R Notebook: Intro to Solving ODEs in R

```
library(mosaic)
library(mosaicCalc)
library(deSolve)
library(ggplot2)
```

Introduction

A problem that we will often encounter is to obtain a numerical solution to an initial value problem (IVP) for an ordinary differential equation (ODE) or a system of ordinary differential equations (ODEs). An IVP for an ODE or ODEs typically takes the form

$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}(t), t)$,

$\mathbf{x}(t_0) = \mathbf{x}_0$.

Here $\mathbf{x}$ is the unknown function of time and $\mathbf{F}$ is the right hand side function. These may be scalars (in case of a sinlge ODE) or vectors (in case of a system of ODEs). In this intro we will focus on the case of solving a single scalar ODE, in the future we will examine systems on ODEs.

For example, we might want to obtain a numerical solution to

$\frac{dN}{dt} = rN$,

$N(0) = 10$,

with $r = 0.5$ and over the time interval $[0, 5]$.

In order to obtain a numerical solution to an IVP for ODEs we must provide at minimum the following information to the computer:

1) A function the describes the right hand side $\mathbf{F}(\mathbf{x}(t), t)$. Note that $\mathbf{F}$ may involve parameters whose values will also need to be specified.

2) A time interval, $[t_0, t_f]$, over which to obtain a numerical solution.

3) Initial conditions $\mathbf{x}_0$.

There are several packages developed for R that provide a means to obtain numerical solutions to an IVP for ODEs. The details of how you provide R with the necessary information 1)-3) depends on which particular package is used. We will proceed by looking at examples using a couple of different approaches.

Finally, once we have obtained numerical solutions to an IVP for ODEs it is typically desirable to plot these solutions in one way or another. We will also see examples of ways to plot our numerical solutions in R.

Numerical Solutions with integrateODE

The first method we emply to obtain numerical solutions for ODEs uses the integrateODE function from the mosaicCalc package. You must make sure that mosaicCalc is installed and loaded.

If you want to learn more than what is presented in this notebook see this YouTube video.

We will look at an example. Consider our problem to obtain a numerical solution to

$\frac{dN}{dt} = rN$,

$N(0) = 10$,

with $r = 0.5$ and over the time interval $[0, 5]$.

Here is the relevant R code:

```
solN <- integrateODE(dN~r*N,r=0.5,tdur=list(from=0,to=5),N=10)
```

Notice that the information input into the integrateODE function is

1) An expression for the equation.
2) Relevant parameter values.
3) A time interval.
4) An initial condition.

The results of calling the numerical solver accessed via integrateODE are stored in the variable we created called solN. Let's see how to access values of our numerical solution:

```
solN$N(0)
```

```
## [1] 10
```

