Boldly Going - Eigenvalues and Friends

1. Introduction

This article features a subprogram, **PCHAR**, which deals with an advanced funtionality very frequently needed in all sort of disciplines in *Science*, *Engineering* and *Computing*, yet not covered at all in the *HP-71* ***Math Pac*** *ROM*, namely the task of computing all eigenvalues real and complex of an arbitrary real or complex square matrix. **PCHAR** helps in this regard and besides it also returns at no cost both the *determinant* and the *inverse* matrix.

One way to compute the eigenvalues is to take advantage of the already existing ***Math Pac***'s keyword **MAT..PROOT** or my subprogram **PZER** (both can work as *global polynomial rootfinders*[1]), because actually the eigenvalues are the roots of the *Characteristic Polynomial* of the matrix. Thus, if we can quickly and accurately compute the coefficients of said polynomial, then afterwards using either **MAT..PROOT** (only for *real* polynomials) or **PZER** (for both *real* and *complex* ones) will complete the task, producing all the eigenvalues at once. **PCHAR** does exactly that and much more, as we'll see. The main points for **PCHAR** are:

- Its **usefulness**: computing the eigenvalues of a matrix is essential in many important areas of science and engineering, as well as analysis of algorithms, etc. See for instance *Chapter 10: "Origins of Matrix Eigenvalue Problems"* from the book *"Numerical Methods for Large Eigenvalues Problems"*. I quote:

  "We list below just ***a few*** *of the applications areas where eigenvalue calculations arise:*
  - *Structural dynamics • Quantum chemistry • Electrical Networks • Markov chain techniques*
  - *Combustion processes • Chemical reactions • Macro-economics • Magnetohydrodynamics*
  - *Normal mode techniques • Control theory*

  One class of applications which has recently gained considerable ground is that related to linear algebra methods in ***data-mining****, [...] The most commonly solved eigenvalue problems today are those issued from the first item in the list, namely those problems associated with the* ***vibration analysis*** *of large structures"*

- Its **versatility**: besides its primary task of computing the *Characteristic Polynomial (CP)*, **PCHAR** also returns the *determinant* and the *inverse* of the input matrix for free, *at no cost whatsoever* . They are side byproducts of the main computation and returning them doesn't involve *any* extra memory or time.

  Furthermore, it transparently works for *real* or *complex* matrices and will return *real* or *complex* determinants/inverses (note that the ***Math Pac*** *can't* compute determinants of complex matrices), with far more precision in some cases, such as certain integer and troublesome matrices. See *Examples 5-9* below.

- Its **simplicity**: this *BASIC* subprogram is just 6 lines *(271 bytes)* long and besides it's quite amenable for conversion to *Assembler* code. See *Section 6* below for full details about the conversion process.

MAT..PROOT or **PZER** already do much of the work and **PCHAR** takes care of the essential preliminary step. As stated above, **MAT..PROOT** only works for polynomials with *real* coefficients while **PZER** works for both *real* and *complex* coefficients, so for complex *Characteristic Polynomials* **PZER** is the only choice. For *real CP*s both **MAT..PROOT** and **PZER** can be used and will produce similarly accurate results but the assembly-language **MAT..PROOT** keyword will produce results much faster than the *BASIC* subprogram **PZER**[2].

[1] "No user-supplied initial guess(es) or stopping criteria should be required" *(July 1984 Hewlett-Packard Journal p.33)*.
[2] See my article *"HP Article VA042 - Boldly Going - Outsmarting PROOT"* for **PZER**'s full documentation.

2. Subprogram Description and Calling Syntax

PCHAR **Characteristic Polynomial**

SUB PCHAR(A(,),P(),D,M(,))

Where **P** is a vector, **A** and **M** are different square matrices, and **D** is a scalar variable. All must be of the same type, real or complex, and can be declared as **REAL**, **SHORT**, **INTEGER**, **COMPLEX** or **COMPLEX SHORT** precision.

Assigns to **P** the coefficients of the ***Characteristic Polynomial*** of **A**, in a format ready for immediate use with either the **MAT..PROOT** keyword (only if they're *real*) or with subprogram **PZER** (real or *complex* coefficients) to find at once all eigenvalues real and/or complex of **A**.

If scalar variable **D** is passed *by reference* the ***determinant*** of matrix **A** will be returned there.

Matrix **M** is also passed by reference and the ***matrix inverse*** of **A** will be returned in it. If the determinant of **A** is *0* or *(0,0)*, then matrix **A** is *singular* and the returned inverse will be meaningless.

3. Source Code Listing and Subprogram Characteristics

```
200  SUB PCHAR(A(,),P(),D,M(,)) @ P(0)=1 @ K=UBND(A,1) @ IF K=1 THEN MAT M=IDN ELSE MAT M=A
210  FOR I=1 TO K-1 @ D=0 @ IF I#1 THEN MAT M=A*M
220  FOR J=1 TO K @ D=D-M(J,J) @ NEXT J @ D=D/I @ P(I)=D
230  FOR J=1 TO K @ M(J,J)=M(J,J)+D @ NEXT J @ NEXT I @ D=0
240  FOR I=1 TO K @ FOR J=1 TO K @ D=D-A(I,J)*M(J,I) @ NEXT J @ NEXT I
250  D=D/K @ P(K)=D @ MAT M=(-1/(D+NOT ABS(D)))*M @ D=(-1)^K*D
```

Note: *the line numbers are arbitrary, the code uses none as there's no explicit branching. Use whatever numbering suits you.*

