Maxima by Example:
Ch. 5, Matrix Solution Methods *

Edwin (Ted) Woollett

March 30, 2016

Contents

*The code examples use **Maxima ver. 5.36.1** using **Windows 7**. This is a live document which will be updated when needed. Check **http://www.csulb.edu/~woollett/** for the latest version of these notes. Send comments and suggestions for improvements to **woollett@charter.net**

Feedback from readers is the best way for this series of notes to become more helpful to users of **Maxima**. *All* comments and suggestions for improvements will be appreciated and carefully considered.

1 Introduction and Quick Start

Maxima has many functions for defining and manipulating matrices, and our file **mbe5.mac** adds additional tools which are used in typical matrix solution problems in what follows. These new tools build on what is already available. In particular, these new tools build on matrix functions defined in two packages written by the Maxima developer, Barton Willis, namely **.../share/linearalgebra/linearalgebra.mac** and **.../share/contrib/diag.mac**.

As you learn about matrices, we recommend you always have a window with the Maxima help manual open. Inside the XMaxima interface, (which we always use) you can use the two key command **Alt+h** to display the Help menu list, and select the first item on the list "Maxima Manual", to see the html help manual, in which you can use either index views, or contents views, and can scroll up and down independently in the detailed righthand panel. We leave this Help manual open as a separate window as we work inside the XMaxima interface.

Our matrix package **mbe5.mac** sets **display2d:false** to allow more information per screen and also to reduce the number of pages in this pdf file. If you want to see a particular result in more transparent notation, (or just override this setting) use **display2d:true** after loading in **mbe5.mac**, or edit this file for your use.

Our **maxima-init.mac** file is in the folder **c:\Users\ted\maxima** on our Windows 7 computer, and in the folder **c:\Documents and Settings\ted\maxima** on our Windows XP computer. The file reads

```
maxima_userdir: "c:/work9" $
maxima_tempdir : "c:/work9"$
file_search_maxima : append(["c:/work9/###.{mac,mc}"],file_search_maxima )$
file_search_lisp : append(["c:/work9/###.lisp"],file_search_lisp )$
display2d:false$
ratsimp : false$
```

With this guidance for Maxima, we can use **load(mbe5)** to load the file **mbe5.mac** instead of the longer **load("mbe5.mac")**, or the even more explicit **load("c:/work9/mbe5.mac")**. Note that you should use forward slashes **/**, (rather than backward slashes \) for a **Maxima string** which defines a file path. Likewise, we can use **load(diag)** instead of **load("diag.mac")**, etc.

We have placed a **shortcut** to the XMaxima interface application, which is the executable **....\bin\xmaxima.exe**, in our work folder **c:\work9**, and we then launch XMaxima (our interface of choice) from **inside** our work folder.

To create a shortcut to **xmaxima.exe** using Windows 7 in folder **c:\work9**, right-click the desktop, choose "new", then "shortcut." Then choose "browse," and select the file **xmaxima.exe** in the Maxima program folder **...\bin** using Windows Explorer. The new shortcut will appear on your desktop, and you can rename it. Then rightclick the shortcut icon, and choose "copy". Then rightclick on folder **c:\work9** (in Windows Explorer) and choose "paste."

To create a shortcut to **xmaxima.exe** using Windows XP, right-click the file **...\bin\xmaxima.exe**, and from the drop-down menu select: "Create Shortcut." A separate shortcut named "Shortcut to xmaxima.exe" will appear next to **xmaxima.exe**, and you can then copy or move the shortcut to your work folder and rename it with a more convenient name.

The setting **fpprintprec:8$** is also made in **mbe5.mac**. If you want 16 significant figure printouts on your screen, use **fpprintprec:16$**.

There are many matrix functions available when you start Maxima and these functions can be used without loading in any additional packages. Here is the list (but several of these – see below – need separate packages loaded) from the Maxima html Help Manual (In the Help Manual index, type: matrixp, and then click on the category: Matrices, to see this list.)

```
Category: Matrices

addcol  addrow  adjoint  augcoefmatrix  cauchy_matrix  charpoly  coefmatrix
col  columnvector  covect  copymatrix  determinant  detout  diag  diagmatrix
doallmxops  domxexpt  domxmxops  domxnctimes  doscmxops  doscmxplus
echelon  eigen  ematrix  entermatrix  genmatrix  ident  invert  list_matrix_entries
lmxchar  matrix  matrix_element_add  matrix_element_mult
matrix_element_transpose  matrixmap  matrixp  mattrace  minor  ncharpoly
newdet  nonscalar  nonscalarp  permanent  rank  ratmx  row  scalarmatrixp
scalarp  setelmx  sparse  submatrix  tracematrix  transpose  triangularize  zeromatrix
```

In this Manual list, **diag** needs package **.../share/contrib/diag.mac** loaded, **eigen** is not found anywhere, **mattrace** and **ncharpoly** need package **.../share/matrix/nchrpl.mac** loaded, and **tracematrix** needs package **.../share/simplification/functs.mac** loaded.

1.1 Examples of Matrix Syntax

To show a few of the core functions at work with a purely symbolic matrix, let's define

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{1.1}$$

```
(%i1) A : matrix([a,b,c],[d,e,f],[g,h,i]);
(%o1) matrix([a,b,c],[d,e,f],[g,h,i])
(%i2) col(A,1);
(%o2) matrix([a],[d],[g])
(%i3) col(A,2);
(%o3) matrix([b],[e],[h])
(%i4) row(A,1);
(%o4) matrix([a,b,c])
(%i5) row(A,2);
(%o5) matrix([d,e,f])
(%i6) determinant(A);
(%o6) a*(e*i-f*h)-b*(d*i-f*g)+c*(d*h-e*g)
(%i7) A[1,1];
(%o7) a
(%i8) A[1,2];
(%o8) b
(%i9) A[2,1];
(%o9) d
(%i10) list_matrix_entries(A);
(%o10) [a,b,c,d,e,f,g,h,i]
(%i11) list_matrix_entries( col(A,1));
(%o11) [a,d,g]
(%i12) list_matrix_entries( row(A,1));
(%o12) [a,b,c]
(%i13) ident(3);
(%o13) matrix([1,0,0],[0,1,0],[0,0,1])
(%i14) invert(A);
(%o14) matrix([(e*i-f*h)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g)),
               (c*h-b*i)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g)),
               (b*f-c*e)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g))],
              [(f*g-d*i)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g)),
               (a*i-c*g)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g)),
               (c*d-a*f)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g))],
              [(d*h-e*g)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g)),
               (b*g-a*h)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g)),
               (a*e-b*d)/(a*(e*i-f*h)+b*(f*g-d*i)+c*(d*h-e*g))])
(%i15) matrixp(A);
(%o15) true
```

```
(%i16) transpose(A);
(%o16) matrix([a,d,g],[b,e,h],[c,f,i])
```

To show a few of these functions at work with a simple numerical matrix, let's define

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{1.2}$$

```
(%i17) A : matrix([1,2,3],[4,5,6],[7,8,9]);
(%o17) matrix([1,2,3],[4,5,6],[7,8,9])
(%i18) col(A,1);
(%o18) matrix([1],[4],[7])
(%i19) row(A,1);
(%o19) matrix([1,2,3])
(%i20) A[1,1];
(%o20) 1
(%i21) invert(A);
expt: undefined: 0 to a negative exponent.
 -- an error. To debug this try: debugmode(true);
(%i22) determinant(A);
(%o22) 0
(%i23) rank(A);
(%o23) 2
```

Matrix multiplication is indicated by a dot with white space separation **A . B**.

```
(%i24) B : matrix([1,1,1],[0,2,2],[0,3,3]);
(%o24) matrix([1,1,1],[0,2,2],[0,3,3])
(%i25) A . B;
(%o25) matrix([1,14,14],[4,32,32],[7,50,50])
(%i26) x : col(A,1);
(%o26) matrix([1],[4],[7])
(%i27) A . x;
(%o27) matrix([30],[66],[102])
```

Addition of matrices, multiplication and division of a matrix by scalars (and division of a matrix by a matrix), and mapping a function onto the elements of a matrix is illustrated next.

```
(%i28) A + B;
(%o28) matrix([2,3,4],[4,7,8],[7,11,12])
(%i29) 2*A;
(%o29) matrix([2,4,6],[8,10,12],[14,16,18])
(%i30) A^2;
(%o30) matrix([1,4,9],[16,25,36],[49,64,81])
(%i31) cos(A);
(%o31) matrix([cos(1),cos(2),cos(3)],[cos(4),cos(5),cos(6)],[cos(7),cos(8),
                                                            cos(9)])
(%i32) A/2;
(%o32) matrix([1/2,1,3/2],[2,5/2,3],[7/2,4,9/2])
(%i33) A/B;
expt: undefined: 0 to a negative exponent.
 -- an error. To debug this try: debugmode(true);
(%i34) B/A;
(%o34) matrix([1,1/2,1/3],[0,2/5,1/3],[0,3/8,1/3])
(%i35) A^^2;
(%o35) matrix([30,36,42],[66,81,96],[102,126,150])
```

Differentiation of a matrix with respect to a symbolic parameter:

```
(%i36) C : matrix([t,t^2],[t^3,t^4]);
(%o36) matrix([t,t^2],[t^3,t^4])
(%i37) diff(C, t);
(%o37) matrix([1,2*t],[3*t^2,4*t^3])
```

triangularize (A) returns the upper triangular form of the matrix **A**, as produced by Gaussian elimination. The return value of **triangularize** is the same as **echelon**, except that the leading nonzero coefficient in each row is not normalized to 1.

```
(%i38) A;
(%o38) matrix([1,2,3],[4,5,6],[7,8,9])
(%i39) triangularize(A);
(%o39) matrix([1,2,3],[0,-3,-6],[0,0,0])
(%i40) echelon(A);
(%o40) matrix([1,2,3],[0,1,2],[0,0,0])
```

1.2 Supplementary Matrix Functions

The code file **mbe5.mac** loads the matrix related supplementary code files **linearalgebra.mac**, and **diag.mac**, and the latter file loads **eigen.mac**. You can check what functions the individual Maxima packages define by restarting a new session of Maxima and load only one package at a time. Here is a list of functions available from **eigen.mac** alone:

```
(%i1) functions;
(%o1) []
(%i2) load(eigen);
(%o2) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/matrix/eigen.mac"
(%i3) functions;
(%o3) [innerproduct(x,y),unitvector(x),columnvector(x),
       gramschmidt(x,[myinnerproduct]),eigenvalues(mat),eigenvectors(mat),
       substvectk(l,n,exp),uniteigenvectors(M),similaritytransform(mat),
       conj(x),inprod(x,y),uvect(x),covect(x),gschmit(x,[f]),eivals(mat),
       eivects(mat),ueivects(mat),simtran(mat)]
```

Here is a list of functions defined by **linearalgebra.mac** alone:

```
(%i1) functions;
(%o1) []
(%i2) load(linearalgebra);
(%o2) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/linearalgebra/linearalgebra.mac"
(%i3) functions;
(%o3) [require_integer(i,pos,fn),require_symbol(x,pos,fn),
       request_rational_matrix(m,pos,fn),dotproduct(a,b),nullspace(m),
       nullity(m),orthogonal_complement([v]),
       locate_matrix_entry(m,r1,c1,r2,c2,fn,rel),columnspace(a),
       linalg_rank(m),rowswap(m,i,j),columnswap(m,i,j),rowop(m,i,j,theta),
       columnop(m,i,j,theta),hipow_gzero(e,x),good_pivot(e,x),
       ptriangularize(m,v),ptriangularize_with_proviso(m,v),
       column_reduce(m,i,x),mat_norm(m,p),mat_fullunblocker(m),
       mat_unblocker(m),mat_trace(m),kronecker_product(a,b),diag_matrix([d]),
       hilbert_matrix(n),hankel([q]),toeplitz([q]),polytocompanion(p,x),
       moore_penrose_pseudoinverse(m)]
```

And here is a list of functions defined when you load **diag.mac**.

```
(%i1) functions;
(%o1) []
(%i2) load(diag);
(%o2) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/contrib/diag.mac"
(%i3) functions;
(%o3) [innerproduct(x,y),unitvector(x),columnvector(x),
       gramschmidt(x,[myinnerproduct]),eigenvalues(mat),eigenvectors(mat),
       substvectk(l,n,exp),uniteigenvectors(M),similaritytransform(mat),
       conj(x),inprod(x,y),uvect(x),covect(x),gschmit(x,[f]),eivals(mat),
       eivects(mat),ueivects(mat),simtran(mat),diag_matrixify(expr),
       diag_zeropad_row(r,indent,width),diag(lst),JF(eival,n),
       diag_calculate_mult_partition(A,multiplicity,eival),jordan(A),
       diag_jordan_info_check(lst),minimalPoly(jordan_info),
       dispJordan(jordan_info),diag_sorted_list_histogram(list),
       diag_kernel_element(A),diag_general_jordan_chain(A,eival,degree),
       diag_find_li_chain(li_rows,chain_expr,variables),diag_mode_matrix(a,F),
       ModeMatrix(A,[jordan_info]),diag_taylor_coefficients(expr,var,maxpow),
       diag_taylor_expand_block(coeffs,var,eigenvalue,size),
       diag_mat_function_jordan(jordan,expr,var),mat_function(f,A)]
```

The functions in the first four lines and all of the fifth line except the last function **diag_matrixify** are defined in **eigen.mac**.

To display, via the same route, the functions defined by **mbe5.mac**, we temporarily comment out the load commands of **linearalgebra** and **diag**, restart Maxima, and load in **mbe5.mac**.

```
(%i1) functions;
(%o1) []
(%i2) load(mbe5);
(%o2) "c:/work9/mbe5.mac"
(%i3) functions;
(%o3) [gschmidt(Mm),inner_prod(Vv,Uu),normed(Vv),normal(Amatrix),
       diagp(Amatrix),modal_matrix(Aa),mJordan(Aa),jordan_chain(Aa,Ee),
       Mexp(Aa),eAt_diag(Amatrix),row_partial_L(Ab1,Nn1,Mm1),
       row_partial(Ab,Nn,Mm),to_solve(Ab1,xL1),solve_aug(Bc1,zL1),
       mcol_solve(known_col,unknown_col,uk_varL),echelon_test1(e,q),
       echelon_test2(e1,q),triangularize_test1(e,q),cvec(zzL),nullity(BB),
       mtrace(Mat),multiplicity(MA),vrank_num(BB,kk),vrank_max(Bb,Mult),
       vrank_numbers(Bb,Mm),vrank(Bb,Vv),chained(Vv1,Bb,Vv2),vL_rank(Bb,VL),
       mcombine(L),row_vector(alist),exp_taylor(%A,%n),eAt_CH(Am),
       FSS_sector(Amatrix,eival,multiplicity),
       FM_sector(Amatrix,eival,multiplicity),FSS(Amatrix),eAt_FSS(A),
       gauss_jordan(Aa,BbL,XxL),NZero_rows(Zz),NZero_rows1(Zz),rinverse(M),
       Jrow(k),dorow(k,r),Mexp_block(kk),expJ(kv,eival),eAt_jordan(Amatrix)]
```

1.3 Linear Inhomogeneous Set of Equations

To solve the system of three linear inhomogeneous equations

$$\begin{aligned} x_1 - 2\,x_2 + 3\,x_3 &= 7 \\ -x_1 + x_2 - 2\,x_3 &= -5 \\ 2\,x_1 - x_2 - x_3 &= 4 \end{aligned}$$

using matrix methods, we (mentally) write the equations in matrix form as

$$A\,\mathbf{x} = \mathbf{b}, \tag{1.3}$$

and find the solution using the inverse A^{-1} of A (if it exists)

$$\mathbf{x} = A^{-1}\,\mathbf{b}. \tag{1.4}$$

The matrix A is a 3×3 matrix, so if $\mathbf{rank}(A) < 3$, then the inverse does not exist, and there is no solution.

In interactive work with Maxima, it is easiest to pick out the elements of the matrix A using the rules for matrix multiplication, and define A using lists of row elements, as in

```
  A : matrix ([1,-2,3], [-1,1,-2], [2,-1,-1] );
```

An alternative approach, which may be useful if you are writing small automated programs, is to use the Maxima function **coefmatrix**, as is done below. We use the Maxima function **invert(A)** to produce the inverse matrix, and use the **mbe5.mac** function **cvec(alist)** to produce a matrix column vector. We also use **rank(A)** to find the (matrix) rank of A.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : coefmatrix([x1 - 2*x2 + 3*x3, -x1 + x2 -2*x3 ,
                      2*x1 - x2 - x3],[x1,x2,x3]);
(%o2) matrix([1,-2,3],[-1,1,-2],[2,-1,-1])
(%i3) bv : cvec([7,-5,4]);
(%o3) matrix([7],[-5],[4])
(%i4) rank(A);
(%o4) 3
(%i5) xv : invert(A) . bv;
(%o5) matrix([2],[-1],[1])
```

The code file **mbe5.mac** has the line **display2d:false$**, in order to allow more information per Maxima screen while working on a problem. You can temporarily override that setting to get a more normal look at the matrices:

```
(%i6) display2d:true$
(%i7) bv;
                                    [  7  ]
                                    [     ]
(%o7)                               [ - 5 ]
                                    [     ]
                                    [  4  ]
(%i8) A;
                               [  1   - 2   3  ]
                               [               ]
(%o8)                          [ - 1   1   - 2 ]
                               [               ]
                               [  2   - 1  - 1 ]
(%i9) xv;
                                    [  2  ]
                                    [     ]
(%o9)                               [ - 1 ]
                                    [     ]
                                    [  1  ]
(%i10) display2d:false$
```

As a (usually not needed) double check on the returned solution, you can use **is (equal (A . xv, bv))**.

```
(%i11) is (equal (A . xv, bv));
(%o11) true
```

and sometimes it helps to use **expand** before checking equality, as in

```
is (equal ( expand (A . xv) , bv));
```

1.4 Eigenvalues and Eigenvectors

Let A be a $n \times n$ matrix.

- An **eigenvalue** of A is a number λ such that $A\,\mathbf{v} = \lambda\,\mathbf{v}$ for some nonzero column vector $\mathbf{v}$.
- An **eigenvector** of A is a nonzero column vector $\mathbf{v}$ such that $A\,\mathbf{v} = \lambda\,\mathbf{v}$ for some number λ.
- The matrix A has n eigenvalues (including each according to its multiplicity).
- The sum of the n eigenvalues of A is the same as the **trace** of A, that is, the sum of the principle diagonal elements of A.
- The product of the n eigenvalues of A is the same as the determinant of A.

Example 1

Given the matrix

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \tag{1.5}$$

what can we say (using Maxima) about its eigenvalues and eigenvectors?

The Maxima functions **eigenvalues** and **jordan** (respectively defined in **eigen.mac** and **diag.mac**) are loaded by **mbe5.mac** and both give eigenvalue information.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ([0,1,1],[1,0,1],[1,1,0] );
(%o2) matrix([0,1,1],[1,0,1],[1,1,0])
(%i3) eigenvalues(A);
(%o3) [[2,-1],[1,2]]
(%i4) jordan(A);
(%o4) [[2,1],[-1,1,1]]
(%i5) [a1,a2] : map ('first, %);
(%o5) [2,-1]
(%i6) a3 : a2;
(%o6) -1
(%i7) eigenvectors(A);
(%o7) [[[2,-1],[1,2]],[[[1,1,1]],[[1,0,-1],[0,1,-1]]]]
```

The first sublist returned by **`eigenvalues(A)`** is a list of the eigenvalues of A, and the second sublist is a list of the corresponding (algebraic) multiplicities of each of these eigenvalues. Thus the eigenvalue $\lambda = -1$ has multiplicity $m = 2$.

The first sublist returned by **`jordan(A)`** is **`[2,1]`**, which says that the eigenvalue $\lambda = 2$ has multiplicity $m = 1$. The second sublist is **`[-1,1,1]`**, says that the eigenvalue $\lambda = -1$ has multiplicity $m = 2$ (the sum of the second and third elements), and that there exist two ordinary (rank 1) eigenvectors corresponding to the eigenvalue $\lambda = -1$ (see Sec. 5). We expect there to exist three ordinary rank 1 eigenvectors since **`rank(A) = length(A)`**.

The Maxima function **`eigenvectors(A)`** (from package **`eigen.mac`**) returns both the eigenvalue and multiplicity information (the first list) returned by **`eigenvalues(A)`** and also ordinary (rank 1) eigenvectors (the second list), corresponding to the eigenvalues. But note that the eigenvectors themselves are returned as lists of the vector elements, and these lists need to be converted to matrix column vectors. We use the **`mbe5.mac`** function **`cvec(alist)`** to create matrix columns from the lists of vector elements. Note that the second set of eigenvectors corresponds to the second eigenvalue $\lambda = -1$.

In Section 4.1 we discuss the concepts of $\mathbf{Null}B$, the nullspace of a matrix B, and the dimension ("nullity") of that nullspace. Here we specialize to the case of a **square** $n \times n$ matrix B. Then $\mathbf{Null}B$ is the set of all n-dimensional column vectors $\mathbf{v}$ such that $B\mathbf{v} = \mathbf{0}$. The **nullity** of the matrix B is number of basis vectors $\mathbf{v}$ needed to span the nullspace (the dimension of the nullspace). The "rank-nullity theorem" states that the sum of the **rank** of a matrix and the **nullity** of a matrix is equal to the number of columns of the matrix. For a square matrix, the number of columns equals the number of rows. In Maxima, **`length(B)`** returns of number of rows of the matrix **`B`**. Since we are assuming B is a square matrix, we can calculate the nullity of a matrix using **`nullity(B) : length(B) - rank(B)`**. The "nullity" of $B = A - \lambda I$ is the number of ordinary rank 1 vectors associated with the matrix A and the eigenvalue λ.

```
(%i8) B1 : A - a1*ident(3);
(%o8) matrix([-2,1,1],[1,-2,1],[1,1,-2])
(%i9) rank(B1);
(%o9) 2
(%i10) nullity(B1);
(%o10) 1
(%i11) B2 : A - a2*ident(3);
(%o11) matrix([1,1,1],[1,1,1],[1,1,1])
(%i12) rank(B2);
(%o12) 1
(%i13) nullity(B2);
(%o13) 2
```

We find it simpler to write down the eigenvectors by using the return value of the **`mbe5.mac`** function **`jordan_chain(A, eival)`**, using the syntax (if the eigenvalue **`eival`** has multiplicity **`m`**):

```
  [v1, v2, ..., vm] : map ('cvec, jordan_chain (A, eival))
```

Using this method, and comparing the vectors with those returned by **`eigenvectors(A)`**, we have

```
(%i14) [evals, evecs] : eigenvectors(A);
(%o14) [[[2,-1],[1,2]],[[[1,1,1]],[[1,0,-1],[0,1,-1]]]]
(%i15) [v1] : map('cvec, jordan_chain(A,2));
(%o15) [matrix([1],[1],[1])]
(%i16) v1;
(%o16) matrix([1],[1],[1])
(%i17) [v2, v3] : map('cvec, jordan_chain(A,-1));
(%o17) [matrix([1],[0],[-1]),matrix([0],[1],[-1])]
(%i18) v2;
(%o18) matrix([1],[0],[-1])
(%i19) v3;
(%o19) matrix([0],[1],[-1])
(%i20) is (equal (A . v1, 2*v1));
(%o20) true
(%i21) is (equal (A . v2, -v2));
(%o21) true
(%i22) is (equal (A . v3, -v3));
(%o22) true
```

For this matrix A, **`eigenvectors(A)`** has returned three ordinary (rank 1) eigenvectors which are linearly independent but not necessarily orthogonal. If we form the 3×3 matrix $M = [v1, v2, v3]$, then the set of three vectors are **linearly independent** if $\det(M) \neq 0$, which is equivalent to **`rank(M) = length(M)`** which implies that M^{-1} exists (see Sec. 3: Inverse of a Matrix). We use the **`mbe5.mac`** function **`mcombine(alist)`** to create **`M`**.

```
(%i293 M : mcombine([v1,v2,v3] );
(%o23) matrix([1,1,0],[1,0,1],[1,-1,-1])
(%i24) rank(M);
(%o24) 3
(%i25) determinant(M);
(%o25) 3
```

which confirms the linear independence of the three eigenvectors. Technically, A has three eigenvalues: $(2, -1, -1)$ but only two **distinct** eigenvalues. For the 3×3 matrix A, the **sum** of its 3 eigenvalues is the same as the trace of A and the **product** of its 3 eigenvalues is the same as the determinant of A. The **`mbe5.mac`** function **`mtrace(A)`** is used to calculate the trace of A, although adding up three zeros mentally is not so hard!

```
(%i26) 2 + (-1) + (-1);
(%o26) 0
(%i27) mtrace(A);
(%o27) 0
(%i28) 2*(-1)*(-1);
(%o28) 2
(%i29) determinant(A);
(%o29) 2
```

To check normalization and orthogonality of this set of three eigenvectors, we use the **`mbe5.mac`** function **`inner_prod(u,v)`**, which has the definition

```
inner_prod(u,v) := ratsimp(expand ( transpose (conjugate (u)) . v))$
```

`v1` is not normalized to 1, but is orthogonal to **`v2`** and **`v3`**.

```
(%i30) inner_prod(v1,v1);
(%o30) 3
(%i31) inner_prod(v1,v2);
(%o31) 0
(%i32) inner_prod(v1,v3);
(%o32) 0
```

Both **`v2`** and **`v3`** are not normalized to 1, and **`v2`** and **`v3`** are not orthogonal to each other.

```
(%i33) inner_prod(v2,v2);
(%o33) 2
(%i34) inner_prod(v3,v3);
(%o34) 2
(%i35) inner_prod(v2,v3);
(%o35) 1
```

To check normalization more quickly we can use the syntax (here, **f** is an undefined function):

```
(%i36) map ('f, [1,2,3],[1,2,3]);
(%o36) [f(1,1),f(2,2),f(3,3)]
```

To check for orthogonality more quickly, we can use the syntax

```
(%i37) map ('f, [1,2,3],[2,3,1]);
(%o37) [f(1,2),f(2,3),f(3,1)]
```

```
(%i38) map ('inner_prod,[v1,v2,v3],[v1,v2,v3] );
(%o38) [3,2,2]
(%i39) map ('inner_prod,[v1,v2,v3],[v2,v3,v1] );
(%o39) [0,1,0]
```

We use the **mbe5.mac** function **gschmidt(L)** which calls the **eigen.mac** package function **gramschmidt**, but adds **ratsimp (expand (...**, to finally arrive at three eigenvectors **v1,v2,v3** which are orthogonal (but still not normalized).

We leave the eigenvector **v1** alone, and use the Gram-Schmidt process to orthogonalize the pair **(v2, v3)**. We use the alias **lme**, defined in **mbe5.mac** for **list_matrix_entries**, which converts a column matrix into a list of values.

```
(%i40) map ('lme,[v2,v3]);
(%o40) [[1,0,-1],[0,1,-1]]
(%i41) gschmidt(%);
(%o41) [[1,0,-1],[-1/2,1,-1/2]]
(%i42) [v2,v3] : map ('cvec,%);
(%o42) [matrix([1],[0],[-1]),matrix([-1/2],[1],[-1/2])]
(%i43) map ('inner_prod,[v1,v2,v3],[v1,v2,v3] );
(%o43) [3,2,3/2]
(%i44) map ('inner_prod,[v1,v2,v3],[v2,v3,v1] );
(%o44) [0,0,0]
(%i45) is (equal (A . v1, 2*v1));
(%o45) true
(%i46) is (equal (A . v2, -v2));
(%o46) true
(%i47) is (equal (A . v3, -v3));
(%o47) true
```

Example 2

We use the same tools to look at the eigenvalues and eigenvectors of the matrix

$$A = \begin{bmatrix} 1 & -1 \\ 1 & 3 \end{bmatrix} \tag{1.6}$$

We find there is one distinct repeated eigenvalue $\lambda = 2$, but only one ordinary (rank 1) eigenvector.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ([1,-1],[1,3]);
(%o2) matrix([1,-1],[1,3])
(%i3) eigenvalues(A);
(%o3) [[2],[2]]
(%i4) jordan(A);
(%o4) [[2,2]]
(%i5) [a1] : map ('first, %);
(%o5) [2]
(%i6) a2 : a1;
(%o6) 2
(%i7) eigenvectors(A);
(%o7) [[[2],[2]],[[[1,-1]]]]
(%i8) [v1,v2] : map ('cvec, jordan_chain(A,a1));
(%o8) [matrix([-1],[1]),matrix([1],[0])]
(%i9) is (equal (A . v1, a1*v1));
(%o9) true
(%i10) is (equal (A . v2, a2*v2));
(%o10) false
```

The list returned by **jordan(A)** is **[[2,2]]** which indicates there is only one **distinct** eigenvalue $\lambda = 2$, with multiplicity $m = 2$, and there is one Jordan chain of generalized eigenvectors with maximum vector rank 2, so there will be one ordinary rank 1 eigenvector (here called **v1**) which satisfies the eigenvalue equation, and one rank 2 generalized eigenvector (here called **v2**), which does not satisfy the eigenvalue equation. (We discuss the concepts of the rank of a vector, generalized eigenvectors and Jordan chains in detail in Sec. 5.)

Defining the square matrix $B = A - \lambda I$, we use the **mbe5.mac** functions **vrank(B , v)** and **chained (va, B, vb)** to first return the vector ranks of the vectors defined via the return value of **jordan_chain(A,eival)** and then confirm that the two vectors are a "Jordan chain" of "generalized vectors."

The "nullity" of $B = A - \lambda I$ is the number of ordinary rank 1 vectors associated with the matrix A and the eigenvalue λ.

```
(%i11) B : A - a1*ident(2);
(%o11) matrix([-1,-1],[1,1])
(%i12) length(B);
(%o12) 2
(%i13) rank(B);
(%o13) 1
(%i14) nullity(B);
(%o14) 1
(%i15) vrank(B,v1);
(%o15) 1
(%i16) vrank(B,v2);
(%o16) 2
(%i17) chained (v1, B, v2);
(%o17) true
(%i18) expand (B . v1);
(%o18) matrix([0],[0])
(%i19) expand (B . v2);
(%o19) matrix([-1],[1])
(%i20) expand (B^^2 . v2);
(%o20) matrix([0],[0])
```

Example 3: Complex Eigenvalues and Eigenvectors

We use our tools to look at the eigenvalues and eigenvectors of the matrix

$$A = \begin{bmatrix} -1/2 & 1 \\ -1 & -1/2 \end{bmatrix} \tag{1.7}$$

We find there are two distinct complex eigenvalues $\lambda_1 = -(1 + 2\,i)/2$ and and $\lambda_2 = -(1 - 2\,i)/2$, which are complex conjugates of each other, and two ordinary (rank 1) eigenvectors, each of which have one element which is a complex number. Since the two eigenvalues are distinct, the corresponding eigenvectors are automatically orthogonal.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ([-1/2, 1], [-1,-1/2] );
(%o2) matrix([-1/2,1],[-1,-1/2])
(%i3) jn : jordan(A);
(%o3) [[-(2*%i+1)/2,1],[(2*%i-1)/2,1]]
(%i4) [a1, a2] : map ('first, jn);
(%o4) [-(2*%i+1)/2,(2*%i-1)/2]
(%i5) a1;
(%o5) -(2*%i+1)/2
(%i6) a2;
(%o6) (2*%i-1)/2
```

The return value of **jordan(A)** indicates each eigenvalue corresponds to an ordinary rank 1 eigenvector.

```
(%i7) [v1] : map ('cvec, jordan_chain(A,a1));
(%o7) [matrix([1],[-%i])]
(%i8) v1;
(%o8) matrix([1],[-%i])
```

```
(%i9) [v2] : map ('cvec, jordan_chain(A,a2));
(%o9) [matrix([1],[%i])]
(%i10) v2;
(%o10) matrix([1],[%i])
(%i11) is (equal (A . v1, a1*v1));
(%o11) true
(%i12) is (equal (A . v2, a2*v2));
(%o12) true
(%i13) inner_prod(v1,v1);
(%o13) 2
(%i14) inner_prod(v2,v2);
(%o14) 2
(%i15) inner_prod(v1,v2);
(%o15) 0
```

Looking at the **nullity** (see previous example) of $B = A - \lambda I$ for each eigenvalue, if $nullity(B) = dim\,\mathbf{Null}B = 1$, then one basis vector is needed to span $\mathbf{Null}B$, and there is one ordinary rank 1 eigenvector corresponding to the eigenvalue λ.

```
(%i16) B1 : A - a1*ident(2);
(%o16) matrix([(2*%i+1)/2-1/2,1],[-1,(2*%i+1)/2-1/2])
(%i17) rank(B1);
(%o17) 1
(%i18) length(B1);
(%o18) 2
(%i19) nullity(B1);
(%o19) 1
(%i20) vrank(B1,v1);
(%o20) 1
(%i21) expand (B1 . v1);
(%o21) matrix([0],[0])
(%i22) B2 : A - a2*ident(2);
(%o22) matrix([(-(2*%i-1)/2)-1/2,1],[-1,(-(2*%i-1)/2)-1/2])
(%i23) rank(B2);
(%o23) 1
(%i24) length(B2);
(%o24) 2
(%i25) nullity(B2);
(%o25) 1
(%i26) vrank(B2,v2);
(%o26) 1
(%i27) expand (B2 . v2);
(%o27) matrix([0],[0])
```

1.5 Initial Value Problem for a Set of Three First Order Linear ODE's

We can use Maxima's symbolic abilities to find the analytic solutions to the set of three first order linear ordinary differential equations (ode's)

$$\begin{aligned}
\frac{d}{dt}u_1(t) &= u_1(t) + 2\,u_2(t) - u_3(t) \\
\frac{d}{dt}u_2(t) &= -4\,u_1(t) - 7\,u_2(t) + 4\,u_3(t) \\
\frac{d}{dt}u_3(t) &= -4\,u_1(t) - 4\,u_2(t) + u_3(t)
\end{aligned}$$

which we can write in matrix form as

$$\frac{d}{dt}\mathbf{u}(t) = A\,\mathbf{u}(t), \tag{1.8}$$

in which the "coefficient matrix" A is

$$A = \begin{bmatrix} 1 & 2 & -1 \\ -4 & -7 & 4 \\ -4 & -4 & 1 \end{bmatrix}. \tag{1.9}$$

As we discuss later, if the initial values of the three dependent variables u_j are given by the column vector $\mathbf{u}(0)$, then the solution in matrix form is

$$\mathbf{u}(t) = e^{t\,A}\mathbf{u}(0), \tag{1.10}$$

in which $e^{t\,A}$ is the "exponential matrix" as a function of the matrix $t\,A$. We later present various ways to analytically evaluate this exponential matrix, including a function (in **`mbe5.mac`**) called **`eAt_jordan(A)`**, which we use here.

