Text reference page 104.

Roundoff Error and Partial Pivoting

Purpose
To explore the floating point number system, how floating point arithmetic leads to roundoff error, and how partial pivoting can be used to reduce roundoff error with linear systems.

MATLAB Functions
`swap, scale, replace, ==, format`

The numbers used in scientific computation on a computer are represented in floating point form. In MATLAB you can get the flavor of floating point arithmetic by typing

```
pi
```

which returns

```
3.1416
```

or

```
e = exp(1)
```

which returns

```
2.7183
```

If you would like to see the full display for these numbers, try

```
format long
pi
e
```

When you are ready to return to the shorter version, type

```
format short
```

Look at the floating point representations of these ordinary fractions:

```
1/3
1/7
```

Floating point number arithmetic can represent large numbers such as

```
pi*10 ^ 41
```

and small numbers such as

```
e*10 ^ -12
```

The precision of floating point numbers is determined by the number of places the computer can process. In the MATLAB decimal display, we can see 16 places. The underlying representation at the processor level is in binary, and at this level the precision is 53 bits. Look at

```
u = 2 ^ -53
```

The display is in decimal, but in binary there are 52 zeros to the right of the (binary) point followed by a 1. When this is added to 1.0, it takes 54 bits to represent the floating point number. Look at what happens when we add `1` and `u`:

```
1 + u
```

MATLAB has a variable, `eps`, for 2^{-52}. You can see this with the MATLAB function `r == s`, which tests for identity of floating point numbers, returning `1` when `r` and `s` are identical as floating point numbers and `0` when they are not identical as floating point numbers.

```
eps == 2 ^ -52
```

while

```
eps == u
```

```
eps/2 == u
```

Now look at

```
eps + 1 == 1
u + 1 == 1
```

This is the source of roundoff error.

Partial Pivoting

We are going to set up a simple linear system to show how roundoff error can lead to poor results.

```
A = [u, 1 ; 1, 1]
b = [1 ; 2]
```

The exact solution to this system is

$$x = \frac{1}{1-u} \quad \text{and} \quad y = \frac{1-2u}{1-u}.$$

This is the unique solution and it is very close to $x = 1$ and $y = 1$. Look at what happens when we form the augmented matrix and row reduce. We use the Laydata functions `swap` (swaps rows), `scale` (multiplies a row by a scalar) and `replace` (adds a multiple of one row to another), to solve this system.

```
C = [A,b]
m = -C(2,1)/C(1,1)
C = replace(C,2,m,1)
m = 1/C(2,2)
C = scale(C,2,m)
m = -C(1,2)/C(2,2)
C = replace(C,1,m,2)
y = C(2,3)/C(2,2)
x = C(1,3)/C(1,1)
```

The solution produced is $x = 0$ and $y = 1$, which is not even close to the correct solution. This is not MATLAB's fault; MATLAB will produce an excellent approximation to the exact solution, that is $x = 1$ and $y = 1$ using the backslash function

```
A\b
```

The problem is intrinisic to floating point arithmetic and begins with the first replacement operation which theoretically produces

$$\begin{bmatrix} u & 1 & 1 \\ 0 & 1-\frac{1}{u} & 2-\frac{1}{u} \end{bmatrix}$$

With roundoff $1 - \frac{1}{u}$ and $2 - \frac{1}{u}$ are nearly the same. Check this out

```
1 - 1/u
2 - 1/u
```

The next scaling operation effectively leaves this system:

$$\begin{bmatrix} u & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

And the last replacement produces

$$\begin{bmatrix} u & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

The heart of the problem is roundoff which makes $1 - \frac{1}{u}$ and $2 - \frac{1}{u}$ nearly the same. It can be avoided by performing the pivot on the $(2, 1)$ entry. We begin with a swap,

```
C = swap(C,1,2)
```

and repeat the operations. (You can complete this example in Exercise 8.) This time we get the system

$$\begin{bmatrix} 0 & 1-u & 1-2u \\ 1 & 1 & 2 \end{bmatrix}$$

Here $1-u$ and $1-2u$ are nearly the same. But this is harmless, as the system

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

has the acceptable solution. Choosing to pivot on the 1 instead of the u makes the difference. While both choices of pivot lead to a roundoff error, the choice of the larger value in the (2, 1) position means that the scalar `m` will be smaller and thus the arithmetic adjustments in the replacement will also be smaller.

