QARPACK: Quadratic Arnoldi Package v.2. User's Guide.

Marta M. Betcke[*]

July 15, 2013

1 What is QARPACK?

QARPACK is a MATLAB Toolbox for computing eigenvalues with real part in a specified interval and the corresponding eigenvectors of a large sparse quadratic or nonlinear eigenvalue problem (QEP/NEP).

For QEPs QARPACK uses following two forms:

$$(\lambda^2 M + \lambda C + K)x = 0, \quad x \neq 0, \tag{1}$$

or

$$(-\omega^2 M + i\omega C + K)x = 0, \quad x \neq 0. \tag{2}$$

The second form can be obtained from (1) through a substitution, $\lambda = i\omega$. (1) is in general preferable when the eigenvalues λ have a dominant real part, while (2) when λ have a dominant imaginary part, hence ω have a dominant real part again.

From version v.1.9 QARPACK also contains NRA a solver for general nonlinear eigenvalue problems

$$\sum_{i=1}^{k} f_i(\lambda) A_i x = 0, \quad x \neq 0. \tag{3}$$

From version v.2.0 NRA also supports multiple eigenvalues.

[*]Department of Computer Science, University College London, Malet Place, London WC1E 6BT, UK. Email: m.betcke@gmail.com

QARPACK implements three iterative methods which directly act on the QEP/NEP avoiding prior linearization (and hence doubling of the problem size in case of QEP):

QRA: Quadratic Restarted Arnoldi (M.M. Betcke, H. Voss, 2011, [1])

QARN: Q-Arnoldi for QEP (K. Meerbergen, 2001, [2])

NRA: Nonlinear Restarted Arnoldi (M.M. Betcke, H. Voss, 2011, [1])

All methods utilise local restarts to enable computation of a large number of the eigenvalues with the real part in an interval in the interior of the spectrum in one go. QRA accepts both QEP forms, QARN accepts only form (2) and NRA accepts only form (3). Furthermore, as not all QEP are formulated respecting the dominant direction in the spectrum, it may be necessary to reformulate the QEP to obtain the desired result, as explained in Section 4. In the essence, the methods will perform best if there is a dominant direction in the eigenvalue distribution in the complex plain e.g. the eigenvalues have a dominant part (real or imaginary). See the associated publications, [2, 1], for more insight on methods in QARPACK.

2 Installation

QARPACK is hosted at

```
http://www.cs.ucl.ac.uk/staff/M.Betcke/codes/qarpack/
```

The basic functionality requires only MATLAB and should be compatible with any reasonably recent release. The external preconditioners like e.g. PARDISO or AMG have to be obtained and installed separately.

Untar the contents of the package into the preferred location

```
tar -xzf qarpack.tar.gz
```

This will create a top level directory ”qarpack” with the following content:

```
qarpack/
qarpack/qra.m
qarpack/nra.m
qarpack/qarn.m
```

```
qarpack/qoptions.m
qarpack/demo_qarpack_nlevp.m
qarpack/plot_ev_conv.m
qarpack/unique_ev.m
qarpack/nlevp2nep.m
qarpack/nep2str.m
qarpack/nep_a_times_fi.m
qarpack/README.txt
qarpack/COPYING.txt
qarpack/gpl-3.0.txt
qarpack/private/
qarpack/private/gmres_jd.m
qarpack/private/lline.m
qarpack/private/permschur.m
qarpack/private/precondition_AMG.m
qarpack/private/precondition_LU.m
qarpack/private/precondition_UMFPACK.m
qarpack/private/precondition_PARDISO.m
qarpack/doc/QARPACK_User_Guide.pdf
```

Add the full path to "qarpack" folder to your MATLAB path.

3 Interface to external preconditioners

Apart from the MATLAB LU factorisations (LU: built-in $PA = LU$, UMFPACK: $PAQ = LU$), QARPACK provides interface to two external preconditioners:

- Sparse LU factorization, PARDISO

  `http://www.pardiso-project.org`

  You will also need to download and compile PARDISO Matlab interface by Peter Carbonetto available from the same website. Check PARDISO User's Guide for detailed instructions.

- MATLAB implementation of Algebraic Multigrid by Jonathan Boyle. This code is not publicly available. If you are interested in using this AMG preconditioner please send your enquiry to David Silvester `d.silvester_at_manchester.ac.uk`. It is however our experience that LU decomposition performs superior to AMG for computation of the eigenvalues in the interior of the spectrum.

You can create an interface to any other preconditioner by implementing a function with an interface

```
precondition_PREC(mode, A, pole, prec_params, res)
```

following the included examples for LU, UMFPACK, PARDISO and AMG. The function should implement four modes:

```
"init", "compute", "apply", "finalize"
```

4 Reformulating a QEP

The methods in QARPACK internally use the ordering the eigenvalues by their increasing real part, when advancing through the spectrum. Thus, the best results will be obtained if the eigenvalues of the QEP/NEP distribute along the real axis, or at least have a dominant real part. This imposes no restriction on what the methods can do, just it may require reformulating of your original QEP/NEP in one of the ways described below. The quadratic methods load the QEP directly from the provided file. QRA accepts both forms of QEP while QARN only accepts QEP in form (2).