We assume

$$\mathbf{u}(0) = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}. \tag{1.11}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ([1,2,-1],[-4,-7,4],[-4,-4,1]);
(%o2) matrix([1,2,-1],[-4,-7,4],[-4,-4,1])
(%i3) jordan(A);
(%o3) [[-3,1],[-1,2]]
(%i4) eAt : eAt_jordan(A);
(%o4) matrix([2*t*%e^-t+%e^-t,%e^-t-%e^-(3*t),t*%e^-t-%e^-t+%e^-(3*t)],
             [-4*t*%e^-t,3*%e^-(3*t)-2*%e^-t,
              (-2*t*%e^-t)+3*%e^-t-3*%e^-(3*t)],
             [-4*t*%e^-t,2*%e^-(3*t)-2*%e^-t,
              (-2*t*%e^-t)+3*%e^-t-2*%e^-(3*t)])
(%i5) u0 : cvec ([1,-1,0]);
(%o5) matrix([1],[-1],[0])
(%i6) ut : eAt . u0;
(%o6) matrix([2*t*%e^-t+%e^-(3*t)],[(-4*t*%e^-t)+2*%e^-t-3*%e^-(3*t)],
             [(-4*t*%e^-t)+2*%e^-t-2*%e^-(3*t)])
(%i7) [ut1,ut2,ut3] : list_matrix_entries(ut);
(%o7) [2*t*%e^-t+%e^-(3*t),(-4*t*%e^-t)+2*%e^-t-3*%e^-(3*t),
       (-4*t*%e^-t)+2*%e^-t-2*%e^-(3*t)]
(%i8) ut1;
(%o8) 2*t*%e^-t+%e^-(3*t)
(%i9) ut2;
(%o9) (-4*t*%e^-t)+2*%e^-t-3*%e^-(3*t)
(%i10) ut3;
(%o10) (-4*t*%e^-t)+2*%e^-t-2*%e^-(3*t)
(%i11) ut1,t=0;
(%o11) 1
(%i12) ut2,t=0;
(%o12) -1
(%i13) ut3,t=0;
(%o13) 0
(%i14) plot2d([ut1,ut2,ut3],['t,0,1],[legend,false],
          [style,[lines,4]])$
```

which produces the plot

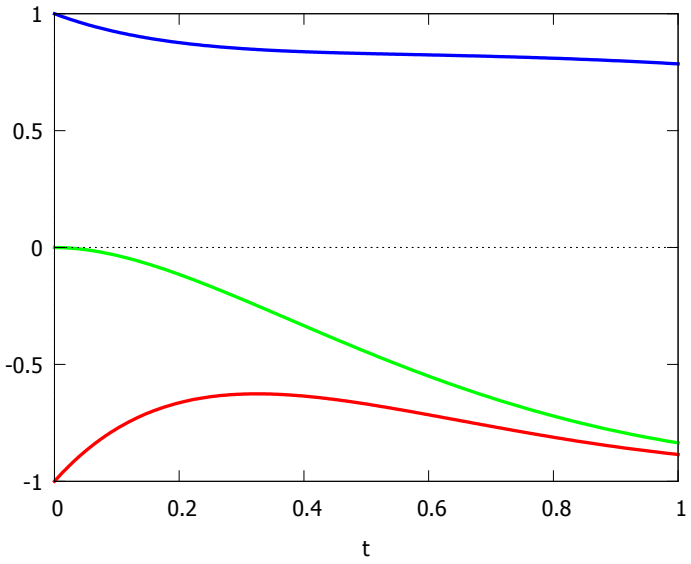

Figure 1: Case: $u_1(0) = 1$, $u_2(0) = -1$, $u_3(0) = 0$

One can use **`coefmatrix`** to derive the coefficient matrix **`A`** from the form of the right hand sides $A\,\mathbf{u}$ of the three differential equations.

```
(%i15) A;
(%o15) matrix([1,2,-1],[-4,-7,4],[-4,-4,1])
(%i16) A : coefmatrix([u1+2*u2-u3,-4*u1-7*u2+4*u3,-4*u1-4*u2+u3],[u1,u2,u3]);
(%o16) matrix([1,2,-1],[-4,-7,4],[-4,-4,1])
```

The analytic solution matching the given initial conditions is then

$$\begin{aligned}
\mathbf{u}_1(t) &= 2\,t\,e^{-t} + e^{-3\,t} \\
\mathbf{u}_2(t) &= (2 - 4\,t)\,e^{-t} - 3\,e^{-3\,t} \\
\mathbf{u}_3(t) &= (2 - 4\,t)\,e^{-t} - 2\,e^{-3\,t}
\end{aligned}$$

1.6 Matrix Notation Used Here

Column vectors are represented by bold face lower-case English letters $(\mathbf{x}, \mathbf{y}, \mathbf{z}, ...)$. The i'th component of the vector $\mathbf{x}$ is x_i. The row vector with the same elements as the column vector $\mathbf{x}$ is the **transpose** of $\mathbf{x}$, and is represented by $\mathbf{x}^{\mathrm{T}}$.

Matrices (other than vectors) are represented by upper-case English letters $(A, B, C, ...)$. Subscripts, as in $A_1, A_2, \ldots$ may be used to distinguish different matrices. *A vector is a matrix with only one column or only one row.*

Lower-case Greek letters, such as λ, α, γ will be used to represent real or complex numbers – never a vector or matrix.

The columns of the matrix A can be indicated by $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(n)}$ or more simply as $\mathbf{a}^1, \ldots, \mathbf{a}^n$

The components of a matrix A are a_{ij}. We can also write

$$A = (a_{ij}) \quad i = 1, \ldots, m \quad j = 1, \ldots, n \tag{1.12}$$

to indicate that A is the matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & \cdots & \cdot \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{1.13}$$

If A has the same number, n, of rows and columns, we write

$$A = (a_{ij}) \quad i, j = 1, \ldots, n \tag{1.14}$$

The rows of a matrix are horizontal, for example the third row

$$[a_{31}, \ldots, a_{3n}]. \tag{1.15}$$

The columns of a matrix are vertical, for example the seventh column

$$\begin{bmatrix} a_{17} \\ \cdot \\ \cdot \\ \cdot \\ a_{m7} \end{bmatrix} \tag{1.16}$$

To save space, we can write instead col$(a_{17}, \cdots, a_{m7})$ for the above column.

The letter $\mathbf{e}$ with a superscript is reserved for cartesian unit vectors, such that $\mathbf{e}^j$ is a column vector with its jth component equal to 1 and with all other components equal to 0.

A determinant is a single number computed from a square matrix ($m = n$). The simplest case is the determinant of a 2×2 matrix

$$\det \begin{bmatrix} \alpha & \delta \\ \gamma & \beta \end{bmatrix} = \alpha\beta - \gamma\delta = \begin{vmatrix} \alpha & \delta \\ \gamma & \beta \end{vmatrix} \tag{1.17}$$

The *transpose* A^{T} of a matrix A is formed by interchanging corresponding rows and columns. Thus row 1 of A becomes column 1 of A^{T}, row 2 of A becomes column 2 of A^{T}, etc. An example for a 2×2 matrix is

$$\begin{bmatrix} \alpha & \delta \\ \gamma & \beta \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \alpha & \gamma \\ \delta & \beta \end{bmatrix} \tag{1.18}$$

Then if $B = A^{\mathrm{T}}$ then $b_{ij} = a_{ji}$.

If $A^{\mathrm{T}} = A$, which implies $a_{ji} = a_{ij}$, then we say that A is a **symmetric matrix**. If $A^{\mathrm{T}} = -A$, which implies $a_{ji} = -a_{ij}$, then we say that A is an **antisymmetric matrix**. The **Hermitian conjugate** of a matrix A is indicated by either $A^{\dagger}$ or A^{H}, and if $B = A^{\dagger}$, then $b_{ij} = \overline{a}_{ji}$, in which the overbar indicates complex conjugation. In other words,

$$A^{\dagger} = \left(\overline{A}\right)^{\mathrm{T}} = \overline{(A^{\mathrm{T}})}. \tag{1.19}$$

If $A^{\dagger} = A$, then we say that A is a **Hermitian matrix** (self-adjoint). A matrix which is both real and symmetric is a special case of a Hermitian matrix.

Matrix Multiplication

$A\,\mathbf{x}$ for a Non-Square Matrix A

Let A be a $m \times n$ matrix (m rows and n columns) with components a_{ij}, with $(i = 1, 2, \ldots, m)$ and $(j = 1, 2, \ldots, n)$. If $\mathbf{x}$ is a n-component column vector with components x_j, then the number of rows of $A\,\mathbf{x}$ equals the number of rows of A, and the m elements of the column vector $A\,\mathbf{x}$ are given by

$$(A\,\mathbf{x})_i = \sum_{j=1}^{n} a_{ij}\, x_j, \quad (i = 1, 2, \ldots, m). \tag{1.20}$$

For example

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 + 2\,x_2 + 3\,x_3 \\ 4\,x_1 + 5\,x_2 + 6\,x_3 \end{bmatrix}. \tag{1.21}$$

$A\,\mathbf{x}$ for a Square Matrix A

Assume A is a square $n \times n$ matrix with matrix elements a_{ij}. The j'th column of A is $\mathbf{a}^{(j)}$, and $a_{ij} = a_i^{(j)}$. Let $\mathbf{x}$ be a n-component column vector with components x_j. The product $A\,\mathbf{x}$ is a column vector which is a linear superposition of the columns of A.

$$A\,\mathbf{x} = \sum_{j=1}^{n} \mathbf{a}^{(j)}\,x_j \tag{1.22}$$

and the i'th component is

$$(A\,\mathbf{x})_i = \sum_{j=1}^{n} a_i^{(j)}\,x_j = \sum_{j=1}^{n} a_{ij}\,x_j. \tag{1.23}$$

The product $C = A\,B$ of two $n \times n$ matrices A and B has matrix components

$$C_{ij} = \sum_k a_{ik}\,b_{kj}. \tag{1.24}$$

The product of two diagonal matrices A and B is again diagonal. Let δ_{ij} denote the Kronecker delta, which is equal to 1 if $i = j$ and otherwise is equal to 0. Then if $A_{ij} = \alpha_i\,\delta_{ij}$, and $B_{ij} = \beta_i\,\delta_{ij}$, then with $C = A\,B$,

$$C_{ij} = \sum_k A_{ik}\,B_{kj} = \sum_k (\alpha_i\,\delta_{ik})\,(\beta_k\,\delta_{kj}) = \alpha_i\,\beta_i\,\sum_k \delta_{ik}\,\delta_{kj} = \alpha_i\,\beta_i\,\delta_{ij} \tag{1.25}$$

Vector Multiplication

If $\mathbf{x}$ and $\mathbf{y}$ are column vectors with the same number *n* of components, then one kind of product is

$$\mathbf{x}^{\mathrm{T}}\,\mathbf{y} = \sum_{i=1}^{n} x_i\,y_i = \mathbf{y}^{\mathrm{T}}\,\mathbf{x}. \tag{1.26}$$

If $\mathbf{x}$ and $\mathbf{y}$ are linearly dependent, then one is a multiple of the other. We also define a product called the **scalar product** or **inner product**, defined by

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \overline{x_i}\,y_i = \overline{\mathbf{x}^{\mathrm{T}}}\,\mathbf{y} \tag{1.27}$$

which is, in the general case, a complex number.

Here is a small example, using the **mbe5.mac** function **inner_prod(u,v)**, which has a definition (both input matrix column vectors are assumed to have the same number of elements) :

```
inner_prod (uA,vA) := ( conjugate( transpose(uA)) . vA )$
```

We first show that **conjugate(%i*a)** just changes **%i** to **-%i**, since **a** is assumed real unless you specifically declare it complex (as in **declare (a, complex)**). We then define a three component column vector **v** which has two complex elements, and show what **conjugate** does to **v**. We then define another three component column vector **u** which has one complex element, and calculate the inner product (u, v), using **inner_prod**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) conjugate(%i*a);
(%o2) -%i*a
```

```
(%i3) declare(b,complex);
(%o3) done
(%i4) conjugate(%i*b);
(%o4) -%i*conjugate(b)
(%i5) v : cvec ([%i,%i*2,3]);
(%o5) matrix([%i],[2*%i],[3])
(%i6) conjugate(v);
(%o6) matrix([-%i],[-2*%i],[3])
(%i7) u : cvec ([4,5,%i*6]);
(%o7) matrix([4],[5],[6*%i])
(%i8) inner_prod(u,v);
(%o8) -4*%i
```

If $(\mathbf{x}, \mathbf{y}) = 0$, then the two vectors $\mathbf{x}$ and $\mathbf{y}$ are said to be **orthogonal**, and are automatically linearly independent. The inner product (scalar product) of a vector with itself is a real non-negative number whose square root is called the **length** of the vector.

$$(\mathbf{x}, \mathbf{x}) = \sum_{i=1}^{n} \overline{x_i}\, x_i = \sum_{i=1}^{n} |x_i|^2. \tag{1.28}$$

Here is an example using **v** defined above:

```
(%i9) inner_prod (v,v);
(%o9) 14
```

1.7 References

1. nineteen dubious ways to compute the Exponential of a matrix, twenty-five years later:
   http://www.cs.cornell.edu/cv/researchpdf/19ways+.pdf

2. Chapter 7 of: Elementary Differential Equations and Boundary Value Problems,
   William E. Boyce and Richard C. DiPrima, John Wiley, 5'th Edition, 1992.

3. Matrix Operations, Richard Bronson, Schaum's Outline Series, McGraw-Hill, 2nd. ed., 2011

4. https://en.wikipedia.org/wiki/Matrix_(mathematics)
   https://en.wikipedia.org/wiki/Generalized_eigenvector
   https://en.wikipedia.org/wiki/Matrix_function
   https://en.wikipedia.org/wiki/Eigenvalue_algorithm
   https://en.wikipedia.org/wiki/Gram-Schmidt_process

5. EE 448/528 Analytical and Computational Methods, John L. Stensby,
       http://www.ece.uah.edu/~stensby/
       http://www.ece.uah.edu/courses/ee448/
   ch. 10: functions of a matrix:
       http://www.ece.uah.edu/courses/ee448/chapter10.pdf

6. Facts About Eigenvalues, David Butler,
    https://www.adelaide.edu.au/mathslearning/play/seminars/evalue-magic-tricks-handout.pdf

7. Matrix Theory, Joel N. Franklin, Prentice-Hall, 1968

8. Linear Algebra Done Right, Sheldon Axler, Springer, 2nd. ed, 1997

2 Row-Echelon Form and Rank of a Matrix

Row-Echelon Form

Quoting Bronson, Ch. 1,

> A **zero row** in a matrix is a row whose elements are all zero. A **nonzero row** is one that contains at least one nonzero element. A matrix is a **zero matrix**, denoted Z (or Z_{mn} to emphasize m rows and n columns), if it contains only zero rows.
>
> A matrix is in **row-echelon** form if it satisfies four conditions:
>
> 1. All nonzero rows precede (that is, appear above) zero rows when both types are contained in the matrix.
> 2. The first (leftmost) nonzero element of each nonzero row is unity.
> 3. When the first nonzero element of a row appears in column c, then all elements in column c in succeeding rows are zero.
> 4. The first nonzero element of any nonzero row appears in a later column (further to the right) than the first nonzero element of any preceding row.

If a matrix A is in row-echelon form, then the non-zero rows of this form are linearly independent.

Elementary Row and Column Operations

Quoting Bronson (pg. 3)

> There are three **elementary row operations** which may be used to transform a matrix into row-echelon form:
>
> 1. Interchange any two rows.
> 2. Multiply the elements of any row by a nonzero scalar.
> 3. Add to any row, element by element, a scalar times the corresponding elements of another row.
>
> Three **elementary column operations** are defined analogously.

Rank

The **rank** (or **row-rank**) of a matrix is the number of **nonzero rows** in the matrix after it has been transformed to row-echelon form via elementary row operations.

The Maxima function **`echelon(A)`** can be used to transform the matrix **`A`** into a (not usually unique) row-echelon form. The core Maxima function **`rank(A)`** can be used to calculate the rank of **`A`**. Here is a simple example using the 2×3 matrix (2 rows and 3 columns).

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 9 & 7 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -5 \end{bmatrix} \tag{2.1}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,2,3],[4,9,7]);
(%o2) matrix([1,2,3],[4,9,7])
(%i3) echelon(A);
(%o3) matrix([1,2,3],[0,1,-5])
(%i4) rank(A);
(%o4) 2
```

Quoting Bronson, p. 9,

> The row-echelon form is not unique if a matrix has rank 2 or greater.

In the example above, which has rank 2, we could add the second row to the first row of the row-echelon form returned, and the result would still be in row-echelon form.

The rank of a matrix equals the number of linearly independent columns in the matrix. The rank of a matrix equals the number of linearly independent rows in the matrix. For a square matrix, the maximum rank is the number of columns (which is the same as the number of rows). A matrix which has the maximum rank is called a "full rank matrix." The product of two full rank matrices is itself full rank. The inverse of a square $n \times n$ matrix exists (and is unique) if and only if the rank of the matrix is n.

Row-Echelon Transformation Algorithm

Quoting Bronson, Ch. 1,

> An algorithm for using elementary row operations to transform a matrix into row-echelon form is as follows:
>
> 1. Let R denote the work row, and initialize $R = 1$ (so the top row is the first work row).
> 2. Find the first column containing a nonzero element in either row R or any succeeding row. If no such column exists, stop; the transformation is complete. Otherwise let C denote this column.
> 3. Beginning with row R and continuing through successive rows, locate the first row having a nonzero element in column C. If this row is not row R, interchange it with row R. Row R will now have a nonzero element in column C. This element is called the **pivot**; let P denote its value.
> 4. If $P \neq 1$, multiply the elements of row R by $1/P$; otherwise continue.
> 5. Search all rows following R for one having a nonzero element in column C. If no such row exists, go to step 8; otherwise designate that row as row N, and the value of the nonzero element in row N and column C as V.
> 6. Add to the elements of row N the scalar $-V$ times the corresponding elements of row R.
> 7. Return to step 5.
> 8. Increase R by 1. If this new value of R is larger than the number of rows in the matrix, stop; the transformation is complete. Otherwise, return to step 2.

In addition to the example here, there are more examples in the section on matrix inversion.

Example 1

Follow the above steps to transform

$$B = \begin{bmatrix} 0 & 1 & 4 \\ 1 & 2 & 3 \end{bmatrix} \tag{2.2}$$

With $R = 1$ (step 1) and $C = 1$ (step 2), we apply step 3 and interchange rows 1 and 2, obtaining

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \end{bmatrix} \tag{2.3}$$

which is in row-echelon form.

Example 2

Follow the above steps to transform

$$B = \begin{bmatrix} 0 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

With $R = 1$ (step 1) and $C = 2$ (step 2), $P = 2$ (step 3), we multiply the elements of row 1 by $1/P = 1/2$ to get a row-echelon form

$$\begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

which is in row-echelon form.

Example 3

Follow the above steps to transform

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 9 & 7 \end{bmatrix} \tag{2.6}$$

With $R = 1$ (step 1) and $C = 1$ (step 2), $N = 2$ and $V = 4$ (step 5), we add to each element of row 2 (-4) times the corresponding element in row 1, to get a row-echelon form

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -5 \end{bmatrix} \tag{2.7}$$

3 Inverse of a Matrix

Quoting Bronson, Ch. 4,

> Matrix B is the **inverse** of a square matrix A if
>
> $$A\,B = B\,A = I. \tag{3.1}$$
>
> For both products to be defined simultaneously, A and B must be square matrices of the same order.
>
> A square matrix is said to be **singular** if it does not have an inverse; a matrix that has an inverse is called **nonsingular** or **invertible**. The inverse of A, when it exists, is denoted as A^{-1}.
>
> **PROPERTIES OF THE INVERSE**
>
> 1. The inverse of a nonsingular matrix is unique.
> 2. If A is nonsingular, then $\left(A^{-1}\right)^{-1} = A$.
> 3. If A and B are nonsingular, then $(A\,B)^{-1} = B^{-1}\,A^{-1}$.
> 4. If A is nonsingular, then so too is A^{T}. Further, $(A^{\mathrm{T}})^{-1} = \left(A^{-1}\right)^{\mathrm{T}}$.

Let A be an $n \times n$ matrix. If any of the following equivalent conditions hold, then A does not have an inverse:

1. $A\,\mathbf{x} = \mathbf{0}$ has a nontrivial solution.
2. The columns of A are linearly dependent.
3. The rank of A is less than n.
4. A is not row equivalent to the identity matrix.
5. $\mathbf{det}(A) = 0$

The meaning of the fourth statement above is found in the algorithm of the derivation of the inverse by elementary row operations discussed below.

Let A be an $n \times n$ matrix with real or complex entries. Then the following statements are equivalent.

1. A^{-1} exists.
2. The only solution to $A\,\mathbf{x} = \mathbf{0}$ is the "trivial solution" $\mathbf{x} = \mathbf{0}$.
3. The columns of A are linearly independent.
4. $\mathbf{rank}(A) = n$

5. A is row equivalent to I. ($A \Leftrightarrow I$.)

6. $\mathbf{det}(A) \neq 0$.

A consequence of the fifth statement above, $A \Leftrightarrow I$, is that there exist row operations that take A to I. If we apply these row operations to the augmented matrix $[A|I]$, then we will produce $[I|B]$. The matrix B satisfies $A\,B = I$. But to be an inverse, it must also satisfy $B\,A = I$. One can show that, by virtue of the row reduction method of finding B, and its reversibility, B satisfies $B\,A = I$ automatically, and thus is the unique inverse of A sought.

Thus A^{-1} exists if and only if $A \Leftrightarrow I$.

3.1 Inversion Using Maxima core functions

In Maxima, we can use either of the two equivalent forms: **`invert(A)`** or **`A^^-1`** (or **`A^^(-1)`**) to determine the inverse of A, if it exists.

Example 1

We use Maxima to find the inverse of

$$A = \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix}. \tag{3.2}$$

```
(%i1) A : matrix([5,3],[2,1]);
(%o1) matrix([5,3],[2,1])
(%i2) B : invert(A);
(%o2) matrix([-1,3],[2,-5])
(%i3) B . A;
(%o3) matrix([1,0],[0,1])
(%i4) A . B;
(%o4) matrix([1,0],[0,1])
(%i5) A^^-1;
(%o5) matrix([-1,3],[2,-5])
(%i6) A^^(-1);
(%o6) matrix([-1,3],[2,-5])
```

which says that

$$A^{-1} = \begin{bmatrix} -1 & 3 \\ 2 & -5 \end{bmatrix}. \tag{3.3}$$

Note that

```
(%i7) rank(A);
(%o7) 2
(%i8) echelon(A);
(%o8) matrix([1,3/5],[0,1])
(%i9) length(A);
(%o9) 2
```

and $rank(A)$ is equal to the number of rows of A and there are no zero rows in the row-echelon form of A.

Example 2

We use Maxima to seek the inverse of

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}. \tag{3.4}$$

```
(%i10) A : matrix([1,2,3],[4,5,6],[7,8,9]);
(%o10) matrix([1,2,3],[4,5,6],[7,8,9])
```

```
(%i11) B : invert(A);
expt: undefined: 0 to a negative exponent.
 -- an error. To debug this try: debugmode(true);
(%i12) echelon(A);
(%o12) matrix([1,2,3],[0,1,2],[0,0,0])
(%i13) rank(A);
(%o13) 2
(%i14) length(A);
(%o14) 3
```

Note that the singular matrix A has a rank which is less than the number of rows, and the row-echelon form has one zero row. A^{-1} does not exist.

3.2 Inversion Using Elementary Row Operations

From Bronson, Ch. 4,

> **CALCULATING INVERSES USING ELEMENTARY ROW OPERATIONS**
>
> Inverses may be found through the use of elementary row operations. This procedure not only yields the inverse when it exists, but also indicates when the inverse does not exist. An algorithm for finding the inverse of a matrix A is as follows:
>
> 1. Form the partitioned matrix $[A\,|I\,]$, where I is the identity matrix having the same order as A.
> 2. Using elementary row operations, transform A into row-echelon form, applying each row operation to the entire partioned matrix formed in Step 1. Denote the result as $[C\,|D\,]$, where C is in row-echelon form.
> 3. If C has a zero row, stop; the original matrix A is singular and does not have an inverse. Otherwise continue; the original matrix is invertible.
> 4. Beginning with the last column of C and progressing backward iteratively through the second column, use elementary row operation (3) to transform all elements **above** the diagonal of C to zero. Apply each operation, however, to the entire matrix $[C\,|D\,]$. Denote the result as $[I\,|B\,]$. The matrix B is the inverse of the original matrix A.
>
> If exact arithmetic is not used in Step 2, then a pivoting strategy should be employed. No pivoting strategy is used in Step 4; the pivot is always one of the unity elements on the diagonal of C. Interchanging any rows after Step 2 has been completed will undo the work of that step and, therefore, is not allowed.

A homemade function **`rinverse(A)`** (available in **`mbe5.mac`**) implements this algorithm in the Maxima language, using **`echelon`** as the first step in the row reduction (see code below). In the **`rinverse`** code we use the function **`mat_unblocker`**, defined in the **`linearalgebra.mac`** package of functions. If this package is not already loaded, the first time use of **`rinverse`** will take a little longer because the package automatically loads when we first use its function **`mat_unblocker`**.

In the examples below, the row reduction is done "by hand" first, as further examples of using the elementary row operations.

Example 1

We use the elementary row operation algorithm to find the inverse of

$$A = \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix}. \tag{3.5}$$

We first form the partitioned matrix $[A\,|I\,]$.

$$[A\,|I\,] = \left[\begin{array}{cc|cc} 5 & 3 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{array}\right]. \tag{3.6}$$

With $R = 1$, $C = 1$, and $P = 5$, we first multiply row 1 by $(1/5)$ to get

$$\left[\begin{array}{cc|cc} 1 & 3/5 & 1/5 & 0 \\ 2 & 1 & 0 & 1 \end{array}\right]. \tag{3.7}$$

With $R = 1$, $C = 1$, $N = 2$, and $V = 2$, we replace row 2 by the sum of row 2 and (-2) times row 1 to get

$$\left[\begin{array}{cc|cc} 1 & 3/5 & 1/5 & 0 \\ 0 & -1/5 & -2/5 & 1 \end{array}\right]. \tag{3.8}$$

With $R = 2$, $C = 2$, and $P = -1/5$, we multiply row 2 by (-5) to get

$$[\,C\,|D\,] = \left[\begin{array}{cc|cc} 1 & 3/5 & 1/5 & 0 \\ 0 & 1 & 2 & -5 \end{array}\right]. \tag{3.9}$$

C does not have a zero row, so A is invertible, and we continue to find the inverse. We replace row 1 by the sum of row 1 plus $(-3/5)$ times row 2, to get

$$[\,I\,|B\,] = \left[\begin{array}{cc|cc} 1 & 0 & -1 & 3 \\ 0 & 1 & 2 & -5 \end{array}\right]. \tag{3.10}$$

This determines the inverse of A as

$$A^{-1} = B = \begin{bmatrix} -1 & 3 \\ 2 & -5 \end{bmatrix}, \tag{3.11}$$

which agrees with our Maxima calculation above.

To employ Maxima, we use both **`invert`** and then **`rinverse`**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([5,3],[2,1]);
(%o2) matrix([5,3],[2,1])
(%i3) rank(A);
(%o3) 2
(%i4) determinant(A);
(%o4) -1
(%i5) B1 : invert(A);
(%o5) matrix([-1,3],[2,-5])
(%i6) A . B1;
(%o6) matrix([1,0],[0,1])
(%i7) B1 . A;
(%o7) matrix([1,0],[0,1])
(%i8) B2 : rinverse(A);
(%o8) matrix([-1,3],[2,-5])
(%i9) A . B2;
(%o9) matrix([1,0],[0,1])
(%i10) B2 . A;
(%o10) matrix([1,0],[0,1])
```

Example 2

We use the elementary row operation algorithm to find the inverse of

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}. \tag{3.12}$$

We first form the partitioned matrix $[A\,|I\,]$.

$$[A\,|I\,] = \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 4 & 5 & 6 & 0 & 1 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right]. \tag{3.13}$$

With $R = 1$, $C = 1$, $N = 2$, and $V = 4$, we replace row 2 by the sum of row 2 and (-4) times row 1 to get

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & -3 & -6 & -4 & 1 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right]. \tag{3.14}$$

With $R = 1$, $C = 1$, $N = 3$, and $V = 7$, we replace row 3 by the sum of row 3 and (-7) times row 1 to get

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & -3 & -6 & -4 & 1 & 0 \\ 0 & -6 & -12 & -7 & 0 & 1 \end{array}\right]. \tag{3.15}$$

We increase R by 1, and with $R = 2$, $C = 2$, and $P = -3$, we replace row 2 by $(-1/3)$ times row 2 to get

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 2 & 4/3 & -1/3 & 0 \\ 0 & -6 & -12 & -7 & 0 & 1 \end{array}\right]. \tag{3.16}$$

With $R = 2$, $C = 2$, $N = 3$, and $V = -6$ we replace row 3 by the sum of row 3 and (6) times row 2 to get

$$[C\,|D\,] = \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 2 & 4/3 & -1/3 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \end{array}\right]. \tag{3.17}$$

The left side of this partitioned matrix is in row-echelon form. Since its third row is a zero row, the original matrix A does not have an inverse, which agrees with our Maxima calculations.

```
(%i11) A : matrix([1,2,3],[4,5,6],[7,8,9]);
(%o11) matrix([1,2,3],[4,5,6],[7,8,9])
(%i12) rank(A);
(%o12) 2
(%i13) determinant(A);
(%o13) 0
(%i14) rinverse(A);
 the inverse does not exist
(%o14) done
```

Example 3

We use the elementary row operation algorithm to find the inverse of

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 5 & 1 & -1 \\ 2 & -3 & -3 \end{bmatrix}. \tag{3.18}$$

We first form the partitioned matrix $[A\,|I\,]$.

$$[A\,|I\,] = \left[\begin{array}{ccc|ccc} 0 & 1 & 1 & 1 & 0 & 0 \\ 5 & 1 & -1 & 0 & 1 & 0 \\ 2 & -3 & -3 & 0 & 0 & 1 \end{array}\right]. \tag{3.19}$$

We interchange $R1$ and $R2$ to get

$$\left[\begin{array}{ccc|ccc} 5 & 1 & -1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 2 & -3 & -3 & 0 & 0 & 1 \end{array}\right]. \tag{3.20}$$

With $R = 1$, $C = 1$, and $P = 5$, replace $R1$ by $(1/5)\,R1$ to get

$$\left[\begin{array}{ccc|ccc} 1 & 1/5 & -1/5 & 0 & 1/5 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 2 & -3 & -3 & 0 & 0 & 1 \end{array}\right]. \tag{3.21}$$

With $R = 1$, $C = 1$, $N = 3$, and $V = 2$, we replace $R3$ with the sum of $R3$ and $(-2)\,R1$ to get

$$\left[\begin{array}{ccc|ccc} 1 & 1/5 & -1/5 & 0 & 1/5 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & -17/5 & -13/5 & 0 & -2/5 & 1 \end{array}\right]. \tag{3.22}$$

With $R = 2$, $C = 2$, $N = 3$, and $V = -17/5$, we replace $R3$ with the sum of $R3$ and $(17/5)\,R2$ to get

$$\left[\begin{array}{ccc|ccc} 1 & 1/5 & -1/5 & 0 & 1/5 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 4/5 & 17/5 & -2/5 & 1 \end{array}\right]. \tag{3.23}$$

With $R = 3$, $C = 3$, and $P = 4/5$, we replace $R3$ by $(5/4)\,R3$ to get

$$[\,C\,|D\,] = \left[\begin{array}{ccc|ccc} 1 & 1/5 & -1/5 & 0 & 1/5 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 17/4 & -2/4 & 5/4 \end{array}\right]. \tag{3.24}$$

The left side of this partitioned matrix is now in row-echelon form and there are no zero rows, so the inverse of the original matrix A exists. We start on column 3 and replace $R2$ by the sum of $R2$ and $(-1)\,R3$ to get

$$\left[\begin{array}{ccc|ccc} 1 & 1/5 & -1/5 & 0 & 1/5 & 0 \\ 0 & 1 & 0 & -13/4 & 2/4 & -5/4 \\ 0 & 0 & 1 & 17/4 & -2/4 & 5/4 \end{array}\right]. \tag{3.25}$$

We next replace $R1$ by the sum of $R1$ and $(1/5)\,R3$ to get

$$\left[\begin{array}{ccc|ccc} 1 & 1/5 & 0 & 17/20 & 2/20 & 5/20 \\ 0 & 1 & 0 & -13/4 & 2/4 & -5/4 \\ 0 & 0 & 1 & 17/4 & -2/4 & 5/4 \end{array}\right]. \tag{3.26}$$

We next replace $R1$ by the sum of $R1$ and $(-1/5)\,R2$ to get

$$[\,I\,|B\,] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 3/2 & 0 & 1/2 \\ 0 & 1 & 0 & -13/4 & 2/4 & -5/4 \\ 0 & 0 & 1 & 17/4 & -2/4 & 5/4 \end{array}\right]. \tag{3.27}$$

This determines the inverse of A as

$$A^{-1} = B = \left[\begin{array}{ccc} 3/2 & 0 & 1/2 \\ -13/4 & 2/4 & -5/4 \\ 17/4 & -2/4 & 5/4 \end{array}\right] = \frac{1}{4}\left[\begin{array}{ccc} 6 & 0 & 2 \\ -13 & 2 & -5 \\ 17 & -2 & 5 \end{array}\right] \tag{3.28}$$

which agrees with a Maxima calculation using either **`invert`** or **`rinverse`**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([0,1,1],[5,1,-1], [2,-3,-3]);
(%o2) matrix([0,1,1],[5,1,-1],[2,-3,-3])
(%i3) B : invert(A);
(%o3) matrix([3/2,0,1/2],[-13/4,1/2,-5/4],[17/4,-1/2,5/4])
(%i4) 4*B;
(%o4) matrix([6,0,2],[-13,2,-5],[17,-2,5])
(%i5) rinverse(A);
(%o5) matrix([3/2,0,1/2],[-13/4,1/2,-5/4],[17/4,-1/2,5/4])
(%i6) 4*%;
(%o6) matrix([6,0,2],[-13,2,-5],[17,-2,5])
```

The code for **rinverse(M)** is:

```
/* rinverse(M) uses echelon and elementary row operations to derive inverse of M,
    mat_unblocker is from linearalgebra.mac, which loads automatically */

rinverse(M) :=
block([nrows,ncols, MI , M_inv ],
    local(RL),          /* local hashed array */
    nrows : length(M),
    ncols : length( transpose(M)),

    if nrows # ncols then (
        print(" need square matrix M "),
        return(done)),

    if rank(M) < nrows then (
        print(" the inverse does not exist"),
        return(done)),

    MI : mat_unblocker( matrix ( [ M, ident(nrows) ] ) ),
    MI : echelon (MI),

    for j thru nrows do   RL[j] : lme (row (MI, j)),

        /* create 0's above 1's   */
    for j:2 thru nrows do
        for k thru (j - 1) do RL[k] : RL[k]  - RL[k][j] * RL[j],

     /* right hand of partitioned matrix should now be M_inverse */

    for j thru nrows do RL[j] : rest ( RL[j], ncols ),

    M_inv : matrix(RL[1]),
    for j:2 thru nrows do M_inv : addrow(M_inv, RL[j] ),
    M_inv)$
```

4 Set of Homogeneous Linear Algebraic Equations $\mathrm{A}\,\mathrm{x} = 0$

Let A be a $m \times n$ matrix (m rows and n columns). Let $\mathbf{x}$ be an n-component column vector. Let $\mathbf{0}$ be a m-dimensional column vector, all of whose elements are 0. Then the matrix equation $A\,\mathbf{x} = \mathbf{0}$ is equivalent to a system of m linear homogeneous equations in n unknowns.

Case $m > n$

In this case there are more equations than unknowns (the system is "overdetermined") and only the trivial solution $\mathbf{x} = \mathbf{0}$ exists. Below we discuss the concepts of **nullspace** of a matrix and the definition of the dimension of the nullspace, called the **nullity** of the matrix. The dimension of the "nullspace" of the matrix in the following $m > n$ example is 0. Here is an example of three equations in two unknowns ($m = 3$, $n = 2$).

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) solve([x + y,2*x + y,x + 2*y],[x,y]);
solve: dependent equations eliminated: (3)
(%o2) [[x = 0,y = 0]]
(%i3) A1 : matrix([1,1],[2,1],[1,2]);
(%o3) matrix([1,1],[2,1],[1,2])
(%i4) rank(A1);
(%o4) 2
(%i5) nullity(A1);
(%o5) 0
```

```
(%i6) nullspace(A1);
(%o6) span()
```

4.1 Nullspace Basis Vectors of a Matrix

Let A be a $m \times n$ matrix (m rows and n columns). In the following, we assume $m \leq n$. The **nullspace** of the matrix A, denoted $\mathbf{Null}A$, is the **set** of all n-dimensional column vectors $\mathbf{x}$ such that $A\,\mathbf{x} = \mathbf{0}$, in which the right hand side is a m-component vector, all of whose elements equal 0. $\mathbf{Null}A$ is a **subspace** of the set of all n-dimensional vectors. Elementary row operations on a matrix A do not change $\mathbf{Null}A$.

