# Implementation and Evaluation of the Gradient Descent Algorithm

## Problem and Background Information

The task for this assignment is to implement and evaluate the gradient descent algorithm in a one dimensional function, i.e. a function on the real line $y = f(x)$. Note that gradient descent is typically used in functions with more independent variables (multidimensional functions), which leads to more interesting applications, but in this case the gradient will simply be the derivative of $f(x)$.

The gradient descent algorithm is typically used to find the minimum of a function $y = f(x)$ for which the equation $f'(x) = 0$ has no closed form solution. However, when developing new optimisation algorithms (gradient descent is an instance of an optimisation algorithm) or when implementing one, it is often useful to test the algorithm with a function with known extrema. The benefits of doing so are two. Firstly, we know what the solution should be and we can assess how good is the approximate solution given by the algorithm. Secondly, it allows to trace, i.e. debug step-by-step the algorithm and compare, for example, two different parameter values of the algorithm to assess which one leads to a better result (i.e. which one is closer to the optimum/minimum).

In this coursework we will find the extremum of the following function on the real line:

$$y = -20e^{-\frac{x^2}{8}} - e^{\frac{1}{2}\cos(2\pi x)} + 20 + e \tag{1}$$

which is a variant of the *Ackley function* used to test multi-dimensional optimisation algorithms.

## The task

Although all the extrema of the function (1) cannot be calculated in closed-form, i.e. in general $y' = 0$ has no closed form solution, there is a critical point at $x = 0$. The first step in this assignment is to show that the function has indeed a critical point at $x = 0$ and see whether it is a maximum or a minimum using the conditions on the first and second derivatives.

Once we know that the function has a extremum the task is to implement (in Python) the gradient descent algorithm to numerically find the extrema. The gradient descent algorithm requires an initial guess of the minimum, so we have to pretend we do not know what the point is and test the algorithm from different starting points. Furthermore, this algorithm finds local minima in the vicinity of the initial guess, which means that, depending on your initial guess, if the function has other minima the gradient descent algorithm might not find the minimum at $x = 0$.

There are several approaches you can take for the implementation.

- **Gradient calculation**: Because the function is relatively simple and its mathematical expression is known you can use the analytical derivative as the gradient, i.e. calculating the derivative by hand and implementing it in your code. An alternative is using finite differences to approximate the derivative, $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ which involves finding a good value of $h$.

- **Finding the step size**: The gradient descent algorithm has one parameter that needs tuning, the step size $\alpha$ (a.k.a. learning rate). Finding an appropriate value of $\alpha$ can be challenging. A large value of $\alpha$ makes the algorithm converge fast towards the optimum, but it could also make the algorithm to diverge. A small value of $\alpha$ ensures the algorithm always converges, but in turn it will require many iterations. Therefore, there is a trade-off between convergence speed and accuracy when trying to find a good value of $\alpha$.

  Because typically $\alpha$ is a number between $0$ and $1$ a good strategy is to use some sort of 'manual' binary search, i.e. trying for instance $\alpha = 0.5$. If the algorithm converges too 'slowly' we can increase $\alpha$. If the algorithm diverges and/or oscillates around a point we need to decrease $\alpha$. If we do not have the slightest clue what a good values of $\alpha$ could be, a good approach is to try different orders of magnitude for $\alpha$, for instance $\alpha = 0.5$, $\alpha = 0.05$, $\alpha = 0.005$ and so on, and see which value leads to good convergence.

  The best value of $\alpha$ depends on the current estimate of the optimum $x_k$, and can be calculated using Newton's Method for root finding on the derivative $f'(x)$. If the current estimate of the extremum is $x_k$ (at iteration $k$) the optimal value of $\alpha$ is $\alpha_k = \frac{1}{f''(x_k)}$ (note the step size now is different for each iteration hence the subindex $k$ in $\alpha_k$). Although this value is optimal it has a problem, it can find a maximum instead of the minimum we are looking for (depending on the initial guess of the extremum).

- **Stopping criterion**: This is another important issue in the gradient descent algorithm as we want to obtain an accurate solution in a reasonable time. As the estimate of the extremum $x_k$ gets closer to the solution the derivative gets closer to zero. However, depending on whether we approach the extremum from the right or from the left the derivative can be a small positive number or a small negative number. Therefore we need to test for the absolute value of the derivative as an ending condition for the algorithm. We also need to select a small number typically smaller than $0.001$, but that depends on the problem.

  As mentioned above, if the parameter $\alpha$ is too small or too big the algorithm will either take too long to converge or will never find the extrema (it will diverge). In case this happens it is a good practice to set an additional stopping criterion based on the number of iterations of the algorithm, i.e. limit the maximum number of $k$ steps, so that the program does not run forever. Typically the stopping criterion is a combination of both, a small derivative and the number of iterations, such that the one which is reached first makes the algorithm return.

# Report and Submission

You must submit through itslearning a (maximum) two pages report (doc/pdf) with your implementation and evaluation results and the code as an appendix (no page limit for the appendix). The report should also include details on the identification of the known extremum $x = 0$, i.e. how did you check whether it is an extrema and of which kind (maximum or minimum). You must provide details on the approach you took for the implementation, i.e.

gradient calculation, adjustment of the step size and value, stopping criteria, and any other information you might find relevant (e.g. does your algorithm find other extrema besides $x = 0$?, when does that occur?,...). You can evaluate the algorithm in different ways, using several random or fixed initial guesses of the extrema assessing the difference between the known extrema and the values returned by the algorithm, comparing execution times and convergence for different values of the step size,... Finally, you can write some concluding lines, limitations of your implementation and, if you can think of any, ways to improve it. A template for the structure of the report could be something like:

1. Problem statement
2. Analytical results on the extrema
3. Implementation details and technique selection
4. Assessment of the implementation
5. Conclusions and possible improvements