

```
1: import numpy as np
2: import sympy as sp
3:
4: def gradient_function_fd(minibatch, epsilon=10**(-15)):
5:     def gradient_fd(x):
6:         dydx1 = (f(x + np.array([epsilon, 0]), minibatch) - f(x, minibatch)) / epsilon
7:         dydx2 = (f(x + np.array([0, epsilon]), minibatch) - f(x, minibatch)) / epsilon
8:         return np.array([dydx1, dydx2])
9:     return gradient_fd
10:
11: def loss(x, w):
12:     z = x - w - 1
13:     left = 10 * (z[0]**2+z[1]**2)
14:     right = (z[0]+2)**2+(z[1]+4)**2
15:     return min(left, right)
16:
17: def f_clear(x, minibatch):
18:     return sum(loss(x, w) for w in minibatch) / len(minibatch)
19:
20: def generate_minibatches(T, n=5, seed=42, shuffle=True):
21:     if shuffle:
22:         T = T.copy()
23:         np.random.seed(seed)
24:         np.random.shuffle(T)
25:     num_rows = T.shape[0]
26:     i = 0
27:
28:     minibatch = np.zeros((n, T.shape[1]), T.dtype)
29:     while True:
30:         for j in range(n):
31:             minibatch[j] = T[i % num_rows]
32:             i += 1
33:             if shuffle and i >= num_rows:
34:                 # begin next epoch
35:                 np.random.shuffle(T)
36:                 i = 0
37:         current_minibatch = minibatch
38:         yield minibatch
39:
40: def generate_optimisation_functions(batch, minibatch_size=5, finite_difference=True, **kwargs):
41:     minibatch_generator = generate_minibatches(
42:         batch, n=minibatch_size, **kwargs)
43:     for minibatch in minibatch_generator:
44:         def optim_func(x):
45:             return f_clear(x, minibatch)
46:         gradf = gradient_function_fd(minibatch)
47:         yield (optim_func, gradf)
48:     yield "finished"
```