Number Representation: Workbook

# Introduction

The accuracy of computations is limited by the finite precision with which computers store numbers. This can result in counterintuitive results of both matrix and scalar operations. Most customers are not familiar with this and may report such results as bugs.

# Activity 1: Binary representation of numbers

**Purpose**: Demonstrate the behavior of the EPS function at the binary level.

**Instructions:**

* View the binary representation of the number 1 with the following command:

>> tstb.utils.ieee754bits(1)

Notice that this breaks down the bits into the sign bit, exponent, and mantissa for you.

* Now, view the binary representation of the number 1+eps:

>> tstb.utils.ieee754bits(1+eps)

What is the difference between this and the binary representation of 1?

* Repeat steps 1 and 2 with the number 2^53-1 in place of 1. Remember that you will have to replace 'eps' with 'eps(2^53-1)'.
* Execute the following lines of code:

>> tstb.utils.ieee754bits(2^53)

>> tstb.utils.ieee754bits(2^53+1)

Can you explain this in terms of eps(2^53)?

**Main Points**

* There isn't any way to represent numbers in between x and x+eps(x) for any exactly representable x.

**Notes:**

# Activity 2: Calculating the rank of a matrix using singular values

**Purpose:** Use the singular value decomposition of a matrix to determine its rank

**Instructions:**

MATLAB calculates the rank of a matrix by counting the number of non-zero singular values. However, calculating these singular values can be susceptible to round-off error.

* Execute the following code :

>> A = ones(3)

* Use the SVD function and count the number of non-zero singular values to determine the rank of A.
* Execute the following code :

>> A = gallery('pei',5,-5)

* Is the matrix A singular? (Hint: Consider what happens when you multiply A by an all-ones vector)
* What is the smallest singular value of A?

**Main Points**

* Even such basic calculations as determining whether or not a matrix is singular are susceptible to round-off error.
* MATLAB's RANK function uses a non-zero threshold, depending on the largest singular value and the size of the matrix for determining whether to consider a singular value to be 0.

**Notes:**

# Activity 3: Calculate the condition number of a matrix using singular values

**Purpose:** Demonstrate the relationship between the condition number of a matrix and its singular values.

**Instructions:**

The condition number of a matrix is a measure to determine how sensitive many linear algebraic operations are to small changes in the matrix. Since not all numbers are representable, this gives some indication of how much error we expect in our final answer given that our input matrix is only an approximation of our intended values.

* Execute the following code:

>> A = gallery('frank',10)

* Use the COND function to calculate the condition number of A.
* Use SVD to find the ratio between the largest and smallest singular values of A.
* Now, we will estimate the error incurred in calculating the inverse for a small perturbation of A. Run the following code:

>> inv(A)-inv(A+eps\*eye(size(A)))

Notice that even though the perturbations were on the order 1e-15, the errors in the computed inverse were on the order 1e-10. In general, the larger the condition number, the larger this discrepancy.

**Main Points**

* Operations on matrices with high condition number are extremely susceptible to round-off error.

**Notes:**

# Activity 4: Improving the condition number of a matrix

**Purpose:** Demonstrate how the condition number of some matrices can be improved by scaling the data.

**Instructions:**

In this activity, we will fit a 7th order polynomial to some data using a Vandermonde matrix (check later on what we do in the Linear Algebra chapter). Since we will have 8 data points, the polynomial should pass through each point exactly.

* Execute the following code:

>> x = 1000:.01:1000.07;

>> y = [1 4 3 4 1 2 1 4];

>> plot(x,y,':o')

* Form the Vandermonde matrix whose columns are 1, x, x2, …, x7. (You may find the VANDER function useful, but be aware of the order of the powers)
* What is the condition number of the Vandermonde matrix?
* Fit the data using the backslash operator.
* Verify the fit by plotting it along with the original data. Make sure that you sample the fit at enough points to get a smooth plot.
* Execute the following code to center and scale the data:

>> z = (x - mean(x))/std(x);

* Form the new Vandermonde matrix whose columns are 1, z, z2, …, z7.
* What is the condition number of the new Vandermonde matrix?
* Fit the data using the new Vandermonde matrix.
* Verify the fit by plotting it along with the original data. Again, make sure that you sample the fit at enough points to get a smooth plot.

**Main Points:**

* Well-spaced data centered about 0 has better numerical properties than data with very narrow spread far from 0.
* This process is exactly the algorithm used by POLYFIT. If customers complain about POLYFIT giving bad fits, centering and rescaling the data may help.

**Notes:**

# Activity 5: The dangers of INV

**Purpose:** Demonstrate some of the reasons we discourage the use of INV.

**Instructions:**

We encourage customers NOT to use INV. There is virtually always a better alternative. This exercise will demonstrate some reasons why we discourage INV.

* Execute the following code:

>> A = [0.780 0.563; 0.913 0.659]

>> b = [0.217; 0.254]

* Observe that the exact solution to the system Ax=b is [1;-1].
* Solve Ax=b using backslash. Make a note of the difference between the exact and calculated values of x.
* Solve Ax=b using INV. Make a note of the difference between the exact and calculated values of x.
* How much larger is the error using INV versus backslash? Use the absolute error.
* Now, execute:

>> A = gallery('lehmer',1000);

>> b = ones(1000,1);

* Use TIC and TOC to compare the running times of A\b and INV(A)\*b. Make sure to run each command several times, to get a good measure of performance.

**Main Points**

* Even for simple 2x2 systems, INV can produce larger errors than backslash. For larger systems, the difference can be even more pronounced.
* INV is much slower than backslash.

**Notes:**

# Activity 6: Debugging numerical issues

**Purpose:** Identify numerical issues as the source of unexpected behavior.

**Instructions:**

* Customer Issue:

*“I have a matrix,*

*I can tell just from looking at it that the determinant is 1. However, MATLAB says that the determinant is 0".*

* Reproduce what the customer says. Note that the determinant result may vary in different MALTAB releases.
* Find the solution and write an e-mail to the customer.

**Main Points:**

* When a customer reports what they believe to be a computational bug, often it is due to a limitation inherent to numerical representation.
* Taking the determinant of a matrix is a poor way to determine whether or not it is invertible.
* After you have updated the customer, take a look at <http://blogs.mathworks.com/cleve/2012/06/03/fibonacci-matrices/>

**Notes:**

# Activity 7: More debugging numerical issues

**Purpose:** Identify numerical issues as the source of unexpected behavior.

**Instructions:**

* Customer Issue:

*“The FLOOR function is broken. When I take the floor of 1, I get 0. See:*

>> 0.3/(0.1+0.1+0.1)

ans =

1.000000000000000

>> floor(ans)

ans =

0*"*

* Reproduce what the customer says.
* Find the solution and write an e-mail to the customer.

**Main Points:**

* When a customer reports what appears to be a bug in an integer function, it is almost always due to a floating-point numerical precision issue.
* Integer functions are, by their nature, discontinuous, so they are particularly susceptible to round-off error.

**Notes:**