Exponential Growth Simulator Utilizing the Ruby Language

Mark Ciotola

April 5, 2013

Abstract

The nature of exponential growth is discussed, and background and characteristics of the Ruby programming language are provided. An exponential growth simulator written in the Ruby programming language is presented, along with sample results.

1 Motivation

Exponential functions are ubiquitous in both the natural and social sciences. Population growth and radioactive decay are examples that can be expressed exponentially. In the social sciences, the rise of dynasties, the spread of religion and the formation of economic bubbles can be modeled with exponential functions.

Yet many participants in the social sciences, especially sociology and history, may not have much experience with programming. Hence, there is a need for exponential simulators that are easily programmed, understood and modified. The Ruby programming language provides such an opportunity. Although exponential models can be simulated using other structured programming languages, the Ruby language involves a much shallower learning curve, and hence allows such simulations to be accessible to a wide audience. Ruby also allows for rapid development and thus is an excellent brainstorming and prototyping tool.

2 Exponential Growth

Exponential growth is the easiest exponential function to understand and model. Exponential growth can be used to model growth a wide range of phenomena, ranging from populations of living organisms to the initial stages of economic bubbles. The key concept is that at any point of time, growth is proportional to current magnitude, which can be expressed by the following differential equation:

$$\frac{dY}{dt} = kY. \tag{1}$$

The solution to the that equation is the typical exponential growth equation shown below, where Y represents the the magnitude of what is being considered, e is the tran-

scendental number, t is time, k is a constant of proportionality and C is an initialization constant:

$$Y = e^{kt} + C \tag{2}$$

A classic example is that of the initial stages of population growth, such as for bacteria or rabbits, where the relevant differential equation is:

$$\frac{dP}{dt} = kP \tag{3}$$

where P is population and k is a constant of proportionality[2]. The solution to the equation is:

$$P(t) = e^{kt} + P_0 \tag{4}$$

where P_0 is the initial population.

3 Ruby Programming Language

3.1 History and Characteristics of Ruby

Computer scientist Yukihiro Matsumoto developed Ruby and first released it in 1995. Ruby draws from Perl, Smalltalk, Eiffel, Ada, and Lisp. [4] When creating Ruby, Matsumoto strived to develop "a scripting language that was more powerful than Perl, and more object-oriented than Python." [5] In fact, Ruby treats everything as an object. [4] Further, "Ruby is designed to be human-oriented. It reduces the burden of programming. It tries to push jobs back to machines. You can accomplish more tasks with less work, in smaller yet readable code." [5] The clean, plain-English code of the Ruby language makes learning its basics easy and intuitive. Beginners can start with a free, 20 minute online course at the Ruby Lang site. [4]. Since Ruby is free, open source software [4], it is easily affordable for new areas of social science that may not have a large funding base.

3.2 Sample Ruby Programs

A minimal hello world program comprises merely one line:

```
puts "Hello World"
```

Some non-programmers are intimidated by object structures. Fortunately, it is possible to develop simple simulations in Ruby without explicitly creating classes or even declaring variables. The following is a simple but complete linear simulation to model income as a function of time. The only non-intuitive aspect is the need to convert variables into strings for purposes of display (e.g. time.to_s). Code that appears after a # represents explanatory comments that do not add any functionality.

```
# Initialize Parameters
income = 0.0
time = 0

# Calculate and Display Results
while time < 5
  income = 10.0 * time
  time = time + 1
  puts "At time " + time.to_s + ", income is " + income.to_s + "."
end
```

The above program produces the following output.

```
At time 1, income is 0.0.
At time 2, income is 10.0.
At time 3, income is 20.0.
At time 4, income is 30.0.
At time 5, income is 40.0.
```

3.3 Ruby on Rails and Comparison of Ruby with Other Languages

Although Ruby functions well as a standalone language, it can also be utilised to develop web applications within the Rails framework.[1] Such web applications allows even wider access to simulations, although developing them does require greater technical skill.

Ruby is slower than traditional scientific languages such as C or FORTRAN, but spares the developer from numerous tasks such as memory management. Ruby also provides a well-supported object-oriented environment that FORTRAN lacks, despite some attempts to the contrary.[3]

4 Methodology and Exponential Growth Simulator Example

Methodologies

There are several types of exponential growth simulators. The simplest type is an analytic growth simulator using a simple equation. Such an approach provides mathematical and conceptual simplicity. It typically involves a solution to the typical population growth differential equation.[2] It has the disdavantage that it is an inflexible model that does not delve into the causes of exponential growth.

Another approach is to build exponential growth from the means of growth, such as reproducing pairs of people or reproducing engines. Such approaches allow for greater structuring and a deeper investigation than empircal data-fitting. However, such approaches are more complicated and thus less suitable for an initial introduction.

Here, an analytical approach will be used for purposes of simplicity and also to reflect classic population growth mathematics. The first simulator is extremely simplified, a literal exponential growth variant of the proverbial "Hello World" program. Paramaters are hard-coded and constants are either 0 or 1. There is minimal quantitative processing, but all output is on screen rather than sent to file.

