nofoma

Normal forms of matrices

1.0

1 June 2022

Meinolf Geck

Meinolf Geck
Email: meinolf.geck@mathematik.uni-stuttgart.de
Homepage: https://pnp.mathematik.uni-stuttgart.de/idsr/idsr1/geckmf/
Address: Fachbereich Mathematik, Pfaffenwaldring 57, 70569
Stuttgart, Germany

Contents

Chapter 1

The nofoma package

1.1 Maximal vectors and normal forms

Let K be a field and A be an $n \times n$-matrix over K. This package provides functions for computing a maximal vector for A, the Frobenius normal form of A and the Jordan-Chevalley decomposition of A.

For any (row) vector $v \in K^n$, the local minimal polynomial is the unique monic polynomial $f \in K[X]$ of smallest possible degree such that $v.f(A) = 0$. (Note that, as usual in GAP, matrices act on the right on row vectors.) It is known that there always exists a $v \in K^n$ such that the local minimal polynomial of v equals the minimal polynomial of A; such a v is called a 'maximal vector' for A.

The currently best algorithm for computing the minimal polynomial of A is probably that of Neunhoeffer–Praeger (already implemented in GAP). One of the purposes of this package is to modify that algorithm so that it also includes the computation of a maximal vector for A; this is done by the function 'MaximalVectorMat'. Once this is available, the Frobenius normal form (or rational canonical form) of A can be computed efficiently by a recursive algorithm.

Finally, we also provide a function for computing the Jordan-Chevalley decomposition of A. Usually, this is obtained as a consequence of the Jordan normal form of A, but in general the latter is difficult to compute because one needs to know the eigenvalues of A. However, there is an elegant algorithmic approach (going back to Chevalley), which is inspired by the Newton iteration for finding the zeroes of a function; and it does not require the knowledge of the eigenvalues of A. This is implemented in the function 'JordanChevalleyDecMat'.

Further references are provided within the help menu of each function.

Any comments welcome! Meinolf Geck, June 2022

1.2 Copyright and installation of the nofoma package

To install this package first unpack it inside some GAP root directory in the subdirectory 'pkg' (see the section 'Installing a GAP Package' of the GAP reference manual). Then 'nofoma' can already be loaded and used (just type LoadPackage("nofoma");).

1.3 The main functions

1.3.1 SpinMatVector

▷ `SpinMatVector(A, v)` (function)

'SpinMatVector' computes the smallest subspace containing the vector v that is invariant under the matrix `A`. The output is a quadruple, with

- 1st component = basis of that subspace in row echelon form;
- 2nd component = matrix with rows v, $v.A$, $v.A^2$, ..., $v.A^{d-1}$ (where d is the degree of the local minimal polynomial of v);
- 3rd component = the coefficients a_0, a_1, ..., a_d of the local minimal polynomial; and
- 4th component = the positions of the pivots of the first component.

Example

```
gap> A:=[ [   5,   2,  -4,   2 ],
>         [  -1,   0,   2,  -1 ],
>         [  -1,  -1,   3,  -1 ],
>         [ -13,  -7,  14,  -6 ] ];
gap> SpinMatVector(a,[1,0,0,0]);
[ [ [ 1, 0, 0, 0 ], [ 0, 1, -2, 1 ] ],
  [ [ 1, 0, 0, 0 ], [ 5, 2, -4, 2 ] ],
  [ -1, 0, 1 ],             # local min. pol. is X^2-1
  [ 1, 2 ] ]                # pivots
gap> SpinMatVector(a,[0,1,0,0]);
[ [ [ 0, 1, 0, 0 ], [ 1, 0, -2, 1 ], [ 0, 0, 1, -1/2 ] ],
  [ [ 0, 1, 0, 0 ], [ -1, 0, 2, -1 ], [ 6, 3, -4, 2 ] ],
  [ 1, -1, -1, 1 ],         # local min. pol. is X^3-X^2-X^2+1
  [ 2, 1, 3 ] ]             # pivots
gap> SpinMatVector(a,[1,1,0,0]);
[ [ [ 1, 1, 0, 0 ], [ 0, 1, 1, -1/2 ] ],
  [ [ 1, 1, 0, 0 ], [ 4, 2, -2, 1 ] ],
  [ 1, -2, 1 ],             # local min. pol. is X^2-2X+1
  [ 1, 2 ] ]                # pivots
```

