Intuitive Interface for Solving Linear and Nonlinear System of Equations

Zhonggang Zeng*

Northeastern Illinois University, USA
zzeng@neiu.edu,
homepages.neiu.edu/∼zzeng

Abstract. An innovative approach is proposed in designing intuitive interfaces for solving systems of linear and nonlinear equations over Cartesian products of general vector spaces. The interfaces enable scientific computing practitioners and learners to enter equations in WYSIWYG manner, lets the software generate vector representations of equations/variables internally and outputs the solutions in the desired forms automatically. Such interfaces save more time than algorithmic improvement for one-time users and students.

Keywords: interface, system of equations

1 Introduction

Solving linear and nonlinear systems of equations is one of the fundamental tasks in scientific computing. Existing software packages, however, require systems to be in matrix-vector form $A\,\mathbf{x} = \mathbf{b}$, or represented using multivariate functions in the form of

$$\begin{cases} f_1(x_1, \ldots, x_n) & = & 0 \\ \vdots & \vdots & \vdots \\ f_m(x_1, \ldots, x_n) & = & 0 \end{cases}. \tag{1}$$

Variables are allowed only as arrays of real/complex numbers. More common in practical computation and classroom teaching, linear equations are formulated in the form of

$$L(\mathbf{x}_1, \ldots, \mathbf{x}_n) \;=\; (\mathbf{b}_1, \ldots, \mathbf{b}_m)$$

where L is a linear transformation, and nonlinear equations are given as

$$\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) \;=\; (0, \ldots, 0)$$

where $\mathbf{f}$ is a differentiable mapping between certain spaces. Furthermore, the variables $\mathbf{x}_1, \ldots, \mathbf{x}_n$ can be arrays of numbers, vectors, matrices, polynomials, functions of certain forms, etc. It is a tedious, time-consuming and error-prone

* Research is supported in part by NSF under grant DMS-1620337.

process to transform the equations into the function representations before being solved by existing software and, after obtaining results, to make backward representations. Such back-and-forth representation processes are particularly daunting for beginners and students. We propose an innovative approach in designing intuitive interfaces for solving equations over Cartesian products of general vector spaces in our Matlab toolbox NAClab[1] for numerical algebraic computation [3]. The interfaces enable computing practitioners and learners to enter equations in WYSIWYG manner, let the software generate vector representations of equations/variables internally and output the solutions in the desired forms automatically. Such interfaces save more time than algorithmic improvement for one-time users and students.

2 Interface for solving linear systems of equations

In practical scientific computing, a general system of linear equations is in the form of

$$L(\mathbf{x}_1, \ldots, \mathbf{x}_n) \quad = \quad (\mathbf{b}_1, \ldots, \mathbf{b}_m) \tag{2}$$

where

$$L \,:\, X_1 \times \cdots \times X_n \longrightarrow Y_1 \times \cdots \times Y_m$$

is a linear transformation between Cartesian products of general vector spaces such as $\mathbb{C}^n$ of n-dimensional vectors, $\mathbb{P}_n$ of univariate or multivariate polynomials of degree up to n, $\mathbb{C}^{m\times n}$ of $m\times n$ matrices, etc. Furthermore, the linear transformation L may be under-determined, overdetermined or rank-deficient. There are two drawbacks in current software for numerical solutions of linear system of equations:

- Systems can only be solved in the matrix-vector form $A\,\mathbf{x} = \mathbf{b}$ to which the general linear system (2) must be transformed by users.
- Rank-deficient linear systems may not be solvable accurately from empirical data.

Our Matlab toolbox NAClab is developed with an attempt to fill these two gaps. The interface module `LinearSolve` for solving general linear systems requires users to provide the following input items:

- The Matlab function or Matlab anonymous function for carrying out the evaluation of the linear transformation L.
- The domain $X_1 \times \cdots \times X_n$ in the form of a typical vector $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ with all the relevant components as a Matlab cell array.
- The parameters of the linear transformation as a cell array.
- The right-hand side vector $(\mathbf{b}_1, \ldots, \mathbf{b}_m)$.
- The error tolerance ε.

We illustrate the interface `LinearSolve` in the following example.

