# Eigopt : Software for Eigenvalue Optimization

Emre Mengi* E. Alper Yıldırım† Mustafa Kılıç‡

August 8, 2014

## 1 Problem Definition

These Matlab routines are implemented for the global optimization of a particular eigenvalue of a Hermitian matrix-valued function depending on its parameters analytically subject to box constraints. Formally, the routines are intended for the global solution of a problem of the form

$$\text{minimize}_{\omega\in\mathcal{B}}\ \lambda\left(\mathcal{A}(\omega)\right) \qquad \text{or} \qquad \text{maximize}_{\omega\in\mathcal{B}}\ \lambda\left(\mathcal{A}(\omega)\right). \tag{1}$$

Above

**(i)** $\mathcal{A}(\omega) : \mathbb{R}^d \to \mathbb{C}^{n\times n}$ is an analytic function of $\omega$ satisfying $\mathcal{A}(\omega) = \mathcal{A}(\omega)^*$ for all $\omega$;

**(ii)** $\mathcal{B} := \mathcal{B}\left(\omega_1^{(l)}, \omega_1^{(u)}, \ldots, \omega_d^{(l)}, \omega_d^{(u)}\right) := \left\{\omega \in \mathbb{R}^d : \omega_j \in \left[\omega_j^{(l)}, \omega_j^{(u)}\right] \text{ for } j = 1, \ldots, d\right\}$ is the box defining the constraints.

**(iii)** In theory, the algorithm works for

- **(a)** the optimization of an eigenvalue $\lambda(\mathcal{A}(\omega))$ of $\mathcal{A}(\omega)$ that is continuous and simple at all $\omega \in \mathcal{B}$ (this simplicity assumption on $\mathcal{B}$ holds generically for many eigenvalue functions of interest), or
- **(b)** the minimization of extremal eigenvalue functions of the form

$$\lambda(\mathcal{A}\left(\omega\right)) = \sum_{k=1}^{j} d_k \cdot \lambda_k\left(\mathcal{A}\left(\omega\right)\right),$$

where $\lambda_k(\cdot)$ denotes the $k$th kargest eigenvalue, and $d_1 \geq d_2 \geq \cdots \geq d_j \geq 0$ are real numbers, for instance minimization of $\lambda_1\left(\mathcal{A}\left(\omega\right)\right)$.

**(iv)** In practice, we observe that the algorithm often seems to work for a continuous and piece-wise analytic eigenvalue function with respect to each parameter disjointly. But be aware that in theory there is no guarantee that it would work in general.

*Department of Mathematics, Koç University, Rumelifeneri Yolu, 34450 Sarıyer-İstanbul, Turkey (emengi@ku.edu.tr). The work of this author was supported in part by the European Commision grant PIRG-GA-268355 and the TÜBİTAK (The Scientific and Technological Research Council of Turkey) carrier grant 109T660

†Department of Industrial Engineering, Koç University, Rumelifeneri Yolu, 34450 Sarıyer-İstanbul, Turkey (alperyildirim@ku.edu.tr). This author was supported in part by TÜBİTAK (Turkish Scientific and Technological Research Council) Grant 109M149 and by TÜBA-GEBİP (Turkish Academy of Sciences Young Scientists Award Program)

‡Department of Mathematics, Koç University, Rumelifeneri Yolu, 34450 Sarıyer-İstanbul, Turkey (mukilic@ku.edu.tr). The work of this author was supported partly by the European Commision grant PIRG-GA-268355.

We assume the knowledge of a constant $\gamma$, when $\lambda\left(\mathcal{A}(\omega)\right)$ is to be minimized, such that

$$\lambda_{\min}\left[\nabla^2\lambda(\omega)\right] \geq \gamma \tag{2}$$

at all $\omega \in \mathcal{B}$ where $\lambda(\omega)$ is differentiable. Such $\gamma$ can be deduced analytically sometimes, especially for the extremal eigenvalue functions [2, Section 6]. In some other cases, it can be deduced or guessed numerically.

If $\lambda\left(\mathcal{A}(\omega)\right)$ is to be maximized, we minimize $-\lambda\left(\mathcal{A}(\omega)\right)$. Thus, in this case, $\gamma$ must satisfy

$$\lambda_{\min}\left[-\nabla^2\lambda(\omega)\right] \geq \gamma \tag{3}$$

at all $\omega \in \mathcal{B}$ where $\lambda(\omega)$ is differentiable.

