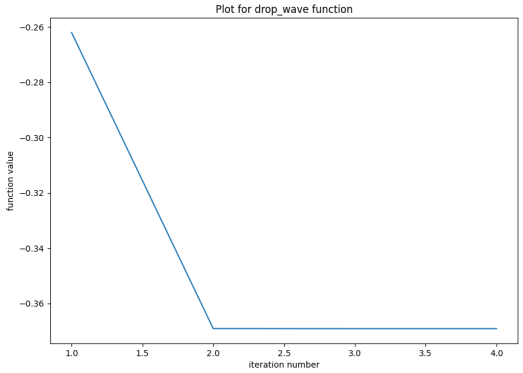
I've implemented steepest descent algorithm for a general convex function. I've outsourced gradient calculation to Harvard's `autograd` library[1]. I've implemented a few of the benchmark functions from the provided list, these functions reside in the `functions.py` file. The code can be run as:

```
$ python3 general_sd.py -list -function <function-name> -n <dim> -eps 1e-3 -seed 0
  -alpha_hat 1e-3
```

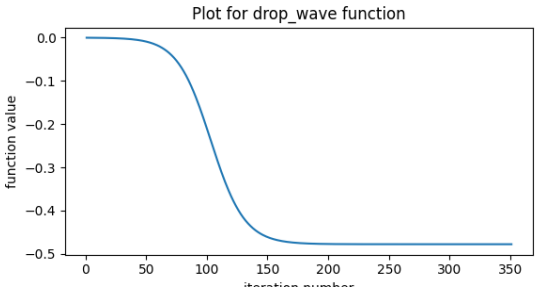
Parameters:

- `list`: list the implemented benchmark functions
- `function`: benchmark function's name on which steepest descent will be run
- `n`: the n in $\mathbb{R}^n \ni x$
- `eps`: the ε used in stopping condition $\nabla f(x) < \varepsilon$
- `seed`: same as described previously
- `alpha_hat`: starting estimate of $\hat{\alpha}$

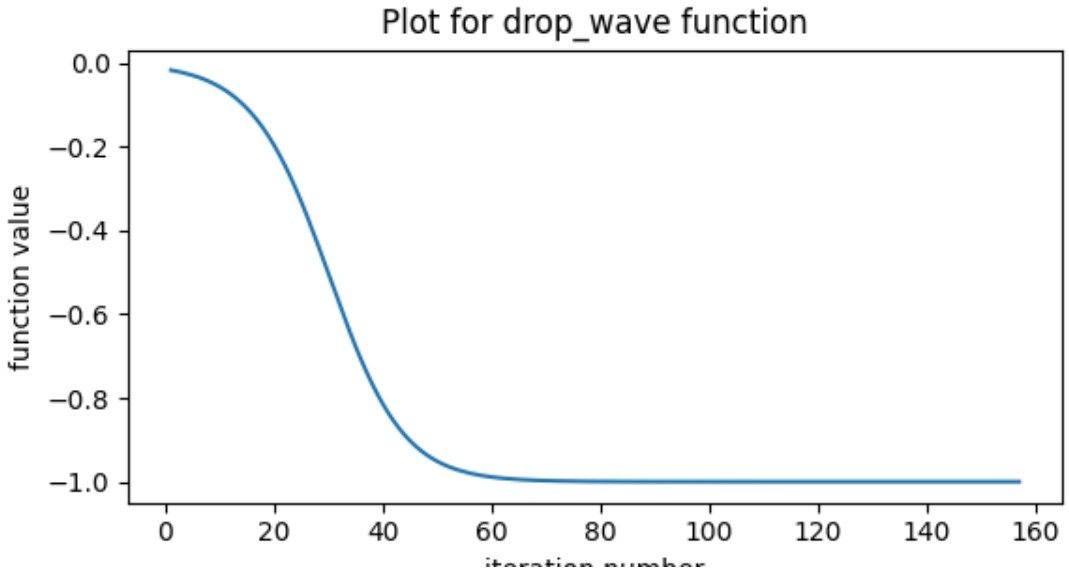
[1] `https://github.com/HIPS/autograd`



(a) Converges to local minima

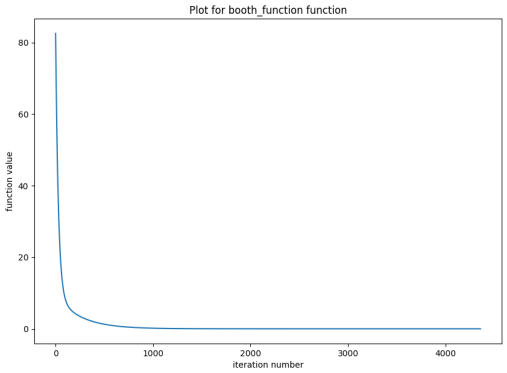


(b) Converges to local minima, but the function curve is very interesting

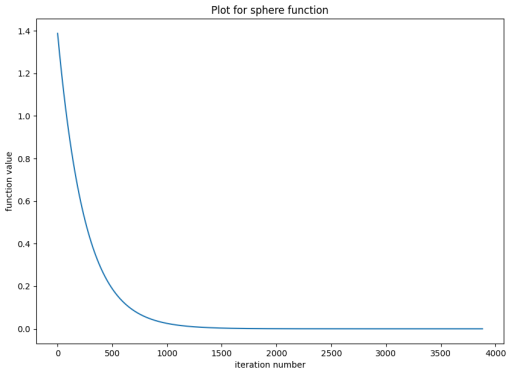


(c) Converges to global minima

Figure 8: Global minima at 0, function has many local minima.



(a) Minimum value = 0, function is plate shaped



(b) Minimum value = 0

Figure 9: Convergence in well behaved functions