JACOBI – Eigenvalues of Symmetric Matrices – Jacobi's Method

Abstract

JACOBI is a program written in 1980 for the HP-41C programmable calculator to find all eigenvalues of a real NxN symmetric matrix using Jacobi's method. One worked example and two solved test cases included.

Keywords: *eigenvalues, symmetric matrix, Jacobi's method, RPN, programmable calculator, HP-41C, HP42S*

1. Introduction

JACOBI is a *215-step RPN* program that I wrote in 1980 for the *HP-41C* programmable calculator (will also run *as-is* in the *HP-41CV/CX* and in the *HP42S* with trivial changes, see ***Note 1***) to obtain numerical values for all eigenvalues of a real NxN symmetric matrix up to *22x22*.

It uses *Jacobi's method*, which annihilates in turn selected off-diagonal elements of the given matrix using elementary orthogonal transformations in an iterative fashion until all off-diagonal elements are *0* when rounded to a user-specified number of decimal places. Once this happens the diagonal elements are the eigenvalues. The method proceeds as follows:

Consider the real NxN symmetric matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1N} \\ a_{12} & a_{22} & ... & a_{2N} \\ ... & ... & ... & ... \\ a_{1N} & a_{2N} & ... & a_{NN} \end{pmatrix}$$

$\boldsymbol{A}$ is *symmetric*, this is: $a_{ij} = a_{ji}$

We want to find its eigenvalues: a given value $\boldsymbol{x}$ is an *eigenvalue* of a matrix $\boldsymbol{A}$ if:

$$\boldsymbol{A}.\bar{e} = \boldsymbol{x}.\ \bar{e} \quad , \text{ where } \bar{e} \text{ is some vector.}$$

This implies that all eigenvalues are the roots of:

$$\begin{vmatrix} a_{11} - x & a_{12} & ... & a_{1N} \\ a_{12} & a_{22} - x & ... & a_{2N} \\ ... & ... & ... & ... \\ a_{1N} & a_{2N} & ... & a_{NN} - x \end{vmatrix} = 0$$

However, expanding this NxN determinant to get the *characteristic polynomial* and then finding all N real roots of the resulting N-th degree equation, which will be the sought-for matrix eigenvalues, is rather cumbersome and present all kinds of potential difficulties.

The *Jacobi's method* does not attempt to explicitly generate and solve the equation but proceeds iteratively. It is based on the fact that an NxN symmetric matrix has exactly N real eigenvalues (not necessarily distinct). We can find another NxN matrix $\boldsymbol{S}$ so that $\boldsymbol{S}\,\boldsymbol{A}\,\boldsymbol{S}^t = \boldsymbol{D}$, where $\boldsymbol{D}$ is a *diagonal matrix* whose elements are precisely the eigenvalues of $\boldsymbol{A}$.

The method starts from the original matrix $\boldsymbol{A}$ and keeps on annihilating selected off-diagonal elements by performing elementary *rotations*. Let's single out an off-diagonal element, say $\boldsymbol{A_{pq}}$, and using an elementary rotation we'll annihilate it. The *rotation matrix* $\boldsymbol{R}$ will be:

$$R_{pp} = \cos Z\ , \quad R_{pq} = \sin Z\ , \quad R_{qp} = -\sin Z\ , \quad R_{qq} = \cos Z$$
$$R_{ii} = 1, \quad R_{pk} = R_{iq} = R_{ik} = 0 \quad \text{for } i \neq p, q \quad \text{and} \quad k \neq p, q$$

Let's denote $\boldsymbol{B} = \boldsymbol{R}^t \boldsymbol{A}\, \boldsymbol{R}$, whose elements are:

$$\left.\begin{aligned} B_{ip} &= A_{ip}\cos Z - A_{iq}\sin Z \\ B_{iq} &= A_{ip}\sin Z + A_{iq}\cos Z \\ B_{ik} &= A_{ik} \end{aligned}\right\}, \quad \text{where} \quad i, k \neq p, q$$

$$\begin{aligned} B_{pp} &= A_{pp}\cos^2 Z + A_{qq}\sin^2 Z - 2\,A_{pq}\sin Z\cos Z \\ B_{qq} &= A_{pp}\sin^2 Z + A_{qq}\cos^2 Z + 2\,A_{pq}\sin Z\cos Z \\ B_{pq} &= 0 \end{aligned}$$

and the remaining elements are symmetric. We also have:

$$\left.\begin{aligned} \sin Z &= \frac{w}{\sqrt{2\,(1+\sqrt{1-w^2})}} \\ \cos Z &= \sqrt{1-\sin^2 Z} \end{aligned}\right\}, \qquad \text{where } w = \frac{sign(M)\,.\,L}{\sqrt{M^2+L^2}} \text{ and } L = -A_{pq}\,, \qquad M = \tfrac{1}{2}(A_{pp} - A_{qq})$$

