When MATLAB Gives “Wrong” Answers

Steven Leon

UMassDartmouth

June 17, 2008

Numerical Linear Algebra for Undergraduates

Undergraduate courses involving Numerical Linear Algebra (NLA)

- A full semester course in Numerical Linear Algebra
- NLA covered as part of a second course in linear algebra (3 to 7 weeks)
- NLA covered as part of a course in Numerical Analysis (2 to 3 weeks)

How NLA differs from LA

- NLA is taught from an algorithmic approach

How NLA differs from LA

- NLA is taught from an algorithmic approach
- NLA is usually taught using MATLAB

How NLA differs from LA

- NLA is taught from an algorithmic approach
- NLA is usually taught using MATLAB
- Algorithms are carried out in finite precision

How NLA differs from LA

- NLA is taught from an algorithmic approach
- NLA is usually taught using MATLAB
- Algorithms are carried out in finite precision
- Important concerns
  - Digits of accuracy
  - Conditioning
  - Numerical rank

The Challenge

- How do we teach about finite precision arithmetic and still make the course interesting?

The Challenge

- How do we teach about finite precision arithmetic and still make the course interesting?
- Proposed solution: Use examples where MATLAB gives or appears to give wrong answers

Some Simple Examples

Logical Statements
In MATLAB set $p = 1e - 20$ and enter the command

$$p > 0$$

MATLAB responds

- ans =

  $$1$$

Enter the command $1 + p > 1$, MATLAB responds

Some Simple Examples

Logical Statements

In MATLAB set $p = 1e - 20$ and enter the command

$$p > 0$$

MATLAB responds

- ans =

  1

Enter the command $1 + p > 1$, MATLAB responds

- ans =

  0

Some Simple Examples

Logical Statements
In MATLAB set $p = 1e - 20$ and enter the command

$$p > 0$$

MATLAB responds

- ans =

  1

Enter the command $1 + p > 1$, MATLAB responds

- ans =

  0

- Use this example to motivate discussion of the machine epsilon

Types of “wrong” answers one may encounter using MATLAB

Five scenarios

- MATLAB gives wrong answers but the fault is not with the software

Types of "wrong" answers one may encounter using MATLAB

Five scenarios

- MATLAB gives wrong answers but the fault is not with the software
- MATLAB may give answers that appear to be wrong but are actually correct

Types of "wrong" answers one may encounter using MATLAB

Five scenarios

- MATLAB gives wrong answers but the fault is not with the software
- MATLAB may give answers that appear to be wrong but are actually correct
- MATLAB gives wrong answers (but it may not matter)

Types of “wrong” answers one may encounter using MATLAB

Five scenarios

- MATLAB gives wrong answers but the fault is not with the software
- MATLAB may give answers that appear to be wrong but are actually correct
- MATLAB gives wrong answers (but it may not matter)
- MATLAB may fail to give important information

Types of “wrong” answers one may encounter using MATLAB

Five scenarios

- MATLAB gives wrong answers but the fault is not with the software
- MATLAB may give answers that appear to be wrong but are actually correct
- MATLAB gives wrong answers (but it may not matter)
- MATLAB may fail to give important information
- The Heisenberg effect: mind-boggling results that don’t seem to make any sense at all

Don't blame it on MATLAB

- User mistakes (Example: first version of ATLAST cogame utility)
- Early Pentium processors (1994) -Bug in floating point division algorithm
  If $y \neq 0$ and $q = \frac{x}{y}$, then we would expect that

$$z = x - qy = x - (\frac{x}{y})y = 0$$

The Pentium Processor Bug

If $x = 5244795$ and $y = 3932159$ and set $q = \frac{x}{y}$ and $z = x - qy$, then the computed values in MATLAB are

- $q = 1.333820682225719$ and $z = 0$

On the early pentium computers the quotient came out to be

$$w = 1.333755578042495.$$

Using w in place of q and computing $z = x - wy$ we get

The Pentium Processor Bug

If $x = 5244795$ and $y = 3932159$ and set $q = \frac{x}{y}$ and $z = x - qy$, then the computed values in MATLAB are

