Applications of Homogeneous Markovian Processes in Economics and Latent Parameters Estimation for Discrete State Hidden Markov Models

Student: Adrian Vrabie adrian@vrabie.net

Academic Adviser: Dr. habil. Dmitrii Lozovanu, University Professor

lozovanu@math.md

May 27, 2016

Abstract

Markovian processes are already ubiquitous in a wide range of real world applications and serendipitously provide remarkable goodness of fit despite its simplistic mathematical model. The Hidden Markov Model is the most suitable class among Markovian processes for modelling applications in Finance and Economics, yet the difficulties in estimating its parameters still present an issue for widespread adoption by the industry and academia.

This thesis is dedicated to surveying the methods and algorithms for estimating the latent parameters of HMMs. The first part of this thesis commences with simple Markovian processes using a hypothetical example for an upstream gas and oil company. The second part elaborates on methods and algorithms of the EM class for estimating latent state transition probabilities of Hidden Markov Models. Once the transition probability matrix is determined we can use the dynamic programming and combinatorial methods proposed by Lozovanu and Pickl (2015) for determining the state-time probabilities and the matrix of limiting probabilities in solving real world questions.

Software implementations in Octave, Python and Julia are provided based on the open-source QuantEcon and GHMM packages.

Contents

List of Figures

Acknowledgements

First of all I would like to express my deepest appreciation to my academic adviser Prof Dr. Habil Dmitrii Lozovanu for his invaluable advise, inspiration and lectures on Markov Chains. He convincingly conveyed a passion for research which sparked my interest in pursuing Markovian processes.

Secondly I want to render my deepest gratitude to Prof Alexandra Tkacenko for her input and for her support. If not for her lectures I would have never fully understood the beauty of optimizations in HMM for parameter estimation.

I would also like to thank Prof Viorel Grigorcea and Prof Valeriu Gutu for administrative support. Their selfless time and care were sometimes all that kept me going. Many thanks to Drd MD Nadeja Negari for taking care of my health and the fact that she started sharing my interest in pursuing applications of Markov chains in genetics and restlessly supported me in every endeavour. To Mr Dorin Vremis for his inspiration in pursuing academic goals and for borrowing me a sense of humour when I lost mine.

I am much indebted to my colleagues from work for their care and encouragement in writing my thesis: Eduard Calancea, Galina Buga and Victoria Minciuc. Sometimes they were more worried than I was about my deadlines and outcomes.

I would like to thank in advance the Examining Committee for their patience and questions and the whole faculty of Mathematics and Computer Science from the State University of Moldova.

Many thanks to my colleagues for their input and encouragement and not least I would like to thank You, the reader, whoever you are.

Preface

The question that sparked my interest in studying Markovian processes is how to estimate the parameters of a HMM. In particular, given a set of observations, how to find the best estimates for the state transition stochastic matrix $\mathbb{A}$.

Markovian processes are ubiquitous in many real world applications, including algorithmic music composition, the Google search engine[1], asset pricing models, information processing, machine learning, computer malware detection [2] and many more.[3] Markov chains can be used to help model how plants grow, chemicals react, and atoms diffuse and applications are increasingly being found in such areas as engineering, computer science, economics, and education. Jeffrey Kuan at Harvard University claimed that Markov chains not only had a tremendous influence on the development of mathematics, but that Markov models might well be **the most *"real world"* useful mathematical concept after that of a derivative.**

As we will see, Markovian chains and Hidden Markov Models have a rich yet accessible mathematical texture and have become increasingly applicable in a wide range of applications. Although the assumptions underpinning Markovian processes can be perceived as unacceptably restrictive at first, Markov models tend to fit the data particularly well. To fast-forward the conclusions of this thesis, it all gets down to how well we define what *a state* is. Also, there are many types of Markovian processes each with its set of particular features. In this paper we deal with only one particular class: the models that exhibit time invariant probability distributions within a state.[4] These models allow the use of the theoretical results from studies focused on the convergence properties of stationary distribution as time $t \mapsto \infty$. This is of use in Economics because it opens avenues not only to reinterpret economic growth in the settings of a stochastic matrix but allows to efficiently compute expected long-term economic growth rates.

The extension of Markovian chains to HMMs allows modelling even a wider scope of applications, suitable not only to describing the behaviour of the economy at a macroeconomic level but also for monetary policy advise. This can resolve the Morgenstern's critique. [5] Also, as one of the

[1] see: Langville and Meyer (2011) and Stachurski and Martin (2008),

[2] For computer viruses and malware detection using Hidden Markov Models, see (Lin and Stamp, 2011)

[3] see speech: http://www.math.harvard.edu/ kmatveev/markov.html

[4] Analogous to difference equations, we are not interested in the systems dependent on time t as this will exponentially complicate our estimation technique.

[5] In his book, On the Accuracy of Economic Observations (1950), Morgenstern expressed his concerns in the way the data is used from the national income accounts to reach conclusions about the state of the economy and about appropriate policies. Note for Mathematicians: Morgenstern was a friend of John von Neumann.

leading graduate Economics textbook puts it "Understanding the causes of aggregate fluctuations is the central goal of macroeconomics"Romer (2006). Moreover, Romer (2006)[Ch4, pp.174] notes "A first important fact about fluctuations is that they do not exhibit any simple regular or cyclical pattern." Markov Chains substantially improve the prediction of the macroeconomic aggregates when compared to time series techniques, such as ARIMA models. Moreover, as shown by Tauchen (1986) we can pretty accurately approximate an ARMA process with a Markov Chain given enough states.

A broadly applicable algorithm for computing maximum likelihood estimates from incomplete data is the EM algorithm, see Dempster77. The work of A. P. Dempster (1977) was based on the Ph.D. thesis of Sundberg (1972) which provided a very detailed treatment of the EM method for exponential functions. The first to describe this EM algorithm in the paradigm of a mathematical maximization technique for probabilistic functions in Markov chains was Baum et al. (1970)[6]. The paper of (Rabiner, 1989) provided a practical guidance to understanding the results of (Baum and Petrie, 1966) and (Baum et al., 1970) and their application into an Engineering framework, specifically voice recognition tasks. In the same token, the papers (Hamilton, 2016), (Hamilton and Raj, 2002) and (Hamilton, 2005) adapted the mathematical techniques presented by (Baum et al., 1970) in estimating the parameters for the regime-switching models in describing economic aggregates like growth rates. The same theoretical aspects discussed by A. P. Dempster (1977), Rabiner (1989) and Baum and Eagon (1967) describe the Hidden Markov Models, moreover the EM algorithm is still the state of the art technique in estimating its parameters (Θ) for the underlying process of generating the observables which we denote by $\mathcal{O}$. Ideally we would want to have a robust method of estimating the parameters of an HMM which performs well not only on past observations but also predict future outcomes. Such models could easily be adjusted to augment SDGE (Stochastic Dynamic General Equilibrium) models which are currently based on systems of difference equations. Unfortunately, there are still no analytical methods for estimating the transition probability that would guarantee the maximum of probabilities of a certain output generated by a Markovian process and we would still need to use a heuristic approach in determining the "right" number of states within a hidden Markov model. This is because any attempt to use any estimation methodologies suitable to the framework of Markovian processes undoubtedly inherits all its problems (for example the EM algorithm does not guarantee you a global minimum while the Clustering algorithms will not be able to determine a reasonable amount of focal points without

[6]This is probably the reason this adaptation of the EM algorithm is called the Baum-Welch algorithm.

an abstract cost function). Therefore, solving a problem with a hidden Markov chain requires a numerical approach.

The good news is that as computers become more powerful, not only more iterations are possible but as we shall see when describing the Baum-Welch algorithm, more *shots* or attempts to find the maximum are possible along with the possibility to adjust the models to a higher order Markovian processes. This will ensure that the probability of the local optimum is closer or equal to the global one. Nevertheless, a heuristic approach in defining the model in combination with algorithms for estimating the transition probability matrix is, in my opinion, the most viable approach at the moment.

1 Introduction

> *One has to keep a particular openness of mind. Solving a problem is like going to a strange place, not to subdue it, but simply to spend time there, to preserve one's openness, to wait for the signals, to wait for the strangeness to dissolve into sense.* – Peter Whittle

A Markovian chain is a dynamical stochastic process which has the *Markovian property*.[7]. Before we formally introduce the notion of a *Markovian property*, it might be useful to take a step back and ask what a dynamical system is instead.

Using the notation of (Fraser, 2008), a dynamical system is a mapping $f(x_t) \mapsto \mathbb{R}^n$, where $x_t \in \mathbb{R}^n$ and t is a time-like index, which transitions the state x_t to x_{t+1}.

If this is also confusing perhaps the best way is to refer to real world examples: in Economics we might refer to x as the "State of the Economy", in tagging problems x could be the part of speech in a sentence, in biology x can be a tag from the set $\{A, T, G, C\}$ from the nucleotide sequences found in human DNA or x could be a binary variable corresponding to whether a student gave a correct answer to a particular problem at the PISA test.[8]. In Economic models, since "the state of the economy" is an abstract term which encapsulates various positive and normative elements of the economy,[9] we could restrict the values the economy can take to a particular set $\mathbb{X} = \{$ recession, mild- recession, mild-growth, growth $\}$. The set X is known as the *state space*.

Given $f(x)$, if $x(t)$ is known, one can deterministically find future values of $x(t+1), x(t+2) \ldots$ independently of previous states $x(t-1), x(t-2) \ldots$, making historical information unnecessary. This "uselessness of history", is also known as a *Markov property*. Statisticians might refer to the Markovian property by conditional independence of previous states given the current state. Therefore, a dynamical system is an instance of a Markov Chain since it satisfies the Markovian property.

One implication is that such models are particularly appealing in models which emphasise fundamental analysis for determining the intrinsic value of financial assets.[10]

[7] We define formally a Markov chain in section 2

[8] For example, at the Math PISA test x_{ij} could be whether student i answered correctly problem j.

[9] The National Bureau of Economic Research (NBER) defines recession as "a significant decline in economic activity spread across the economy, lasting more than a few months, normally visible in real GDP, real income, employment, industrial production, and wholesale-retail sales.", this is a much broader view than simply a decrease in GDP.

[10] For example, the Gordon Dividend Discount Model which is augmented with a stream of dividends that are

Dynamic stochastic general equilibrium models (abbreviated DSGE or sometimes SDGE or DGE), used by the most influential central banks could also be augmented with Markovian processes. We could think of any dynamical systems and find ways to improve it with Markovian processes.

Although Markov chains are useful in their own right, as we will show in section 2, one problem we face in practice when the state $x(t)$ is latent[11]. Usually we have lagged or only partial information about $x(t)$ and thus we can only estimate it. The information that is available to us, is called also called *emissions* in the literature, denoted by $y(t)$[12]. The observed variables are a function of the state the system is in, therefore we can represent:

$$y(t) \sim f(x(t)) \tag{1}$$

There are many types of Markov Chains, choosing the appropriate model depends on the specific problem being modelled:

1. First order Markov Processes
2. N-Order Markov Models
3. Hidden Markov Model (HMM)
4. Semi Markov Chains and Semi Hidden Markov Chains
5. Markov Chain Monte Carlo (MCMC)

Furthermore, each model can refer to different type of data { discrete, continuous }, on the other hand, if the state space model is continuous rather than finite and discrete then it is referred to the Harris chain. We will mostly focus on the HMM.

governed by a state transition matrix or a HMM which we will present in section 3.

[11] which is a fancy way of saying that variables of interest are not always directly observable. Example: Suppose you're looking for a partner and you want it to be intelligent. The IQ however is not directly observable, and you would have to infer it using his or her behaviour as a function of IQ.

[12] Which is a fancy way of saying "observations". The reason for that can be traced back to the applications of the Markovian processes in speech recognition tasks.

2 Markov Chains and Applications in Economics

" We may regard the present state of the universe as the effect of its past and the cause of its future " – Marquis de Laplace

First Order Markov Chain

Given a set $\mathbf{X}$ which forms the state space and $\mathbf{\Sigma}$ a σ-algebra on $\mathbf{X}$ and a probability measure Pr, a Markov Chain $\{x_t\}$ is a sequence of random variables with the property that the probability of moving from the present state x_t to next state x_{t+1} depends only on the present state.[13] This property can be written as:

$$Pr\left(X_t = x_i | X_{t-1}, X_{t-2}, ..., X_1\right) = Pr\left(X_t = x_i | X_{t-1}\right) \tag{2}$$

It is useful for abstraction purposes to represent a first order Markov process by a matrix $\mathbb{A}$ where the current state is represented by the row index. For this row to form a discrete probability distribution it must sum to 1.

$$\sum_{j \in X} a_{i,j} = 1 \quad , \forall i \in X \tag{3}$$

Therefore, a first order Markov process is simply a reinterpretation of a probability transition matrix $\mathbb{A}$, also called the stochastic matrix, where $a_{ij} \in \mathbb{A}$ represents the probability of observing outcome j at $t+1$ if at time t we are observing i. In it's simplest form, the future state depends only on the state we are currently in.[14] We will deal only with time-homogeneous Markov chains in this paper.