The **dimension** of the nullspace $\mathbf{Null}A$ is called the **nullity** of the matrix A, and can be denoted by $\mathbf{nullity}(A)$ or (more often) by $\mathbf{dim}\mathbf{Null}A$. Since $A\,\mathbf{0} = \mathbf{0}$, the set $\mathbf{Null}A$ always contains $\mathbf{0}$, the trivial solution, so using set notation, $\mathbf{Null}A \neq \emptyset$. If $\mathbf{Null}A$ **only** contains the trivial solution vector, then the nullspace has dimension 0.

If the **square** matrix A is **invertible** (A^{-1} exists), then (again using set notation) $\mathbf{Null}A = \{\mathbf{0}\}$, since the solution to $A\,\mathbf{x} = \mathbf{0}$ is given by the unique solution $\mathbf{x} = A^{-1}\,\mathbf{0} = \mathbf{0}$.

The **rank-nullity theorem** states that the sum of the rank of a matrix and the nullity of a matrix is equal to the number of columns of the matrix. This theorem is used to define the function **`nullity(A)`** (in **`mbe5.mac`**):

```
  nullity(BB)  := ( length(transpose(BB))  -  rank(BB) )$
```

Here is a simple example, which eventually uses **`nullspace(A)`**, a function defined in **`linearalgebra.mac`**, which automatically loads when you load **`mbe5.mac`**.

We seek the nullspace of A, $\mathbf{Null}A$, for the 2×3 matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}. \tag{4.1}$$

In our work below, **`A`** is a 2×3 matrix (two rows and three columns), **`xs`** is a 3-element column vector of unknowns, and the matrix product **`A . xs`** produces a 2-element column vector, so the matrix equation $A\,\mathbf{x_s} = \mathbf{0}$ requres the right hand side to be a 2-element column of 0's, and the result is equivalent to two equations in three unknowns, an under-determined system, whose solution will involve one arbitrary parameter.

The work below then shows the nullspace of **`A`** is spanned by one basis vector, and the most general vector belonging the the nullspace of **`A`** is an arbitrary constant times the basis vector. This is consistent with **`nullity(A) --> 1`**. In the following, we show two different methods to obtain the matrix column vector **`v1`**, given the return value from **`nullspace(A)`**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ([1,2,3],[4,5,6]);
(%o2) matrix([1,2,3],[4,5,6])
(%i3) rank(A);
(%o3) 2
(%i4) nullity(A);
(%o4) 1
(%i5) nspc : nullspace(A);
(%o5) span(matrix([-3],[6],[-3]))
(%i6) [v1] : nspc, 'span = "[";
(%o6) [matrix([-3],[6],[-3])]
(%i7) v1;
(%o7) matrix([-3],[6],[-3])
(%i8) v1 : part (nspc, 1);
(%o8) matrix([-3],[6],[-3])
(%i9) A . v1;
(%o9) matrix([0],[0])
```

```
(%i10) v1/(-3);
(%o10) matrix([1],[-2],[1])
```

We compare the use of **nullspace** (above) with the direct use of **solve** for this system of algebraic equations, getting the equations from the matrix definition for practice. We see below that **solve** produces a solution vector depending on one arbitrary parameter, which we call **r**. We let **Axs** be the matrix column vector (two rows, one column) given by the matrix product **A . xs**.

```
(%i11) xs : cvec([x,y,z]);
(%o11) matrix([x],[y],[z])
(%i12) Axs : A . xs;
(%o12) matrix([3*z+2*y+x],[6*z+5*y+4*x])
(%i13) eq1 : Axs[1,1];
(%o13) 3*z+2*y+x
(%i14) eq2 : Axs[2,1];
(%o14) 6*z+5*y+4*x
(%i15) solns : solve([eq1,eq2],[x,y,z]);
(%o15) [[x = %r1,y = -2*%r1,z = %r1]]
(%i16) solns : solns, %r1 = r;
(%o16) [[x = r,y = -2*r,z = r]]
(%i17) soln : solns[1];
(%o17) [x = r,y = -2*r,z = r]
(%i18) xL : map('rhs,soln);
(%o18) [r,-2*r,r]
(%i19) xvec : cvec(xL);
(%o19) matrix([r],[-2*r],[r])
(%i20) A . xvec;
(%o20) matrix([0],[0])
(%i21) xvec/r;
(%o21) matrix([1],[-2],[1])
```

In general, one finds the nullspace of a matrix by finding a basis, and the dimension of the basis is then known. The dimension of the nullspace of A is equal to the number of basis vectors needed to "span the space", i.e., allow any solution of $A\,\mathbf{x} = \mathbf{0}$ to be written in terms of the set of basis vectors, using multiplicative scalar constants (parameters).

If $\mathbf{u}$ is a basis vector of a one dimensional vector space, so that $A\,\mathbf{u} = \mathbf{0}$, then the most general vector in that space is $\mathbf{x} = c\,\mathbf{u}$, in which c is a scalar parameter, since $A\,c\,\mathbf{u} = c\,A\,\mathbf{u} = c\,\mathbf{0} = \mathbf{0}$.

Here is another example. We seek the nullspace of A, $\mathbf{Null}A$, for the 3×5 matrix

$$A = \begin{bmatrix} -3 & 6 & -1 & 1 & -7 \\ 1 & -2 & 2 & 3 & -1 \\ 2 & -4 & 5 & 8 & -4 \end{bmatrix}. \tag{4.2}$$

The elements of $\mathbf{Null}A$ will be 5-component vectors, and the matrix equation $A\,\mathbf{x} = \mathbf{0}$ stands for three algebraic equations.

A Maxima solution for the nullspace of A can be found using the Maxima contributed package **linearalgebra.mac**, which has its own definition of the functions **rank(A)**, **nullity(A)**, and **nullspace(A)** which are illustrated here. When you load **mbe5.mac**, the file **linearalgebra.mac** will also load. The function **nullspace** returns the vectors which form a basis of the nullspace in the special form **span(v1,v2,...vn)**, and one can use **subst('span = "[", returned-result)** to get an ordinary Maxima list of basis vectors, if you wish.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([-3,6,-1,1,-7],[1,-2,2,3,-1], [2,-4,5,8,-4]);
(%o2) matrix([-3,6,-1,1,-7],[1,-2,2,3,-1],[2,-4,5,8,-4])
(%i3) rank(A);
(%o3) 2
(%i4) nullity(A);
(%o4) 3
```

```
(%i5) nspc : nullspace(A);
(%o5) span(matrix([-10],[-5],[0],[0],[0]),matrix([0],[0],[-20],[15],[5]),
            matrix([0],[5],[20],[-10],[0]))
(%i6) [v1,v2,v3] : nspc, 'span = "[";
(%o6) [matrix([-10],[-5],[0],[0],[0]),matrix([0],[0],[-20],[15],[5]),
       matrix([0],[5],[20],[-10],[0])]
(%i7) v1;
(%o7) matrix([-10],[-5],[0],[0],[0])
(%i8) A . v1;
(%o8) matrix([0],[0],[0])
(%i9) map(lambda([u], A . u),[v1,v2,v3]);
(%o9) [matrix([0],[0],[0]),matrix([0],[0],[0]),matrix([0],[0],[0])]
```

Instead of using **`nullspace`**, we can use the core Maxima function **`solve`** to find the nullspace of A, as follows here.

Without using the rank-nullity theorem, we can use the Maxima function **`solve`** to "solve" for the basis vectors of the nullspace of A and thus infer the dimension of the nullspace. We use the symbol **`xL`** for a **list** of the five (unknown) elements of $\mathbf{x}$. We use the symbol **`x`** to represent, in matrix form, the 5-element column vector $\mathbf{x}$. We let the symbol **`Ax`** represent the column vector produced by the matrix product **`A . x`**, and we let the symbol **`AxL`** represent the list of the elements of the column vector **`Ax`**.

```
(%i10) xL : [x1,x2,x3,x4,x5];
(%o10) [x1,x2,x3,x4,x5]
(%i11) x : cvec(xL);
(%o11) matrix([x1],[x2],[x3],[x4],[x5])
(%i12) Ax : A . x;
(%o12) matrix([(-7*x5)+x4-x3+6*x2-3*x1],[(-x5)+3*x4+2*x3-2*x2+x1],
              [(-4*x5)+8*x4+5*x3-4*x2+2*x1])
(%i13) AxL : list_matrix_entries(Ax);
(%o13) [(-7*x5)+x4-x3+6*x2-3*x1,(-x5)+3*x4+2*x3-2*x2+x1,(-4*x5)+8*x4+5*x3-4*x2
                                                              +2*x1]
(%i14) solns : solve(AxL,xL);
solve: dependent equations eliminated: (3)
(%o14) [[x1 = 2*%r3+%r2-3*%r1,x2 = %r3,x3 = 2*%r1-2*%r2,x4 = %r2,x5 = %r1]]
```

The solution depends on three arbitrary parameters, which could be called anything, so this subspace is three dimensional, spanned by three (5-element) basis vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$, which will respectively be represented by **`u`**, **`v`**, and **`w`** in our Maxima work. The Maxima function **`solve`** uses **`%rj`** as the symbol for arbitrary parameters, starting with $j = 1$. In our own summary, we can use any symbols for these arbitrary parameters. If we write this solution as

$$\mathbf{x} = r_1\,\mathbf{u} + r_2\,\mathbf{v} + r_3\,\mathbf{w}, \tag{4.3}$$

then we have

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = r_1 \begin{bmatrix} -3 \\ 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} + r_2 \begin{bmatrix} 1 \\ 0 \\ -2 \\ 1 \\ 0 \end{bmatrix} + r_3 \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

from which we make the identification of three possible basis vectors of the three dimensional nullspace of A:

$$\mathbf{u} = \begin{bmatrix} -3 \\ 0 \\ 2 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ -2 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.5}$$

Using our function **`cvec(L)`** in **`mbe5.mac`** to define Maxima column vectors in terms of matrices,

```
(%i15 u : cvec([-3,0,2,0,1]);
(%o15) matrix([-3],[0],[2],[0],[1])
```

```
(%i16) v : cvec([1,0,-2,1,0]);
(%o16) matrix([1],[0],[-2],[1],[0])
(%i17) w : cvec([2,1,0,0,0]);
(%o17) matrix([2],[1],[0],[0],[0])
```

If these three basis vectors are linearly dependent, then there exist scalar constants c_1, c_2, and c_3 (not all zero) such that $\mathbf{z} = c_1\,\mathbf{u} + c_2\,\mathbf{v} + c_3\,\mathbf{w} = \mathbf{0}$ so that we can express one of the vectors $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ in terms of the others.

```
(%i18) cL : [c1,c2,c3];
(%o18) [c1,c2,c3]
(%i19) z : c1*u + c2*v + c3*w;
(%o19) matrix([2*c3+c2-3*c1],[c3],[2*c1-2*c2],[c2],[c1])
(%i20) zL : list_matrix_entries(z);
(%o20) [2*c3+c2-3*c1,c3,2*c1-2*c2,c2,c1]
(%i21) solve(zL, cL);
solve: dependent equations eliminated: (5 1)
(%o21) [[c1 = 0,c2 = 0,c3 = 0]]
```

and the only solution for the set of three constants c_j is the trivial solution $c_j = 0$. This should have been expected, since we are trying to solve five equations for three constants, an overdetermined system. We conclude that the three basis vectors found, $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$, are linearly independent, and are a basis of the nullspace of A.

We can obtain a different set of basis vectors of the nullspace of A by using **`solve`** with **`B : echelon(A)`**.

```
(%i22) B : echelon(A);
(%o22) matrix([1,-2,1/3,-1/3,7/3],[0,0,1,2,-2],[0,0,0,0,0])
(%i23) Bx : B . x;
(%o23) matrix([(7*x5)/3-x4/3+x3/3-2*x2+x1],[(-2*x5)+2*x4+x3],[0])
(%i24) BxL : list_matrix_entries(Bx);
(%o24) [(7*x5)/3-x4/3+x3/3-2*x2+x1,(-2*x5)+2*x4+x3,0]
(%i25) soln : solve(BxL,xL);
solve: dependent equations eliminated: (3)
(%o25) [[x1 = %r3,x2 = (%r3-%r2+3*%r1)/2,x3 = 2*%r1-2*%r2,x4 = %r2,x5 = %r1]]
```

If we again write this solution as

$$\mathbf{x} = r_1\,\mathbf{u}' + r_2\,\mathbf{v}' + r_3\,\mathbf{w}', \tag{4.6}$$

then we have

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = r_1 \begin{bmatrix} 0 \\ 3/2 \\ 2 \\ 0 \\ 1 \end{bmatrix} + r_2 \begin{bmatrix} 0 \\ -1/2 \\ -2 \\ 1 \\ 0 \end{bmatrix} + r_3 \begin{bmatrix} 1 \\ 1/2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

from which we make the identification of a set of three possible basis vectors of the three dimensional nullspace of A:

$$\mathbf{u}' = \begin{bmatrix} 0 \\ 3/2 \\ 2 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{v}' = \begin{bmatrix} 0 \\ -1/2 \\ -2 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{w}' = \begin{bmatrix} 1 \\ 1/2 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.8}$$

Case $m < n$

A system of m linear homogeneous equations in n unknowns always has a nontrivial solution if $m < n$. Here is an example of two equations in three unknowns ($m = 2$, $n = 3$).

```
(%i26) solve([x+y+z,2*x+y+z],[x,y,z]);
(%o26) [[x = 0,y = -%r1,z = %r1]]
```

where **`%r1`** can be chosen at will.

Case $m = n$

Here we consider the case $m = n$, the number of equations equals the number of unknowns. We assume here that A is a square $n \times n$ matrix, (the "coefficient matrix"), and the n-component column vector $\mathbf{x}$ is to be found. The matrix equation $A\,\mathbf{x} = \mathbf{0}$ then stands for n equations in n unknowns.

This system of equations has the unique trivial solution $x_j = 0$ for all j if $\mathbf{det}A \neq 0$. A nontrivial solution, defined up to a multiplicative constant, exists if and only if $\mathbf{det}A = 0$. An alternative requirement is that the rank of the matrix be one less than the number of columns. Here is a 3×3 singular matrix example, using for A

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}. \tag{4.9}$$

If we use the **nullspace** function from **linearalgebra.mac**, we find that the dimension of the nullspace of A is 1, and the single basis vector is proportional to $[-3, 6, -3]^{\mathrm{T}}$, which means the general solution is $\mathbf{x} = c\,[1, -2, 1]^{\mathrm{T}}$, where c is a constant.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,2,3],[4,5,6],[7,8,9]);
(%o2) matrix([1,2,3],[4,5,6],[7,8,9])
(%i3) determinant(A);
(%o3) 0
(%i4) rank(A);
(%o4) 2
(%i5) nullity(A);
(%o5) 1
(%i6) nullspace(A);
(%o6) span(matrix([-3],[6],[-3]))
```

If instead we use the Maxima function **solve**, we get the same result.

```
(%i7) xL : [x1,x2,x3];
(%o7) [x1,x2,x3]
(%i8) x : cvec(xL);
(%o8) matrix([x1],[x2],[x3])
(%i9) Ax : A . x;
(%o9) matrix([3*x3+2*x2+x1],[6*x3+5*x2+4*x1],[9*x3+8*x2+7*x1])
(%i10) AxL : list_matrix_entries(Ax);
(%o10) [3*x3+2*x2+x1,6*x3+5*x2+4*x1,9*x3+8*x2+7*x1]
(%i11) soln : solve(AxL,xL);
solve: dependent equations eliminated: (3)
(%o11) [[x1 = %r1,x2 = -2*%r1,x3 = %r1]]
```

We can use **echelon** to triangularize A without changing the solution of the homogeneous equation, as another way to get the general solution.

```
(%i12) B : echelon(A);
(%o12) matrix([1,2,3],[0,1,2],[0,0,0])
(%i13) rank(B);
(%o13) 2
(%i14) nullity(B);
(%o14) 1
(%i15) determinant(B);
(%o15) 0
(%i16) Bx : B . x;
(%o16) matrix([3*x3+2*x2+x1],[2*x3+x2],[0])
(%i17) BxL : list_matrix_entries(Bx);
(%o17) [3*x3+2*x2+x1,2*x3+x2,0]
(%i18) BxL : rest(BxL,-1);
(%o18) [3*x3+2*x2+x1,2*x3+x2]
(%i19) soln : solve(BxL,xL);
(%o19) [[x1 = %r2,x2 = -2*%r2,x3 = %r2]]
```

which produces the same general solution as when using the original matrix A.

4.2 Examples of Eigenvalues and Eigenvectors

The set of eigenvalues of a matrix is often called the **spectrum** of the matrix. It is important to note that not all matrices have eigenvalues. For example the matrix

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{4.10}$$

does not have nonzero eigenvalues.

```
(%i1) A : matrix([0,1],[0,0]);
(%o1) matrix([0,1],[0,0])
(%i2) rank(A);
(%o2) 1
(%i3) eigenvalues(A);
(%o3) [[0],[2]]
```

Let A be a square $n \times n$ matrix. Let the column vectors $\mathbf{v}^j$ be n-component eigenvectors of A, and let λ_j be the corresponding scalar (nonzero) eigenvalues (either real or complex) which satisfy the equation

$$A\,\mathbf{v}^j = \lambda_j\,\mathbf{v}^j, \quad 1 \le j \le n \tag{4.11}$$

From linear algebra

$$A\,\mathbf{v} = \lambda\,\mathbf{v} \quad \Leftrightarrow \quad (A - \lambda\,I)\,\mathbf{v} = 0 \tag{4.12}$$

and $\mathbf{v} \neq \mathbf{0}$ if and only if

$$p(\lambda) = \mathbf{det}(A - \lambda\,I) = 0. \tag{4.13}$$

Each of the set of eigenvalues λ_j satisfy the "characteristic equation"

$$p(\lambda) = 0. \tag{4.14}$$

The "fundamental theorem of algebra" states that if the roots of (4.14) are counted with multiplicities, then $p(\lambda)$ has exactly n roots $\lambda_1, \ldots, \lambda_n$ and

$$p(\lambda) = (-1)^n\,(\lambda - \lambda_1)\ldots(\lambda - \lambda_n). \tag{4.15}$$

If A has n distinct eigenvalues $\lambda_1, \ldots, \lambda_n$, then there exist n linearly independent eigenvectors $\mathbf{v^1}, \ldots, \mathbf{v^n}$.

Determinant Test for Linear Independence of Eigenvectors

Let M be the $n \times n$ matrix whose columns $\mathbf{m}^{(j)}$ are the eigenvectors $\mathbf{v}^{(j)}$ of the matrix A, and let $\mathbf{c}$ be an n-component column vector whose elements are constant numbers c_j such that the equation

$$M\,\mathbf{c} = \sum_j \mathbf{v}^{(j)}\,c_j = \mathbf{0} \tag{4.16}$$

is satisfied. A non-trivial solution, in which not all of the c_j are equal to zero, exists if and only if $\mathbf{det}(M) = 0$, in which case we can always express one of the $\mathbf{v}^{(j)}$ as a linear combination of the other eigenvectors. Thus we can only have a linearly independent set of eigenvectors $\mathbf{v}^{(j)}$ if $\mathbf{det}(M) \neq 0$.

Some Simple Facts to Remember

1. The sum of the eigenvalues of a matrix is equal to the **trace** of that matrix, which is the sum of the elements on its main diagonal.
2. The product of the eigenvalues (counting multiplicities) of a matrix is equal to the determinant of that matrix.
3. Eigenvectors corresponding to different eigenvalues are linearly independent.
4. A matrix is singular if and only if it has a zero eigenvalue, in which case its determinant is zero.

In general, a $n \times n$ matrix A has n eigenvalues. If two or more of these eigenvalues are equal, we say that A is **degenerate**. If A has n distinct eigenvalues, A is said to be **nondegenerate**. A has at least as many linearly independent eigenvectors as it has distinct eigenvalues.

Eigenvalues and Eigenvectors of Hermitian Matrices

Quoting Boyce and DiPrima (Ch. 7), and using a dagger, as in $A^\dagger$ instead of A^* for the Hermitian conjugation symbol, and using the overbar $\overline{a}_{ji}$ for complex conjugation:

> An important class of matrices, called **self-adjoint** or **Hermitian** matrices, are those for which $A^\dagger = A$, that is $\overline{a}_{ji} = a_{ij}$. Hermitian matrices include as a subclass real symmetric matrices, that is, matrices which have real elements and for which $A^{\mathrm{T}} = A$. The eigenvalues and eigenvectors of Hermitian matrices always have the following useful properties.

1. All eigenvalues are real.
2. There always exists a full set of *n* linearly independent eigenvectors, regardless of the multiplicities of the eigenvalues.
3. If $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ are eigenvectors that correspond to different eigenvalues, then the inner product $(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) = 0$. Thus, if all eigenvalues are "simple" (multiplicity one), then the associated eigenvectors form an orthogonal set of vectors.
4. Corresponding to an eigenvalue of multiplicity *m* , it is possible to choose *m* eigenvectors that are mutually orthogonal. Thus the full set of *n* eigenvectors can always be chosen to be orthogonal as well as linearly independent.

Eigenvalues and Eigenvectors of Non-Hermitian Matrices

If the $n \times n$ matrix A is **not** Hermitian, consider first the case that A is **real**. Then there are three possibilities for the eigenvalues of A:
1. All eigenvalues are real and distinct.
2. Some eigenvalues occur in complex conjugate pairs, but the real eigenvalues are all distinct.
3. Some real eigenvalues are repeated.

The first case leads to no difficulties. There is a single linearly independent real eigenvector for each eigenvalue.

In the second case, there exist n linearly independent eigenvectors, with those corresponding to the complex eigenvalues being themselves complex.

In the third case the number of linearly independent eigenvectors corresponding to a single repeated eigenvalue may be less than the multiplicity of that repeated eigenvalue.

If A is non-Hermitian but is **complex**, then complex conjugate eigenvalues need not occur in conjugate pairs and the eigenvectors are normally complex even if the corresponding eigenvalue is real.

Example 1: Distinct Real Eigenvalues of a Real Non-Hermitian Matrix

We take the case

$$A = \begin{bmatrix} 8 & -5 & 10 \\ 2 & 1 & 2 \\ -4 & 4 & -6 \end{bmatrix}. \tag{4.17}$$

To find the trace of a given matrix, we use the function **`mtrace`** from the file **`mbe5.mac`**. The three eigenvalues of the given matrix A are real and distinct (each eigenvalue has multiplicity $m = 1$). We can use **`eigenvalues(A)`**, **`jordan(A)`**, **`eigenvectors(A)`**, and **`jordan_chain(A,eival)`** to get the distinct eigenvalues, multiplicities, and eigenvectors (see also Sec. 5).

We first show that A is **not** a "normal matrix", since $A\,A^{\mathrm{H}} \neq A^{\mathrm{H}}\,A$, where in general $A^{\mathrm{H}} = \overline{A^{\mathrm{T}}} = \left(\overline{A}\right)^{\mathrm{T}}$ is the Hermitian conjugate of A, which reduces to the transpose of A if all the elements of A are real. Only if A is a normal matrix can we

be assured that there exists a set of eigenvectors which can be arranged to form an **orthogonal** set. To determine if A is a normal matrix, we can use the **mbe5.mac** function **normal(A)** which returns either **true** or **false**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([8,-5,10],[2,1,2],[-4,4,-6]);
(%o2) matrix([8,-5,10],[2,1,2],[-4,4,-6])
(%i3) normal(A);
(%o3) false
(%i4) diagp(A);
(%o4) true
(%i5) eigenvalues(A);
(%o5) [[-2,2,3],[1,1,1]]
(%i6) jordan(A);
(%o6) [[-2,1],[2,1],[3,1]]
(%i7) [a1,a2,a3] : map('first, %);
(%o7) [-2,2,3]
(%i8) eigenvectors(A);
(%o8) [[[-2,2,3],[1,1,1]],[[[1,0,-1]],[[0,1,1/2]],[[1,1,0]]]]
(%i9) [v1] : map ('cvec, jordan_chain(A,a1));
(%o9) [matrix([1],[0],[-1])]
(%i10) [v2] : map ('cvec, jordan_chain(A,a2));
(%o10) [matrix([0],[1],[1/2])]
(%i11) [v3] : map ('cvec, jordan_chain(A,a3));
(%o11) [matrix([1],[1],[0])]
```

Each eigenvector is defined up to an overall constant multiplier. The eigenvalue $\lambda_1 = -2$ corresponds to the eigenvector

$$\mathbf{v}^1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}. \tag{4.18}$$

The eigenvalue $\lambda_2 = 2$ corresponds to the eigenvector

$$\mathbf{v}^2 = \begin{bmatrix} 0 \\ 1 \\ 1/2 \end{bmatrix}. \tag{4.19}$$

The eigenvalue $\lambda_3 = 3$ corresponds to the eigenvector

$$\mathbf{v}^3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}. \tag{4.20}$$

If in doubt about the correctness of the returned results, we can check the eigenvalue equation for the three eigenvectors:

```
(%i12) is (equal (A . v1, a1*v1));
(%o12) true
(%i13) is (equal (A . v2, a2*v2));
(%o13) true
(%i14) is (equal (A . v3, a3*v3));
(%o14) true
```

The three rank 1 eigenvectors found are not normalized to 1, nor are they mutually orthogonal.

```
(%i15) map ('inner_prod, [v1,v2,v3],[v1,v2,v3] );
(%o15) [2,5/4,2]
(%i16) map ('inner_prod, [v1,v2,v3], [v2,v3,v1]);
(%o16) [-1/2,1,1]
```

We see that **none** of the eigenvectors produced are orthogonal to the others. If we **try** to use the **mbe5.mac** function **gschmidt(L)** which calls the **eigen.mac** package function **gramschmidt**, despite the fact that A is not a "normal" matrix, we find that although we can end up with orthogonal vectors, two of them are no longer eigenvectors of A.

We use the alias **lme**, defined in **mbe5.mac** for **list_matrix_entries**, which converts a column matrix into a list of values.

```
(%i17) [v1l,v2L,v3L] : map ('lme, [v1,v2,v3]);
(%o17) [[1,0,-1],[0,1,1/2],[1,1,0]]
(%i18) gschmidt(%);
(%o18) [[1,0,-1],[1/4,1,1/4],[2/9,-1/9,2/9]]
(%i19) [v1,v2,v3] : map ('cvec, %);
(%o19) [matrix([1],[0],[-1]),matrix([1/4],[1],[1/4]),
                      matrix([2/9],[-1/9], [2/9])]
(%i20) map ('inner_prod, [v1,v2,v3],[v1,v2,v3] );
(%o20) [2,9/8,1/9]
(%i21) map ('inner_prod, [v1,v2,v3], [v2,v3,v1]);
(%o21) [0,0,0]
```

which shows mutual orthogonality, but two of the vectors are no longer eigenvectors of A corresponding to the eigenvalues of A:

```
(%i22) is (equal (A . v1, a1*v1));
(%o22) true
(%i23) is (equal (A . v2, a2*v2));
(%o23) false
(%i24) is (equal (A . v3, a3*v3));
(%o24) false
```

Ex. 1: Finding the Eigenvalues and Eigenvectors "by Hand"

If we work this out "by hand", instead of using **`eigenvectors`**, we first form the characteristic polynomial $p(\lambda)$ (4.13) whose roots are the eigenvalues of A. We will use **`a`** instead of **`lambda`** to save space. The Maxima function **`charpoly(A, a)`** returns the appropriate characteristic polynomial, as a function of **`a`**, given the matrix **`A`**. In our Maxima work, we let **`v`** be a general vector with unknown components **`u1,u2,u3`**. (This is easier to type than **`v11,v12,v13`**.)

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) p : expand (charpoly(A, a));
(%o2) (-a^3)+3*a^2+4*a-12
(%i3) a_solns : solve(p);
(%o3) [a = -2,a = 2,a = 3]
(%i4) [a1,a2, a3] : map('rhs, a_solns);
(%o4) [-2,2,3]
(%i5) v : cvec ( [u1, u2, u3] );
(%o5) matrix([u1],[u2],[u3])
```

We first solve for an eigenvector **`v1`** corresponding to the eigenvalue **`a1 = -2`**. In the first step, the column vector **`w`** should have all zero components if **`v`** is an eigenvector corresponding to **`a1`**.

```
(%i6) w : A . v - a1*v;
(%o6) matrix([10*u3-5*u2+10*u1],[2*u3+3*u2+2*u1],[(-4*u3)+4*u2-4*u1])
(%i7) w[1,1];
(%o7) 10*u3-5*u2+10*u1
(%i8) u_solns : solve([w[1,1],w[2,1],w[3,1]],[u1,u2,u3]);
solve: dependent equations eliminated: (3)
(%o8) [[u1 = -%r10,u2 = 0,u3 = %r10]]
```

We then have the freedom to choose the arbitrary constant **`%r10 = -1`** to define the column eigenvector **`v1`** corresponding to the eigenvalue **`a1 = -2`** (the same result returned by **`jordan_chain(A,a1)`**). We then perform an unnecessary check on the eigenvector solution.

```
(%i9) map ('rhs, first(u_solns));
(%o9) [-%r10,0,%r10]
(%i10) v1 : cvec (%), %r10 = -1;
(%o10) matrix([1],[0],[-1])
(%i11) is (equal (A . v1, a1*v1));
(%o11) true
```

We are solving a set of three equations for three unknowns **u1**, **u2**, and **u3**, in which the right hand sides are zero. The coefficient matrix $B = A - \lambda\, I$ of this homogeneous set of equations

$$B\,\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.21}$$

can be simplified by replacing any row by a multiple of that row, and by replacing any target row by adding or subtracting another row to or from the target row, in a way that simplifies the coefficient matrix and the resulting algebra. Maxima has the function **triangularize(M)** and the function **echelon(M)** for this task, with the latter function giving the simplest result.

Reconsidering the eigenvector **v1** corresponding to **a1 = -2**, **w2** below represents a column vector, all of whose elements should be zero.

```
(%i12) B : A - a1*ident(3);
(%o12) matrix([10,-5,10],[2,3,2],[-4,4,-4])
(%i13) B1 : triangularize(B);
(%o13) matrix([10,-5,10],[0,40,0],[0,0,0])
(%i14) B2 : echelon(B);
(%o14) matrix([1,-1/2,1],[0,1,0],[0,0,0])
(%i15) w2 : B2 . v;
(%o15) matrix([u3-u2/2+u1],[u2],[0])
```

From the form of **w2** we immediately see that **u2 = 0**, and then using this in the first row, that **u3 = -u1**, and then the value of **u1** is arbitrary. Taking **u1 = 1** then leads again to the same eigenvector **v1**, without using **solve**.

We now use **echelon** to find the eigenvector **v2** corresponding to the eigenvalue **a2 = 2**.

```
(%i16) B : echelon(A - a2*ident(3));
(%o16) matrix([1,-5/6,5/3],[0,1,-2],[0,0,0])
(%i17) w : B . v;
(%o17) matrix([(5*u3)/3-(5*u2)/6+u1],[u2-2*u3],[0])
(%i18) u_solns : solve([w[1,1],w[2,1]],[u1,u2,u3]);
(%o18) [[u1 = 0,u2 = 2*%r2,u3 = %r2]]
```

If we choose **u2 = 1**, then **u3 = 1/2**.

```
(%i19) map ('rhs, first(u_solns));
(%o19) [0,2*%r11,%r11]
(%i20) v2 : cvec (%), %r11 = 1/2;
(%o20) matrix([0],[1],[1/2])
(%i21) is (equal (A . v2, a2*v2));
(%o21) true
```

We repeat this for the third eigenvalue **a3 = 3** and eigenvector **v3**:

```
(%i22) B : echelon(A - a3*ident(3));
(%o22) matrix([1,-1,2],[0,0,1],[0,0,0])
(%i23) w : B . v;
(%o23) matrix([2*u3-u2+u1],[u3],[0])
(%i24) u_solns : solve([w[1,1],w[2,1]],[u1,u2,u3]);
(%o24) [[u1 = %r3,u2 = %r3,u3 = 0]]
```

We choose **u1 = u2 = 1**.

