Matlab codes for optimization under unitary matrix constraint

Traian Emanuel Abrudan (joint work with Jan Eriksson and Visa Koivunen)
SMARAD CoE, Department of Signal Processing and Acoustics,
Aalto Univestity, Espoo, Finland

COPYRIGHT and TERMS OF USE:

These Matlab® scripts are available online at:
http://signal.hut.fi/sig-legacy/unitary_optimization/

Comments, questions and suggestions may be sent to abrudant@gmail.com

1 General description

This set of codes may be used to optimize a smooth (differentiable) cost function $\mathcal{J}(\mathbf{W})$ over the Lie group of $n \times n$ unitary matrices. The optimization problem with unitary matrix constraint is stated as:

$$\text{optimize } \mathcal{J}(\mathbf{W}) \text{ subject to } \mathbf{W}^H\mathbf{W} = \mathbf{W}\mathbf{W}^H = \mathbf{I}_n$$

where $\mathbf{W}$ is an $n \times n$ unitary matrix and $\mathbf{I}_n$ is the $n \times n$ identity matrix.

The codes we provide can be used to either minimize or maximize an arbitrary smooth cost function under unitary matrix constraint. The proposed Riemannian gradient algorithms and line search methods are implemented step-by-step, as in the Tables given in [1,2].

- The following gradient-based optimization algorithms are implemented:
  - SD/SA: Steepest Descent/Ascent (see Table 1 in [2]),
  - CG-PR: Conjugate Gradient (Polak-Ribière formula) (see Table 3 in [1])
  - CG-FR: Conjugate Gradient (Fletcher-Reeves formula) (see Table 3 in [1])
- The following line search methods are implemented and used together with the gradient algorithms:
  - Armijo step method (see Table 2 in [2]),
  - polynomial approximation-based method (see Table 1 in [1])
  - DFT approximation-based method (see Table 2 in [1])

All combinations of Riemannian gradient methods and line search methods proposed in [1,2] are tested. In practical applications, just one of these algorithms (SD/SA, CG-PR/CG-FR) together with one of the line search methods (Armijo, polynomial of DFT-based) is sufficient to solve the problem at hand. The algorithm/method to be used may be chosen based on experimental testing. The difference in performance may depend on the cost function. In general, based on our experience, first we recommend CG-PR with the polynomial-based line search method, and second, SD/SA with the DFT-based line search method.

As a test example, the Brockett cost function: $\mathcal{J}_{\mathrm{B}} = \mathrm{trace}\{\mathbf{W}^H\mathbf{S}\mathbf{W}\mathbf{N}\}$ is minimized/maximized, where $\mathbf{S}$ is an n-times-n positive Hermitian matrix, and $\mathbf{N}$ is a diagonal matrix whose diagonal elements are $1, \ldots, n$. The solution $\mathbf{W}$ of this optimization problem is known, i.e. it is given by the eigenvectors of $\mathbf{S}$. Therefore, the matrix $\mathbf{W}^H\mathbf{S}\mathbf{W}$ will be a diagonal matrix that contains the eigenvalues of $\mathbf{S}$ along the diagonal. The eigenvectors can be obtained either by minimizing, or by maximizing the Brockett cost function, only the ordering along the diagonal of the diagonalized matrix $\mathbf{W}^H\mathbf{S}\mathbf{W}$ will be different (increasing/decreasing order).

NOTE: This set of codes was designed such that the Riemannian optimization scripts are separated from the cost function specific scripts. Therefore, these codes can also be easily used to optimize other smooth cost functions, simply by changing the cost function-specific parameters/scripts. The cost function specific parts are: the cost function evaluation, the gradient computation and the order of the cost function in the coefficients of $\mathbf{W}$ (details are provided in the next section).

2 Detailed Description

The architecture of this set of codes and the interaction among them is shown in Figure 1. The main testing code *main_code.m* calls the code *riemann_grad_unit_opt.m* that performs Riemannian optimization under unitary matrix constraint. Steepest Descent/Ascent (SD/SA) algorithm (see Table 1 in [2]), Conjugate Gradient (CG-PR, CG-FR) algorithms (both Fletcher-Reeves and Polak-Ribière formulas, see Table 3 in [1]) are implemented. This code further calls the codes implementing line search methods: Armijo method *geod_search_armijo.m* [2], as well as the polynomial and DFT-based approximation line search methods proposed in [1] *geod_search_poly.m, geod_search_dft.m.*

The main objective of this architecture was to separate the Riemannian optimization part from the cost function specific part. This way, the codes can be easily adapted to any other smooth cost function. The cost function specific scripts are *cf_eval.m* that computes the value of the cost function and *euclid_grad_eval.m* that computes the Euclidean gradient of the cost function at a given point on the unitary group. The cost function specific scripts/parameters are shown with blue color in Figure 1. Here, the Brockett function is taken as an example, but the changes that would be required to optimize other cost function are straight-forward. Additional small scripts are also called from the unitary optimization core, such as *unit_crit_eval.m, diag_crit_eval.m, innerprod.m, skew.m.*

As an example, the Brockett criterion is optimized (minimized or maximized, depending on the choice). One typical simulation is shown in Figure 2 in the case of maximization, and in Figure 3 in the case of minimization.