Definition 2.1 (Time Homogeneous Markov Chain)**.** A time homogeneous Markov Chain is a MC that has a time invariant state-space probability matrix.

Using the example from Hamilton (2005)[15] in his paper "What's Real About the Business

[13] We could also think of an zero-order Markov process, the case when the current state is completely independent of the previous state, like throwing a dice. But then we simply get back to a classical probability distribution. If $(\mathbf{\Omega}, \mathbf{\Sigma}, \mathbf{Pr})$ is a discrete sample space where $\mathbf{\Omega}$ is the set of all the possible outcomes, $Pr : \Sigma \mapsto \mathbb{R}$ where $\sum_{x_i \in \mathbf{\Omega}} Pr(x_i) = 1$. In this case: $Pr\left(X_t = x_i | X_{t-1}, X_{t-2}, ..., X_1\right) = Pr\left(X_t = x_i\right)$, therefore, it is completely redundant to introduce a zero-order Markov processes.

[14] Please note that in a simple Markov Chain, unlike a Hidden Markov Model which we will define later, the states are observable.

[15] One of the key insights of this paper is that a linear statistical model with homoskedastic errors cannot capture the nineteenth-century notion of a recurring cyclical pattern in key economic aggregates and that a simple Markov chain has a much better goodness of fit.

Cycle?", we can write the state space transition matrix corresponding to the states $\mathbf{X} = \{$normal growth, mild recession, severe recession $\}$ as:

$$\mathbb{A} = \begin{pmatrix} 0.971 & 0.029 & 0 \\ 0.145 & 0.778 & 0.077 \\ 0 & 0.508 & 0.492 \end{pmatrix}$$

How is this useful in practice? [16] One example is to determine how many periods[17] of time a state will persist given that the current state $X_t = x_t$ as asked in the seminal paper of (Rabiner, 1989). One way to answer this question is to simulate the states generated by this matrix using Monte Carlo methods and then use the frequency approach to get an estimate. A better approach, shown in the paper of Rabiner (1989) is to observe that staying in a particular number of periods in a state follows a geometric series e.i. if we want to compute the probability the system will stay exactly 2 periods of time in the normal growth given that the current period $x_t =$ normal growth it will be:

$$\mathbb{P} = a_{11}^2(1 - a_{11}) \tag{4}$$

We know that the average value of x denoted by $\bar{x}$ is

$$\bar{x} = \sum x\mathbb{P}(x) \tag{5}$$

similar to our case: the average (expected) number of days to stay in a particular state:

$$p\bar{e}r = \sum_{per=1}^{\infty} pera_{ii}^{d-1}(1 - a_{ii}) \tag{6}$$

which is the same as the well know geometric series[18]:

$$\sum_{k=1}^{n} kz^k = z\frac{1 - (n+1)z^n + nz^{n+1}}{(1-z)^2} \tag{7}$$

[19] Now taking $(1 - a_{ii})$ in front and taking the limit $\lim_{t\to\infty} a_{ii}^t = 0$ we get

$$p\bar{e}r = (1 - a_{ii})\frac{1}{(1 - a_{ii})^2} = \frac{1}{(1 - a_{ii})} \tag{8}$$

[16] You can also simulate a Markov Chain given a stochastic matrix at http://setosa.io/ev/markov-chains/

[17] period is an abstract term, in the paper of Rabiner (1989) days are assumed, in the example of Hamilton and Raj (2002) months, however for economic aggregates usually quarters are assumed

[18] according to a Math professor from MIT (quote needed), this is the second most beautiful series in Math after e^x

[19] other beautiful series derived from the geometric series: http://lycofs01.lycoming.edu/ sprgene/M332/Sums_Series.pdf

So if we are in the state of *normal growth* at t then we would expect:

$$\mathbb{E}[\text{periods of growth}|x_t = \text{normal growth}] = \frac{1}{1 - 0.971} \approx 34$$

This brings us back to the original question I posed about estimating a Markovian chain. If we know the expected number of periods a state persists in, we can calculate $a_i i$ from the stochastic matrix $\mathbb{A}$.

What is more useful in practice is that this matrix is of use when calculating the expected pay-off of a particular pro-cyclical investment project[20] or even better suited for financial assets. Let's assume the following net pay-offs as a function of the state:

$$\mathbb{E} = \begin{pmatrix} 0.215 \\ 0.015 \\ -0.18 \end{pmatrix}$$

That is, if the economy is in normal growth state, we expect the Internal Rate of Return to be 21.5 $\%$. [21]

If the state x_t is known at t the expected pay-off is trivial:

$$E[t+1|X_t = \texttt{mild recession}] = \sum_{j \in X} \mathbf{a}_{2,j} \mathbf{e}_j$$

If the current state is not known, we can use a discrete distribution to assess in which state the economy is at time t, following (Lozovanu and Pickl, 2015) we will denote it by μ.

$$\mu = \begin{pmatrix} 0.1 \\ 0.55 \\ 0.35 \end{pmatrix}$$

We can already calculate the expected pay-off using the Octave programming language.[22]

```
A = [0.97100 0.02900 0.00000;
    0.14500 0.77800 0.07700;
    0.00000 0.50800 0.49200 ];
E = [0.215, 0.015, -0.18];
E = transpose(E);
mu = [0.1, 0.55, 0.35];
```

[20] data from the upstream oil industry as an example

[21] In practice it is easier to estimate the profits of a given project in a year given the state of the economy.

[22] Octave programming language is very similar to Matlab, except that it is free and open source.

```
#for the next period, given the state
A*E
```

```
ans =

    0.209200
    0.028985
   -0.080940
```

```
#if the state is not known
>> mu*A*E
ans = 0.0085328
```

Given that the expected pay-off is basically zero, we might wrongly conclude that the project is not worth investing in.

In petroleum industries however, as in the most other industries, the time horizon is not uncommon to be around 25 years. According to the signed Production Sharing Contracts (PSC) from the Kurdistan Region of Iraq published by the KRG Ministry of Natural Resources[23] we can freely examine a sample of 43 PSCs, the majority of them (41 out of 43) have a development period of 25 years (including the 5 years optional extension period).[24] Using this data, we can compute the expected pay-offs at time $t = 25$ in the following way:

$$E_{\text{Payoff}}[t = 25] = \mu \mathbb{A}^{25} \mathbb{E} \tag{9}$$

We can see that even in our example, computing $\mathbb{A}^{25}$ by hand is a very tedious task. And since matrix multiplication is very computationally intensive, computing matrices when $t \mapsto \infty$ becomes an issue.

One way to solve this problem is to check if all the column vectors in $\mathbb{A}$ are independent. We can solve for the eigenvalues of $\mathbb{A}$ and if we have as many different eigenvalues n as columns in $\mathbb{A}$

[23] The Production Sharing Contracts can be found at http://cabinet.gov.krd/p/p.aspx?l=12&p=1

[24] This can be usually found at Article 6 clause 6.9, 6.10, 6.11 and 6.12, for example the Contract signed between Marathon and KRG at http://cabinet.gov.krd/p/p.aspx?l=12&r=296&h=1&s=030000&p=70

then we can find n distinct eigenvectors and decompose $\mathbb{A}$ into its canonical form:

$$\mathbb{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-\mathbf{1}} \tag{10}$$

where $\mathbf{S}$ is the matrix of eigenvectors and $\mathbf{\Lambda}$ is the identity matrix multiplied by the vector of eigenvalues. Now to solve for

$$E_{\text{Payoff}}[t = 25] = \mu\mathbb{A}^{25}\mathbb{E} = \mathbf{S}\mathbf{\Lambda}^{25}\mathbf{S}^{-\mathbf{1}}\mathbb{E} \tag{11}$$

Solving for $\mathbf{\Lambda}^{25}$ is a lot easier than $\mathbb{A}^{25}$ since $\mathbf{\Lambda}$ is a diagonal matrix. Moreover, the highest eigenvalue of a the stochastic matrix generating a Markov chain is 1. This is important since if a matrix is:

$$\mathbf{x}\mathbb{A} = \lambda_i\mathbf{x} \tag{12}$$

then the stochastic matrix following a Markov chain can be written as:

$$\mathbf{x}\mathbb{A} = \mathbf{x} \tag{13}$$

which implies that $\mathbf{x}$ is a stationary probability distribution.[25] To find the eigenvalues of $\mathbb{A}$ we proceed as follows:

$$\mathbf{v}(\mathbb{A} - I\lambda) = \mathbf{0}$$

where $\mathbf{v}$ is one of the non-trivial eigenvectors[26]. Determining the set of all λs for which the determinant of $\mathbb{A} - I\lambda$ in the above equation is zero is simply solving an n^{th} order polynomial which I would rather do in Octave as follows:

$$|\mathbb{A} - I\lambda| = \begin{vmatrix} 0.971 - \lambda & 0.029 & 0 \\ 0.145 & 0.778 - \lambda & 0.077 \\ 0 & 0.508 & 0.492 - \lambda \end{vmatrix} = 0$$

```
>> eigs(A)
ans =
  1.00000
  0.85157
  0.38943
```

[25] $\mathbf{x}\mathbb{A} = \mathbf{x}$ should not be confused with $\mathbb{A}\mathbf{x} = \mathbf{x}$ since any vector $\mathbf{x}$ which has $x_i = x_j$ satisfies this equality.

[26] that is $\mathbf{v}$ is not a zero vector

That is:

$$\lambda = \begin{pmatrix} 1.00000 \\ 0.85157 \\ 0.38943 \end{pmatrix}$$

We can verify that these are indeed the eigenvalues of $\mathbb{A}$ by comparing the trace of $\mathbb{A}$ with the sum of the eigenvalues and the determinant of $\mathbb{A}$ should be equal to the product of the eigenvalues.[27]

```
>> trace(A)
ans = 2.2410
>> sum(eig(A))
ans = 2.2410
>> prod(eig(A))
ans = 0.33163
>> det(A)
ans = 0.33163
```

Now having found the eigenvalues, we substitute each of the eigenvalues into λ and get a 3 degenerate matrices. We can easily verify this:

```
>> det(A-eye(3))
ans = -1.6611e-18
```

which we assume is 0 due to rounding errors.

Then we find the null space of these new matrices and find the eigenvectors corresponding to each eigenvalue.

```
>> [evects, evals] = eigs(A)
evects =

  0.5773503 -0.1389312 0.0098689
  0.5773503 0.5721371 -0.1979135
  0.5773503 0.8083052 0.9801698

evals =
```

[27] In Octave as well as in other programming languages directly comparing sum(eig(A)) == trace(A) will usually not work due to rounding errors.

```
Diagonal Matrix


  1.00000        0        0
        0  0.85157        0
        0        0  0.38943
```

Therefore, since $\mathbf{S}$ is:

$$\mathbf{S} = \begin{pmatrix} 0.5773503 & -0.1389312 & 0.0098689 \\ 0.5773503 & 0.5721371 & -0.1979135 \\ 0.5773503 & 0.8083052 & 0.9801698 \end{pmatrix}$$

then $\mathbf{S}^{-1}$ is:

```
>> pinv(evects)
ans =
```

$$\mathbf{S}^{-1} = \begin{pmatrix} 1.407811 & 0.281562 & 0.042678 \\ -1.328512 & 1.094198 & 0.234314 \\ 0.266324 & -1.068188 & 0.801864 \end{pmatrix} \tag{14}$$

Now we are able to compute the values of the vector from equation 10 :

```
>> evects*(evals^25)*pinv(evects)
ans =
```

$$\mathbf{S}\mathbf{\Lambda}^{25}\mathbf{S}^{-1} = \begin{pmatrix} 0.816125 & 0.159822 & 0.024054 \\ 0.799109 & 0.173836 & 0.027055 \\ 0.793458 & 0.178491 & 0.028051 \end{pmatrix} \tag{15}$$

What if $\mathbb{A}$ is not irreducible[28] like in a for of a diagonal matrix? In this case, even if we get n different eigenvalues we will not get n orthogonal[29] eigenvectors. To solve these types of problems we can use the algorithms proposed by (Lozovanu and Pickl, 2015) for solving for $\mathbb{Q}$[30] which can solve it in $\mathcal{O}(n^3)$ operations.