The *partial pivoting strategy* is to *pivot on the largest entry (in absolute value) below the pivot entry and in the same column.* In our example, the two eligible entries in the first column are u and 1. The partial pivoting strategy chooses 1 since it is larger. MATLAB utilizes Gaussian Elimination with partial pivoting in the `A\b` function, which, as we have seen provides an accurate answer to the example system.

MATLAB Exercises

1. Let

```
x = pi*10^32
y = sqrt(2)*10^-101
```

Show that `x + y` and `x` are identical floating point numbers. Then progressively raise the exponent on `y` until you find a value for which `x + y` and `x` are different. Looking at the exponents on `x` and `y`, explain what happened here.

2. MATLAB can represent complex numbers easily. Look at `sqrt(-1)`. THen use the quadratic formula to find the roots of $x^2 + x + 1$ with MATLAB.

3. MATLAB has some important conventions for displaying matrices. Try the following:

```
A = rand(4)*10
1000*A
```

Do you get the same thing? The entries of `A` and `A*1000` appear to be the same (in base 10), but the exponents are different. Where does the display show this difference in the exponents? Now try the following:

```
v = [1 2^-53]
```

Is `v(1)` the same as 1? Is `v(2)` the same as 0?

4. Roundoff error will occur with very simple computations. Try the following:

```
1/5 == .2
1/5 + 1/5 == .4
1/5 + 1/5 + 1/5 == .6
1/5 + 1/5 + 1/5 + 1/5 == .8
```

When does MATLAB start to indicate that these expressions are not equal? As a general rule, you should not rely on the function `==`. As we see here, small differences can emerge between values that are mathematically the same. A better check is to test for proximity with

```
abs(1/5 + 1/5 + 1/5 - .6) < 2
```

5. Look at the following:

```
2 ^ 1023
```

and

```
2 ^ 1024
```

This is *overflow.* Try the following to see if `Inf` behaves in a reasonable way:

```
Inf*2

Inf  ^ 2

1/Inf

Inf + 2

2 - Inf

Inf*Inf

Inf + Inf
```

6. Try these:

```
Inf - Inf

Inf / Inf
```

The indeterminate forms `Inf - Inf` and `Inf / Inf` are `NaN`, read as "not a number." Describe what happens with the following:

```
NaN*2

NaN  ^ 2

1/NaN

NaN + 2
```

7. *Underflow* occurs when a computation leads to an exponent smaller than the minimum exponent allowed by the floating point arithmetic. Compare

```
2 ^ -1100
```

with

```
2 ^ -1000
```

Find the largest exponent e for which 2^e causes underflow.

8. Use the MATLAB commands `swap`, `scale`, and `replace` to find the solution to the example system used in the discussion.

9. Use the MATLAB commands `swap`, `scale`, and `replace` and partial pivoting to find the solution to the system `A = magic(3); b = [1;2;3]`. The partial pivoting strategy will require a row switch.

10. Build the matrix

```
u = 2 ^ -52;
A = [1, 1/u ; 1, 1]; b = [1 ; 2];
```

and then use MATLAB's function

```
A\b.
```

You will notice that MATLAB gives the message

```
Warning:  Matrix is close to singular or badly scaled.
Results may be inaccurate.  RCOND = 1.110223e-016.
```

Some matrices are intrinsically dangerous for computational uses. `RCOND` is a value MATLAB uses to measure the level of danger. A matrix with a low `RCOND` value is intrinsically bad for computations. A system that does not have a unique solution will have a low `RCOND` value. Try the following:

```
A = magic(4); b = [1 ; 2 ; 3 ; 4];
```

The resulting message is a real danger sign, and the warning

```
Results may be inaccurate.
```

should be taken seriously.