Proceedings of the Third International DERIVE/TI-92 Conference

Linear Algebra Activities on the TI-92 Plus

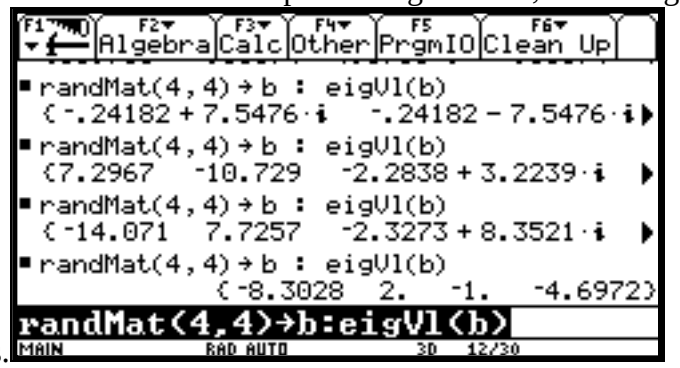
Dr. Dennis D. Pence
Department of Mathematics and Statistics
Western Michigan University
Kalamazoo, MI 49008-5152 USA
dennis.pence@wmich.edu

Abstract

Texas Instruments has announced the Plus Module for the TI-92 with Advanced Mathematical Software. Among the many improvements offered on this plug-in module are substantial new capabilities for matrix computations. We will explore some of the ways to combine the symbolic, numerical, and graphical capabilities of the TI-92 Plus for linear algebra topics.

Eigenvalues and Eigenvectors

It is nicer to start with real eigenvalues. For a random 4×4 matrix (using integer entries from –9 to 9) this does not immediately happen. However, if we repeated execute the command line to generate a random matrix and compute its eigenvalues, we soon get an example with all real eigenvectors.



Continuing, we can show how this matrix can be diagonalized knowing the eigenvalues and eigenvectors.

■ b

$$\begin{bmatrix} 3 & -4 & -3 & -1 \\ 4 & -5 & -8 & -8 \\ 9 & -9 & -6 & 2 \\ -4 & 4 & 2 & -4 \end{bmatrix}$$

■ Define d = diag(eigVl(b)) : d

$$\begin{bmatrix} -8.3028 & 0 & 0 & 0 \\ 0 & 2. & 0 & 0 \\ 0 & 0 & -1. & 0 \\ 0 & 0 & 0 & -4.6972 \end{bmatrix}$$

■ Define y = eigVc(b) : y

$$\begin{bmatrix} .30031 & -.30066 & -.70711 & -.14441 \\ .39517 & .46256 & -.70711 & .1075 \\ .75019 & -.79791 & 2.5382\text{E}{-13} & -.73264 \\ -.43688 & .24284 & -8.4233\text{E}{-14} & .65637 \end{bmatrix}$$

■ $y \cdot d \cdot y^{-1}$

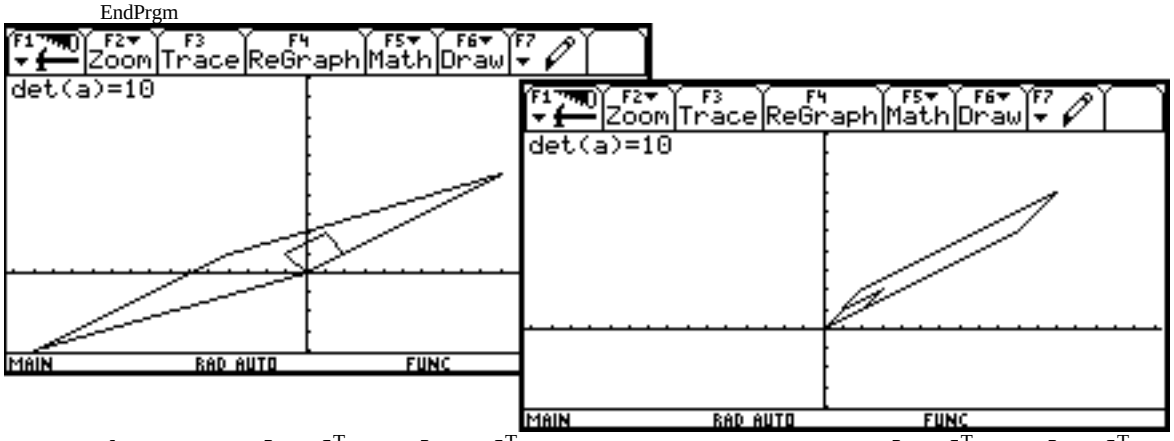
$$\begin{bmatrix} 3. & -4. & -3. & -1. \\ 4. & -5. & -8. & -8. \\ 9. & -9. & -6. & 2. \\ -4. & 4. & 2. & -4. \end{bmatrix}$$