If eigenvectors seem too foreign or the method is too confusing and since we only have 3 states and $t = 25$ we can directly solve it using any programming language. In Octave or Matlab it is as

[28] we will define what irreducible is later.

[29] orthogonal is just another way of saying perpendicular or independent vectors

[30] the limiting probability matrix is denoted by $\mathbb{Q}$

simple as:

```
>> mu*A^25*E
ans = 0.16948
```

Now the expected return is 17% and the decision for Marathon to invest in developing this region will depend on on their ability to attract capital with less than 17% interest as well as the availability of other more attractive opportunities.[31]

Another curiosity is the range the IRR takes as a function of μ. Again, with a Markovian chain this is trivial once we have our limiting probability matrix $\mathbb{Q}$ or $\mathbb{A}^t$. Using the transition matrix provided by Hamilton (2005), the long term expectation of being in a particular state can be written as:

$$E[X] = \mu\mathbb{Q} \tag{16}$$

Even if we take μ to be $[0, 0, 1]$ e.i. the worst case scenario:

```
>> mu
mu =
   0   0   1
>> mu*A^25
ans =
   0.793458 0.178491 0.028051
>> mu*A^25*E
ans = 0.16822
```

we can still expect a 16.8% return. To understand why this is so, we have to introduce the Fundamental theorem of Markov Chains.

Irreducible Stochastic Matrices

Definition 2.2. A stochastic matrix, $\mathbb{A}$ is **irreducible** if its graph is strongly connected, that is: there $\exists\ t \geq 0 : Pr(X_t = j|X_0 = i) > 0$

We will denote by $P^t(i, j) = Pr(X_t = j|X_0 = i)$. In the example from section 2 our stochastic

[31] Of course we are assuming that the space state transition probability matrix $\mathbb{A}$ computed by Hamilton (2005) for the United States is valid for Kurdistan Region of Iraq. For these conclusions to have any validity, the study of Hamilton needs to be replicated for KRG and more than 3 states would be desirable to be used.

matrix is irreducible since we can end up in any state from any state after $t \geq 1$ steps.

Aperiodic Stochastic Matrices

Definition 2.3. A stochastic matrix, $\mathbb{A}$ is **aperiodic** if the greatest common divisor of the set S(x) defined as

$$S(x) = \{t \geq 1 : P^t(x, x) > 0\}$$

equals 1.

We can easily check that matrix $\mathbb{A}$ from our example is aperiodic.

Stationary Distribution

Definition 2.4. A probability distribution π over X is **stationary** over $\mathbb{A}$ if:

$$\pi = \pi\mathbb{A}$$

As we have shown, the stochastic matrix in the Markov chain presented by Hamilton (2005) is both irreducible and aperiodic. A theorem proven by Häggström (2002) states that:

Theorem 2.1 (Fundamental Theorem of Markov Chains)**.** *If a stochastic matrix $\mathbb{A}$ is irreducible and aperiodic then there is a unique probability distribution π that is stationary on $\mathbb{A}$.*

Proof. provided by Häggström (2002). □

A direct result of this theorem is that Marathon Oil company can expect:

$$\lim_{t\to\infty} \mu\mathbb{A}^t = \pi \tag{17}$$

It would be interesting to compare our results with the limiting probability distribution, e.i. when $t \to \infty$. To find the limiting probability distribution, observe that any $\lambda_i < 1$ will tend to be 0 when $t \to \infty$, therefore we can write:

$$\phi = \lim_{t\to\infty} \mathbf{S}\Lambda^t\mathbf{S}^{-1} = \mathbf{S}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\mathbf{S}^{-1} \tag{18}$$

```
evects*[1,0,0;0,0,0;0,0,0]*evects^(-1)
ans =

   0.812800 0.162560 0.024640
   0.812800 0.162560 0.024640
   0.812800 0.162560 0.024640
```

Therefore the limiting probability distribution ϕ:

$$\phi = \left(\ 0.812800, 0.162560, 0.024640 \ \right)$$

which is surprisingly close to our results when $t = 25$.

Having the state space stochastic matrix, we can answer how long does a recession usually last? To answer this question, as well as questions related to analysis of variance (ANOVA), we could use Monte Carlo simulations.

Simulating a Markov Chain

In order to have a better understanding of the implicit variance of the Markov chain presented above we can simulate it. There are a plethora of open source libraries providing frameworks to accomplish such simulations. We will make our own using the open source QuantEcon package from GitHub which was written by Stachurski and Sargent. (2016). We will simulate our example using the Julia programming language for the example from Hamilton based on software recommendations from Stachurski and Sargent. (2016). Below we provide a working example. The requirements to replicate this example will be provided in the Annex.

Inputting the data: The probability transition matrix $\mathbb{A}$, the initial state probability vector μ and pay-off expectations $\mathbb{E}$

```
A = [0.971 0.029 0; 0.145 0.778 0.077; 0 0.508 0.492] # Probability
   Transition Matrix
mu = [0.1 0.55 0.35] # initial state probability vector
E = [0.215 0.015 -0.18]' #column vector of pay-off expectations -
   project specific

using QuantEcon
```

A sample for Markov Chain is given by the function *MarkovChain_sample* which takes as input matrix $\mathbb{A}$ - the probability transition matrix and the initial state probability distribution vector μ. Optionally, we can provide the *sample_size* or t which in our case for the petroleum industry is 25 years.

```
function MarkovChain_sample(A, mu; sample_size=25)
   X = Array(Int16, sample_size)
   p_mu = DiscreteRV(vec(mu))
   X[1] = draw(p_mu)
   Pr_A = [DiscreteRV(vec(A[i,:])) for i in 1:(size(A)[1])]
   for t in 2:(sample_size)
      X[t]=QuantEcon.draw(Pr_A[X[t-1]])
   end
   return X
end
```

The array X is the container of Integers where we will store the state from one instance of the Markov process. The function *DiscreteRV* from the QuantEcon package take as input a vector whose elements sum to 1 and converts it into a discrete probability distribution. The variable P_μ is therefore the initial state discrete probability distribution from which we determine the first state our system will take. The *draw* function form the QuantEcon package takes a probability distribution as input and outputs a sample output. In line 4 we convert our probability transition matrix into n distinct discrete probability distribution and then we assign a state to X_t from a random draw according to the pdf of X_{t-1}. Here is a sample output when we call *MarkovChain_sample*

```
iaka = MarkovChain_sample(A,mu)
iaka'

Out:
1x25 Array{Int16,2}:
 2  2  1  1  1  1  1  1  1  1  1  1  1  1  1  3  3  2  2  2  2  1  1  1
```

Once we have a sample path and the pay-off vector $\mathbb{E}$, we can calculate the result of the investment

project. Assuming for simplicity that the discount β rate is zero[32] the pay-off would simply be:

$$\text{Payoff} = \prod_{t=1}^{25}(1 + E(x_t)) \tag{19}$$

Below we present a Monte Carlo procedure to simulate a Markov Chain without discount.

```
function MarkovChain_simulation(ff::Function, A, mu, E; nr_iter=100,
   t=25)
   X = Array(Float32,nr_iter)
   #sample_path = Array(Int16, t)
   for i in 1:nr_iter
      sample_path = ff(A,mu, sample_size= t)
      X[i] = sum([log(1+E[sample_path[i]]) for i in
         1:length(sample_path) ])
   end
   return X
end
```

where in addition to the parameters from *MarkovChain_sample* we have *ff* which is a reference to a function, and vector $\mathbb{E}$ the pay-offs vector. The function *MarkovChain simulation* returns an array X of logarithmic returns without a discount. These classes of simulations are useful for simulating the Future Value of a financial asset. Also, this algorithm is not very efficient since it uses iterative processes which do not take advantage for multiprocessor architectures in modern computers. Therefore, we propose a slightly improved procedure for simulating a Markovian process:

```
function MC_sim(ff, A, mu, E; nr_iter=10,t=25)
   X = Array(Float32,nr_iter)
   for i in 1:nr_iter
      sample_path = ff(A,mu, sample_size= t)
      X[i]=mapreduce(x->log(1+E[x]),+, sample_path)
   end
   return X
end
```

[32] we will present a model when the discount rate is zero

Here we take advantage of the multi-processor architecture using the mapreduce procedures with the plus binary operator. To be of any use for our petroleum example, we would also need to discount each return with respect to the internal cost of capital β. Without loss of generality we assume $\beta = 0.1$

```
function MC_sim_discount(mc::Function, A, mu, E; nr_iter=10,t=25,
   discount=0.1)
   """
   Calculates the cummulative Economic profit, based on the discount
      rate
   Required:
   MC_sim_discount - Markov Chain Simulation with discount
   mc - The function that we want to simulate (ex: Markov Chain sample
      path )
   A - Probability Transition Matrix nxn stochastic matrix
   mu - initial state probability distribution 1xn vector
   E - expected pay-off emission probability nx1 vector

   Optional:
   nr_iter - Integer+
   t - number of periods in the mc function
   discount - the internal cost of capital or normal rate of return
   """
   X = Array(Float32,nr_iter)
   for i in 1:nr_iter
      MarkovChain_sample_path = mc(A,mu, sample_size= t)
      X[i]=mapreduce(x->log(1+E[x[2]])-(log(1+discount)),+,
         enumerate(MarkovChain_sample_path))
   end
   return X
end
```

Here we take advantage of the log returns property to discount each pay-off as a function of the state the system is in. We can still improve the above algorithm, at the cost of readability, by getting rid of the first *for* loop and calculate the average log return rather than the cumulative economic

return.

```
function MC_sim(mc::Function, A, mu, E; nr_iter=10,t=25)
    """
    Markov Chain Simulation computes the average log return
    Required:

    mc - The function that we want to simulate (ex: Markov Chain sample
        path )
    A - Probability Transition Matrix nxn stochastic matrix
    mu - initial state probability distribution 1xn vector
    E - expected pay-off emission probability nx1 vector

    Optional:
    nr_iter - Integer+
    t - number of periods in the mc function
    """

    X = Array(Float32,nr_iter)
    map!(y->mapreduce(x->log(1+E[x]),+, mc(A,mu, sample_size= t))/t,X)
    return X
end
```

Now, thanks to the very high level paradigm of Julia language and onion code, our simulation function is effectively only three lines of code, though much less readable.

Now we can proceed to simulate our Markov Chain using different number of iterations to observe how the variance behaves. Since a picture is a thousand words we will plot the log returns for $n = \{10^3, 10^4, 10^5, 10^6\}$ where n is the number of iterations.

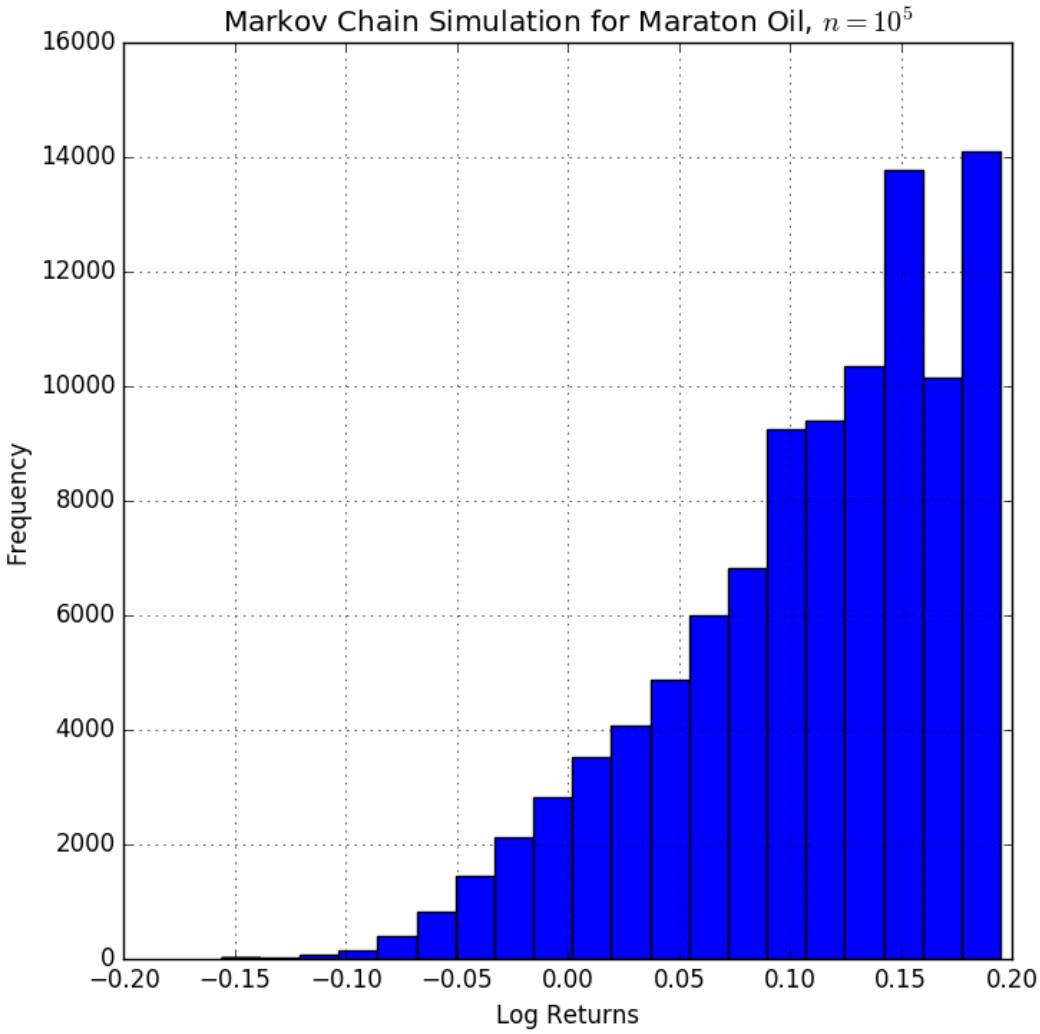
```
n = 1000
simulation = MC_sim(MarkovChain_sample,A, mu, E; nr_iter=n);

fig = figure("pyplot_histogram",figsize=(8,8))
ax = axes()
h = plt[:hist](simulation,20)
grid("on")
```

```
xlabel("Log Returns")
ylabel("Frequency")

title("Markov Chain Simulation for Marathon Oil, \$n=10^3\$")
```

Figure 1: Markov Chain Simulation



For $n = \{10^3, 10^4, 10^6\}$ see the Annex.