- $q = 1.333820682225719$ and $z = 0$

On the early pentium computers the quotient came out to be

$$w = 1.333755578042495.$$

Using w in place of q and computing $z = x - wy$ we get

- $z = 256$

Correct answers that appear to be wrong

- Singular matrices and determinants - Vandermonde example
- Nonsingular matrices: Hilbert and Pascal matrices

Determinants – Vandermonde Example

- Set V = vander(1:6); V = V − diag(sum(V'))

$$V = \begin{pmatrix} -5 & 1 & 1 & 1 & 1 & 1 \\ 32 & -47 & 8 & 4 & 2 & 1 \\ 243 & 81 & -337 & 9 & 3 & 1 \\ 1024 & 256 & 64 & -1349 & 4 & 1 \\ 3125 & 625 & 125 & 25 & -3901 & 1 \\ 7776 & 1296 & 216 & 36 & 6 & -9330 \end{pmatrix}$$

Determinants – Vandermonde Example

- Set V = vander(1:6); V = V − diag(sum(V'))

$$V = \begin{pmatrix} -5 & 1 & 1 & 1 & 1 & 1 \\ 32 & -47 & 8 & 4 & 2 & 1 \\ 243 & 81 & -337 & 9 & 3 & 1 \\ 1024 & 256 & 64 & -1349 & 4 & 1 \\ 3125 & 625 & 125 & 25 & -3901 & 1 \\ 7776 & 1296 & 216 & 36 & 6 & -9330 \end{pmatrix}$$

- rank(V) = 5 and if we set $\mathbf{x}$= ones(6,1) then $V\mathbf{x} = \mathbf{0}$, however, the command $d = \det(V)$ yields

Determinants – Vandermonde Example

- Set V = vander(1:6); V = V − diag(sum(V'))

$$V = \begin{pmatrix} -5 & 1 & 1 & 1 & 1 & 1 \\ 32 & -47 & 8 & 4 & 2 & 1 \\ 243 & 81 & -337 & 9 & 3 & 1 \\ 1024 & 256 & 64 & -1349 & 4 & 1 \\ 3125 & 625 & 125 & 25 & -3901 & 1 \\ 7776 & 1296 & 216 & 36 & 6 & -9330 \end{pmatrix}$$

- rank(V) = 5 and if we set $\mathbf{x}$= ones(6,1) then $V\mathbf{x} = \mathbf{0}$, however, the command $d = \det(V)$ yields
- $d = 10$ — What is going on?

Vandermonde Example Explained

The MATLAB commands [L, U] = lu(V); format short e, U, yield:

- $U =$

$$\begin{pmatrix} 7.7760e+3 & 1.2960e+3 & 2.1600e+2 & 3.6000e+1 & 6.0000e+0 & -9.3300e+3 \\ 0 & 1.0417e+2 & 3.8194e+1 & 1.0532e+1 & -3.9034e+3 & 3.7505e+3 \\ 0 & 0 & -3.5860e+2 & 3.7800e+0 & 1.5205e+3 & -1.1656e+3 \\ 0 & 0 & 0 & -1.3623e+3 & 3.2190e+3 & -1.8567e+3 \\ 0 & 0 & 0 & 0 & -1.8253e+3 & 1.8253e+3 \\ 0 & 0 & 0 & 0 & 0 & -1.4211e-14 \end{pmatrix}$$

Vandermonde Example Explained

The MATLAB commands [L, U] = lu(V); format short e, U, yield:

- $U =$

$$\begin{pmatrix} 7.7760e+3 & 1.2960e+3 & 2.1600e+2 & 3.6000e+1 & 6.0000e+0 & -9.3300e+3 \\ 0 & 1.0417e+2 & 3.8194e+1 & 1.0532e+1 & -3.9034e+3 & 3.7505e+3 \\ 0 & 0 & -3.5860e+2 & 3.7800e+0 & 1.5205e+3 & -1.1656e+3 \\ 0 & 0 & 0 & -1.3623e+3 & 3.2190e+3 & -1.8567e+3 \\ 0 & 0 & 0 & 0 & -1.8253e+3 & 1.8253e+3 \\ 0 & 0 & 0 & 0 & 0 & -1.4211e-14 \end{pmatrix}$$

- The command d = prod(diag(U)) gives $d = 10.2645$. Since V was an integer matrix, MATLAB rounds the value of the determinant to the nearest integer.

