Ordinary Differential Equations

A differential equation is an equation in which the unknown is a function. This function appears in the equation in the form of its derivatives.

See an example of a simple differential equation:

$$ \frac{dy(t)}{dt}= 2 y(t) $$

This equation can be read as “what is the function y(t) whose derivative is equal to twice herself?”

Solving such equations can be very hard work[1], but if a supernatural inspiration whispers “the answer is $\exp(2x)$”, it's very easy to verify whether the solution is right.

If $y(t)=\exp(2x)$, the derivative of $y(t)$ is $2\exp(2x)$ (by the chain rule). This derivative, is, in fact, twice the function $y(t)$.

Normally, we write the ODE with the derivative of $y(t)$ on the left hand side. A simple case is an equation in which the right hand side does not depend on y, but only on t:

$$ \frac{dy(t)}{dt}= f(t) $$

We can solve this as:

- $ dy = f(t) dt $
- $ \int{dy} = \int{f(t) dt} $

The general solution for this is:

$$ y = \int{f(t) dt} $$

A more complicated case is the one in which the derivate of y depends on both y and t. We can write:

$$ \frac{dy(t)}{dt}= f(y, t) $$

We will get back to this case in the tutorial on numerical solutions.

A simple ODE on Maxima

Let's use Maxima to solve a simple ODE for us. Remember the ODE from last section, but we will change the constant 2 by a parameter r:

$$ \frac{dy(t)}{dt} = r y(t)$$

We can read this equation as: the instantaneous rate of change of our variable of interest is proportional to itself. That is, the higher the value of y, the higher the growth rate!

To solve this in the Maxima, use

```
'diff(y(t),t)=r*y(t);
ode2(%, y(t), t);
```

This equation looks familiar? We will return to it later on in the course: is the equation of exponential population growth model, the basic structure of many other models.

Make the graph of this function for $ r = 0.2 $ and initial state of 10!

```
  plot2d(10*exp(0.2*t),[t,0,20]);
```

Another simple function

Let's think on another case, in which the instantaneous growth rate is positive and approaches zero as our variable approaches one. In other words, the function increases a lot when it's small, but increases very slightly when it gets near one.

$$ f(x)=f(x)* (1-f(x)) $$

```
'diff(f(t),t)=f(t)*(1-f(t));
ode2(%,f(t),t);
```

You may not recognize this function, but apply the exponentiation on both sides that it will appear more friendly. It is the solution of the previous example multiplied by a term that acts as a brake tightening stronger as it arrives near one. This is the basis of logistic population growth models.

Numerical solutions

These first equations were easy to solve on . But do not get used to it, as this is not generally the case. A lot of equation do not have an algebraic solution[2] and need to be solved by “brute force”. This methods may require lots of mathematical operations, but a personal computer works wonders…

The base process is very simple, and looks like what we did to solve the derivatives. But there are lots of more robust methods as well.

Euler's Method

This is a very simple method, that consists on approximating the solution using the tangent lines in different points. Let's illustrate it with the function:

- $\frac{dN}{dt} = rN $ com r=2 e N(0) =20.

As we've seen:

- $\frac{dN}{dt} \approx \frac{N_{t + \Delta t} - N_t}{\Delta t}$

We can use any arbitrary time interval, such as 0.1. This gives:

- $\frac{N_{t + 0.1} - N_t}{0.1}\approx 2N$

In other words, if we have $N(0)=20$, on time 0.1, we will have approximately:

- $N_{t + 0.1} - N_t= 2N * 0.1$
- $N_{t + 0.1} = 2N_t*0.1 + N_t$
- $N_{t + 0.1} = 40 * 0.1 + 20 $
- $N(0.1) = 24 $

On time 0.2, we will have:

- $N_{0.1 + 0.1} - N_0.1 = 2N_0.1 * 0.1$
- $N_{0.1 + 0.1} = 2 * 24 *0.1 + 24$
- $N_{0.1 + 0.1} = 48 * 0.1 + 24 $
- $N(0.2) = 28.4 $

