leigqNEWTON_classify_multiplicity

Diagnostic grouping of (nearly) identical left eigenvalues from repeated Newton "hits".

This page documents: leigqNEWTON_classify_multiplicity

The main solver `leigqNEWTON` may return multiple accepted eigenpairs ("hits"). Depending on the sampling/restart strategy, many hits can be near-duplicates of the same isolated left eigenvalue. This helper groups such hits and provides simple diagnostics.

Notes - Quaternion left-eigenvalue multiplicities are not uniquely defined in the literature. This routine is intended as a practical diagnostic helper, not a definitive theory tool. - The main paper focuses on the number of **distinct isolated** left eigenvalues.

Convert to Live Script: File → Save As… → Live Script (*.mlx)

Setup (requirements + reproducibility)

```
hasQuat = true;
try
    quaternion(0,0,0,0);
catch
    hasQuat = false;
end
if ~hasQuat
    error('This doc page requires MATLAB built-in quaternion (Aerospace Toolbox).');
end

root = fileparts(which('leigqNEWTON'));
if isempty(root)
    error('leigqNEWTON not found on the MATLAB path. Add the toolbox root to path
first.');
end

rng(1);  % reproducible demo
```

Compute a small set of solver hits (fast demo)

We intentionally cap the number of trials to keep this page fast and predictable.

```
n = 3;
A = quaternion(randn(n), randn(n), randn(n), randn(n));

NumHitsDemo = 8;       % requested number of accepted hits
TrialsCap   = 40;      % hard cap (also disables AutoExtend)

[lambda, V, res, info] = leigqNEWTON(A, NumHitsDemo, ...
    'Trials', TrialsCap, 'MaxTrials', TrialsCap, 'AutoExtend', false, ...
    'InfoLevel','summary', 'Verbose', 0);
```

```
fprintf('Solver returned K=%d hit(s) (requested %d), TrialsCap=%d.\n', ...
    numel(lambda), NumHitsDemo, TrialsCap);
```

```
Solver returned K=3 hit(s) (requested 8), TrialsCap=40.
```

```
% Show the raw hits (often includes near-duplicates in practice)
if ~isempty(lambda)
    disp('Raw lambda hits (first few):');
    disp(lambda(1:min(end,5)));
end
```

```
Raw lambda hits (first few):
     -1.8903 -   1.841i +  1.5793j - 0.20153k
       1.529 - 0.93632i +  1.5783j +    1.37k
       1.529 - 0.93632i +  1.5783j +    1.37k
```

(Optional) inject near-duplicate hits (to guarantee grouping in the demo)

Depending on random seeds and the matrix, the solver may already return duplicates. To make the grouping behavior visible in a deterministic way, we append 2 extra hits that are extremely close to the first one (within default LambdaTol=1e-10).

```
if ~isempty(lambda)
    lamNoise = quaternion(1e-12,0,0,0);  % smaller than default LambdaTol
    lambdaHits = [lambda(:); lambda(1)+lamNoise; lambda(1)];
    if ~isempty(V)
        VHits = [V, V(:,1), V(:,1)];
    else
        VHits = [];
    end
else
    lambdaHits = lambda(:);
    VHits = V;
end

fprintf('Classifying %d hit(s) (including injected near-duplicates, if any).\n',
numel(lambdaHits));
```

```
Classifying 5 hit(s) (including injected near-duplicates, if any).
```

Group hits and estimate geometric multiplicity (complex embedding)

Basic usage:

```
R = leigqNEWTON_classify_multiplicity(A, lambdaHits, VHits);

% The output struct contains:
%   R.lambdaUnique  : unique representatives (one per group)
%   R.hitCount      : number of hits per group
%   R.groups        : index sets of hits per group
```

```
%   R.geomMultC     : estimated geometric multiplicity in complex embedding (chi)
%   R.geomMultH     : estimated geometric multiplicity in real embedding (optional)
%
fprintf('Groups found: %d\n', numel(R.lambdaUnique));
```

```
Groups found: 2
```

```
for g = 1:numel(R.lambdaUnique)
    fprintf('#%02d  hits=%d  geomMultC=%g   lambda = ', g, R.hitCount(g),
R.geomMultC(g));
    disp(R.lambdaUnique(g));
end
```

```
#01  hits=3  geomMultC=2   lambda =
     -1.8903 -   1.841i +  1.5793j - 0.20153k
#02  hits=2  geomMultC=2   lambda =
       1.529 - 0.93632i +  1.5783j +    1.37k
```

Tuning knobs (when results look "over" or "under" grouped)

`LambdaTol` controls which hits are considered duplicates (default 1e-10). `VectorTol` controls optional grouping of eigenvectors by direction (default 1e-8). `NullTol` / `NullTolFactor` control the nullity threshold used for multiplicity estimates.

Example: more aggressive grouping of eigenvalues: R2 = leigqNEWTON_classify_multiplicity(A, lambdaHits, VHits, 'LambdaTol', 1e-8);

See also

leigqNEWTON, leigqNEWTON_refine_batch, checkNEWTON