[1] `http://homepages.neiu.edu/~zzeng`

Example 1. Suppose, with the data error bound 10^{-4}, the polynomials

$$\begin{aligned} \tilde{f}(x) &= 5.99999 - 9\,x + 3\,x^3 - 4\,x^4 + 6\,x^5 - 2\,x^7 \\ \tilde{g}(x) &= -2 + 5\,x - 3\,x^2 - x^3 + x^4 + 2\,x^5 - 3.00002\,x^6 + x^8 \end{aligned}$$

are empirical data of a polynomial pair f and g with a greatest common divisor

$$u = \mathit{gcd}(f,g) \approx 2 - 2.99999\,x + 1.00001\,x^3.$$

given approximately. It is known that the greatest common divisor is in the range of the linear transformation

$$\begin{array}{rcccc} L & : & \mathbb{P}_5 \times \mathbb{P}_4 & \longrightarrow & \mathbb{P}_{12} \\ & & (p,q) & \longmapsto & p\,f + q\,g \end{array} \tag{3}$$

and the kernel

$$\mathit{Kernel}(L) = \mathit{span}\left\{\left(\frac{f}{u}, \frac{g}{u}\right)\right\}$$

where the notation $\mathbb{P}_n$ denotes the vector space of polynomials with degrees up to n. The question: *Can we use the empirical data of f, g and u to solve the linear equation*

$$p\,\tilde{f} + q\,\tilde{g} = \tilde{u}$$

for $(p,q) \in \mathbb{P}_5 \times \mathbb{P}_4$ approximately with an accuracy comparable to the data accuracy? There appear to be no available software for finding numerical solution of such a rank-deficient linear system, and all software systems require users to go through a tedious process of transforming the equation into a matrix-vector form.

NACLAB provides a comprehensive linear equation solver `LinearSolve` for such problems directly. Instead of constructing the representation matrix for the linear transformation, write a simple Matlab anonymous function implementing the linear transformation (3) with parameters f and g as it is:

```
>> L = @(p,q,f,g) ...          % Matlab anonymous function for the linear transformation
   PolynomialPlus(PolynomialTimes(p,f),PolynomialTimes(q,g));   % L : (p,q) |--> p*f+q*g
```

Here the syntax rules require the input items to start with variables p and q followed by parameters f and g. The modules `PolynomialPlus` and `PolynomialTimes` are interface functionalities in NACLAB to perform polynomial additions and polynomial multiplications with polynomials entered as character strings in WYSIWYG style:

```
>> f = '5.99999 - 9*x + 3*x^3 - 4*x^4 + 6*x^5 - 2*x^7';              % data for polyn. f
>> g = '-2 + 5*x - 3*x^2 - x^3 + x^4 + 2*x^5 - 3.00002*x^6 + x^8';   % data for polyn. g
>> u = '2-2.99999*x+1.00001*x^3';                                    % data for polyn. u
```

Define the domain of the linear transformation by providing two polynomials in $\mathbb{P}_5$ and $\mathbb{P}_4$ with all the relevant monomials, along with the parameter:

```
>> domain = {'1+x+x^2+x^3+x^4+x^5','1+x+x^2+x^3+x^4'};   % domain of variable (p,q) of L
```

```
>> parameter = {f,g};                    % parameter cell array of the linear transf. L
>> error = 1e-4;                                                  % error tolerance
```

Call the NACLAB interface `LinearSolve` with input items consist of the linear transformation array {`L, domain, parameter`}, the right-hand side `u`, and error tolerance 10^{-4}:

```
>> [Z,K,lcond,res] =  % solve L(p,q) = u, in the 'domain' with 'parameter' within 'error'
   LinearSolve({L,domain,parameter}, u, error);
```

The cell array `Z` contains the numerical minimum-norm solution in $\mathbb{P}_5 \times \mathbb{P}_4$ in WYSIWYG style

```
>> Z                                      % display the numerical minumum-norm solution
Z =
 '0.315213638138925 + 0.0325895732095521*x + 0.0217037544785598*x^2 + 0.0542602119088428*x^3
+ 0.135649516609882*x^4 + 0.023908132818779*x^5'  '-0.054358026776926 + 0.0434095337991097*x
+ 0.108521146411912*x^2 + 0.271298482527209*x^3 + 0.0478164654136125*x^4'
```

