5DV005, Fall 2018, Lab session 1

Carl Christian Kjelgaard Mikkelsen

November 11, 2018

Contents

1 The time and the place

Our first lab session will take place on

Wednesday, November 14th, 2018, (kl. 13.00-16.00), Room MA416-426.

2 Compilation of LaTeX documents

Documents written in LaTeXhave the extention `.tex`. They are translated into .pdf files using a multi-pass compiler such at `pdflatex`. The simplest case requires two pass compilation, i.e.,

```
>pdflatex lab2
>pdflatex lab2
```

If the document reads a .bib file to generate a list of references, then the compilation sequence is

```
>pdflatex lab2
>bibtex lab2
>pdflatex lab2
>pdflatex lab2
```

3 The problems

Problem 1

1. Copy the function `lab2/scripts/l2p1.m` into `/lab2/work/my_sum` and identify the differences between this function and `my_simple_sum`. In particular:

   (a) How is the correct value of the unit roundoff identified?

   (b) What is the effect of the reshape command?

   (c) How is user stupidity handled?

2. Extend the function `my_sum` to include the a priori error bound given by [**?**], Theorem 4.3. The appropriate call sequence is

   ```
   [y, aeb, reb]=my_sum(a)
   ```

   where

   - `y` is the computed sum
   - `aeb` is the a priori error bound
   - `reb` is the running error bound

   (a) Remember to update the documentation.

   (b) Remember to include comments related to the calculation of the error bounds.

   (c) Remember to update the revision history.

   (d) Remember to update the list of programmers.

3. Copy `scripts/l2p1_mwe1.m` into `work/my_sum_mwe1.m` and complete the script so that it computes the two sums

$$a = \sum_{j=1}^{n} \frac{1}{j^4}, \quad b = \sum_{j=1}^{n} \frac{(-1)^j}{j}, \quad n = 2^{20}, \tag{1}$$

   using both single and double precision as well as the error, an a priori error bound and a running error bound. You can obtain a *very* accurate value of the true sums using Kahan's method `kahan_sum` and double precision arithmetic.

4. Verify that the errors are bounded by the a priori error bounds.

5. Verify that the errors are bounded by the running error bounds.

6. Which technique gives the best error bounds?

7. Why is it reasonable that the running error bounds are smaller than the a priori error bounds? **Hint:** Which technique draws on the largest body of relevant information?

Problem 2

1. Copy `/lab2/scripts/l2p2.m` into `/lab2/work/my_recursive_sum.m` and complete the function according to its specification.

2. Extend `my_recursive_sum` to return a running error bound.

3. Develop a minimal working example `my_recursive_sum_mwe1` which computes the same sums as the previous problem using recursion threshold $m = 2^{10}$.

4. Verify that that the absolute value of each error is less than corresponding running error bound.

5. What are the consequences of reducing the recursion threshold m from $m = 2^{10}$ to $m = 1$?

   (a) What happens to the error?

   (b) What happens to the run-time? Use `tic`, `toc` to measure the run-time.

Problem 3 You will find the command `close all` very useful for this problem.

1. Copy `lab2/scripts/l2p3.m` in the function `lab2/work/my_horner.m` and complete the function according to the specification.

2. Develop a minimal working example `my_horner_mwe1` which plots the two polynomials

   ```
   p = @(x)(x-2)^3
   q = @(x)x.^3 - 6 x.^2 + 12 x - 8
   ```

   for $x \in 2 + [2^{-15}, 2^{15}]$ using 101 equidistant points. Let MATLAB evaluate p as stated, but use `my_horner` to evaluate q. Be careful and specify the coefficients in the right order!

3. Prove mathematically that $p(x) = q(x)$ for all $x \in \mathbb{R}$, i.e., the two polynomials are in fact identical, although the computed values are quite different.

4. Extend `my_horner` to include the calculation of the a priori error bound given by [**?**], Theorem 4.10. The appropriate call sequence is

   ```
   [y, aeb] = my_horner(a,x)
   ```

5. Extend `my_horner` further to include the calculation of a running error bound. The appropriate call sequence is

   ```
   [y, aeb, reb] = my_horner(a,x)
   ```

6. Extend the minimal working example `my_horner_mwe1` to create a second plot which plots the absolute value of q, the a priori error bound and the running error bound all in a single figure.

7. Use the running error bound to identify the region where the sign of the computed values of q can be trusted. A command such as `find(array1>array2)` will prove useful.

Problem 4 In this problem you will apply the bisection algorithm to compute a firing solution for a target located at a distance d from the gun.

1. Read through the documentation of the function `bisection` and execute the minimal working example to verify that the code makes sense.

In order to automatically hit a target at coordinates $(d, 0)$ we have to set up a function which can be passed to the bisection algorithm along with pertinant information. The following steps takes you through this process. Rename the script `lab2/script/l2p4.m` to `lab2/work/l2s4.m` and completed it as suggested.