Second and N-order Markov Processes

A second order, third order up to nth order Markov process behaves the same as described by 2, except:

$$Pr\left(X_t = x_i | X_{t-1}, X_{t-2}, ..., X_1\right) = Pr\left(X_t = x_i \middle| X_{t-1} = x_{t-1}, ..., X_{t-n} = x_{t-n}\right)$$

Given observable outcomes, choosing a model is a trade-off between a parsimonious model and a better goodness of fit. Given a likelihood function of the model $\mathbf{\Lambda}$, once can use a loss function, maximize a probability distribution or use AIC, e.i. $2k - \ln \mathbf{\Lambda}$. Another favourable characteristic of the N-order Markov Processes to be useful in practice is that the transition probability matrix $\mathbb{A}$ be ***irreducible***.

Theorem 2.2. *Any N-order Markovian Process can be represented by a first order Markovian process.*

Proof. This is rather a trivial proof. Suppose we observe a process that is indeed governed by the function $f(x, y) \to \mathbb{R}$ where x, $y \in$ state spaces S, V respectively and we want to transform $f(x, y)$ into $f(z)$. We can define a state space $z \in T = \{(x_i, y_j), \quad x_i \in S \quad y_j \in V\}$ and then rewrite $f(x, y)$ as $f(z)$. □

Of course the proof goes beyond saying that we can represent any 2-nd order Markov Chains, e.i.:

$$Pr\left(X_t | X_{t-1}, X_{t-2}, ..., X_1\right) = Pr\left(X_t = x_i | X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}\right)$$

as:

$$Pr\left(X_t | X_{t-1}, X_{t-2}, ..., X_1\right) = Pr\left(X_t = x_i | (X_{t-1}, X_{t-2})\right)$$

where (X_{t-1}, X_{t-2}) is defined as a new state. What the second fundamental theorem of Markov Chains suggests is that even if we have a very complex dynamic process which emits observations ϵ_i, we can still describe it by a first order Markov chain, given enough states.

Semi Markov Chains and Semi Hidden Markov Models

The SMC and the SHMM are a generalization of the MC and HMM in the sense that they:

1. Allow arbitrarily distributed sojourn times in any state

2. Still have the Markovian hypothesis, but in a more flexible manner.

As defined by Barbu and Limnios (2008)[pp. 2] a process that has these two properties will be called a semi-Markov process.

Continuous State Markov Chains

A continuous state Markov Chain extends the model presented in section 2 by allowing a probability density distribution on the states. These have been analysed extensively in the paper of Benesch (2001). More formally, a stochastic kernel on $\mathbb{S}$ is a function $p : \mathbb{S} \times \mathbb{S} \in \mathbb{R}$ with the property that:

$$p(x, y) \geq 0 \quad \forall x, y \in \mathbb{S}$$
$$\int_{-\infty}^{+\infty} p(x, y)dy = 1 \quad \forall x \in \mathbb{S}$$

For example, suppose the random variable X_t is characterized by the famous normally distributed random walk:

$$X_{t+1} = X_t + \xi_{t+1} \quad \text{where} \quad \{\xi_t\} \overset{\text{IID}}{\sim} N(0, 1) \tag{20}$$

We could characterize this random distribution through the use of a continuous state Markov Chain, specifically by defining the transition probability $p(x_t, x_{t+1})$ analogous to $\mathbb{A}$[33] to be:

$$p(x_t, x_{t+1}) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{(x_{t+1} - x_t)^2}{2}\right\} \tag{21}$$

Combining the ideas from the blog of Stachurski and Sargent. (2016) as well as the seminal paper Tauchen (1986) we can connect Stochastic difference equations to the probability kernel.

$$X_{t+1} = \mu(X_t) + \sigma(X_t)\xi_{t+1} \tag{22}$$

where $\{\xi_t\} \overset{\text{IID}}{\sim} \phi$ and μ, σ are functions. This is clearly a Markov process, since the state of the system at $t+1$ depends only on the current state t. Under this equation, which we will call *generic Markov process*, we can write the normally distributed random walk stochastic process, as shown in 20 as a special case of equation 22 when $\sigma(x_t) = 1$ and $\mu(x_t) = x_t$.

Consider X_t following an ARCH(1) process:

$$y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + ... + a_q y_{t-q-1} + X_t$$
$$y_{t+1} = a_0 + a_1 y_t + a_2 y_{t-1} + ... + a_q y_{t-q} + X_{t+1}$$
$$X_{t+1} = \alpha X_t + \sigma_t \xi_{t+1}$$
$$\sigma_t^2 = \beta + \gamma X_t^2, \quad \text{where} \beta, \gamma > 0$$

This is a special case of equation 22 with $\sigma(x) = (\beta + \gamma x^2)^{1/2}$ and $\mu(x) = \alpha x$

Moreover, it is useful to write equation 22 in a form of a probability kernel.

[33] also called stochastic kernel in literature, see for example http://quant-econ.net/jl/stationary_densities.html

Theorem 2.3. *Any Markov Process in the form* $X_{t+1} = \mu(X_t) + \sigma(X_t)\xi_{t+1}$ *as specified in equation 22 where* $\xi_{t+1} \sim \phi$ *can be written as:*

$$Pr(x_{t+1}|x_t) = \frac{1}{\sigma(x_t)}\phi\left(\frac{x_{t+1} - \mu(x)}{\sigma(x)}\right) \tag{23}$$

Proof. Let U and V be two random variables with probability density functions $f_U(u)$ and $f_V(v)$ and the cumulative probability distributions F_U and F_V respectively and $V = a + bU$ where $a, b \in \mathbb{R}$ and $b > 0$. Theorem 8.1.3 from Stachurski and Sargent. (2016) proves that in this case: $f_V(v) = \frac{1}{b} f_U\left(\frac{v-a}{b}\right)$ and since $\sigma(x)$ is the square root of $\sigma^2(x)$ results that $\sigma(x) > 0$ and we can apply Theorem 8.1.3 directly in our case which completes the proof. Proving theorem 8.1.3 from Stachurski and Sargent. (2016) is also straightforward: We know that $F_V(v) = \mathbb{P}\{V \leq v\}$, and from the assumption that $V = a + bU$ we substitute V and obtain $F_V(v) = \mathbb{P}\{a + bU \leq v\} = \mathbb{P}\{U \leq (v - a)/b\}$. We can now write that:

$$F_V(v) = F_U((v - a)/b) \tag{24}$$

and since the probability density function $f_V(v)$ is the derivative of the cumulative probability distribution $F_V(v)$ with respect to v, we take the derivative of $F_V(v)$ and obtain:

$$f_V(v) = \frac{1}{b} f_U\left(\frac{v - a}{b}\right) \tag{25}$$

□

For example, following the Solow-Swan model[34] presented in Romer (2006) the capital per capita k difference equation is:

$$k_{t+1} = sA_{t+1}f(k_t) + (1 - \delta)k_t \tag{26}$$

where

1. s is the savings ratio
2. δ is the normal depreciation rate of the capital
3. A_{t+1} is the production shock at time $t + 1$ which is latent at time t

[34] In a nutshell, the Solow–Swan model assumes a closed market economy. A single good (output) is produced using two factors of production, labour L and capital K in an aggregate production function that satisfies the Inada conditions, which imply that the elasticity of substitution must be asymptotically equal to one. https://en.wikipedia.org/wiki/Solow-Swan_model

4. and $f\colon \mathbb{R}_+ \to \mathbb{R}_+$ is a production function, usually of the labour augmenting Cobb-Douglas form.

Equation 26 is the driving force of the most popular economic growth model in Economics.[35] Since in equation 26 the production shock A_{t+1} is a random variable, it is also a special case of equation 22 with $\mu(x) = (1-\delta)x$ and $\sigma(x) = sf(x)$. Now we can also write the probability kernel of equation 26 as:

$$p(x, y) = \frac{1}{sf(x)}\phi\left(\frac{y-(1-\delta)x}{sf(x)}\right) \tag{27}$$

where $\phi \sim A_{t+1}$, $x = k_t$ and $y = k_{t+1}$.

Having defined the continuous probability transition density, we can generalize the formula for the probability state density at $t+1$. In the continuous case, if the distribution of $X_t \sim \psi_t$ then ψ_{t+1}, analogous to the sum as specified in section 2 for the discrete case:

$$\psi_{t+1}(y) = \int p(x, y)\psi_t(x)\, dx, \qquad \forall y \in S \tag{28}$$

Simulating a Continuous State Markov Chain

Once we have established a probability kernel and having the initial probability state distribution ψ_0, we can proceed to estimating the state density distribution ψ_1 at $t = 1$. The straight forward way for simulating the growth of capital as described in equation 26: $k_{t+1} = sA_{t+1}f(k_t)+(1-\delta)k_t$ is to:

1. draw k_0 from the initial probability density ψ_0
2. draw n parameters from the probability density ϕ, in the case of the model at equation 26 these are the technology shocks $A_1 ... A_n$.
3. repeat by computing k_{t+1} from 26 and store them in an array.

Once we have n instances of k_{t+1}s, we can use kernel density estimates functions to make inferences about the probability distribution.[36]

The paper of Stachurski and Martin (2008), based on the look-ahead estimator, provides an improved Monte Carlo algorithm for computing marginal and stationary densities of stochastic

[35] Robert Solow has was awarder the Nobel Prize in Economics.

[36] For example the open source library KernelDensity for Julia programming language which is hosted at https://github.com/JuliaStats/KernelDensity.jl

models with the Markov property, establishing global asymptotic normality and fast convergence. The idea is that, by the strong law of large numbers:

$$\frac{1}{n}\sum_{i=1}^{n} p(k_{t-1}^{i}, y) \to \mathbb{E}p(k_{t-1}^{i}, y) = \int p(x, y)\psi_{t-1}(x)\, dx = \psi_t(y) \tag{29}$$

where $p(x, y)$ is the example specific stochastic kernel e.i. $p(x, y) = \frac{1}{sf(x)}\phi\left(\frac{y-(1-\delta)x}{sf(x)}\right)$

Therefore, we can write our continuous state probability density as the average of probabilities:

$$\psi_t^n(y) = \frac{1}{n}\sum_{i=1}^{n} p(k_{t-1}^{i}, y) \tag{30}$$

Since an efficient implementation of the Monte Carlo simulation for the continuous state Markov process requires a more significant number of steps than in the discrete case, as shown in section 2, and also taking into account that the continuous case is not conceptually different than in the discrete case, providing a step by step implementation as in section Simulating a Markov Chain is beyond the scope of this thesis. On the other hand, I will present the implementation of Stachurski and Martin (2008) in Julia language in the Annex for convenience purposes.