The cell array `K` contains the 1-dimensional numerical Kernel in $\mathbb{P}_5 \times \mathbb{P}_4$ in WYSIWYG style

```
>> K{:}                                          % display the basis for numerical kernel
ans =
    '0.249999389255342 - 0.250001189657339*x - 0.250002152061042*x^5'    '0.749997620229951
- 0.500002204088944*x^4'
```

The numerical solution can be verified using the linear transformation function L.

```
>> p = Z{1};  q = Z{2};                                                % extract p and q
>> h = L(p,q,f,g);                              % evaluate h = L(p,q) with parameter f and g
>> PolynomialClear(h,1e-5)                    % clear numerical tiny coefficients below 1e-5
ans =
1.99999473025102 - 2.9999948313716*x + 1.00000604534541*x^3
```

The output `lcond` and `res` show the condition number for the linear equation is healthy at 52.1 and the residual is below the error tolerance at about 5.27×10^{-6}.

The user can study the linear equation further by investigating the representation matrix of the linear equation by calling `LinearTransformMatrix` in NACLAB using the above defined input items

```
>> A = LinearTransformMatrix(L,domain,parameter);          % representation matrix of L
```

obtaining the 13×11 matrix with respect to natural bases for the domain and codomain

$$\begin{bmatrix}
5.99999 & 0 & 0 & 0 & 0 & 0 & -2.0 & 0 & 0 & 0 & 0 \\
-9.0 & 5.99999 & 0 & 0 & 0 & 0 & 5.0 & -2.0 & 0 & 0 & 0 \\
0 & -9.0 & 5.99999 & 0 & 0 & 0 & -3.0 & 5.0 & -2.0 & 0 & 0 \\
3.0 & 0 & -9.0 & 5.99999 & 0 & 0 & -1.0 & -3.0 & 5.0 & -2.0 & 0 \\
-4.0 & 3.0 & 0 & -9.0 & 5.99999 & 0 & 1.0 & -1.0 & -3.0 & 5.0 & -2.0 \\
6.0 & -4.0 & 3.0 & 0 & -9.0 & 5.99999 & 2.0 & 1.0 & -1.0 & -3.0 & 5.0 \\
0 & 6.0 & -4.0 & 3.0 & 0 & -9.0 & -3.00002 & 2.0 & 1.0 & -1.0 & -3.0 \\
-2.0 & 0 & 6.0 & -4.0 & 3.0 & 0 & 0 & -3.00002 & 2.0 & 1.0 & -1.0 \\
0 & -2.0 & 0 & 6.0 & -4.0 & 3.0 & 1.0 & 0 & -3.00002 & 2.0 & 1.0 \\
0 & 0 & -2.0 & 0 & 6.0 & -4.0 & 0 & 1.0 & 0 & -3.00002 & 2.0 \\
0 & 0 & 0 & -2.0 & 0 & 6.0 & 0 & 0 & 1.0 & 0 & -3.00002 \\
0 & 0 & 0 & 0 & -2.0 & 0 & 0 & 0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 0 & 0 & -2.0 & 0 & 0 & 0 & 0 & 1.0
\end{bmatrix}$$

This is the matrix representation of the linear system the user would have to construct without using the interface `LinearSolve`. □

In fact, the module `LinearTransformMatrix` is the most crucial component of the interface `LinearSolve` that automates the process of solving linear systems and frees users from the tedious representation process.

Notice that there is a profound difference in `LinearSolve` compared to currently standard linear system solving implementations: The representing matrix shown above is considered numerically rank-deficient by virtue of a singular value 0.000003318035559 that is below the error tolerance. A number of magnitude below error tolerance can be considered a zero.

Standard linear system solvers treat the matrix as full-ranked but the condition number 6.6×10^6 suggests a questionable solution due to an error tolerance 10^{-4}. Numerical kernel is not returned in standard implementations.