Transformation between QEP form (1)

$$\lambda^2 Mx + \lambda Cx + Kx = 0,$$

and QEP form (2)

$$-\omega^2 \tilde{M}x + i\omega\tilde{C}x + \tilde{K}x = 0$$

i $\Re(\lambda)$ **dominant** $\leftrightarrow$ $\Re(\omega)$ **dominant:**
(1) $\to$ (2): $\lambda = \omega, \quad \tilde{M} = M, \quad \tilde{G} = iG, \quad \tilde{K} = -K$
(2) $\to$ (1): $\omega = \lambda, \quad M = \tilde{M}, \quad G = -i\tilde{G}, \quad K = -\tilde{K}$

ii $\Im(\lambda)$ **dominant** $\leftrightarrow$ $\Re(\omega)$ **dominant:**
(1) $\to$ (2): $\lambda = i\omega, \quad \tilde{M} = M, \quad \tilde{G} = G, \quad \tilde{K} = K$
(2) $\to$ (1): $\omega = -i\lambda, \quad M = \tilde{M}, \quad G = \tilde{G}, \quad K = \tilde{K}$

Transformation between QEP of the same form i.e. either both in form (1):

$$\lambda^2 Mx + \lambda Cx + Kx = 0, \text{ with dominant } \Im(\lambda)$$

to

$$\omega^2 \tilde{M}x + \omega\tilde{C}x + \tilde{K}x = 0, \text{ with dominant } \Re(\omega),$$

or both in form (2)

$$-\lambda^2 Mx + i\lambda Cx + Kx = 0, \text{ with dominant } \Im(\lambda)$$

to

$$-\omega^2 \tilde{M}x + i\omega\tilde{C}x + \tilde{K}x = 0, \text{ with dominant } \Re(\omega)$$

iii $\Im(\lambda)$ **dominant** $\to$ $\Re(\omega)$ **dominant:**
$\lambda = i\omega, \quad \tilde{M} = M, \quad \tilde{G} = -iG, \quad \tilde{K} = -K$

5 Describing NEP

The general nonlinear eigenvalue solver NRA requires a structure describing the nonlinear eigenvalue problem (3)

nep.k: number of additive terms

nep.Ai: matrix coefficients

nep.fi: functions of eigenvalue

nep.dfi: derivative of fi

nep.pcoeff: preconditioner weights

nep.n: size of A_i

nep.type: {” = ’generic’, ’quadratic’}: type of the NEP

nep.prop: {”, ’symmetric’,’skewsymmetric’}: properties of the matrix coefficients

The preconditioner weights are used to set up the preconditioner

$$P \approx \left(\sum_{i=1}^{k} p_i A_i f_i(\sigma) \right)^{-1}. \tag{4}$$

The type of NEP determines the solver for projected problems. If NEP type is 'quadratic' the linearization will be used. In general case we use the safeguarded iteration. The properties of the coefficient matrices are used for efficient projection.

The NEP can be reformulate from imaginary to real dominant eigenvalues using the same transformation as for QEP $\lambda = i\omega$.

6 QARPACK functionality

The core of QARPACK are three iterative projection methods for computation of eigenvalues with a real part in a specified interval, NRA, its quadratic version QRA and QARN. Furthermore some auxiliary routines are provided including interfaces to external preconditioners and some visualisation methods.