This is iterated using a strategy for selecting each non-diagonal element in turn until all non-diagonal elements are zero when rounded to a user-specified number of decimal places. Once this happens the matrix diagonal contains the eigenvalues.

1.1 Program characteristics

This program is 46 registers long and can handle symmetric matrices from *2x2* to *22x22*, both limits included. If a bare-bones *HP-41C* is used (no memory modules), matrices up to *3x3* may be treated[1]. On the other hand, if plugging all 4 memory modules in (or using the *HP-41CV/CX*) matrices up to *22x22* can be dealt with.

All inputs and outputs are labeled and there's a warning to prevent a memory allocation error (**SIZE**) .

The accuracy and running times depend on the display setting, however the computed eigenvalues are often more accurate than that: for instance, using **FIX 5** is quite possible to get 7-8 correct digits.

The program has been written to be as fast as possible and to fit into a minimum amount of program memory. For example, the matrix is stored taking into account its symmetry ($\boldsymbol{a_{ij}}$ is stored only for $i \leq j$) and this results in fast execution times and less storage registers needed, as the NxN matrix uses $(N^2+N)/2$ registers instead of N^2. The minimum **SIZE** to work with an NxN matrix is **SIZE (N²+N)/2+10**.

Running times depend on the display setting as well as the matrix itself. However, the method converges always and is stable against rounding errors. The following times for the original, physical *HP-41C* are indicative:

2x2	*3x3*	*4x4*	*5x5*	*6x6*
8 sec.	50 sec.	2,5 min.	6 min.	10 min.

[1] Up to 4x4 if using *synthetic instructions*, in which case be careful not to insert a final **END** instruction, let the final **.END.** of program memory do the job instead or 4x4 matrices won't be feasible. See the footnote in the following page.