And so on! $\lim\limits_{x \to \infty} f(x)$ Notice that smaller time intervals lead to better precision. Remember than in the continuous functions, $\Delta t \rightarrow 0 $. We can repeat this for the next time intervals in :

```
f <- function (N, t)
{
        return (2*N)
}
# Initial time, N is equal 20:
N0 <- 20
# The time step is dt, and we will run the function up to tmax
dt <- 0.1
tmax <- 2

euler <- function (f, N0, dt, tmax)
{
        # res will be used to return a vector with all the results
        res <- NULL
        N <- N0
        for (time in seq(0, tmax, dt))
        {
                N <- N + f(N,time)*dt
                res <- rbind(res, N)
        }
        return (res)
}

# Examine the numeric solution
numerics <- euler(f, N0, dt, tmax)
numerics

x<- seq(0, tmax, dt)
```

```
# The exact solution
plot(x, 20*exp(2*x), typ='l', col='green')
# Let's compare it with the numeric solution
points(x, numerics, col='red', pch=4, ce=0.2)
```

Is this approximation good? Repeat the code with dt = 0.01 e 0.001 to compare.

Numerical Integration in R

We don't need to repeat all this process to do numerical integration on R, as there are available functions that are more efficient and robust than ours. Let's run the numerical integration for some functions using the package *deSolve* and the function *ode*. Before you start, you need to install and load the package. To install, you can use the menu or paste the following line:

```
install.packages("deSolve")
```

Load the package and take a look at the help page for function *ode*:

```
library(deSolve)
?ode
```

Very well! Now that we have the package loaded, let's run some numerical integrations.

A simple function

Let's try out a simple function:

$$ \frac{dy}{dt} = y-\frac{y^2}{K} $$

- 1. First, we need to create a function, with the following parameters:
  - time, which we will create as a numerical sequence
  - the initial state of the independent variable
  - the parameters of the ODE

```
fy1 <- function(time,y,parms)
{
 n=y[1]
 K=parms[1]
 dy.dt=n-(n^2/K)
 return(list(c(dy.dt)))
}
```

- 2. Now we set those parameters:

```
prmt = 10
y0 = 1
```

```
  st=seq(0.1,20,by=0.1)
```

- 3. Let's solve our differential equation and plot its graph:

```
  res.fy1= ode(y=y0,times=st, fy1,parms=prmt)
  plot(res.fy1[,1], res.fy1[,2], type="l", col="red",lwd=2, xlab="tempo",
ylab="y")
```

What kind of magic is behind this *ode* function? It just uses a method similar to Euler's, which we saw above, to find a numerical solution for an ODE.

Another simple function

Now, our equation will be:

$$\frac{dy}{dt}= y(ay^2 + by + r) $$

See the code below:

```
fy2 <- function(time,y,parms)
  {
   n=y[1]
   a=parms[1]
   b=parms[2]
   r=parms[3]
   dy.dt=n*(a*n^2 + b*n + r)
   return(list(c(dy.dt)))
  }
prmt = c(a=-1,b=4, r=-1)
y0 = 1
st=seq(0.1,20,by=0.1)
res.fy2= ode(y=y0,times=st, fy2,parms=prmt)
plot(res.fy2[,1], res.fy2[,2], type="l", col="red",lwd=2, xlab="tempo",
ylab="y")
```

Now it's your turn

- Change the starting condition to see what happens: 0.5; 0.3; 0.01

Now it's REALLY your turn

Run the numerical integration for the following functions:

- 1. $ \frac{dy}{dt} = y-y^2*f(t)$
  - with: $f(t)= 0.01 + 0.01 \sin(2\pi Mt)$;
  - M=15
- 2. $\frac{dn}{dt} = r(t)*n $

    - with: $r(t)= 0.1 - 100 sin(2 \pi t)$

maxima, equação diferencial, R

1) there are lots of courses about these equations on a maths major

2) and some have more than one solution