Abstract

This is the Users’ Guide for NLEVP: a collection of nonlinear eigenvalue problems provided in the form of a MATLAB toolbox. A separate paper describes the collection and its organization.

1 Introduction

This document describes how to install and use the NLEVP MATLAB toolbox, which provides a collection of nonlinear eigenvalue problems.

For details of the design and organization of the collection, and the problems it contains, see [1].

This document describes 3.0 of the toolbox. The collection will grow and contributions are welcome (see Section 6).

2 Installation and Usage

The collection is available from
http://www.mims.manchester.ac.uk/research/numerical-analysis/nlevp.html

It is provided as both a zip file and a tar file. To install the toolbox create the directory nlevp in a suitable location and make it the current directory. Download nlevp.zip or nlevp.tar into this directory. Then use appropriate “unzip” software (making sure to preserve the directory structure) or type tar xvf nlevp.tar. This creates the subdirectory private. Put the nlevp directory on the MATLAB path, which can be done using the addpath command (ideally in startup.m). If you are using GNU Octave then you must also put the nlevp/private directory on the path.

To try the toolbox, run the demonstration script by typing nlevp_example at the MATLAB command prompt. Then execute the following commands:

help nlevp
nlevp query problems
3 Release History

3.1 First Release, 1.0

The first release of the toolbox was version 1.0, dated 4-Apr-2008, and contained 26 problems.

3.2 Second Release, 2.0

The second release, version 2.0, was dated 15-Nov-2010, contained 46 problems, and had the following changes:

- Problem string has been renamed spring. spring has been generalized to include more parameters but is backward compatible with string. Invoking nlevp(‘string’) still works: it invokes nlevp(‘spring’) and produces a warning message.
- The matrices generated by problems acoustic_wave_1d and acoustic_wave_2d have been modified in order to more closely match the formulation in the paper from which this problem is taken. The eigenvalues now lie in the upper half-plane instead of the left half-plane.
- New problems are: fiber, foundation, genhyper2, Hadeler, intersection, metal strip, pdde_stability, plasma_drift, omnicam1, omnicam2, qep1, qep2, qep3, qep4, railtrack2, relative_pose_5pt, relative_pose_6pt, shaft, speaker_box, surveillance.
- New functionality: nlevp(‘eval’,...) and [coeffs,fun] = nlevp(‘name’,...).
- Automatic testing of problem properties via nlevp_test.
- Cosmetic changes have been made to some of the functions.
- Citations to the sources of the problems have been updated, where necessary.

3.3 Third Release, 3.0

The third release, version 3.0, is dated 22-Dec-2011, contains 52 problems, and has the following changes:

- New problems gen_tantipal2, gen_tpal2, mirror, planar_waveguide, qep5, time_delay.
- For scalable problems the first dimension parameter now specifies the size, \( n \), of the coefficient matrices (or an approximation to it). Previously, \( n \) was a function of this parameter in some cases. It is now possible to generate all scalable problems of a given size (or approximately that size). A warning message, with identifier NLEVP:truescale, is printed when the affected problems are called.
- Scalable problems now return coefficient matrices in the MATLAB sparse format when the coefficient matrices are sparse.
• A new problem property `random` has been introduced to specify problems that use random numbers in their construction. Such problems include an optional input argument that is used to seed the random number generator, which is useful for generating a repeatable sequence of problem instances. If that optional input argument is not provided then the same (fixed) problem is generated each time, while an argument 'noseed' ensures that the random number generator is not seeded. A random number seed argument has been added to `gen_hyper` and may result in different matrices being generated than with previous versions of NLEVP. All problems with the `random` property can use either the old or new `(rng)` MATLAB syntax for seeding the random number generator, as chosen through an input argument.

• `dirac` has been vectorized. The coefficients may differ at the level of rounding error from those produced by the previous, unvectorized code.

• The first two input arguments of `spring_dashpot` have been interchanged, so that the first is the dimension.

• A bug in the computation of the derivatives of the `gun` problem has been corrected.

• This release is compatible with GNU Octave [2], as far as possible. It has been tested with Octave 3.2.4 under Windows and Octave 3.4.3 under Linux. Since Octave does not have a function `poyeig` the function `nlevp_example` will not run.