2. Use the minimal working example of `range_rkx.m` to initialize a gun and define a range function using those values, i.e.

   ```
   range=@(theta)range_rkx(param,v0,theta,method,dt,maxit)
   ```

3. Similarly, define a function which measures the distance from the point of impact to the target, i.e

   ```
   res=@(theta)range(theta)-d
   ```

Here `res` is an abbreviation of the word "residual", i.e. "that which remains". Hitting the target precisely is equivalent to solving `res = 0`. In practice, we can settle for a residual which has a small absolute value, because the shell destroys anything with a certain radius ρ. This so-called "kill-radius" depends among other things on the amount and type of explosive and the type of target.

In order to use the bisection method, we need to find a bracket, i.e. a pair of angles θ_1 and θ_2 such that the two residuals have different signs. In terms of physics, this corresponds to firing a pair of shells such that one passes over the target and the other falls short of the target. Suitable brackets can be found if a ballistic table has already been computed.

4. Compute a suitable ballistic table using the function `compute_range`.

5. Explain to your friends, why the command `find(table(2,:)>d)` can be used to rapidly determine a bracket!

6. Compute the elevation necessary to hit a xenomorph at $d = 12345$ using the bisection algorithm, see Figure 1. Xenomorphs are very sturdy, so we want a residual which is less than 1 meter. Moreover, we want both the high and the low angle. Use the function `range_rkx` to generate the trajectories and plot them in the same figure. You should end up with a figure which looks like Figure 2.

7. A UAV (Unmanned Aerial Vehicle, i.e. a drone) has spotted a group of xenomorphs hiding behind a large hill. Your friend has analyzed the telemetry and gives you the new range. Compute the high and the low trajectory. Explain why the high trajectory is the probably the only viable option.

Figure 1: A typical xenomorph worker/warrior

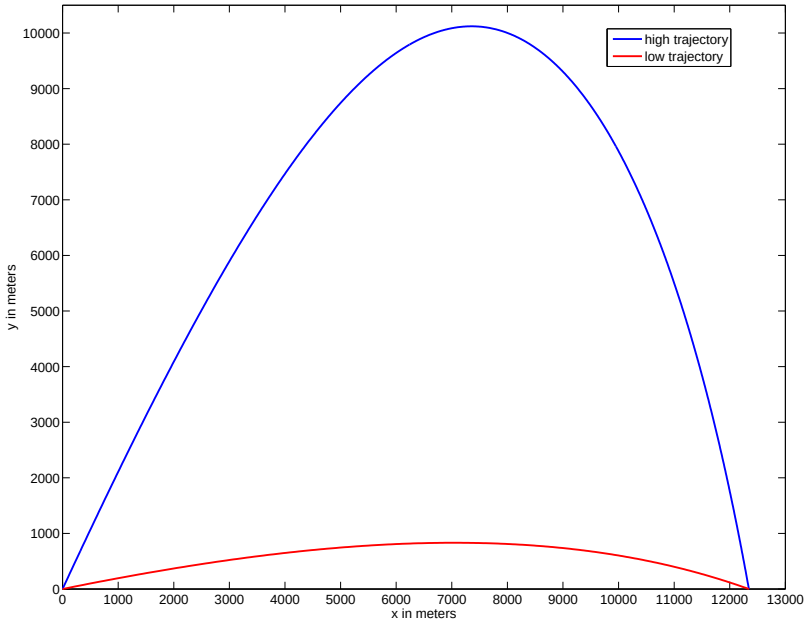

Figure 2: A plot of the high and the low trajectories for hitting a target at `d = 12345`.

4 Concluding remarks

1. Programs without proper description of the call sequence, input/output specification are useless to the user and the next programmer. Programs without meaningful comments are impossible to maintain. Good programmers are not afraid of writing their name on their products. Without contact information it is impossible to report bugs or ask for clarification.

   Programs which do not follow these principles will not be accepted.

2. Recursion is an elegant solution to many problems, but be wary of the overhead incurred by the function calls. Recursion is appropriate when the overhead is insignificant compared with the actual computation. This is not case if the computation consists of a single addition. Your recursive summation function uses a threshold `m` which sacrifies some accuracy for the sake of speed. It is possible to do tree-wise summation without recursion, but the code is a tad more complicated as demonstrated by `tree_sum`.

3. There is a lesson to be learned from question about the UAV. You should never blindly rely on the numbers generated by a computer. Always ask yourself what the numbers mean in the real world. In this case the low trajectory is likely to intersect the hill and it is useless for the practical purpose of destroying the xenomorphs. The hill is not part of the simulation, but it is part of the real world. The low trajectory is a valid firing solution, but only within the confines of the simulation.