3 Interface for solving nonlinear system of equations

Similar to linear cases, a general system of nonlinear equations in scientific computing is usually in the form of

$$\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) \quad = \quad (0, \ \ldots, \ 0)$$

where $\mathbf{f} : X_1 \times \cdots \times X_n \longrightarrow Y_1 \times \cdots \times Y_m$ is a differentiable mapping between Cartesian products of vector spaces where $\mathbf{f}$ may be square (i.e. $m = n$) or overdetermined ($m > n$). The most commonly used method for solving such a nonlinear system is Newton's iteration or, in overdetermined case, the Gauss-Newton iteration. For beginners and those who needs to solve such a nonlinear system only once or twice, the most daunting and time consuming part of computation is to transform the system into its multivariate function form (1) and to construct the corresponding Jacobian matrix. Our interface `GaussNewton` in NACLAB is an attempt to simplify the representation into a few WYSIWYG steps. We shall use the following example to illustrate the process.

Example 2. A defective eigenvalue λ_* of a matrix $A \in \mathbb{C}^{n \times n}$ can be calculated accurately by solving the equation

$$\mathbf{g}(\lambda, \ X) \quad = \quad (O, O)$$

where, knowing the multiplicity support is $m \times k$, the holomorphic mapping $\mathbf{g}$ is given as

$$\begin{array}{rccl} \mathbf{g} & : & \mathbb{C} \times \mathbb{C}^{n\times k} & \longrightarrow & \mathbb{C}^{n\times k} \times \mathbb{C}^{m\times k} \\ & & (\lambda, X) & \longmapsto & (A\,X - \lambda\,X - X\,S,\ C^{\mathsf{H}}\,X - T)\,. \end{array}$$

with constant parameters A, S, C and T. Detailed theoretical and computational issues on such equations can be found in [4]. For purpose of illustration, we use the following parameters:

$$n = 9, \quad m = 3, \quad k = 2,$$

$$A = \begin{bmatrix} 3 & 3 & 3 & 3 & -2 & 2 & 1 & -1 & -1 \\ 1 & 4 & 4 & 3 & 0 & 2 & 0 & -1 & 0 \\ 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & -1 \\ -2 & -3 & -4 & -2 & 6 & -2 & -2 & 1 & 3 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 3 \\ 2 & 4 & 4 & 5 & -3 & 5 & 3 & -3 & -1 \\ 1 & 0 & 1 & 1 & -2 & 0 & 3 & 0 & -1 \\ 1 & 6 & 7 & 7 & 2 & 6 & 1 & -4 & 2 \\ 0 & 0 & 0 & 0 & -6 & 0 & 0 & 0 & -4 \end{bmatrix},$$

$$S = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix},$$

and C is a 9×3 random matrix. The exact eigenvalues is $\lambda_* = 2$. Using a standard numerical eigenvalue solver, one can only obtain a few accurate digits of the eigenvalue, say $\lambda_0 = 1.9999$ due to the defectiveness of the eigenvalue.

On top of the time-consuming task of transforming the system of equations in $\mathbf{g}(\lambda, X) = (O, O)$, the main difficulties for a one-time user include formulating the Jacobian matrix. As a key aspect of our interface, it is much more convenient to consider the Jacobian of the mapping $\mathbf{g}$ at any (λ_0, X_0) as a linear transformation that is also known as Fréchet derivative

$$\begin{array}{rccl} J(\lambda_0, X_0) & : & \mathbb{C} \times \mathbb{C}^{n\times k} & \longrightarrow & \mathbb{C}^{n\times k} \times \mathbb{C}^{m\times k} \\ & & (\lambda, X) & \longmapsto & (-\lambda\,X_0 + (A - \lambda_0 I)\,X - X\,S,\ C^{\mathsf{H}}\,X)\,. \end{array}$$

with the same domain/codomain of the mapping $\mathbf{g}$ and additional paraemters λ_0, X_0. The main innovation of our interface development is allow users to enter both the mapping $\mathbf{g}$ and the Jacobian $J(\lambda_0, X_0)$ into Matlab as either function m-files or in-line anonymous functions exactly as they are:

```
>> g = @(lambda, X, A, S, C, T) ...          % Matlab anonymous function for the mapping g
      {A*X - lambda*X - X*S, C'*X - T}; % g : (lambda,X) |--> (A*X-lambda*X-X*S, C'*X-T)
>> J = @(lambda, X, lambda0, X0, A, S, C, T) ... % Matlab anonymous function for Jacobian
      {-lambda*X0 + A*X - lambda0*X - X*S, C'*X};
```

The syntax rules stipulate that the input items start with the common variables `lambda, X` followed by the common parameters `A, S, C, T`, and the additional parameters `lambda0, X0` for J are in between. The transformation of the system into the multivariate function form (1) is not needed either by the user or in the internal implementation since what is required is the evaluation of the mapping. The construction of the Jacobian matrix is carried out internally by the interface `LinearSolve` described in the previous section.