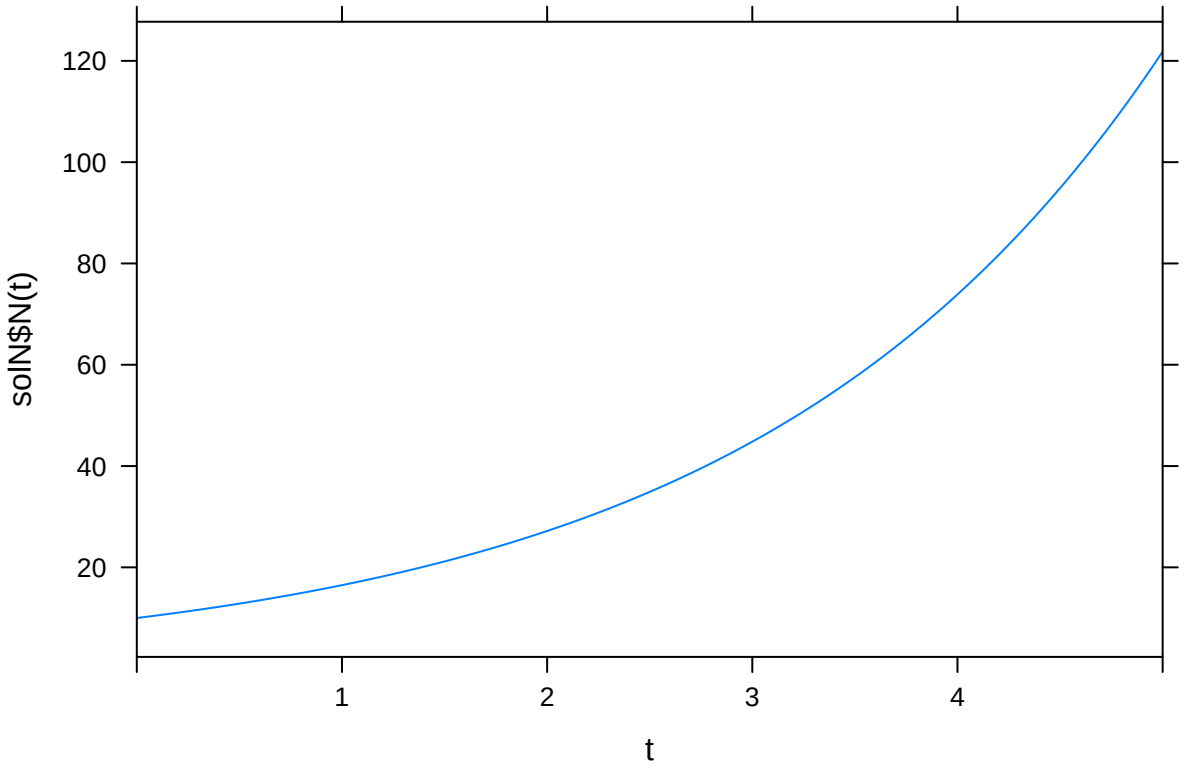
Note that this returns our initial condition $N(0) = N_0 = 10$. To get the solution at another time value, say for example $N(2)$ we simply type

```