6.1 QRA: Quadratic Restarted Arnoldi

QRA/NRA utilises the min-max characterisation the eigenvalues of the QEP or NEP to number eigenvalues. Once numbered the eigenvalues are processed one after another, minimising the risk of missing out eigenvalues. To be able to handle eigenvalues in the interior of the spectrum local eigenvalue numbering w.r.t. a chosen anchor eigenvalue have been introduced. The local numbering can be temporarily disturbed by spurious eigenvalues, which are eigenvalues on their way to their appropriate place in the spectrum but outside of the interval corresponding to the current search subspace. QRA/NRA has safeguards in place to recognise spurious eigenvalues and effectively drive them out of the search subspace are restore the local numbering. QRA/NRA will restart whenever the search subspace dimension exceeded a specified threshold, or convergence become to slow. At the restart the anchor is reset to one the previously computed eigenpairs, and the search continues for the eigenvalues with the real part larger than the anchor eigenvalue building a new search subspace.

```
[X, cosX, lambda, resvec, hit, lu_counter, lu_hit, time_comps, time_nonlins,
  time_lus, flag, iter_dim, la_hist, la_hist_hit] = qra(data_file, anchor,
  vanchor, up_bound, problem_type, scale, inverted, max_dim, solver, max_iter,
  prec_params, opt);

Nonlinear Arnoldi/Jacobi Davidson with local restarts for the solution
of a quadratic eingevalue problem (QEP) in a form
  -lambda^2*M*x + i*lambda*C*x + K*x = 0,   (opt.QEP_FORM = 1)
for eigenvalues (i*lambda) i,e. with dominant imaginary part or
   lambda^2*M*x + lambda*C*x + K*x = 0,     (opt.QEP_FORM = 0)
for eigenvalues lambda with dominant real part.

The projected problems are solved by linerization. The residual norm
||T(lambda)x||/||x|| < RES_TOL is used as a stopping criterium.

The local restarts use a chosen eigenpair, called an anchor,
to introduce a local numbering in the subspace relatively
to the anchor. During the iteration the local numbering can be
distorted by "spurious" eigenvalues. The spurious eigenvalues
are handled by the following deliberate iteration.
The computed eigenvalues are identified in the search subspace and
the not matching values are checked if they are "spurious" or genuine
previously missed out eigenvalues by computing their residual norm.
The first found "spurious" eigenvalue is iterated i.e. its residual
is used to expand the search subspace until the subspace is cleared
from the "spurious" eigenvalues or the previously missed eigenvalue
has converged and the local numbering is restored. In this way a direct
effort is made to accelerate the convergence of a spurious eigenvalue
to drive it away from the subspace corresponding to
(anchor, nr(last computed eigenvalue)+1) or to detect it unambiguously
as a previously missed eigenvalue and include on its right position
into the search subspace.

Restart can be triggered by:
1. exceeding the maximal subspace dimension, approached within a specified
margin, opt.MAX_DIM_MARG to avoid restarts while an eigenvalue is iterated
2. slow convergence rate of the residuals (see opt.CONV_RATE_TOL)

When self scheduleding of restarts is enabled the algorithm measures
the CPU time spent in average on an eigenvalue pair computation up to now
```

```
and registers whether the iterated eigenpair takes longer than the average
time pro eigenpair or not, by counting up or down
  if too slow, toolong_counter = toolong_counter + 1,
  else         toolong_counter = max(toolong_counter-1, 0)
If toolong_counter exeeeds allowed number opt.TOOLONG, restart is triggered.

@Input:
datafile      - name of the file with the matrices M, C (or G), K,
                 for QEP in the form:
                  -lambda^2*M*x + i*lambda*C*x + K*x = 0
                 for i*lambda - eigenvalues dominated by their imaginary part
                 (set opt.QEP_FORM = 1) or
                   lambda^2*M*x + lambda*C*x + K*x = 0
                 for lambda - eigenvalues dominated by their real part
                 (set QEP_FORM = 0)
anchor        - a known eigenvalue (an anchor) or a lower bound on the first
                wanted eigenvalue. In the latter case the first eigenpair
                will be iterated to convergence and then it will be used
                as an anchor. If a multiple eigenvalue is specified as
                an achor, the computed eigenpairs will include none of
                the eigenvectors corresponding to the (multiple) anchor
vanchor       - eigenvector corresponding to the anchor, if empty the first
                eigenpair (lambda, x) with real(lambda) >= anchor will be
                iterated to convergence and then it will be used as an anchor.
                The same measure is taken if the residual of the anchor pair
                is not small enough.
up_bound      - upper bound for wanted eigenvalues
problem_type  - type of QEP: {'gyroscopic', 'symmetric', 'quadratic'}.
                If no C or G exists, we set G = 0 and problem is relabelled
                'linear' but it is computationally treated as quadratic.
scale         - scale the eigenvalues of the QEP:
                (new eigenvalue) = scale * (old eigenvalue)
inverted      - {0, 1} do not/do use the reversal of the polynomial, which
                is equivalent to the problem obtained through substitution
                lambda <- (-1/lambda) for opt.QEP_FORM = 1,
                 M*x + i*lambda*C*x - lambda^2*K*x = 0 or
                 M*x + lambda*C*x + lambda^2*K*x = 0
                with lambda <- (1/lambda) for opt.QEP_FORM = 0
max_dim       - maximal size of the searche subspace. If exeeded a restart
                is triggered
```

```
solver        - the solver used for the subspace expansion
                 'arn': Residual Inverse Iteration (RII),
                 'iter': RII with GMRES instead of a direct solve,
                 'jd': Jacobi Davidson
max_iter      - maximal iteration number, if exeeded the algorithms will
                terminate regardless of its state
prec_params   - preconditioner parameters structure:
                 type: 'LU' (default), 'AMG', 'PARDISO', 'UMFPACK'
                For LU, UMFPACK and PARDISO:
                 reorder: reordering algorithm for A, returning p
                 such that A(p,p), e.g. @symamd, @symrcm,
                 [] no reordering (default)
                 diag_scale: 0 - none, 1 - left, 2 - right (default)
                For LU only:
                 tol: tolerance for LU preconditioner if Inf then LU
                      is used (default), otherwise an incomplete LU
                      with specified tolerance tol is used
                For AMG (code by J. Boyle, not included):
                 tol: tolerance for amg_solver (default opt.RES_TOL),
                 diag_scale: 0 - none, 1 - left, 2 - right (default)
                 maxit: maximal number of iterations of the iterative solver
                        (default opt.JD_GMRES_MAXIT)
                 amg_smoother: 'PDJ' (default), 'PGS'
                 amg_solver: @minres, @pcg, @gmres (default)
                 reuse_mg: number of times the same multi grid is reused
                           (default 0 do not reuse preconditioner)
                 restart: as in e.g. gmres, set [] for no restarts (default)
                Default the complete matlab 'LU'
opt           - further options for the solver, see qoptions.m

@Output:
X             - eigenvectors
cosX          - cosines of angles between the consecutive eigenvectors
lambda        - eigenvalues, ordered by the real part
resvec        - residual norm history
hit           - numbers of iteration at which eigenvalues converged
lu_counter    - number of (incomplete) LU factorizations
lu_hit        - numbers of iteration when the preconditioner was recomputed
time_comps    - computation time recorded after every converged eigenvalue,
                the last entry contains total time
```

```
time_nonlins  - time spent on solving of projected eigenvalue problems
                recorded after every converged eigenvalue, reset at restart
time_lus      - CPU times for computation of new preconditioners
flag          - outcome of the algorithm
                 0 - the upper bound have been reached,
                 1 - iteration exited prematurely i.e. the limit of iterations
                     has been reached. The stopping criterium is not satisfied
                     but some eigenvalues converged
                 2 - iteration exited prematurely i.e. the limit of iterations
                     has been reached. The stopping criterium is not satisfied
                     and no eigenvalue converged so far
iter_dim      - search subspace dimension at each iteration
la_hist       - history of eigenvalue convergence (it records convergence of
                multiple copies of the same eigenpair)
la_hist_hit   - number of iterations at which an eigenpair converged (possibly
                a copie of already converged value)
restart_cause - records the cause for the restart as sum of
                 1000: emergency restart (opt.MARG,
                 100 : search space dimension exeeded,
                 10  : slow convergence rate of  the residual norms,
                 1   : self scheduling triggered restart (average convergence
                       time pro eigenpair exeeded too often)
```