1.3.2 MaximalVectorMat

▷ `MaximalVectorMat(A)` (function)

'MaximalVectorMat' returns the minimal polynomial and a maximal vector of the matrix `A`, that is, a vector whose local minimal polynomial is that of `A`. This is done by repeatedly spinning up vectors until a maximal one is found. The exact algorithm is a combination of

- the minimal polynomial algorithm by Neunhoeffer-Praeger; see J. Comput. Math. 11, 2008 (http://doi.org/10.1112/S1461157000000590); and
- the method described by Bongartz (see https://arxiv.org/abs/1410.1683) for computing maximal vectors.

See also the article by Geck at https://doi.org/10.13001/ela.2020.5055.

Example

```
gap> A:=[ [  2,  2,  0,  1,  0,  2,  1 ],
>         [  0,  4,  0,  0,  0,  1,  0 ],
>         [  0,  1,  1,  0,  0,  1,  1 ],
>         [  0, -1,  0,  1,  0, -1,  0 ],
>         [  0, -7,  0,  0,  1, -5,  0 ],
>         [  0, -2,  0,  0,  0,  1,  0 ],
>         [  0, -1,  0,  0,  0, -1,  1 ] ];
# (Example taken from Ballester-Bolinches et al., Oper. Matrices 12, 2018.)
gap> MaximalVectorMat(A);
[ [ 1, -2, 1, 1, 0, 0, 1 ], x_1^4-7*x_1^3+17*x_1^2-17*x_1+6 ]
gap> v:=last[1];                    # maximal vector for A
gap> SpinMatVector(A,v)[3];         # check result: 3rd component =
[ 6, -17, 17, -7, 1 ]               #   coeffs of local minimal polynomial
```

In the following example, M_2 is the (challenging) test matrix from the paper by Neunhoeffer-Praeger:

Example

```
gap> LoadPackage("AtlasRep");; g:=AtlasGroup("B",1); M2:=g.1+g.2+g.1*g.2;
<matrix group of size 4154781481226426191177580544000000 with 2 generators>
<an immutable 4370x4370 matrix over GF2>
gap> SetInfoLevel(Infonofoma,1);
gap> MaximalVectorMat(M2);;time;
#I Degree of minimal polynomial is 2097
6725
gap> MinimalPolynomial(M2);;time;        # built-in GAP function
13415
gap> LoadPackage("cvec");                # package compact vectors
gap> MinimalPolynomial(CMat(M2));;time;
9721
```

1.3.3 FrobeniusNormalForm

▷ `FrobeniusNormalForm(A)` (function)

'FrobeniusNormalForm' returns the invariant factors of a matrix A and an invertible matrix P such that $P.A.P^{-1}$ is the Frobenius normal form of A. The algorithm first computes a maximal vector and an A-invariant complement following Jacob's construction (as described in matrix language in Geck, Electron. J. Linear Algebra 36, 2020, https://doi.org/10.13001/ela.2020.5055); then the algorithm continues recursively. It works for matrices over any field that is available in GAP. The output is a triple with

- 1st component = list of invariant factors;
- 2nd component = base change matrix P; and
- 3rd component = indices where the various blocks in the normal form begin.