Pascal Matrices

- The MATLAB command pascal(n) generates an $n \times n$ matrix whose entries are binomial coefficients.
- For example the command pascal(5) generates the matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}$$

- The Pascal matrices are all symmetric positive definite and all have determinants equal to 1.
- The command r = rank(pascal(15)) results in a returned value of r =14.

The Hilbert Matrix

- The $n \times n$ Hilbert matrix and its inverse can be generated in MATLAB using the commands hilb(n) and invhilb(n).
- For example, the command invhilb(5) generates the integer matrix

$$\begin{pmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{pmatrix}$$

- Even though hilb(12) has an integer matrix inverse, the command rank(hilb(12)) gives an answer of 11.

Numerical Rank

- Both P = pascal(15) and H = Hilbert(12) are extremely ill-conditioned.
- In finite precision arithmetic these matrices are indistinguishable from rank deficient matrices.
- MATLAB does give the correct numerical rank for each matrix

The MATLAB backslash operator

If we set H = sym(hilb(12)) and b=sum(H)′ the exact solution to Hx = b is x = ones(12,1).

The commands:

c = double(b), y = double(invhilb(12)∗ c) and z = hilb(12)\c generate solutions

$$y = \begin{pmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0273 \\ 1.0391 \\ 0.4844 \\ 1.5000 \\ 0.9063 \\ 1.0859 \\ 1.0029 \end{pmatrix}, \qquad z = \begin{pmatrix} 1.0000 \\ 1.0000 \\ 1.0003 \\ 0.9956 \\ 1.0320 \\ 0.8593 \\ 1.3925 \\ 0.2891 \\ 1.8339 \\ 0.3891 \\ 1.2541 \\ 0.9542 \end{pmatrix}$$

Rank Deficiency warning

- If e = c - b, then

$$H^{-1}c = H^{-1}(b+e) = x + H^{-1}e$$

- The vector e has entries on the order of 10^{-15} and H^{-1} has entries on the order of 10^{15}.
- When the backslash operator was used, MATLAB gave the warning:

  Warning: Matrix is close to singular or badly scaled.
  Results may be inaccurate. RCOND = 2.458252e-017.

Wrong answers that do or don't matter?

Example: Wilkinson matrix

$$W = \begin{pmatrix} 1 & -1 & -1 & \cdots & -1 & -1 \\ 0 & 1 & -1 & \cdots & -1 & -1 \\ 0 & 0 & 1 & \cdots & -1 & -1 \\ \vdots & & & & & \\ 0 & 0 & 0 & \cdots & 1 & -1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

- When $n = 60$ and $c = 1.05$ compare solutions to

$$W\mathbf{x} = \mathbf{b} \quad \text{and} \quad cW\mathbf{x} = c\mathbf{b}$$

- When $n = 100$ compute the smallest singular value of W.

$W\mathbf{x} = \mathbf{b}$ and $cW\mathbf{x} = c\mathbf{b}$

Set e = ones(60,1), $b = W * e$, $M = 1.05 * W$, $d = 1.05 * b$
Set $x = W \backslash b$ and $y = M \backslash d$ and compare $x(1:5)$ and $y(1:5)$

$$x(1:5) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \qquad y(1:5) = \begin{pmatrix} -16.5601 \\ -7.7800 \\ -3.3900 \\ -1.1950 \\ -0.0975 \end{pmatrix}$$

Numerical rank is 59, but MATLAB gives no warning of rank deficiency!
MATLAB does not use condition estimator if original matrix is triangular.