6.2 NRA: Nonlinear Restarted Arnoldi

NRA can be applied to general nonlinear eigenvalue problems, while QRA is an implementation limited to the quadratic case by the choice of the solver for the projected problem.

```
[X, cosX, lambda, resvec, hit, lu_counter, lu_hit, time_comps, time_nonlins,
 time_lus, flag, iter_dim, la_hist, la_hist_hit] = nra(nep, anchor,
 vanchor, up_bound, max_dim, solver, max_iter, prec_params, opt);

Nonlinear Arnoldi/Jacobi Davidson with local restarts for the solution
of a nonlinear eingevalue problem (NEP) of a form
 T(lambda) x = sum_i fi(lambda) Ai x = 0

The projected problems are solved for general problems with
safeguarded iteration and for quadratic problems with linearization.
The residual norm ||T(lambda)x||/||x|| < RES_TOL is used as a stopping criterium.
The local restarts use a chosen eigenpair, called an anchor,
to introduce a local numbering in the subspace relatively
to the anchor. During the iteration the local numbering can be
distorted by "spurious" eigenvalues. The spurious eigenvalues
are handled by the following deliberate iteration.
The computed eigenvalues are identified in the search subspace and
the not matching values are checked if they are "spurious" or genuine
previously missed out eigenvalues by computing their residual norm.
The first found "spurious" eigenvalue is iterated i.e. its residual
is used to expand the search subspace until the subspace is cleared
from the "spurious" eigenvalues or the previously missed eigenvalue
has converged and the local numbering is restored. In this way a direct
effort is made to accelerate the convergence of a spurious eigenvalue
to drive it away from the subspace corresponding to
(anchor, nr(last computed eigenvalue)+1) or to detect it unambiguously
as a previously missed eigenvalue and include on its right position
into the search subspace.

Restart can be triggered by:
1. exceeding the maximal subspace dimension, approached within a specified
margin, opt.MAX_DIM_MARG to avoid restarts while an eigenvalue is iterated
2. slow convergence rate of the residuals (see opt.CONV_RATE_TOL)
3. When self scheduleding of restarts is enabled the algorithm measures
the CPU time spent in average on an eigenvalue pair computation up to now
and registers whether the iterated eigenpair takes longer than the average
time pro eigenpair (multiplied by opt.ALPHA) or not, by counting up or down
  if too slow, toolong_counter = toolong_counter + 1,
  else         toolong_counter = max(toolong_counter-1, 0)
If toolong_counter exeeeds allowed number opt.TOOLONG, restart is triggered.

This version supports multiple eigenvalues.

@Input:
nep             - data structure containing matrix and function coefficients
                  of NEP
                    T(lambda)*x = sum_i nep.fi{i}(lambda)*nep.Ai{i}*x = 0
anchor          - a known eigenvalue (an anchor) or a lower bound on the first
                  wanted eigenvalue. In the latter case the first eigenpair
                  will be iterated to convergence and the bound will be used
```

```
                 as an anchor. If a multiple eigenvalue is specified as
                 an achor, the computed eigenpairs will include all of
                 the eigenvectors corresponding to the (multiple) anchor
vanchor        - eigenvector corresponding to the anchor, if empty the first
                 eigenpair (lambda, x) with real(lambda) >= anchor will be
                 iterated to convergence and the bound will be used as an anchor.
                 The same measure is taken if the residual of the anchor pair
                 is not small enough.
up_bound       - upper bound for wanted eigenvalues
max_dim        - maximal size of the searche subspace. If exeeded a restart
                 is triggered
solver         - the solver used for the subspace expansion
                    'arn': Residual Inverse Iteration (RII),
                    'iter': RII with GMRES instead of a direct solve,
                    'jd': Jacobi Davidson
max_iter       - maximal iteration number, if exeeded the algorithms will
                  terminate regardless of its state
prec_params     - preconditioner parameters structure:
                    type: 'LU' (default), 'AMG', 'PARDISO', 'UMFPACK'
                  For LU, UMFPACK and PARDISO:
                    reorder: reordering algorithm for A, returning p
                                    such that A(p,p), e.g. @symamd, @symrcm,
                                    [] no reordering (default)
                    diag_scale: 0 - none, 1 - left, 2 - right (default)
                  For LU only:
                    tol: tolerance for LU preconditio ner if Inf then LU
                          is used (default), otherwise an incomplete LU
                          with specified tolerance tol is used
                  For AMG (code by J. Boyle, not included):
                    tol: tolerance for amg_solver (default opt.RES_TOL),
                    diag_scale: 0 - none, 1 - left, 2 - right (default)
                    maxit: maximal number of iterations of the iterative solver
                                (default opt.JD_GMRES_MAXIT)
                    amg_smoother: 'PDJ' (default), 'PGS'
                    amg_solver: @minres, @pcg, @gmres (default)
                    reuse_mg: number of times the same multi grid is reused
                                      (default 0 do not reuse preconditioner)
                    restart: as in e.g. gmres, set [] for no restarts (default)
                  Default the complete matlab 'LU'
opt             - further options for the solver, see qoptions.m
```

```
@Output:
X               - eigenvectors
cosX            - cosines of angles between the consecutive eigenvectors
lambda          - eigenvalues, ordered by the real part
resvec          - residual norm history
hit             - numbers of iteration at which eigenvalues converged
lu_counter      - number of (incomplete) LU factorizations
lu_hit          - numbers of iteration when the preconditioner was recomputed
time_comps      - computation time recorded after every converged eigenvalue,
                  the last entry contains total time
time_nonlins    - time spent on solving of projected eigenvalue problems
                  recorded after every converged eigenvalue, reset at restart
time_lus        - CPU times for computation of new preconditioners
flag            - outcome of the algorithm
                      0 - the upper bound have been reached,
                      1 - iteration exited prematurely i.e. the limit of iterations
                          has been reached. The stopping criterium is not satisfied
                          but some eigenvalues converged
                      2 - iteration exited prematurely i.e. the limit of iterations
                          has been reached. The stopping criterium is not satisfied
                          and no eigenvalue converged so far
iter_dim      - search subspace dimension at each iteration
la_hist       - history of eigenvalue convergence (it records convergence of
                multiple copies of the same eigenpair)
la_hist_hit   - number of iterations at which an eigenpair converged (possibly
                a copie of already converged value)
restart_cause - records the cause for the restart as sum of
                  1000: emergency restart (opt.MARG,
                  100 : search space dimension exeeded,
                  10  : slow convergence rate of  the residual norms,
                  1   : self scheduling triggered restart (average convergence
                        time pro eigenpair exeeded too often)
```