Given the Solow-Swan model, and using the implementation of the `LAE, lae_est` as provided in the Annex, we can now ask how fast does the initial density probability function for the continuous state Markov Chain converge to the steady state distribution of capital per capita k. To answer this question we only need to write the function for the probability kernel $p(x, y)$.

```
function p(x, y)
    #=
    Stochastic kernel for the growth model with Cobb-Douglas production.
    Both x and y must be strictly positive.
    =#
    d = s * x.^alpha
    pdf_arg = clamp((y .- (1-sigma) .* x) ./ d, eps(), Inf)
    return pdf(phi, pdf_arg) ./ d
end
```

The idea is that any initial state density distribution for the k_0 will converge to a steady state density distribution. Suppose the initial density distribution follows:

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0, 1] \tag{31}$$

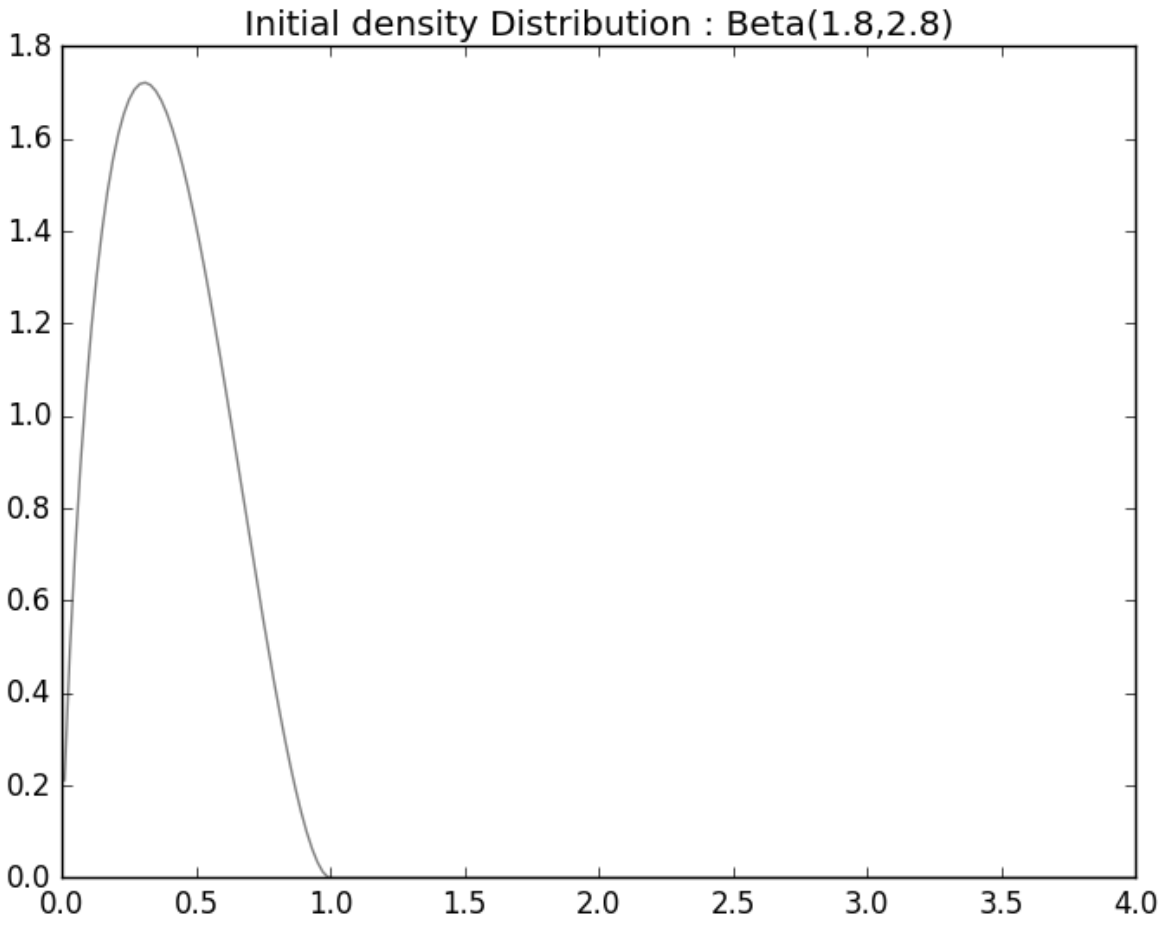
which is known as the Beta Distribution.

```
a_sigma = 0.4
phi = LogNormal(0.0, a_sigma) #Technological Change Distribution
psi_0 = Beta(1.8, 2.8)
ygrid = linspace(0.01, 4.0, 200)
```

```
fig, ax = subplots()
#for (x,y) in zip(ygrid, ygrid)
ax[:plot](ygrid, pdf(psi_0, ygrid), color="0.5")
t=LaTeXString(" Initial density Distribution: Beta(1.8,2.8) ")
ax[:set_title](t)
show()
```

We can plot the initial distribution of ψ_0 and then by iteratively applying the lae function with

Figure 2: Beta Distribution



the density kernel, we can observe the speed of convergence.

```
s = 0.2 # Savings Rate
\delta = 0.1 # Capital Depreciation Rate
a_\sigma = 0.4 # A = exp(B) where B ~ N(0, a_sigma)
\alpha = 0.4 # We set f(k) = k**alpha
\psi_0 = Beta(1.8, 2.8) # Initial Continuous State distribution
\phi = LogNormal(0.0, a_\sigma)


n = 10000 # Number of observations at each date t
T = 30 # Compute density of k_t at 1,...,T+1

# Generate matrix s.t. t-th column is n observations of k_t
k = Array(Float64, n, T)
A = rand!(\phi, Array(Float64, n, T))

# Draw first column from initial distribution
k[:, 1] = rand(\psi_0, n) # divide by 2 to match scale=0.5 in py
   version
for t=1:T-1
   k[:, t+1] = s*A[:, t] .* k[:, t].^\alpha + (1-\delta) .* k[:, t]
end
```

Now let us take a more sophisticated probability distribution and observe the convergence. Suppose the initial continuous state distribution follows:

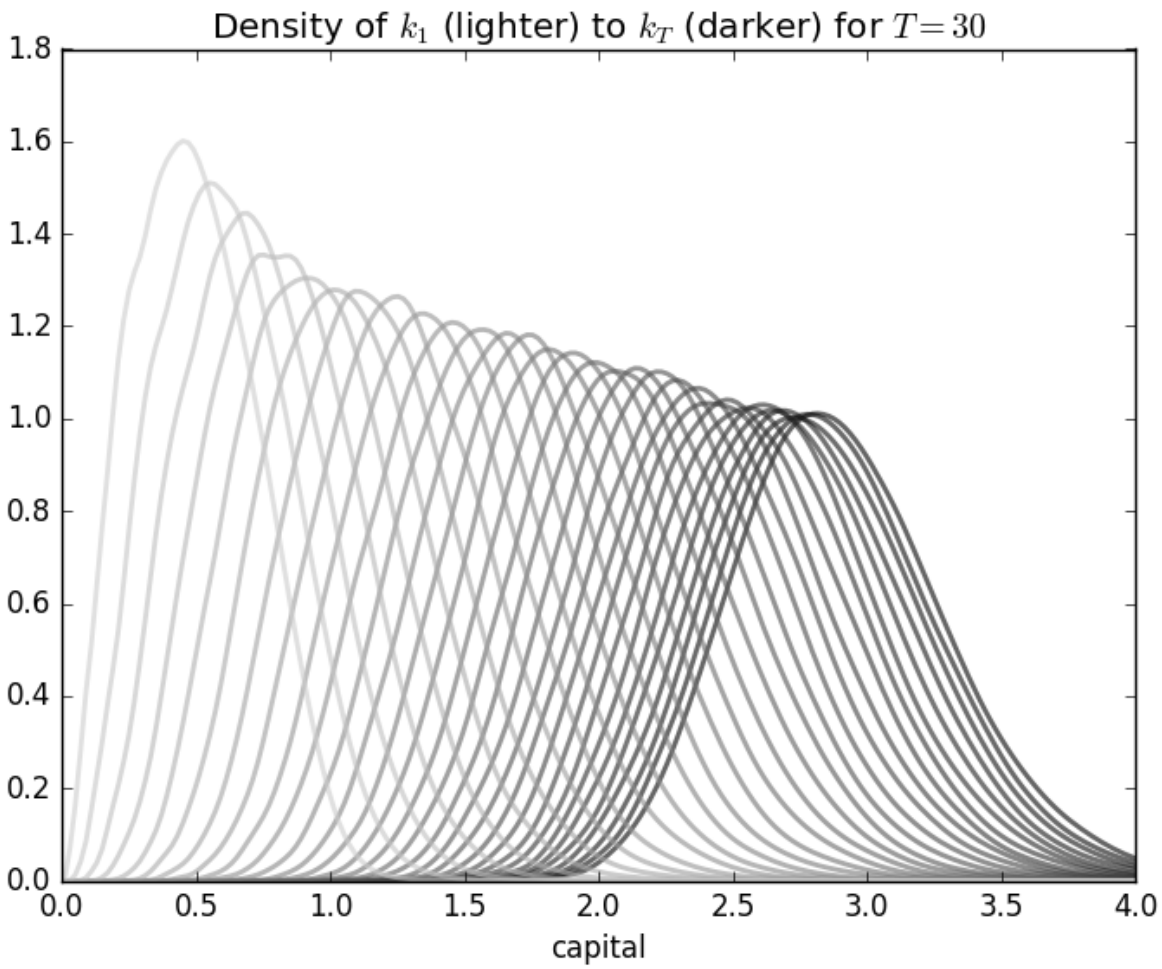
$$f(x;\xi,\sigma,\mu) = \begin{cases} \frac{1}{\sigma}\left[1+\left(\frac{x-\mu}{\sigma}\right)\xi\right]^{-1/\xi-1}\exp\left\{-\left[1+\left(\frac{x-\mu}{\sigma}\right)\xi\right]^{-1/\xi}\right\} & \text{for } \xi \neq 0 \\ \frac{1}{\sigma}\exp\left\{-\frac{x-\mu}{\sigma}\right\}\exp\left\{-\exp\left[-\frac{x-\mu}{\sigma}\right]\right\} & \text{for } \xi = 0 \end{cases} \tag{32}$$

for

$$x \in \begin{cases} \left[\mu - \frac{\sigma}{\xi}, +\infty\right) & \text{for } \xi > 0 \\ (-\infty, +\infty) & \text{for } \xi = 0 \\ \left(-\infty, \mu - \frac{\sigma}{\xi}\right] & \text{for } \xi < 0 \end{cases} \tag{33}$$

known in the literature as the Generalized extreme value distribution. Unfortunately, this distribution throws an error.

Figure 3: Look Ahead Estimate



When trying to test the pareto distribution:

$$f(x;\alpha,\theta) = \frac{\alpha\theta^{\alpha}}{x^{\alpha+1}}, \quad x \geq \theta \tag{34}$$

with parameters (3,2), Julia kernel dies.

I have not enough information for the reasons why not all initial distributions converge. I have not tested whether restricting the range will solve this problem or whether there is a bug in the implementations of the distributions.