After using `LinearSolve` to solve the overdetermined linear system of equations

$$\mathbf{g}(\lambda_0, X) \;=\; (O, O) \quad \text{for} \quad X \in \mathbb{C}^{n\times k}$$

for its least squares solution X_0, we can apply the Gauss-Newton iteration from the initial iterate (λ_0, X_0) and solve the nonlinear equation $\mathbf{g}(\lambda, X) = (O, O)$ by the NACLAB interface `GaussNewton` with a few intuitive statements:

```
>> S = [0 1; 0 0]; T = [1 0; 0 0; 0 0]; C = rand(9,3);        % enter parameters S, T, C
>> domain = {1, ones(9,2)}                    % domain of (lambda,X) for mappings g and J
>> parameters = {A, S, C, T};                        % common parameters for both g and J
>> z0 = {1.9999,X0}                                                % initial iterate
>> [z,res,fcond] = ...                              % call GaussNewton with input items
   GaussNewton({g,domain,parameters},J,z0,2,5e-10);      % and display the 1st component
      Step   1:     residual =   2.89e-03
   1.989243561668298
      Step   2:     residual =   6.89e-04
   2.000311883756314
      Step   3:     residual =   6.13e-08
   2.000000057498796
      Step   4:     residual =   4.94e-15
   1.999999999999999
  ...
```

The result includes an accurate approximation of the eigenvalue $\lambda_* = 2$.

As shown in the example, the users do not need to transform the system into multivariate form, nor do they need to construct the representation matrix of the Jacobian. Instead, the system is entered into Matlab directly as it is defined and the Jacobian is conveniently entered as a linear transformation in WYSIWYG style.

4 Underlying theory and technical contribution

The underlying theory of the interface is quite simple in basic linear algebra. Let $\mathcal{V}$ and $\mathcal{W}$ be general vector spaces, possibly Cartesian products of vector spaces over a number field, say $\mathbb{C}$. Vectors in $\mathcal{V}$ and $\mathcal{W}$ can be represented as column vectors in $\mathbb{C}^n$ and $\mathbb{C}^m$ respectively via isomorphisms that can be denoted by $\phi : \mathcal{V} \to \mathbb{C}^n$ and $\psi : \mathcal{W} \to \mathbb{C}^m$ respectively. A linear transformation $L : \mathcal{V} \to \mathcal{W}$ can thus be represented as a matrix T in $\mathbb{C}^{m\times n}$ that makes the following diagram commute:

$$\begin{array}{ccc} \mathcal{V} & \xrightarrow{\;L\;} & \mathcal{W} \\ {\scriptstyle\phi}\downarrow & & \uparrow{\scriptstyle\psi^{-1}} \\ \mathbb{C}^n & \xrightarrow{\;T\;} & \mathbb{C}^m \end{array}$$

Namely, we have the identity

$$L(\mathbf{v}) \;\equiv\; \psi^{-1}\big(T\,\phi(\mathbf{v})\big).$$

Let $\{\mathbf{v}_1, \ldots \mathbf{v}_n\}$ and $\{\mathbf{w}_1, \ldots \mathbf{w}_m\}$ be some kind of standard bases for $\mathcal{V}$ and $\mathcal{W}$ respectively. Then the representation matrix can be constructed column-by-column according to

$$T \quad = \quad \big[\psi(L(\mathbf{v}_1)), \ldots, \psi(L(\mathbf{v}_n))\big]$$

where the j-th column is $\psi(L(\mathbf{v}_j)) \in \mathbb{C}^m$ for $j = 1, 2, \ldots, n$.