```
(%i25) map ('rhs, first(u_solns));
(%o25) [%r12,%r12,0]
(%i26) v3 : cvec (%), %r12=1;
(%o26) matrix([1],[1],[0])
```

We have arrived at the same set of eigenvectors of A as we did using **jordan_chain**, but we needed to make a choice for an arbitrary parameter returned by **solve** for each eigenvector.

Example 2: Repeated Real Eigenvalues of a Hermitian Matrix

Let A be the matrix

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} . \tag{4.22}$$

A is real and symmetric ($A^{\mathrm{T}} = A$), a special case of a *Hermitian* or *self-adjoint* matrix. We know that any hermitian matrix is a normal matrix and there exists a set of mutually orthogonal eigenvectors.

The two distinct eigenvalues of the given matrix A are real with one eigenvalue having multiplicity two. We can use **`eigenvalues(A)`**, **`jordan(A)`**, **`eigenvectors(A)`**, and **`jordan_chain(A,eival)`** to get the distinct eigenvalues, multiplicities, and eigenvectors.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([0,1,1],[1,0,1],[1,1,0]);
(%o2) matrix([0,1,1],[1,0,1],[1,1,0])
(%i3) jn : jordan(A);
(%o3) [[2,1],[-1,1,1]]
(%i4) [a1,a2] : map ('first,jn);
(%o4) [2,-1]
(%i5) a3 : a2;
(%o5) -1
(%i6) [v1] : map ('cvec, jordan_chain(A,a1));
(%o6) [matrix([1],[1],[1])]
(%i7) [v2,v3] : map ('cvec, jordan_chain(A,a2));
(%o7) [matrix([1],[0],[-1]),matrix([0],[1],[-1])]
```

Each eigenvector is defined up to an overall constant multiplier. The eigenvalue $\lambda_1 = 2$ corresponds to the eigenvector

$$\mathbf{v}^1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} . \tag{4.23}$$

The eigenvalue $\lambda = -1$ has multiplicity 2. The eigenvalue $\lambda_2 = -1$ corresponds to the eigenvector

$$\mathbf{v}^2 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} . \tag{4.24}$$

The eigenvalue $\lambda_3 = -1$ corresponds to the eigenvector

$$\mathbf{v}^3 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} . \tag{4.25}$$

Here is a check of the eigenvalue equation solutions found:

```
(%i8) is (equal (A . v1, a1*v1));
(%o8) true
(%i9) is (equal (A . v2, a2*v2));
(%o9) true
(%i10) is (equal (A . v3, a3*v3));
(%o10) true
```

and a check of inner products.

```
(%i11) map ('inner_prod, [v1,v2,v3], [v1,v2,v3] );
(%o11) [3,2,2]
(%i12) map ('inner_prod, [v1,v2,v3], [v2,v3,v1] );
(%o12) [0,1,0]
```

We leave **v1** alone and orthogonalize the pair **(v2,v3)** using the Gram-Schmidt process.

```
(%i13) normal(A);
(%o13) true
(%i14) map ('lme,[v2,v3]);
(%o14) [[1,0,-1],[0,1,-1]]
(%i15) gschmidt(%);
(%o15) [[1,0,-1],[-1/2,1,-1/2]]
(%i16) [v2,v3] : map ('cvec, %);
(%o16) [matrix([1],[0],[-1]),matrix([-1/2],[1],[-1/2])]
(%i17) map ('inner_prod, [v1,v2,v3], [v1,v2,v3] );
(%o17) [3,2,3/2]
(%i18) map ('inner_prod, [v1,v2,v3], [v2,v3,v1] );
(%o18) [0,0,0]
```

Finally, we can produce an orthonormal set of eigenvectors using the **mbe5.mac** function **normed(v)** which returns a vector normalized to 1.

```
(%i19) [v1,v2,v3] : map ('normed, [v1,v2,v3])$
(%i20) map ('inner_prod, [v1,v2,v3], [v1,v2,v3] );
(%o20) [1,1,1]
(%i21) map ('inner_prod, [v1,v2,v3], [v2,v3,v1] );
(%o21) [0,0,0]
```

and again check the eigenvalue equation.

```
(%i22) is (equal (A . v1, a1*v1));
(%o22) true
(%i23) is (equal (A . v2, a2*v2));
(%o23) true
(%i24) is (equal (A . v3, a3*v3));
(%o24) true
```

Ex 2: Solving for the Eigenvectors by Hand

Given the eigenvalues, we solve for the eigenvectors by hand as an example of using Maxima at a lower level. First for $\lambda_1 = 2$, we need to solve for the eigenvector $\mathbf{v}$ which satisfies the eigenvector equation $(A - \lambda_1\, I)\,\mathbf{v} = B\,\mathbf{v} = \mathbf{0}$. We can simplify the algebra by reducing the coefficient matrix B to a simple triangular form using **echelon**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([0,1,1],[1,0,1],[1,1,0]);
(%o2) matrix([0,1,1],[1,0,1],[1,1,0])
(%i3) B : A - 2*ident(3);
(%o3) matrix([-2,1,1],[1,-2,1],[1,1,-2])
(%i4) B : echelon(B);
(%o4) matrix([1,-1/2,-1/2],[0,1,-1],[0,0,0])
(%i5) v : cvec ( [u1, u2, u3] );
(%o5) matrix([u1],[u2],[u3])
(%i6) w : B . v;
(%o6) matrix([(-u3/2)-u2/2+u1],[u2-u3],[0])
(%i7) w[1,1];
(%o7) (-u3/2)-u2/2+u1
(%i8) w[2,1];
(%o8) u2-u3
(%i9) u_solns : solve([w[1,1],w[2,1]],[u1,u2,u3]);
(%o9) [[u1 = %r1,u2 = %r1,u3 = %r1]]
```

All three components of $\mathbf{v}^1$ are equal, but otherwise not constrained, so the simplest choice is to take them all equal to 1, which is the choice made by **eigenvectors** above.

For the case $\lambda = -1$ we proceed as above:

```
(%i10) B : echelon(A + ident(3));
(%o10) matrix([1,1,1],[0,0,0],[0,0,0])
(%i11) w : B . v;
(%o11) matrix([u3+u2+u1],[0],[0])
```

The single constraint is the equation $u_1 + u_2 + u_3 = 0$, so two values can be chosen arbitrarily, and the third value is given by this equation. Choosing $u_1 = 1, u_2 = 0$, we then get $u_3 = -1$ for the components of $\mathbf{v}^2$. Since A is a Hermitian matrix, we know that $\mathbf{v}^1$ and $\mathbf{v}^2$ are automatically orthogonal (see below for a check).

To get a linearly independent eigenvector $\mathbf{v}^3$ corresponding to the same eigenvalue, we can make a different choice for u_1 and u_2 in such a way that $\mathbf{v}^3$ is not proportional to $\mathbf{v}^1$ nor to $\mathbf{v}^2$, for example $u_1 = 0, u_2 = 1$, we then get $u_3 = -1$ for the components of $\mathbf{v}^3$. Note that with this choice $\mathbf{v}^3$ is not orthogonal to $\mathbf{v}^2$, but *is* linearly independent. We then check that the determinant of the matrix whose columns are the three eigenvectors is not zero, to confirm the linear independence of our chosen set of eigenvectors.

```
(%i12) v1 : cvec([1,1,1]);
(%o12) matrix([1],[1],[1])
(%i13) v2 : cvec([1,0,-1]);
(%o13) matrix([1],[0],[-1])
(%i14) v3 : cvec([0,1,-1]);
(%o14) matrix([0],[1],[-1])
(%i15) determinant (mcombine ([v1,v2,v3]));
(%o15) 3
(%i16) map ('inner_prod, [v1,v2,v3],[v2,v3,v1] );
(%o16) [0,1,0]
```

Since $\mathbf{v}^2$ and $\mathbf{v}^3$ correspond to the same eigenvalue, they are not automatically orthogonal. If we require $\mathbf{v}^3$ to be orthogonal to $\mathbf{v}^2$, as well as corresponding to the eigenvalue $\lambda = -1$, then we need to satisfy two equations:

```
(%i17) eqn1 : u1+u2+u3 = 0;
(%o17) u3+u2+u1 = 0
(%i18) eqn2 : inner_prod(v2,v) = 0;
(%o18) u1-u3 = 0
(%i19) usolns : solve([eqn1,eqn2],[u1,u2,u3]);
(%o19) [[u1 = %r4, u2 = -2*%r4, u3 = %r4]]
```

Making the choice **`%r4 = 1`**,

```
(%i20) map ('rhs, first(usolns));
(%o20) [%r4, -2*%r4, %r4]
(%i21) v3 : cvec (%), %r4 = 1;
(%o21) matrix([1],[-2],[1])
(%i22) map ('inner_prod, [v1,v2,v3],[v2,v3,v1] );
(%o22) [0,0,0]
(%i23) determinant (mcombine ([v1,v2,v3]));
(%o23) -6
```

Example 3: Repeated Real Eigenvalues of a Real Non-Hermitian Matrix

For a non-Hermitian matrix, it may not be possible to find as many linearly independent eigenvectors (corresponding to a repeated eigenvalue) as the multiplicity of that eigenvalue. Thus, consider:

$$A = \begin{bmatrix} 1 & -1 \\ 1 & 3 \end{bmatrix}. \tag{4.26}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,-1],[1,3]);
(%o2) matrix([1,-1],[1,3])
(%i3) eigenvalues(A);
(%o3) [[2],[2]]
(%i4) jordan(A);
(%o4) [[2,2]]
(%i5) eigenvectors(A);
(%o5) [[[2],[2]],[[[1,-1]]]]
```

The eigenvalues are $\lambda = 2$ with a multiplicity $m = 2$, but only one rank 1 eigenvector is found by **`eigenvectors`**. The return value of **`jordan(A)`** indicates that there exists a set of generalized eigenvectors, a chain consisting of one rank 1 eigenvector, and one rank 2 generalized eigenvector (see Sec. 5).

```
(%i6) [v1,v2] : map ('cvec, jordan_chain(A,2));
(%o6) [matrix([-1],[1]),matrix([1],[0])]
(%i7) is (equal (A . v1, 2*v1));
(%o7) true
(%i8) is (equal (A . v2, 2*v2));
(%o8) false
(%i9) B : A - 2*ident(2);
(%o9) matrix([-1,-1],[1,1])
(%i10) vL_rank(B, [v1,v2] );
(%o10) [1,2]
(%i11) chained (v1, B, v2);
(%o11) true
```

If we look at the eigenvector problem by hand, from the point of view that if $\mathbf{v}$ is a rank 1 eigenvector of A corresponding to the eigenvalue $\lambda = 2$, and $B = A - 2\,I$, then $B\,\mathbf{v} = \mathbf{0}$.

```
(%i12) v : cvec([u1, u2]);
(%o12) matrix([u1],[u2])
(%i13) B : echelon(A - 2*ident(2));
(%o13) matrix([1,1],[0,0])
(%i14) w : B . v;
(%o14) matrix([u2+u1],[0])
```

and the only constraint on the eigenvector $\mathbf{v}$ is that $u_2 = -u_1$. With the arbitrary choice $u_1 = 1$, we get (up to an overall constant)

$$\mathbf{v} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}. \tag{4.27}$$

Thus there is only one rank 1 eigenvector associated with the repeated eigenvalue.

5 Generalized Eigenvectors, Jordan Chains, Canonical Basis, Jordan Canonical Form

There may not always exist a full set of n linearly independent eigenvectors of an $n \times n$ matrix A. This happens when the algebraic multiplicity m of at least one eigenvalue λ_i is greater than its "geometric multiplicity", m_g, defined as the **nullity** of the matrix $(A - \lambda_i\,I)$, which is the dimension of the **nullspace** of the matrix $(A - \lambda_i\,I)$.

The difference of the algebraic and geometric multiplicities $(m - m_g)$ is the number of extra "generalized eigenvectors" needed to get a complete set of n linearly independent eigenvectors of an $n \times n$ matrix A, and these extra vectors will have "vector rank" greater than 1.

The so-called "generalized eigenvectors" satisfy criteria which are more relaxed than those for an ordinary eigenvector. Quoting Bronson, Ch.9, where he defines the concept of **vector rank** of a **generalized eigenvector**,

> A non-zero vector $\mathbf{v}_m$ is a generalized (right) eigenvector of **vector rank** m for the square matrix A and associated eigenvalue λ if
>
> $$(A - \lambda\,I)^m\,\mathbf{v}_m = \mathbf{0} \quad \text{but} \quad (A - \lambda\,I)^{m-1}\,\mathbf{v}_m \neq \mathbf{0}. \tag{5.1}$$

In particular, for the usual eigenvectors with **vector rank** 1 which correspond to the case $m = 1$ (such as are returned by **`eigenvectors(A)`** or **`nullspace(A - eival*I)`**), because $B^0 = I$ for any square matrix B, and hence $B^0\,\mathbf{v} = \mathbf{v}$, for any B and any non-zero (conformable) column vector $\mathbf{v}$,

$$(A - \lambda\,I)\,\mathbf{v}_1 = \mathbf{0} \quad \text{but} \quad (A - \lambda\,I)^0\,\mathbf{v}_1 \neq \mathbf{0}. \tag{5.2}$$

Every $n \times n$ matrix A has n linearly independent generalized eigenvectors associated with it. A generalized eigenvector of vector rank 1 is an ordinary eigenvector. If λ_i is an eigenvalue of algebraic multiplicity m, then A will have m linearly independent generalized eigenvectors corresponding to λ_i (including the ordinary vector rank 1 eigenvectors). The set of

vectors spanned by all generalized eigenvectors for a given λ_i forms the **generalized eigenspace** for λ_i.

A generalized eigenvector $\mathbf{v}_i$ corresponding to λ_i, having multiplicity m, together with the matrix $B = (A - \lambda_i I)$ generate a **Jordan chain** of independent generalized eigenvectors which form either a **canonical basis** for the subspace associated with the matrix A and the eigenvalue λ_i or a "chained part" of a canonical basis.

An alternative set of m linearly independent generalized eigenvectors (a basis but not a "canonical basis") associated with the matrix A and the eigenvalue λ_i is returned by **`nullspace(B^^p)`**, in which $B = (A - \lambda_i I)$, and **`p`** is the smallest positive integer such that the dimension of the nullspace of **`B^^p`** equals the multiplicity m of the eigenvalue λ_i.

As discussed in the Examples below, the **`mbe5.mac`** function **`vrank_max(B,m)`** returns the positive integer **`p`** which satisfies the condition **`nullity(B^^p) = m`**.

Jordan Chains

Let $\mathbf{v}_m$ be a generalized eigenvector of vector rank m corresponding to the matrix A and the eigenvalue λ, such that

$$(A - \lambda I)^m \mathbf{v}_m = \mathbf{0} \quad \text{but} \quad (A - \lambda I)^{m-1} \mathbf{v}_m \neq \mathbf{0}. \tag{5.3}$$

The chain generated by $\mathbf{v}_m$ is a set of vectors $\{\mathbf{v}_m, \mathbf{v}_{m-1}, \ldots, \mathbf{v}_1\}$ with the properties

$$\begin{aligned}
\mathbf{v}_{m-1} &= (A - \lambda I)\mathbf{v}_m, \\
\mathbf{v}_{m-2} &= (A - \lambda I)^2 \mathbf{v}_m = (A - \lambda I)\mathbf{v}_{m-1}, \\
\mathbf{v}_{m-3} &= (A - \lambda I)^3 \mathbf{v}_m = (A - \lambda I)\mathbf{v}_{m-2}, \\
&\vdots \\
\mathbf{v}_1 &= (A - \lambda I)^{m-1} \mathbf{v}_m = (A - \lambda I)\mathbf{v}_2.
\end{aligned}$$

Thus, in general,

$$\mathbf{v}_j = (A - \lambda I)^{m-j} \mathbf{v}_m = (A - \lambda I)\mathbf{v}_{j+1} \qquad (j = 1, 2, \ldots, m-1). \tag{5.4}$$

and $\mathbf{v}_j$ is a generalized eigenvector of vector rank j corresponding to the eigenvalue λ. A chain is a linearly independent set of vectors.

Canonical Basis

Let A be an $n \times n$ matrix. A set of n linearly independent generalized eigenvectors is a **canonical basis** if it is composed entirely of Jordan chains. Thus, once we have determined that a generalized eigenvector of vector rank m is in a canonical basis, it follows that the $m - 1$ vectors $\mathbf{v}_{m-1}, \mathbf{v}_{m-2}, \ldots, \mathbf{v}_1$ that are in the Jordan chain generated by $\mathbf{v}_m$ are also in the canonical basis.

5.1 Example 1

Suppose

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \tag{5.5}$$

5.1.1 Example 1 Using mcol_solve(known_col,unknown_col, ukvarL)

This simple example introduces a group of useful **`mbe5.mac`** functions which can be used to analyze and check Jordan chains.

In the following Maxima session, we use **`eigenvalues`** to find that there is only one eigenvalue $\lambda = 1$ and its algebraic multiplicity is $m = 2$.

We use the rank 1 ordinary eigenvector returned by **eigenvectors**, calling it **v1**. For convenience, we define **B : A - ident(2)**, corresponding to $\lambda = 1$.

The function **vrank(B,v)** returns the vector rank of the column vector **v**.

The function **vrank_max(B,m)** returns the smallest positive integer **p** such that the dimension of the nullspace of **B^^p** equals the multiplicity **m** of the eigenvalue **eival**, with **B : A - eival*I**. This number **p** is then the maximum vector rank needed in the set of generalized eigenvectors. This is criterion is equivalent to **rank(B^^p) = n - m**, using the rank-nullity theorem (**n** is the number of rows of the matrix **A** or **B**).

The function **vrank_num(B,k)** returns the number of generalized eigenvectors which have vector rank equal to **k**.

The function **vrank_numbers(B,m)** returns the list **[[1,n1], [2,n2],...[p,np]]**, in which **nj** is the number of generalized eigenvectors (assoc. with the matrix A and eigenvalue λ with multiplicity m) which have rank **j**, and **p =** maximum vector rank, in which $B = (A - \lambda\, I)$.

We use **nullity(B)** to confirm that the "geometric multiplicity" m_g, (the dimension of the nullspace of the matrix **B**) is equal to 1, one less than the algebraic multiplicity, indicating that we need one generalized eigenvector of rank 2 (chained to the eigenvector of rank 1) to arrive at a full set of generalized eigenvectors corresponding to the single eigenvalue $\lambda = 1$. We finally check that **v1** is a rank 1 matrix, using both **vrank** and explicitly checking that **B . v1** is $[0, 0]^{\mathrm{T}}$, represented by **matrix([0],[0])**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,1],[0,1]);
(%o2) matrix([1,1],[0,1])
(%i3) eigenvalues(A);
(%o3) [[1],[2]]
(%i4) B : A - ident(2);
(%o4) matrix([0,1],[0,0])
(%i5) determinant(B);
(%o5) 0
(%i6) rank(B);
(%o6) 1
(%i7) nullity(B);
(%o7) 1
(%i8) vrank_max(B,2);
(%o8) 2
(%i9) nullity(B^^2);
(%o9) 2
(%i10) vrank_num(B,1);
(%o10) 1
(%i11) vrank_numbers(B,2);
 maximum vector rank =  2
(%o11) [[1,1],[2,1]]
(%i12) [eivals,vecs] : eigenvectors(A);
(%o12) [[[1],[2]],[[1,0]]]
(%i13) v1L : part(vecs,1,1);
(%o13) [1,0]
(%i14) v1 : cvec(v1L);
(%o14) matrix([1],[0])
(%i15) vrank(B,v1);
(%o15) 1
```

To solve for the rank 2 generalized eigenvector $\mathbf{v}^2$, represented in Maxima by **v2**, we define **v2** in terms of two unknowns, **v21** and **v22**, and require that the matrix equation $\mathbf{v}^1 = (A - \lambda\, I)\, \mathbf{v}^2$, represented in Maxima as **v1 = B . v2** is satisfied.

(Note: We cannot simply use **v2 : invert(B) . v1** here because the matrix **B** is singular and the inverse does not exist; either **rank(B) = 1 < 2** or equivalently, **determinant(B) = 0**, shows that **B** is singular.)

We let the symbol **Bv2** stand for the column matrix **B . v2**, which will be a function of the two unknowns **v21** and **v22**. We then let **solve** tell us the constraints imposed on these two unknown scalars, using the **mbe5.mac** function **mcol_solve(v1,Bv2,[v21,v22])**, which are then used to define a value of **v2** which satisfies the matrix equation $\mathbf{v}^1 = (A - \lambda\, I)\, \mathbf{v}^2$ between columns.

We then finally check that **v1 = B . v2** and **B^^2 . v2 = 0**, as is needed if **v2** is to be a rank 2 generalized eigenvector. The first property can be checked using **chained(v1,B,v2)**, which returns true if this relation is satisfied. The second property can be checked by using **vrank(B,v2)**, which returns 2.

```
(%i16) v2L : [v21,v22];
(%o16) [v21,v22]
(%i17) v2 : cvec(v2L);
(%o17) matrix([v21],[v22])
(%i18) Bv2 : B . v2;
(%o18) matrix([v22],[0])
(%i19) solns : mcol_solve(v1,Bv2,v2L);
(%o19) [v21 = %r1,v22 = 1]
(%i120) v2L : map('rhs, solns);
(%o120) [%r1,1]
(%i121) v2 : cvec(v2L);
(%o121) matrix([%r1],[1])
(%i22) vrank(B,v2);
(%o22) 2
(%i23) B^^2 . v2;
(%o23) matrix([0],[0])
(%i24) chained(v1,B,v2);
(%o24) true
(%i25) v1;
(%o25) matrix([1],[0])
(%i26) B . v2;
(%o26) matrix([1],[0])
```

The arbitrary scalar **%r1** can be left arbitrary, or simplified, say by letting it be zero.

```
(%i27) v2 : v2, %r1 = 0;
(%o27) matrix([0],[1])
(%i28) vrank(B,v2);
(%o28) 2
(%i29) chained(v1,B,v2);
(%o29) true
```

Our code for the **mbe5.mac** function **mcol_solve** is:

```
mcol_solve(known_col, unknown_col, uk_varL) :=
block([eqns:[], ssL ],
    if not matrixp(known_col) then (
        print(" need column matrices on both sides"),
            return(done)),
    if not matrixp(unknown_col) then (
        print(" need column matrices on both sides"),
        return(done)),
    for j thru length (unknown_col) do
        if unknown_col [j,1] # 0 then eqns : cons(known_col[j,1] = unknown_col[j,1], eqns),
    ssL : solve ( eqns, uk_varL),
    if length(ssL) = 1 then first(ssL)
        else ssL)$
```

5.1.2 Example 1 Using jordan_chain(A, eival)

The contributed package **diag.mac** is loaded automatically when we load **mbe5.mac**, and contains several valuable functions.

The **diag.mac** function **jordan(A)** for this example returns **[[1,2]]**, indicating there is a single eigenvalue $\lambda = 1$ and the single Jordan chain ends (begins) with a maximum vector rank 2 generalized eigenvector.

In a more complicated problem, such as Example 3, in which A is a 6×6 matrix (Bronson, Prob. 9.10) :

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([4,2,1,0,0,0],[0,4,-1,0,0,0],[0,0,4,0,0,0],[0,0,0,4,2,0],
                              [0,0,0,0,4,0],[0,0,0,0,0,7])$
(%i3) eigenvalues(A);
(%o3) [[4,7],[5,1]]
(%i4) jordan(A);
(%o4) [[4,3,2],[7,1]]
```

in which the eigenvalue $\lambda = 4$ has multiplicity $m = 5$ and the eigenvalue $\lambda = 7$ has multiplicity $m = 1$, **jordan(A)**'s first sublist describes the chain content of the $\lambda = 4$ subspace: two chains, the first consisting of chained vectors with ranks 1,2, and 3, and the second chain consisting of chained vectors with ranks 1 and 2, adding up to a total of 5 generalized eigenvectors spanning the subspace.

The **mbe5.mac** interface function **jordan_chain(A, eival)** calls the **diag.mac** function **ModeMatrix** to produce a set a generalized eigenvectors of the matrix **A** corresponding to the eigenvalue **eival**. The elements of the generalized vectors are returned by **jordan_chain** as lists. The vector rank of a list **vL** of generalized eigenvectors can be inspected all at once, as shown below, using the **mbe5.mac** function **vL_rank(B,vL)**.

```
(%i5) A : matrix([1,1],[0,1]);
(%o5) matrix([1,1],[0,1])
(%i6) eigenvalues(A);
(%o6) [[1],[2]]
(%i7) jordan(A);
(%o7) [[1,2]]
(%i8) chain : jordan_chain(A,1);
(%o8) [[1,0],[0,1]]
(%i9) [v1,v2] : map('cvec, chain);
(%o9) [matrix([1],[0]),matrix([0],[1])]
(%i10) B : A - ident(2);
(%o10) matrix([0,1],[0,0])
(%i11) vL_rank(B,[v1,v2]);
(%o11) [1,2]
(%i12) chained(v1,B,v2);
(%o12) true
```

The advantage of using **jordan_chain(A,eival)** is that one obtains a set of generalized eigenvectors, without having to assign values to the arbitrary parameters, such as **%r1**, which **solve** returns with its solutions. Of course, in general, the set will look different from a set derived "by hand", using **mcol_solve**. In this case, the set of generalized eigenvectors happens to be the same set as follows from our decision to use **%r1 = 0** above.

An "almost diagonal" matrix J in Jordan canonical form, related to A by a similarity transformation, is obtained by first forming a "generalized modal matrix" M whose columns are a set of two chained generalized eigenvectors of A. The columns of M are a **canonical basis** for A, and the connection with the Jordan canonical form is $A\,M = M\,J$, which implies $J = M^{-1}\,A\,M$. In this simple case M turns out to be the same as the identity matrix.

$$M = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.6}$$

To illustrate a general approach, we use our **mbe5.mac** function **mcombine(L)**, where **L** is a list of column vectors, to combine the two known generalized eigenvectors into one square matrix, and then the Jordan canonical (normal) form **J**.

```
(%i14) M : mcombine([v1,v2]);
(%o14) matrix([1,0],[0,1])
(%i15) J : M^^(-1) . A . M;
(%o15) matrix([1,1],[0,1])
```

In this simple case, because **M** is the same as the identity matrix **I2**, and the inverse of the identity matrix is the same identity matrix, the matrix product of **A** and **M** give the Jordan canonical form.

```
(%i16) I2 : ident(2);
(%o16) matrix([1,0],[0,1])
(%i17) I2^^(-1);
(%o17) matrix([1,0],[0,1])
(%i18) A . M;
(%o18) matrix([1,1],[0,1])
```

We can also directly use the **mbe5.mac** interface functions **modal_matrix(A)** and **mJordan(A)**, both of which call functions defined in **diag.mac**.

```
(%i19) M : modal_matrix(A);
(%o19) matrix([1,0],[0,1])
(%i20) J : M^^(-1) . A . M;
(%o20) matrix([1,1],[0,1])
(%i21) mJordan(A);
(%o21) matrix([1,1],[0,1])
```

5.2 Example 2: One Chain Per Eigenvalue Example

Suppose A is the 5×5 matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 \\ 6 & 3 & 2 & 0 & 0 \\ 10 & 6 & 3 & 2 & 0 \\ 15 & 10 & 6 & 3 & 2 \end{bmatrix}. \tag{5.7}$$

In the following Maxima session, we use either **eigenvalues** or **eigenvectors** to find that there are two eigenvalues: $\lambda_1 = 1$ with algebraic multiplicity $m_1 = 2$ and geometric multiplicity $m_{g1} = 1$, and $\lambda_2 = 2$ with algebraic multiplicity $m_2 = 3$ and geometric multiplicity $m_{g2} = 1$.

5.2.1 Example 2 Using jordan_chain(A, eival)

We described the origin and use of **jordan_chain(A,eival)** above in Example 1 of our generalized vectors section. For convenience, we define **B1 : A - I5**, in which **I5** is the 5×5 identity matrix, and **B2 : A - 2*I5**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,0,0,0,0],[3,1,0,0,0], [6,3,2,0,0],[10,6,3,2,0],[15,10,6,3,2])$
(%i3) eigenvalues(A);
(%o3) [[1,2],[2,3]]
(%i4) multiplicity(A);
(%o4) [[1,2],[2,3]]
(%i5) jordan(A);
(%o5) [[1,2],[2,3]]
(%i6) I5 : ident(5)$
(%i7) B1 : A - I5;
(%o7) matrix([0,0,0,0,0],[3,0,0,0,0],[6,3,1,0,0],[10,6,3,1,0],[15,10,6,3,1])
(%i8) vrank_numbers(B1,2);
 maximum vector rank =  2
(%o8) [[1,1],[2,1]]
(%i9) chain1 : jordan_chain(A,1);
(%o9) [[0,3,-9,9,-3],[1,0,-15,44,-60]]
(%i10) [x1,x2] : map('cvec, chain1);
(%o10) [matrix([0],[3],[-9],[9],[-3]),matrix([1],[0],[-15],[44],[-60])]
```

```
(%i11) vL_rank(B1,[x1,x2]);
(%o11) [1,2]
(%i12) chained(x1,B1,x2);
(%o12) true
```

so **x2** is a generalized eigenvector of rank 2, corresponding to $\lambda = 1$, chained to **x1** and reflecting a particular choice of a free arbitrary parameter.

We next generate the three generalized eigenvectors corresponding to $\lambda = 2$ and multiplicity $m = 3$.

```
(%i13) B2 : A - 2*I5;
(%o13) matrix([-1,0,0,0,0],[3,-1,0,0,0],[6,3,0,0,0],
              [10,6,3,0,0],[15,10,6,3,0])
(%i14) vrank_numbers(B2,3);
 maximum vector rank =  3
(%o14) [[1,1],[2,1],[3,1]]
(%i15) chain2 : jordan_chain(A,2);
(%o15) [[0,0,0,0,9],[0,0,0,3,6],[0,0,1,0,0]]
(%i16) [y1,y2,y3] : map('cvec, chain2);
(%o16) [matrix([0],[0],[0],[0],[9]),matrix([0],[0],[0],[3],[6]),
        matrix([0],[0],[1],[0],[0])]
(%i17) vL_rank(B2,[y1,y2,y3]);
(%o17) [1,2,3]
(%i18) chained(y1,B2,y2);
(%o18) true
(%i19) chained(y2,B2,y3);
(%o19) true
```

so **y2** is a rank 2 generalized eigenvector corresponding to $\lambda = 2$, and **y3** is a rank 3 generalized eigenvector corresponding to $\lambda = 2$, both corresponding to some unknown choice of free and arbitrary parameters.

The Jordan chained sets of generalized eigenvectors are simply picked out by **jordan_chain(A)** as the columns of the (not unique) modal matrix using **modal_matrix(A)**:

```
(%i20) M : modal_matrix(A);
(%o20) matrix([0,1,0,0,0],[3,0,0,0,0],[-9,-15,0,0,1],
                       [9,44,0,3,0],[-3,-60,9,6,0])
```

5.2.2 Example 2 Using mcol_solve(known_col,unknown_col, ukvarL)

We use the rank 1 ordinary eigenvectors returned by **eigenvectors**, calling the rank 1 vector **x1** for the $\lambda_1 = 1$ case and calling the rank 1 vector **y1** for the $\lambda_2 = 2$ case.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,0,0,0,0],[3,1,0,0,0], [6,3,2,0,0],[10,6,3,2,0],[15,10,6,3,2])$
(%i3) eigenvalues(A);
(%o3) [[1,2],[2,3]]
(%i4) I5 : ident(5)$
(%i5) B1 : A - I5;
(%o5) matrix([0,0,0,0,0],[3,0,0,0,0],[6,3,1,0,0],[10,6,3,1,0],[15,10,6,3,1])
(%i6) vrank_numbers(B1,2);
 maximum vector rank =  2
(%o6) [[1,1],[2,1]]
(%i7) B2 : A - 2*I5;
(%o7) matrix([-1,0,0,0,0],[3,-1,0,0,0],[6,3,0,0,0],[10,6,3,0,0],[15,10,6,3,0])
(%i8) vrank_numbers(B2,3);
 maximum vector rank =  3
(%o8) [[1,1],[2,1],[3,1]]
(%i9) [eivals,vecs] : eigenvectors(A);
(%o9) [[[1,2],[2,3]],[[[0,1,-3,3,-1]],[[0,0,0,0,1]]]]
(%i10) x1L : part(vecs,1,1);
(%o10) [0,1,-3,3,-1]
(%i11) x1 : cvec(x1L);
(%o11) matrix([0],[1],[-3],[3],[-1])
```

```
(%i12) vrank(B1,x1);
(%o12) 1
(%i13) y1L : part(vecs,2,1);
(%o13) [0,0,0,0,1]
(%i14) y1 : cvec(y1L);
(%o14) matrix([0],[0],[0],[0],[1])
(%i15) vrank(B2,y1);
(%o15) 1
```

We then follow steps similar to those in example 1, for each subspace, first for $\lambda = 1$.

```
(%i16) x2L : [x21,x22,x23,x24,x25];
(%o16) [x21,x22,x23,x24,x25]
(%i17) x2 : cvec(x2L);
(%o17) matrix([x21],[x22],[x23],[x24],[x25])
(%i18) B1x2 : B1 . x2;
(%o18) matrix([0],[3*x21],[x23+3*x22+6*x21],[x24+3*x23+6*x22+10*x21],
              [x25+3*x24+6*x23+10*x22+15*x21])
(%i19) solns : mcol_solve(x1,B1x2,x2L);
(%o19) [x21 = 1/3,x22 = (-%r1)-20,x23 = 3*%r1+55,
                                    x24 = -(9*%r1+136)/3,x25 = %r1]
(%i20) x2L : map('rhs, solns), %r1 = 0;
(%o20) [1/3,-20,55,-136/3,0]
(%i21) x2 : cvec (x2L);
(%o21) matrix([1/3],[-20],[55],[-136/3],[0])
(%i22) vrank(B1,x2);
(%o22) 2
(%i23) chained(x1,B1,x2);
(%o23) true
```

so **x2** is a generalized eigenvector of rank 2, corresponding to $\lambda = 1$, chained to **x1** and reflecting a particular choice of the arbitrary parameter **%r1**.

We can then find a set of generalized eigenvectors corresponding to $\lambda = 2$ with $m_2 = 3$ in the same way, starting with **y1**.

```
(%i24) y2L : [y21,y22,y23,y24,y25];
(%o24) [y21,y22,y23,y24,y25]
(%i25) y2 : cvec(y2L);
(%o25) matrix([y21],[y22],[y23],[y24],[y25])
(%i26) B2y2 : B2 . y2;
(%o26) matrix([-y21],[3*y21-y22],[3*y22+6*y21],[3*y23+6*y22+10*y21],
              [3*y24+6*y23+10*y22+15*y21])
(%i27) solns : mcol_solve(y1,B2y2,y2L);
solve: dependent equations eliminated: (4)
(%o27) [y21 = 0,y22 = 0,y23 = 0,y24 = 1/3,y25 = %r2]
(%i28) y2L : map('rhs, solns),%r2 = 0;
(%o28) [0,0,0,1/3,0]
(%i29) y2 : cvec(y2L);
(%o29) matrix([0],[0],[0],[1/3],[0])
(%i30) vrank(B2,y2);
(%o30) 2
(%i31) chained(y1,B2,y2);
(%o31) true
(%i32) y3L : [y31,y32,y33,y34,y35];
(%o32) [y31,y32,y33,y34,y35]
(%i33) y3 : cvec(y3L);
(%o33) matrix([y31],[y32],[y33],[y34],[y35])
(%i34) B2y3 : B2 . y3;
(%o34) matrix([-y31],[3*y31-y32],[3*y32+6*y31],[3*y33+6*y32+10*y31],
              [3*y34+6*y33+10*y32+15*y31])
(%i35) solns : mcol_solve(y2,B2y3,y3L);
solve: dependent equations eliminated: (4)
(%o35) [y31 = 0,y32 = 0,y33 = 1/9,y34 = -2/9,y35 = %r3]
(%i36) y3L : map('rhs, solns), %r3 = 0;
(%o36) [0,0,1/9,-2/9,0]
(%i37) y3 : cvec(y3L);
(%o37) matrix([0],[0],[1/9],[-2/9],[0])
(%i38) vrank(B2,y3);
(%o38) 3
```

```
(%i39) chained(y2,B2,y3);
(%o39) true
```

so **y2** is a rank 2 generalized eigenvector corresponding to $\lambda = 2$, and **y3** is a rank 3 generalized eigenvector corresponding to $\lambda = 2$.

The above steps have produced a set of **basis vectors** for each of the **generalized eigenspaces** of A, which have included choices of the values of three free variables (arbitrary parameters) which **solve** includes in the solutions. Those choices affect the actual values of the vector elements, but any choices produce a valid complete set of basis vectors.