- This *BASIC* subprogram is 6 lines (271 bytes) long and uses several matrix-related keywords from the ***Math Pac*** so the *Math ROM* must be available (either physically plugged in or as a virtual *ROM* image.)
- It can be called either directly from the command line or from another program or subprogram.
- It accepts four parameters, namely:
  - **A**(,) *input*, square matrix whose *CP*, determinant and/or inverse are sought. Returns unaltered.
  - **P**( ) *output*, column vector where the *CP*'s coefficients will be returned. Always *monic*.
  - **D** *output*, scalar num. variable where the determinant will be returned (if passed by reference).
  - **M**(,) *output*, square matrix where the inverse will be returned, if it exists. Else, meaningless.
- All parameters must be of the same type, **REAL** or **COMPLEX**.
- All parameters can be of full **REAL** (**COMPLEX**), **SHORT** (**COMPLEX SHORT**) or **INTEGER** precision, but if **SHORT** or **INTEGER** are specified accuracy can degrade, and **INTEGER** can result in *Overflow* warnings.
- Both matrices **A** and **M** must have the same dimensions and must be declared with **OPTION BASE 1** in effect. Also, **M** can't be the same matrix as **A**, they must be different matrices.
- The column vector **P**( ) must be declared with the same upper index as the matrices but with **OPTION BASE 0**.

4. Usage Instructions

The subprogram **PCHAR** can be called either from the command line or from any other program or subprogram, by performing the following steps:

1) Specify **OPTION BASE 1** and dimension the square matrices **A** and **M** (the names are arbitrary but they can't be the same matrix). Both matrices must be of the same type (**REAL** or **COMPLEX)** and of any precision (**REAL**, **REAL SHORT**, **COMPLEX**, **COMPLEX SHORT** or even **INTEGER** (the latter is absolutely discouraged, as it can lead to massive loss of precision in the results or even `Overflow` errors or warnings). E.g.:

`OPTION BASE 1 @ DIM A(3,3),M(3,3)`	*specifies two 3x3 real matrices of full precision*
`OPTION BASE 1 @ REAL A(3,3),M(3,3)`	*ditto*
`OPTION BASE 1 @ SHORT A(7,7),M(7,7)`	*specifies two 7x7 real matrices of short precision*
`OPTION BASE 1 @ COMPLEX A(5,5),M(5,5)`	*specifies two 5x5 complex matrices of full precision*
`OPTION BASE 1 @ COMPLEX SHORT A(4,4),M(4,4)`	*specifies two 4x4 complex matrices of short precision*

2) Specify **OPTION BASE 0** and dimension vector **P** and scalar variable **D** (names are arbitrary but they must be same type **REAL**/**COMPLEX** as matrices **A** and **M**, and **P**'s upper bound must be the same as well). E.g.:

`OPTION BASE 0 @ DIM P(3),D`	*4-element real vector and scalar w/ full precision*
`OPTION BASE 0 @ REAL P(3),D`	*ditto*
`OPTION BASE 0 @ SHORT P(7),D`	*8-element real vector and scalar w/ short precision*
`OPTION BASE 0 @ COMPLEX P(5),D`	6-*element complex vector and scalar w/ full precision*
`OPTION BASE 0 @ COMPLEX SHORT P(4),D`	*5-element complex vector and scalar w/ short precision*

3) Assign the elements of matrix **A** (e.g.: using **MAT INPUT**, **READ**/**DATA** or computing the elements).

4) Call the subprogram **PCHAR**:

```
CALL PCHAR(A,P,D,M)
```

5) Upon returning from the call:

- **A** will be left *unaltered*,
- **P** will contain the coefficients of the *Characteristic Polynomial* of **A**,
- **D** will contain the *determinant* of **A** and if **D** isn't ***0*** or ***(0,0)*** then
- ***M*** will contain the *matrix inverse* of **A** (else the contents of **M** will be meaningless).

6) If desired, you can display the coefficients of the *CP* using **MAT DISP P** , the *determinant* using **DISP D**, and the elements of the *inverse matrix* using **MAT DISP M;** , for instance.

7) Whether **A** is real or complex, you can now compute all its eigenvalues by performing the following steps:

7.a) Dimension a *complex* vector **R** (say) to hold the *N* eigenvalues of the *NxN* square matrix **A**. E.g.:

`OPTION BASE 1 @ COMPLEX R(7)`	*assuming A is a 7x7 real matrix*

7.b) Compute and display the eigenvalues by executing *either*

`MAT R=PROOT(P) @ MAT DISP R`	*only if **P** has <u>real</u> coefficients,i.e.: real **A** (fastest)*

or

`CALL PZER(P,R,0,0) @ MAT DISP R`	*if **P** has real or <u>complex</u> coefficients[1] (complex **A**)*

[1] In this case you must have subprogram **PZER** available either in the same file as subprogram **PCHAR** or in some other file in main *RAM*. You can find the *BASIC* source listing and full documentation for **PZER** in my article *"HP Article VA042 - Boldly Going - Outsmarting PROOT"*.

5. Examples *{all of them demonstrated as if directly executed from the command line }*

The "Basic" Cases

Example 1: *A simple 3x3 real matrix.*
Example 2: *A simple 3x3 complex matrix.*
Example 3: *Singular real and complex matrices.*
Example 4: *The 5x5 symmetric real matrix featured in my article "HP Article VA012", pp 6-7.*

