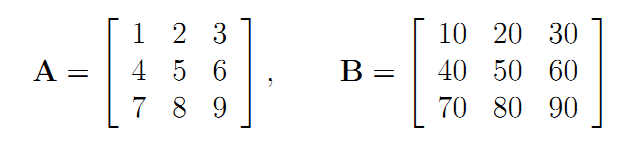
Numerical Linear Algebra ----MatLab

Sep 20 2011

Implements the following experiments and paste the numerical results and matlab commands at each step of the experiments.

* Matrices and linear equations

1. Produce two matrices

.

>> A=[1 2 3; 4 5 6; 7 8 9]

A =

1 2 3

4 5 6

7 8 9

>> B=A\*10

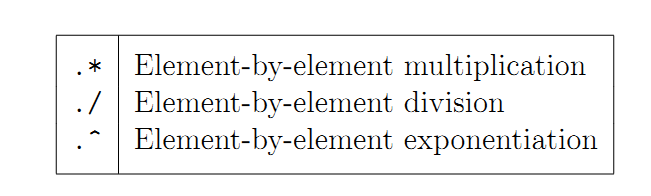
B =

10 20 30

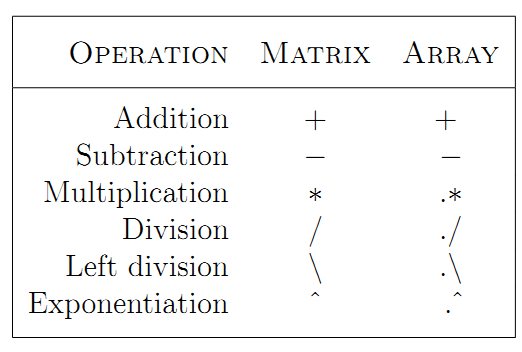
40 50 60

70 80 90

1. The array operations are given as follows:



1. The matrix operations are given as follows:



1. Compute A\*B, A.\*B, A./B, A.^2, A.^B.

>> A\*B

ans =

300 360 420

660 810 960

1020 1260 1500

>> A.\*B

ans =

10 40 90

160 250 360

490 640 810

>> A./B

ans =

0.1000 0.1000 0.1000

0.1000 0.1000 0.1000

0.1000 0.1000 0.1000

>> A.^2

ans =

1 4 9

16 25 36

49 64 81

>> A.^B

ans =

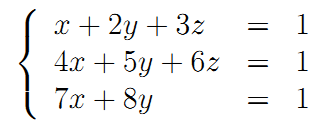
1.0e+085 \*

0.0000 0.0000 0.0000

0.0000 0.0000 0.0000

0.0000 0.0000 7.6177

1. Convert the following linear system into a matrix multiplication AX =b

.

1. Solve it by calculating the inverse of the system matrix A;
2. Use \ to solve the linear system.

>> A=[1 2 3; 4 5 6; 7 8 0];

>> b=[1 1 1]';

>> inv(A)\*b

ans =

-1.0000

1.0000

-0.0000

>> A\b

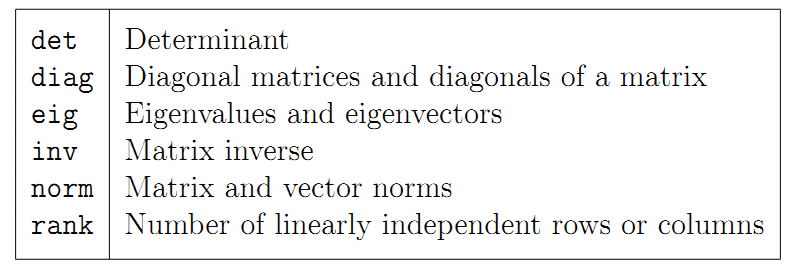
ans =

-1.0000

1.0000

-0.0000

1. Use the online helps to find how to use these functions

.

Calculate det(A), diagonals of A, a diagonal matrix through using diag to set diagonal elements as the first column elements of A), the 2-norm and Frobenius norms of A and 1-norm of A(:), and the rank of A.

>> A=[1 2 3; 4 5 6; 7 8 9];

>> det(A)

ans =

6.6613e-016

>> diag(A)

ans =

1

5

9

>> diag(A(:,1))

ans =

1 0 0

0 4 0

0 0 7

>> norm(A,2)

ans =

16.8481

>> norm(A,'fro')

ans =

16.8819

>> norm(A(:),1)

ans =

45.0000

1. Find the trace of A.(use online help).

>> trace(A)

ans =

15

* Experiment 1: Discrete Legendre Polynomials

1. Create a Vandermonde matrix A by discretizing [-1, 1] by 257 equally spaced points.
2. Find is the reduced QR factorization of A (use the matlab commond qr and read the online helps on qr).
3. Rescale the matrix Q by the last the row of Q. Plot the columns of the rescaled Q on a figure.