Perturbed Wilkinson Matrix

When $n = 100$ we can make the matrix singular by changing its (100,1) entry to -2^{-98}

$$\begin{pmatrix} 1 & -1 & -1 & \cdots & -1 & -1 \\ 0 & 1 & -1 & \cdots & -1 & -1 \\ 0 & 0 & 1 & \cdots & -1 & -1 \\ \vdots & & & & & \\ 0 & 0 & 0 & \cdots & 1 & -1 \\ -\frac{1}{2^{98}} & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

$$
\begin{pmatrix}
1 & -1 & -1 & \cdots & -1 & -1 \\
0 & 1 & -1 & \cdots & -1 & -1 \\
0 & 0 & 1 & \cdots & -1 & -1 \\
\vdots & & & & & \\
0 & 0 & 0 & \cdots & 1 & -1 \\
-\frac{1}{2^{98}} & 0 & 0 & \cdots & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2^{98} \\ 2^{97} \\ 2^{96} \\ \vdots \\ 2^{0} \\ 1
\end{pmatrix}
=
\begin{pmatrix}
0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0
\end{pmatrix}
$$

The smallest singular value of W must satisfy

$$\sigma_{100} \leq \frac{1}{2^{98}} \approx 3.1554e - 030$$

However, MATLAB computes the singular value to be

$$s_{100} = 3.0981e - 018$$

One can obtain a much better estimate of σ_{100} by doing a single inverse iteration on $B = W^T W$.
Using the MATLAB commands:

```
z=ones(100,1);
x=B\(B'\z);
s=1/sqrt(max(abs(x)))
```

we obtain an approximate smallest singular value

$$s = 1.9323e - 030.$$

MATLAB may fail to detect rank deficiencies

Kahan matrices

$$K_n = \begin{pmatrix} 1 & -c & -c & \cdots & -c & -c \\ 0 & s & -sc & \cdots & -sc & -sc \\ 0 & 0 & s^2 & \cdots & -s^2c & -s^2c \\ \vdots & & & & & \\ 0 & 0 & 0 & \cdots & s^{n-2} & -s^{n-2}c \\ 0 & 0 & 0 & \cdots & 0 & s^{n-1} \end{pmatrix}$$

where $c^2 + s^2 = 1$ and $c \neq 0$.

These matrices are rank deficient for n large. In the case of $m \times n$ Kahan matrices ($m > n$), MATLAB does not detect the rank deficiencies when solving $K\mathbf{x} = \mathbf{b}$.

Nontriangular Example

- Generate a 50×45 Kahan matrix by setting
  K = gallery('kahan',[50,45],pi/4);
  Set A = orth(ones(50)+eye(50))*K1;
  z=ones(45,1); b = A*z; x = A\b
- The first 5 entries of the solution x are

$$x(1:5) = \begin{pmatrix} -3.3440 \\ -1.5447 \\ -0.4906 \\ 0.1268 \\ 0.4885 \end{pmatrix}$$

- *A* has rank 44, but MATLAB does not warn of rank deficiency
- MATLAB's column pivoting strategy fails to detect the rank deficiency

Conclusions

- In a NLA class it is important for students to understand the difference between finite precision calculations and exact arithmetic.

Conclusions

- In a NLA class it is important for students to understand the difference between finite precision calculations and exact arithmetic.
- One can illustrate these differences by looking at examples where the software gives "wrong" answers.

Conclusions

- In a NLA class it is important for students to understand the difference between finite precision calculations and exact arithmetic.
- One can illustrate these differences by looking at examples where the software gives "wrong" answers.
- These examples provide students with good intuition into how to work with finite precision algorithms.

Conclusions

- In a NLA class it is important for students to understand the difference between finite precision calculations and exact arithmetic.
- One can illustrate these differences by looking at examples where the software gives "wrong" answers.
- These examples provide students with good intuition into how to work with finite precision algorithms.
- The examples motivate topics such as machine precision, conditioning, and numerical rank.

