MATCAS

a **MAT**lab tool using **C**oefficient **A**pproximation for the automatic solution of multichannel **S**chrödinger equations

Department of Applied Mathematics and Computer Science
Ghent University

WHAT

A Matlab tool for the automatic computation of bound-state eigenvalues and wavefunctions of coupled-channel Schrödinger equations. The user may specify the required eigenvalue by its index.

Details on the numerical methods can be found in:
V. Ledoux, M. Van Daele, Automatic computation of quantum-mechanical bound states and wavefunctions (2012).

Programming language

Tested with Matlab(R) version newer than 6.5. However, in principle, any recent version of Matlab(R) should work.

How to start

Download the package from http://www.nummath.ugent.be/SLsoftware
To run MATCAS the source files must be integrated into the Matlab path. This can be achieved by using the `addpath` command of Matlab. Or change your current folder to the MATCAS folder. To run the example.m files, enter e.g. `example1` in the Matlab command window.

Structure and Usage

The package consists of several .m files: some example driver scripts and two files containing the Matlab functions `computeEigenvalues` and `computeEigenfunction`. Some additional auxiliary functions are collected in the folder called `source`. A user of the package will, however, not call a function from the source folder directly.

- `example1.m / example11.m`

  example driver scripts
  Each `example.m` file computes the eigenvalues and eigenfunctions of a specific Schrodinger problem. The `example.m` files illustrate how to specify the problem and how to call the functions `computeEigenvalues` and `computeEigenfunction`.

As an example we consider the problem from [L.Gr. Ixaru, Phys. Rev. A 77 (2008) 064102] which has known eigenvalues and solutions. The 2×2 matrix potential for $x \geq 0$, is given by

$$V_{1,1}(x) = V_{2,2}(x) = V_{\mathrm{PT}}(x; 45, 1) + V_{\mathrm{PT}}(x; 39/2, 1/2) \tag{1}$$

$$V_{1,2}(x) = V_{2,1}(x) = V_{\mathrm{PT}}(x; 45, 1) - V_{\mathrm{PT}}(x; 39/2, 1/2) \tag{2}$$

where V_{PT} is the Pöschl-Teller potential

$$V_{\mathrm{PT}}(x; \nu, \alpha) = -\nu / \cosh^2(\alpha x).$$

The wavefunction tends to zero when $x \to +\infty$. In our experiments the infinite integration interval $x \geq 0$ has been cut at $x = 30$. The following code computes some eigenvalues and an eigenfunction of this problem.

```
% TEMPLATE

%Specify the endpoints of the integration interval:
system.a=0;
system.b=30;

%Specify the coefficient matrices of the boundary conditions:
system.A1=eye(2);
system.A2= zeros(2,2);
system.B1=eye(2);
system.B2= zeros(2,2);

%potential matrix function (function handle)
system.V=@potentialMatrix;

%Use the function computeEigenvalues to computes eigenvalues with specified index.
%Here indices 0 to 10, and input tolerance 1e-8
[EigvData,meshData]=computeEigenvalues(system,0,10,1e-8);

%The structure EigvData contains information on the computed eigenvalues:
disp(['Number of intervals in the mesh: ' num2str(length(meshData.h))]);
disp(sprintf('k \t  E_k \t\t\t\t ErrEst \t status'))
for i=1:length(EigvData.eigenvalues)
    disp(sprintf('%-3.0f\t %-16.12f\t %-2.0e\t\t %d', EigvData.indices(i),...
    EigvData.eigenvalues(i), EigvData.errorEstimations(i),EigvData.status(i)));
end

%make plots of the eigenfunction corresponding to the first eigenvalue which
%has multiplicity 1
[x,Y,YP]=computeEigenfunction(system,meshData,EigvData.eigenvalues(1),1);
figure
plot(x,Y(1,:,1))
hold on
plot(x,Y(2,:,1),'r')
title('Eigenfunction')
legend('y_1(x)','y_2(x)')
hold off
figure
plot(x,YP(1,:,1))
hold on
plot(x,YP(2,:,1),'r')
hold off
xlabel('x')
title('First derivative of eigenfunction')
legend('y''_1(x)','y''_2(x)')

function r=potentialMatrix(x)
  %returns the potential matrix evaluated in x
```

```
  %x may be a vector of values: make sure that the entries are vectorized, e.g.
  % x.^2 instead of x^2.
  r=zeros(2,2,length(x));
  r(1,1,:)=-45./cosh(1*x).^2-(39/2)./cosh(1/2*x).^2;
  r(2,2,:)=-45./cosh(1*x).^2-(39/2)./cosh(1/2*x).^2;
  r(1,2,:)=-45./cosh(1*x).^2+(39/2)./cosh(1/2*x).^2;
  r(2,1,:)=-45./cosh(1*x).^2+(39/2)./cosh(1/2*x).^2;
end
```