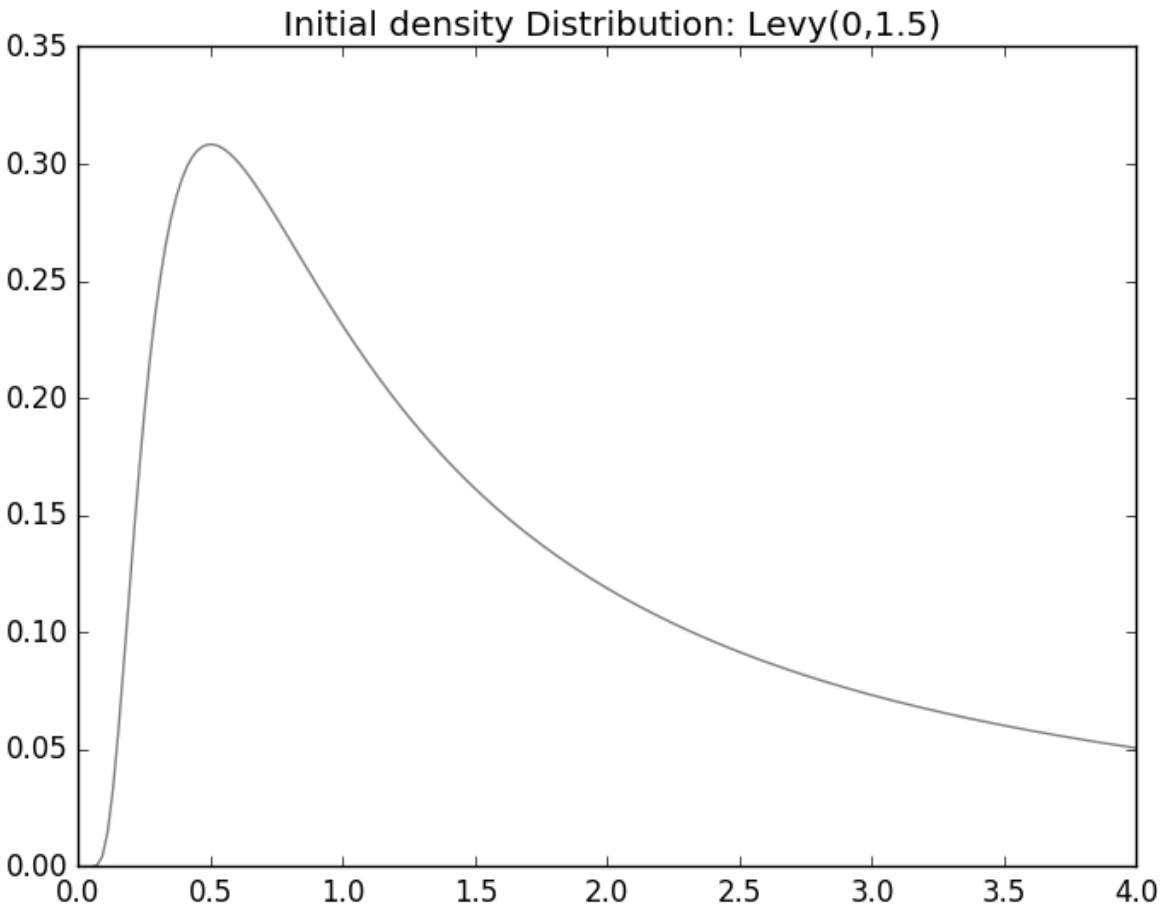
On the other hand, the most popular distributions do work. For example the Levy distribution.

$$f(x;\mu,\sigma) = \sqrt{\frac{\sigma}{2\pi(x-\mu)^3}}\exp\left(-\frac{\sigma}{2(x-\mu)}\right), \quad x > \mu \tag{35}$$

after 30 iterations of look ahead estimate with the same parameters as before:

```
s = 0.2 # Savings Rate
\delta = 0.1 # Capital Depreciation Rate
```

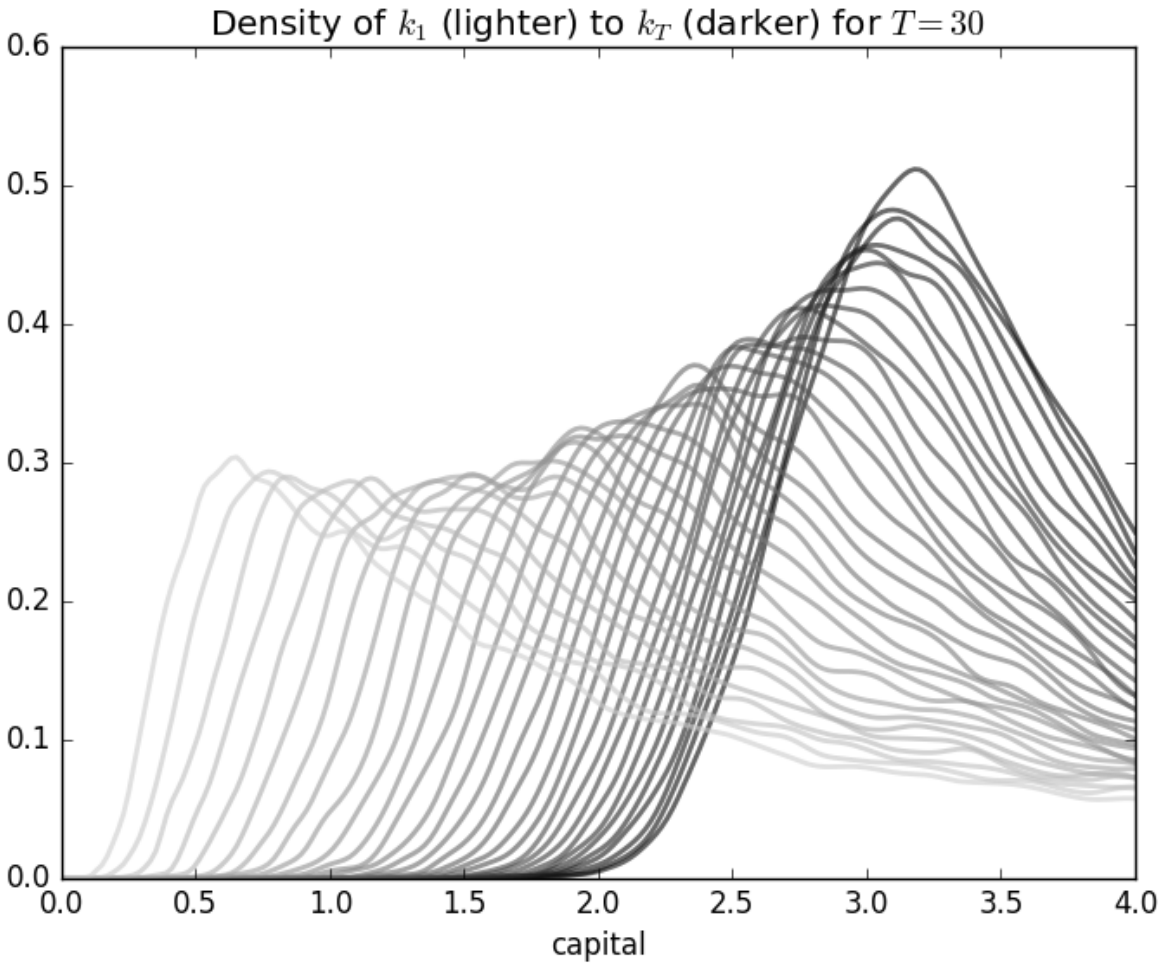
Figure 4: Levy Distribution




```
a_\sigma = 0.4 # A = exp(B) where B ~ N(0, a_sigma)
\alpha = 0.4 # We set f(k) = k**alpha
\psi_0 = Levy(0,1.5) # Initial Continuous State distribution
\phi = LogNormal(0.0, a_\sigma)
```

The conclusion about Continuous State Markov chains is that they are powerful tools to expend point estimates of economic aggregates. As we can see in our example, not only that capital per capita can have a wide range as a function of modest technology shocks, we cannot expect that with time the variance of our distribution will decrease as $t \to \infty$. In the case of the Levy distribution, we can see that the range the capital per effective capita, as seen in figure 5 can increase.

Figure 5: Levy Distribution Convergence

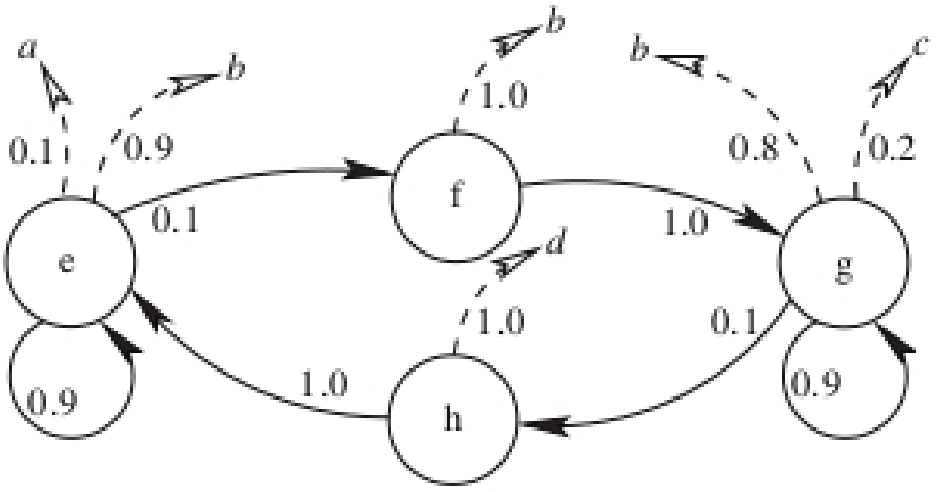


3 The Hidden Markov Model and Latent Parameters Estimation

A great introduction into the workings of the Hidden Markov processes was presented by Rabiner (1989). In a nutshell, a HMM is defined by a stochastic matrix $\mathbb{A}$ that changes the the states s_i of the system according to some probability, where $s_i \in S$ and S is the set of all possible States. These states are not observable. Each state can emit some observable outcomes with its own probability distribution which we will summarize in the emission matrix $\mathbb{B}$. We will take the example given by (Fraser, 2008, pp.9).

As we can see in figure 6, we have an oriented graph which can easily be transposed to a HMM model.

Figure 6: Hidden Markov Model from (Fraser, 2008, pp.9)



$$\mathbb{A} = \begin{array}{c} \\ e \\ f \\ g \\ h \end{array} \begin{pmatrix} 0.9 & 0.1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.9 & 0.1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The emission probabilities for the events $\Sigma = \{a, b, c, d\}$, as specified in the 2:

$$\mathbb{B} = \begin{array}{c} \\ e \\ f \\ g \\ h \end{array} \begin{pmatrix} 0.1 & 0.9 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Lastly, to fully define a HMM, we would either need a starting point, or an initial probability distribution among the states usually denoted by π.

Having defined a hidden Markov model, we can endeavour to find answers to the following questions:

1. When was the last recession?
2. Are we still in the recession?
3. What is the probability for the economy to follow a particular path of states?
4. What is the unconditional probability of observing certain outcome e_i or what is the probability of being in a particular state s_i e $Pr(S = s_i|e_i)$?

We conclude that a Hidden Markov Model extends the class of the Markov Chain models, since any order of a Markov model can be represented as an HMM. We will denote a Hidden Markov Model by $\lambda(\mathbb{A}, \mathbb{B}, \pi)$.

Simulating a Hidden Markov Model

In this example we will generate a sequence of a balanced and an unbalanced coin that follows an HMM ($\lambda(\mathbb{A}, \mathbb{B}, \pi)$) in the Python programming language using the ghmm package. For installing the ghmm library please refer to the Annex and http://ghmm.org.

$$\mathbb{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix} \tag{36}$$

```
import ghmm
A = [[0.9, 0.1], [0.2, 0.8]]
efair = [1.0 / 2] * 2
eloaded = [0.15, 0.85]
sigma = ghmm.IntegerRange(0,2)
B = [efair, eloaded]
pi = [0.5]*2
m = ghmm.HMMFromMatrices(sigma, ghmm.DiscreteDistribution(sigma), A,
   B, pi)
```

invoking the print method for the HMM model m

```
print m

DiscreteEmissionHMM(N=2, M=2)
  state 0 (initial=0.50)
    Emissions: 0.50, 0.50
    Transitions: ->0 (0.90), ->1 (0.10)
  state 1 (initial=0.50)
    Emissions: 0.15, 0.85
    Transitions: ->0 (0.20), ->1 (0.80)
```

generating a sample of 40 observations and printing them.

```
obs_seq = m.sampleSingle(40)
```

```
obs = map(sigma.external, obs_seq)

print obs
```

```
[1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
   1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0]
```

For a dice simulation generated by an HMM, refer to appendix D.

A formal introduction to an HMM

Suppose we have identified a sequence $\mathcal{O} \in \mathbf{\Omega}$ that we assume is generated by an HMM: $\lambda(\mathbb{A}, \mathbb{B}, \pi)$ as defined in chapter 3. Given λ, we would like to know:

$$Pr(\mathcal{O}|\lambda) = ?$$

A direct approach is that given the parameters of the HMM, we can compute the probability of observing a particular observation on a given chain of states:

$$Pr\left(o_1, o_2, ..., o_T | x_1, x_2, ..., x_k, ...x_T, \lambda\right) = \pi_i b_i(o_1) \prod_{t=2}^{t=T} a_{t-1,t} b_{a_t}(o_t) \tag{37}$$

And then we could sum the probabilities from equation 37 over all possible states X:

$$Pr\left(\mathcal{O}|\lambda\right) = \sum_X \pi_i b_i(o_1) \prod_{t=2}^{t=T} a_{t-1,t} b_{a_t}(o_t) Pr(X|\lambda) \tag{38}$$

Therefore, the direct approach is very difficult to assess. The difficulty of this problem consists in the fact that the total number of possibilities of sequences of states that can generate $\mathcal{O}$ is exponential in the number of observations T and would require $\mathcal{O}(N^T)$ operations, where N is the total number of states. At a first glance this might not appear to be a particular big issue in Economics: consider analysing yearly aggregates of a span of 10 years with 2 states $\Omega = \{Recession, Growth\}$. If, however, we would increase the number of states to 3 and use quarterly data, finding the maximum would require years of computation - clearly not feasible for any practical purpose.[37] A better approach to calculate the unconditional probability $Pr(\mathcal{O}|\lambda)$ is the forward/backward algorithm which is a class of dynamic programming algorithms and takes

[37] For a 2 state economy over a span of 10 years, we would call our function 1024 times, in contrast, a 3 state economy on quarterly data would require 12157665459056928801 ($1.22 * 10^{19}$) calls.

advantage of the assumptions of the Markovian processes to filter all possible combinations of states X.

