Numerical Methods ITGS219

Lecture 3

By: Zahra Elashaal

Errors

- **Numerical Errors**

  Occurs as the numerical methods are usually not exact (that is, they are approximate methods).

  Ex: the notation $1/3 \approx 0.3333$. Obviously the more three's we retain the more accurate the answer.

  Errors can be expressed as two basic types:

  Absolute error: This is the difference between the exact answer and the numerical answer.

  Relative error: This is the absolute error divided by the size of the answer (either the numerical one or the exact one), which is often converted to a percentage.

  - Let $\hat{x}$ be some approximation of x. So that:

$$\text{absolute error} = |x - \hat{x}|, \qquad \text{relative error} = \frac{|x - \hat{x}|}{|x|}, x \neq 0$$

- **User Error**
  - The elimination of user error is critical in achieving accurate results.
  - In practice user error is usually more critical than numerical errors, and after some practice their identification and elimination becomes easier.
  - User error is avoidable, whereas many numerical errors are intrinsic to the numerical techniques or the computational machinery being employed in the calculation.

Numerical Errors

Example 2.10 Suppose an error of £1 is made in a financial calculation of interest on £5 and on £1,000,000. In each case the **absolute error** is £1 ($|x - \hat{x}| = |5-4 = 1$ and |1,000,000 – 999,999|=1),

whereas the **relative errors** are 20% and 0.0001% respectively. ($|x - \hat{x}|/|x| =$ 1/5*100=20% and 1/1000000 * 100 = 0.0001%

Example 2.11 Suppose a **relative error** of 20% is made in the above interest calculations on £5 and £1,000,000. The corresponding **absolute errors** are £1 and £200,000.

$$\text{relative error} = \frac{|x - \hat{x}|}{|x|}$$

Example 2.12 Estimate the error associated with taking 1.6 to be a **root** of the equation

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x^2 - x - 1 = 0.$$

- The exact values for the roots are (1±√5)/2 (let us take the positive root). As such the absolute error is:

$$\left|\frac{1+\sqrt{5}}{2} - 1.6\right| \approx 0.01803398874989$$

- and the relative error is the absolute error divided by the value 1.6 (or alternatively the exact root) which is approximately equal to 0.01127124296868 or 1.127%.
- We could also substitute x = 1.6 into the equation to see how wrong it is: $1.6^2 - 1.6 - 1 = -0.04$. Although it is difficult to understand how this can be used, it is often the only option (particularly if the exact answer cannot be found).

Numerical Errors

Example 2.13 Determine a value of x such that $f(x) = x^2 + 4x = 40.$

We start by guessing that x = 6 is the root we require:

- *x = 6,* *f(6) = 60 > 40 which is too big, try x = 5.*
- *x = 5,* *f(5) = 45 > 40 which is still too big, try x = 4.*
- *x = 4,* *f(4) = 32 < 40 now this is too small, so we shall try x = 4.5.*
- *x = 4.5,* *f(4.5) = 38.25 < 40 a bit too small, try x = 4.75*
- *x = 4.75,* *f(4.75) = 41.5625 > 40 a bit too big, back down again to x = 4.625*
- *x = 4.625,* *f(4.625) = 39.890625 < 40 a bit too small, back up again to x = (4.625 + 4.75)/2*
- *x = 4.6875, f(4.6875) = 40.72265625 > 40 and we can continue this process.*

Here we have just moved around to try to find the value of x such that f(x) = 40, *but we could have done this in a* systematic manner *(actually using the* ***size of the errors****).*

Errors And *eps* variable

The MATLAB variable ***eps*** is defined as the smallest positive number such that 1+eps is different from 1. ***eps***, with no arguments, is the distance from 1.0 to the next larger double precision number, that is eps with no arguments returns 2^(-52) = 2.2204e-16.

```
x1 = 1+eps;
y1 = x1-1
x2 = 1+eps/2;
y2 = (x2-1)*2
```

Consider the calculations:

- In both cases using simple algebra you would expect to get the same answer, namely *eps*;
- but in fact y2 = zero. This is because MATLAB cannot distinguish between 1 and 1+eps/2.
- The quantity *eps* is very useful, especially when it comes to testing routines.

Example 2.14 Calculate the absolute errors associated with the following calculations:

```
sin(15*pi)
(sqrt(2))^2
0.001*1000
1e10*1e-10
```

To calculate the absolute errors we need to know the exact answers which are 0, 2, 1 and 1 respectively. We can use the code:

```
abs(sin(15*pi)-0)
abs((sqrt(2))^2-2)
abs(1000*0.001-1)
abs(1e10*1e-10-1)
```

notice in the last case the exponent form of the number takes precedence in the calculation: we could of course make sure of this using brackets). The errors are 10^{-15}, 10^{-16} and zero (in the last two cases).

User Error :

- The elimination of user error is critical in achieving accurate results.
- In practice user error is usually more critical than numerical errors, and after some practice their identification and elimination becomes easier.

Once all the syntax errors have been eliminated within a code, the next level of errors are harder to find. These are usually due to:

1. An incorrect use of variable names.

This may be due to a typographical error which has resulted in changes during the coding.

2. An incorrect use of operators.

The most common instance of this error occurs with dot arithmetic.

3. Syntactical errors which still produce feasible MATLAB code.

For instance in order to evaluate the expression cos *x, we should use cos(x):*

4. Mathematical errors incorporated into the numerical scheme the code seeks to implement.

These usually occur where the requested calculation is viable but incorrect.

5. Logical errors in the algorithm.

This is where an error has occurred during the coding and we find we are simply working out what is a wrong answer.

- Practice is needed to Avoiding all of these errors. and usually after quite a lot of frustration.

User Errors … Cont :

This code purports to obtain three numbers *a*, *b* and *c* from a user and then produce the results

a + b + c, a/((b + c)(c + a)) and a/(bc).

Debag the next code

```
a = input(' Please entere a )
b = 1nput(' Please enter b )
a = Input ' Please enter c '
v1 = a+ B+d
v2 = a/((b+c)(c+a)
v3 = a/b*c
disp(' a + b + c= ' int2str(v1)])
disp(['v2 = ' num2str v2 ]
disp(['v3 = num2str(v4) ]);
```

The corrected code looks like:

```
clear all
a = input(' Please enter a ');
b = input(' Please enter b ');
c = input(' Please enter c ');
v1 = a+b+c;
v2 = a/((b+c)*(c+a));
v3 = a/(b*c);
disp([' a + b + c= ' num2str(v1)])
disp([' v2 = ' num2str(v2)])
disp([' v3 = ' num2str(v3)])
```

- *now work through the lines one-by-one detailing where the errors are and how they can be fixed.*

(**Tasks Home Work** *)*

User Errors … Cont :

Let $\hat{x}$ be some approximation of an exact value x. Which of the statements below is FALSE?

(A) The relative error is always smaller than the absolute error.

(B) The relative error can be smaller than the absolute error provided x is large enough.

(C) The relative error gives a better idea of the number of correct significant digits.

(D) None of the above.

Answer: (A). Because absolute error is $E_x = |x - \hat{x}|$ *and relative error is* $R_x = |x - \hat{x}|/|x| = E_x/|x|$*, the only time* $R_x < E_x$ *is when* $|x| > 1$.

Any Question?