- `computeEigenvalues.m`.

  This .m-file implements the `computeEigenvalues` function:

```
[EigvData,meshData]=computeEigenvalues(system,kmin,kmax,tol)
INPUT:
     - system: a structure containing information on the Schrodinger problem
               system.a, system.b: endpoints of the integration interval
               system.A1,system.A2:
               system.B1,system.B2: parameters of the boundary conditions
               system.V: function handle to the potential matrix
                         function
     - kmin, kmax: lowest and highest index of the range of eigenvalues
                   to be computed
     - tol: user input tolerance (~ relative error in the eigenvalue
            approximations)
OUTPUT:
     - eigvData: a Matlab structure
                 eigvData.eigenvalues = vector containing the eigenvalue
                                        approximations
                 eigvData.status = vector of same length as eigvData.eigenvalues
                      eigvData.status(i)= 1 : no difficulties detected in the
                                        approximation of the i-th eigenvalue
                      eigvData.status(i)= -1: the newton-raphson process didn't
                                        converge to an eigenvalue,
                                        try a lower input tolerance
                      eigvData.status(i)= -2: too many iterations in the newton-
                                        raphson process. The requested accuracy
                                        may not have been reached.
                 eigvData.indices = eigenvalue indices

     - meshData: structure collecting all data associated to the mesh
                 constructed
                 meshData.tol : expected accuracy for the results
                                computed over this mesh
                 meshData.v0: vector of V_0 matrices, length of the vector
                              equals the number of mesh intervals
                 meshData.v1: vector of {\bar V}_1 matrices, length of the vector
                              equals the number of mesh intervals
                 meshData.v2: vector of {\bar V}_2 matrices, length of the vector
                              equals the number of mesh intervals
                 meshData.h: vector containing the length of the mesh intervals
                 meshData.dimat: vector of D matrices (orthogonal matrix
                                 used by the diagonalization process), the length of
                                 the vector equals the number of mesh intervals
                 meshData.cu,cup,cv,cvp: C-coefficients used by the
                                         sixth order CP algorithm
```

- `computeEigenfunction.m`

  This .m-file implements the `computeEigenfunction` function:

```
function [x,YM,YPM]=computeEigenfunction(system,md,E,multiplicity)
%computes the eigenfunctions and their first order derivatives corresponding
% to the eigenvalue E in the meshpoints of the mesh given by the input argument md.
OUTPUT:   x : vector of x-values (=meshpoints) where the eigenfunctions were
              evaluated
          YM: 3-dim matrix (n times nsteps times multiplicity)
               contains information on the eigenfunctions
               YM(i,j,k) = i'th element (1<=i<=dimension n of the system) in
                     the k'th eigenfunction evaluated in the jth meshpoint .
               Each YM(:,:,k) with ( 1<= k <= multiplicity ) represents an
                     eigenfunction.
           YMP: similar for the first order derivatives
```

NOTE: the eigenfunction and its derivative are only evaluated in the meshpoints (the values of these points are returned by the function `computeEigenfunction` in the vector `x`). This may not be sufficient to produce a nice smooth plot of a higher eigenfunction. The evaluation of the eigenfunctions in user specified points is future work.

