PRA1004 Scientific Computing 2013

Frans Oliehoek
<frans.oliehoek@maastrichtuniversity.nl>

Lab Assignments Week 5

At the end of this weeks lab, you

- . . . have had some hands on experience with numerical integration.
- . . . know about simple programming constructs such as `if...else...` and `for` and `while` loops.
- . . . understand how *functions* allow you to reuse your code.
- . . . can compute the digits (up to machine precision) of $\sqrt{n}$ using your hand-coded algorithm.

1 Numerical Integration (1h)

The height of a population of people can be modeled as a bell-curve called *Gaussian, or normal distribution.* See Figure 1 for an illustration. In particular it expresses the 'probability density' of height h. The formula is

$$p(h) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h-\mu}{\sigma}\right)^2} \tag{1}$$

where

- h is the height for which we want to know the probability density.
- μ is the mean height (use 180).
- σ is the standard deviation (use 10).

Now, given this model, we can find the probability P that a randomly drawn person has height $h \in (190, 210)$ (or some other range) by taking the integral:

$$P = \int_{190}^{210} p(h) dh.$$

In this assignment, we will compute this probability numerically by using the `quad` functions in Matlab. In order to do this, we will need to define a user-defined function `HeightProb(h)` that will return $p(h)$ for each value of h.

- To do this, open up a new `.m` file and call it `HeightProb.m`. In it type:

```
function p = HeightProb(h)
%Helper function that returns the probability density at height h
mu = 180;
sigma = 10;
p = TODO;
return;
```

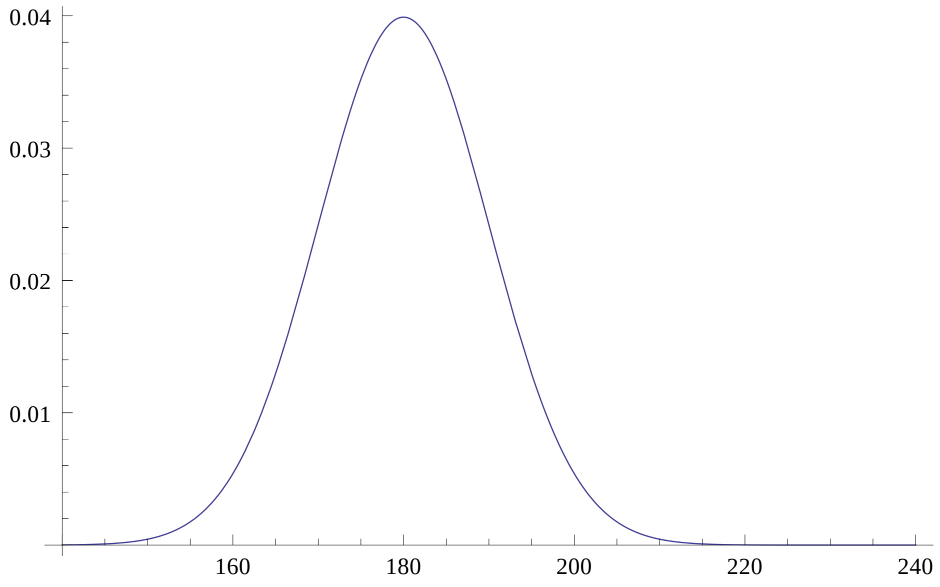

Figure 1: Used model of probability distribution of heights.

  - Note that it is important that the file name (`HeightProb.m`) and the function name (`HeightProb`) match!
  - Also note that it is important to realize that the input argument `h` should be allowed to be a vector (the function `quad` that we will use below will give vectors as arguments to this function). Therefore, make sure that you use element-wise operators to transform `h` into the corresponding densities `p`.

- Finish the implementation by replacing 'TODO' by the definition of (1).
- Verify your implementation by recreating the plot of Figure 1. To apply your own formula to some vector of heights, `heights`, you can use the command
  `probs=arrayfun(@HeightProb, heights)`
  next you can plot the `heights` versus the `probs`.

Now that we have defined the function, we can easily perform numerical integration using

- `quad('HeightProb', 190, 210)`
- What is the probability? Does is seem to be a reasonable estimate?

You should now that even when an integral seems very difficult, Mathematica is often able to find a solution for it. That is, when faced with an integral, Mathematica is probably the first thing you want to try out. Let us try to do this for this problem

- Check in Mathematica what the 'closed-form' solution is.
  Some pointers:

  - in Mathematica, all build in functions start with capital letters.
  - functions use square brackets.
  - use `p[h_] = (1/(sigma*Sqrt[2*Pi]))*Exp[...]` to define the function $p(h)$.
  - use `Integrate[p[h], h]` to find the indefinite integral.

- Also find in Mathematica the numerical approximation.

  1. use `Integrate[p[h], {h,190,210}]` to find the definite integral.
  2. use `N[...]` to find a numerical evaluation.

2 Loops (1h)

An important feature of computers is that they can repeat similar instructions over and over, for instance, to generate a table. This behavior is typically realized with *loops*. Here you are asked to practice with some loops. Remember that you can always look up the exact syntax with the `help` command (e.g., `help for`).

- print out the numbers 1–25 using a `for` loop. (use `disp` to display the numbers.)
- Do the same using a `while` loop.

Again, we will consider evolution of predator and a prey. Now, however, we can simulate this ourselves using a for loop.