The "Troublesome" Cases

Example 5: *The 7x7 real matrix AM#1 featured in my article "HP Article VA016 - Mean Matrices".*
Example 6: *A simple symmetric 3x3 real matrix, which the Math Pac handles inaccurately .*
Example 7: *A very simple 2x2 complex matrix, which the Math Pac handles quite inaccurately .*
Example 8: *A seemingly slight inaccuracy which actually isn't, unlike what the Math Pac does.*
Example 9: *The Mother of All Troublesome Matrices, my 7x7 real matrix AM#7 (see my "HP Article VA016").*

~~~~~~

Example 1: *A simple 3x3 real matrix.*

Let's find the characteristic polynomial, determinant, inverse and all the eigenvalues of this 3x3 real matrix:

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 5 \\ 3 & 3 & 1 \\ 4 & 6 & 4 \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ DIM A(3,3),M(3,3) @ COMPLEX R(3)
>OPTION BASE 0 @ DIM P(3),D

>MAT INPUT A
A(1,1)?      3,1,5,3,3,1,4,6,4

>CALL PCHAR(A,P,D,M)

>MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -10 \\ 4 \\ -40 \end{bmatrix}$$ so its characteristic polynomial is: $\boldsymbol{P(x) = x^3 - 10\,x^2 + 4\,x - 40}$

```
>D
```

40 and its determinant is *40*. As it's non-zero, the inverse exists and we now display it:

```
>FIX 2 @ MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.15 & 0.65 & -0.35 \\ -0.20 & -0.20 & 0.30 \\ 0.15 & -0.35 & 0.15 \end{bmatrix}$$ the exact inverse matrix.

And finally we compute and display all its eigenvalues:

```
>MAT R=PROOT(P) @ MAT DISP R
```

$$\mathbf{R} = \begin{bmatrix} 5.76\,.\,10^{-17} + 2i \\ 5.76\,.\,10^{-17} - 2i \\ 10 \end{bmatrix}$$ so there's a real eigenvalue and two complex conjugate ones, $\pm\, \boldsymbol{2\,i}$.
~~~~~~

Example 2: *A simple 3x3 complex matrix.*

Find the characteristic polynomial, determinant, inverse and all the complex eigenvalues of the following matrix:

$$\mathbf{A} = \begin{bmatrix} 1+2\,i & 2+3\,i & 3+i \\ -1+2\,i & 2-i & -1-i \\ 3\,i & -2 & 2+2\,i \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ COMPLEX A(3,3),M(3,3)
>OPTION BASE 0 @ COMPLEX P(3),D

>MAT INPUT A
A(1,1)?      (1,2),(2,3),(3,1),(-1,2),(2,-1),(-1,-1),(0,3),(-2,0),(2,2)

>CALL PCHAR(A,P,D,M)
```

Let's display the characteristic polynomial:

```
>MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -5-3\,i \\ 17-i \\ -44+6\,i \end{bmatrix}$$ so the characteristic polynomial is $\boldsymbol{P(z) = z^3 + (-5-3\,i)\,z^2 + (17-i)\,z - 44 + 6\,i}$

and the determinant is:

```
>D
   (44,-6)
```

this is: $\boldsymbol{Det = 44 - 6\,i}$. As for the inverse matrix:

```
>FIX 4 @ MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.0892 + 0.0122\,i & -0.0527 - 0.2799\,i & -0.1217 - 0.1075\,i \\ 0.2160 - 0.0842\,i & 0.0314 - 0.0639\,i & -0.1582 + 0.1602\,i \\ 0.0081 - 0.2262\,i & -0.1866 + 0.2018\,i & 0.2617 + 0.0811\,i \end{bmatrix}$$

We can check the accuracy of the inverse by multiplying it by the original matrix and seeing how close the product is to the 3x3 complex *Identity* matrix:

```
>STD @ MAT M=A*M @ MAT DISP M;
```

$$\mathbf{A*A}^{-1} = \begin{bmatrix} \mathbf{1} - 1.5\,.\,10^{-12}\,i & 5\,.\,10^{-13} + 6\,.\,10^{-13}\,i & 2\,.\,10^{-12} \\ -5\,.\,10^{-13} + 3\,.\,10^{-13}\,i & \mathbf{1} - 6\,.\,10^{-13}\,i & -2\,.\,10^{-12}\,i \\ 7\,.\,10^{-13} - 2\,.\,10^{-13}\,i & 4\,.\,10^{-13} + 8\,.\,10^{-13}\,i & \mathbf{1} \end{bmatrix}$$

The sum of the absolute values of the errors is $9.0866\,.\,10^{-12}$, while using the ***Math Pac***'s **MAT..INV** keyword results in a sum equal to ~$1.8847\,.\,10^{-11}$, which is about *207% higher*.

Regrettably, the ***Math Pac*** can't compute the complex determinant (obtained above as ***44 – 6 i***) because its **DET** keyword doesn't work with complex matrices[1]. Also, although we obtained the complex characteristic polynomial just fine, we *can't* use the **MAT..PROOT** keyword to compute the eigenvalues either because, again, it doesn't work for polynomials having complex coefficients, which is the case here. However, we can use **PZER**:

```
>OPTION BASE 1 @ COMPLEX R(3) @ CALL PZER(P,R,0,0) @ FIX 4 @ MAT DISP R
```

$\mathbf{R}^{T} = [\ 2.5815 + 1.2406\,i,\ \ 2.6226 + 4.6544\,i,\ \ -0.2041 - 2.8950\,i\ ]$, which are the eigenvalues.

[1] This limitation of **DET** has been removed by J-F Garnier in his version, ***Math Pac 2***, including many other enhancements.