4 The MATLAB Function `nlevp`

The toolbox has just one main user-callable function, `nlevp`, which is as follows.

```matlab
function varargout = nlevp(name,varargin)

%NLEVP  Collection of nonlinear eigenvalue problems.
% [COEFFS,FUN,OUT3,OUT4,...] = NLEVP(NAME,ARG1,ARG2,...)
% generates the matrices and functions defining the problem specified by
% NAME (a case insensitive string).
% ARG1, ARG2,... are problem-specific input arguments.
% All problems are of the form
% T(lambda)*x = 0
% where
% T(lambda)= f0(lambda)*A0 + f1(lambda)*A1 + ... + fk(lambda)*Ak.
% The matrices A0, A1, ..., Ak are returned in a cell array:
% COEFFS = {A0,...,Ak}.
% FUN is a function handle that can be used to evaluate the functions
% f1(lambda),...,fk(lambda). For a scalar lambda,
% F = FUN(lambda) returns a row vector containing
% F = [f1(lambda), f2(lambda), ..., fk(lambda)].
% If lambda is a column vector, FUN(lambda) returns a row per element in
% lambda.
% [F,FP] = FUN(lambda) also returns the derivatives
% FP = [f1'(lambda), f2'(lambda), ..., fk'(lambda)].
% [F,FP,FPP,FPPP,...] = FUN(lambda) also returns higher derivatives.
% OUT3, OUT4, ... are additional problem-specific output arguments.
% See the list below for the available problems.
% PROBLEMS = NLEVP('query','problems') or NLEVP QUERY PROBLEMS
% returns a cell array containing the names of all problems
% in the collection.
% NLEVP('help','name') or NLEVP HELP NAME
% gives additional information on problem NAME, including number and
% meaning of input/output arguments.
% NLEVP('query','name') or NLEVP QUERY NAME
```
returns a cell array containing the properties of the problem NAME.

PROPERTIES = NLEVP('query','properties') or NLEVP QUERY PROPERTIES
returns the properties used to classify problems in the collection.
NLEVP('query',property1,property2,...) or NLEVP QUERY PROPERTY1...
lists the names of all problems having all the specified properties.

[T,TP,TPP,...] = NLEVP('eval',NAME,LAMBDA,ARG1,ARG2,...)
evaluates the matrix function T and its derivatives TP, TPP,...
for problem NAME at the scalar LAMBDA.

NLEVP('version') or NLEVP VERSION
prints version, release date, and number of problems
of the installed NLEVP collection.
V = NLEVP('version')
returns a structure V containing version information.
V consists of the fields v.number, v.date, and v.problemcount.

Available problems:

% Available problems:
% acoustic_wave_1d Acoustic wave problem in 1 dimension.
% acoustic_wave_2d Acoustic wave problem in 2 dimensions.
% bicycle 2-by-2 QEP from the Whipple bicycle model.
% bilby 5-by-5 QEP from Bilby population model.
% butterfly Quartic matrix polynomial with T-even structure.
% cd_player QEP from model of CD player.
% closed_loop 2-by-2 QEP associated with closed-loop control system.
% concrete Sparse QEP from model of a concrete structure.
% damped_beam QEP from simply supported beam damped in the middle.
% dirac QEP from Dirac operator.
% fiber NEP from fiber optic design.
% foundation Sparse QEP from model of machine foundations.
% gen_hyper2 Hyperbolic QEP constructed from prescribed eigenpairs.
% gen_tpal2 T-palindromic QEP with prescribed eigenvalues on the
% unit circle.
% gen_tantipal2 T-anti-palindromic QEP with eigenvalues on the unit
% circle.
% gun NEP from model of a radio-frequency gun cavity.
% hadeler NEP due to Hadeler.
% hospital QEP from model of Los Angeles Hospital building.
% intersection 10-by-10 QEP from intersection of three surfaces.
% loaded_string REP from finite element model of a loaded vibrating
% string.
% metal_strip QEP related to stability of electronic model of metal
% strip.
% mirror Quartic PEP from calibration of cadioptric vision system.
% mobile_manipulator QEP from model of 2-dimensional 3-link mobile manipulator.
% omnicam1 9-by-9 QEP from model of omnidirectional camera.
% omnicam2 15-by-15 QEP from model of omnidirectional camera.
% orr_sommerfeld Quartic PEP arising from Orr-Sommerfeld equation.
% pdde_stability QEP from stability analysis of discretized PDDE.
% planar_waveguide Quartic PEP from planar waveguide.
% plasma_drift Cubic PEP arising in Tokamak reactor design.
% power_plant 8-by-8 QEP from simplified nuclear power plant problem.
% QEP1 3-by-3 QEP with known eigensystem.
% QEP2 3-by-3 QEP with known, nontrivial Jordan structure.
% QEP3 3-by-3 parametrized QEP with known eigensystem.
% QEP4 3-by-4 QEP with known, nontrivial Jordan structure.
% QEP5 3-by-3 nonregular QEP with known Smith form.
% railtrack QEP from study of vibration of rail tracks.
% railtrack2 Palindromic QEP from model of rail tracks.
% relative_pose_5pt Cubic PEP from relative pose problem in computer vision.
% relative_pose_6pt QEP from relative pose problem in computer vision.
% schrodinger QEP from Schrodinger operator.
% shaft QEP from model of a shaft on bearing supports with a damper.
% sign1 QEP from rank-1 perturbation of sign operator.
% sign2 QEP from rank-1 perturbation of 2*sin(x) + sign(x) operator.
% sleeper QEP modelling a railtrack resting on sleepers.
% speaker_box QEP from finite element model of speaker box.
% spring QEP from finite element model of damped mass-spring system.
% spring_dashpot QEP from model of spring/dashpot configuration.
% time_delay 3-by-3 NEP from time-delay system.
% surveillance 27-by-20 QEP from surveillance camera callibration.
% wing 3-by-3 QEP from analysis of oscillations of a wing in an airstream.
% wiresaw1 Gyroscopic system from vibration analysis of wiresaw.
% wiresaw2 QEP from vibration analysis of wiresaw with viscous damping.

