VXQR: Derivative-free unconstrained optimization based on QR factorizations

Arnold Neumaier, Hannes Fendl, Harald Schilly, Thomas Leitner

(University of Vienna, Austria)

We consider the unconstrained optimization problem of minimizing a real-valued function f defined on a subset of $\mathbb{R}^n$ by an oracle that returns for a given $x \in \mathbb{R}^n$ the function value $f(x)$.

In addition, scaling information is assumed to be available in the form of a search box

$$\mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\}.$$

The expectation (which may or may not be true) is that the minimizer of interest lies in the interior of the search box; but the search may lead out of the box.

The best point found is therefore not guaranteed to lie in the box.

Two traditions in BBO

1. **Evolutionary Computing (EC)**
2. **Derivative-free optimization (DFO)**

Both address the same problem but from very different perspectives!

1. Archetype:
Evolution of Darwin finches, Drosophila flies, etc.

- **Mutations and Crossover of generic information determines the children**
- **Less fit individuals are put to a disatvantage**
- **Wait long enough**

(millions of years in the motivating instances)

$\Rightarrow$ **evolutionary computation (EC)**

- **(Very) slow but sure – with probability one**

2. Archetype:
Evolution of cars and air planes

- **Model building and surrogate optimization determines the children**
- **Children are *designed* for likely high fitness**
- **Extract info from a (good or bad) model of the fitness landscape**

$\Rightarrow$ **Derivative-free optimization (DFO)**

- **Fast but may get stuck in local minima**

But with a limit $nfmax$ on the number of function values, the differences are not so big.

Two recent large-scale studies:

BBOB 2009 (Hansen et al.): $n \leq 40, nfmax \leq 10^6 n$
(appropriate when function values are cheap)
144 test problems from EC background

Rios/Sahinidis 2009: $n \leq 300, nfmax \leq 2500$
(appropriate when function values are expensive)
250 test problems from DFO background

BBOB 2009 (Hansen et al.): $n \leq 40, nfmax \leq 10^6 n$

- **31 solvers, tests done by their authors**

$n = 2, nfmax = 1000$**:**
Nelder-Mead with restart is best (99% solved)

$n > 2, nfmax \leq 500n$**:**
NEWUOA, MCS, GLOBAL are best (all DFO)

$n > 2, nfmax = 10^6 n$**:**
variants of CMA-ES are best (all EC)

- **For** $n = 20$**, about 70% of the test problems were solvable with** $nfmax = 10^6 n$**.**

Rios/Sahinidis 2009: $n \leq 300, nfmax \leq 2500$

• 15 solvers, tests done independent of their authors

MCS best for small $nfmax$

LGO (no free access) best for larger $nfmax$

• Best solvers are of DFO type

• 2010:
New TOMLAB DFO solvers are usually even better than both MCS and LGO

• No open-source description available

The message is clear:

For expensive functions, getting a fast improvement via DFO is essential, But these algorims are not very robust in the long run.

Remarkably, the best techniques of DFO and EC seem to converge towards each other.

CMA-ES evolves not individuals but stochastic models of the domain of attraction, and hence amounts to a stochastic model-based solver.

It is remarkably robust, far more than any of the traditional DFO algorithms. But it is initially extremely slow and useless for expensive problems.

It seems possible to look for the best of both worlds combined!

My group in Vienna is working towards that, by adding stochastic features to otherwise traditional DFO techniques.

The present results are encouraging, though we are still far from the desired goal.

The algorithms to be described are designed for the case when

• a call to the oracle is fairly expensive, so that

• one wants to keep the number of function evaluations needed as small as possible,

• aiming for a rapid decrease of the objective function rather than for necessarily reaching the global minimum.

Such a fast decrease is achievable only if the dimension of the problem is small, or if the function to be optimized has an appropriate structure that can be exploited by the algorithms.