6.3 QARN: Quadratic Arnoldi

QARN utilises the partial Schur form of the linearization of the QEP to lock the converged eigenvalues thereby preventing repeated convergence of the same eigenvalues. It does so without doubling the size of the problem, since all the computations are directly executed on the original QEP matrices.

Traversing of the complex plane is achieved by choosing a new pole with a larger real part then the current one, which lets the algorithm systematically work through the interval containing the real part of the wanted eigenvalues. QARN will restart whenever, the search subspace becomes to large. If the restart subspace is not large enough to accommodate all the wanted eigenvalues, some of the locked eigenvalues will have to be purged. To this end the partial Schur form is rearranged, locking the eigenvalues closest to the current pole, and purging the eigenvalues farther away. This systematic motion of the pole in the interval, makes the algorithm progress through the spectrum along the real axis. However, there is no guarantee that the eigenvalue will not converge multiple times or that it will not be missed out if the new pole has been chosen to far from the current one.

QARN is limited to quadratic eigenvalue problems.

```
[X, lambda, resvec, hit, lu_counter, poles, hit_poles, time_comps,
  time_nonlins, time_lus, flag] = qarn(data_file, sigma, start_vector,
  stop_crit, problem_type, scale, inverted, max_subspace, restart_sub,
  solver, max_iter, prec_params, opt, noisy);

Solves the quadratic eigenvalue problem (QEP):
    Q(lambda)x = 0 with Q(lambda) = -lambda^2*M + i*lambda*C + K.
using Q-Arnoldi method (Meerbergen, 2001) for nr_ev_wanted eigenvalues
nearest to the pole sigma or eigenvalues smaller than a given upper bound.
This QEP form is particularly suited for problems with eigenvalues
with dominant imaginary part and can be obtrained from a standard
    T(omega) = omega^2*M + omega*C + K
through a substitution: omega <- i*lambda.
Notice, that after the reformulation the eigenvalues of Q(lambda)
have a dominant real part.

Local restarts are implemented to enable the algorithm to compute
all wanted eigenvalues with the real part in an interval
in the spectrum in one go.

The algorithm uses partial Schur form of the symmetric linearization
to lock and purge eigenvalues. If the dimension of the restart subspace
is not large enough to accomodate all the wanted eigenvalues, some of
the computed eigenvalues will have to be purged from the local Schur
form at the restart.
```

```
Every opt.POLE_CHANGE iterations (see qoptions.m for more details),
the algorithm will select a new pole with a real part larger than
this of the current pole. Thus the algorithm works its way from
the left to the right along the real axis. Therefore, if a large
number of eigenvalues is required, the initial pole should be chosen
close to the left end of the interval containing the wanted eigenvalues.

Whenever, a new pole is computed the local Schur form is reordered,
so that the eigenvalues closest to the pole are in the left upper corner.
The eigenvalues closest to the current pole (see opt.LOCK_TYPE for
definitions of closeness) are kept in the subspace after restart.

It makes sense to let the restarts coincide with the pole change
and the preconditioner change i.e. to choose
     k*opt.POLE_CHANGE = max_subspace = j*opt.PREC_CHANGE,
with k,j integer multiples.

As with iterative algorithms in general, eigenvalues may be overlooked.

@Input:
data_file          - name of file with the matrices K, G (or C), M
                     of the QEP in form:
                     -lambda^2*M*x + i*lambda*C*x + K*x = 0, or
                     -lambda^2*M*x + i*lambda*G*x + K*x = 0
sigma              - initial pole value (pick on the lower end of
                     the real part of the spectrum)
start_vector       - initial vector, if empty a random vector is used
stop_crit          - number of wanted eigenvalues or an upper bound
                     on the real part of the eigenvalues as specified
                     by opt.STOP_COND, {0: number, 1: upper bound}
problem_type       - structure of the QEP: {'quadratic', 'gyroscopic',
                     'symmetric'}. The structure in only enforced
                     at Q(alpha) times vector product. This is
                     not rigorous, and can slow down convergence.
                     In the latter case disregard the structure
                     and revert to the general unstructured problem
                     choosing 'quadratic'
                     If no C or G exists, we set C = 0 and the problem
                     is relabeled 'linear' but still computationally
                     treated as quadratic, see eigs.m for linear solver
```

```
scale             - scale the eigenvalues:
                     (new eigenvalue) = scale * (old eigenvalue)
                    i.e. solve.
                    Q(lambda) = -lambda^2*M + i*lambda*scale*C + scale^2*K
                    If scale is empty, the quotient of 1-norms
                    of K and M is used