For 2×2 matrices, it is nice to add eigenvectors to the graphical understanding of a linear transformation. First we create a matrix with known eigenvalues and eigenvectors by reversing the steps above.

$$\text{Define } x=\begin{bmatrix}1 & 2\\ 1 & 1\end{bmatrix} : \text{Define } a = x\cdot\text{diag}(\langle 2 \quad 5\rangle)\cdot x^{-1} : a \qquad \begin{bmatrix}8 & -6\\ 3 & -1\end{bmatrix}$$

Then we use the following TI–92 program to display an initial parallelogram with the input side vectors **b** = [b1, b2]^T and **c** = [c1, c2]^T and the transformed parallelogram with side vectors **A b** and **A c.**

```
parallel()
Prgm
Local b1,b2,c1,c2,tb1,tb2,tc1,tc2,atemp
ClrIO:ClrDraw:ClrGraph:FnOff
Prompt b1,b2,c1,c2
Line 0,0,b1,b2:Line 0,0,c1,c2
Line b1,b2,b1+c1,b2+c2:Line c1,c2,b1+c1,b2+c2
Pause
a*[[b1][b2]]>>atemp:atemp[1,1]>>tb1:atemp[2,1]>>tb2
a*[[b1][b2]]>>atemp:atemp[1,1]>>tc1:atemp[2,1]>>tc2
Line 0,0,tb1,tb2:Line 0,0,tc1,tc2
Line tb1,tb2,tb1+tc1,tb2+tc2:Line tc1,tc2,tb1+tc1,tb2+tc2
PxlText "det(a)="&string(det(a)),1,1
EndPrgm
```



input arbitrary **b** = [2, 1]^T, **c** = [–1, 1]^T input eigenvectors **b** = [1, 1]^T, **c** = [2, 1]^T

For any input vectors, a parallelogram is transformed into another parallelogram, with the det(a) giving the factor of change for the area of the transformed parallelogram. However inputting the eigenvectors gives a much clearer picture of the ways things are "stretched" in the transformation.

Solving Systems of Linear Equations

The recommended method for numerically solving a nonsingular matrix equation **A x** = **b** consists of the following steps:

1. Use Gauss elimination with partial pivoting to decompose the matrix **A** so that **L U** = **P A** where **L** is a lower triangular matrix with ones on the diagonal, **U** is an upper triangular matrix, and **P** is a permutation matrix chosen to minimize rounding errors in the elimination process.
2. Permute the entries of **b** (e.g. find **c** = **P b**).
3. Use forward substitution to solve the triangular system **L y** = **c**.
4. Use backward substitution to solve the triangular system **U x** = **y**.

Since the TI-92 Plus provides the LU decomposition (in both exact and numerical form), we can explore these steps individually. Consider a 5×5 Hilbert matrix as the matrix $\mathbf{A} = \{a_{ij}\} = \{1/(i+j-1)\}$.