In particular, our algorithms are based on the expectation that either

- **the objective function resembles a separable function, so that the variation of single coordinates is a reasonable option, or**
- **that the function is smooth (twice continuously differentiable) and not strongly oscillating along typical rays.**

This enables one to make fast progress using line searches, a basic tool from traditional deterministic optimization algorithms.

However, we also exploit the robustness that can be gained from a stochastic approach, and thus combine deterministic and stochastic features in a novel way.

Thus none of the structural expectations is essential for the algorithm to work, although the performance of problems not matching the stated expectations may be erratic or poor.

VXQR stands for

valley exploration based on QR factorizations

The name emphasizes the linear algebra tool (QR factorization) that dominates the part of the execution cost of the algorithms that is independent of the cost of the function evaluation routine.

The VXQR class of algorithms we consider proceed in several phases.

After the initial scaling phase 1, scout phase 2 and subspace phase 3 alternate (according to a tunable strategy) until a stopping criterion is reached.

All phases crucially depend on well-implemented line searches; it also matters for performance which kind of line search is employed at which stage.

VXQR1 is a particular realization of this general scheme, chosen in a trial-and-error fashion through experimenting with various strategies and tuning parameters.

For our experiments, we used randomiized test problems of various forms and dimensions to check the quality of particular algorithms, with the number of function values bounded by 2500 or 25000.

The Matlab code of VXQR1 is available online.

Phase 1 (scaling phase)

Search for a well-scaled initial point x with finite $f(x)$.

This is simply done by a uniform random search in the specified search box, followed by a line search in the direction of the point in the search region with the absolutely smallest components to get an initial point with an appropriate order of magnitude.

Phase 2 (scout phase)

Search for a direction of good expected progress.

This is done by a sequence of line searches from the best point, either in coordinate directions, or in a direction chosen from an orthonormal basis that adapts itself during the course of the algorithm.

The orthonormal basis is created at the beginning of each scout phase by taking the columns of the orthogonal factor Q of a QR factorization of the matrix formed by the differences of the best point from the results of the scout line searches of the previous scout phase.

Phase 3 (subspace phase)

The direction generated by the scout phase is used to extend a low-dimensional search subspace until a saturation dimension (in VXQR1, this dimension is 10 when $n > 20$) is reached; in this case, the scout direction replaces an old subspace direction.

In the new subspace, a local quadratic model is created and minimized, subject to some safeguards. Then a line search is performed from the best point found so far to the model minimizer.

Line searches

Line searches are done by function evaluation from the best point along a specified direction.

This is the place where the algorithm expects that the objective function is smooth; performance is degraded if this is not the case.

We use two versions of the line search: a global one and a local one.

A global line search evaluates the function at a (tunable) number of points uniformly distributed on a random line segment along this direction containing the best point.

This is followed by safeguarded parabolic interpolation steps at the local minima of the grid function so determined, and possibly by piecewise-linear interpolation steps, to approximately locate local minima along the direction searched.

A local line search evaluates the function at a random point along the search direction, then reflects the worse point at the best point to get a third point on the line; this is repeated as long as the function value improves.

We then perform a single safeguarded parabolic interpolation.

If this did not lead to an improvement, a few further points on the line are evaluated according to heuristic criteria.

Because all line searches start from the best point available at the time, the VXQR algorithms have a greedy tendency.

This makes them efficient for a low number of function values, but puts them at a disadvantage for highly oscillating functions where the greedy strategy often confines progress to a small neighborhood of the best point, with escape to a different valley often being the result of pure chance rather than strategy.

We tested VXQR1 on the benchmark problems from the scalability test set of Herrera et al. in high dimension $n \in \{50, 100, 200, 500, 1000\}$, using an upper bound of $5000n$ function evaluations, as specified by this benchmark. Because of the stochastic nature of the algorithm, each optimization was repeated 25 times.

The benchmark consists of 19 functions F1–F19* given by explicit dimension-dependent expressions f_k $(k = 1 : 19^*)$ in an arbitrary dimension n.