- `source\computeVcoeffs.m`

  The function `computeVcoeffs` returns the coefficient matrices $\mathbf{V}_m$ in the polynomial approximation of degree 2 underlying the sixth order CP scheme:

$$\mathbf{V}(x_i+\delta) \approx \sum_{m=0}^{2} \mathbf{V}_m h^m P_m^*(\delta/h_i)$$

  with $P_m^*(\delta/h_i)$ the shifted Legendre polynomials and the matrix weigths $\mathbf{V}_m$ given by the method of least squares:

$$\mathbf{V}_0 = \frac{1}{h_i}\int_0^{h_i} \mathbf{V}(x_i+\delta)d\delta,$$
$$\mathbf{V}_m = \frac{(2m+1)}{h_i^{m+1}}\int_0^{h_i} \mathbf{V}(x_i+\delta)P_m^*(\delta/h_i)d\delta, \;\; m=1,2.$$

  These weigths are computed by a Gauss-Legendre quadrature.

- `source\constructMesh.m`

  This function implements the automatic stepsize selection algorithm.

- `source\getBracket.m`

  This function generates a good initial guess for the Newton-Raphson procedure which is used in the eigenvalue shooting process. The Prüfer procedure is applied to generate reasonably tight upper and lower bounds for the eigenvalue E_k sought, i.e. a bracket $[\hat{E}, \bar{E}]$ is determined such that $\mathcal{I}(\hat{E}) = k$ and $\mathcal{I}(\bar{E}) = k+1$ or $\bar{E} - \hat{E} < tol$. $E = (\hat{E}+\bar{E})/2$ is then passed as the initial trial value for the eigenvalue E_k.

- `source\getIndex.m`

  This function returns the value of the indexing function $\mathcal{I}(E)$ and thus implements the numerical Prüfer procedure.

- `source\getTransferMatrices.m`

  This function returns the entries $\mathbf{U}, \mathbf{U}', \mathbf{W}, \mathbf{W}'$ of the propagation matrix for the sixth order CP scheme, which are given by:

$$\mathbf{U}^D(h_i) = \boldsymbol{\xi}(\mathbf{Z}) + \sum_{m=1}^{2} \mathbf{C}_m^{(U)} \boldsymbol{\eta}_m(\mathbf{Z}), \tag{3}$$

$$h_i(\mathbf{U}^D)'(h_i) = \mathbf{Z}\boldsymbol{\eta}_0(\mathbf{Z}) + \sum_{m=0}^{2} \mathbf{C}_m^{(U')} \boldsymbol{\eta}_m(\mathbf{Z}), \tag{4}$$

$$\mathbf{W}^D(h_i)/h_i = \boldsymbol{\eta}_0(\mathbf{Z}) + \mathbf{C}_2^{(W)} \boldsymbol{\eta}_2(\mathbf{Z}), \tag{5}$$

$$(\mathbf{W}^D)'(h_i) = \boldsymbol{\xi}(\mathbf{Z}) + \sum_{m=1}^{2} \mathbf{C}_m^{(W')} \boldsymbol{\eta}_m(\mathbf{Z}) \tag{6}$$