- use the following dynamics:
```
predators(t+1)= 0.5*predators(t) + 0.25 * sqrt(predators(t)*prey(t));
prey(t+1) = 1.05 * prey(t) - 0.1 * sqrt(predators(t)*prey(t));
```
- write a for loop that loops over `t` with $1 \leq t \leq 50$.
- try different initial populations (e.g., 40 predators and 400 prey).
- plot the populations over time.
- try playing with the numbers a bit, is it easy to make one species go extinct?

3 Conditions (1h)

As you know, Matlab represents true using non-zero values, and false with 0. We already saw this when looking at what entries of a table have a value lower than some specific value (e.g., try `rand(4,4)<0.5`). The operator '<' is known as a *relational operator*. The following is the complete list of them:

relational operator	interpretation
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

Such 0s and 1s that decode truth values are called *Booleans* in computer lingo. An important feature when making the computer more advanced task is the ability to only execute certain statements when certain conditions hold. For instance, we may only want to continue updating the numbers of predators and/or preys if they are still positive (and otherwise set them to 0). This type of behavior is realized with `if` statements.

For instance try

- `if 3>2; disp('it is true:  3>2'); end`
- `if 3>5; disp('I thought 3 was smaller than 5'); end`

- `if(mod(44,11)==0); disp('indeed 44 is divisible by 11'); end`

Not that it is common practice to spread the if statement over lines (e.g., see `help if`), however, in simple cases it is okay to use the above formulation to put them on one line.

The following function defines a coin flip:

```
function heads = coinflip()
%coinflip is a function that performs a coinflip and returns 1 if the result is heads
heads = round(rand())
```

You can call the function is you save this piece of code to a file called `coinflip.m`.

- We can now show the display the outcome of the coin flips using:

  ```
  outcome_is_heads = coinflip()
  if(outcome_is_heads)
      disp('heads')
  if(not(outcome_is_heads))
      disp('tails')
  ```

Note that we used the command `not` to negate the truth value of the variable `outcome_is_heads`. For this same purpose, matlab also uses the tilde (~) symbol. For instance, try:

- `~1`
- `~0`
- `~42.42`

In many cases, it is easier to not explicitly specify the other condition, but just what needs to happen if a first condition does not hold. E.g., we could specify

```
you_win_at_the_casino = coinflip()
if(you_win_at_the_casino)
    disp('congratulations!')
else
    disp('Sorry, better luck next time.')
```

The `not` operator above is called a *logical operator.* There are three in total:

logical operator	interpretation
~	not
&	and
|	or

These can be used to create larger conditions from simple ones. E.g., `A & B` will test if both A and B are true, while `A | B` will test whether either A or B is true (or whether both are true! try: `1 | 1` to confirm this).

4 Loops and Conditions (30mins)

In this section we will try combining loops and conditional statements.

- print all the powers of 2 that are smaller than 100 using a `while` loop.
- loop through the numbers 1-50 and print the number if it is divisible by 3 or 7.
- print out all the prime numbers between 1 and 50. Tip: use the `factor` function to check if a number is prime.

For more information and practice, see chapter 5 in the book.

5 Newton's Method: using a While loop(1h)

In this section you will implement Newton's method to compute the digits of $\sqrt{2}$.

- Write down the function f for which $\sqrt{2}$ is a root.
- Write down the derivative f'.
- Now the Newton method will perform iterations, computing a new estimate of the root as follows:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

- Implement Newton's method to find the digits of $\sqrt{2}$. Use a while loop that continues iterations until the distance between the new and previous x is smaller than `eps(x)` (remember that `eps` gives the spacing around a point!)
- Use `format` to display all 16 significant digits.
- How many iterations does the algorithm need if you start with a crude approximation $x_0 = 4$?

6 Hand-in Assignment: Newton's Method for the Cliff Problem

Make a copy of your script that you made for section 5 and modify it such that it will solve a variant of our 'projectile' problem (illustrated in Figure 2):

> Anne is standing at the edge of a cliff of height h (m) and throws a rock with speed v (m/s) under an angle θ. We assume there is no friction, but that otherwise the experiment takes place on earth (so the gravitational acceleration $g = 9.8m/s^2$). How far does the rock travel horizontally before hitting the ground when $h = 14.5$m, $v = 30$m/s and $\theta = 34°$?

Hint: try first expressing both the horizontal and vertical position as a function of time, then find the time at which the rock hits the ground.

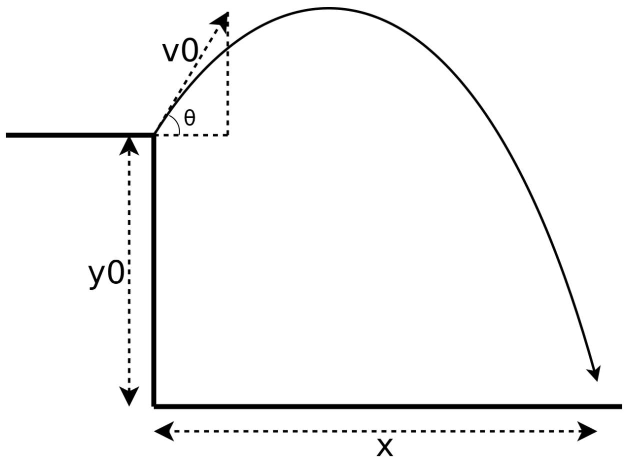

Figure 2: The cliff problem.