2. Program Listing[1]

```
01 ♦LBL "JB"
02  FIX 0
03  CF 29
04  "N=?"
05  PROMPT
06  STO 00
07  ENTER↑
08  X↑2
09   +
10   2
11   /
12   9
13   +
14  SF 25
15  RCL IND X
16  FS?C 25
17  GTO 00 ▸
18  "SIZE "
19   1
20   +
21  ARCL X
22  PROMPT
23 ♦LBL 00
24   10
25  STO 04
26  RCL 00
27  1 E3
28   /
29   1
30   +
31  STO 02
32 ♦LBL 12
33  RCL 02
34  STO 03
35 ♦LBL 07
36  "A"
37  ARCL 02
38  "├,"
39  ARCL 03
40  "├=?"
41  PROMPT
42  STO IND 04
43  ISG 04
44  X<>Y
45  ISG 03
46  GTO 07 ▸
47  ISG 02
48  GTO 12 ▸
49  "FIX=?"
50  PROMPT
51  FIX IND X
52 ♦LBL 70
53   2
54  STO 03
55 ♦LBL 85
56   1
57 ♦LBL 87
58  STO 02
59  CF 00
60  RCL 03
61  XEQ 90 ▸
62  X<>Y
63  RND
64  X=0?
65  GTO 84 ▸
66  SF 00
67  LASTX
68  ST- IND Z
69  STO 01
70  ST+ 01
71  CHS
72  STO 06
73  RCL 02
74  RCL 02
75  XEQ 90 ▸
76  STO 08
77  RDN
78  RCL 03
79  RCL 03
80  XEQ 90 ▸
81  STO 09
82  RDN
83   -
84  STO 07
85   2
86   /
87  SIGN
88  RCL 06
89  LASTX
90  X↑2
91  RCL 06
92  X↑2
93   +
94  SQRT
95   /
96   *
97   1
98  RCL Y
99  X↑2
100  -
101 SQRT
102  1
103  +
104 ST+ X
105 SQRT
106  /
107 STO 06
108 ST* 01
109 X↑2
110 ST* 07
111  1
112 STO 04
113 X<>Y
114  -
115 SQRT
116 ST* 01
117 X<> 07
118 RCL 01
119  +
120 ST- IND 08
121 ST+ IND 09
122 ♦LBL 01
123 RCL 03
124 RCL 04
125 X=Y?
126 GTO 08 ▸
127 RCL 02
128 X=Y?
129 GTO 08 ▸
130 XEQ 90 ▸
131 STO 05
132 RDN
133 STO 08
134 RCL 07
135 STO 09
136  *
137 RCL 03
138 RCL 04
139 XEQ 90 ▸
140 STO T
141 RDN
142 ST* 09
143 RCL 06
144 ST* 08
145  *
146  -
147 STO IND 05
148 RCL 08
149 RCL 09
150  +
151 STO IND Z
152 RCL 04
153 RCL 00
154 X≤Y?
155 GTO 84 ▸
156 ♦LBL 08
157  1
158 ST+ 04
159 RCL 00
160 RCL 04
161 X≤Y?
162 GTO 01 ▸
163 ♦LBL 84
164 RCL 03
165 RCL 02
166  1
167  +
168 X≠Y?
169 GTO 87 ▸
170 X<>Y
171 RCL 00
172 X≠Y?
173 ISG 03
174 X<>Y
175 X≠Y?
176 GTO 85 ▸
177 FS?C 00
178 GTO 70 ▸
179 RCL 00
180 1 E3
181  /
182  1
183  +
184 STO 06
185 ♦LBL 13
186 "X="
187 RCL 06
188 INT
189 ENTER↑
190 XEQ 90 ▸
191 X<>Y
192 ARCL X
193 PROMPT
194 ISG 06
195 GTO 13 ▸
196 RTN
197 ♦LBL 90
198 X>Y?
199 X<>Y
200 RCL 00
201 ST+ X
202 X<>Y
203  -
204  1
205 ST- L
206 X<> L
207  *
208  2
209  /
210  +
211  9
212  +
213 RCL IND X
214 X<>Y
215 .END.
```

-215 program steps

- uses $R_{00} - R_{09}$ as auxiliary registers, plus the ones used to store the matrix

- uses flag 00

- FIX 5 setting is recommended

*The symbols ♦ and ▸ are purely cosmetic, to indicate branching; ├ is the **Append** alpha function; = see footnote.*

[1] If you're able to use *synthetic instructions* you can save three additional registers for other purposes by using registers **M, N** and **O** instead of ***R_{07} -R_{09}*** , respectively. To that effect, perform the following changes in the program listing above:

- change all instructions using registers **07/08/09** to use instead registers **M/N/O**, respectively.
- change line *12* **9** to *12* **6** , line *24* **10** to *24* **7** and line *211* **9** to *211* **6**
- the size required for NxN matrices will now be **SIZE (N²+N)/2+7** and 4x4 will be possible with no extra *RAM*.

3. Usage Instructions

To find all the eigenvalues of a real NxN symmetric matrix ***A***, follow these instructions:

In **RUN** Mode, execute the program:

XEQ "JB"	→	*N=?*	*{ asks for the size of the matrix }*
N **R/S**	→	*Ai, j=?*	*{ asks for the matrix elements in turn, from left to right and from top to bottom }*
a_{ij} **R/S**	→	*...*	*{ due to symmetry, the program will only ask for the upper-half elements }*
	→	*FIX=?*	*{ asks for the desired number of decimal places (accuracy and run time depend on this value), specifying 5 decimals is recommended }*
dec. **R/S**	→	*...*	*{ the program runs until all eigenvalues have been computed, and finally ... }*
...	→	*X=(1st eigenvalue)*	
R/S	→	*X=(2nd eigenvalue)*	**R/S** → ... **R/S** → *X=(Nth eigenvalue)*

All eigenvalues are displayed rounded to the specified number of decimals. However, each value is also placed without rounding in the X stack register so they can be seen to any number ***n*** of decimals by using **FIX *n***. For another matrix, repeat the procedure above.