with

$$\begin{aligned}
\mathbf{C}_1^{(U)} &= -\bar{\mathbf{V}}_1/2 + [\bar{\mathbf{V}}_1, \bar{\mathbf{V}}_0]/24 \\
\mathbf{C}_2^{(U)} &= -\bar{\mathbf{V}}_1^2/24 + [4\bar{\mathbf{V}}_1 - 3\bar{\mathbf{V}}_2, \bar{\mathbf{V}}_0]/24 \\
\mathbf{C}_0^{(U')} &= \bar{\mathbf{V}}_2/2 + [\bar{\mathbf{V}}_1, \bar{\mathbf{V}}_0]/24 - [\bar{\mathbf{V}}_1, \bar{\mathbf{V}}_2]/120 \\
\mathbf{C}_1^{(U')} &= -3\bar{\mathbf{V}}_2/2 - \bar{\mathbf{V}}_1^2/24 + [\bar{\mathbf{V}}_1 - \bar{\mathbf{V}}_2, \bar{\mathbf{V}}_0]/8 \\
\mathbf{C}_2^{(U')} &= -7\bar{\mathbf{V}}_1^2/24 + [\bar{\mathbf{V}}_1, \bar{\mathbf{V}}_2]/8 + [3\bar{\mathbf{V}}_2, \bar{\mathbf{V}}_0] \\
\mathbf{C}_2^{(W)} &= -\bar{\mathbf{V}}_2/2 + [\bar{\mathbf{V}}_1, \bar{\mathbf{V}}_0]/24 \\
\mathbf{C}_1^{(W')} &= \bar{\mathbf{V}}_1/2 + [\bar{\mathbf{V}}_1, \bar{\mathbf{V}}_0]/24 \\
\mathbf{C}_2^{(W')} &= -\bar{\mathbf{V}}_1^2/24 + [-2\bar{\mathbf{V}}_1 + 3\bar{\mathbf{V}}_2, \bar{\mathbf{V}}_0]/24
\end{aligned}$$

For notational brevity in the $\mathbf{C}_m$ matrices, the upper label D has been suppressed and we introduced $\bar{\mathbf{V}}_s = \mathbf{V}_s h_i^{s+2}, s = 0, 1, 2$ and the matrix commutator notation $[\mathbf{A}, \mathbf{B}] = \mathbf{AB} - \mathbf{BA}$.

- `source\shootForEigenvalue.m`

  This function implements the shooting algorithm

Algorithm 1 Shooting for the eigenvalue E_k

1: Input: a trial value E, a mesh $a = x_0 < x_1 < \cdots < x_N = b$, with stepsizes $h_i = x_{i+1} - x_i$.
2: Choose a meshpoint $c = x_m, 0 \le m \le N$ as the matching point.
3: Set up initial values for $\mathbf{\Psi}_L$ and $\mathbf{\Psi}_R$ satisfying the boundary conditions at a and b respectively.
4: **repeat**
5: **for** $i = 0$ to $m - 1$ **do**
6: Compute $\mathbf{U}(h_i), \mathbf{W}(h_i), \mathbf{U}'(h_i), \mathbf{W}'(h_i)$ by the 6th order CA method.
7: Propagate $\mathbf{\Psi}_L$ over the interval $[x_i, x_{i+1}]$:
 $\mathbf{\Psi}_L(x_{i+1}) = [\mathbf{W}(h_i) + \mathbf{U}(h_i)\mathbf{\Psi}_L(x_i)][\mathbf{W}'(h_i) + \mathbf{U}'(h_i)\mathbf{\Psi}_L(x_i)]^{-1}$
8: **end for**
9: **for** $i = N$ down to $m + 1$ **do**
10: Compute $\mathbf{U}(h_i), \mathbf{W}(h_i), \mathbf{U}'(h_i), \mathbf{W}'(h_i)$ by the 6th order CA method.
11: Propagate $\mathbf{\Psi}_R$ over the interval $[x_{i-1}, x_i]$:
 $\mathbf{\Psi}_R(x_{i-1}) = [-\mathbf{W}(h_i) + \mathbf{W}'(h_i)\mathbf{\Psi}_R(x_i)][\mathbf{U}(h_i) - \mathbf{U}'(h_i)\mathbf{\Psi}_R(x_i)]^{-1}$
12: **end for**
13: Adjust E to solve the equation $\det(\mathbf{\Psi}_L(c) - \mathbf{\Psi}_R(c)) = 0$.
14: **until** E sufficiently accurate (e.g. until the difference between two subsequent E values is smaller than some user input tolerance)

Disclaimer