Example

```
gap> A:=[ [  2,  2,  0,  1,  0,  2,  1 ],
>         [  0,  4,  0,  0,  0,  1,  0 ],
```

```
>         [  0,  1,  1,  0,  0,  1,  1 ],
>         [  0, -1,  0,  1,  0, -1,  0 ],
>         [  0, -7,  0,  0,  1, -5,  0 ],
>         [  0, -2,  0,  0,  0,  1,  0 ],
>         [  0, -1,  0,  0,  0, -1,  1 ] ];
gap> f:=FrobeniusNormalForm(A);
[ [ x_1^4-7*x_1^3+17*x_1^2-17*x_1+6, x_1^2-3*x_1+2, x_1-1 ],
                                  # f[1] = List of invariant factors
  [ [    1,   -2,    1,    1,    0,    0,    1 ],
    [    2,   -7,    1,    2,    0,   -1,    3 ],
    [    4,  -26,    1,    4,    0,   -8,    6 ],
    [    8,  -89,    1,    8,    0,  -35,   11 ],
    [ -1/2,   -2,    0,  1/2,    0,   -2, -3/2 ],
    [   -1,   -4,    0,    0,    0,   -4,   -2 ],
    [    0,  9/4,    0,   -3,    1,  5/4,  1/4 ] ],
                                  # f[2] = base change matrix P
  [ 1, 5, 7 ]  ]                  # f[3] = indices where the blocks begin
gap> PrintArray(f[2]*A*f[2]^-1);
[ [   0,   1,   0,   0,   0,   0,   0 ],
  [   0,   0,   1,   0,   0,   0,   0 ],
  [   0,   0,   0,   1,   0,   0,   0 ],
  [  -6,  17, -17,   7,   0,   0,   0 ],
  [   0,   0,   0,   0,   0,   1,   0 ],
  [   0,   0,   0,   0,  -2,   3,   0 ],
  [   0,   0,   0,   0,   0,   0,   1 ] ]
(This is the Frobenius normal form; there are 3 diagonal blocks,
 one of size 4, one of size 2 and one of size 1.)
```

You can also use 'CreateNormalForm(f[1]);' to produce the Frobenius normal form. (This function just builds the block diagonal matrix with diagonal blocks given by the companion matrices corresponding to the various invariant factors of `A`.)

1.3.4 JordanChevalleyDecMat

▷ `JordanChevalleyDecMat(A, f)` (function)

'JordanChevalleyDecMat' returns the unique pair of matrices D, N such that the matrix `A` is written as $A = D + N$, where N is a nilpotent matrix and D is a matrix that is diagonalisable (over some extension field of the default field of `A`), such that $D.N = N.D$; the argument `f` is a polynomial such that $f(A) = 0$ (e.g., the minimal polynomial of `A`). This is called the Jordan-Chevalley decomposition of `A`; the algorithm is based on the preprint at https://arxiv.org/abs/2205.05432. Note that this algorithm does not require the knowledge of the eigenvalues of `A`; it works over any perfect field that is available in GAP.

Example

```
gap> A:=[ [  6, -2,  6,  1,  1 ],
>         [  1, -1,  2,  1, -2 ],
>         [ -2,  0, -1,  0, -1 ],
>         [ -1,  0, -2,  2, -1 ],
>         [ -4,  4, -6, -2,  3 ] ];
gap> jc:=JordanChevalleyDecMat(A,MinimalPolynomial(A));
[ [ [  4,  0,  4, -1,  1 ],
```

```
     [  1,  0,  1,  1, -1 ],
     [ -1, -1,  0,  1, -1 ],
     [  0,  0, -2,  3,  0 ],
     [ -3,  2, -4, -1,  2 ] ],
   [ [  2, -2,  2,  2,  0 ],
     [  0, -1,  1,  0, -1 ],
     [ -1,  1, -1, -1,  0 ],
     [ -1,  0,  0, -1, -1 ],
     [ -1,  2, -2, -1,  1 ] ] ]
gap> MinimalPolynomial(jc[1]);
x_1^3-5*x_1^2+9*x_1-5
gap> Factors(last);
[ x_1-1, x_1^2-4*x_1+5 ]   # diagonalisable over quadratic extension of Q
gap> MinimalPolynomial(jc[2]);
x_1^2                      # nilpotent
```

If the input matrix is very large, then 'JordanChevalleyDecMatF(*A*);' may be more efficient; this function first computes the Frobenius normal form of *A* and then applies 'JordanChevalleyDecMat' to each diagonal block. (The result will be the same as that of 'JordanChevalleyDecMat(*A*);)'

1.3.5 Testnofoma

▷ `Testnofoma(`*lev*`)` (function)

'Testnofoma' runs a number of tests on the functions in this package; the argument *lev* is a positive integer specifying the InfoLevel.

1.4 Further documentation

The above functions, as well as a number of further auxiliary functions, are all contained and defined in the file 'pgk/nofoma-1.0/gap/nofoma.gi'; in that file, you can also find further inline documentation for the auxiliary functions.