solN$N(2)
```

```
## [1] 27.18282
```

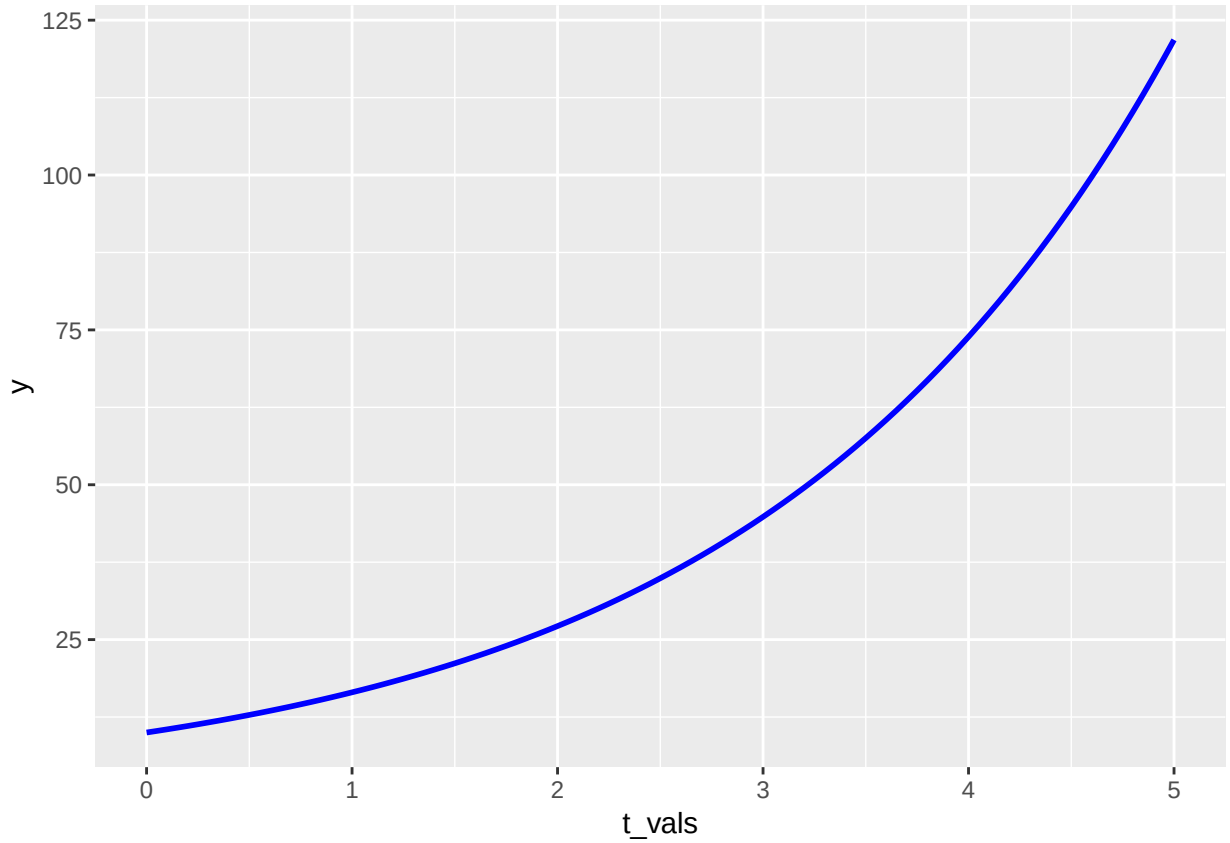
We can plot the solution $N(t)$ over the time interval as follows:

```
plotFun(solN$N(t)~t,t.lim=range(0,5))
```