Secondly, another question of interest is to find the most likely sequence of states from X given $\mathcal{O}$, e.i. the states that generated the highest probability $Pr(\mathcal{O}|X, \lambda)$.

Finally, the question that sparked my interest in studying Markovian processes is how do we estimate the parameters of an HMM, in particular $\mathbb{A}$ and $\mathbb{B}$. Unfortunately, this is still an unsolved problem in Mathematics and requires numerical methods. Once the matrix $\mathbb{A}$ is determined we can use the dynamic programming and combinatorial methods proposed by Lozovanu and Pickl (2015) for determining the state-time probabilities and the matrix of limiting probabilities. These methods can be directly applied to refine not only the point estimates of long-term economic growth at the country level, as defined by the International Monetary Fond, but also update the concept of long term growth as an integral part of limiting probabilities of a HMM.

To summarize, we could enumerate the key issues we would have to address and solve efficiently for an HMM model to be useful in practice. As expressed by Fraser (2008) and Hamilton and Raj (2002):

1. Evaluation - e.i. find the probability of an observed sequence given an HMM (relatively easy to do in practice)

2. Decoding - find the sequence of Hidden States that most likely generated the observation. That is: find the highest probability of a sequence. (until the Viterbi algorithm it was possible only theoretically)

3. Learning - Generate the best possible HMM for the observed outcome (really difficult - there are no analytic solutions to this problem. Use Baum-Welch algorithm)

The Forward/Backward Algorithm

Given an HMM$(\mathbb{A}, \mathbb{B}, \pi)$, as introduced in section 3 which we will denote by $\lambda(\mathbb{A}, \mathbb{B}, \pi)$, where $\mathbb{A}$ is the state transition probability e.i. $a_{ij} = a[i][j] = Pr(x_{t+1} = j|x_t = i)$ and $\mathbb{B}$ is the emissions probability, e.i. $b_i(k) = Pr(o_t = k|x_t = i)$ [38], and π is the initial probability distribution of states at $t = 1$, e.i. $\pi_i = Pr(x_1 = i)$, where $i \in \{1..N\}$, the objective of the forward/backward algorithm is to compute the probability of being in a particular state x_t at time t, given a sequence

[38] usually denoted by $\varepsilon_i(k)$ in the literature

of observations $\mathcal{O}$ e.i.:

$$Pr(x_k|\mathcal{O}) =?, k \in \{1..T\}$$

We can use Bayesian rule[39] to write $Pr(x_k|\mathcal{O}, \lambda)$ as

$$Pr(x_k|O, \lambda) = \frac{Pr(x_k, \mathcal{O}|\lambda)}{Pr(\mathcal{O}|\lambda)} = \frac{Pr(x_k, o_1, o_2, ...o_k|\lambda) * Pr(o_{k+1}, ..., o_T|x_k, o_1, ...o_k, \lambda)}{Pr(\mathcal{O}|\lambda)} \quad (39)$$

Using the Markovian property in the second half of 39[40] :

$$Pr(x_k|\mathcal{O}, \lambda) = \frac{Pr(x_k, o_1, o_2, ...o_k|\lambda)Pr(o_{k+1}, ..., o_T|x_k, \lambda)}{Pr(\mathcal{O}|\lambda)} \quad (40)$$

The first part in the numerator of equation 40, $Pr(x_k, o_1, o_2, ...o_k|\lambda)$ is called the forward part and $Pr(o_{k+1}, ..., o_T|x_k, \lambda)$ the backward part.

Once we can find an algorithm to compute the forward and the backward part, we can compute $Pr(x_k|O, \lambda)$ which enables us to answer the following questions: [41]

1. The Probability of being in a transition: $Pr(x_k \neq x_{k-1}|\mathcal{O})$
2. Helps augment the numerical methods in estimating the parameters of λ
3. Make samples on $x_k|\mathcal{O}$

Forward Algorithm

Given an HMM as presented in section 3: $\lambda(\mathbb{A}, \mathbb{B}, \pi)$, where $\mathbb{A}$ is the state transition probability e.i. $a_{ij} = a[i][j] = Pr(x_{t+1} = j|x_t = i)$ and $\mathbb{B}$ is the emissions probability, e.i. $b_i(k) = Pr(o_t = k|x_t = i)$ [42], and π is the initial probability distribution of states at $t = 1$, e.i. $\pi_i = Pr(x_1 = i)$, the objective of the forward algorithm is to compute

$$\alpha_i(k) = \alpha\left(x_k = i\right) = Pr(o_1, o_2, ...o_k, x_k|\lambda) \qquad o_k \in \mathcal{O}, k = \overline{1, T} \quad (41)$$

[43]

[39] In the appendix I will present the Bayesian rules in probability

[40] Since the current outcome depends on the current state and not on past outcomes $Pr(o_k|x_k, o_{k-1}, o_{k-2}...) = Pr(o_k|x_k)$. In practice, it is reasonable to assume that the expected outcome of a function (the growth of Economy) depends on the intrinsic values that define the system, rather than past outcomes.

[41] Mathematicalmonk channel present a gentle introduction to HMM and builds intuition what types of questions we can answer https://www.youtube.com/watch?v=7zDARfKVm7s&list=PLD0F06AA0D2E8FFBA

[42] usually denoted by $\varepsilon_i(k)$ in the literature

[43] I follow the notation of: http://personal.ee.surrey.ac.uk/Personal/P.Jackson/tutorial/hmm_tut2.pdf

The forward algorithm is also known as the *filtering* algorithm as it uses the available information up to the point of $o_k \in \mathcal{O}$. Again, computing $Pr(o_1, o_2, ...o_k, x_k|\lambda)$ directly would require a computation time exponential on T and would not be feasible for practical purposes. On the other hand, we can use the *Law of Total Probability*[44] and the Markov property to express:

If $k = 1$, from the definition of λ:

$$\alpha(x_k) = \pi_{x_k}$$

else:

$$\alpha_i(x_k) = \sum_{x_{k-1}} Pr(o_1, o_2, ...o_k, x_{k-1}, x_k|\lambda) \tag{42}$$

$$= \sum_{x_{k-1}} Pr(o_k|o_1, o_2, ...o_{k-1}, x_{k-1}, x_k, \lambda)\times \tag{43}$$

$$Pr(x_k|o_1, o_2, ...o_{k-1}, x_{k-1}, \lambda)Pr(o_1, o_2, ...o_{k-1}, x_{k-1}|\lambda) \tag{44}$$

$$= \sum_{x_{k-1}} Pr(o_k|x_k, \lambda)Pr(x_k|x_{k-1}, \lambda)\alpha(x_{k-1}) \tag{45}$$

$$= \sum_{x_{k-1}} \mathbf{b}_{x_k}(o_k)\mathbf{a}_{(k-1,k)}\alpha(x_{k-1}) \tag{46}$$

$$= \mathbf{b}_{x_k}(o_k) \sum_{x_{k-1}} \mathbf{a}_{(k-1,k)}\alpha(x_{k-1}), \quad x_{k-1} = \overline{1, N}, k = \overline{2, T} \tag{47}$$

Therefore, we can calculate the probability of observing a particular state x at time t with a cost of $\mathcal{O}(N^2T)$ operations using this recursive algorithm which is linear in T, rather than exponential for the naive approach.

Once we have a procedure to efficiently calculate $\alpha_i(k)$ we can also express the unconditional probability of observing $\mathcal{O}$ as the sum of all $\alpha_i(T)$ over all states.

$$Pr(\mathcal{O}|\lambda) = \sum_{i=1}^{N} \alpha_i(x_T) \tag{48}$$

Backward Algorithm

The backward algorithm is the second part of the forward/backward algorithm in equation 40.

As in subsection 3 we assume that an HMM as presented in section 3: $\lambda(\mathbb{A}, \mathbb{B}, \pi)$ is given, where $\mathbb{A}$ is the state transition probability and $\mathbb{B}$ is the emission probability matrix, e.i. $b_i(k) =$

[44] I used the steps of the algorithm from Wikipedia, https://en.wikipedia.org/wiki/Forward_algorithm except that I corrected for mistakes. For example: Wiki says it uses chain rule, when they meant the law of total probability.

$Pr(o_t = k|x_t = i)$ and π is the initial state probability distribution, the objective of the backward algorithm is to compute:

$$\beta_i(k) = \beta(x_k = i) = Pr(o_{k+1}, o_{k+2}, ..., o_T|x_k = i, \lambda) \quad i = \overline{1, N}, k = \overline{1, T} \tag{49}$$

Again, to get the recursive approach we will make use of the law of total probability to write:

$$\beta(x_k) = Pr(o_{k+1}, o_{k+2}, ..., o_T|x_k = i, \lambda) \tag{50}$$

$$= \sum_{x_{k+1}} Pr(o_{k+1}, o_{k+2}, ..., o_T, x_{k+1}|x_k = i, \lambda) \tag{51}$$

Now we can divide se sequence of observables and isolate the o_{k+1} observation as follows:

$$\beta(x_k) = \sum_{x_{k+1}} Pr(o_{k+2}, ..., o_T|x_{k+1}, x_k = i, o_{k+1}, \lambda) \times \tag{52}$$

$$Pr(o_{k+1}|x_{k+1}, x_k = i, \lambda) \times Pr(x_{k+1}|x_k = i, \lambda) \tag{53}$$

Now using the Markovian property and the fact that $Pr(o_{k+1}|x_{k+1}, x_k = i, \lambda) = Pr(o_{k+1}|x_{k+1}, \lambda)$

$$\beta(x_k) = \sum_{x_{k+1}} \beta(x_{k+1}) \mathbf{b}_{k+1}(o_{k+1}) \mathbf{a}_{k,k+1}, \qquad k = \overline{1, T-1} \tag{54}$$

The trick in the backward algorithm is that as shown above, k takes values from 1 to $T - 1$. For $k = T$, we define:

$$\beta(x_T) = 1$$

The intuition for this is clear when applying these algorithms to tagging problems in the Natural Language Programming, since we can define the state of the end of a sentence to be a special symbol "STOP". And since every sentence ends with this special symbol, the probability of getting $x_{T+1} = STOP$ equals 1.

The Viterbi Algorithm

Given an HMM model $\lambda(\mathbb{A}, \mathbb{B}, \pi)$ and an observable sequence $\mathcal{O} = \{o_1, o_2, ..., o_T\}$, we want to find the sequence $X = \{x_1, x_2, ..., x_T\}$ that maximizes the probability in equation 37:

$$Pr\left(\mathcal{O}, x_1, x_2, ...x_T, |\lambda\right) = \underset{X}{\operatorname{argmax}}\, \pi_i b_i(o_1) \prod_{t=2}^{t=T} a_{t-1,t} b_{a_t}(o_t) \tag{55}$$

In a nutshell, the Viterbi algorithm examines at each step all the possibilities of getting to that particular state and retains only the most likely path to it, thus eliminating all other possibilities.[45]

For the first step, the probability of being in state x_k at time $t = 1$ and observing o_1 is simply an updated version of π_k.

$$Pr(x_1 = k|o_1, \lambda) = \frac{Pr(x_1 = k, o_1)}{\sum_{x_1 \in S} Pr(x_1, o_1)} = \frac{\pi_{x_1} b_{x_1}(o_1)}{\sum_{x_i \in S} \pi_{x_i} b_{x_i}(o_1)}$$

where $s \in S$ and $S = \{1..N\}$ the set of all possible states. But we can still reduce the complexity of this formula by dropping the denominator, since maximizing the probability depends only on the numerator part, therefore:

$$V_{1,k} = \pi_{x_k} b_{x_k}(o_1) \tag{56}$$

For the next iterations:

$$V_{t,k} = \max_{x_{t-1} \in S} \left(V_{t-1,x_{t-1}} \mathbf{a}_{x_{t-1},k} \mathbf{b}_k(o_t) \right) \tag{57}$$

$$\mathbf{V}_t = \{V_{t,k} | k \in S\}$$

We denote the sequence of $X = \{x_1, x_2, ..., x_T\}$ that generates V_T:

$$\mathbf{x}_T = \arg\max_{x \in S} \mathbf{V}_{T,x}$$

The algorithm's complexity is $\mathcal{O}(N^2 * T)$ which is linear in T.