Notes:

- if after introducing N a message ***SIZE nnn*** does show up this means the current register allocation is insufficient to run the program so simply execute **SIZE** *nnn* as directed and then press **R/S** to resume. In general, if the matrix dimension is N, a minimum of $(N^2+N)/2+10$ registers are required.
- input of the matrix elements may use the stack but don't disturb the stack while the output is taking place.
- accuracy and running time both depend on the number of digits ***d***. As a guideline, use ***d = 5*** to speed things up.

4. Examples

The following examples can be useful to check that the program is correctly entered and to understand its usage.

4.1 Example 1

Find all the eigenvalues of the following 4x4 matrix. Specify 5 decimal digits.

$$\boldsymbol{A} = \begin{pmatrix} \mathbf{25} & \mathbf{-41} & \mathbf{10} & \mathbf{-6} \\ -41 & \mathbf{68} & \mathbf{-17} & \mathbf{10} \\ 10 & -17 & \mathbf{5} & \mathbf{-3} \\ -6 & 10 & -3 & \mathbf{2} \end{pmatrix}$$

XEQ "JB"	→ *N=?*													
4 **R/S**	→ *A1,1=?*	25	**R/S**	→	*A1,2=?*	-41	**R/S**	→	*A1,3=?*	10	**R/S**	→	*A1,4=?*	-6
R/S	→ *A2,2=?*	68	**R/S**	→	*A2,3=?*	-17	**R/S**	→	*A2,4=?*	10				
R/S	→ *A3,3=?*	5	**R/S**	→	*A3,4=?*	-3								
R/S	→ *A4,4=?*	2												
R/S	→ *FIX=?*	5	**R/S**	→	... *{ and after 2 min. 22 sec. the eigenvalues are displayed }*									
R/S	→ *X=0.03302*													
R/S	→ *X=98.52170*		*these are the four eigenvalues; they're correct to more than the digits shown*											
R/S	→ *X=1.18609*													
R/S	→ *X=0.25920*													

4.2 Test case 1

Compute the eigenvalues of the following 3x3 matrix. Specify 8 decimal digits.

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & \frac{1}{2} \\ 1 & 1 & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{4} & 2 \end{pmatrix}$$

The program should give: $X_1 = 2.53652586$, $X_2 = -0.01664728$, $X_3 = 1.48012142$, in about 50 sec.

4.3 Test case 2

Compute the eigenvalues of the following 5x5 matrix. Specify 5 decimal digits.

$$\boldsymbol{A} = \begin{pmatrix} 1 & -2 & 4 & 3 & 6 \\ -2 & 2 & -3 & 0 & -1 \\ 4 & -3 & 3 & 6 & 4 \\ 3 & 0 & 6 & 5 & 2 \\ 6 & -1 & 4 & 2 & -2 \end{pmatrix}$$

The program should give: $X_1 = 0.61259$, $X_2 = 15.39409$, $X_3 = -3.23854$,
$X_4 = 3.11890$, $X_5 = -6.88703$, in about 5 min. 57 sec.

All values can be tested: simply subtract the eigenvalue from each diagonal element and check that ***Det(A) ~ 0***.

Notes

*1.*This program can easily be adapted to run under other *RPN* versions with minimal changes, e.g. to run it on the *HP42S* simply change the *names* of intructions such as **FIX 0** to **FIX 00**, **∗** to **x**, **ST+/ST-/**... to **STO+/STO-/**... , **ARCL X** to **ARCL ST X** and so on. See the ***HP42S Owner's Manual*** section titled *"Using HP-41C Programs"* for full details. The size of the resulting program for the *HP42S* should be 337 bytes.

Also, the *HP42S* is much faster and has a wider numeric range and greater precision (12-digit instead of 10-digit), so the results might differ slightly. If running the resulting *HP42S* program on the *Free42* emulator (which is thousands of times faster than the original, physical *HP-41C*) all ***Examples*** above run almost instantly.

2. This program was published in *PPC Technical Notes V1N3 pp9-13 (October 1980).*

References

Francis Scheid (1988). *Schaum's Outline of Theory and Problems of Numerical Analysis, 2nd Edition.*

Hewlett-Packard (1988). *HP42S Owner's Manual ("Using HP-41C Programs" section), Edition 1 June 1988.*

Copyrights