Together, the two chains of generalized eigenvectors span the space of all 5-dimensional column vectors; any such vector can be written as a linear combination of these five generalized eigenvectors.

$$\{\mathbf{x}_1, \mathbf{x}_2\} = \left\{ \begin{bmatrix} 0 \\ 1 \\ -3 \\ 3 \\ -1 \end{bmatrix} \begin{bmatrix} 1/3 \\ -20 \\ 55 \\ -136/3 \\ 0 \end{bmatrix} \right\}, \quad \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\} = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/3 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1/9 \\ -2/9 \\ 0 \end{bmatrix} \right\} \tag{5.8}$$

An "almost diagonal" matrix J in Jordan canonical form, related to A by a similarity transformation, is obtained by first forming a "generalized modal matrix" M whose columns are a set of five generalized eigenvectors of A:

$$M = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{y}_1 & \mathbf{y}_2 & \mathbf{y}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 \\ 1 & -20 & 0 & 0 & 0 \\ -3 & 55 & 0 & 0 & 1/9 \\ 3 & -136/3 & 0 & 1/3 & -2/9 \\ -1 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.9}$$

We use our **mbe5.mac** function **mcombine(L)**, where **L** is a list of column vectors.

```
(%i40) M : mcombine([x1,x2,y1,y2,y3]);
(%o40) matrix([0,1/3,0,0,0],[1,-20,0,0,0],[-3,55,0,0,1/9],
              [3,-136/3,0,1/3,-2/9],[-1,0,1,0,0])
```

The columns of M are a (not unique) **canonical basis** for A, and the connection with the Jordan canonical form is $A\,M = M\,J$, which implies $J = M^{-1}\,A\,M$.

```
(%i41) J : M^^(-1) . A . M;
(%o41) matrix([1,1,0,0,0],[0,1,0,0,0],[0,0,2,1,0],[0,0,0,2,1],[0,0,0,0,2])
(%i42) display2d:true$
(%i43) J;
                                [ 1  1  0  0  0 ]
                                [               ]
                                [ 0  1  0  0  0 ]
                                [               ]
(%o43)                          [ 0  0  2  1  0 ]
                                [               ]
                                [ 0  0  0  2  1 ]
                                [               ]
                                [ 0  0  0  0  2 ]
```

which produces

$$J = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \tag{5.10}$$

The contributed package **diag.mac** has the function **dispJordan(L)**, in which **L** is the list returned by **jordan(A)**, which produces the Jordan canonical form without the necessity of interactively finding a canonical generalized basis of A and constructing the generalized modal matrix M (as done above). The **mbe5.mac** function **mJordan(A)** is our interface to **dispJordan**.

```
(%i44) display2d:false$
(%i45) dispJordan(jordan(A));
(%o45) matrix([1,1,0,0,0],[0,1,0,0,0],[0,0,2,1,0],[0,0,0,2,1],[0,0,0,0,2])
(%i46) mJordan(A);
(%o46) matrix([1,1,0,0,0],[0,1,0,0,0],[0,0,2,1,0],[0,0,0,2,1],[0,0,0,0,2])
```

The generalized modal matrix is not unique, but we get the same Jordan canonical form. Here we use the **mbe5.mac** interface function **modal_matrix(A)**.

```
(%i47) M : modal_matrix(A);
(%o47) matrix([0,1,0,0,0],[3,0,0,0,0],[-9,-15,0,0,1],
                      [9,44,0,3,0],[-3,-60,9,6,0])
(%i48) J : M^^(-1) . A . M;
(%o48) matrix([1,1,0,0,0],[0,1,0,0,0],[0,0,2,1,0],[0,0,0,2,1],[0,0,0,0,2])
```

5.3 Example 3: Two Chains Per Eigenvalue Example

Example 3 is Bronson, Prob. 9.10. The given matrix A is the 6×6 matrix

$$A = \begin{bmatrix} 4 & 2 & 1 & 0 & 0 & 0 \\ 0 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \end{bmatrix}. \tag{5.11}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([4,2,1,0,0,0],[0,4,-1,0,0,0],[0,0,4,0,0,0],[0,0,0,4,2,0],[0,0,0,0,4,0],[0,0,0,0,0,7])$
(%i3) eigenvalues(A);
(%o3) [[4,7],[5,1]]
(%i4) jordan(A);
(%o4) [[4,3,2],[7,1]]
```

in which the eigenvalue $\lambda = 4$ has multiplicity $m = 5$ and the eigenvalue $\lambda = 7$ has multiplicity $m = 1$, **jordan(A)**'s first sublist **[4,3,2]** describes the chain content of the $\lambda = 4$ subspace: two chains, the first consisting of chained vectors with ranks 1,2, and 3, and the second chain consisting of chained vectors with ranks 1 and 2, adding up to a total of 5 generalized eigenvectors corresponding to $\lambda = 4$.

```
(%i5) I6 : ident(6)$
(%i6) B4 : (A - 4*I6)$
(%i7) vrank_max(B4,5);
(%o7) 3
(%i8) vrank_numbers(B4,5);
 maximum vector rank =  3
(%o8) [[1,2],[2,2],[3,1]]
(%i9) nullity(B4);
(%o9) 2
```

The dimension of the nullspace of the matrix **B4** is 2 and we get two ordinary rank 1 eigenvectors corresponding to $\lambda = 4$ using **eigenvectors(A)**.

```
(%i10) eigenvectors(A);
(%o10) [[[4,7],[5,1]],[[[1,0,0,0,0,0],[0,0,0,1,0,0]],[[0,0,0,0,0,1]]]]
```

The geometric multiplicity of the $\lambda = 4$ sector is 2, algebraic multiplicity $m = 5$, so we need 3 generalized eigenvectors with rank greater than 1. Starting with two known eigenvectors of rank 1, one needs to add 3 generalized eigenvectors of rank greater than 1.

```
(%i11) [v1,v2,v3,v4,v5] : map('cvec,jordan_chain(A,4));
(%o11) [matrix([-2],[0],[0],[0],[0],[0]),matrix([1],[-1],[0],[0],[0],[0]),
       matrix([0],[0],[1],[0],[0],[0]),matrix([0],[0],[0],[2],[0],[0]),
       matrix([0],[0],[0],[0],[1],[0])]
(%i12) vL_rank(B4, [v1,v2,v3,v4,v5]);
(%o12) [1,2,3,1,2]
(%i13) vrank(B4,v3);
(%o13) 3
(%i14) B4^^3 . v3;
(%o14) matrix([0],[0],[0],[0],[0],[0])
(%i15) chained (v2,B4,v3);
(%o15) true
(%i16) chained (v1,B4,v2);
(%o16) true
(%i17) vrank(B,v5);
(%o17) 2
(%i18) B4^^2 . v5;
(%o18) matrix([0],[0],[0],[0],[0],[0])
(%i19) chained (v4,B4,v5);
(%o19) true
```

The sixth eigenvector is the rank 1 vector corresponding to $\lambda = 7$ having multiplicity $m = 1$.

```
(%i20) B7 : (A - 7*I6);
(%o20) matrix([-3,2,1,0,0,0],[0,-3,-1,0,0,0],[0,0,-3,0,0,0],[0,0,0,-3,2,0],
              [0,0,0,0,-3,0],[0,0,0,0,0,0])
(%i21) nullity(B7);
(%o21) 1
(%i22) vrank_numbers(B7,1);
 maximum vector rank =  1
(%o22) [[1,1]]
(%i23) [v6] : map('cvec,jordan_chain(A,7));
(%o23) [matrix([0],[0],[0],[0],[0],[1])]
(%i24) v6;
(%o24) matrix([0],[0],[0],[0],[0],[1])
(%i25) vrank(B7,v6);
(%o25) 1
(%i26) B7 . v6;
(%o26) matrix([0],[0],[0],[0],[0],[0])
```

An "almost diagonal" matrix J in Jordan canonical form, related to A by a similarity transformation, is obtained by first forming a "generalized modal matrix" M whose columns are a set of six generalized eigenvectors of A, which we then compare with our interface function **`modal_matrix(A)`**.

```
(%i27) M : mcombine([v1,v2,v3,v4,v5,v6]);
(%o27) matrix([-2,1,0,0,0,0],[0,-1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,2,0,0],
              [0,0,0,0,1,0],[0,0,0,0,0,1])
(%i28) modal_matrix(A);
(%o28) matrix([-2,1,0,0,0,0],[0,-1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,2,0,0],
              [0,0,0,0,1,0],[0,0,0,0,0,1])
```

The columns of M are a (not unique) **canonical basis** for A, and the connection with the Jordan canonical form is $A\,M = M\,J$, which implies $J = M^{-1}\,A\,M$. We compare with using our interface function **`mJordan(A)`**.

```
(%i29) J : M^^(-1) . A . M;
(%o29) matrix([4,1,0,0,0,0],[0,4,1,0,0,0],[0,0,4,0,0,0],[0,0,0,4,1,0],
              [0,0,0,0,4,0],[0,0,0,0,0,7])
(%i30) mJordan(A);
(%o30) matrix([4,1,0,0,0,0],[0,4,1,0,0,0],[0,0,4,0,0,0],[0,0,0,4,1,0],
              [0,0,0,0,4,0],[0,0,0,0,0,7])
```

which shows agreement.

6 Set of Inhomogeneous Linear Algebraic Equations $\mathrm{A}\,\mathrm{x} = \mathrm{b}$

We assume here that A is a given square $n \times n$ matrix, (the "coefficient matrix"), the n-component column vector $\mathbf{b}$ is known and the n-component column vector $\mathbf{x}$ is to be found. The matrix equation

$$A\,\mathbf{x} = \mathbf{b} \tag{6.1}$$

then stands for a system of n linear algebraic equations in n unknowns.

If the square matrix A is singular, that is, $\mathbf{det}\ A = \mathbf{0}$, then solutions of $A\,\mathbf{x} = \mathbf{b}$ either do not exist, or do exist but are not unique.

A system of simultaneous linear algebraic equations may possess no solutions, exactly one solution, or more than one solution. A set of simultaneous equations is **consistent** if it possesses at least one solution; otherwise it is **inconsistent**.

The **augmented matrix** corresponding to (6.1) is the partitioned matrix $[A \,|\, \mathbf{b}\,]$.

Bronson (Ch. 2) presents two theorems and three simplifying operations.

1. The system $A\,\mathbf{x} = \mathbf{b}$ is consistent if and only if the rank of A equals the rank of $[A \,|\, \mathbf{b}\,]$.
2. Denote the rank of A as k, and the number of unknowns as n. If the system $A\,\mathbf{x} = \mathbf{b}$ is consistent, then the solution contains $n-k$ arbitrary scalars, given in Maxima by the difference **`length(A) - rank(A)`**.

Simplifying Operations

Three operations that alter the form of a system of simultaneous linear equations but do not alter its solution set are:
(O1): Interchanging the sequence of two equations.
(O2): Multiplying an equation by a nonzero scalar.
(O3): Adding to one equation a scalar times another equation.

Applying operations O1, O2, and O3 to the system (6.1) is equivalent to applying the elementary row operations to the augmented matrix $[A \,|\, \mathbf{b}\,]$.

Gaussian elimination is an algorithm for applying these operations systematically, to obtain a set of equations that is easy to analyze for consistency and easy to solve if it is consistent.

In linear algebra, **Gaussian elimination** (also known as row reduction) is an algorithm for solving systems of linear equations. This method can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix.

Errors due to rounding can become a problem in **Gaussian elimination**. To minimize the effect of round-off errors, a variety of pivoting strategies have been proposed (partial, scaled, or complete)-pivoting, each modifying Step 3 of the algorithm for transformation of the augmented matrix to row-echelon form. Pivoting strategies are merely criteria for choosing the pivot element.

Gaussian elimination with partial pivoting involves searching the work column of the augmented matrix for the largest element in absolute value appearing in the current work row or a succeeding row. That element becomes the new pivot. To use **partial pivoting**, replace Step 3 of the algorithm for transforming a matrix to row-echelon form with the following:

New Step 3: Beginning with row R and continuing through successive rows, locate the largest element in absolute value appearing in the work column C. Denote the first row in which this element appears as row I.

If I is different from R, interchange rows I and R. Row R will now have, in column C, the largest nonzero element in absolute value appearing in column C of row R or any row suceeding it. This element in row R and column C is called the **pivot**; let P denote its value.

There are more powerful pivoting strategies which require more computations. Since the goal is to avoid significant roundoff error, it is not necessary to find the best pivot element at each stage, but rather to avoid bad ones. Thus, **partial pivoting** is the strategy most often implemented.

The Maxima functions **`triangularize`** and **`echelon`** both use a version of Gaussian elimination (not including partial pivoting) to return the upper triangular form of a matrix, with the same return value, except that in the return value of **`triangularize`**, the leading nonzero coefficient in each row is not normalized to 1. Both of these Maxima functions convert input numbers to rational numbers (ratios of integers), and use integer arithmetic to arrive at the row echelon form (or upper triangular form), producing exact answers in terms of rational numbers, which can then be used by **`solve`** to produce "exact" answers (and floating point answers accurate to 15 digits after the exact answer has been converted to floating point), eliminating the need for using the methods of "partial pivoting." Here is an example of using Maxima to convert floating point numbers to rational numbers using the Maxima function **`rationalize`**.

```
(%i1) fpprintprec:16$
(%i2) rationalize(0.1);
(%o2) 3602879701896397/36028797018963968
(%i3) float(%);
(%o3) 0.1
(%i4) rationalize(1e-20);
(%o4) 6646139978924579/664613997892457936451903530140172288
(%i5) float(%);
(%o5) 1.0e-20
```

(See the two sections below which carry out tests using small values for a parameter in the given matrix.)

Both **`echelon`** and **`triangularize`** immediately switch rows to get a 1 in the working row and column without additional work, and without any knowledge of the size of quantities represented by symbols. Here is an example of the behavior of **`triangularize`** with symbolic and integer inputs:

```
(%i6) triangularize(matrix([a,b,c],[d,e,f]));
(%o6) matrix([a,b,c],[0,a*e-b*d,a*f-c*d])
(%i7) triangularize(matrix([a,b,c],[1,e,f]));
(%o7) matrix([1,e,f],[0,b-a*e,c-a*f])
(%i8) triangularize(matrix([1,b,c],[d,e,f]));
(%o8) matrix([1,b,c],[0,e-b*d,f-c*d])
```

and here is an example of the behavior of **`echelon`** with symbolic and integer inputs.

```
(%i9) echelon(matrix([a,b,c],[d,e,f]));
(%o9) matrix([1,b/a,c/a],[0,1,(a*f-c*d)/(a*e-b*d)])
(%i10) echelon(matrix([a,b,c],[1,e,f]));
(%o10) matrix([1,e,f],[0,1,(a*f-c)/(a*e-b)])
(%i11) echelon(matrix([1,b,c],[d,e,f]));
(%o11) matrix([1,b,c],[0,1,(f-c*d)/(e-b*d)])
```

6.1 Examples

Example 1

Consider solving the system (6.1) if

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 5 \\ 4 & 0 & 5 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ 8 \\ 2 \end{bmatrix}. \tag{6.2}$$

We use the Maxima function **`addcol(A,bL)`** to define the augmented matrix **`Ab`**, given the coefficient matrix **`A`** and the list **`bL`** of the elements of the column vector **`b`**. We find that the rank of A is 3, and the rank of the augmented matrix $[A|\mathbf{b}]$

is 3, so this system is **consistent** and has at least one solution.

Since the rank of A is the same as the number of unknowns (the number of rows of A), there should be a single unique solution. We use **`Cd : echelon(Ab)`** to reduce the partitioned matrix to row-echelon form. We use our homemade functions (available in **`mbe5.mac`**) **`solve_aug(Cd, xL)`** and **`gauss_jordan(A,bL,xL)`** to independently find the solution vector **`xs`**.

The code for these homemade functions is listed in the next section.

Neither of these methods uses **`invert(A)`** or **`determinant(A)`**. We finally check that **`A . xs = b`**. We use the homemade function **`cvec`**, as in **`b : cvec(bL)`** for example, to convert a list into a matrix column vector.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,1,1],[2,3,5],[4,0,5]);
(%o2) matrix([1,1,1],[2,3,5],[4,0,5])
(%i3) rank(A);
(%o3) 3
(%i4) determinant(A);
(%o4) 13
(%i5) bL : [5,8,2];
(%o5) [5,8,2]
(%i6) Ab : addcol(A,bL);
(%o6) matrix([1,1,1,5],[2,3,5,8],[4,0,5,2])
(%i7) rank(Ab);
(%o7) 3
(%i8) Cd : echelon(Ab);
(%o8) matrix([1,0,5/4,1/2],[0,1,5/6,7/3],[0,0,1,-2])
(%i9) xL : [x1,x2,x3];
(%o9) [x1,x2,x3]
(%i10) solve_aug(Cd,xL);
(%o10) [x1 = 3,x2 = 4,x3 = -2]
(%i11) gauss_jordan(A,bL,xL);
(%o11) [x1 = 3,x2 = 4,x3 = -2]
(%i12) xsL : map('rhs,%);
(%o12) [3,4,-2]
(%i13) b : cvec(bL);
(%o13) matrix([5],[8],[2])
(%i14) xs : cvec(xsL);
(%o14) matrix([3],[4],[-2])
(%i15) A . xs;
(%o15) matrix([5],[8],[2])
```

Example 2: A Singular Coefficient Matrix

Consider solving the system (6.1) if

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 3 & 6 & 1 \\ 5 & 7 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ 1 \\ 8 \end{bmatrix}. \tag{6.3}$$

Since the rank of A is 2, and the rank of the augmented matrix $[A|\mathbf{b}]$ is 3, this system is **inconsistent** and no solution exists.

We note that $\mathbf{det}A = 0$, which indicates that the inverse does not exist.

The use of **`echelon`** on the augmented matrix **`Ab`** shows that one of the three equations has the form $0\,x_1 + 0\,x_2 + 0\,x_3 = 1$, which cannot be satisfied.

If we nevertheless try to use **`invert(A)`**, we get an error return, which warns that we are trying to divide by 0.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([2,1,0],[3,6,1], [5,7,1]);
(%o2) matrix([2,1,0],[3,6,1],[5,7,1])
```

```
(%i3) rank(A);
(%o3) 2
(%i4) determinant(A);
(%o4) 0
(%i5) bL : [5,1,8];
(%o5) [5,1,8]
(%i6) Ab : addcol(A,bL);
(%o6) matrix([2,1,0,5],[3,6,1,1],[5,7,1,8])
(%i7) rank(Ab);
(%o7) 3
(%i8) Cd : echelon(Ab);
(%o8) matrix([1,1/2,0,5/2],[0,1,2/9,-13/9],[0,0,0,1])
(%i9) xL : [x1,x2,x3];
(%o9) [x1,x2,x3]
(%i10) solve_aug(Cd,xL);
(%o10) []
(%i11) gauss_jordan(A,bL,xL);
 inconsistent problem: no solution
(%o11) done
(%i12) invert(A);
expt: undefined: 0 to a negative exponent.
 -- an error. To debug this try: debugmode(true);
```

Example 3: A Solution Vector Depending on One Arbitrary Scalar

Consider solving the system (6.1) if

$$A = \begin{bmatrix} 0 & 4 & 1 \\ 2 & 6 & -2 \\ 4 & 8 & -5 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}. \tag{6.4}$$

Since the rank of A is 2, and the rank of the augmented matrix $[A|\mathbf{b}]$ is also 2, this system is **consistent** and at least one solution exists.

Because the rank is 1 less than the number of unknowns, the solution vector will depend on one arbitrary scalar, which we call **t** in our Maxima solution. Since there are an infinite number of values we can choose for t, there are an infinite number of solution vectors.

We also note that $\mathbf{det}A = 0$, which indicates that the inverse does not exist. However, there still exist an infinite number of solution vectors.

We again check the solution vector at the end.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([0,4,1],[2,6,-2],[4,8,-5]);
(%o2) matrix([0,4,1],[2,6,-2],[4,8,-5])
(%i3) rank(A);
(%o3) 2
(%i4) determinant(A);
(%o4) 0
(%i5) bL : [2,3,4];
(%o5) [2,3,4]
(%i6) Ab : addcol(A,bL);
(%o6) matrix([0,4,1,2],[2,6,-2,3],[4,8,-5,4])
(%i7) rank(Ab);
(%o7) 2
(%i8) Cd : echelon(Ab);
(%o8) matrix([1,3,-1,3/2],[0,1,1/4,1/2],[0,0,0,0])
(%i9) xL : [x1,x2,x3];
(%o9) [x1,x2,x3]
(%i10) solve_aug(Cd,xL);
solve: dependent equations eliminated: (3)
(%o10) [x1 = (7*%r1)/4,x2 = -(%r1-2)/4,x3 = %r1]
```

```
(%i11) gauss_jordan(A,bL,xL);
 solution depends on  1  arbitrary parameter(s)
 transfer to solve

solve: dependent equations eliminated: (3)
(%o11) [x1 = (7*%r2)/4,x2 = -(%r2-2)/4,x3 = %r2]
(%i12) %, %r2 = t;
(%o12) [x1 = (7*t)/4,x2 = -(t-2)/4,x3 = t]
(%i13) xsL : map('rhs,%);
(%o13) [(7*t)/4,-(t-2)/4,t]
(%i14) xs : cvec(xsL);
(%o14) matrix([(7*t)/4],[-(t-2)/4],[t])
(%i15) b : cvec(bL);
(%o15) matrix([2],[3],[4])
(%i16) A . xs;
(%o16) matrix([2],[(3*t)/2-(3*(t-2))/2],[2*t-2*(t-2)])
(%i17) ratsimp(%);
(%o17) matrix([2],[3],[4])
```

6.2 Example of a Matrix Containing a Small Parameter ϵ

partial pivoting can usually be avoided

If the coefficient matrix contains small parameters, and we use conventional double precision (16 digit) arithmetic, experience shows that one should try to avoid dividing by a small number, producing very large numbers which dominate the subsequent numbers dealt with, to reduce the chances of losing many significant digits when subtracting two 16 digit numbers which are very close in value.

Using Maxima's symbolic capability, we can avoid the use of partial pivoting by leaving small parameters in symbolic form, and getting an exact symbolic solution. The solutions can then be reduced to numerical form by substitution of numbers for the symbolic parameters.

We consider the issue of "roundoff errors" and the use of partial pivoting with a simple 2×2 example. We let ϵ here be a small number and seek the solution of the pair of equations

$$\begin{aligned} \epsilon\, x_1 + x_2 &= 1 + \epsilon \\ x_1 + x_2 &= 2 \end{aligned}$$

We can easily check that the correct answer is $x_1 = x_2 = 1$. Using matrices, we solve the matrix equation

$$A\,\mathbf{x} = \mathbf{b}, \quad A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1+\epsilon \\ 2 \end{bmatrix} \tag{6.5}$$

for the unknown column vector $\mathbf{x}$.

6.2.1 Maxima Symbolic Solutions

Use of invert(A) with a Symbolic Problem

Using the Maxima function **`invert`**, we use the solution method:

$$\mathbf{x} = A^{-1}\,\mathbf{b}. \tag{6.6}$$

We leave ϵ, represented by **e**, as an undefined symbol. A is nonsingular if $\epsilon \neq 1$, since $\mathbf{det}\ A = \epsilon - 1$, so the inverse exists if $\epsilon \neq 1$. The problem is **consistent** since the rank of the coefficient matrix A is the same as the rank of the augmented matrix. The problem only involves a symbolic parameter and integers, so **`invert(A)`** produces the exact inverse matrix.

The function **`cvec(L)`**, defined in **`mbe5.mac`**, converts a Maxima list into a matrix column vector.

Of course, we could instead use **transpose(matrix(bL))** to generate the matrix column vector **b**. And, if the **eigen.mac** package has been loaded (it loads automatically if we use **eigenvalues** or **eigenvectors**), the function **columnvector(bL)** is available and does the same job, and a shorter "alias" is also available: **covect(bL)**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([e,1],[1,1]);
(%o2) matrix([e,1],[1,1])
(%i3) bL : [1+e,2];
(%o3) [e+1,2]
(%i4) Ab : addcol(A, bL);
(%o4) matrix([e,1,e+1],[1,1,2])
(%i5) rank(A);
(%o5) 2
(%i6) rank(Ab);
(%o6) 2
(%i7) determinant(A);
(%o7) e-1
(%i8) A_inv : invert(A);
(%o8) matrix([1/(e-1),-1/(e-1)],[-1/(e-1),e/(e-1)])
(%i9) b : cvec( bL );
(%o9) matrix([e+1],[2])
(%i10) xs : A_inv . b;
(%o10) matrix([(e+1)/(e-1)-2/(e-1)],[(2*e)/(e-1)-(e+1)/(e-1)])
(%i11) xs : ratsimp(xs);
(%o11) matrix([1],[1])
(%i12) A . xs;
(%o12) matrix([e+1],[2])
```

which produces the correct exact answers.

Use of solve_aug(Ab,xL) with a Symbolic Problem

The function **solve_aug(M,L)** is defined in **mbe5.mac**, and takes as input an augmented matrix (either the starting matrix or a version reduced to a row-echelon form) and a list of the solution variables. This method calls **solve**, and is useful for simple algebraic solution problems which **solve** can handle.

```
(%i13) xL : [x1,x2];
(%o13) [x1,x2]
(%i14) solns : solve_aug(Ab,xL);
(%o14) [x1 = 1,x2 = 1]
(%i15) Cd : echelon(Ab);
(%o15) matrix([1,1,2],[0,1,1])
(%i16) solns : solve_aug(Cd,xL);
(%o16) [x1 = 1,x2 = 1]
(%i17) xsL : map('rhs,solns);
(%o17) [1,1]
(%i18) xs : cvec(xsL);
(%o18) matrix([1],[1])
(%i19) A . xs;
(%o19) matrix([e+1],[2])
(%i20) b;
(%o20) matrix([e+1],[2])
```

Note that **echelon** immediately switched the first and second rows to get a 1 in the $(1,1)$ position. Note also that in this simple problem use of **echelon** causes ϵ to disappear from the problem, and we get exact correct answers whether we use the original augmented matrix or the "partial pivoted," row-echelon form, augmented matrix produced by **echelon**.

Below is our definition of **solve_aug(Mb, xL)**. Note that **mbe5.mac** allows one to use **lme(M)** as an alias for **list_matrix_entries(M)**. Note also this code uses the dot product of two lists, a method illustrated by the example

```
(%i21) x1L : [c,d,e];
(%o21) [c,d,e]
(%i22) x2L : [f,g,h];
(%o22) [f,g,h]
```

```
(%i23) x1L . x2L;
(%o23) e*h+d*g+c*f
(%i24) kill(x1L,x2L);
(%o24) done
```

Here is our code for **solve_aug (Mb, xL)**.

```
/* define a short alias name to use for list_matrix_entries  */
alias(lme, list_matrix_entries)$

/* generate list of n equations from augmented matrix Ab1 with n rows
   and n+1 columns, given n-element list xL1   */
to_solve(Ab1, xL1) :=
block( [rL, eqn1, eqn_list : [], j],
    for j thru length(xL1) do (
        rL : lme (row (Ab1,j)),
        eqn1 : rest(rL,-1) . xL1 = last(rL),
        eqn_list : cons (eqn1, eqn_list)),
    reverse (eqn_list) )$

/* use solve  with the list of equations produced by to_solve */
solve_aug(Bc1,zL1) :=
block([ssL],
    ssL : to_solve(Bc1,zL1),
    ssL : solve(ssL, zL1),
    if length(ssL) = 1 then first(ssL)
        else ssL)$
```

Gauss-Jordan Elimination Method Using gauss_jordan(A,bL,xL)

Our file **mbe5.mac** contains code **gauss_jordan(A, bL, xL)** which implements the Gauss-Jordan elimination method, with **A** being the coefficient matrix, **bL** being a list of the elements of the column vector $\mathbf{b}$, and **xL** being a list of the names of the unknowns.

Our code first makes use of **echelon**, and in the normal case that **rank(A) = length(A)**, makes a further row reduction so that we end up with 1's on the diagonal of the coefficient matrix side of the augmented matrix, and 0's everywhere else (i.e., the identity matrix), in which case the sought for list of unknowns is just the list of the last elements of the augmented matrix rows.

If **rank(A) < length(A)** the solution is passed off to **solve_aug**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([e,1],[1,1]);
(%o2) matrix([e,1],[1,1])
(%i3) solns : gauss_jordan(A, [1+e,2], [x1,x2] );
(%o3) [x1 = 1,x2 = 1]
(%i4) xs : cvec (map('rhs, solns));
(%o4) matrix([1],[1])
(%i5) A . xs;
(%o5) matrix([e+1],[2])
```

Here is our code for **gauss_jordan (A, bL, xL)**.

```
/*
   gauss_jordan(Aa,BbL, XxL)  uses echelon and then normally reduces the
    echelon form further to get not only 1's on the coefficient side diagonal but
        also 0's above the diagonal 1's, so the coefficient side is reduced to a unit
    matrix. The solution vector elements are then the last elements of
         the reduced augented matrix rows.
```

```
    Aa is square matrix of coefficients, BbL is a list of the elements of the  given
        column vector  which equals the  right hand side of the eqn
      Aa . x  =  Bb ;   XxL is a list of unknown variable names.
*/

gauss_jordan(Aa,BbL, XxL) :=
block([nrows, ncols, AaBb,CcDd, nzero_rows, ssL:[] ],
    local(RL),
    nrows : length(Aa),
    ncols : length( transpose(Aa)),

    if nrows # ncols then (
        print(" need square matrix Aa "),
        return(done)),
    if length(BbL) # nrows then (
        print(" length of BbL should be equal to number of rows of Aa"),
        return(done)),

    AaBb : addcol(Aa, BbL),   /* augmented matrix  */
    if rank(Aa) # rank(AaBb) then (
        print(" inconsistent problem: no solution "),
        return(done)),
    CcDd : echelon(AaBb),

    if rank(Aa) < length(Aa) then (
        print(" solution depends on ", length(Aa) - rank(Aa)," arbitrary parameter(s)"),
        print(" transfer to solve_aug "),
        return( solve_aug(CcDd, XxL))),

      /*  case:  rank(Aa) = length(Aa) = number of unknowns:
        assume we have 1's on the diagonal and no 0's on the
        diagonal, replace elements above 1's with 0's                  */

    for j thru nrows do   RL[j] : lme (row (CcDd, j)),

    for j:2 thru nrows do
        for k thru (j-1) do RL[k] : RL[k]  - RL[k][j] * RL[j],

         /* collect last element of rows in a list ssL */
    for j thru nrows do  ssL : cons( last(RL[j]), ssL),
    ssL : reverse (ssL),
    map ("=", XxL, ssL))$
```

Use of "by hand" row-echelon reduction with a Symbolic Problem

For a fourth approach, we use Maxima list arithmetic to transform **Ab** to row-echelon form **by hand** (using Maxima list arithmetic) instead of using **echelon** or **triangularize**. We first define the two rows of the augmented matrix **Ab** as ordinary lists, first using the Maxima function **row(A,n)**, and then using the alias **lme** (defined in **mbe5.mac** as an alias for **list_matrix_entries**) to produce an ordinary Maxima list of the row elements.

```
(%i6) bL : [1+e,2];
(%o6) [e+1,2]
(%i7) Ab : addcol(A, bL);
(%o7) matrix([e,1,e+1],[1,1,2])
(%i8) R1 : lme(row(Ab,1));
(%o8) [e,1,e+1]
(%i9) R2 : lme(row(Ab,2));
(%o9) [1,1,2]
```

Without using partial pivoting, we have $R = 1$, $C = 1$, $P = e$, so transform $R1 \to R1 = R1/P$.

```
(%i10) R1 : R1/e;
(%o10) [1,1/e,(e+1)/e]
```

With $R = 1$, $C = 1$, $N = 2$, $V = 1$, we transform row 2 $R2 \to R2 = R2 - R1$

```
(%i11) R2 : R2 - R1;
(%o11) [0,1-1/e,2-(e+1)/e]
(%i12) R2 : ratsimp(R2);
(%o12) [0,(e-1)/e,(e-1)/e]
```

With $R = 2$, $C = 2$, $P = (e-1)/e$, we transform row 2 $R2 \to R2 = R2/P$.

```
(%i13) R2 : R2/R2[2];
(%o13) [0,1,1]
```

Starting with these two transformed rows, we can now define the transformed augmented matrix **Cd** in two steps (in the second step using **addrow(M,list)**).

```
(%i14) Cd : matrix(R1);
(%o14) matrix([1,1/e,(e+1)/e])
(%i15) Cd : addrow(Cd, R2);
(%o15) matrix([1,1/e,(e+1)/e],[0,1,1])
```

With a list **xL** of symbols for the solution vector defined, we can then use **solve_aug(Cd, xL)**.

```
(%i16) xL : [x1,x2];
(%o16) [x1,x2]
(%i17) solns : solve_aug(Cd,xL);
(%o17) [x1 = 1,x2 = 1]
```

Instead of using **solve_aug**, we can also work directly with the transformed rows of the augmented matrix to find an exact symbolic solution "by hand." The symbols **eq1** and **eq2** stand for the algebraic equations implied by the rows of the row-echelon form **Cd**.

```
(%i18) eq1 : rest(R2,-1) . xL = R2[3];
(%o18) x2 = 1
(%i19) X2 : rhs(%);
(%o19) 1
(%i20) eq2 : rest(R1,-1) . xL = R1[3];
(%o20) x2/e+x1 = (e+1)/e
(%i21) eq2 : eq2, x2 = X2;
(%o21) x1+1/e = (e+1)/e
(%i22) solve(eq2, x1);
(%o22) [x1 = 1]
```

An intermediate route uses the function **to_solve (Cd, xL)**:

```
(%i23) eqnL : to_solve (Cd, xL);
(%o23) [x2/e+x1 = (e+1)/e,  x2 = 1]
(%i24) X2 : rhs (eqnL[2]);
(%o24) 1
(%i25) eq2 : eqnL[1],  x2 = X2;
(%o25) x1+1/e = (e+1)/e
(%i26) solve (eq2, x1);
(%o26) [x1 = 1]
```

6.2.2 Maxima Numerical Solutions

A "by hand" reduction of a numerical augmented matrix, with no partial pivoting

We first define the two rows **R1** and **R2** of the starting augmented matrix **Ab** as ordinary lists. We then use Maxima list arithmetic on the rows to transform **Ab** to row-echelon form by hand, instead of using **echelon**. We then define the transformed augmented matrix **Cd** from the transformed rows. We do not do partial pivoting here. Here is an example using **e = 1e-15**, first defining the rows of the augmented matrix.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) fpprintprec:16$
(%i3) e : 1e-15;
(%o3) 1.0e-15
(%i4) R1 : [e,1,1+e];
(%o4) [1.0e-15,1,1.000000000000001]
(%i5) R2 : [1,1,2];
(%o5) [1,1,2]
```

Without using partial pivoting, we have $R = 1$, $C = 1$, $P = e$, so we transform $R1 \rightarrow R1 = R1/P$.

```
(%i6) R1 : R1/e;
(%o6) [1.0,9.999999999999999e+14,1.000000000000001e+15]
```

With $R = 1$, $C = 1$, $N = 2$, $V = 1$, we transform row 2, $R2 \rightarrow R2 = R2 - R1$

```
(%i7) R2 : R2 - R1;
(%o7) [0.0,-9.999999999999989e+14,-9.999999999999991e+14]
```

With $R = 2$, $C = 2$, $P = (e - 1)/e$, we transform row 2 $R2 \rightarrow R2 = R2/P$.

```
(%i8) R2 : R2/R2[2];
(%o8) [0.0,0.9999999999999999,1.0]
```

We can now define the transformed augmented matrix **Cd** in two steps, one for each row.

```
(%i9) Cd : matrix(R1);
(%o9) matrix([1.0,9.999999999999999e+14,1.000000000000001e+15])
(%i10) Cd : addrow(Cd, R2);
(%o10) matrix([1.0,9.999999999999999e+14,1.000000000000001e+15],
              [0.0,0.9999999999999999,1.0])
```

With a list **xL** of symbols for the solution vector defined, we can then use **solve_aug(Cd, xL)**.

```
(%i11) xL : [x1,x2];
(%o11) [x1,x2]
(%i12) solns : solve_aug(Cd,xL);
(%o12) [x1 = 2, x2 = 1]
```

which gives the **wrong answer** for **x1**. We can also work directly with the transformed rows **R1** and **R2** of the augmented matrix to find a solution "by hand."

```
(%i13) eq1 : rest(R2,-1) . xL = R2[3];
(%o13) 0.9999999999999999*x2 = 1.0
(%i14) soln : solve(eq1,x2);
(%o14) [x2 = 1]
(%i15) X2 : rhs(soln[1]);
(%o15) 1
(%i16) eq2 : rest(R1,-1) . xL = R1[3];
(%o16) 9.999999999999999e+14*x2+1.0*x1 = 1.000000000000001e+15
(%i17) eq2 : eq2, x2 = X2;
(%o17) 1.0*x1+9.999999999999999e+14 = 1.000000000000001e+15
(%i18) soln : solve(eq2,x1);
(%o18) [x1 = 9/8]
(%i19) X1 : rhs(soln[1]);
(%o19) 9/8
```

which gives a **different wrong answer** for **x1**.

A third route uses the function **`to_solve(Cd,xL)`**:

```
(%i20) eqnL : to_solve(Cd,xL);
(%o20) [9.9999999999999e+14*x2+1.0*x1 = 1.000000000000001e+15,
        0.9999999999999999*x2 = 1.0]
(%i21) soln : solve(eqnL[2],x2);
(%o21) [x2 = 1]
(%i22) X2 : rhs(soln[1]);
(%o22) 1
(%i23) eq2 : eqnL[1], x2 = X2;
(%o23) 1.0*x1+9.999999999999999e+14 = 1.000000000000001e+15
(%i24) soln : solve(eq2,x1);
(%o24) [x1 = 9/8]
(%i25) X1 : rhs(soln[1]);
(%o25) 9/8
```

which is **still wrong**.

"by hand" reduction of a numerical matrix, with partial pivoting

We interchange the starting rows:

```
(%i26) R1 : [1,1,2];
(%o26) [1,1,2]
(%i27) R2 : [e,1,1+e];
(%o27) [1.0e-15,1,1.000000000000001]
```

With $R = 1$, $C = 1$, $N = 2$, $V = e$, we transform row 2: $R2 \rightarrow R2 = R2 - e\,R1$

```
(%i28) R2[1];
(%o28) 1.0e-15
(%i29) R2 : R2 - R2[1]*R1;
(%o29) [0.0,0.9999999999999999,0.9999999999999991]
```

With $R = 2$, $C = 2$, $P = (1 - e)$, we transform row 2: $R2 \rightarrow R2 = R2/P$.

