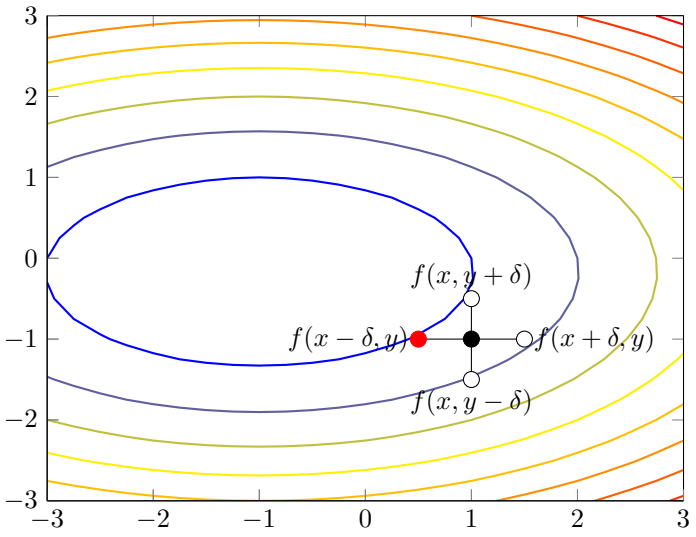
Function minimization

1 Function minimization

This exercise is about parallelizing a code that computes the minimum of a convex function $z = f(x, y)$ using a derivative-free method. This works iteratively starting at a point (x_0, y_0) and building a succession of points (x_k, y_k), $k = 1, \ldots$ until it gets reasonably close to the point that minimizes the function. Let's assume that at iteration k, the current candidate point is (x_k, y_k); the function is evaluated at the four surrounding points $(x_k \pm \delta, y_k \pm \delta)$ for a given parameter δ. Among these four, the one for which f is smallest is selected as the new candidate point (x_{k+1}, y_{k+1}), as illustrated in the figure below. We will make the assumption that $f()$ is a complex function which is costly to evaluate; f is also thread-safe, i.e., it can be evaluated simultaneously by multiple threads.

$f(x, y)$ and its gradient



If, at some iteration, none of the four surrounding points improves the solution, δ is divided by two and the process is repeated.

If, at some iteration, the improvement of the solution is smaller than 0.1% or a maximum number of iterations `maxit` is reached, the iterations are stopped.

2 Package content

In the `minimizer` directory you will find the following files:

- `main.c`: This file contains the main program and subroutines. This reads from command line maximum number of iterations `maxit`. Then it calls three versions of the minimization procedure setting the initial point (x_0, y_0). These are `minimize_seq`, `minimize_no_task` and `minimize_task`. For each of them, it prints the execution time and the result. **Only the `minimize_no_task` and `minimize_task` routines must be modified for this exercise.**
- `aux.c, aux.h`: these two files contain auxiliary routines and declarations and **must not be modified**.

The code can be compiled with the `make` command: just type `make` inside the `minimizer` directory; this will generate a `main` program that can be run like this:

```
$ ./main maxit
```

where `maxit` is the maximum number of iterations to perform.

3 Assignment

-  The objective of this exercise is twofold:

  1. Parallelize the `minimize_no_task` routine **WITHOUT using the OpenMP task directive**. In this case, make sure that enough parallelism is generate to achieve good performance using 2 and 4 threads.
  2. Parallelize the `minimize_task` routine **using the OpenMP task directive**.

  The `minimize_seq` is a sequential variant and is provided as a reference. Note that, at the beginning, the three routines are exactly the same. Moreover, after the parallelization, **they must produce exactly the same result**, i.e., find the same minimum.