Example 3: *Singular real and complex matrices.*

Find the *CP*s, determinants, inverses and all eigenvalues for each of these two matrices:

$$\mathbf{A} = \begin{bmatrix} -7 & 1 & 5 \\ 3 & -7 & 1 \\ 4 & 6 & -6 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 7-14\,\mathrm{i} & -1+2\,\mathrm{i} & -5+10\,\mathrm{i} \\ -3+6\,\mathrm{i} & 7-14\,\mathrm{i} & -1+2\,\mathrm{i} \\ 4-8\,\mathrm{i} & 6-12\,\mathrm{i} & -6+12\,\mathrm{i} \end{bmatrix}$$

We'll deal with the real matrix first:

```
>DESTROY ALL @ STD @ OPTION BASE 1 @ DIM A(3,3),M(3,3) @ OPTION BASE 0 @ DIM P(3),D
>MAT INPUT A
A(1,1)?      -7,1,5,3,-7,1,4,6,-6

>CALL PCHAR(A,P,D,M) @ MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ 20 \\ 104 \\ 0 \end{bmatrix}$$ so its characteristic polynomial is: $\boldsymbol{P(x) = x^3 + 20\,x^2 + 104\,x}$

```
>D
```

0 but its determinant is *0* so the matrix is *singular* and its inverse matrix *doesn't exist*.

Anyway, we can display the meaningless *"inverse"* matrix, just to have a look at it:

```
>MAT DISP M;
```

$$\mathbf{M} = \begin{bmatrix} -36 & -36 & -36 \\ -22 & -22 & -22 \\ -46 & -46 & -46 \end{bmatrix}$$ which is itself *singular*, as the rows are proportional.

If we then go on and compute **A*M** we get the *Zero* matrix, <u>not</u> the *Identity* matrix. As for the eigenvalues:

```
>OPTION BASE 1 @ COMPLEX R(3) @ MAT R=PROOT(P) @ MAT DISP R
```

$$\mathbf{R} = \begin{bmatrix} 0 \\ -10+2i \\ -10-2i \end{bmatrix}$$ so there's a real *0* eigenvalue and two complex conjugate ones.

Now for the complex matrix:

```
>DESTROY ALL @ STD @ OPTION BASE 1 @ COMPLEX A(3,3),M(3,3) @ OPTION BASE 0 @ COMPLEX P(3),D
>MAT INPUT A
A(1,1)?      (7,-14),(-1,2),(-5,10),(-3,6),(7,-14),(-1,2),(4,-8),(6,-12),(-6,12)

>CALL PCHAR(A,P,D,M) @ MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -8+16\,\mathrm{i} \\ 36+48\,\mathrm{i} \\ 0 \end{bmatrix}$$ so its characteristic polynomial is: $\boldsymbol{P(z) = z^3 + (-8 + 16\,i)\,z^2 + (36 + 48\,i)\,z}$

```
>D
```

(0,0) but its determinant is *(0,0)* so the matrix is *singular* and its inverse matrix *doesn't exist*. As in the real case, **A*M** is the complex *Zero* matrix, <u>not</u> the complex *Identity* matrix.

As for the complex eigenvalues, **PROOT** won't do but we can use **PZER** just fine, like this

```
>OPTION BASE 1 @ COMPLEX R(3) @ CALL PZER(P,R,0,0) @ FIX 4 @ MAT DISP R
```

$\mathbf{R}^{\mathrm{T}} = [\ 9.2915 - 18.5830\,\mathrm{i},\ -1.2915 + 2.5830\,\mathrm{i},\ 0.0000 + 0.0000\,\mathrm{i}\ ]$, which are the eigenvalues.

Example 4: *The 5x5 symmetric real matrix featured in my article "HP Article VA012", pp 6-7.*

Compute the characteristic polynomial, the determinant, the inverse and all the eigenvalues of the following 5x5 symmetric real matrix, and further check that the sum of the eigenvalues equals the *trace* (sum of the main diagonal elements) and the product of the eigenvalues equals the *determinant*:

$$\mathbf{A} = \begin{bmatrix} 5 & 1 & 2 & 0 & 4 \\ 1 & 4 & 2 & 1 & 3 \\ 2 & 2 & 5 & 4 & 0 \\ 0 & 1 & 4 & 1 & 3 \\ 4 & 3 & 0 & 3 & 4 \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ DIM A(5,5),M(5,5) @ COMPLEX R(5)
>OPTION BASE 0 @ DIM P(5),D

>MAT INPUT A
A(1,1)?      5,1,2,0,4,1,4,2,1,3,2,2,5,4,0,0,1,4,1,3,4,3,0,3,4

>CALL PCHAR(A,P,D,M)
>MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -19 \\ 79 \\ 146 \\ -1153 \\ 1222 \end{bmatrix}$$

so the characteristic polynomial is: $P(x) = x^5 - 19x^4 + 79x^3 + 146x^2 - 1153x + 1222$

and the five eigenvalues of **A** are its five roots, which we'll presently compute and display like this:

```
>MAT R=PROOT(P) @ MAT DISP R
```

$$\mathbf{R} = \begin{bmatrix} 1.49765770722 \\ 3.36187557654 \\ -3.55783865798 \\ 5.6725513961 \\ 12.0257539781 \end{bmatrix}$$

so all 5 eigenvalues are indeed real. Let's check them:

```
>S=0 @ FOR I=1 TO 5 @ S=S+REPT(R(I)) @ NEXT I @ S
```

19 thus their sum does indeed equal the trace: *5 + 4 + 5 + 1 + 4 = 19.*

```
>S=1 @ FOR I=1 TO 5 @ S=S*REPT(R(I)) @ NEXT I @ S
```

-1222 and their product does equal the computed determinant, which is:

```
>D
```