The SCAL benchmark functions F1–F11 are traditional test functions shifted so that their global minimum is attained at a particular point specified by the benchmark for each of these functions. F12-F19* are artificial hybrid functions created from F1-F11.

Good cases

Within $5000n$ function evaluations, VXQR1 solved at least half of the 25 solution calls to high accuracy 7 problems F1 (Sphere), F4 (Rastrigin), F5 (Griewank), F6 (Ackley), F7 (Schwefel 2.22), F8 (Schwefel 1.2), and F10 (Bohachevsky) in dimensions $n \in \{50, 100, 200, 500, 1000\}$.

On these problems, VXQR1 matched or outperformed a number of other methods used for comparison, among others some good versions of DE and G_CMA-ES.

As expected, the good cases include

- **the smooth problems without large oscillations,**
- **the separable problems,**
- **and a few others.**

(F4, F5, and F6 are multimodal;
F1, F4, F6 and F7 are separable.)

show figures?

An exception was the smooth function F3 (extended Rosenbrock), which performed reasonably in the *best* runs in all dimensions except $n = 1000$, but not frequently enough to have good median results.

F3 is bimodal in dimensions $n > 3$, with global minimum value 0 and local minimum value close to 4, while typical function values are huge.

VXQR1 quickly gets into the banana-shaped valley connecting the two minima.

But it is not fast enough to move along the valley to the global minimizer.

The following table exhibits the number of function values sufficient to reach in at least half the cases the global minimum to an accuracy close to the limit accuracy, but at least 10^{-6} and thus gives an idea of how VXQR1 scales with the dimension when it is successful.

test function	**F1**	**F3**	**F4**	**F5**
function values	$10n$	$40n^2 + 250n$	$2000n$	$150n$
test function	**F6**	**F7**	**F8**	**F10**
function values	$400n$	$300n$	$1200n$	$800n$

Poor cases. On the remaining 11 problems, the performance of VXQR1 was fairly poor in all dimensions $n \in \{50, 100, 200, 500, 1000\}$.

The hybrid functions F15 and F19* (created from F10 and F7) were solved well in dimensions $n = 50$ and $n = 100$ but poorly in higher dimensions.

Since the nonsmoothness of F7 combines with the nonseparability of F10, the assumptions made in developing VXQR1 are no longer satisfied well enough to ensure scalability.

On the eight highly oscillatory problems, the greedy strategy of VXQR1 forced the progress path to take numerous tiny turns, resulting in slow but steady progress, but far too slow to come close to the global minimizer within the budget available.

On these problems, DE performed much better than VXQR1.

On the problem F2, which required to minimize

$$f_2(x) := \|x - z'\|_\infty = \max_{i=1:n} |x_i - z'_i|, \qquad (1)$$

many line searches were completely inefficient since they searched for improvements on a constant piece.

Attempts with the p-norm in place of the maximum norm – which corresponds to $p = \infty$ – showed that success could be achieved for increasing p up to $p = 20$, with deteriorating tendency.

On this problem, G_CMA-ES performed very well.

Conclusion

VXQR1 performs an approximate unconstrained minimization of a not necessarily smooth function of many continuous arguments. No gradients are needed. A limited amount of noise is tolerated.

The primary goal is to get in high dimensions quick improvements with few function evaluations, not necessarily to find the true global minimum.

The algorithm is typically very fast on smooth problems with not too rugged graphs, and on problems with a roughly separable structure.

For best results in high dimensions, it seems (given the limited number of test problems compared systematically) that one should use

- **VXQR1 on smooth problems and on nearly separable problems,**
- **DE on problems with large oscillations, and**
- **G_CMA-ES on problems that minimize the maximum of many functions.**

More information can be found at the VXQR1 home page at

`http://www.mat.univie.ac.at/users/~neum/software/vxqr1`