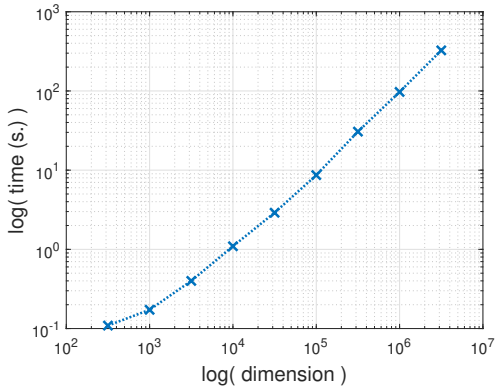

Figure 1: Time taken by our QN method (forward estimate only, $N = 25$) to perform 500 iterations VS dimension. The problem is a random logistic regression. The algorithm scales well even for large scale problem ($d \approx 5 \cdot 10^6$): as predicted by the theory, the log-log plot exhibits a clear linear dependence between the time taken and the dimension. The limitation was the memory allocation of Matlab when creating the logistic regression problem.

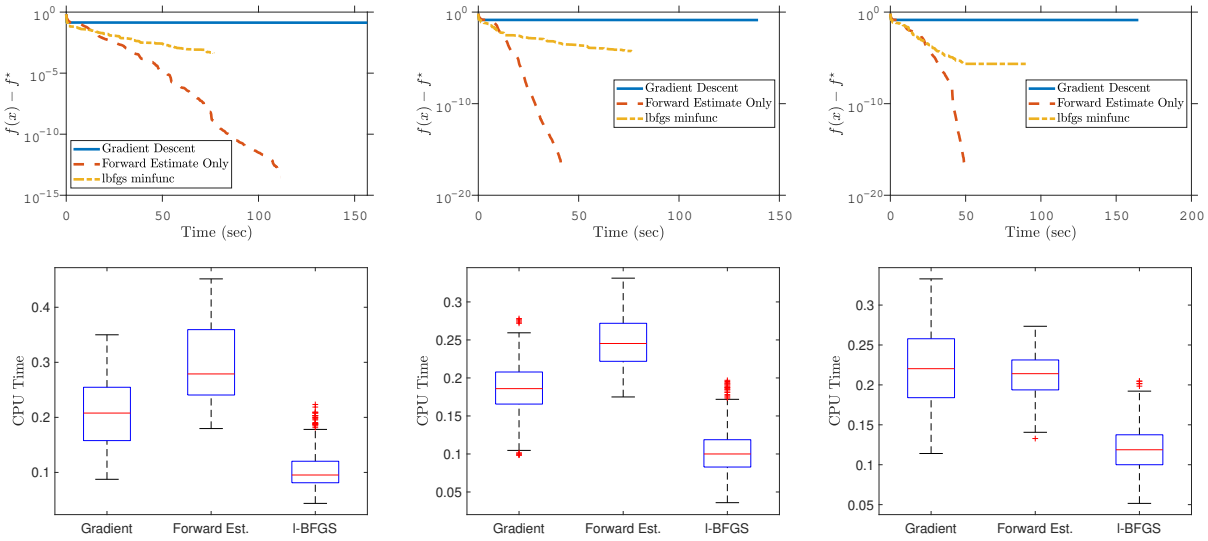

Figure 2: Time comparison between the Gradient method, L-BFGS, and the QN method presented in our paper (orthogonal forward estimate only). The problem is a logistic regression on the dataset Sid0. **Top**: time comparison VS accuracy. **Bottom**: Box plot of the per-iteration time for each method. **(Left to right)** $N = 5, 25, 100$, where N is the memory parameter of our method and L-BFGS. Gradient descent iterations are slower than BFGS for two reasons: the backtracking line-search for GD, combined with keeping track of function values, may take up to $4\times$ the L-BFGS time. Overall, the time taken by the forward estimate only is comparable to gradient descent, and is surprisingly worse when N is smaller. After investigation, this is because the condition in the backtracking line-search is always ensured when the algorithm is close to $x^\star$, introducing less computation (this indicates a potential super-linear convergence regime). Overall, our algorithm scales well with N and is much better than L-BFGS, despite being a (very) suboptimal implementation.