(1)

>> x[-1:1/128:1]'; % Consider if we can use other matlab commend to substitute this line

>> Vander\_A=[x.^0 x.^1 x.^2 x.^3];

See Vander\_A.mat

(2)

>> [Q R]=qr(Vander\_A,0);

See Q.mat

R =

-16.0312 -0.0000 -5.3855 0

0 9.2917 0.0000 5.6185

0 0 -4.8168 -0.0000

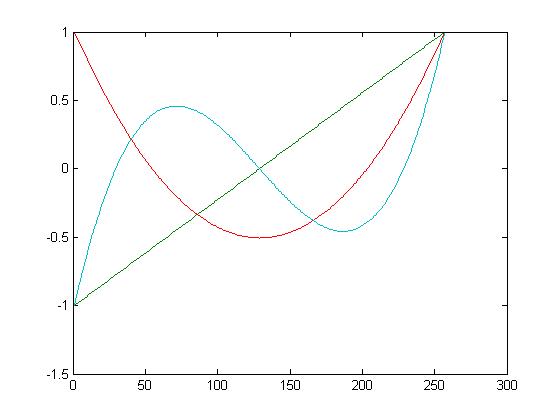
0 0 0 2.4519

(3)

>> scale=Q(257,:);

>> Q=Q\*diag(1./scale);

>> plot(Q)



* Experiment 2: Classical vs. Modified Gram-Schmidt (read the textbook)

1. Set U and V to random orthogonal matrices. (see textbook)
2. Set a diagonal matrix S with exponentially graded entries S = diag(2.^(-1:-1:-80)).
3. Set A = USV.
4. Compute QR factorization by using the classic and modified Gram-Schmidt (Use matlab commands clgs and mgs. Read the online helps of each command).
5. Plot the diagonal elements rjj produced by both computations on a logarithmic scaled figure.
6. Summarize the observations from the figure? What does cause the different plots of Classical and Modified Gram-Schmidt?

(1)(2)(3)

>> [U,X]=qr(randn(80));

>> [V,X]=qr(randn(80));

>> S=diag(2.^(-1:-1:-80));

>> A= U\* S\*V;

See Q.mat, V.mat, S.mat, A.mat

(4)

See clgs.m, msg.m

(5)

>> [Q1,R1]=clgs(A);

>> [Q2,R2]=mgs(A);

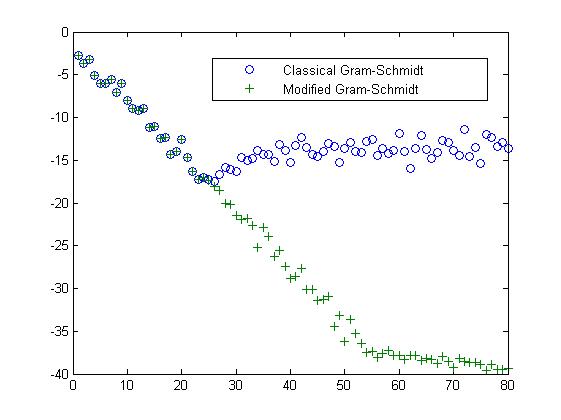
>> r1=log(diag(R1));

>> r2=log(diag(R2));

>> lgth1=[1:length(r1)];

>> lgth2=[1:length(r2)];

>> plot(lgth1,r1,'o',lgth2,r2,'+');%Results are clearly presented in the graph. Please label the axes next in the later project



(6)

There is a big bias of the two lines start from about the 25th point. And the consistency of MGS looks much better than the CLGS.

I think this situation happens because of the stability of these two different algorithm, the stability of MGS is much better than the CLGS, and when computer computing, especially when do division, it may occur errors and the denominator is too small, but the computer will only get part of the results and omit the rest, which depends on the default settings of computer.

* Experiment 3: Numerical loss of orthogonality

1. Generate a rank-deficient matrix as follows:

A = [0.7000, 0.7011; 0.70001, 0.70711 ];

1. Compute the QR factorization by using the matlab command qr.
2. Compute the QR factorization by using Modified G-S.
3. Test the orthogonality of Q matrices obtained from step 2 and 3 respectively. Make your conclusion based on the observations of the orthogonality tests.

(1)(2)(3)

>> A = [0.7000,0.7011;0.70001,0.70711];

>> [Q R]=qr(A)

Q =

-0.707101730441863 -0.707111831895155

-0.707111831895155 0.707101730441863

R =

-0.989956564754232 -0.995754870664174

0 0.004242599271053

>> a=norm(Q'\*Q-eye(2))

a =

2.351490101248793e-016

>> [Q1 R1]=mgs(A)

Q1 =

0.707101730441863 -0.707111831895160

0.707111831895155 0.707101730441859

R1 =

0.989956564754232 0.995754870664174

0 0.004242599271053

>> b=norm(Q1'\*Q1-eye(2))

b =

6.827871601444713e-015

(4)

>> Q(:,1)'\*Q(:,2)

ans =

-5.551115123125783e-017

>> Q1(:,1)'\*Q1(:,2)

ans =

-6.827871601444713e-015

The results are very small, but not zero, this is because the computer will treat these small numbers or results as zero when they are smaller than some default tiny number.

Comments:

Shuowen, you are one the best students in my session. The codes are logically organized and well written. The only issue is the highlight part. I got different results after running your code. (The results are shown as follows) Therefore, the conclusion of Experiment 3 is not totally correct. Please let me know if you have any concern about the class.

>> Q(:,1)'\*Q(:,2)

ans = 2.3014e-011