% Examples:
% coeffs = nlevp('railtrack')
% generates the matrices defining the railtrack problem.
% nlevp('help','railtrack')
% prints the help text of the railtrack problem.
% nlevp('query','railtrack')
% prints the properties of the railtrack problem.

% For example code to solve all polynomial eigenvalue problems (PEPs)
% in this collection of dimension < 500 with MATLAB's POLYEIG
% see NLEV_EXAMPLE.M.

% Reference:
% T. Betcke, N. J. Higham, V. Mehrmann, C. Schröder, and F. Tisseur.
% NLEVP: A Collection of Nonlinear Eigenvalue Problems,
% MIMS EPrint 2011.116, Manchester Institute for Mathematical Sciences,
% The University of Manchester, UK, 2011

% Check inputs
if nargin < 1, error('Not enough input arguments'); end
if ~ischar(name), error('NAME must be a string'); end

name = lower(name);

if strcmp(name,'query')
    if nargin == 1
        error('Not enough input arguments')
    end
    [varargout{1:nargout}] = nlevp_query(varargin{:});
    return
end

if strcmp('string',name)
name = 'spring';
warning('NLEVP:string_renamed','Problem string has been renamed spring. ')
end

if strcmp('version',name)
    [varargout{1:nargout}] = nlevp_version(varargin{:});
    return
end

switch name
    case 'help'
        if nargin < 2
            help nlevp
        else
            if ~nlevp_isoctave
                help(varargin{1})
            else
                % Uglier code necessary for Octave.
                eval(['help ', varargin{1}]);
            end
        end
    case 'eval'
        [varargout{1:max(nargout,1)}] = nlevp_eval(varargin{:});
    otherwise
        [varargout{1:nargout}] = feval(name,varargin{:});
end

5 The MATLAB Function nlevp_example

The toolbox contains a function nlevp_example.m that illustrates the use of nlevp. Running it provides a quick test that the toolbox is correctly installed. This function can be adapted in order to test the user’s own methods on subsets of NLEVP problems.

function nlevp_example(fname)
    %NLEVP_EXAMPLE Run POLYEIG on PEP problems from NLEVP.
    % NLEVP_EXAMPLE solves all the not-too-large PEP problems in NLEVP
    % by POLYEIG, sending output to the screen.
    % NLEVP_EXAMPLE(fname) directs partial output to the file named fname
    % (intended for generating output for NLEVP paper).

    if nargin == 0
        fid = 1;
    else
        fid = fopen(fname,'w');
    end
    s_rand = warning('off', 'NLEVP:random'); % For gen_hyper2.