-1222 as expected; finally, the inverse matrix is:

```
>FIX 4 @ MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.2021 & -0.0106 & 0.0851 & -0.2021 & -0.0426 \\ -0.0106 & 0.3609 & 0.0360 & -0.2201 & -0.0949 \\ 0.0851 & 0.0360 & 0.0966 & 0.0687 & -0.1637 \\ -0.2021 & -0.2201 & 0.0687 & 0.1252 & 0.2733 \\ -0.0426 & -0.0949 & -0.1637 & 0.2733 & 0.1588 \end{bmatrix}$$

the inverse matrix is symmetric too.

Example 5: *The 7x7 real matrix AM#1 featured in my article "HP Article VA016 - Mean Matrices".*

As seen in the article, this is a much more difficult matrix but will be dealt with accurately and effortlessly. Let's compute the characteristic polynomial, the determinant, the inverse and all the eigenvalues of the following 7x7 all-integer real matrix, **AM#1**:

$$\mathbf{AM\#1} = \begin{bmatrix} 58 & 71 & 67 & 36 & 35 & 19 & 60 \\ 50 & 71 & 71 & 56 & 45 & 20 & 52 \\ 64 & 40 & 84 & 50 & 51 & 43 & 69 \\ 31 & 28 & 41 & 54 & 31 & 18 & 33 \\ 45 & 23 & 46 & 38 & 50 & 43 & 50 \\ 41 & 10 & 28 & 17 & 33 & 41 & 46 \\ 66 & 72 & 71 & 38 & 40 & 27 & 69 \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ DIM A(7,7),M(7,7) @ COMPLEX R(7)
>OPTION BASE 0 @ DIM P(7),D

>MAT INPUT A
A(1,1)?      58,71,67,36,35,19,60,50,71,71,56,45,20,52,64,40,84,50,51,43,69
A(4,1)?      31,28,41,54,31,18,33,45,23,46,38,50,43,50,41,10,28,17,33,41,46
A(7,1)?      66,72,71,38,40,27,69

>CALL PCHAR(A,P,D,M) @ MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -427 \\ 34952 \\ -1293342 \\ 22042627 \\ -149615896 \\ 197215484 \\ -1 \end{bmatrix}$$

so the characteristic polynomial is:

$$P(x) = x^7 - 427\,x^6 + 34952\,x^5 - 1293342\,x^4 + 22042627\,x^3 - 149615896\,x^2 + 197215484\,x - 1$$

and the seven eigenvalues of **AM#1** are its seven roots, which we now compute and display, like this:

```
>MAT R=PROOT(P) @ MAT DISP R
```

$$\mathbf{R} = \begin{bmatrix} 5.07059579484 \cdot 10^{-9} \\ 1.70591335651 \\ 14.5014291232 \\ 17.9482205987 \\ 29.8437818861 + 21.037406773\,\mathrm{i} \\ 29.8437818861 - 21.037406773\,\mathrm{i} \\ 333.156873144 \end{bmatrix}$$ so five of them are real and the other two are complex.

Notice that though the first real eigenvalue is very small it's <u>*not*</u> ***0***, else the matrix would be *singular (Det=0)*. We can check the eigenvalues as usual by computing their sum and their product:

```
>S=0 @ FOR I=1 TO 7 @ S=S+REPT(R(I)) @ NEXT I @ S
```

427 which exactly equals the trace: *58 + 71 + 84 + 54 + 50 + 41 + 69 = 427*

```
>COMPLEX T @ T=1 @ FOR I=1 TO 7 @ T=T*R(I) @ NEXT I @ T
```

(1, 1.33262749258E-13) which (negligible imaginary part aside) does equal the determinant, ***1***

```
>D
```

1 which indeed is the *exact* value. Finally, the inverse matrix is:

```
>FIX 4 @ MAT DISP M;
```

$$\mathbf{AM\#1}^{-1} = \begin{bmatrix} 96360245 & 320206 & -537449 & 2323650 & -11354863 & 30196318 & -96509411 \\ 4480 & 15 & -25 & 108 & -528 & 1404 & -4487 \\ -39436 & -131 & 220 & -951 & 4647 & -12358 & 39497 \\ 273240 & 908 & -1524 & 6589 & -32198 & 85625 & -273663 \\ -1846174 & -6135 & 10297 & -44519 & 217549 & -578534 & 1849032 \\ 13150347 & 43699 & -73346 & 317110 & -1549606 & 4120912 & -13170704 \\ -96360787 & -320208 & 537452 & -2323663 & 11354927 & -30196488 & 96509954 \end{bmatrix}$$

which is also *exact*. On the other hand, the ***HP-71 Math Pac*** produces these *highly inaccurate* results:

```
>DET(A)
```

.97095056196 instead of the correct value ***1*** we obtained above. The error is *~2.91%* , way too high.

```
>MAT M=INV(A) @ FIX 2 @ MAT DISP M;
```

$$\mathbf{M} = \begin{bmatrix} 99243204.31 & \dots & -99396833.15 \\ \dots & \dots & \dots \\ -99243762.53 & \dots & 99397392.39 \end{bmatrix}$$

which is in error by some *3%* either. The determinant of *this* inverse is computed as *1.059*, error = *5.91%*.

Note:

In this and all the following examples *(#6 - #9)* the inverse matrices and determinants returned by **PCHAR** are significantly more (even *much* more) accurate than the ones returned by the assembler keywords **MAT..INV**, **DET** (which doesn't work for complex matrices) or even **MAT..SYS** (which can be used to more accurately compute the inverse matrix), most especially when dealing with such difficult matrices as my **AM#1** *(Example 5 above)* and **AM#7** *(Example 9 below)*, where the results produced by said keywords go from very inaccurate to garbage.

Even for non-troublesome, simple integer matrices of low dimensions (as in *Examples 6*, *7* and *8 below*), **PCHAR** will produce the *exact* inverse matrix and the *exact* integer determinant where the ***Math Pac*** keywords will instead introduce rounding errors which turn the exact integer elements into *inaccurate* real elements.

Also, apart from the sheer speed inherent to their assembly-language nature, the ***Math Pac*** keywords have an intrinsic advantage that **PCHAR** doesn't have, namely they compute their results using the internally-available *15-digit* and *±50,000* exponent range numeric forms and arithmetic while **PCHAR** has to make do with the user-accessible 12-digit and ±499 exponent range numeric forms and operations.

This can be remedied by converting **PCHAR** from its original *BASIC* code into a *binary* (assembly-language) subprogram or a *LEX* keyword. See ***Section 6*** below for details.

Example 6: *A simple symmetric 3x3 real matrix, which the Math Pac handles inaccurately*

Find the characteristic polynomial, determinant, and inverse of this 3x3 real matrix:

$$\mathbf{A} = \begin{bmatrix} 60 & 30 & 20 \\ 30 & 20 & 15 \\ 20 & 15 & 12 \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ DIM A(3,3),M(3,3)
>OPTION BASE 0 @ DIM P(3),D

>MAT INPUT A
A(1,1)?      60,30,20,30,20,15,20,15,12

>CALL PCHAR(A,P,D,M)

>MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -92 \\ 635 \\ -100 \end{bmatrix}$$ so its characteristic polynomial is: $P(x) = x^3 - 92\,x^2 + 635\,x - 100$