inverted          - {0, 1} do not/do compute with the inverted problem,
                    obtained through a substitution: lambda<-(-1/lambda),
                    resulting in:
                     M*x + i*lambda*C*x - lambda^2*K*x = 0.
                    The invertion is carried out after scaling.
max_subspace      - maximal subspace dimension, if exeeded the restart
                    is triggered
restart_sub       - size of the subspace after restart. The search subspace
                    at restart has to contain restart_sub vectors:
                     restart_sub = opt.NOT_CONV_VEC_NR (not converged)
                                  + CONV_VEC_NR (locked).
                    If restart_sub is large enough to accomodate all
                    the wanted eigenvalues, no computed eigenvalues
                    will be purged at restart. Otherwise,
                    CONV_VEC_NR <= restart_sub locked vectors in addition
                    to opt.NOT_CONV_VEC_NR not converge vectors closest
                    to the pole will be kept in at restart and the rest
                    will be purged
solver            - the solver used for the subspace expansion
                    {'arn' (LU), 'iter' (GMRES),
                     'jd' (Jacobi Davidson with GMRES)}
max_iter          - maximal iteration number, if exeeded the algorithm
                    will terminate regardless of its state
prec_params       - For 'iter' and 'jd': preconditioner, leave empty for
                    no preconditioning
                    For 'arn': subspace expansion operator, leave empty
                    for the default Matlab LU direct solver.
                    Preconditioner parameters structure:
                     type: 'LU' (default), 'AMG', 'PARDISO' , 'UMFPACK'
                    For LU, UMFPACK and PARDISO:
                     reorder: reordering algorithm for A, returning p
                     such that A(p,p), e.g. @symamd, @symrcm,
                     [] no reordering (default)
                     diag_scale: 0 - none, 1 - left, 2 - right (default)
                    For LU only:
                     tol: tolerance for LU preconditioner if Inf then LU
                          is used (default), otherwise an incomplete LU
                          with specified tolerance tol is used
                    For AMG (code by J. Boyle, not included):
                     tol: tolerance for amg_solver (default opt.RES_TOL),
                     diag_scale: 0 - none, 1 - left, 2 - right (default)
                     maxit: maximal number of iterations of the iterative solver
                            (default opt.JD_GMRES_MAXIT)
                     amg_smoother: 'PDJ' (default), 'PGS'
                     amg_solver: @minres, @pcg, @gmres (default)
                     reuse_mg: number of times the same multi grid is reused
                               (default 0 do not reuse preconditioner)
                     restart: as in e.g. gmres, set [] for no restarts (default)
                    Default the complete matlab 'LU'
opt               - further options for the solver, see also qoptions.m
noisy             - switch on the debbuging output. Set noisy
                    {0 .. nr_ev_wanted eigenvalue},
                    to debug from noisy-th eigenvalue

@Output:
X                 - eigenvectors
lamda             - eigenvalues, in the order of convergence
resvec            - residual norm history. At each iteration it contains
                    the residual of either if any vector convereged,
                    the last converged Schur vector, or the of first
                    not converged Schur vector
hit               - for each eigenvalue it records the iteration at which
                    the eigenvalue converged
lu_counter        - number of times the preconditioner was recomputed
                    (e.g. LU factorizations)
poles             - poles used throughout the computation. The method
                    for choosing a new pole is specified by opt.POLE_TYPE
hit_poles         - records the iteration number at which the pole has
                    changed
time_comps        - total computation time recorded for every converged
                    eigenvalue
time_nonlins      - computational time taken by Schur factorization and
                    reordering, recorded for every converged eigenvalue
time_lus          - time for recomputing the preconditioner, recorded at
```

```
                   every preconditioner change
flag             -  0 - the upper bound has been reached or the number
                        of wanted eigenvalues have been computed
                    1 - iteration exited prematurely i.e. the limit of
                        iterations has been reached. The stopping criterium
                        is not satisfied but some eigenvalues converged
                    2 - iteration exited prematurely i.e. the limit of
                        iterations has been reached. The stopping criterium
                        is not satisfied and no eigenvalue converged so far
```