Conclusions

- In a NLA class it is important for students to understand the difference between finite precision calculations and exact arithmetic.
- One can illustrate these differences by looking at examples where the software gives "wrong" answers.
- These examples provide students with good intuition into how to work with finite precision algorithms.
- The examples motivate topics such as machine precision, conditioning, and numerical rank.
- The examples are interesting and fun.

Heisenberg Effect

- Walter Gander reports extremely unusual results for MATLAB programs developed by his students.
- Gander referred to these results as the Heisenberg effect. The term ‘Heisenberg effect’ is used to describe a system or event which cannot be observed or measured without changing the outcome the event.

Student program to compute smallest normalized floating point number

```
function x = myrealmin()
x = 1; temp = x;
while eps * temp / 2 > 0
        temp = (eps * temp / 2);
        if (temp > 0)
                x = temp;
        end
end
```

Student program to compute smallest normalized floating point number

- ```
  function x = myrealmin()
  x = 1; temp = x;
  while eps * temp / 2 > 0
          temp = (eps * temp / 2);
          if (temp > 0)
                  x = temp;
          end
  end
  ```
- The program produced a value of 0 for x!
  ```

Student program to compute smallest normalized floating point number

- ```
  function x = myrealmin()
  x = 1; temp = x;
  while eps * temp / 2 > 0
          temp = (eps * temp / 2);
          if (temp > 0)
                  x = temp;
          end
  end
  ```
- The program produced a value of 0 for x!
- When semicolons are removed for debugging, the computed value of x becomes $8.0948e - 320$???
  ```

Solution from a second student

- The following commands were carried out interactively

```
a = 1;
while a * eps>0
        last = a; a = a/2.0;
end;
a = last
```

Solution from a second student

- The following commands were carried out interactively

```
a = 1;
while a * eps>0
        last = a; a = a/2.0;
end;
a = last
```

- This produced the correct answer a = 2.2251e−308

Solution from a second student

- The following commands were carried out interactively

```
a = 1;
while a * eps>0
        last = a; a = a/2.0;
end;
a = last
```

- This produced the correct answer a = 2.2251e−308
- However when these commands are made into a script file test.m, execution of the script file resulted in a different computed value

  a = 4.9407e−324 ???

All is explained by Walter Gander

- Modern processors use 80-bit registers for processing real numbers and store results as 64-bit numbers according to the IEEE standard for double precision floating point numbers.

All is explained by Walter Gander

- Modern processors use 80-bit registers for processing real numbers and store results as 64-bit numbers according to the IEEE standard for double precision floating point numbers.
- In MATLAB when the file is compiled, efficient Pentium code is generated and executed. This code keeps some variables in the 80-bit registers on the chip. These have a longer fraction and a bigger exponent.

All is explained by Walter Gander

- Modern processors use 80-bit registers for processing real numbers and store results as 64-bit numbers according to the IEEE standard for double precision floating point numbers.
- In MATLAB when the file is compiled, efficient Pentium code is generated and executed. This code keeps some variables in the 80-bit registers on the chip. These have a longer fraction and a bigger exponent.
- The "Heisenberg-effect" occurs because without intermediate printing the computation is done completely in the 80-bit registers and the final result becomes 0 due to underflow when it is stored as 64-bit floating point number.

All is explained by Walter Gander

- Modern processors use 80-bit registers for processing real numbers and store results as 64-bit numbers according to the IEEE standard for double precision floating point numbers.
- In MATLAB when the file is compiled, efficient Pentium code is generated and executed. This code keeps some variables in the 80-bit registers on the chip. These have a longer fraction and a bigger exponent.
- The "Heisenberg-effect" occurs because without intermediate printing the computation is done completely in the 80-bit registers and the final result becomes 0 due to underflow when it is stored as 64-bit floating point number.
- Removing the semicolon in the second case, thus allowing for printing of the intermediate results, clears the registers or at least the variable temp is stored in memory such that it becomes a 64-bit floating point number.