```
(%i30) R2 : R2/R2[2];
(%o30) [0.0,1.0,1.0]
```

We can now define the transformed augmented matrix **`Cd`** in two steps, one for each row.

```
(%i31) Cd : matrix(R1);
(%o31) matrix([1,1,2])
(%i32) Cd : addrow(Cd, R2);
(%o32) matrix([1,1,2],[0.0,1.0,1.0])
```

With a list **`xL`** of symbols for the solution vector defined, we can then use **`solve_aug(Cd, xL)`**.

```
(%i33) xL;
(%o33) [x1,x2]
(%i34) solns : solve_aug(Cd,xL);
(%o34) [x1 = 1,x2 = 1]
```

which produces the correct answer, showing the benefit of partial pivoting.

We can repeat this partial pivoted solution for **`e = 1e-30`**.

```
(%i35) e : 1e-30;
(%o35) 1.0e-30
(%i36) R1 : [1,1,2];
(%o36) [1,1,2]
(%i37) R2 : [e,1,1+e];
(%o37) [1.0e-30,1,1.0]
```

```
(%i38) R2 : R2 - R2[1]*R1;
(%o38) [0.0,1.0,1.0]
(%i39) Cd : matrix(R1);
(%o39) matrix([1,1,2])
(%i40) Cd : addrow(Cd, R2);
(%o40) matrix([1,1,2],[0.0,1.0,1.0])
(%i41) solns : solve_aug(Cd,xL);
(%o41) [x1 = 1,x2 = 1]
```

echelon test with numerical e and no partial pivot

We use **`echelon_test1(e,q)`** in **`mbe5.mac`**, with the small floating point number **`e`** as the first argument.

If **`q = "no"`**, then **`echelon`** is tested with no partial-pivot switch of rows 1 and 2. Otherwise, **`echelon`** is tested with a partial pivot switch of rows 1 and 2.

Note that **`echelon`** natively converts floating point numbers to ratios of integers and, generates answers via integer arithmetic, and returns answers in terms of ratios of integers (rational numbers). The rational number answers generated by **`echelon`** or **`triangularize`** should be left in that form when generating equations to be solved by **`solve`**, for maximum accuracy. This means that one should **not** use **`numer:true`** in the code, since that automatically sets the evflag **`float`** to **`true`**.

Using the exact rational number output of **`echelon`**, or **`triangularize`**, in the equations to be solved, we will get the same 15 digit precision floating point answers whether or not we do partial pivoting.

For example, we get the correct answers (to 15 digit accuracy) if **`e = 1e-20`** and we don't use partial pivoting:

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) fpprintprec:16$
(%i3) echelon_test1(1e-20,"no")$
-------------------------------------------
 e =  1.0e-20
test echelon with numerical e and no partial pivot
 start with:
  R1 =  [1.0e-20,1,1.0]
  R2 =  [1,1,2]
 after echelon
  R1 =  [1,10000000000000000000,10000000000000000000]
  R2 =  [0,1,9999999999999999998/9999999999999999999]
 -------------------------------------------
 method 1:  use  solve_aug(Cd, xL)
solns =
        [x1 = 10000000000000000000/9999999999999999999,
         x2 = 9999999999999999998/9999999999999999999]
 float(solns) =  [x1 = 1.0,x2 = 1.0]
 -------------------------------------------
 method 2:   solve by hand
 x1 =  10000000000000000000/9999999999999999999
  x2 =  9999999999999999998/9999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -------------------------------------------
 Method 3:    use function:    to_solve(Cd, xL)
 x1 =  10000000000000000000/9999999999999999999
  x2 =  9999999999999999998/9999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -------------------------------------------
```

If we compare the "no partial pivot" result above with the partial pivot result (next), we see that there is no difference in the ratio of integer answers returned, and hence no difference in the floating point answer.

```
(%i4) echelon_test1(1e-20,"yes")$
-----------------------------------------
 e =  1.0e-20
 test echelon with numerical e and with partial pivot
 start with:
  R1 =  [1,1,2]
  R2 =  [1.0e-20,1,1.0]
 after echelon
  R1 =  [1,1,2]
  R2 =  [0,1,99999999999999999998/99999999999999999999]
 -----------------------------------------------
 method 1:  use  solve_aug(Cd, xL)
solns =
        [x1 = 10000000000000000000/99999999999999999999,
         x2 = 99999999999999999998/99999999999999999999]
 float(solns) =  [x1 = 1.0,x2 = 1.0]
 -----------------------------------------------
 method 2:   solve by hand
 x1 =  10000000000000000000/99999999999999999999
  x2 =  99999999999999999998/99999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -----------------------------------------------
 Method 3:    use function:    to_solve(Cd, xL)
 x1 =  10000000000000000000/99999999999999999999
  x2 =  99999999999999999998/99999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -----------------------------------------------
```

triangularize test with numerical e and no partial pivot

We use **`triangularize_test1(e,q)`** in **`mbe5.mac`**. If **`q = "no"`**, then **`triangularize`** is tested with no partial-pivot switch of rows 1 and 2. Otherwise, **`triangularize`** is tested with a partial pivot switch of rows 1 and 2. Note that **`triangularize`** natively converts floating point numbers to ratios of integers and, generates answers via integer arithmetic, and returns answers in terms of ratios of integers.

Thus we get correct answers, using **`triangularize`** with no partial pivoting, for **`e = 1e-20`**.

```
(%i5) triangularize_test1(1e-20,"no")$
-----------------------------------------
 e =  1.0e-20
test triangularize with numerical e and no partial pivot
 start with:
  R1 =  [1.0e-20,1,1.0]
  R2 =  [1,1,2]
 after triangularize
  R1 =  [1,10000000000000000000,10000000000000000000]
  R2 =  [0,-99999999999999999999,-99999999999999999998]
 -----------------------------------------------
 method 1:  use  solve_aug(Cd, xL)
solns =
        [x1 = 10000000000000000000/99999999999999999999,
         x2 = 99999999999999999998/99999999999999999999]
 float(solns) =  [x1 = 1.0,x2 = 1.0]
 -----------------------------------------------
 method 2:   solve by hand
 x1 =  10000000000000000000/99999999999999999999
  x2 =  99999999999999999998/99999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -----------------------------------------------
 Method 3:    use function:    to_solve(Cd, xL)
 x1 =  10000000000000000000/99999999999999999999
  x2 =  99999999999999999998/99999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -----------------------------------------------
```

Using partial pivoting does not improve the answers.

```
(%i6) triangularize_test1(1e-20,"yes")$
-------------------------------------------
 e =  1.0e-20
 test triangularize with numerical e and with partial pivot
 start with:
  R1 =  [1,1,2]
  R2 =  [1.0e-20,1,1.0]
 after triangularize
  R1 =  [1,1,2]
  R2 =  [0,9999999999999999999,99999999999999999998]
 -------------------------------------------
 method 1:  use  solve_aug(Cd, xL)
solns =
        [x1 = 100000000000000000000/99999999999999999999,
         x2 = 99999999999999999998/99999999999999999999]
 float(solns) =  [x1 = 1.0,x2 = 1.0]
 -------------------------------------------
 method 2:   solve by hand
 x1 =  100000000000000000000/99999999999999999999
  x2 =  99999999999999999998/99999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -------------------------------------------
 Method 3:    use function:    to_solve(Cd, xL)
 x1 =  100000000000000000000/99999999999999999999
  x2 =  99999999999999999998/99999999999999999999
 float(x1) =  1.0  float( x2) =  1.0
 -------------------------------------------
```

Symbolic echelon solution, followed by numerical substitution

If we leave small parameters in symbolic form, and let **`echelon`** and **`solve`** find symbolic solutions (assuming this is possible for a given problem), then it makes more sense to insert the actual numerical values of the parameters at the end to generate a final numerical solution.

The example we are using here is trivial, in this respect, since the use of **`echelon`** for the case of symbolic **`e`** removes the presence of the one small parameter entirely, before we start to solve for the symbolic answers.

Here we use **`echelon_test2(e1,q)`**, in which the calculations are carried out in terms of the symbolic **`e`**, and the final numerical answers are generated via statements like **`x : ev(x, e = e1)`**. The approach used in this code can be adapted to more complicated problems in which the use of **`echelon`** does not remove the presence of the small parameters.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) fpprintprec:16$
(%i3) echelon_test2(1e-14,"no")$
-------------------------------------------
 e1 =  1.0e-14
test echelon with symbolic e and no partial pivot
 start with:
  R1 =  [e,1,e+1]
  R2 =  [1,1,2]
 after echelon
  R1 =  [1,1,2]
  R2 =  [0,1,1]
 -------------------------------------------
 method 1:  use  solve_aug(Cd, xL)
symbolic solns =  [x1 = 1,x2 = 1]
 numerical solns =  [x1 = 1,x2 = 1]
 -------------------------------------------
 method 2:   solve by hand
 symbolic  x1 =  1
 symbolic  x2 =  1
 numerical x1 =  1  numerical  x2 =  1
 -------------------------------------------
```

```
Method 3:    use function:    to_solve(Cd, xL)
symbolic  x1 =  1
 symbolic  x2 =  1
numerical x1 =  1  numerical  x2 =  1
-----------------------------------------------
```

7 Diagonalizable Matrices

The $n \times n$ matrix A is **diagonalizable** if it is **similar** to a diagonal matrix, i.e., if there exists a nonsingular matrix S such that $S^{-1}\,A\,S$ is diagonal. Such a nonsingular matrix S exists if and only if the matrix A has n linearly independent (ordinary, rank 1) eigenvectors.

We know that if all n eigenvalues of A are distinct (no repeats), then n linearly independent rank 1 eigenvectors exist (and are returned by **`eigenvectors`**). If some of the eigenvalues have algebraic multiplicities greater than one, it may occur that n linearly independent rank 1 eigenvectors can still be found (and returned by **`eigenvectors`**).

If the set of rank 1 eigenvectors $\mathbf{v}^1, \ldots, \mathbf{v}^n$ are linearly independent, then the $n \times n$ matrix S composed of the n column vectors $\mathbf{v}^j$

$$S = [\mathbf{v}^1 \quad \cdots \quad \mathbf{v}^n] \tag{7.1}$$

is nonsingular and can be used to transform A into diagonal form. We can write

$$A\,S = [A\,\mathbf{v}^1 \quad \cdots \quad A\,\mathbf{v}^n] = [\lambda_1\,\mathbf{v}^1 \quad \cdots \quad \lambda_n\,\mathbf{v}^n] = S\,\Lambda, \tag{7.2}$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix} \tag{7.3}$$

Since S is nonsingular,

$$S^{-1}\,A\,S = \Lambda, \quad A = S\,\Lambda\,S^{-1}. \tag{7.4}$$

If A is diagonalizable, then S is the same as **`modal_matrix(A)`**, and the resulting diagonal matrix is the same as **`mJordan(A)`**.

The function **`diagp(A)`** (defined in **`mbe5.mac`**) returns **`true`** if the matrix **`A`** is diagonalizable, otherwise returns **`false`**.

If a matrix **`A`** is **not** diagonalizable, the function **`mJordan(A)`** returns the **Jordan Normal Form**, which is an "almost diagonal" matrix which has the eigenvalues on the leading diagonal and either zeroes or ones on the super diagonal. The invertible matrix **`T`** which will produce the Jordan canonical form **`J`** via the relation **`J = invert(T) . A . T`** is returned by **`T : modal_matrix(A)`**.

What About Repeated Eigenvalues?

A matrix **might** have repeated eigenvalues and still be diagonalizable. In this case, the Maxima function **`eigenvectors`** will return a full set of linearly independent rank 1 eigenvectors, which can then be combined to define S. Here is an example of a 3×3 matrix which has one distinct eigenvalue and one repeated eigenvalue, but can be diagonalized.

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}. \tag{7.5}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
```

```
(%i2) A : matrix([1,0,1],[0,1,0],[0,0,2]);
(%o2) matrix([1,0,1],[0,1,0],[0,0,2])
(%i3) normal(A);
(%o3) false
(%i4) diagp(A);
(%o4) true
(%i5) jordan(A);
(%o5) [[1,1,1],[2,1]]
(%i6) [a1,a2] : map ('first, %);
(%o6) [1,2]
(%i7) jordan_chain(A,a1);
(%o7) [[1,0,0],[0,1,0]]
(%i8) [v1,v2] : map ('cvec,%);
(%o8) [matrix([1],[0],[0]),matrix([0],[1],[0])]
(%i9) jordan_chain(A,a2);
(%o9) [[1,0,1]]
(%i10) [v3] : map ('cvec,%);
(%o10) [matrix([1],[0],[1])]
(%i11) S : mcombine([v1,v2,v3]);
(%o11) matrix([1,0,1],[0,1,0],[0,0,1])
(%i12) A_diag : invert(S) . A . S;
(%o12) matrix([1,0,0],[0,1,0],[0,0,2])
(%i13) mJordan(A);
(%o13) matrix([1,0,0],[0,1,0],[0,0,2])
(%i14) modal_matrix(A);
(%o14) matrix([1,0,1],[0,1,0],[0,0,1])
(%i15) is (equal (A . v1, a1*v1));
(%o15) true
(%i16) is (equal (A . v2, a1*v2));
(%o16) true
(%i17) is (equal (A . v3, a2*v3));
(%o17) true
```

We see that **`modal_matrix(A)`** returns **`S`** and **`mJordan(A)`** returns the diagonal matrix **`invert(S) . A . S`**. Thus eigenvalues with multiplicity greater than one **might** have linearly independent rank 1 eigenvectors.

Here is an example of a 2×2 matrix with only one rank 1 eigenvector. Combining the ordinary rank 1 eigenvector with a generalized eigenvector does **not** produce a matrix which transforms A to diagonal form.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,1],[0,1]);
(%o2) matrix([1,1],[0,1])
(%i3) jordan(A);
(%o3) [[1,2]]
(%i4) diagp(A);
(%o4) false
(%i5) eigenvalues(A);
(%o5) [[1],[2]]
(%i6) chain : jordan_chain(A,1);
(%o6) [[1,0],[0,1]]
(%i7) [v1,v2] : map('cvec, chain);
(%o7) [matrix([1],[0]),matrix([0],[1])]
(%i8) B : A - ident(2);
(%o8) matrix([0,1],[0,0])
(%i9) vL_rank(B,[v1,v2]);
(%o9) [1,2]
(%i10) chained(v1,B,v2);
(%o10) true
(%i11) S : mcombine([v1,v2]);
(%o11) matrix([1,0],[0,1])
(%i12) AD : S^^(-1) . A . S;
(%o12) matrix([1,1],[0,1])
```

Here is an example of a 3×3 matrix with three distinct eigenvalues. Combining the three rank 1 eigenvectors returned by **`eigenvectors`** or **`jordan_chain`** to form **`S`** (the same as **`modal_matrix(A)`**), we find the diagonal form using **`invert(S) . A . S`**, which is also returned by **`mJordan(A)`**.

```
(%i13) A : matrix([8,-5,10],[2,1,2],[-4,4,-6]);
(%o13) matrix([8,-5,10],[2,1,2],[-4,4,-6])
(%i14) normal(A);
(%o14) false
(%i15) diagp(A);
(%o15) true
(%i16) jn : jordan(A);
(%o16) [[-2,1],[2,1],[3,1]]
(%i17) [a1,a2,a3] : map ('first, jn);
(%o17) [-2,2,3]
(%i18) [v1] : map ('cvec, jordan_chain(A,a1));
(%o18) [matrix([1],[0],[-1])]
(%i19) [v2] : map ('cvec, jordan_chain(A,a2));
(%o19) [matrix([0],[1],[1/2])]
(%i20) [v3] : map ('cvec, jordan_chain(A,a3));
(%o20) [matrix([1],[1],[0])]
(%i21) S : mcombine([v1,v2,v3]);
(%o21) matrix([1,0,1],[0,1,1],[-1,1/2,0])
(%i22) A_diag : invert(S) . A . S;
(%o22) matrix([-2,0,0],[0,2,0],[0,0,3])
(%i23) mJordan(A);
(%o23) matrix([-2,0,0],[0,2,0],[0,0,3])
(%i24) modal_matrix(A);
(%o24) matrix([1,0,1],[0,1,1],[-1,1/2,0])
```

Among matrices which have possible complex entries, all unitary, all Hermitian, and all skew-Hermitian matrices are diagonalizable.

Among matrices with all real entries, all orthogonal, all symmetric, and all skew-symmetric matrices are diagonalizable.

8 Functions of a Matrix $\mathbf{f}(\mathbf{A})$ and Solution of $\dot{\mathbf{u}}(\mathbf{t}) = \mathbf{A}\,\mathbf{u}(\mathbf{t})$ for Constant $\mathbf{A}$

8.1 Taylor Series Definition of a Function of a Matrix

Quoting Bronson, Ch. 8,

> If a function $f(z)$ of a complex variable z has a Maclaurin series expansion
>
> $$f(z) = \sum_{n=0}^{\infty} a_n\, z^n \tag{8.1}$$
>
> which converges for $|z| < R$, then the matrix series $\sum_{n=0}^{\infty} a_n\, A^n$ converges, provided A is square and each of its eigenvalues has absolute value less than R. In such a case, $f(A)$ is defined as
>
> $$f(A) = \sum_{n=0}^{\infty} a_n\, A^n \tag{8.2}$$
>
> and is called a *well-defined function.* By convention, $A^0 = I$
>
> For example,
>
> $$e^z = 1 + \frac{1}{1!}\,z + \frac{1}{2!}\,z^2 + \ldots = \sum_{j=0}^{\infty} \frac{1}{j!}\,z^j \tag{8.3}$$
>
> converges for all values of z (that is $R = \infty$). Since every eigenvalue λ of any square matrix satisfies the condition that $|\lambda| < \infty$,
>
> $$e^A = I + \frac{1}{1!}\,A + \frac{1}{2!}\,A^2 + \ldots = \sum_{j=0}^{\infty} \frac{1}{j!}\,A^j \tag{8.4}$$

is well defined for all square matrices A.

A second example is the definition of $\cos(A)$. The Maclaurin series for $\cos(z)$ is

$$\cos(z) = 1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \frac{z^6}{6!} \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n\, z^{2n}}{(2n)!} \tag{8.5}$$

which converges for all values of z (that is, $R = \infty$). Every eigenvalue of any square matrix satisfies the condition that $|\lambda| < \infty$, so

$$\cos(A) = I - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n\, A^{2n}}{(2n)!} \tag{8.6}$$

is well defined for every square matrix A.

8.2 Matrix Solution of Initial Value Problems

The initial value problem described by the set of linear, ordinary first order differential equations with constant coefficients represented by the matrix equation

$$\boxed{\dot{\mathbf{u}}(\mathbf{t}) = A\,\mathbf{u}(\mathbf{t}), \quad \mathbf{u}(0) = \mathbf{u}_0} \tag{8.7}$$

in which the elements of the square matrix A are independent of time t, has the solution

$$\boxed{\mathbf{u}(t) = e^{A\,t}\,\mathbf{u}_0, \quad t \geq 0} \tag{8.8}$$

since for $t \to 0$, $e^{A\,t} \to I$, allowing the initial condition to be satisfied, and

$$e^{At} = I + \frac{1}{1!}At + \frac{1}{2!}A^2t^2 + \frac{1}{3!}A^3t^3 + \ldots \tag{8.9}$$

so

$$\frac{d}{dt}e^{At} = 0 + A + \frac{A^2\,2\,t}{2!} + \frac{A^3\,3\,t}{3!} + \ldots = A(I + \frac{1}{1!}At + \frac{1}{2!}A^2t^2 + \frac{1}{3!}A^3t^3 + \ldots) = A\,e^{At} \tag{8.10}$$

8.3 Properties of the "Matrix Exponential" $\mathbf{e^{A\,t}}$

1. Exponential series ($A^0 \equiv I$) with convergence for any matrix A and finite t:

$$e^{A\,t} = \sum_{m=0}^{\infty} (A\,t)^m\,/m! \tag{8.11}$$

2.

$$\frac{d}{dt}e^{A\,t} = A\,e^{A\,t} = e^{A\,t}\,A \tag{8.12}$$

3. If $D = [d_{ij}]$ is a diagonal matrix ($d_{ij} = 0$ for $i \neq j$), then $e^{D\,t}$ is a diagonal matrix with entries $e^{d_{ii}\,t}$. For example,

$$exp\left(\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} t\right) = \begin{bmatrix} e^{at} & 0 \\ 0 & e^{bt} \end{bmatrix} \tag{8.13}$$

4. Special Case ($d_{ii} = r$ for all i):

$$e^{(rI)\,t} = e^{rt}\,I \tag{8.14}$$

5. If $A\,B = B\,A$ (they "commute"), then

$$e^{(A+B)\,t} = e^{A\,t}\,e^{B\,t} = e^{B\,t}\,e^{A\,t} \tag{8.15}$$

Note: If $A\,B \neq B\,A$, then in general

$$e^{(A+B)\,t} \neq e^{A\,t}\,e^{B\,t} \neq e^{B\,t}\,e^{A\,t} \tag{8.16}$$

6. $e^{A\,t}$ is nonsingular, and
$$\left(e^{A\,t}\right)^{-1} = e^{-A\,t} \tag{8.17}$$

7. If S is nonsingular, then
$$e^{\left(SAS^{-1}\right)t} = S\,e^{A\,t}\,S^{-1} \tag{8.18}$$

8. If $\mathbf{v}$ is an eigenvector of A associated with the eigenvalue λ, then
$$e^{A\,t}\,\mathbf{v} = e^{\lambda\,t}\,\mathbf{v}. \tag{8.19}$$

8.4 Using the Maxima Function mat_function(f, A) from Package diag.mac

The contributed Maxima package **diag.mac** has the function **mat_function(f, A)** which attempts to calculate **f(A)**, and we use it here. The Maxima help manual has the information:

```
Returns f(A), where f is an analytic function and A a matrix. This computation is based
  on the Taylor expansion of f. It is not efficient for numerical evaluation, but can give
  symbolic answers for small matrices.
```

The code in **diag.mac** is based on exploiting the Jordan canonical form of a given matrix. A section below presents a brief description and an interactive example of using this method, as well as using the code **eAt_jordan(A)** defined in **mbe5.mac**. In this section we simply show two quick examples of using **mat_function(f,A)**. The code file **mbe5.mac** automatically loads in the package **diag.mac**. Consider the square matrix

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}. \tag{8.20}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([2,4],[1,2]);
(%o2) matrix([2,4],[1,2])
(%i2) mat_function(exp, A);
(%o2) matrix([%e^4/2+1/2,%e^4-1],[%e^4/4-1/4,%e^4/2+1/2])
(%i4) float(%);
(%o4) matrix([27.799075,53.59815],[13.399538,27.799075])
(%i5) mat_function(cos, A);
(%o5) matrix([cos(4)/2+1/2,cos(4)-1],[cos(4)/4-1/4,cos(4)/2+1/2])
(%i6) float(%);
(%o6) matrix([0.17317819,-1.6536436],[-0.41341091,0.17317819])
```

which says that
$$e^{A} = \begin{bmatrix} \frac{1}{2}\left(e^4+1\right) & e^4-1 \\ \frac{1}{4}\left(e^4-1\right) & \frac{1}{2}\left(e^4+1\right) \end{bmatrix} \tag{8.21}$$

and
$$\cos(A) = \begin{bmatrix} \frac{1}{2}\left(\cos(4)+1\right) & \cos(4)-1 \\ \frac{1}{4}\left(\cos(4)-1\right) & \frac{1}{2}\left(\cos(4)+1\right) \end{bmatrix}. \tag{8.22}$$

If A is a diagonal matrix, then e^{A} is also a diagonal matrix, whose diagonal elements have the form e^{α}, with α being the corresponding diagonal element of A. Thus, for example, if

$$A = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}, \tag{8.23}$$

then
$$e^{A} = \begin{bmatrix} e^{\alpha} & 0 \\ 0 & e^{\beta} \end{bmatrix}. \tag{8.24}$$

Here is a concrete example:

```
(%i7) A : matrix([2,0],[0,4]);
(%o7) matrix([2,0],[0,4])
(%i8) mat_function(exp, A);
(%o8) matrix([%e^2,0],[0,%e^4])
```

The Maxima Help Manual has the following example of one way to use **`mat_function`** for integer powers.

```
(%i9) display2d:true$
(%i10) A: matrix([1,2,0], [0,1,0], [1,0,1])$
(%i11) A;
                                  [ 1  2  0 ]
                                  [         ]
(%o11)                            [ 0  1  0 ]
                                  [         ]
                                  [ 1  0  1 ]
(%i12) integer_pow(x) := block ([k], declare (k, integer), x^k)$
(%i13) Atok : mat_function(integer_pow, A);
                               [ 1     2 k     0 ]
                               [                 ]
(%o13)                         [ 0      1      0 ]
                               [                 ]
                               [ k  (k - 1) k  1 ]
(%i14) Atok,k = 20;
                                 [ 1   40   0 ]
                                 [            ]
(%o14)                           [ 0    1   0 ]
                                 [            ]
                                 [ 20  380  1 ]
(%i15) A^^20;
                                 [ 1   40   0 ]
                                 [            ]
(%o15)                           [ 0    1   0 ]
                                 [            ]
                                 [ 20  380  1 ]
```

If we let $M = e^{At}$ and return to the 2×2 example

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}. \tag{8.25}$$

```
(%i16) A : matrix([2,4],[1,2]);
(%o16) matrix([2,4],[1,2])
(%i17) M : mat_function(exp, t*A);
(%o17) matrix([%e^(4*t)/2+1/2,%e^(4*t)-1],[%e^(4*t)/4-1/4,%e^(4*t)/2+1/2])
(%i18) M, t=0;
(%o18) matrix([1,0],[0,1])
```

which shows that $M \to I$ for $t \to 0$, and e^{At} has the form

$$e^{At} = \begin{bmatrix} \frac{1}{2}\left(e^{4t}+1\right) & e^{4t}-1 \\ \frac{1}{4}\left(e^{4t}-1\right) & \frac{1}{2}\left(e^{4t}+1\right) \end{bmatrix} \tag{8.26}$$

8.5 Initial Value Problem Example Using mat_function(exp, t*A)

Here is a simple 2×2 example.

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{8.27}$$

and

$$u(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{8.28}$$

and letting **u0** represent $u(0)$, we get the solution **ut** representing $u(t)$:

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([2,1],[1,2]);
(%o2) matrix([2,1],[1,2])
(%i3) M : mat_function(exp,t*A);
(%o3) matrix([%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
             [%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i4) M, t=0;
(%o4) matrix([1,0],[0,1])
(%i5) u0 : cvec ([0,1] );
(%o5) matrix([0],[1])
(%i6) display2d:true$
(%i7) ut : M . u0;
                                [   3 t     t ]
                                [ %e      %e  ]
                                [ ----- - --- ]
                                [   2       2 ]
(%o7)                           [             ]
                                [   3 t     t ]
                                [ %e      %e  ]
                                [ ----- + --- ]
                                [   2       2 ]
(%i8) ut, t=0;
                                     [ 0 ]
(%o8)                                [   ]
                                     [ 1 ]
```

Hence the analytic solution column vector is

$$u(t) = \frac{1}{2}\begin{bmatrix} e^{3t} - e^{t} \\ e^{3t} + e^{t} \end{bmatrix} \tag{8.29}$$

Here we use **diff** to "check" the solution we have found.

```
(%i9) dudt : diff(ut,t);
                              [     3 t     t ]
                              [ 3 %e      %e  ]
                              [ ------- - --- ]
                              [    2        2 ]
(%o9)                         [               ]
                              [     3 t     t ]
                              [ 3 %e      %e  ]
                              [ ------- + --- ]
                              [    2        2 ]
(%i10) Aut : A . ut;
                       [   3 t         3 t     t      t ]
                       [ %e          %e      %e     %e  ]
                       [ ----- + 2 (----- - ---) + --- ]
                       [   2           2       2      2 ]
(%o10)                 [                               ]
                       [   3 t         3 t     t      t ]
                       [ %e          %e      %e     %e  ]
                       [ ----- + 2 (----- + ---) - --- ]
                       [   2           2       2      2 ]
(%i11) Aut : ratsimp(%);
                              [     3 t     t ]
                              [ 3 %e    - %e  ]
                              [ ------------- ]
                              [       2       ]
(%o11)                        [               ]
                              [     3 t     t ]
                              [ 3 %e    + %e  ]
                              [ ------------- ]
                              [       2       ]
(%i12) is (equal (dudt,Aut));
(%o12)                              true
(%i13) ratsimp(dudt - Aut);
                                    [ 0 ]
(%o13)                              [   ]
                                    [ 0 ]
```

8.6 $\mathbf{e^{A\,t}}$ if A is a Diagonalizable Matrix

We assume A is diagonalizable, with $A = S\Lambda S^{-1}$, in which Λ is a diagonal matrix whose diagonal elements are the eigenvalues of A (with some possible repititions). Positive powers of Λ are then diagonal matrices whose diagonal elements are the corresponding powers of the respective eigenvalues.

$$\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}, \quad \Lambda^2 = \begin{bmatrix} \lambda_1^2 & & 0 \\ & \ddots & \\ 0 & & \lambda_n^2 \end{bmatrix}, \quad \Lambda^3 = \begin{bmatrix} \lambda_1^3 & & 0 \\ & \ddots & \\ 0 & & \lambda_n^3 \end{bmatrix} \quad \dots \tag{8.30}$$

Because $S^{-1}\,S = I$, positive powers of the matrix A can be written as $A^j = S\,\Lambda^j\,S^{-1}$. For example,

$$A^2 = (S\,\Lambda\,S^{-1})\,(S\,\Lambda\,S^{-1}) = S\,\Lambda^2\,S^{-1}. \tag{8.31}$$

In addition, we have, as usual, $A^0 = I$ and $\Lambda^0 = I$. If the given function has the power series expansion

$$f(x) = \sum_{j=0}^{\infty} \alpha_j\,x^j, \tag{8.32}$$

then the definition of $f(A)$ is

$$f(A) = \sum_{j=0}^{\infty} \alpha_j\,A^j = \sum_{j=0}^{\infty} \alpha_j\,S\,\Lambda^j\,S^{-1} = S\sum_{j=0}^{\infty} \alpha_j\,\Lambda^j\,S^{-1}. \tag{8.33}$$

The sum begins with the terms $\alpha_0\,I + \alpha_1\,\Lambda + \alpha_2\,\Lambda^2$, which take the form

$$\begin{bmatrix} \alpha_0 & & 0 \\ & \ddots & \\ 0 & & \alpha_0 \end{bmatrix} + \begin{bmatrix} \alpha_1\,\lambda_1 & & 0 \\ & \ddots & \\ 0 & & \alpha_1\,\lambda_n \end{bmatrix} + \begin{bmatrix} \alpha_2\,\lambda_1^2 & & 0 \\ & \ddots & \\ 0 & & \alpha_2\,\lambda_n^2 \end{bmatrix} \tag{8.34}$$

so the infinite sum becomes

$$\sum_{j=0}^{\infty} \alpha_j\,\Lambda^j = \begin{bmatrix} \alpha_0 + \alpha_1\,\lambda_1 + \alpha_2\,\lambda_1^2 + \dots & & 0 \\ & \ddots & \\ 0 & & \alpha_0 + \alpha_1\,\lambda_n + \alpha_2\,\lambda_n^2 + \dots \end{bmatrix} \tag{8.35}$$

or, finally,

$$\sum_{j=0}^{\infty} \alpha_j\,\Lambda^j = \begin{bmatrix} f(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & f(\lambda_n) \end{bmatrix}. \tag{8.36}$$

We then obtain $f(A)$

$$f(A) = S\begin{bmatrix} f(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & f(\lambda_n) \end{bmatrix} S^{-1}. \tag{8.37}$$

In a similar manner we get $f(A\,t)$

$$f(A\,t) = \sum_{j=0}^{\infty} \alpha_j\,t^j\,A^j = S\sum_{j=0}^{\infty} \alpha_j\,t^j\,\Lambda^j\,S^{-1} = S\begin{bmatrix} f(t\,\lambda_1) & & 0 \\ & \ddots & \\ 0 & & f(t\,\lambda_n) \end{bmatrix} S^{-1}. \tag{8.38}$$

In particular, we get

$$e^{A\,t} = S\begin{bmatrix} e^{\lambda_1\,t} & & 0 \\ & \ddots & \\ 0 & & e^{\lambda_n\,t} \end{bmatrix} S^{-1}. \tag{8.39}$$

Example 1, Real Eigenvalues

We use this method for the diagonalizable matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{8.40}$$

The distinct eigenvalues are $\lambda_1 = 3$ and $\lambda_2 = 1$, and we compare our method with using **mat_function(exp,t*A)**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([2,1],[1,2]);
(%o2) matrix([2,1],[1,2])
(%i3) eigenvectors(A);
(%o3) [[[3,1],[1,1]],[[1,1]],[[1,-1]]]
(%i4) v1 : cvec([1,1]);
(%o4) matrix([1],[1])
(%i5) v2 : cvec([1,-1]);
(%o5) matrix([1],[-1])
(%i6) S : mcombine([v1,v2]);
(%o6) matrix([1,1],[1,-1])
(%i7) modal_matrix(A);
(%o7) matrix([1,1],[1,-1])
(%i8) eAt : S . matrix([exp(3*t),0],[0,exp(t)]) . invert(S);
(%o8) matrix([%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
             [%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i9) eAt2 : mat_function(exp, t*A);
(%o9) matrix([%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
             [%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i10) is(equal(eAt,eAt2));
(%o10) true
(%i11) eAt - eAt2;
(%o11) matrix([0,0],[0,0])
(%i12) expand (2 * eAt);
(%o12) matrix([%e^(3*t)+%e^t,%e^(3*t)-%e^t],[%e^(3*t)-%e^t,%e^(3*t)+%e^t])
```

with the result

$$e^{A\,t} = \frac{1}{2}\begin{bmatrix} e^{3\,t} + e^{t} & e^{3\,t} - e^{t} \\ e^{3\,t} - e^{t} & e^{3\,t} + e^{t} \end{bmatrix}. \tag{8.41}$$

This method for diagonalizable matrices is automated by the function **eAt_diag(A)** defined in **mbe5.mac**.

```
(%i13) eAt1 : eAt_diag(A);
(%o13) matrix([%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
              [%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i14) is(equal(eAt1,eAt2));
(%o14) true
```

The code for **eAt_diag(A)** is

```
eAt_diag(Amatrix) :=
block ( [mJ,mJL:[],D,S  ],
    if not diagp(Amatrix) then (
        print (" not diagonalizable "),
        return(false)),
    mJ :  mJordan(Amatrix),
    for k thru length(mJ) do mJL : cons(mJ[k,k],mJL),
    mJL : exp ( t*reverse (mJL)),
    D : apply ('diag_matrix, mJL),
    S : modal_matrix(Amatrix),
    S . D . invert(S))$
```