6.4 Default Solver Options

Routine `qoptions.m` will generate a default set of parameters for each of the solvers.

```
opt = qoptions(solver, varargin)

@Input:
solver:   choose the solver from {'nra','qarn'} for which
          the parameters are needed. Each of the solvers requires
          different subset of the parameters
varargin: the default values of the parameters can be overwritten
           qoptions('nra', 'res_tol', 1e-8)
          overwrites the default opt.res_tol value, and sets
           opt.res_tol = 1e-8
@Output:
opt: structure containing the QEP solver parameters

@Default solver parameters:

Common parameters:
Tolerance threshold for reorthogonalization
 ORTH_TOL = 1e-1;
Residual norm tolerance
 RES_TOL = 1e-4;
Relative threshold for the elimination of the imaginary part
due to the roundoff error
 IMAG_TOL = 1e4*eps;
Minimal relative gap between two (not multiple) eigenvalues.
It is needed for finding spurious ev's.
```

```
 RGAP_TOL = 1e-5;
Maximal number of iterations of GMRES used by 'jd', 'iter' (NRA)
 JD_GMRES_MAXIT = 15;
No of GMRES iterations before restart, used with 'jd' (QARN), 'iter'
 GMRES_RESTART = 30;


QRA/NRA parameters:
Tolerance for the ratio of the last two residual norms. If it is
not met a restart will be triggered.
  CONV_RATE_TOL = 0.5;

Number of eigenvectors (corresponding to the eigenvalues closest
to the anchor) which will be included into the search subspace V
after the restart.
  LOCKED_VEC_NR = 0;

The algorithm will try to restart whenever the subspace size
reached max_dim - MAX_DIM_MARG to avoid restarts while
an eigenpair converges. Then the restart is triggered immediately
after an eigenvalue converged or after MAX_DIM_MARG iteration
in any case.
  MAX_DIM_MARG = 5;

Initial tolerance for the residual in JD algorithm. It will
decrease during the iteration as: JD_TOL_FACT*JD_TOL
  JD_INIT_TOL = 0.5;
  JD_TOL_FACT = 0.75;

Factor which modifies the RES_TOL when the relative distance
of two computed eigenvalues is close to RGAP_TOL
  MULT_EV_RES_FACT = 50;

Parameters for self scheduling of restarts:
How often the average time pro eigenvalue in the entire cycle
(multiplied by factor ALPHA) can be exceeded by the CPU time
necessary for the current eigenvalue.
Set TOOLONG large to disable self scheduling of restarts.
  TOOLONG = 1e4;
  ALPHA = 1;
```

```
Choose solver for projected problem, linearization type:
{'L1', 'L2'}
  PLIN = 'L2';

Choose the form of the QEP: {0,1}
 0: Q(sigma) = sigma^2*M+sigma*G+K,
    for eigenvalues with dominant real part
 1: Q(sigma) = -sigma^2*M+i*sigma*G+K,
    for eigenvalues with dominant imaginary part
  QEP_FORM = 1;

Choose if the projected QEP is solved by QR or QZ algorithm
{'QR', 'QZ'}
  PSOLVER = 'QZ';

NRA only:
Choose if the safeguarded iteration solves the EP: Tm(lambda) y = 0
or the GEP: Tm(lambda) y = mu Tm'(lambda) y = 0. If set the GEP will
be solved.
  SG_DERIV = 1;



QARN parameters:
Choose stopping criterion from
 0: eigenvalue number,
 1: upper bound on the real part of the eigenvalue
  STOP_COND = 1;

Tolerance for the iterative solver
  ITER_TOL = 1e-2;
Maximal number of consecutive failures of the iterative
solver used by 'jd'. Every time the solver fails new pole
will be set to the current iterate, S(q+1,q+1). After
MAX_FAIL_ITER-1 consecutive failures the tolerance
ITER_TOL will be reduced to sqrt(ITER_TOL). If this does
not help either the algorithm will terminate with an error
message.
  MAX_FAIL_ITER = 3;
```

```
Keep the pole for this many iterations in the initial
phase. It has to be larger then 1. If no pole change wanted,
set to max_iter
  INIT_POLE_KEEP = 40;
Relative distance wrt the largest distance to the pole
in the search subspace. It is needed for POLE_TYPE = 'far_pole'
  POLE_DIST = 1.2;
For direct solve this is how frequently the LU of Q(sigma)
is recomputed. For iterative solve this refers to
the preconditioner. We use LU of real(K - sigma^2M)
in both cases. For real K, M it is real anyway since we use
real poles.
Req: PREC_CHANGE >= POLE_CHANGE
  PREC_CHANGE = 80;
How frequently the pole is adjusted for both direct
and iterative solves.
  POLE_CHANGE = 20;

In case no of wanted ev's exceeds the restart subspace size some
eigenvalues will be purged. NOT_CONV_VEC_NR specifies how many of
the not converged vectors are kept after the pole has been changed.
Hence, restart_sub-NOT_CONV_VEC_NR converged vectors are kept
  NOT_CONV_VEC_NR = 1;

Strategy for the change of the pole
{'far_pole', 'next_ev', 'none'}
In each case the new pole has to be larger then the current one.
The poles are real. If the choice of pole is not possible some
fall-back values are used dependent on the chosen strategy.
 'far_pole': the pole is chosen between the not converged Schur
             values, at the distance at least POLE_DIST*(maximal
             distance among the locked vectors to the current pole)
 'next_ev' : the pole is chosen between the last converged value
             and the first not converged value.
 'none'    : do not change the pole
  POLE_TYPE = 'next_ev';

Choose type of locking:
 'sigma':    works through the Schur values closest to the pole,
             locking the converged pairs. It stops when it
             encounters the first not converged pair. In general
             the absolute distance to the pole is used however,
             if in addition POLE_TYPE = 'next_ev' it works through
             the Schur pairs from the closest to the pole
             in the increasing real part direction first, followed
             by the decreasing real part direction until
             it encounters an unconverged pair.
 'residual': locks all newly converged Schur vectors. The systematic
             moving through the spectrum is not forced.
  LOCK_TYPE = 'residual';

For JD: Choose Q projector type {'Z*R', 'P*R'}. The first one
is an oblique and less stable projector. The second is an orthogonal
projector but it requires an extra reorthogonalization of the residual.
  Q_TYPE = 'Z*R';

For JD: Choose from {'omega', sigma}.
 'omega': after initial INIT_POLE_KEEP iterations sigma
          in the Jacobi Davidson correction equation will be set
          to the current approximation.
 'sigma': sigma in the Jacobi Davidson correction equation
          will be chosen as the pole.
  SIGMA_TYPE = 'omega';
```