Hilbert matrices are frequently used as test matrices (see N. J Higham, *Accuracy and Stability in Numerical Algorithms*, SIAM, 1996, Chapter 26). Even though an exact formula is known for the inverse of a Hilbert matrix (and all entries are integers), the condition number for a Hilbert matrix is very large and numerical solutions will be subject to significant rounding errors. We can explore this in several ways. First we create a right-hand-side **b** which is the floating-point approximation for a system which has exact solution $\mathbf{x} = [1, 1, 1, 1, 1]^T$. Then we numerically go through the solution steps to try to recover **x**.

```
▪ h                                   [1    1/2  1/3  1/4  1/5]
                                      [1/2  1/3  1/4  1/5  1/6]
                                      [1/3  1/4  1/5  1/6  1/7]
                                      [1/4  1/5  1/6  1/7  1/8]
                                      [1/5  1/6  1/7  1/8  1/9]

▪ approx(h·[1,1,1,1,1]) → b           [2.28333333333]
                                      [1.45          ]
                                      [1.09285714286 ]
                                      [.884523809524 ]
                                      [.745634920635 ]
▪ LU approx(h), l, u, p                          Done
▪ Define c=p·b
▪ Define y=simult(l,c)
▪ Define x=simult(u,y) : x            [.999999999986]
                                      [1.00000000027 ]
                                      [.999999998835 ]
                                      [1.00000000175 ]
                                      [.999999999149 ]
```

We can see the effects of rounding in floating-point computations by how the final results differ from the exact solution $\mathbf{x} = [1, 1, 1, 1, 1]^T$. If we add the variable *e* to the first component of **b** and redo the computations, we can see how a change in this component effects the final solution.

```
Define be=b+[e;0;0;0;0]:be
Define xe=simult(h,be):xe
```

$$\begin{bmatrix} 25\cdot e + .999999999985 \\ 1.00000000028 - 300\cdot e \\ 1050\cdot e + .999999998778 \\ 1.00000000184 - 1400\cdot e \\ 630\cdot(e + .001587301586) \end{bmatrix}$$

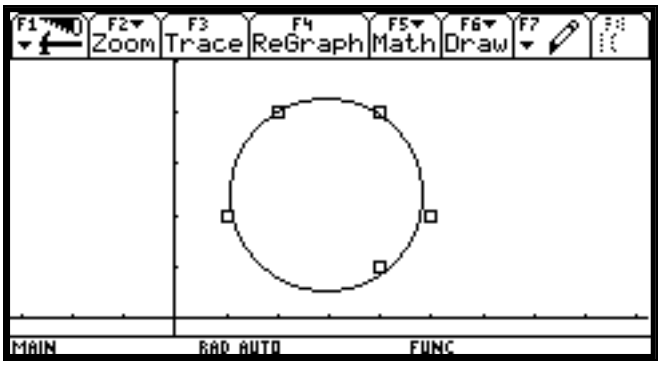
Orthogonal Decompositions

While the LU decomposition still works for a nonsquare matrix, a far more useful matrix factorization is the QR decomposition. Here $\mathbf{A} = \mathbf{Q}\,\mathbf{R}$ where **Q** is a matrix with orthogonal columns and **R** is an upper triangular square matrix. If we wish to "solve $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ in a least square error sense," this can easily be done by solving $\mathbf{R}\,\mathbf{x} = \mathbf{Q}^T\,\mathbf{b}$ in the case where **A** has full column rank (i.e. when **R** is nonsingular).

Suppose we are given the coordinates for five points in the plane which nearly lie on circle. For example, consider {(4, 1), (5, 2), (4, 4), (2, 4), (1, 2)}. Then we can set up the five equations in three unknowns that we would like to solve in a least square sense.

$(x-h)^2 + (y-k)^2 = r^2$ becomes $2\,x\,h + 2\,y\,k + c = x^2 + y^2$ where $c = r^2 - h^2 - k^2$.

$$\begin{bmatrix} 8 & 2 & 1 \\ 10 & 4 & 1 \\ 8 & 8 & 1 \\ 4 & 8 & 1 \\ 2 & 4 & 1 \end{bmatrix} \begin{bmatrix} h \\ k \\ c \end{bmatrix} = \begin{bmatrix} 17 \\ 29 \\ 32 \\ 20 \\ 5 \end{bmatrix}$$



If we define **A** and **b** for this system, give the command QR a,q,r : simult(r,q^T*b) we find there that $h = 279/94 \approx 2.96808510638$, $k = 453/188 \approx 2.40957446809$, and $c = -1027/94 \approx -10.9255319149$. This implies that $r = \sqrt{(130421)}/188 \approx 1.9209493489$. We have used a statistical plot of the data and the Circle command to get a plot in the figure above.