Here is another approach to plotting our numerical solution:

```
t_vals <- seq(0,5,by=0.1)
ggplot(mapping = aes(x=t_vals)) +
  stat_function(fun=solN$N,color="blue",lwd=1)
```



Exercise: Modify what we have done so far to obtain and plot a numerical solution to the IVP

$\frac{dN}{dt} = rN,$

$N(0) = 5,$

with $r = 2.5$ and over the time interval $[0, 10]$.

In order to understand a little more about what is going on, recall that the analytic solution to

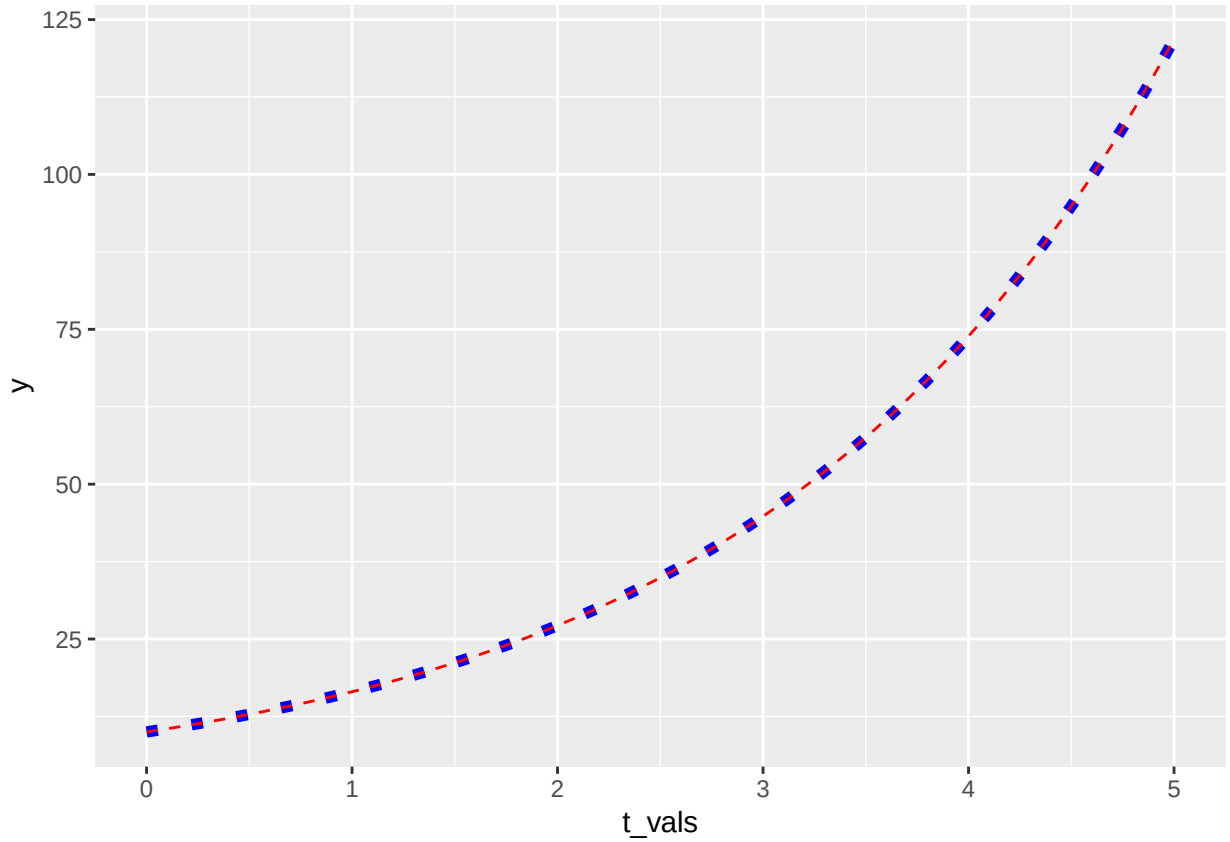
$\frac{dN}{dt} = rN,$

$N(0) = 10,$

with $r = 0.5$ is $N(t) = 10e^{0.5t}$. Let's plot this together with the numerical solution:

```
t_vals <- seq(0,5,by=0.1)

ggplot(mapping = aes(x=t_vals)) +
  stat_function(fun=solN$N,color="blue",linetype="dotted",lwd=2) +
  stat_function(fun=function(t) 10*exp(0.5*t),color="red",linetype="dashed")
```



The red dashed line is the analytical solution while the blue dotted line is the numerical solution.

Every approach to the numerical solution of differential equations in R has its pros and cons. Because of this, it is useful to know more than one way to solve the problem. (In general it is useful to know more than one way to solve almost any problem.) Next, we look at solving ODEs with the deSolve package.

Numerical Solutions with deSolve

To use the methods described in this section you must make sure that the deSolve package is installed and loaded. The main function for obtaining numerical solutions in the deSolve package is the ode function. In order to use it, we must write a function in R syntax that describes the right hand side of our differential equation. Let's do this for our example problem

$\frac{dN}{dt} = rN,$

$N(0) = 10,$

with $r = 0.5$ and over the time interval $[0, 5]$.

```
My_RHS <- function(t,state,parameters){
  with(as.list(c(state,parameters)),{
    dN <- r*N

    list(c(dN))
  })
}
```

In addition to describing the right hand side of the differential equation, we must also specify the times at

which to obtain the numerical solution and the initial conidition. This is done as follows:

```
state <- c(N=10)
times <- seq(0,5,by=0.1)
```

Now we are ready to obtain the numerical solution:

```
solN2 <- ode(y=state,times=times,func=My_RHS,parms=c(r=0.5))
```

Let's look at what has been returned by our call to the numerical solver, the results of which are stored in the variable solN2.