6.5 Filtering out multiple copies of an eigenvalue

QARN can find multiple copies of the same eigenvalue. A routine `unique_en.m` is provided to identify eigenvalues equal up to a given tolerance. See also `demo_qarpack_nlevp.m` for examples how to use the function.

```
[ulambda, iulambda, ifirst_conv] = unique_ev(lambda, TOL)
Returns a vector of unique eigenvalues extracted from "lambda".
The uniqueness is understood up to the specified "TOL".
Two eigenvalues lambda1 and lambda2 are considered equal if
 abs(real(lambda2)-real(lambda1))/abs(lambda1) < TOL and
 abs(imag(lambda2)-imag(lambda1))/abs(lambda1) < TOL

The vector of unique complex values "ulambda" is sorted by
the real part first and within identical real part by
the imaginary part.

"iulambda" gives the indices of the eigenvalues "lambda"
in "ulambda" i.e. it holds: ulambda(iulambda) = lambda,
up to the given tolerance TOL.

Assuming that lambda is entered in their convergence order,
"ifirst_conv" contains the indices of the first converged
copy of each of the unique eigenvalues in "lambda" i.e.
lambda(ifirst_conv) returns the first converged copies in lambda.
```

6.6 Visualisation

For the visualisation of the performance of the methods the routine `plot_ev_conv.m` is provided. See `demo_qarpack_nlevp.m` for examples of how to use this function.

```
plot_ev_conv(experiment, fig, REAL_FLAG)
Plot eigenvalue and residual norm convergence history
from results stored in the "experiment" file, into figures
with numbers in "fig" (1x4 array).
"REAL_FLAG" specifies which part of the spectrum
is dominant, real (1) or imaginary (0).

The file "experiment" has to contain the following outputs
from one or both the methods with names precisely as below:
QRA:
lambda, la_hist, la_hist_hit, anchor_lambda, lu_hit, resvec,
hit, restart_cause, optQRA
NRA:
lambda_n, la_hist_n, la_hist_hit_n, anchor_lambda_n, lu_hit_n, resvec_n,
hit_n, restart_cause_n, optNRA
QARN:
lambda_q, poles_q, hit_poles_q, resvec_q, hit_q, optQARN
```

7 Acknowledgements and Credits

Almost all of the code in QARPACK was written by Marta M. Betcke with the exception of

- `permschur.m` Schur form reordering, based on an M-File `swapschur.m`, which origin could not be determined

QARPACK is released under the GNU Public License, as follows:

If QARPACK helped you to generate results, please cite: [3, 1]

References

[1] M M Betcke and H Voss. Nonlinear Restarted Arnoldi: Interior Eigenvalue Computation. Institute of Numerical Simulation, Technical Report 157, 2011, `https://www.mat.tu-harburg.de/ins/forschung/rep/rep157.pdf`

[2] K Meerbergen. Locking and Restarting Quadratic Eigenvalue Solvers. SIAM Journal on Scientific Computing. 22 (5) pp. 1814-1839, 2001.

[3] M M Betcke. QARPACK: Quadratic Arnoldi Package. User's Guide. UCL, 2011, `http://www.cs.ucl.ac.uk/staff/M.Betcke/codes/qarpack/QARPACK_User_Guide.pdf`