At current stages of development, we assume the domain $\mathcal{V}$ and the codomain $\mathcal{W}$ are Cartesian products of matrix spaces and polynomial spaces. A standard basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ for the domain $\mathcal{V}$ can be extracted automatically from a typical vector as input item for the domain, so can a standard basis $\{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$ for the codomain $\mathcal{W}$. The aforementioned isomorphisms ϕ and ψ can be implemented in a straightforward process due to the simplicity of the standard bases for the spaces of matrices and polynomials.

The theoretical foundation for the numerical solution of rank-deficient linear systems of equations is related to the subject of numerical rank-revealing [1, 2] and beyond the scope of this abstract.

For the nonlinear system of equations $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = (0, \ldots, 0)$ where $\mathbf{f} : X_1 \times \cdots \times X_n \to Y_1 \times \cdots \times Y_m$ is a differentiable mapping between Cartesian products of general vector spaces, the Gauss-Newton iteration

$$\begin{aligned}(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_n^{(k+1)}) \;&=\; (\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)}) - J(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)})^{\dagger}\, \mathbf{f}(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)}) \\ &\qquad k = 0, 1, \ldots\end{aligned}$$

where

$$J(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)}) \,:\, X_1 \times \cdots \times X_n \longrightarrow Y_1 \times \cdots \times Y_m$$

is the Jacobian (linear transformation) of $\mathbf{f}$ at $(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)})$ and $(\cdot)^{\dagger}$ denote the pseudo-inverse. At every step of the Gauss-Newton iteration, linear system of equations

$$\begin{aligned}J(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)})\,(\Delta\mathbf{x}_1, \ldots, \Delta\mathbf{x}_n) \;&=\; -\,\mathbf{f}(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)}) \\ &\qquad \text{for} \quad (\Delta\mathbf{x}_1, \ldots, \Delta\mathbf{x}_n) \in X_1 \times \cdots \times X_n\end{aligned} \tag{4}$$

is to be solved for its least squares solution so that

$$\mathbf{x}_j^{(k+1)} \quad = \quad \mathbf{x}_j^{(k)} + \Delta\mathbf{x}_j \qquad \text{for} \qquad j = 1, 2, \ldots, n.$$

As a result, an interface for solving the system $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = (0, \ldots, 0)$ of general nonlinear systems of equations can be built on top of the interface for solving general linear systems so that the system (4) can be solved at every step of the Gauss-Newton iteration. The key is to treat the Jacobian J at every point $(\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)})$ as a linear transformation with the same pair of domain/codomain as the mapping $\mathbf{f}$ and can be implemented as Matlab function in WYSIWYG style. Most importantly, the Jacobian as a linear transformation

can be derived from elementary differentiation rules. In contrast, the Jacobian as a matrix is tedious to construct.

The interfaces `LinearSolve` and `GaussNewton` in NACLAB are particularly useful for research in numerical algebraic computation and classroom teaching, in which we often need to experiment with various formulations of computational models and each system of equations needs to be solved mostly only once or twice. In such settings, the efficiency and complexity of the underlying algorithm are secondary since the sizes of testing problems are usually small. Substantial portion of time and effort are spent on setting up the systems into a format acceptable to the existing software, particularly in representing the system into matrix-vector form or multivariate function form. For students and beginners, such time-consuming and error-prone processes are daunting and may even be intimidating. The equation-solving interfaces make it possible for our graduate students to conduct computing projects involving sophisticated algebraic equations.

References

1. T.-Y. Li and Z. Zeng, *A rank-revealing method with updating, downdating and applications* , SIAM J. Matrix Analysis and Applications, vol. 26, pp 918–946, 2005.
2. T.-L. Lee, T.-Y. Li and Z. Zeng, *A rank-revealing method with updating, downdating and applications. Part II* , SIAM J. Matrix Analysis and Applications, vol. 31, pp 503–525, 2009.
3. Z. Zeng, *NAClab: A Matlab toolbox for numerical algebraic computation*, ACM Communications in Computer Algebra, vol 47, issue 3/4, pp 170-173, 2013.
4. Z. Zeng, *Sensitivity and computation of a defective eigenvalue*, SIAM J. Matrix Analysis and Applications, vol. 37, pp. 798–817, 2016.