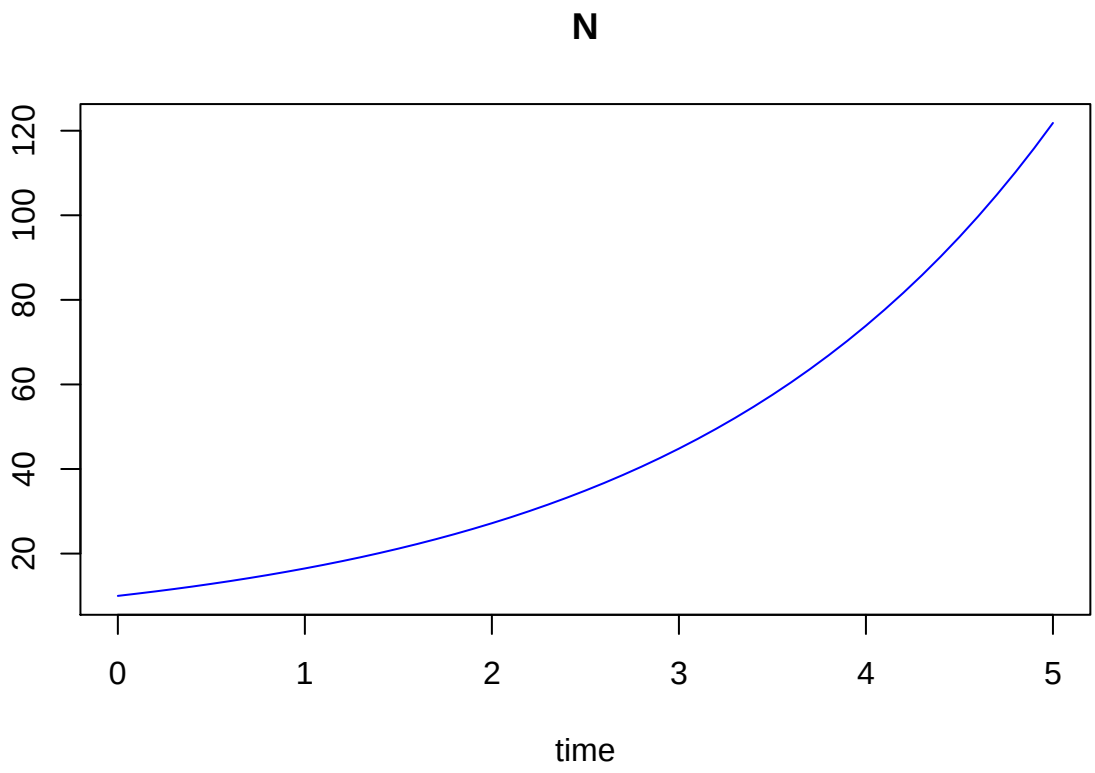
```
solN2
```

```
##    time         N
## 1   0.0  10.00000
## 2   0.1  10.51272
## 3   0.2  11.05171
## 4   0.3  11.61834
## 5   0.4  12.21403
## 6   0.5  12.84026
## 7   0.6  13.49859
## 8   0.7  14.19068
## 9   0.8  14.91825
## 10  0.9  15.68313
## 11  1.0  16.48722
## 12  1.1  17.33254
## 13  1.2  18.22120
## 14  1.3  19.15542
## 15  1.4  20.13754
## 16  1.5  21.17001
## 17  1.6  22.25542
## 18  1.7  23.39648
## 19  1.8  24.59604
## 20  1.9  25.85711
## 21  2.0  27.18283
## 22  2.1  28.57652
## 23  2.2  30.04167
## 24  2.3  31.58194
## 25  2.4  33.20118
## 26  2.5  34.90344
## 27  2.6  36.69298
## 28  2.7  38.57427
## 29  2.8  40.55202
## 30  2.9  42.63116
## 31  3.0  44.81691
## 32  3.1  47.11472
## 33  3.2  49.53034
## 34  3.3  52.06982
## 35  3.4  54.73950
## 36  3.5  57.54605
## 37  3.6  60.49650
## 38  3.7  63.59822
## 39  3.8  66.85897
## 40  3.9  70.28690
## 41  4.0  73.89059
```

```
## 42  4.1  77.67904
## 43  4.2  81.66173
## 44  4.3  85.84862
## 45  4.4  90.25017
## 46  4.5  94.87740
## 47  4.6  99.74186
## 48  4.7 104.85574
## 49  4.8 110.23181
## 50  4.9 115.88351
## 51  5.0 121.82499
```