References

1. T. Abrudan, J. Eriksson, V. Koivunen, "Conjugate Gradient Algorithm for Optimization Under Unitary Matrix Constraint", *Signal Processing*, vol. 89, no. 9, Sep. 2009, pp. 1704–1714.
   PDF available online here
   BibTeX citation here

2. T. Abrudan, J. Eriksson, V. Koivunen; "Steepest Descent Algorithm for Optimization under Unitary Matrix Constraint", *IEEE Transactions on Signal Processing*, vol. 56, no. 3, Mar. 2008, pp. 1134–1147.
   PDF available online here
   BibTeX citation here

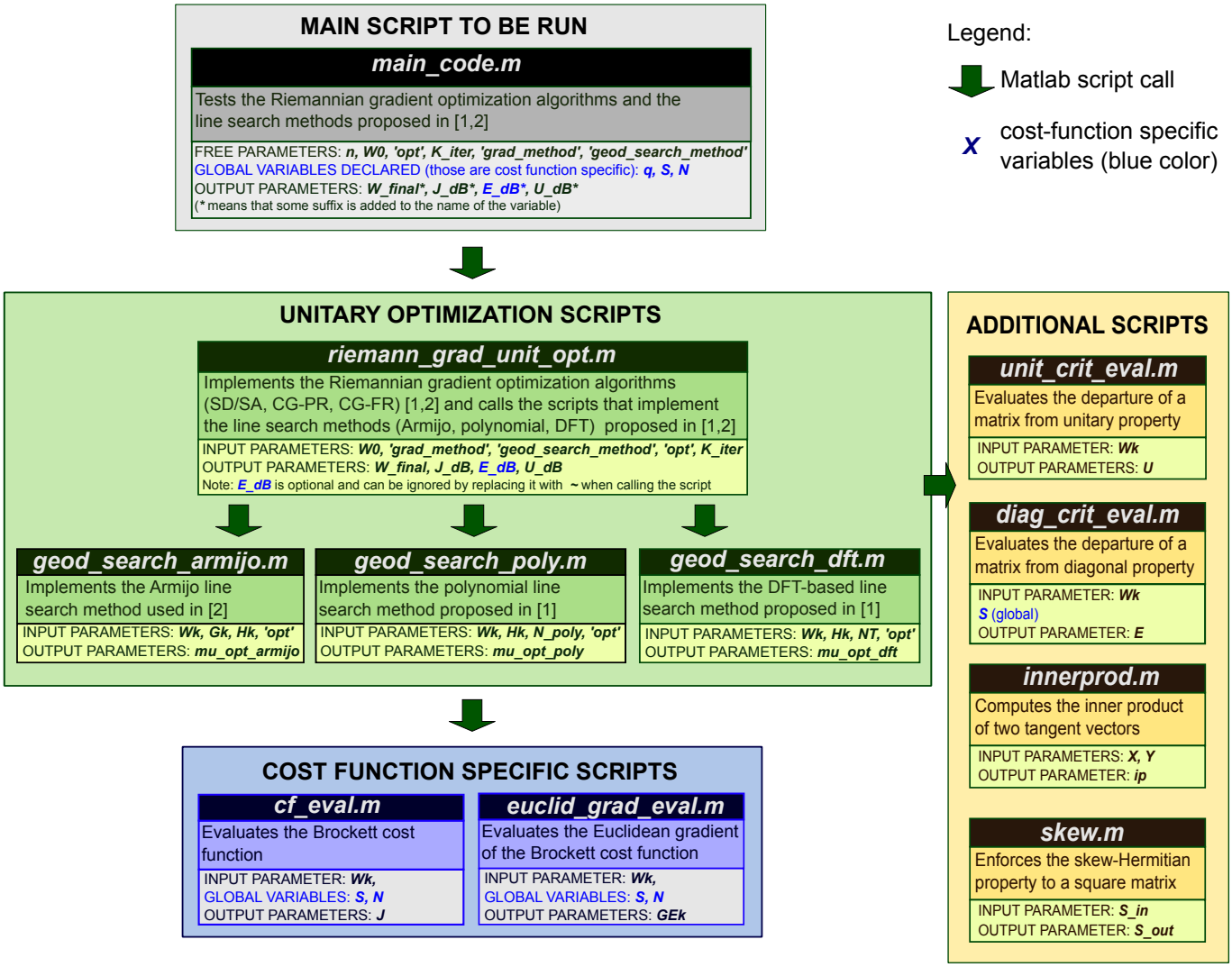

Figure 1: The interaction among scripts.