```
>D
        100
```

and its determinant is *100* so the matrix is far from singular *(Det = 0)* and its inverse does exist, which we now proceed to display:

```
>MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.15 & -0.6 & 0.5 \\ -0.6 & 3.2 & -3 \\ 0.5 & -3 & 3 \end{bmatrix}$$ which is *exact* and *symmetric* too, as it should.

However, despite its simplicity, very small size and very small integer elements, the ***Math Pac***, working internally with 15-digit precision, nevertheless fail to produce such accurate result. The inverse comes out as:

```
>MAT M=INV(A) @ MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.149999999998 & -0.599999999991 & 0.499999999991 \\ -0.599999999988 & 3.19999999994 & -2.99999999994 \\ 0.499999999988 & -2.99999999994 & 2.99999999994 \end{bmatrix}$$

which not only is noticeably inaccurate but also is <u>*not*</u> symmetric (as it should), symmetry is somehow lost !
The determinant is computed more accurately but also fails to be an integer:

```
>DETL
        100.000000002
```

Example 7: *A very simple 2x2 complex matrix, which the Math Pac handles quite inaccurately*

Find the characteristic polynomial, determinant, inverse and all eigenvalues of this 2x2 complex matrix:

$$\mathbf{A} = \begin{bmatrix} 15 + 19\,i & 20 + 13\,i \\ 16 + 13\,i & 19 + 7\,i \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ COMPLEX A(2,2),M(2,2)
>OPTION BASE 0 @ COMPLEX P(2),D

>MAT INPUT A
A(1,1)?      (15,19),(20,13),(16,13),(19, 7)

>CALL PCHAR(A,P,D,M)

>MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -34 - 26\,i \\ 1 - 2\,i \end{bmatrix}$$ so its characteristic polynomial is: $P(z) = z^2 + (-34 - 26\,i)\,z + 1 - 2\,i$

```
>D
```

$$1 - 2\,i$$

and the (complex) determinant is *nonzero* , so the complex matrix does have an inverse, namely:

```
>MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 1 + 9\,i & 1.2 - 10.6\,i \\ 2 - 9\,i & -4.6 + 9.8\,i \end{bmatrix}$$ which is *exact.*

Again, despite its simplicity, trivial size (2x2!) and really small integer elements, the ***Math Pac***, working at 15-digit precision, still utterly fails to produce accurate results, e.g., the inverse comes out as:

```
>MAT M=INV(A) @ MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 0.999999997058 + 9.000000002\mathbf{17}\,i & 1.200000002\mathbf{98} - 10.600000002\mathbf{3}\,i \\ 2.000000003\mathbf{55} - 9.000000001\mathbf{12}\,i & -4.600000003\mathbf{62} + 9.800000001\mathbf{2}\,i \end{bmatrix}$$

which is noticeably *inaccurate*. As for the complex determinant, the ***Math Pac*** can't compute it so no way to know how accurate or inaccurate it would be (but see the footnote at the bottom of *page **5*** above).

Also, we can't get either the complex eigenvalues using **PROOT** , but **PZER** will readily deliver them, like this:

```
>OPTION BASE 1 @ COMPLEX R(2) @ CALL PZER(P,R,0,0) @ FIX 4 @ MAT DISP R
```

$$\mathbf{R} = \begin{bmatrix} 34.0099 + 26.0513\,i \\ -0.0099 - 0.0513\,i \end{bmatrix}$$, which are the complex eigenvalues.

Example 8: *A seemingly slight inaccuracy which actually isn't, unlike what the Math Pac does.*

Find the characteristic polynomial, determinant, and inverse of this 3x3 real matrix:

$$\mathbf{A} = \begin{bmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ DIM A(3,3),M(3,3) @ COMPLEX R(3)
>OPTION BASE 0 @ DIM P(3),D

>MAT INPUT A
A(1,1)?      -149,-50,-154,537,180,546,-27,-9,-25

>CALL PCHAR(A,P,D,M)

>MAT DISP P
```

$$\mathbf{P} = \begin{bmatrix} 1 \\ -6 \\ 11 \\ -6 \end{bmatrix}$$

so its characteristic polynomial is: $P(x) = x^3 - 6x^2 + 11x - 6$

```
>D
```

6 and the determinant is nonzero, thus the inverse matrix does exist, namely:

```
>MAT DISP M;
```

$$\mathbf{A}^{-1} = \begin{bmatrix} 69.0000000001 & 22.6666666667 & 70.0000000001 \\ -219.5 & -72.1666666668 & -224 \\ 4.50000000001 & 1.5 & 5.00000000001 \end{bmatrix}$$

where the five highlighted elements have *one-ulp* inaccuracies. Why is this ?