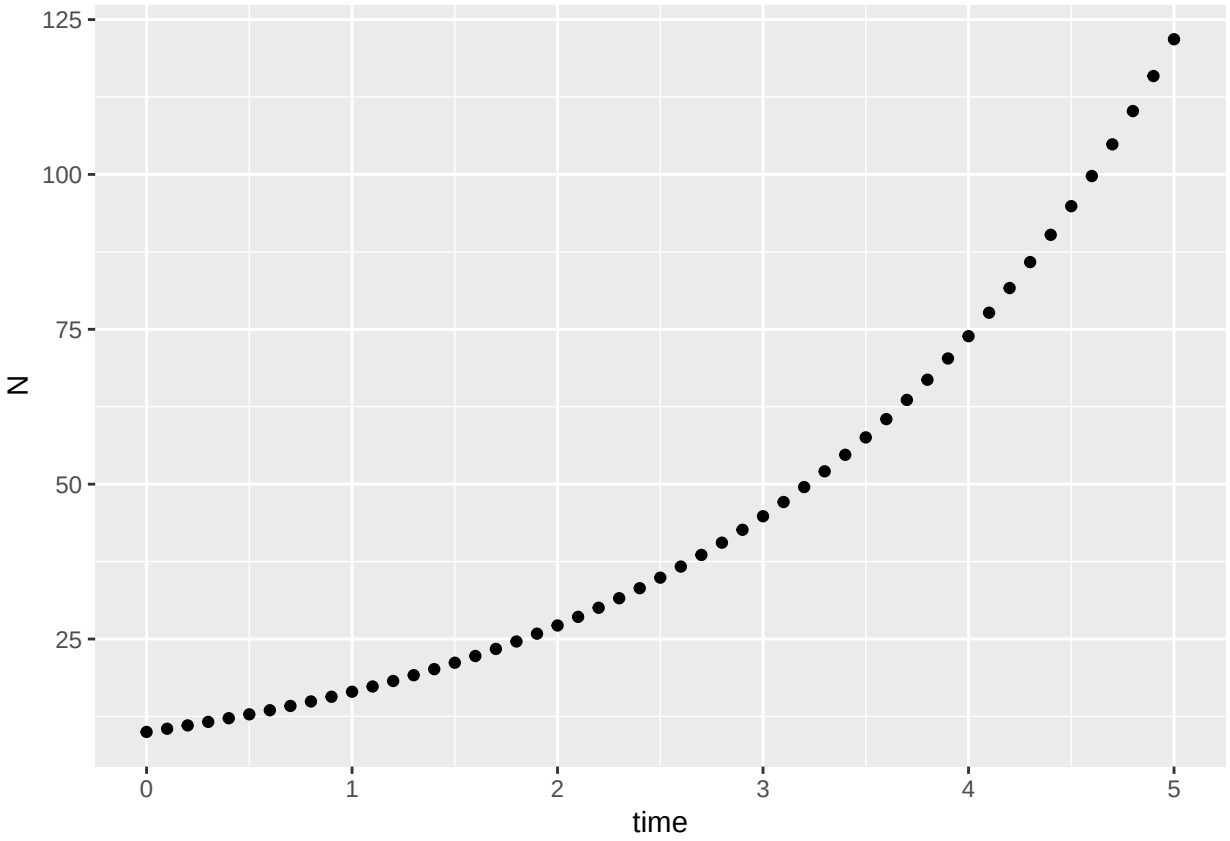
The method from the deSolve package returns a matrix where the first column contains the time values at which we obtained numerical solution values and another column that contains the solution values corresponding to each of the time points. We can easily plot the solution values versus the time points as follows:

```
plot(solN2,col="blue")
```