    nmax = 500;
    probs = nlevp('query','pep');
    nprobs = length(probs);
    nprobs_total = length(nlevp('query','problems'));
    fprintf(fid,'NLEVP contains %2.0f problems in total,\n', nprobs_total);
    fprintf(fid,'of which %2.0f are polynomial eigenvalue problems\n', nprobs);
    fprintf(fid,'Run POLYEIG on the PEP problems of dimension at most %2.0f:\n', nmax);

    fprintf(fid,'Problem Dim Max and min magnitude of eigenvalues\n');
fprintf(fid, ' ------- --- ------------------------------------
');
m = ceil(nprobs/4);
j = 1;
for i=1:nprobs
    if fid ~= 1 && i == 9
        fprintf(fid, ' ...
');
        fid_save = fid;
        fid = 1;  % Omit output from this point on when writing to file.
    end
    coeffs = nlevp(probs{i});
    [n, nc] = size(coeffs{1});
    if n >= nmax
        fprintf(fid, '%20s %3.0f is a PEP but is too large for this test.
', ...
            probs{i}, n);
    elseif n ~= nc
        fprintf(fid, '%20s %3.0f is a PEP but is nonsquare.
', probs{i}, n);
    else
        % POLYEIG will convert sparse input matrices to full.
        e = polyeig(coeffs{:});
        fprintf(fid, '%20s %3.0f %9.2e %9.2e
', ...
            probs{i}, n, max(abs(e)), min(abs(e)));
        subplot(m,4,j)
        plot(real(e), imag(e), '.');
        title(probs{i}, 'Interpreter', 'none')
        % Tweaks.
        if strcmp(probs{i}, 'sign1'), ylim([-1 1]*1.5), end
        if strcmp(probs{i}, 'damped_beam')
            title([' ' probs{i}], 'Interpreter', 'none')
        end
        if strcmp(probs{i}, 'relative_pose_6pt')
            title([' ' probs{i}], 'Interpreter', 'none')
        end
        if strcmp(probs{i}, 'speaker_box') || strcmp(probs{i}, 'intersection')
            title([' ' probs{i}], 'Interpreter', 'none')
        end
        j = j+1;
    end
end
if nargin > 0, fclose(fid_save); end
warning(s_rand)

Part of the output of the function is shown in [1].

6 Contributing to the Collection

Contributions of suggested new problems for the collection are welcome. They can be sent to any of the authors. The following rules should be followed when providing new problems.

Write a LATEX file called \texttt{problem\_name.tex}, where \texttt{problem\_name} is the proposed name of your example, describing the problem. Here, \texttt{problem\_name} should be a string in lower case without any spaces. The \texttt{tex} file should consist of a \texttt{problem} environment, with first line stating the relevant identifiers for the problem (these properties are listed by \texttt{nlevp query properties}, and are explained in the companion document [1]):

\begin{problem}{problem\_name}{identifier1,identifier2,...}
This is a xxx-problem of dimension nnn.

7
It arises in ...
end\{problem\}

Provide your citations in a \texttt{bib} file; one \texttt{bib} file suffices even if multiple \texttt{tex} files are provided.

Write an M-file generating the coefficients of the example called \texttt{problem_name.m}. Document the M-file in the leading comment lines with the most important information from the \texttt{tex} file. If the problem is parameter dependent, set default values for any parameters not specified when the function is called. If you need extra data files, their names should begin with \texttt{problem_name}, e.g., \texttt{problem_name.mat}.

To specify a polynomial problem the first output of the M-file should be a cell array containing the coefficient matrices starting with the constant term. Thus if the first output is called \texttt{coeffs} and you want to define a PEP \( P(\lambda) = \sum_{i=0}^{k} \lambda^i A_i \), then \texttt{coeffs{1}}=\(A_0\), \texttt{coeffs{2}}=\(A_1\), …, \texttt{coeffs{k+1}}=\(A_k\).

The second output argument must be a function that computes the nonlinear scalar functions in the definition of the problem and their derivatives; for a polynomial eigenvalue problem this is trivially provided by a line of the form

\begin{verbatim}
fun = @(lam) nlevp_monomials(lam,k);
\end{verbatim}

Here, \texttt{nlevp_monomials.m} is a function provided with NLEVP in the \texttt{private} directory.

If a supposed solution is provided it should be returned in a structure \texttt{sol} with the following format:

- \texttt{sol.eval}: an \( m \times 1 \) vector, where \( m \) eigenvalues are provided,
- \texttt{sol.evec}: an \( m \times n \) matrix, where column \texttt{j} is the eigenvector corresponding to \texttt{sol.eval(j)}.

If both left and right eigenvectors are known, they should be returned in \texttt{sol.evec_left} and \texttt{sol.evec_right}.

References


8