The simulator begins with a simple exponential growth formula, with P representing current population (or production) and t representing time,

$$P(t) = e^{kt} + C. \tag{5}$$

Setting the parameters as $k = 1$ and $C = 0$, the equation becomes

$$P(t) = e^{t}. \tag{6}$$

4.1 Source Code

The following is an example of source code for the above *Exponential Growth Simulator*. Note the use of the *Math* object for the exponential function.

```
# Initialize parameters

  period = 1
  lineargrowth = 1.0
  exponentialgrowth = 1.0

# Run Simulation

  puts "\n\n"
  puts "PERIOD\t\tLINEAR GROWTH\t\tEXPONENTIAL GROWTH"
  puts "------\t\t-------------\t\t------------------ \n"

finalperiodlimit = 21

while period < finalperiodlimit do
  lineargrowth = 1.0 * period
  exponentialgrowth = Math.exp(period)

  # display variables with type conversions

  per = period.to_s
  lin = lineargrowth.to_s
  exp = exponentialgrowth.to_s
  periodstring = ("%2d\t\t%11.3f\t\t%1.3f")
```

```
  puts sprintf periodstring, per,lin,exp

  period = period + 1

end

puts "\n Done. \n"
```

4.2 Results

Sample results, comparing linear with exponential growth as a function of time, are shown below.

PERIOD	LINEAR GROWTH	EXPONENTIAL GROWTH
1	1.000	2.718
2	2.000	7.389
3	3.000	20.086
4	4.000	54.598
5	5.000	148.413
6	6.000	403.429
7	7.000	1096.633
8	8.000	2980.958
9	9.000	8103.084
10	10.000	22026.466
11	11.000	59874.142
12	12.000	162754.791
13	13.000	442413.392
14	14.000	1202604.284
15	15.000	3269017.372
16	16.000	8886110.521
17	17.000	24154952.754
18	18.000	65659969.137
19	19.000	178482300.963
20	20.000	485165195.410

5 Future Steps

Ruby offers a high level language for those inexperienced with programming to develop and change simulations with code that is easily read and understood.

It is possible to create simulators that obtain their parameters though user input or files. Likewise, it is possible to save both parameters and output to files. Much more

complicated simulators are possible.

Appendix A—Obtaining Ruby and Running the Exponential Growth Simulation

Command Line Utility

To use Ruby, the user needs access to their computer's command line. On Mac OSX, the Terminal application can be used. Console is available for free for Windows. Most Linux applications come with a terminal utility under various names.

Obtaining Ruby

To run the simulation, the user must also have Ruby installed on their machine (for the examples in this article, preferably Ruby 1.9.2 or higher). Some computers have Ruby pre-installed. To find out if your computer has Ruby and its version, on the command line enter:

```
ruby -v
```

If you see something similar to the following, then you already have Ruby installed.

```
ruby 1.9.2p290 (2011-07-09 revision 32553) [x86_64-darwin10.7.1]
```

Otherwise, you need to download it from the following source and install it: http://www.ruby-lang.org/en/

Running the Simulation

The file for this simulator can copy and pasted from this article into a text file, or it can be downloaded at: https://github.com/mciotola/e-growth-static

The user runs the simulator by entering "ruby [path] e-growth-static_1_3_1.rb" at their computer's command line, the latter part being the file's name. Then enter ¡Return¿ or ¡Enter¿ depending on the system. The simulator then shows the output on screen.

Appendix B—Object-Oriented Example of Ruby Code with Results

While not necessary for simple simulations, the object-oriented capabilities of Ruby can make the program more modular, allow better modeling of complicated entities, and help avoid the need to rewrite code.

In orbject-oriented programming, objects represent entities, such as animals or companies. Objects belong to classes and subclasses. Subclasses inherit the characteristics of classes they are derived from, but also add their own particular characteristics to themselves.

The following simple program partly derived from [4] demonstrates Ruby's object-oriented nature, and illustrates several object concepts.

Source Code

The source code for the sample object-oriented Ruby program is shown below. This program illustates the modular nature of object-oriented programs. It first creates a Regime class that applies to many different types of governments. Then the Dynasty subclass is derived from the Regime class, but adding additional descriptive information. A regime object *sampleregime* is created for the First French Republic, and then a dynasty object *sampledynasty* is created for the Romanov dynasty. The code snippet *sampledynasty.title* includes *sampledynasty* which is the name of the object and *title* which is the method being called.

```
# Create the Regime class
class Regime
  def initialize(name) # Capitalize name of Regime
    @name= name.capitalize
  end
  def title  # Create a method to output the regime name
        print @name
  end
  def description  # Add a description to the regime name
        print @name + " is the name of a regime."
        puts
  end
end

# Create the Dynasty class as a subclass of Regime
class Dynasty < Regime
  def description  # Add a description to the dynasty name
    print " is the name of a regime that is also a dynasty."
        puts
  end
end

# Create new Regime and Dynasty object.
sampleregime = Regime.new("First French Republic")
sampledynasty = Dynasty.new("Romanov")

# Output
puts
puts "OUTPUT FROM OBJECT DEMONSTRATION"
```

```
puts "_____"
sampleregime.description
sampledynasty.title  # title inherited from regime class
sampledynasty.description
puts  # puts by itself adds an extra line of space
```

Results

The output displays the title of each created object, along with an appropriate description for tht type of object.

OUTPUT FROM OBJECT DEMONSTRATION

```
First french republic is the name of a regime.
Romanov is the name of a regime that is also a dynasty.
```

References

[1] 37Signals, *Ruby on Rails*. http://rubyonrails.org/, last viewed on 22 March 2013.

[2] Blanchard, Paul, Robert L. Devaney and Glen R. Hall, 2002, *Differential Equations, second edition*. Brooks/Cole.

[3] Ruby-Doc.org, *Programming Ruby: The Pragmatic Programmer's Guide*. http://www.ruby-doc.org/docs/ProgrammingRuby/, last viewed on 22 March 2013.

[4] Ruby-Lang.org, *About*. http://www.ruby-lang.org/en/about/, last viewed on 22 March 2013.

[5] Bruce Steward, *An Interview with the Creator of Ruby*. Addison Wesley, Massachusetts, http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html, last viewed on 25 October 2012.