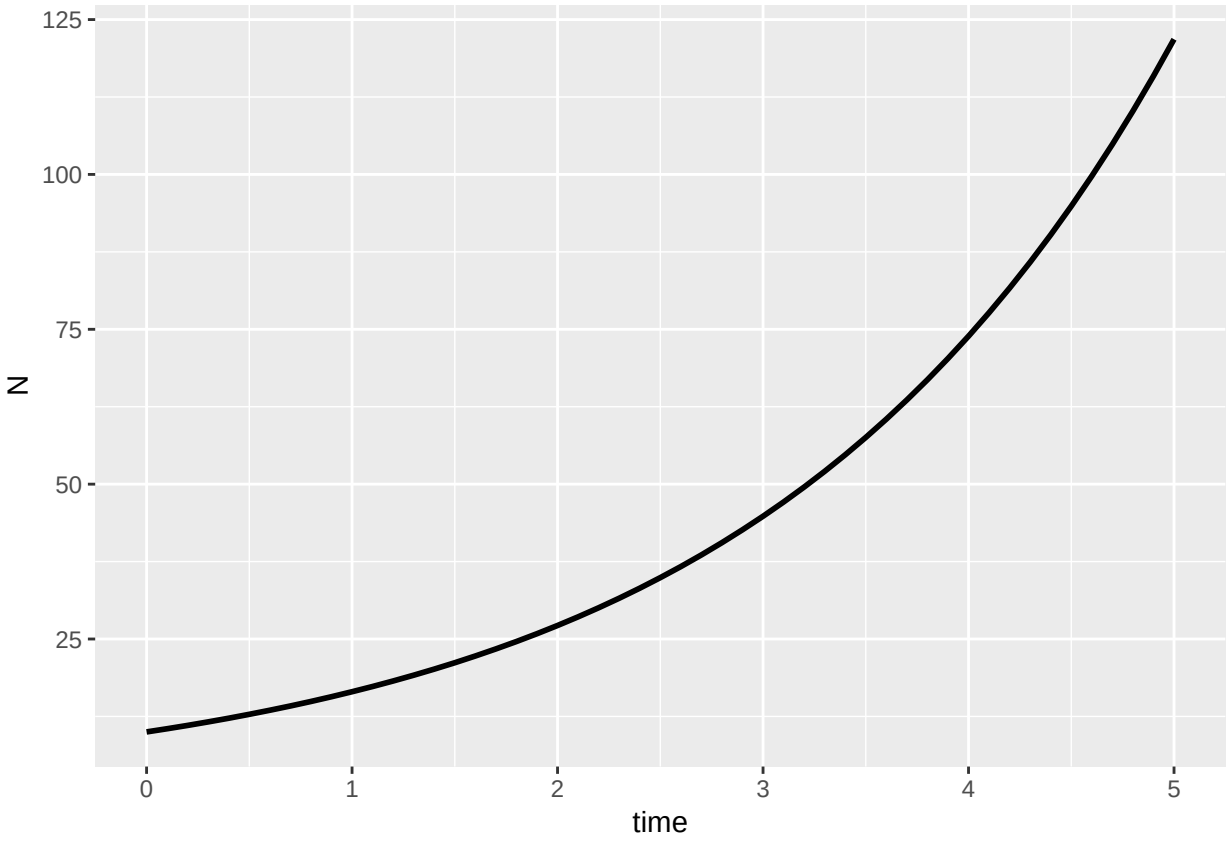
A slightly fancier plot can be obtained as follows:

```
ggplot(as.data.frame(solN2),aes(x=time,y=N)) + geom_point()
```