Example 2, Complex Eigenvalues

Our second example is

$$A = \begin{bmatrix} -1/2 & 1 \\ -1 & -1/2 \end{bmatrix}. \tag{8.42}$$

We use **a1** and **a2** to represent the two distinct eigenvalues of A.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([-1/2,1],[-1,-1/2]);
(%o2) matrix([-1/2,1],[-1,-1/2])
(%i3) diagp(A);
(%o3) true
(%i4) [vals, vecs] : eigenvectors (A);
(%o4) [[[-(2*%i+1)/2,(2*%i-1)/2],[1,1]],[[1,-%i]],[[1,%i]]]
(%i5) [a1,a2] : vals[1];
(%o5) [-(2*%i+1)/2,(2*%i-1)/2]
(%i6) [v1L, v2L] : [vecs[1][1], vecs[2][1] ];
(%o6) [[1,-%i],[1,%i]]
(%i7) S : mcombine([cvec(v1L),cvec(v2L)]);
(%o7) matrix([1,1],[-%i,%i])
(%i8) eAt : S . matrix([exp(a1*t),0],[0,exp(a2*t)]) . invert(S);
(%o8) matrix([%e^-(((2*%i+1)*t)/2)/2+%e^(((2*%i-1)*t)/2)/2,
              (%i*%e^-(((2*%i+1)*t)/2))/2-(%i*%e^(((2*%i-1)*t)/2))/2],
             [(%i*%e^(((2*%i-1)*t)/2))/2-(%i*%e^-(((2*%i+1)*t)/2))/2,
              %e^-(((2*%i+1)*t)/2)/2+%e^(((2*%i-1)*t)/2)/2])
(%i9) eAt : rectform(eAt);
(%o9) matrix([%e^-(t/2)*cos(t),%e^-(t/2)*sin(t)],
              [-%e^-(t/2)*sin(t),%e^-(t/2)*cos(t)])
(%i10) eAt2 : mat_function(exp,t*A);
(%o10) matrix([%e^-(((2*%i+1)*t)/2)/2+%e^(((2*%i-1)*t)/2)/2,
               (%i*%e^-(((2*%i+1)*t)/2))/2-(%i*%e^(((2*%i-1)*t)/2))/2],
              [(%i*%e^(((2*%i-1)*t)/2))/2-(%i*%e^-(((2*%i+1)*t)/2))/2,
               %e^-(((2*%i+1)*t)/2)/2+%e^(((2*%i-1)*t)/2)/2])
(%i11) eAt2 : rectform(eAt2);
(%o11) matrix([%e^-(t/2)*cos(t),%e^-(t/2)*sin(t)],
              [-%e^-(t/2)*sin(t),%e^-(t/2)*cos(t)])
(%i12) is (equal (eAt, eAt2));
(%o12) true
(%i13) eAt - eAt2;
(%o13) matrix([0,0],[0,0])
```

with the result

$$e^{A\,t} = e^{-\frac{t}{2}} \begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix}. \tag{8.43}$$

Here we check the automated function for diagonalizable matrices:

```
(%i14) eAt1 : eAt_diag(A);
(%o14) matrix([%e^-(((2*%i+1)*t)/2)/2+%e^(((2*%i-1)*t)/2)/2,
               (%i*%e^-(((2*%i+1)*t)/2))/2-(%i*%e^(((2*%i-1)*t)/2))/2],
              [(%i*%e^(((2*%i-1)*t)/2))/2-(%i*%e^-(((2*%i+1)*t)/2))/2,
               %e^-(((2*%i+1)*t)/2)/2+%e^(((2*%i-1)*t)/2)/2])
(%i15) eAt1 : rectform (eAt1);
(%o15) matrix([%e^-(t/2)*cos(t),%e^-(t/2)*sin(t)],
              [-%e^-(t/2)*sin(t),%e^-(t/2)*cos(t)])
(%i16) is (equal (eAt1,eAt2));
(%o16) true
```

Example 3

Our third example uses the matrix

$$A = \begin{bmatrix} 1 & g \\ g & 1 \end{bmatrix}. \tag{8.44}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ( [1, g], [g, 1] );
(%o2) matrix([1,g],[g,1])
(%i3) diagp(A);
(%o3) true
(%i4) eigenvalues(A);
(%o4) [[1-g,g+1],[1,1]]
```

```
(%i5) jordan(A);
(%o5) [[1-g,1],[g+1,1]]
(%i6) [a1,a2] : map ('first,%);
(%o6) [1-g,g+1]
(%i7) [v1] : map ('cvec, jordan_chain(A,a1));
(%o7) [matrix([1],[-1])]
(%i8) [v2] : map ('cvec, jordan_chain(A,a2));
(%o8) [matrix([1],[1])]
(%i9) S : mcombine ([v1,v2] );
(%o9) matrix([1,1],[-1,1])
(%i10) eAt : S . matrix([exp(a1*t),0],[0,exp(a2*t)]) . invert(S);
(%o10) matrix([%e^((g+1)*t)/2+%e^((1-g)*t)/2,%e^((g+1)*t)/2-%e^((1-g)*t)/2],
              [%e^((g+1)*t)/2-%e^((1-g)*t)/2,%e^((g+1)*t)/2+%e^((1-g)*t)/2])
(%i11) eAt2 : mat_function(exp,t*A);
(%o11) matrix([%e^(g*t+t)/2+%e^(t-g*t)/2,%e^(g*t+t)/2-%e^(t-g*t)/2],
              [%e^(g*t+t)/2-%e^(t-g*t)/2,%e^(g*t+t)/2+%e^(t-g*t)/2])
(%i12) is (equal (eAt,eAt2));
(%o12) true
(%i13) ratsimp (eAt - eAt2);
(%o13) matrix([0,0],[0,0])
```

and here we check the automated method for diagonalizable matrices.

```
(%i14) eAt1 : eAt_diag(A);
(%o14) matrix([%e^((g+1)*t)/2+%e^((1-g)*t)/2,%e^((g+1)*t)/2-%e^((1-g)*t)/2],
              [%e^((g+1)*t)/2-%e^((1-g)*t)/2,%e^((g+1)*t)/2+%e^((1-g)*t)/2])
(%i15) is(equal(eAt1,eAt2));
(%o15) true
```

Because our result for **eAt** is proportional to **%e^t**, we multiply **eAt** by **exp(-t)** to work with a simpler matrix we call **M**. In order to use the Maxima function **rectform**, which converts complex exponentials into **sin** and **cos** functions (as does **demoivre**), we make the replacement **g -> %i*h**. After using **rectform**, we then use the reverse replacement **h -> -%i*g**. Maxima, by default, converts **cos(%i*x) -> cosh(x)** and **sin(%i*x) -> %i*sinh(x)**.

```
(%i16) M : expand (exp(-t)*eAt);
(%o16) matrix([%e^(g*t)/2+%e^-(g*t)/2,%e^(g*t)/2-%e^-(g*t)/2],
              [%e^(g*t)/2-%e^-(g*t)/2,%e^(g*t)/2+%e^-(g*t)/2])
(%i17) M : M, g = %i*h;
(%o17) matrix([%e^(%i*h*t)/2+%e^-(%i*h*t)/2,%e^(%i*h*t)/2-%e^-(%i*h*t)/2],
              [%e^(%i*h*t)/2-%e^-(%i*h*t)/2,%e^(%i*h*t)/2+%e^-(%i*h*t)/2])
(%i18) M : rectform(M);
(%o18) matrix([cos(h*t),%i*sin(h*t)],[%i*sin(h*t),cos(h*t)])
(%i19) M : M, h = -%i*g;
(%o19) matrix([cosh(g*t),sinh(g*t)],[sinh(g*t),cosh(g*t)])
```

So we finally have the simpler result:

$$e^{A\,t} = e^{t}\begin{bmatrix}\cosh(g\,t) & \sinh(g\,t)\\ \sinh(g\,t) & \cosh(g\,t)\end{bmatrix} \tag{8.45}$$

8.7 $\mathbf{e^{A\,t}}$ Using the Jordan Canonical Form of a Matrix

Similar Matrices

A matrix A is **similar** to a matrix C if there exists a nonsingular (invertible) matrix S such that

$$A = S^{-1}\,C\,S. \tag{8.46}$$

If A is similar to C, then C is also similar to A and both matrices have the same order and are square. Similar matrices have the same eigenvalues.

Modal Matrix

Given a square matrix A, there always exists a **canonical basis**. A **modal matrix** M for A is a matrix with the same order as A whose columns are all of the vectors of a canonical basis for A. Since a canonical basis is a set of linearly independent vectors, M is invertible. A modal matrix is not unique.

Jordan Block

A **Jordan block** $J_b(k,\lambda)$ is a $k \times k$ matrix whose diagonal elements are all equal to λ and whose superdiagonal elements (those immediately above the main diagonal) are all equal to 1, and all other elements are 0. A Jordan block has the form

$$J_b(k,\lambda) = \begin{bmatrix} \lambda & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda & 1 & \cdots & 0 & 0 \\ 0 & 0 & \lambda & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda & 1 \\ 0 & 0 & 0 & \cdots & 0 & \lambda \end{bmatrix} \tag{8.47}$$

The smallest examples are

$$J_b(1,\lambda) = \lambda, \quad J_b(2,\lambda) = \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix}, \quad J_b(3,\lambda) = \begin{bmatrix} \lambda & 1 & 0 \\ 0 & \lambda & 1 \\ 0 & 0 & \lambda \end{bmatrix}. \tag{8.48}$$

A Jordan block matrix is completely determined by its order and the value of all of its diagonal elements.

Function of a Jordan Block

It can be shown that a matrix function f (i.e., defined by an infinite Taylor series expansion) of a $k \times k$ **Jordan block** is

$$f(J_b(k,\lambda)) = \begin{bmatrix} f(\lambda) & f'(\lambda) & \frac{1}{2!}f''(\lambda) & \cdots & \frac{1}{(k-1)!}f^{(k-1)}(\lambda) \\ 0 & f(\lambda) & f'(\lambda) & \cdots & \frac{1}{(k-2)!}f^{(k-2)}(\lambda) \\ & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \frac{1}{2!}f''(\lambda) \\ & \mathbf{0} & & \ddots & f'(\lambda) \\ & & & & f(\lambda) \end{bmatrix}, \tag{8.49}$$

in which all derivatives are taken with respect to λ. The smallest examples are

$$f(J_b(1,\lambda)) = f(\lambda), \quad f(J_b(2,\lambda)) = \begin{bmatrix} f(\lambda) & f'(\lambda) \\ 0 & f(\lambda) \end{bmatrix} \quad f(J_b(3,\lambda)) = \begin{bmatrix} f(\lambda) & f'(\lambda) & \frac{1}{2}f''(\lambda) \\ 0 & f(\lambda) & f'(\lambda) \\ 0 & 0 & f(\lambda) \end{bmatrix}. \tag{8.50}$$

It can also be shown that if D is a **diagonal matrix**

$$D = \begin{bmatrix} d_{11} & & & \mathbf{0} \\ & d_{22} & & \\ & & \ddots & \\ \mathbf{0} & & & d_{kk} \end{bmatrix}, \tag{8.51}$$

then

$$f(D) = \begin{bmatrix} f(d_{11}) & & & \mathbf{0} \\ & f(d_{22}) & & \\ & & \ddots & \\ \mathbf{0} & & & f(d_{kk}) \end{bmatrix}. \tag{8.52}$$

The Jordan Canonical Form J_c of a Matrix

Every square matrix A is **similar** to a matrix J_c in the (partitioned) Jordan canonical form

$$J_c = \begin{bmatrix} C_1 & & & 0 \\ & C_2 & & \\ & & \ddots & \\ 0 & & & C_m \end{bmatrix} \tag{8.53}$$

in which each C_j is either a diagonal matrix or a Jordan block matrix. Thus it can happen that a Jordan canonical form similar to A is simply a diagonal matrix. If M is a modal matrix for A, then a Jordan canonical form J_c of A is similar to A:

$$A = M\,J_c\,M^{-1}, \quad J_c = M^{-1}\,A\,M. \tag{8.54}$$

The matrix $f(A)$ is defined by an infinite Taylor series expansion. Since $A = M\,J_c\,M^{-1}$, we have

$$A^2 = \left(M\,J_c\,M^{-1}\right)\left(M\,J_c\,M^{-1}\right) = M\,J_c^2\,M^{-1}, \tag{8.55}$$

and

$$A^n = M\,J_c^n\,M^{-1}, \tag{8.56}$$

which implies that

$$f(A) = M\,f(J_c)\,M^{-1}. \tag{8.57}$$

In particular,

$$e^A = M\,e^{J_c}\,M^{-1}, \tag{8.58}$$

and

$$e^{J_c} = \begin{bmatrix} e^{C_1} & & & 0 \\ & e^{C_2} & & \\ & & \ddots & \\ 0 & & & e^{C_m} \end{bmatrix}. \tag{8.59}$$

Example 1

We use the Jordan canonical form route to calculate $\mathbf{e^{At}}$ interactively for the 3×3 matrix (this is prob. 10.41 in Bronson):

$$A = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \quad B = t\,A = \begin{bmatrix} -t & t & 0 \\ 0 & 2\,t & t \\ 0 & 0 & 2\,t \end{bmatrix} \tag{8.60}$$

The matrix A is already in Jordan canonical form with the eigenvalues on the diagonal, $\lambda_1 = -1$, $m_1 = 1$, and $\lambda_2 = 2$, $m_2 = 2$.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([-1,1,0],[0,2,1],[0,0,2]);
(%o2) matrix([-1,1,0],[0,2,1],[0,0,2])
(%i3) eigenvalues(A);
(%o3) [[-1,2],[1,2]]
(%i4) jordan(A);
(%o4) [[-1,1],[2,2]]
(%i5) diagp(A);
(%o5) false
```

The eigenvalue chain contents are returned by **jordan(A)**, which shows that a chain of generalized eigenvectors of maximum rank 2 is needed for $\lambda_2 = 2$, so the matrix A is not diagonalizable.

Continuing with the matrix $B = t\,A$, we see that B is not in a Jordan canonical form, since we have t's on the superdiagonal instead of the needed 1's.

```
(%i6) B : t*A;
(%o6) matrix([-t,t,0],[0,2*t,t],[0,0,2*t])
(%i7) eigenvalues(B);
(%o7) [[-t,2*t],[1,2]]
(%i8) jordan(B);
(%o8) [[-t,1],[2*t,2]]
(%i9) mJordan(B);
(%o9) matrix([-t,0,0],[0,2*t,1],[0,0,2*t])
```

The **mbe5.mac** function **mJordan(B)** displays a Jordan canonical form of B which corresponds to the returned eigenvalue and chain contents of **jordan(B)** (in particular, the order of the "blocks" in the partitioned matrix). We can construct that same Jordan canonical form using the matrix **MB** returned by the **mbe5.mac** function **modal_matrix(B)**.

```
(%i10) MB : modal_matrix(B);
(%o10) matrix([1,-3*t,1],[0,-9*t,0],[0,0,-9])
(%i11) JBc : invert(MB) . B . MB;
(%o11) matrix([-t,0,0],[0,2*t,1],[0,0,2*t])
```

which produces the same matrix as **mJordan(B)**.

Since $f(B) = M f(J_c) M^{-1}$, we first need $f(J_c) = e^{J_c}$. The $(1,1)$ element of e^{J_c} is e^{-t}. The remaining "diagonal elements" of e^{J_c} is the exponential function of a 2×2 matrix in Jordan canonical form, and we can use if

$$J_2(\bar{\lambda}) = \begin{bmatrix} \bar{\lambda} & 1 \\ 0 & \bar{\lambda} \end{bmatrix} \tag{8.61}$$

then

$$f(J_2(\bar{\lambda})) = \begin{bmatrix} f(\bar{\lambda}) & f'(\bar{\lambda}) \\ 0 & f(\bar{\lambda}) \end{bmatrix}, \tag{8.62}$$

where the derivative is with respect to $\bar{\lambda}$. (In our case, $\bar{\lambda} = 2\,t$). Replacing $f(a)$ by e^a, $e^{\bar{\lambda}} = e^{2\,t}$. We also have

$$f'(\bar{\lambda}) = \frac{d}{d\,\bar{\lambda}} e^{\bar{\lambda}} = e^{\bar{\lambda}} = e^{2\,t}. \tag{8.63}$$

In this way we get

$$e^{J_2(\bar{\lambda})} = \begin{bmatrix} e^{2\,t} & e^{2\,t} \\ 0 & e^{2\,t} \end{bmatrix} \tag{8.64}$$

and

$$e^{J_c} = \begin{bmatrix} e^{-t} & 0 & 0 \\ 0 & e^{2\,t} & e^{2\,t} \\ 0 & 0 & e^{2\,t} \end{bmatrix}. \tag{8.65}$$

The **mbe5.mac** function **expJ(k,lambda)** returns the $k \times k$ block matrix $e^{J_k(\lambda)}$, and the **diag.mac** function **diag([eJ1,eJ2,...,eJn])** produces a partitioned matrix from its blocks. We let **eJc** stand for the Jordan canonical matrix form found using **diag**, and then use **eAt : MB . eJc . invert(MB);** to construct the value of $e^B = e^{t\,A}$, and compare the matrix returned with the result of using **mat_function(exp,B)**.

```
(%i12) eJ1 : expJ(1,-t);
(%o12) matrix([%e^-t])
(%i13) eJ2 : expJ(2,2*t);
(%o13) matrix([%e^(2*t),%e^(2*t)],[0,%e^(2*t)])
(%i14) eJc : diag([eJ1,eJ2]);
(%o14) matrix([%e^-t,0,0],[0,%e^(2*t),%e^(2*t)],[0,0,%e^(2*t)])
```

```
(%i15) eAt : MB . eJc . invert(MB);
(%o15) matrix([%e^-t,%e^(2*t)/3-%e^-t/3,(t*%e^(2*t))/3-%e^(2*t)/9+%e^-t/9],
               [0,%e^(2*t),t*%e^(2*t)],[0,0,%e^(2*t)])
(%i16) eAt2 : mat_function(exp,B);
(%o16) matrix([%e^-t,%e^(2*t)/3-%e^-t/3,(t*%e^(2*t))/3-%e^(2*t)/9+%e^-t/9],
               [0,%e^(2*t),t*%e^(2*t)],[0,0,%e^(2*t)])
(%i17) is(equal(eAt,eAt2));
(%o17) true
```

Using the eigenvalue and chain contents returned by **jordan(B)** together with two uses of **expJ(k,lambda)** to define **eJ1** and **eJ2** (in the same order as the structure of **jordan(B)** returns), followed by the use of **diag([eJ1,eJ2])**, allows one to easily determine the matrix e^{J_c}.

This Jordan canonical form route is easy to automate, following the general path shown above interactively, and is contained in the **mbe5.mac** function **eAt_jordan(A)** which returns the matrix $e^{A\,t}$.

```
(%i18) eAt1 : eAt_jordan(A);
(%o18) matrix([%e^-t,%e^(2*t)/3-%e^-t/3,(t*%e^(2*t))/3-%e^(2*t)/9+%e^-t/9],
               [0,%e^(2*t),t*%e^(2*t)],[0,0,%e^(2*t)])
(%i19) is(equal(eAt1,eAt2));
(%o19) true
```

The **diag.mac** function **mat_function(exp, t*A)** follows the same Jordan canonical form route, but in a less transparent manner than **eAt_jordan(A)**.

We can write the result for $e^{A\,t}$ as

$$e^{A\,t} = \frac{1}{9}\begin{bmatrix} 9\,e^{-t}, & -3\,e^{-t}+3\,e^{2t}, & e^{-t}-e^{2t}+3\,t\,e^{2t} \\ 0, & 9\,e^{2t}, & 9\,t\,e^{2t} \\ 0, & 0, & 9\,e^{2t} \end{bmatrix}. \tag{8.66}$$

Example 2

Here is an example of a 5×5 matrix (Bronson, prob. 10.4)

$$A = \begin{bmatrix} 4 & 1 & 1 & 2 & 2 \\ -1 & 2 & 1 & 3 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix} \tag{8.67}$$

in which there are two eigenvalues, the first with multiplicity 1, the second with multiplicity 4. This second eigenvalue needs two separate chains of generalized eigenvectors, one chain having the maximum rank 3, and the second being an ordinary rank 1 eigenvector.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ( [4,1,1,2,2],[-1,2,1,3,0],[0,0,3,0,0],[0,0,0,2,1],[0,0,0,1,2] );
(%o2) matrix([4,1,1,2,2],[-1,2,1,3,0],[0,0,3,0,0],[0,0,0,2,1],[0,0,0,1,2])
(%i3) B : t*A;
(%o3) matrix([4*t,t,t,2*t,2*t],[-t,2*t,t,3*t,0],[0,0,3*t,0,0],[0,0,0,2*t,t],
               [0,0,0,t,2*t])
(%i4) MB : modal_matrix(B);
(%o4) matrix([1,2*t^2,t,0,1],[-3,-2*t^2,t,0,0],[0,0,0,1,7],[4/3,0,0,0,-2],
               [-4/3,0,0,0,-2])
(%i5) eigenvalues(B);
(%o5) [[t,3*t],[1,4]]
(%i6) jordan(B);
(%o6) [[t,1],[3*t,3,1]]
(%i7) eJ1 : expJ(1,t);
(%o7) matrix([%e^t])
```

```
(%i8) eJ2 : expJ(3,3*t);
(%o8) matrix([%e^(3*t),%e^(3*t),%e^(3*t)/2],[0,%e^(3*t),%e^(3*t)],[0,0,
                                                                  %e^(3*t)])
(%i9) eJ3 : expJ(1,3*t);
(%o9) matrix([%e^(3*t)])
(%i10) eJc : diag([eJ1,eJ2,eJ3]);
(%o10) matrix([%e^t,0,0,0,0],[0,%e^(3*t),%e^(3*t),%e^(3*t)/2,0],
              [0,0,%e^(3*t),%e^(3*t),0],[0,0,0,%e^(3*t),0],[0,0,0,0,%e^(3*t)])
(%i11) eAt : expand (MB . eJc . invert(MB));
(%o11) matrix([t*%e^(3*t)+%e^(3*t),t*%e^(3*t),t^2*%e^(3*t)+t*%e^(3*t),
               (7*t^2*%e^(3*t))/4+(11*t*%e^(3*t))/4-(3*%e^(3*t))/8+(3*%e^t)/8,
               (7*t^2*%e^(3*t))/4+(5*t*%e^(3*t))/4+(3*%e^(3*t))/8-(3*%e^t)/8],
              [-t*%e^(3*t),%e^(3*t)-t*%e^(3*t),t*%e^(3*t)-t^2*%e^(3*t),
               (-(7*t^2*%e^(3*t))/4)+(3*t*%e^(3*t))/4+(9*%e^(3*t))/8
                                   -(9*%e^t)/8,
               (-(7*t^2*%e^(3*t))/4)+(9*t*%e^(3*t))/4-(9*%e^(3*t))/8
                                   +(9*%e^t)/8],[0,0,%e^(3*t),0,0],
              [0,0,0,%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
              [0,0,0,%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i12) eAt2 : expand( mat_function(exp,t*A));
(%o12) matrix([t*%e^(3*t)+%e^(3*t),t*%e^(3*t),t^2*%e^(3*t)+t*%e^(3*t),
               (7*t^2*%e^(3*t))/4+(11*t*%e^(3*t))/4-(3*%e^(3*t))/8+(3*%e^t)/8,
               (7*t^2*%e^(3*t))/4+(5*t*%e^(3*t))/4+(3*%e^(3*t))/8-(3*%e^t)/8],
              [-t*%e^(3*t),%e^(3*t)-t*%e^(3*t),t*%e^(3*t)-t^2*%e^(3*t),
               (-(7*t^2*%e^(3*t))/4)+(3*t*%e^(3*t))/4+(9*%e^(3*t))/8
                                   -(9*%e^t)/8,
               (-(7*t^2*%e^(3*t))/4)+(9*t*%e^(3*t))/4-(9*%e^(3*t))/8
                                   +(9*%e^t)/8],[0,0,%e^(3*t),0,0],
              [0,0,0,%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
              [0,0,0,%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i13) is(equal(eAt,eAt2));
(%o13) true
(%i14) eAt1 : eAt_jordan(A);
(%o14) matrix([t*%e^(3*t)+%e^(3*t),t*%e^(3*t),t^2*%e^(3*t)+t*%e^(3*t),
               (7*t^2*%e^(3*t))/4+(11*t*%e^(3*t))/4-(3*%e^(3*t))/8+(3*%e^t)/8,
               (7*t^2*%e^(3*t))/4+(5*t*%e^(3*t))/4+(3*%e^(3*t))/8-(3*%e^t)/8],
              [-t*%e^(3*t),%e^(3*t)-t*%e^(3*t),t*%e^(3*t)-t^2*%e^(3*t),
               (-(7*t^2*%e^(3*t))/4)+(3*t*%e^(3*t))/4+(9*%e^(3*t))/8
                                   -(9*%e^t)/8,
               (-(7*t^2*%e^(3*t))/4)+(9*t*%e^(3*t))/4-(9*%e^(3*t))/8
                                   +(9*%e^t)/8],[0,0,%e^(3*t),0,0],
              [0,0,0,%e^(3*t)/2+%e^t/2,%e^(3*t)/2-%e^t/2],
              [0,0,0,%e^(3*t)/2-%e^t/2,%e^(3*t)/2+%e^t/2])
(%i15) is(equal(eAt1,eAt2));
(%o15) true
(%i16) display2d:true$
(%i17) col(eAt1,1);
                              [     3 t     3 t ]
                              [ t %e    + %e    ]
                              [                 ]
                              [          3 t    ]
                              [    - t %e       ]
(%o17)                        [                 ]
                              [        0        ]
                              [                 ]
                              [        0        ]
                              [                 ]
                              [        0        ]
```

8.8 F.S.S. and "Fundamental Matrix" (F.M.) Definition of $\mathbf{e}^{\mathbf{A}\,\mathbf{t}}$

We review the topic of "fundamental sets of eigenvector solutions" (F.S.S.) of the homogeneous system of scalar first order differential equations, written in matrix form as

$$\dot{\mathbf{x}} = A\,\mathbf{x}. \tag{8.68}$$

A is a square $n \times n$ matrix whose components are independent of time t and $\mathbf{x}(\mathbf{t})$ is a time dependent n-component column vector. The column vectors $\mathbf{v}^j$ are constant n-component eigenvectors of A, and the λ_j are the corresponding

scalar eigenvalues (either real or complex) which satisfy the equation

$$A\,\mathbf{v}^j = \lambda_j\,\mathbf{v}^j, \quad 1 \le j \le n \tag{8.69}$$

Each of the set of eigenvalues λ_j satisfy the "characteristic equation"

$$p(\lambda) = 0, \tag{8.70}$$

where

$$p(\lambda) = \mathbf{det}(A - \lambda\,I). \tag{8.71}$$

in which I is the $n \times n$ identity matrix. The "fundamental theorem of algebra" states that if the roots of (8.70) are counted with multiplicities, then $p(\lambda)$ has exactly n roots $\lambda_1, \ldots, \lambda_n$ and

$$p(\lambda) = (-1)^n\,(\lambda - \lambda_1)\ldots(\lambda - \lambda_n). \tag{8.72}$$

We first illustrate the use of the fundamental set of solutions and the "fundamental matrix" method in the case in which the eigenvalues of A are all distinct.

If $\lambda_1, \ldots, \lambda_n$ are n distinct eigenvalues of A, and if $\mathbf{v^1}, \ldots, \mathbf{v^n}$ are the corresponding n linearly independent eigenvectors, then

$$\mathbf{x}^j(t) = e^{\lambda_j\,t}\,\mathbf{v}^j, \quad 1 \le j \le n, \tag{8.73}$$

are a **fundamental set of solutions** of (8.68), since

$$\frac{d}{d\,t}\,e^{\lambda_j\,t}\,\mathbf{v}^j = \lambda_j\,\left(e^{\lambda_j\,t}\,\mathbf{v}^j\right) \tag{8.74}$$

and

$$A\,\left(e^{\lambda_j\,t}\,\mathbf{v}^j\right) = e^{\lambda_j\,t}\,A\,\mathbf{v}^j = \lambda_j\,\left(e^{\lambda_j\,t}\,\mathbf{v}^j\right). \tag{8.75}$$

"Fundamental Matrix" (F.M.) Definition of $\mathbf{e^{A\,t}}$ for Distinct Eigenvalue Case

The **Fundamental Matrix** (F.M.), here denoted as $X(t)$, is the square matrix whose columns are the fundamental set of column vector solutions $\mathbf{x}^{(j)}(t)$ defined by (8.73) for the distinct eigenvalues case:

$$X(t) = \left[\mathbf{x}^{(1)}(t), \ldots, \mathbf{x}^{(n)}(t)\right]. \tag{8.76}$$

The **general solution** $\mathbf{x}(t)$ of the matrix differential equation (8.68), $\dot{\mathbf{x}} = A\,\mathbf{x}$ is a linear combination of the fundamental solutions $\mathbf{x}^{(j)}(t)$ which we can write in terms of a set of constants c_j and the fundamental matrix $X(t)$ and a column vector $\mathbf{c}$ whose elements are the c_j:

$$\mathbf{x}(t) = \sum_j \mathbf{x}^{(j)}\,c_j = X(t)\,\mathbf{c}. \tag{8.77}$$

A particular solution (assume) must have the initial value $\mathbf{x}_0$, so that

$$\mathbf{x}(t=0) = X(0)\,\mathbf{c} = \mathbf{x}_0 \tag{8.78}$$

which can be solved for the elements of the column vector $\mathbf{c}$:

$$\mathbf{c} = (X(0))^{-1}\,\mathbf{x}_0, \tag{8.79}$$

which gives for the particular solution:

$$\mathbf{x}(t) = X(t)\,(X(0))^{-1}\,\mathbf{x}_0 = e^{A\,t}\,\mathbf{x}_0, \quad t \ge 0. \tag{8.80}$$

This provides the **fundamental matrix** definition of $e^{A\,t}$ (we will also be able to use this definition if A has eigenvalues with multiplicity greater than one, but the definition of the fundamental matrix $X(t)$ is altered):

$$e^{A\,t} = X(t)\,(X(0))^{-1}, \quad t \ge 0 \tag{8.81}$$

Interactive Example for the Distinct Eigenvalues Case

The interactive session which follows assumes a 3×3 matrix A which has three distinct real eigenvalues, and uses the fundamental set of solutions and fundamental matrix method to find the matrix e^{At}.

$$A = \begin{bmatrix} 8 & -5 & 10 \\ 2 & 1 & 2 \\ -4 & 4 & -6 \end{bmatrix}. \tag{8.82}$$

We can use the parts of the return value of **`eigenvectors(A)`** to give the names **`a1`**, **`a2`**, and **`a3`** to the eigenvalues and the names **`v1`**, **`v2`**, and **`v3`** to the corresponding eigenvectors (up to an overall constant). We check the linear independence of the three eigenvectors, using the condition $\mathbf{det}(M) \neq 0$, in which M is a matrix whose columns are the three eigenvectors. We use **`mcombine([v1,v2,v3])`** from **`mbe5.mac`** to create M. We can then define the time dependent fundamental set of solutions (F.S.S.) of (8.68), denoted here by **`x1`**, **`x2`**, and **`x3`**, and the fundamental matrix (F.M.) $X(t)$, denoted here by **`Xt`**, which is a matrix whose columns are the fundamental set of solutions.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([8,-5,10],[2,1,2],[-4,4,-6]);
(%o2) matrix([8,-5,10],[2,1,2],[-4,4,-6])
(%i3) eigenvalues(A);
(%o3) [[-2,2,3],[1,1,1]]
(%i4) jordan(A);
(%o4) [[-2,1],[2,1],[3,1]]
(%i5) [a1,a2,a3] : map ('first, %);
(%o5) [-2,2,3]
(%i6) [v1] : map ('cvec, jordan_chain(A,a1));
(%o6) [matrix([1],[0],[-1])]
(%i7) [v2] : map ('cvec, jordan_chain(A,a2));
(%o7) [matrix([0],[1],[1/2])]
(%i8) [v3] : map ('cvec, jordan_chain(A,a3));
(%o8) [matrix([1],[1],[0])]
(%i9) determinant (mcombine ([v1,v2,v3]));
(%o9) 1/2
(%i10) x1 : exp(a1*t) * v1;
(%o10) matrix([%e^-(2*t)],[0],[-%e^-(2*t)])
(%i11) x2 : exp(a2*t) * v2;
(%o11) matrix([0],[%e^(2*t)],[%e^(2*t)/2])
(%i12) x3 : exp(a3*t) * v3;
(%o12) matrix([%e^(3*t)],[%e^(3*t)],[0])
(%i13) Xt : mcombine([x1,x2,x3]);
(%o13) matrix([%e^-(2*t),0,%e^(3*t)],[0,%e^(2*t),%e^(3*t)],
              [-%e^-(2*t),%e^(2*t)/2,0])
```

We can now compute the fundamental matrix definition of e^{At} 8.81 and then compute the particular solution $\mathbf{x}(t)$ corresponding to the the assumed initial values

$$\mathbf{x}(0) = [1, 1, 1]^{\mathrm{T}}. \tag{8.83}$$

Note that the upper case **`Xt`** is the fundamental matrix at time t and **`X0`** is the fundamental matrix at the initial time $t = 0$.

```
(%i14) X0 : Xt, t = 0;
(%o14) matrix([1,0,1],[0,1,1],[-1,1/2,0])
(%i15) eAt : Xt . invert(X0);
(%o15) matrix([2*%e^(3*t)-%e^-(2*t),%e^-(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^-(2*t)],
              [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^(2*t)],
              [%e^-(2*t)-%e^(2*t),%e^(2*t)-%e^-(2*t),2*%e^-(2*t)-%e^(2*t)])
(%i16) eAt, t = 0;
(%o16) matrix([1,0,0],[0,1,0],[0,0,1])
(%i17) x0 : cvec( [1,1,1] );
(%o17) matrix([1],[1],[1])
(%i18) xt : eAt . x0;
(%o18) matrix([3*%e^(3*t)-2*%e^-(2*t)],[3*%e^(3*t)-2*%e^(2*t)],
              [2*%e^-(2*t)-%e^(2*t)])
(%i19) xt, t = 0;
(%o19) matrix([1],[1],[1])
(%i20) xt, t=2, numer;
(%o20) matrix([1210.2497],[1101.0901],[-54.561519])
```

We can use **mat_function (exp, t*A)** to check on the matrix **eAt**.