The reason is that the ***Math Pac*** includes a **MAT..(n)*A** keyword for *scalar-matrix multiplication* but it doesn't include **MAT..A/(n)** for the *matrix-divided-by-scalar* operation, so the subprogram mimics it by using **MAT..(1/n)*A** thus multiplying by ***1/6***, which can't be represented exactly and introduces the inaccuracies seen here. An *assembler* version (or coding the divide operation as a *BASIC* loop) would settle the issue for good.

The eigenvalues are computed and displayed as usual:

```
>MAT R=PROOT(P) @ MAT DISP R
```

and they're all real: ***1***, ***2*** and ***3***. As for how the ***Math Pac*** does with this matrix, we have:

```
>MAT M=INV(A) @ MAT DISP M;
```

$$\mathbf{A} = \begin{bmatrix} 68.9999991119 & 22.6666663738 & 69.9999990756 \\ -219.499997173 & -72.1666657343 & -223.999997057 \\ 4.4999999\mathbf{4123} & 1.499999\mathbf{98062} & 4.999999\mathbf{93883} \end{bmatrix}$$

which loses as many as 4-5 significant digits for this simple matrix. The determinant is likewise inaccurate:

```
>DETL
```

6.00000007752

Example 9: *The Mother of All Troublesome Matrices, my 7x7 real matrix AM#7 (see my "HP Article VA016")*

As seen in the article, this is an even more difficult matrix than **AM#1** is, despite looking quite similar. However it won't give us any more trouble here than **AM#1** did, namely *none*, but the same can't be said of the ***Math Pac***, which fails *catastrophically*. Let's compute just the determinant and the inverse of **AM#7**:

$$\mathbf{AM\#7} = \begin{bmatrix} 13 & 72 & 57 & 94 & 90 & 92 & 35 \\ 40 & 93 & 90 & 99 & 01 & 95 & 66 \\ 48 & 91 & 71 & 48 & 93 & 32 & 67 \\ 07 & 93 & 29 & 02 & 24 & 24 & 07 \\ 41 & 84 & 44 & 40 & 82 & 27 & 49 \\ 03 & 72 & 06 & 33 & 97 & 34 & 04 \\ 43 & 82 & 66 & 43 & 83 & 29 & 61 \end{bmatrix}$$

```
>DESTROY ALL @ STD
>OPTION BASE 1 @ DIM A(7,7),M(7,7)
>OPTION BASE 0 @ DIM P(7),D

>MAT INPUT A

A(1,1)?     13,72,57,94,90,92,35,40,93,90,99,1,95,66,48,91,71,48,93,32,67
A(4,1)?     7,93,29,2,24,24,7,41,84,44,40,82,27,49,3,72,6,33,97,34,4
A(7,1)?     43,82,66,43,83,29,61

>CALL PCHAR(A,P,D,M)

>D
```

1 which is the *exact* value. Now, the computed inverse matrix is:

```
>MAT DISP M;
```

$$\mathbf{AM\#7}^{-1} = \begin{bmatrix} 71082 & -507460 & -2128901626 & -36543896 & 265158513 & 3554051 & 2129774383 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ -133357 & 952047 & 3994038146 & 68560103 & -497464609 & -6667765 & -3995675528 \end{bmatrix}$$

which is also the *exact* inverse and it identically matches the one featured in the aforementioned article, where it was obtained using the powerful, arbitrary-precision software *Mathematica* running on a PC.

As usual, we can easily check its correctness by mutiplying this inverse and the original **AM#7**:

```
>MAT M=A*M @ MAT DISP M;
```

which indeed results in the *exact* 7x7 *Identity* matrix. Alas, when using the ***Math Pac*** keywords, we get:

```
>MAT M=INV(A) @ MAT DISP M;
```

which is *totally* unlike the exact inverse above. As for the determinant, it just comes out as sheer garbage:

```
>DETL
```

0.0699243217409 , instead of the correct value ***1*** we obtained above.

6. Detailed analysis for eventual conversion to Assembler

- This *BASIC* subprogram can be converted to either a *LEX* keyword (**MAT..PCHAR**) or a *binary subprogram*.

  Implementing it as a keyword would fit in with the already existing **MAT** keywords but has the added difficulty of having to implement specific parse and decompile routines for it, as it would require 4 parameters and no existing **MAT** keyword caters for that. Thus, no existing code can be reused or even mimicked, adding to the complexity of the coding itself plus the memory requirements for the *ad-hoc* parse and decompile routines.

  On the other hand, implementing it instead as a *binary subprogram* would take away all that drudge and memory requirements, as the system already caters for binary subprograms so the only thing needed would be to write the code for the functionality itself, as the parsing and decompiling would be taken care of by the *System ROM*s, as documented in the various *IDS* volumes.

  Sure enough, there are instances of this approach implemented and released by *HP*, e.g. the ***HP-71 Curve Fitting Pac*** does include *seven binary subprograms* to provide certain important functionalities, such as quick and accurate (15-digit arithmetic) evaluation of real polynomials up to the 19th degree.

  One thing worth considering is whether the built-in subprogram header parsing allows for *optional* parameters. That's not the case for *BASIC* subprograms but perhaps binary ones can allow it, in which case the user would be able to call the binary subprogram passing just **A** and **P** (disregarding the determinant and the inverse, so no need to dimension their variables and pass them to the subprogram), or just **A**, **P** and **D** (disregarding the inverse) or all four, as described here. *Note that passing less parameters doesn't save on either memory or running time; the determinant and inverse are but side effects of the main computation, a free bonus so to say.*

  Also, another possibility is not having to pass the **D** parameter to return the value of the determinant if the assembler code can use instead the internal variable **DETL** (or parameterless **DET**) to return the value. It would simply compute it and update **DETL** to hold the value. Then the call would be reduced to **CALL PCHAR(A,P,M)** and, if desired, the user would optionally execute **DET** (or **DETL**) instead of **D** to retrieve the determinant.