# 2 Algorithm

The software is an implementation of the algorithm discussed in [2]; this is a realization of the algorithm due to Breiman and Cutler [1] for eigenvalue optimization. For the minimization problem in (1) it constructs piece-wise quadratic support functions lying globally underneath $\lambda(\mathcal{A}(\omega))$. Then, it calculates a global minimizer of this support function, and refines the piece-wise quadratic support function, i.e., adds one more quadratic piece using the calculated global minimizer, the piece-wise quadratic support function is the maximum of all such quadratic pieces constructed so far. The algorithm is guaranteed to converge to a global minimizer globally [2, Section 7].

Finding the global minimizer of the piece-wise quadratic support functions is equivalent to solving a bunch of quadratic programs. The quadratic programs require the minimization of quadratic functions with (typically) negative, but constant curvature, subject to linear constraints. This is efficiently solvable when the number of parameters $d$ is small, since the solution is attained at one of the vertices of the feasible region, and also there is a special structure of the underlying polyhedra.

# 3 Usage

The Matlab routine must be called as follows:

```
>>  [f,z parsout] = eigopt(funname,bounds,parsin)
```

Here is an explanation of the parameters.

- `funname` (input, string) is the name of the function evaluating $\lambda(\mathcal{A}(\omega))$ and its gradient. Analytic expressions [2, Section 3.2] for the derivatives of $\lambda(\mathcal{A}(\omega))$ or finite difference formulas can be used for gradient computation.
- `bounds` (input, struct) contains the bounds for the box $\mathcal{B}$ with the following fields:
  - `bounds.lb` ($d \times 1$ real array) contains the lower bounds for the box constraints;
  - `bounds.ub` ($d \times 1$ real array) contains the upper bounds for the box constraints.
- `parsin` (input, struct) contains the parameters with the fields below, and the parameters to be passed to `funname`:
  - `gamma` (input, real) is the bound for the second derivatives as defined in (2) for minimization, or as defined in (3) for maximization;
  - `tol` (input, real) is the absolute tolerance for the accuracy of the computed globally minimal or maximal value of $\lambda(\mathcal{A}(\omega))$ in (1);

  - `itertol` (input, integer) is the maximal number of iterations;
  - `isprint` (input, integer) is to indicate whether to print out the details of the algorithm; does not print if isprint = 0 (default), print otherwise if isprint $\neq 0$;
  - `isplot` (input, integer) is to indicate whether to plot the underlying graph; plots if isplot $\neq 0$ and the problem is 2-dimensional, otherwise it does not plot (default);
  - `iskeyboard` (input, integer) is to indicate whether to interact with the user; does not interact if iskeyboard = 0 (default) interact otherwise if iskeyboard $\neq 0$;
  - `minmax` (input, integer) is to indicate whether a minimization or a maximization problem will be solved; minimization if minmax = 0 (default), maximization otherwise if minmax $\neq 0$.

- `f` (output, real) is the computed globally minimal or maximal value of $\lambda(\mathcal{A}(\omega))$ in (1),
- `z` (output, $d \times 1$ real array) contains the computed global minimizer or maximizer, and
- `parsout` (output, struct) contains the output parameters with the following fields:
  - `tnfevals` (output, integer) is the total number of function and gradient evaluations performed;
  - `lbound` (output, real) is a lower bound for the globally minimal or an upper bound for the globally maximal value; if it is a minimization problem $f - parsout.lbound < parsin.tol$. or if it is a maximization problem $parsout.lbound - problem.f < parsin.tol$;
  - `nvertices` (output, integer array) keeps the number of vertices at every iteration;
  - `newvertexlist` (output, integer array) keeps the number of newly added vertices at every iteration;
  - `deadvertexlist` (output, integer array) keeps the number of dead vertices at every iteration;
  - `cpulist` (output, real array) keeps the cpu times required at every iteration.

The termination occurs either **(i)** when the computed globally optimal value does not differ from the exact value by no more than parsin.tol, or **(ii)** when the number of iterations is equal to or exceeds parsin.itertol.

The user is expected to write the routine evaluating $\lambda(\mathcal{A}(\omega))$ and its gradient. In the tar file, the following routines are provided for function and gradient evaluations for some common eigenvalue optimization problems.