```
(%i21) eAt2 : expand (mat_function(exp,t*A));
(%o21) matrix([2*%eˆ(3*t)-%eˆ-(2*t),%eˆ-(2*t)-%eˆ(3*t),
               2*%eˆ(3*t)-2*%eˆ-(2*t)],
              [2*%eˆ(3*t)-2*%eˆ(2*t),2*%eˆ(2*t)-%eˆ(3*t),
               2*%eˆ(3*t)-2*%eˆ(2*t)],
              [%eˆ-(2*t)-%eˆ(2*t),%eˆ(2*t)-%eˆ-(2*t),2*%eˆ-(2*t)-%eˆ(2*t)])
(%i22) is (equal (eAt, eAt2));
(%o22) true
```

F.S.S. Method for Repeated Eigenvalues

If λ is an eigenvalue of A with algebraic multiplicity m, then there exists an integer p, $0 < p \leq m$, such that

$$dimNull((A - \lambda\, I)^p) = m$$
$$dimNull((A - \lambda\, I)^{p-1}) < m$$

Any nonzero vector $\mathbf{v}$ in the space $Null((A - \lambda\, I)^p)$ is a **generalized eigenvector** for λ with the property

$$(A - \lambda\, I)^p\, \mathbf{v} = \mathbf{0}. \tag{8.84}$$

Now let $A = \lambda\, I + B$, so

$$e^{A\,t} = e^{\lambda\, I\, t + B\, t} = e^{\lambda\, t}\, e^{B\, t} \tag{8.85}$$

where $B = A - \lambda\, I$. Then

$$e^{A\,t}\, \mathbf{v} = e^{\lambda\, t}\, e^{B\, t}\, \mathbf{v}. \tag{8.86}$$

But because of (8.84) there are only a finite number of non-zero terms in the expansion of $e^{B\,t}$ when acting on $\mathbf{v}$:

$$e^{B\,t}\, \mathbf{v} = \sum_{j=0}^{p-1} (t^j/j!)\, B^j\, \mathbf{v} \tag{8.87}$$

and hence we recover an exact matrix expression.

Now let $\mathbf{v^1}, \ldots, \mathbf{v^m}$ be a basis of $Null(A - \lambda\, I)^p$. Then the set of column vectors

$$\mathbf{x}^i = e^{\lambda\, t} \sum_{j=0}^{p-1} (t^j/j!)\, (A - \lambda\, I)^j\, \mathbf{v}^i, \quad 1 \leq i \leq m, \tag{8.88}$$

are m linearly independent solutions of $\dot{\mathbf{x}} = A\,\mathbf{x}$ corresponding to the eigenvalue λ which has algebraic multiplicity m.

The three **mbe5.mac** Maxima functions **vrank(B,v)**, which returns the vector rank of **v** relative to **B**, **vrank_max(B,m)** which returns the integer **p**, and **nullity(Bˆˆp)** which returns **m**, and the **linearalgebra.mac** package function **nullspace(Bˆˆp)** which returns **span(v1,...,vm)**, a set of **m** linearly independent generalized eigenvectors of the m dimensional subspace $Null((B^p))$, can be used to implement this method, as shown in the examples and functions shown below.

Here is a simple 2×2 matrix which has the single eigenvalue $\lambda = -1$ with (of course) multiplicity $m = 2$.

$$A = \begin{bmatrix} 1 & 4 \\ -1 & -3 \end{bmatrix}. \tag{8.89}$$

We compare the matrix $e^{A\,t}$ predicted by the fundamental sets of solutions method with the value predicted by **mat_function(exp,t*A)**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,4],[-1,-3]);
(%o2) matrix([1,4],[-1,-3])
(%i3) eigenvalues(A);
(%o3) [[-1],[2]]
(%i4) eigenvectors(A);
(%o4) [[[-1],[2]],[[1,-1/2]]]
(%i5) I2 : ident(2);
(%o5) matrix([1,0],[0,1])
(%i6) B : A + I2;
(%o6) matrix([2,4],[-1,-2])
(%i7) p : vrank_max(B,2);
(%o7) 2
(%i8) nullity(B^^p);
(%o8) 2
(%i9) nspc : nullspace(B^^p);
(%o9) span(matrix([0],[1]),matrix([1],[0]))
(%i10) [v1,v2] : nspc, span = "[";
(%o10) [matrix([0],[1]),matrix([1],[0])]
(%i11) v1;
(%o11) matrix([0],[1])
(%i12) v2;
(%o12) matrix([1],[0])
(%i13) vrank(B,v1);
(%o13) 2
(%i14) vrank(B,v2);
(%o14) 2
(%i15) eBt : I2 + t*B;
(%o15) matrix([2*t+1,4*t],[-t,1-2*t])
(%i16) x1 : %e^(-t)* (eBt . v1);
(%o16) matrix([4*t*%e^-t],[(1-2*t)*%e^-t])
(%i17) x2 : %e^(-t)* (eBt . v2);
(%o17) matrix([(2*t+1)*%e^-t],[-t*%e^-t])
(%i18) Xt : mcombine([x1,x2]);
(%o18) matrix([4*t*%e^-t,(2*t+1)*%e^-t],[(1-2*t)*%e^-t,-t*%e^-t])
(%i19) X0 : Xt, t=0;
(%o19) matrix([0,1],[1,0])
(%i20) eAt : Xt . invert(X0);
(%o20) matrix([(2*t+1)*%e^-t,4*t*%e^-t],[-t*%e^-t,(1-2*t)*%e^-t])
(%i21) % / exp(-t);
(%o21) matrix([2*t+1,4*t],[-t,1-2*t])
(%i22) eAt2 : mat_function(exp,t*A);
(%o22) matrix([2*t*%e^-t+%e^-t,2*t*(2*%e^-t-%e^-t/t)+2*%e^-t],
              [-t*%e^-t,-t*(2*%e^-t-%e^-t/t)])
(%i23) eAt2 / exp(-t);
(%o23) matrix([(2*t*%e^-t+%e^-t)*%e^t,(2*t*(2*%e^-t-%e^-t/t)+2*%e^-t)*%e^t],
              [-t,-t*(2*%e^-t-%e^-t/t)*%e^t])
(%i24) expand(%);
(%o24) matrix([2*t+1,4*t],[-t,1-2*t])
(%i25) is(equal(eAt,eAt2));
(%o25) true
(%i26) eAt - eAt2;
(%o26) matrix([(2*t+1)*%e^-t-2*t*%e^-t-%e^-t,
               (-2*t*(2*%e^-t-%e^-t/t))+4*t*%e^-t-2*%e^-t],
              [0,t*(2*%e^-t-%e^-t/t)+(1-2*t)*%e^-t])
(%i27) expand(%);
(%o27) matrix([0,0],[0,0])
```

We have a function **eAt_FSS(A)** defined in **mbe5.mac** which automates the above method.

```
eAt_FSS(A) :=
block([XXt, XX0],
    XXt : mcombine (FSS(A)),
    XX0 : ev (XXt, t = 0),
    XXt . invert(XX0) )$
```

with the application to the above simple example:

```
(%i28) eAt1 : eAt_FSS(A);
(%o28) matrix([(2*t+1)*%eˆ-t,4*t*%eˆ-t],[-t*%eˆ-t,-(2*t-1)*%eˆ-t])
(%i29) eAt1 / exp(-t);
(%o29) matrix([2*t+1,4*t],[-t,1-2*t])
(%i30) is(equal(eAt,eAt1));
(%o30) true
```

The function **eAt_FSS(A)**, in the code shown above, calls the function **FSS(A)**.

```
/*  FSS(A) calls FSS_sector(A,eival,mult) for each distinct eigenvalue eival
    and returns the list [x1[t], x2[t], ..., xn[t] ] implied by the n x n matrix A
*/
FSS(Amatrix) :=
block( [ eivals, mvals, xjtL : [ ]  ],

    [eivals, mvals] : eigenvalues(Amatrix),

    for j thru length(eivals) do (
        xtL : FSS_sector(Amatrix,eivals[j],mvals[j]),
        for k thru length(xtL) do xjtL : cons(xtL[k], xjtL)),
    reverse(xjtL))$
```

The function **FSS(A)**, in the code shown above, calls **FSS_sector(A,lambda,m)**.

```
/* FSS_sector(A,lambda,m) returns a list of time dependent
    linearly independent solutions xj(t) (as matrix column vectors)
        of the equation dxj(t)/dt = A . xj(t) for the sector eival = lambda,
        multiplicity = m, given the square matrix A .

        calls nullspace (Bˆˆp) where p is the smallest positive
        integer such that nullity(Bˆˆp) = m, ie., the
        dimension of Null Bˆˆp is equal to the multiplicity
        of the eigenvalue lambda.
        */

FSS_sector(Amatrix,eival,multiplicity) :=
block([Nn,In,Bb,xxL:[ ], Pp, eBbt, nspc  ],
    local(vv),
    Nn : length(Amatrix),
    In : ident(Nn),
    Bb : Amatrix - eival*In,
    Pp : vrank_max(Bb, multiplicity),
    eBbt : sum( (tˆj/j!) * Bbˆˆj, j,0, Pp - 1),

    nspc : nullspace(Bbˆˆ Pp),
    for j thru multiplicity do  vv[j] : part( nspc,j ),

    for j thru multiplicity do (
        vv[j] : ratsimp ( exp( eival*t ) * eBbt . vv[j] ),
        xxL : cons(vv[j], xxL)),
    reverse (xxL))$
```

Here is an example of a 3×3 matrix A which has one repeated eigenvalue.

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}. \tag{8.90}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
```

```
(%i2) A : matrix([0,1,1],[1,0,1],[1,1,0]);
(%o2) matrix([0,1,1],[1,0,1],[1,1,0])
(%i3) eigenvalues(A);
(%o3) [[2,-1],[1,2]]
(%i4) eAt : eAt_FSS(A);
(%o4) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
             [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
             [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i5) eAt2 : mat_function(exp,t*A);
(%o5) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
             [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
             [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i6) is(equal(eAt,eAt2));
(%o6) true
(%i7) eAt - eAt2;
(%o7) matrix([0,0,0],[0,0,0],[0,0,0])
```

If you do this example "by hand", you need to define the fundamental matrix $X(t)$, represented here by **Xt**, by first joining the basis vectors from the two sectors (corresponding to the two eigenvalues) into one list of three basis vectors.

```
(%i8) L1 : FSS_sector(A,2,1);
(%o8) [matrix([3*%e^(2*t)],[3*%e^(2*t)],[3*%e^(2*t)])]
(%i9) L2 : FSS_sector(A,-1,2);
(%o9) [matrix([-%e^-t],[%e^-t],[0]),matrix([0],[%e^-t],[-%e^-t])]
(%i10) Xt : mcombine (flatten( cons (L1,L2) ));
(%o10) matrix([3*%e^(2*t),-%e^-t,0],[3*%e^(2*t),%e^-t,%e^-t],[3*%e^(2*t),0,
                                                              -%e^-t])
(%i11) X0 : Xt, t=0;
(%o11) matrix([3,-1,0],[3,1,1],[3,0,-1])
(%i12) eAt : Xt . invert (X0);
(%o12) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i13) eAt - eAt2;
(%o13) matrix([0,0,0],[0,0,0],[0,0,0])
```

We can also use the automated FSS/FM method with a matrix, all of whose eigenvalues are distinct.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([8,-5,10],[2,1,2],[-4,4,-6]);
(%o2) matrix([8,-5,10],[2,1,2],[-4,4,-6])
(%i3) eigenvalues(A);
(%o3) [[-2,2,3],[1,1,1]]
(%i4) eAt : eAt_FSS(A);
(%o4) matrix([2*%e^(3*t)-%e^-(2*t),%e^-(2*t)-%e^(3*t),2*%e^(3*t)-2*%e^-(2*t)],
             [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
              2*%e^(3*t)-2*%e^(2*t)],
             [%e^-(2*t)-%e^(2*t),%e^(2*t)-%e^-(2*t),2*%e^-(2*t)-%e^(2*t)])
(%i5) eAt2 : mat_function(exp,t*A);
(%o5) matrix([2*%e^(3*t)-%e^-(2*t),%e^-(2*t)-%e^(3*t),2*%e^(3*t)-2*%e^-(2*t)],
             [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
              2*%e^(3*t)-2*%e^(2*t)],
             [%e^-(2*t)-%e^(2*t),%e^(2*t)-%e^-(2*t),2*%e^-(2*t)-%e^(2*t)])
(%i6) is(equal(eAt,eAt2));
(%o6) true
(%i7) eAt - eAt2;
(%o7) matrix([0,0,0],[0,0,0],[0,0,0])
```

8.9 $\mathbf{e^{A\,t}}$ Using the Cayley-Hamilton Theorem

Simple Direct Method to Evaluate $\mathbf{e^{At}}$ if A Has Only a Single Eigenvalue

The Cayley-Hamilton theorem and its use will look less mysterious if we first consider the case that a matrix has only one eigenvalue. Let the single eigenvalue of A be λ. Then there will be some finite positive integer k such that

$$(A - \lambda\, I)^k = \mathbf{0}. \tag{8.91}$$

After finding this value of k (see the Maxima example below), we use the identity

$$A \equiv \lambda\, I + (A - \lambda\, I) \tag{8.92}$$

and then proceed as follows, using the general properties of the matrix exponential.

$$\begin{aligned} e^{A\,t} &\equiv e^{\lambda\, I\, t + (A - \lambda\, I)\, t} \\ &= e^{\lambda\, I\, t}\, e^{(A - \lambda\, I)\, t} \\ &= e^{\lambda\, t}\, e^{(A - \lambda\, I)\, t} \\ &= e^{\lambda\, t}\, (I + \sum_{j=1}^{k-1} (A - \lambda\, I)^j\, (t^j/j!)) \end{aligned}$$

and only k terms of the exponential series are required, with the first term being I and the last term corresponding to $j = k - 1$.

Consider the matrix

$$A = \begin{bmatrix} 1 & 4 \\ -1 & -3 \end{bmatrix}. \tag{8.93}$$

which has one repeated eigenvalue $\lambda = -1$. We first use **`mat_function(exp,t*A)`** from the package **`diag.mac`**.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,4],[-1,-3]);
(%o2) matrix([1,4],[-1,-3])
(%i3) eigenvalues(A);
(%o3) [[-1],[2]]
(%i4) eAt2 : expand (mat_function (exp,t*A));
(%o4) matrix([2*t*%eˆ-t+%eˆ-t,4*t*%eˆ-t],[-t*%eˆ-t,%eˆ-t-2*t*%eˆ-t])
(%i5) eAt2, t=0;
(%o5) matrix([1,0],[0,1])
(%i6) expand (eAt2/exp(-t));
(%o6) matrix([2*t+1,4*t],[-t,1-2*t])
```

which implies that

$$e^{A\,t} = e^{-t} \begin{bmatrix} 1 + 2t & 4t \\ -t & 1 - 2t \end{bmatrix}. \tag{8.94}$$

We now use the above simple method applicable to the case that the matrix A has only a single eigenvalue, here using $\lambda = -1$:

```
(%i7) I : ident(2);
(%o7) matrix([1,0],[0,1])
(%i8) lambda : -1$
(%i9) B : A - lambda*I;
(%o9) matrix([2,4],[-1,-2])
(%i10) Bˆˆ2;
(%o10) matrix([0,0],[0,0])
```

which shows that $k = 2$.

We separate out the factor $\exp(\lambda t)$, and let **Ms1** be the series contribution:

```
(%i11) Ms1 : I + B*t;
(%o11) matrix([2*t+1,4*t],[-t,1-2*t])
```

which implies that

$$e^{A\,t} = e^{-t}\begin{bmatrix} 1+2t & 4t \\ -t & 1-2t \end{bmatrix}, \tag{8.95}$$

agreeing with the result found using **mat_function**.

The Cayley-Hamilton Theorem

Quoting Bronson in his Ch. 7

> Every square matrix A satisfies its own characteristic equation. That is, if
>
> $$\mathbf{det}(A - \lambda\, I) = b_n\,\lambda^n + b_{n-1}\,\lambda^{n-1} + \cdots + b_2\,\lambda^2 + b_1\,\lambda + b_0 = 0 \tag{8.96}$$
>
> then with $I = A^0$ the identity matrix,
>
> $$b_n\,A^n + b_{n-1}\,A^{n-1} + \cdots + b_2\,A^2 + b_1\,A + b_0\,I = \mathbf{0}. \tag{8.97}$$

As a simple example, in Problem 7.15, Bronson verifies the Cayley-Hamilton theorem for the case

$$A = \begin{bmatrix} 3 & 5 \\ -2 & -4 \end{bmatrix}. \tag{8.98}$$

Here we use Maxima for this verification. Recall that **charpoly(A, lambda)** computes $p(\lambda) = \mathbf{det}(A - \lambda\, I)$, which should be set to 0 (the "characteristic equation") to determine the eigenvalues of A.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix( [3,5],[-2,-4] );
(%o2) matrix([3,5],[-2,-4])
(%i3) eigenvalues(A);
(%o3) [[1,-2],[1,1]]
(%i4) expand (charpoly (A, lambda));
(%o4) lambda^2 + lambda - 2
(%i5) A^^2 + A - 2*ident(2);
(%o5) matrix([0,0],[0,0])
```

Computing Functions of Matrices Using the Cayley-Hamilton Theorem

Quoting Bronson from his Ch. 8:

> An infinite series expansion for $f(A)$ is not generally useful for computing the elements of the matrix $f(A)$. It follows (with some effort) from the Cayley-Hamilton theorem that every well-defined function of an $n \times n$ matrix A can be expressed as a polynomial of degree $n-1$ in A.

Thus, if $f(A)$ is a well-defined function of a $n \times n$ matrix A, there exist n scalars a_j such that

$$f(A) = a_0\,I + a_1\,A + \cdots + a_{n-1}\,A^{n-1}. \tag{8.99}$$

If the function $f(A)$ depends on some scalar parameter, such as t, in addition to A, then the n scalars a_j depend, in general, on that parameter. We need n independent equations to determine the n scalars a_j. Knowing the scalars a_j, they can be substituted into (8.99) to find $f(A)$. Here is a path (adapting Bronson's discussion) which can be used:

1. Let
$$r(\lambda) = a_0 \; + a_1\,\lambda + \cdots + a_{n-1}\,\lambda^{n-1}, \tag{8.100}$$
which is the right hand side of (8.99) with A^j replaced by λ^j ($j = 0, 1, \ldots, n-1$), which includes, for $j = 0$, $I = A^0 \to \lambda^0 = 1$.

2. For each eigenvalue λ_j of A, we have the equation
$$f(\lambda_j) = r(\lambda_j), \tag{8.101}$$
since if we multiply both sides of (8.99) by the eigenvector $\mathbf{v}^j$ corresponding to the eigenvalue λ_j, then $f(A)\,\mathbf{v}^j = f(\lambda_j)\,\mathbf{v}^j$ on the left hand side, and on the right hand side we get $r(\lambda_j)\,\mathbf{v}^j$. Since $\mathbf{v}^j \neq \mathbf{0}$, we arrive at (8.101).

   If there are n distinct eigenvalues, then we have n independent equations in the n unknowns a_i.

3. If λ_i is an eigenvalue of multiplicity $m_i > 1$, then this sector contributes m_i equations, the first being (8.101), and the additional $m_i - 1$ equations, involving derivatives of (8.101) with respect to λ, evaluated at λ_i are:
$$\begin{aligned} f'(\lambda)\big|_{\lambda=\lambda_i} &= r'(\lambda)\big|_{\lambda=\lambda_i} \\ f''(\lambda)\big|_{\lambda=\lambda_i} &= r''(\lambda)\big|_{\lambda=\lambda_i} \\ \vdots &= \vdots \\ f^{m-1}(\lambda)\big|_{\lambda=\lambda_i} &= r^{m-1}(\lambda)\big|_{\lambda=\lambda_i} \end{aligned}$$

4. We then solve a set of n independent equations generated as above for the n scalars a_j, and substitute them in (8.99) to find $f(A)$.

$\mathbf{e^{A\,t}}$ for a 2×2 Matrix A with One Repeated Eigenvalue

The 2×2 matrix
$$A = \begin{bmatrix} 1 & 4 \\ -1 & -3 \end{bmatrix}. \tag{8.102}$$
has the repeated eigenvalue $\lambda = -1$. We then have
$$e^{A\,t} = a_0\,I + a_1\,A \tag{8.103}$$
where the first equation needed is
$$e^{\lambda\,t} = a_0 + a_1\,\lambda \tag{8.104}$$
and the second equation needed is the first derivative
$$t\,e^{\lambda\,t} = a_1 \tag{8.105}$$
which implies
$$a_0 = (1 - \lambda\,t)\,e^{\lambda\,t} \tag{8.106}$$
which gives
$$e^{A\,t} = (1 - \lambda\,t)\,e^{\lambda\,t}\,I + t\,e^{\lambda\,t}\,A \tag{8.107}$$
which, with $\lambda = -1$, gives
$$e^{A\,t} = (1 + t)\,e^{-t}\,I + t\,e^{-t}\,A \tag{8.108}$$

Using this method interactively,

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([1,4],[-1,-3]);
(%o2) matrix([1,4],[-1,-3])
(%i3) eigenvalues(A);
(%o3) [[-1],[2]]
(%i4) eAt : (1+t)*exp(-t)*ident(2) + t*exp(-t)*A;
(%o4) matrix([(t+1)*%eˆ-t+t*%eˆ-t,4*t*%eˆ-t],[-t*%eˆ-t,
                                        (t+1)*%eˆ-t-3*t*%eˆ-t])
(%i5) eAt : expand(eAt);
(%o5) matrix([2*t*%eˆ-t+%eˆ-t,4*t*%eˆ-t],[-t*%eˆ-t,%eˆ-t-2*t*%eˆ-t])
(%i6) eAt2 : mat_function(exp,t*A);
(%o6) matrix([2*t*%eˆ-t+%eˆ-t,2*t*(2*%eˆ-t-%eˆ-t/t)+2*%eˆ-t],
             [-t*%eˆ-t,-t*(2*%eˆ-t-%eˆ-t/t)])
(%i7) is (equal (eAt, eAt2));
(%o7) true
(%i8) expand(eAt / %eˆ(-t));
(%o8) matrix([2*t+1,4*t],[-t,1-2*t])
```

We can write the result as

$$e^{A\,t} = e^{-t}\begin{bmatrix}(1+2\,t) & 4\,t\\ -t & (1-2\,t)\end{bmatrix}. \tag{8.109}$$

Examples of the Cayley-Hamilton Method for $f(A) = \mathbf{e^{A\,t}}$ and $n = 3$

For the case $f(A) = e^{A\,t}$ and $n = 3$ we have

$$f(A) = e^{A\,t} = a_0\,I + a_1\,A + a_2\,A^2, \tag{8.110}$$

and

$$f(\lambda) = e^{\lambda\,t}, \tag{8.111}$$

and

$$r(\lambda) = a_0 + a_1\,\lambda + a_2\,\lambda^2. \tag{8.112}$$

Three Distinct Eigenvalues Example

We can write the set of three equations, generated by the three distinct eigenvalues, in matrix form as

$$\mathbf{f} = \begin{bmatrix}f(\lambda_1)\\ f(\lambda_2)\\ f(\lambda_3)\end{bmatrix} = \begin{bmatrix}e^{\lambda_1\,t}\\ e^{\lambda_2\,t}\\ e^{\lambda_3\,t}\end{bmatrix} = V\,\mathbf{a} = \begin{bmatrix}1 & \lambda_1 & \lambda_1^2\\ 1 & \lambda_2 & \lambda_2^2\\ 1 & \lambda_3 & \lambda_3^2\end{bmatrix}\begin{bmatrix}a_0\\ a_1\\ a_2\end{bmatrix} \tag{8.113}$$

and the three unknown scalars a_j (which here depend upon the parameter t) can then be found as the elements of the column matrix given by

$$\mathbf{a} = V^{-1}\,\mathbf{f}. \tag{8.114}$$

The square 3×3 matrix V (the *Vandermonde matrix*) is always invertible as long as the eigenvalues are all distinct.

A 3×3 example in which the eigenvalues are distinct is provided by

$$A = \begin{bmatrix}8 & -5 & 10\\ 2 & 1 & 2\\ -4 & 4 & -6\end{bmatrix}. \tag{8.115}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix ([8,-5,10],[2,1,2],[-4,4,-6]);
(%o2) matrix([8,-5,10],[2,1,2],[-4,4,-6])
(%i3) eigenvalues(A);
(%o3) [[-2,2,3],[1,1,1]]
```

We have $\lambda_1 = -2$, $\lambda_2 = 2$, and $\lambda_3 = 3$. We first use the Cayley-Hamilton method and then check that the answer is the same as returned by **mat_function**.

```
(%i4) (lam1:-2,  lam2:2,  lam3:3)$
(%i5) V : matrix([1,lam1,lam1^2],[1,lam2,lam2^2],[1,lam3,lam3^2]);
(%o5) matrix([1,-2,4],[1,2,4],[1,3,9])
(%i6) fc : transpose(matrix([exp(lam1*t),exp(lam2*t),exp(lam3*t)]));
(%o6) matrix([%e^-(2*t)],[%e^(2*t)],[%e^(3*t)])
(%i7) ac : invert(V) . fc;
(%o7) matrix([(-(4*%e^(3*t))/5)+(3*%e^(2*t))/2+(3*%e^-(2*t))/10],
             [%e^(2*t)/4-%e^-(2*t)/4],[%e^(3*t)/5-%e^(2*t)/4+%e^-(2*t)/20])
(%i8) ac : ratsimp(ac);
(%o8) matrix([-(%e^-(2*t)*(8*%e^(5*t)-15*%e^(4*t)-3))/10],
             [(%e^-(2*t)*(%e^(4*t)-1))/4],
             [(%e^-(2*t)*(4*%e^(5*t)-5*%e^(4*t)+1))/20])
(%i9) a0 : ac[1,1];
(%o9) -(%e^-(2*t)*(8*%e^(5*t)-15*%e^(4*t)-3))/10
(%i10) a1 : ac[2,1];
(%o10) (%e^-(2*t)*(%e^(4*t)-1))/4
(%i11) a2 : ac[3,1];
(%o11) (%e^-(2*t)*(4*%e^(5*t)-5*%e^(4*t)+1))/20
(%i12) eAt : ratsimp ( a0*ident(3) + a1*A + a2*A^^2);
(%o12) matrix([%e^-(2*t)*(2*%e^(5*t)-1),-%e^-(2*t)*(%e^(5*t)-1),
               %e^-(2*t)*(2*%e^(5*t)-2)],
              [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^(2*t)],
              [-%e^-(2*t)*(%e^(4*t)-1),%e^-(2*t)*(%e^(4*t)-1),
               -%e^-(2*t)*(%e^(4*t)-2)])
(%i13) eAt, t=0;
(%o13) matrix([1,0,0],[0,1,0],[0,0,1])
(%i19) eAt2 : mat_function(exp,t*A);
(%o19) matrix([2*%e^(3*t)-%e^-(2*t),%e^-(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^-(2*t)],
              [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^(2*t)],
              [%e^-(2*t)-%e^(2*t),%e^(2*t)-%e^-(2*t),2*%e^-(2*t)-%e^(2*t)])
(%i20)  ratsimp(expand (eAt - eAt2));
(%o20) matrix([0,0,0],[0,0,0],[0,0,0])
```

Example of a Repeated Eigenvalue

Let λ_1 be the eigenvalue with multiplicity 1 and λ_2 be the eigenvalue with multiplicity 2. A suitable matrix is

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}. \tag{8.116}$$

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix([0,1,1],[1,0,1],[1,1,0]);
(%o2) matrix([0,1,1],[1,0,1],[1,1,0])
(%i3) eigenvalues(A);
(%o3) [[2,-1],[1,2]]
```

Here $\lambda_1 = 2$ with multiplicity $m = 1$ and $\lambda_2 = -1$ with multiplicity $m = 2$. Our first equation is $f(\lambda_1) = r(\lambda_1)$, or

$$e^{2\,t} = a_0 + 2\,a_1 + 4\,a_2. \tag{8.117}$$

Our second equation is $f(\lambda_2) = r(\lambda_2)$, or

$$e^{-t} = a_0 - a_1 + a_2. \tag{8.118}$$

Our third independent equation is

$$\frac{df(\lambda)}{d\lambda}\Big|_{\lambda=-1} = \frac{dr(\lambda)}{d\lambda}\Big|_{\lambda=-1} \tag{8.119}$$

or

$$t\,e^{-t} = a_1 - 2\,a_2. \tag{8.120}$$

We can write the above set of three equations in matrix form as

$$\mathbf{f} = \begin{bmatrix} e^{2\,t} \\ e^{-t} \\ t\,e^{-t} \end{bmatrix} = V\,\mathbf{a} = \begin{bmatrix} 1 & 2 & 4 \\ 1 & -1 & 1 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \tag{8.121}$$

and the three unknown scalars a_j (which here depend upon the parameter t) can then be found as the elements of the column matrix given by

$$\mathbf{a} = V^{-1}\,\mathbf{f}. \tag{8.122}$$

We first use the Cayley-Hamilton method and then check that the answer is the same as returned by **mat_function**.

```
(%i4) V : matrix([1,2,4],[1,-1,1],[0,1,-2]);
(%o4) matrix([1,2,4],[1,-1,1],[0,1,-2])
(%i5) fc : transpose(matrix([exp(2*t),exp(-t),t*exp(-t)]));
(%o5) matrix([%e^(2*t)],[%e^-t],[t*%e^-t])
(%i6) ac : invert(V) . fc;
(%o6) matrix([%e^(2*t)/9+(2*t*%e^-t)/3+(8*%e^-t)/9],
             [(2*%e^(2*t))/9+(t*%e^-t)/3-(2*%e^-t)/9],
             [%e^(2*t)/9-(t*%e^-t)/3-%e^-t/9])
(%i7) a0 : ac[1,1];
(%o7) %e^(2*t)/9+(2*t*%e^-t)/3+(8*%e^-t)/9
(%i8) a1 : ac[2,1];
(%o8) (2*%e^(2*t))/9+(t*%e^-t)/3-(2*%e^-t)/9
(%i9) a2 : ac[3,1];
(%o9) %e^(2*t)/9-(t*%e^-t)/3-%e^-t/9
(%i10) eAt : expand ( a0*ident(3) + a1*A + a2*A^^2);
(%o10) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i11) eAt, t=0;
(%o11) matrix([1,0,0],[0,1,0],[0,0,1])
(%i12) eAt2 : expand (mat_function (exp,t*A));
(%o12) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i13) is (equal (eAt, eAt2));
(%o13) true
```

Automated Code eAt_CH(A) for Cayley-Hamilton Method

The **mbe5.mac** function **eAt_CH** function automates the above process for any square matrix, as shown by the following examples.

First for the 2×2 matrix used above:

```
(%i14) kill (V,fc,ac,a0,a1,a2);
(%o14) done
(%i15) A : matrix([1,4],[-1,-3]);
(%o15) matrix([1,4],[-1,-3])
(%i16) eigenvalues (A);
(%o16) [[-1],[2]]
(%i17) eAt : eAt_CH(A);
(%o17) matrix([2*t*%e^-t+%e^-t,4*t*%e^-t],[-t*%e^-t,%e^-t-2*t*%e^-t])
(%i18) eAt2 : expand (mat_function(exp,t*A));
(%o18) matrix([2*t*%e^-t+%e^-t,4*t*%e^-t],[-t*%e^-t,%e^-t-2*t*%e^-t])
(%i19) is (equal (eAt,eAt2));
(%o19) true
```

Secondly, with a 3×3 matrix with two eigenvalues, also used above:

```
(%i20) A : matrix([0,1,1],[1,0,1],[1,1,0]);
(%o20) matrix([0,1,1],[1,0,1],[1,1,0])
(%i21) eigenvalues(A);
(%o21) [[2,-1],[1,2]]
```

```
(%i22) eAt : eAt_CH(A);
(%o22) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i23) eAt2 : expand (mat_function(exp,t*A));
(%o23) matrix([%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3,%e^(2*t)/3-%e^-t/3],
              [%e^(2*t)/3-%e^-t/3,%e^(2*t)/3-%e^-t/3,%e^(2*t)/3+(2*%e^-t)/3])
(%i24) is(equal(eAt,eAt2));
(%o24) true
```

Finally the example of the 3×3 matrix used above, with three distinct eigenvalues:

```
(%i25) A : matrix([8,-5,10],[2,1,2],[-4,4,-6]);
(%o25) matrix([8,-5,10],[2,1,2],[-4,4,-6])
(%i26) eigenvalues(A);
(%o26) [[-2,2,3],[1,1,1]]
(%i27) eAt : eAt_CH(A);
(%o27) matrix([2*%e^(3*t)-%e^-(2*t),%e^-(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^-(2*t)],
              [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^(2*t)],
              [%e^-(2*t)-%e^(2*t),%e^(2*t)-%e^-(2*t),2*%e^-(2*t)-%e^(2*t)])
(%i28) eAt2 : expand (mat_function(exp,t*A));
(%o28) matrix([2*%e^(3*t)-%e^-(2*t),%e^-(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^-(2*t)],
              [2*%e^(3*t)-2*%e^(2*t),2*%e^(2*t)-%e^(3*t),
               2*%e^(3*t)-2*%e^(2*t)],
              [%e^-(2*t)-%e^(2*t),%e^(2*t)-%e^-(2*t),2*%e^-(2*t)-%e^(2*t)])
(%i29) is (equal (eAt, eAt2));
(%o29) true
```

We refer the interested reader to the code file **mbe5.mac** for the code for **eAt_CH**, which makes crucial use of the matrix function **coefmatrix**. The symbol **t** is hardwired into this code.

8.10 Brute Force Taylor Expansion of $\mathbf{e}^{\mathbf{A}}$

If A is a purely numerical matrix, then we can approximately evaluate e^A by taking the first 10 or 20 terms of the Taylor series expansion. We can define the exponential of a square matrix A as the sum from $n = 0$ to ∞ of $A^n/n!$, with the general term written in Maxima syntax as **(A ^^n)/factorial(n)**. It is more efficient to multiply the previous term in the sum by **A**, with an appropriate scalar muliplier, as shown in our do-loop code below.

Here is a 10 term Taylor series expansion of **exp(A)**, where A is a purely numerical matrix, using an **interactive** calculation.

```
(%i1) load(mbe5);
(%o1) "c:/work9/mbe5.mac"
(%i2) A : matrix( [0.5, 1], [0, 0.6] );
(%o2) matrix([0.5,1],[0,0.6])
(%i3) B : A;
(%o3) matrix([0.5,1],[0,0.6])
(%i4) f : ident(2);
(%o4) matrix([1,0],[0,1])
(%i5) for i thru 10 do (
        f : f + B,
        B : A . B /(i+1))$
(%i6) f;
(%o6) matrix([1.6487213,1.7339753],[0.0,1.8221188])
(%i7) fpprintprec : 16$
(%i8) f;
(%o8) matrix([1.648721270687366,1.733975296074915],[0.0,1.822118800294857])
```

In the Chapter 4 code file **mbe5.mac** is code for a simple do-loop Taylor series expansion of a square matrix called **exp_taylor(A,n)**, which expands the square matrix **A** in a Taylor series using **n** terms, following the same path as our interactive calculation above.

This code is

```
exp_taylor(%A, %n) :=
block([%nr, %nc, %B, %M, numer], numer:true,
    %nr : length(%A),
    %nc : length( transpose(%A)),
    if %nr # %nc then (
        print(" not a square matrix "),
        return()),
    %B : %A,
    %M : ident(%nc),
    for i thru %n do (
        %M : %M + %B,
        %B : %A . %B/(i+1)),
    %M )$
```

We compare the use of this brute force Taylor series exansion definition **`exp_taylor`**, using 10 or 20 terms, with the numerical value found using **`mat_function`** (which returns the analytical solution). Note that the code file **`mbe5.mac`** sets **`fpprintprec:8$`**. The arithmetic results use 16 digit accuracy, of course.

```
(%i9) fpprintprec:8$
(%i10) A : matrix([1/2, 1],[0,3/5]);
(%o10) matrix([1/2,1],[0,3/5])
(%i11) eA_exact : expand (mat_function(exp,A));
(%o11) matrix([sqrt(%e),10*%e^(3/5)-10*sqrt(%e)],[0,%e^(3/5)])
(%i12) eA_numer : eA_exact, numer;
(%o12) matrix([1.6487213,1.7339753],[0,1.8221188])
(%i13) A : A,numer;
(%o13) matrix([0.5,1],[0,0.6])
(%i14) eA10 : exp_taylor(A,10);
(%o14) matrix([1.6487213,1.7339753],[0.0,1.8221188])
(%i15) eA20 : exp_taylor(A,20);
(%o15) matrix([1.6487213,1.7339753],[0.0,1.8221188])
(%i16) fpprintprec : 16$
(%i17) eA10 - eA_numer;
(%o17) matrix([-1.276267980188095e-11,-8.288889574714631e-10],
              [0.0,-9.565170877579021e-11])
(%i18) eA20 - eA_numer;
(%o18) matrix([-4.440892098500626e-16,4.440892098500626e-15],
              [0.0,2.220446049250313e-16])
```

We see that stopping after 20 terms gives numerical results which are satisfactory.