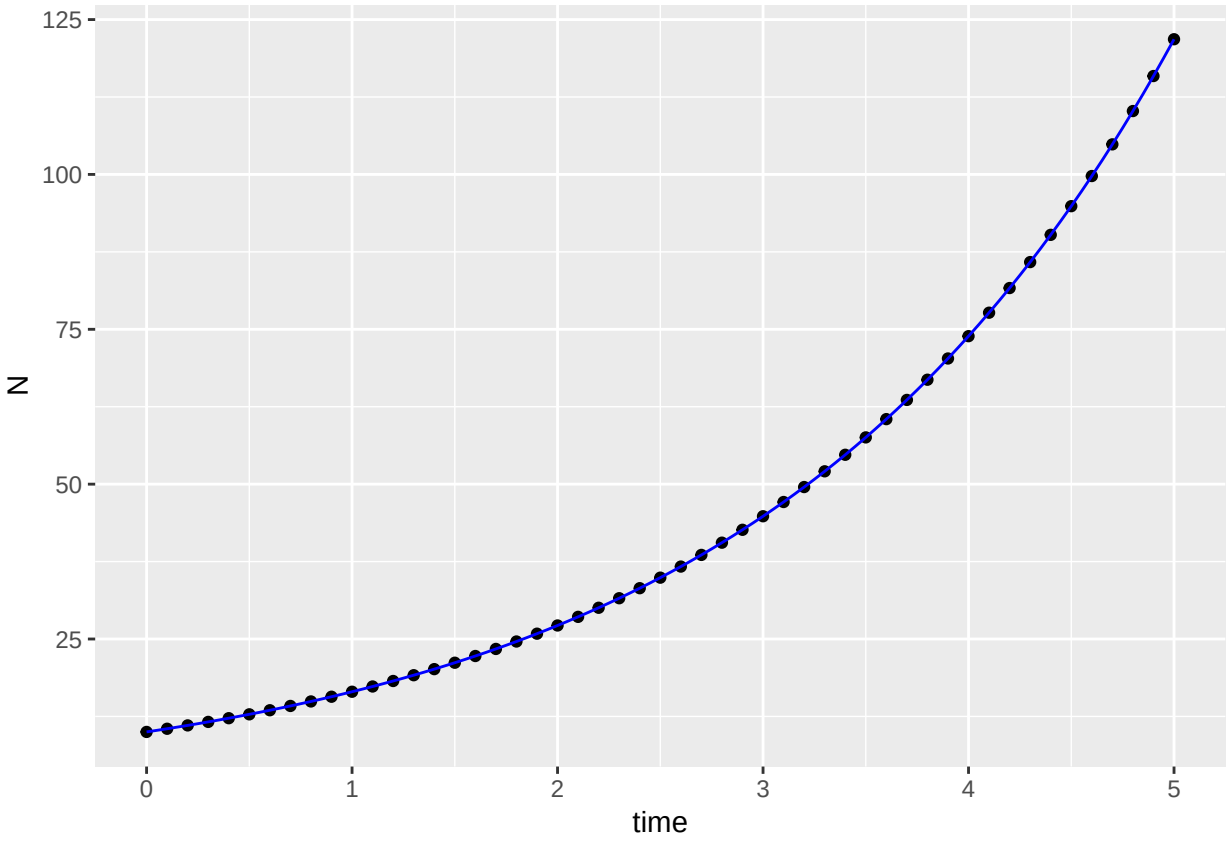
If instead we want a curve simply type:

```
ggplot(as.data.frame(solN2),aes(x=time,y=N)) + geom_line(lwd=1)
```



We can plot the numerical solution together with the analytical solution as follows:

```
ggplot(as.data.frame(solN2),aes(x=time,y=N)) + geom_point() +
  stat_function(fun=function(t) 10*exp(0.5*t),color="blue")
```



Exercise: Modify what we have done using the deSolve package to obtain and plot a numerical solution to the IVP

$\frac{dN}{dt} = rN,$

$N(0) = 5,$

with $r = 2.5$ and over the time interval $[0, 10]$.

Conclusion

We have now seen two different methods that can be used to obtain and plot a numerical solution for an initial value problem for a single scalar ordinary differential equation with R. An obvious question is why did we look at two different ways to accompish more or less the same thing. For one thing, basic results from the numerical analysis of differential equations tell us that there isn't really a single blackbox method that will perform well universally across all possible differential equations problems. Second, later in the course we will examine the problem of obtaining numerical solutions to systems of ODEs. The process is very similar to what we have done here with a single scalar equation. However, as systems of differential equations become larger and more complex you might find that it is simpy faster or more convenient to use one approach for numerical solutions over another.