- `fdist_instab.m` : continuous distance to instability
- `fnum_rad.m` : numerical radius
- `H_infinity.m` : $H_\infty$ norm of a linear system
- `fdist_instab_disc.m` : discrete distance to instability
- `fdist_uncont.m` : distance to uncontrollability
- `fdist_defective.m` : distance to defectiveness
- `fdist_triple.m` : distance to the nearest matrix with a triple eigenvalue
- `minimize_max_affine.m`: minimization of the largest eigenvalue of an affine matrix function
- `minimize_max_quad.m`: minimization of the largest eigenvalue of a quadratic matrix function

The expressions for the derivatives of the eigenvalue functions are derived in [2]; see, in particular, equations (3.3), (3.6) and (3.8) in [2] for the first derivatives.

# 4 Sample Calls

Assuming `parsin.A` contains a square matrix, the call

```
>> bounds.lb = -7; bounds.ub = 7;
>> parsin.gamma = -2; parsin.tol = 10^-10;
>> [f,z,parsout] = eigopt('fdist_instab',bounds,parsin);
```

computes the distance to instability from the matrix `parsin.A` to ten digit precision. Here, it is assumed that the global minimizer of the optimization problem lies in $[-7, 7]$.

Similarly, assuming `parsin.A` contains an $n \times n$ matrix and `parsin.B` contains an $n \times m$ matrix, the call

```
>> bounds.lb = [-4; -5];  bounds.ub = [4; 5];
>> parsin.gamma = -4; parsin.tol = 10^-5;
>> [f,z,parsout] = eigopt('fdist_uncont',bounds,parsin);
```

computes the distance to uncontrollability from the linear system (`parsin.A`, `parsin.B`) to five digit precision. Here, it is assumed that the global minimizer of the optimization problem lies inside the box $[-4, 4] \times [-5, 5]$.

# 5 An Example : $H_\infty$ Norm

The $H_\infty$ norm of the transfer function for the linear system

$$x'(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

has the singular value optimization characterization

$$\sup_{s \in \mathbb{R}} \sigma_1(s) \qquad \text{where} \qquad \sigma_1(s) := \sigma_1(C(siI - A)^{-1}B + D)$$

and $\sigma_1$ denotes the largest singular value. Note that assuming $A$ is stable, the matrix function

$$H(s) := \begin{bmatrix} 0 & C(siI - A)^{-1}B + D \\ \big(C(siI - A)^{-1}B + D\big)^* & 0 \end{bmatrix}$$

is analytic everywhere. Using the expressions from [2], in particular equation (3.8), we have

$$\frac{d\sigma_1(s)}{ds} = \Im\left(u_1(s)^*\, C(siI - A)^{-2}B\, v_1(s)\right)$$

where $u_1(s), v_1(s)$ consist of a consistent pair of unit left and right singular vectors associated with $\sigma_1(s)$. For instance a (rather naive; ideally one would use `svds` rather than `svd`) Matlab implementation evaluating $\sigma_1(s)$ and its derivative is given below.

```
function [f,g] = H_infinity(s,parsin)
% calculates f = sigma_max(C*(s*i*I - A)^-1*B + D) and its derivative g
% note:
% the input matrices A,B,C,D must be passed through
% parsin.A, parsin.B, parsin.C, parsin.D.

A = parsin.A; B = parsin.B;
C = parsin.C; D = parsin.D;
[m,n]=size(A);
```

```
H = C*((s*i*eye(m)-A)^-1)*B + D;
[U,S,V] = svd(H);

f = S(1,1);
g = imag(U(:,1)'*C*((s*i*eye(m)-A)^-2)*B*V(:,1));

return
```

For instance, assuming `parsin.A`, `parsin.B`, `pars.C` and `parsin.D` contain matrices of size $n \times n$, $n \times m$, $q \times n$ and $q \times m$ where `parsin.A` is stable, the call

```
>> bounds.lb = -20; bounds.ub = 20;
>> parsin.gamma = -norm(parsin.A); parsin.tol = 10^-8; parsin.minmax = 1;
>> [f,z,tnfevals] = eigopt('H_infinity',bounds,parsin);
```

computes the $H_\infty$ norm of the associated linear system (`parsin.A`, `parsin.B`, `parsin.C`, `parsin.D`) to eight digits precision. Once again, here it is assumed that the optimal $s$ lies in $[-20, 20]$. It is also assumed that `-norm(parsin.A)` is a global lower bound for the second derivatives satisfying (3).

# References

[1] L. Breiman and A. Cutler. A deterministic algorithm for global optimization. *Math. Program.*, 58(2):179–199, 1993.

[2] E. Mengi, E.A. Yildirim, and M. Kilic. Numerical optimization of eigenvalues of Hermitian matrix functions. *SIAM J. Matrix Anal. Appl.*, 35(2):699–724, 2014.