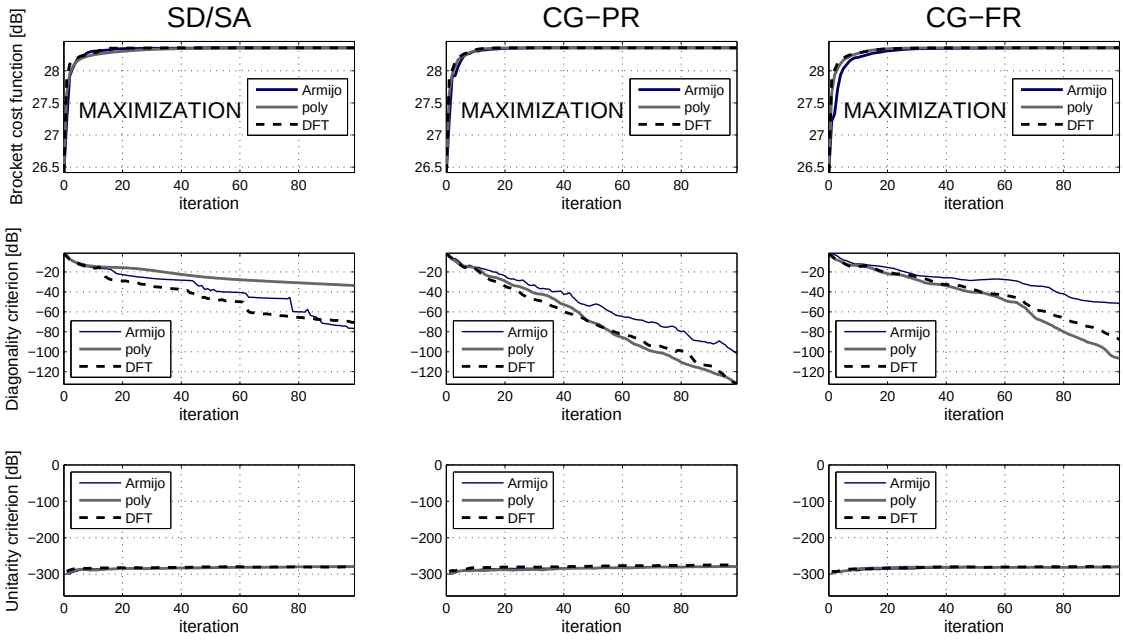

Figure 2: Example of maximizing the Brockett cost function.

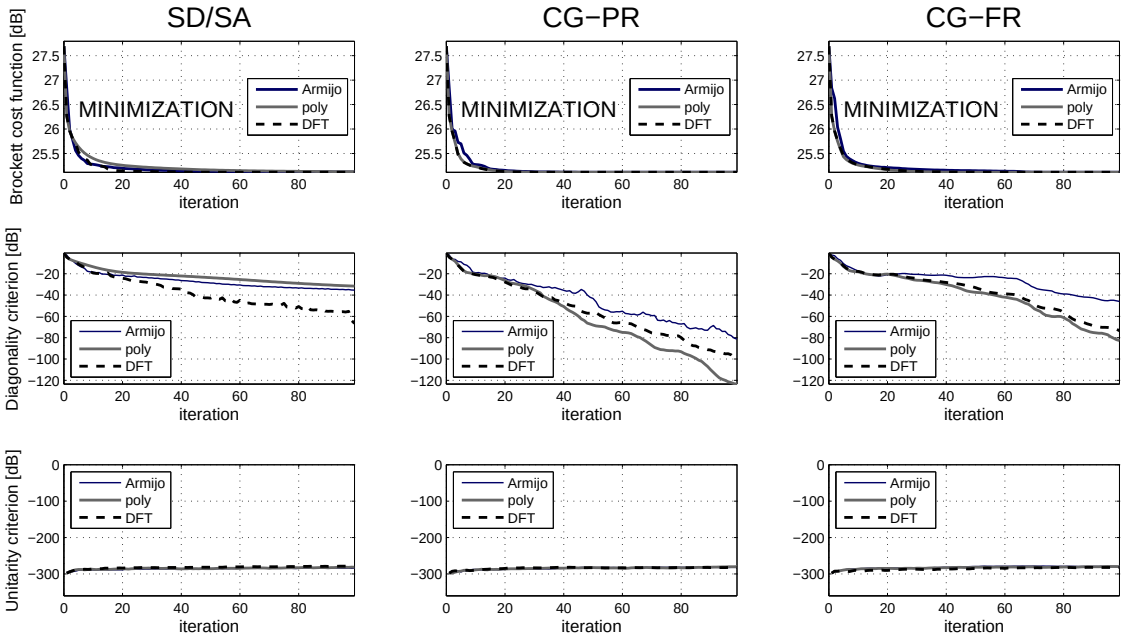

Figure 3: Example of minimizing the Brockett cost function.