- The *BASIC* subprogram's body doesn't explicitly declare any variables, auxiliary or otherwise so the assembler version doesn't need to allocate extra matrices or vectors or even complex variables and such. The only implicitly created variables are **I**, **J** and **K**, used for loop indexing purposes, which could be **INTEGER** variables or even **BYTE** variables as they are always positive numbers less than 256, thus the assembler code just needs to find or allocate space for as few as 3 bytes, nothing more.

  This is so because parameters **M** and **D**, passed by reference (and thus already allocated by the calling program), are used as auxiliary variables before being ultimately assigned their correct output values at the very end.

  The 256-limit is anything but, as a 256x256 matrix would have 65,536 elements which exceeds system limits.

  Also, all loops are for byte-sized indexes going from 1 to the upper bound of the matrix row/col index, and incrementing by one each time so they should be fairly trivial to implement in the assembler code. There are no early exits either from the loops or the subprogram itself, no branching and at most 1 level of loop nesting.

  Finally, the *BASIC* code assumes that both matrices have indexes beginning at **1** (**OPTION BASE 1**) while the column vector has an index beginning at **0** (**OPTION BASE 0**). The assembler code can assume this or perhaps it could simply begin processing from the *first* element of both matrices and vector, regardless of its index.

Let's now carefully analyze how to convert the *BASIC* code to assembler code line by line:

- *Line 200:* the assembler code doesn't need to call the ROM routine for **UBND** as this returns just the upper limit of the dimension of **A**, which can be retrieved trivially.

- *Line 200:* the internal routines for **MAT..IDN** and **MAT..=** can either be called by the assembler code

or else implemented *in line*, as one is just assigning **1** to the *only* element of **A** and the other is a simple copy operation, whichever is easier or shorter.

- *Line 210:* **`MAT M=A*M`**, the assembler code just needs to call the internal routine for matrix multiplication.

- *Line 220:* this loop just computes the –*trace* of **M**, i.e.: the –sum of the diagonal elements.

- *Line 230:* this loop just increments by **D** each diagonal element of **M**.

- *Line 240:* the innards of this double loop is just a running sum of a product of some **A** and **M** elements. It avoids a much more costly full *matrix-matrix* multiplication plus an additional trace.

- *Line 250:* the operation **`MAT M=(1/`***scalar*`)* M`** is an *ad-hoc* replacement for the nonexistent keyword **`MAT M=M/(`***scalar*`)`**, and as seen in *Example 8* it might introduce unnecessary inaccuracies. The assembler code should just divide every element of **M** by that *scalar*.

The concoction **`D+NOT ABS(D)`** simply avoids division by ***0*** if matrix **A** is *singular* (and so its determinant happens to be ***0*** or ***(0,0)***) by replacing it with **1** in such cases. If **D** isn't ***0*** or ***(0,0)*** it's left unaffected, of course.

This works nicely for both real and complex **D**. The assembler code can either compute this expression by calling the necessary routines, or else check *in-line* for ***0*** or ***(0,0)*** as required, depending on the real or complex type of **D**.

Also, to compute **`D=(-1)^K*D`** the assembler code doesn't need to do any raising to the $\mathbf{K}^{th}$ power or any multiplication, it just needs to perform a simple parity check to either change the current sign of **D** or not.

This *BASIC*-code version computes the *CP*, determinant and inverse matrix about 4 times slower than the assembler-code **`MAT..INV`** keyword (which computes the inverse and determinant for real matrices but *not* the determinant of complex matrices, nor of course does it compute the *CP*). I'd expect an assembler-code version to run much faster, most specially taking into account that **`FOR..NEXT`** loops (even empty ones, and branching in general) are very slow in *BASIC*, while they should be orders of magnitude faster in assembler, particularly the ones featured here that go from 1 to a small byte-sized integer in increments of one. As the *BASIC* code has several such nested loops, the increase in speed of the converted assembler code should be *very* noticeable.

Finally, the whole computation should be performed using the internal 15-digit arithmetic as much as possible, without rounding intermediate results to the user-accessible 12-digit form until the computation is finished and the values are about to be stored in the parameters passed by reference to be returned to the user. I'm aware that this might not be possible for all intermediate computations but whenever possible it must be done so.

To wit: the optimum approach is to *minimize errors by using the 15-digit forms and internal arithmetic to the maximum extent possible*. For difficult matrices and for most any large matrix (think *least-squares* fit to a set of data) this might make the difference between useful results and unusable ones.

7. Final Conclusions

Subprogram **`PCHAR`** is a very short yet very powerful addition to any matrix-handling library, capable of quickly and accurately computing the *Characteristic Polynomial* of a real or complex square matrix, which is left stored in a format ready for computing all real/complex eigenvalues of the matrix with a single use of **`MAT..PROOT`** or **`PZER`**, so we get the task done using just *two* statements: the call to **`PCHAR`** and the execution of **`MAT..PROOT`** or **`PZER`**.

As a really *big* bonus, it will also compute real/complex *determinants* and *inverse* matrices with accuracies far exceeding in some significant cases what the assembler-code **`MAT..INV`**, **`MAT..SYS`** and **`DET`** keywords can achieve, and at no extra cost either in memory usage or computing time besides what the computation of the *CP* requires.