The EM algorithm

The Expectation Maximization algorithm was describe in the paper of A. P. Dempster (1977). The Baum-Welch algorithm extends the class of the EM algorithm and so we will focus on Baum-Welch instead.

The Baum-Welch Algorithm

Given an observable sequence $\mathcal{O} = \{o_1, o_2, ..., o_T\}$ and assuming this sequence was generated by an HMM model $\lambda(\mathbb{A}, \mathbb{B}, \pi)$ as define in section 3, we want to find out the most likely set of parameters of λ that generated the sequence $\mathcal{O}$. Denoting by $\theta = \{\mathbb{A}, \mathbb{B}, \pi\}$ the set of parameters of λ, we want to find out θ that maximizes the probability:

$$\theta^\star = \arg\max_{\theta} Pr\left(\mathcal{O}|\lambda(\theta)\right)$$

[45] Simple as it may sound, according to David Forney

The Baum-Welch algorithm uses as the basis the EM algorithm, which in turn is very similar to the k-means algorithm.

The first step is to infer the number of states and initial parameters of the $\lambda(\mathbb{A}, \mathbb{B}, \pi)$ using heuristic methods.

Secondly, using the forward/backward algorithm as described in section 3 from page 39 we find the updated Bayesian probability of observing o_t at time t in state x_i :

$$Pr\left(x_t = i|\mathcal{O}, \lambda(\theta)\right) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j\in S}\alpha_j(t)\beta_j(t)} \tag{58}$$

we will denote equation 58 by:

$$\gamma_i(t) = Pr\left(x_t = i|\mathcal{O}, \lambda(\theta)\right) \tag{59}$$

please note that in equation 58 the denominator $\sum_{j\in S}\alpha_j(t)\beta_j(t)$ was used rather that a computationally more efficient $\sum_{i\in S}\alpha_i(x_T) = Pr(\mathcal{O}|\lambda(\theta))$ in order to make $\gamma_i(t)$ a probability distribution.

Also, using the same backward and forward procedures as described in equation 49 and 41 we can also compute the probability of switching from state $x_k = i$ to $x_{k+1} = j$ given a particular model $\lambda(\theta)$:

$$Pr(X_t = i, X_{t+1} = j|\mathcal{O}, \lambda) = \frac{\alpha_i(t)a_{ij}b_j(o_{t+1})\beta_j(t)}{Pr(\mathcal{O}|\lambda(\theta))} \tag{60}$$

we will denote equation 60 by

$$\xi_{ij}(t) = Pr(X_t = i, X_{t+1} = j|\mathcal{O}, \lambda) \tag{61}$$

Again, we need to make 60 a probability distribution, and even though it looks computationally attractive to rewrite the denominator as the probability of observing $\mathcal{O}$ either by the forward or backward algorithm, we cannot write 60, although wikipedia does[46], as:

$$\xi_{ij}(t) = \frac{\alpha_i(t)a_{ij}b_j(o_{t+1})\beta_j(t+1)}{\sum_{j\in S}\alpha_j(T)} \tag{62}$$

for $\xi_{ij}(t)$ to be a probability distribution:

$$\xi_{ij}(t) = \frac{\alpha_i(t)a_{ij}b_j(o_{t+1})\beta_j(t+1)}{\sum_{i\in S}\sum_{j\in S}\alpha_i(t)a_{ij}b_j(o_{t+1})\beta_j(t+1)} \tag{63}$$

Now recall that a_{ij} is the probability of moving to state j while already being in state i, e.i.

$$a_{ij} = Pr(x_{t+1} = j|x_t = i, \mathcal{O}, \lambda(\theta) \tag{64}$$

[46]https://en.wikipedia.org/wiki/Baum-Welch_algorithm

We can rewrite the above equation of a_{ij} by applying the Bayesian rule:

$$Pr(x_{t+1} = j | x_t = i, \mathcal{O}, \lambda(\theta) = \frac{Pr(x_{t+1} = j, x_t = i | \mathcal{O}, \lambda(\theta)}{Pr(x_t = i | \mathcal{O}, \lambda(\theta)} \tag{65}$$

Using equations 60 and 58 in equation 65 we can finally obtain the posterior distribution of $\bar{a_{ij}}$:

$$\bar{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{66}$$

equivalently:

$$\bar{a_{ij}} = \frac{\frac{Pr(x_{t+1} = j, x_t = i | \mathcal{O}, \lambda(\theta)}{Pr(x_t = i | \mathcal{O}, \lambda(\theta)}}{\frac{\alpha_i(t)\beta_i(t)}{\sum_{j \in S} \alpha_j(t)\beta_j(t)}} \tag{67}$$

substituting

$$\bar{a_{ij}} = \frac{\frac{\alpha_i(t) a_{ij} b_j(o_{t+1}) \beta_j(t+1)}{\sum_{i \in S} \sum_{j \in S} \alpha_i(t) a_{ij} b_j(o_{t+1}) \beta_j(t+1)}}{\frac{\alpha_i(t)\beta_i(t)}{\sum_{j \in S} \alpha_j(t)\beta_j(t)}} \tag{68}$$

now we have a formula for re-estimating the transition probabilities as a function of the backward and forward probabilities, both of which are linear in T.

To update the initial state probability distribution:

$$\bar{\pi_i} = \gamma_1(i) = \frac{\alpha_i(1)\beta_i(1)}{\sum_{i \in S} \alpha_i(1)\beta_i(1)} \tag{69}$$

For the emissions probability we have the expected number of the system being in state i and observing o_k :

$$\bar{b_i(o_k)} = \frac{\sum_{t \in T} \gamma_t(i) \mathbf{1}_{o_t = o_k}}{\sum_{t \in T} \gamma_t(i)} \tag{70}$$

Therefore, given a model λ , we can compute $\bar{lambda}(\bar{\mathbb{A}}, \bar{\mathbb{B}}, \bar{\pi})$ and using the results of the theorem provided by Baum and Eagon (1967), specifically:

Theorem 3.1. *Let $P(x) = P(\{x_{ij}\})$ be a polynomial with non-negative coefficients homogeneous of degree d. If $x_{ij} \geq 0$ and $\sum_j x_{ij} = 1$. Denoting:*

$$\mathfrak{I}(x)_{ij} = \frac{\left(x_{ij} \frac{\partial P}{\partial x_{ij}} \Big|_{(x)} \right)}{\left(\sum_j x_{ij} \frac{\partial P}{\partial x_{ij}} \right)} \tag{71}$$

then $\mathbb{P}(\mathfrak{I}(x_{ij})) \geq \mathbb{P}(x_{ij})$

Proof. The proof is provided by Baum and Eagon (1967). □

we conclude that $\bar{\lambda}$ is a *better* HMM model than λ. By •better we mean that

$$\mathbb{P}(\mathcal{O}|\bar{\lambda} \geq \mathbb{P}(\mathcal{O}|\lambda) \tag{72}$$

Therefore we have obtained an improved version of λ. Repeating this process iteratively we will converge to a local maximum. There is no guarantee however that this local maximum will also be a global maximum. Since the optimization surface of a Markov process can be extremely complex, what we can do is randomly assign initial probability distributions μ_i for $i \in \bar{1..n}$ and compare the local optima for each i, hoping we reached the highest value. Most importantly, these new estimated probabilities $\bar{A}, \bar{B}$ need to make sense, therefore the holistic and heuristic reviews of the most likely models λ_i need to be performed by experts in the field.

The implementation of the Baum Welch algorithm is at the heart of the open source HMM implementation[47].

To show for convenience purposes how the Baum Welch algorithm works in practice, we will use the data from the example provided by byu.edu[48], consider the nucleotide sequence $\{a, c, g, t\}$, assume there is an HMM model $\lambda(\mathbb{A}, \mathbb{B}, \pi)$ and the observations $\mathcal{O}$:

```
['g', 'c', 'c', 'g', 'g', 'c', 'g', 'c', 'g', 'c', 'g', 'c', 'c', 'g',
   'c', 'g', 'c', 'g', 'c', 'g', 'c', 'c', 'g', 'c', 'g', 'c', 'c',
   'c','t', 't', 't', 't', 't', 't', 'a', 't', 'a', 'a', 'a', 'a',
   't', 't', 't', 'a', 't', 'a', 't', 'a', 'a', 'a', 't', 'a', 't',
   't', 't', 't','g', 'c', 'c', 'g', 'g', 'c', 'g', 'c', 'g', 'c',
   'g', 'c', 'c', 'g', 'c', 'g', 'c', 'g', 'c', 'g', 'c', 'c', 'g',
   'c', 'g', 'c', 'c', 'c','t', 't', 't', 't', 't', 't', 'a', 't',
   'a', 'a', 'a', 'a', 't', 't', 't', 'a', 't', 'a', 't', 'a', 'a',
   'a', 't', 'a', 't', 't', 't', 't']
```

generated by two states $normal, island$. We would like to know what was the sequence of states that generated these observations and would also like to see the most likelihood matrix $\mathbb{A}$ from the λ.

The software implementation is provided in the Annex D.

As we see we have obtained $\bar{\lambda}$ with updated transition probabilities.

[47] see: http://ghmm.org/

[48] link: http://dna.cs.byu.edu/bio465/Labs/hmmtut.shtml

4 Conclusions

As we have seen in the first part of this thesis Markov Chains have a wide area of applicability in modelling topics in Economics and Finance. Moreover, they provide a fresh paradigm for viewing traditional concepts in Economics such as long term economic growth or stock returns in Finance. The extension of the discrete state Markov chain to continuous states allows us to see the dynamics of the very probability distribution of the variable of interest rather than point estimates. Also, we have seen that the initial distribution assumptions play a crucial role in the variance of subsequent distributions and most importantly, the dispersion not only remains persistent after long periods of time (30 cycles in our case) but it can also increase casting doubt about concepts such as long run equilibrium *steady states*. On the other hand, Markov Chains are sensitive to how we define and what we incorporate in such abstract concepts as *states*. Even if we assume that the economy is governed by distinct economic states, it is still a matter of opinion when deciding the number of states. Nevertheless, I avail myself to conclude that the advantages of augmenting a model even with a parsimonious 2 state Markov Model can be expected to yield significant improvements.

A more general model of the Markov Chains that we have discussed in section 3, suitable with latent states or imperfect information, is the Hidden Markov Model (also called regime switching model in some sources). It not only allows better fit of traditional econometrics models such as ARMA and ARIMA, it does so using fewer parameters. The drawbacks of the HMM is its complexity and computational requirements in estimating the state transition probabilities. Fortunately, we have concluded that the use of dynamic programming techniques such as the Baum-Welch algorithm, extensively described in 3 partially solves these problems. The drawback of the Baum-Welch algorithm is that it does not guarantee a global optimum but only a local one and depending on the model, many random iterations of the initial probability distributions are needed to find the parameters of the HMM that generated the observations.

On future work I would like to continue on focusing on Hidden Markov Models and provide solutions to the problem of finding the global optimum. One way is to take advantage of computing parallelism or network distributed systems. This is possible since the bottleneck of estimating the transition probability matrix is in computing power and not in the network of the dimensions of the training set. Using mapreduce functions the central node can easily retain the highest likelihoods and discard others. Furthermore, I would like to further advance the HMM in the continuous state space and allow dimensionality reduction of external variables when defining abstract states of an HMM.

On a more philosophical note, we can conclude that history doesn't matter for the future if we know *the present*. Therefore, predicting future returns, especially growth rates, solely on past returns is the wrong path. But do we know *the present*?

A Replication

All script files can be found at my GitHub repository at https://github.com/moldovean/[49]. For any questions, please do not hesitate to contact me at adrian@vrabie.net

Installing the GHMM library

Unfortunately the GHMM cannot be installed using pip command, on the other hand, the installation instructions to build the GHMM package can be found at http://ghmm.org and are fairly easy. For convenience purposes I will list the terminal commands for Ubuntu (Debian Linux). First of all the ghmm package has some requirements:

```
sudo apt-get update
sudo apt-get install build-essential automake autoconf libtool
sudo apt-get install python-dev libxml++2.6-dev swig
```

Now create a new folder (ex. ghmm) and copy the source files from sourceforge[50]. Extract them in the newly created folder. Now we are ready to install the ghmm package for Python2.7+.

```
cd ghmm
sh autogen.sh
sudo ./configure
sudo make
sudo make install
sudo ldconfig
```

Check your installation. In Python:

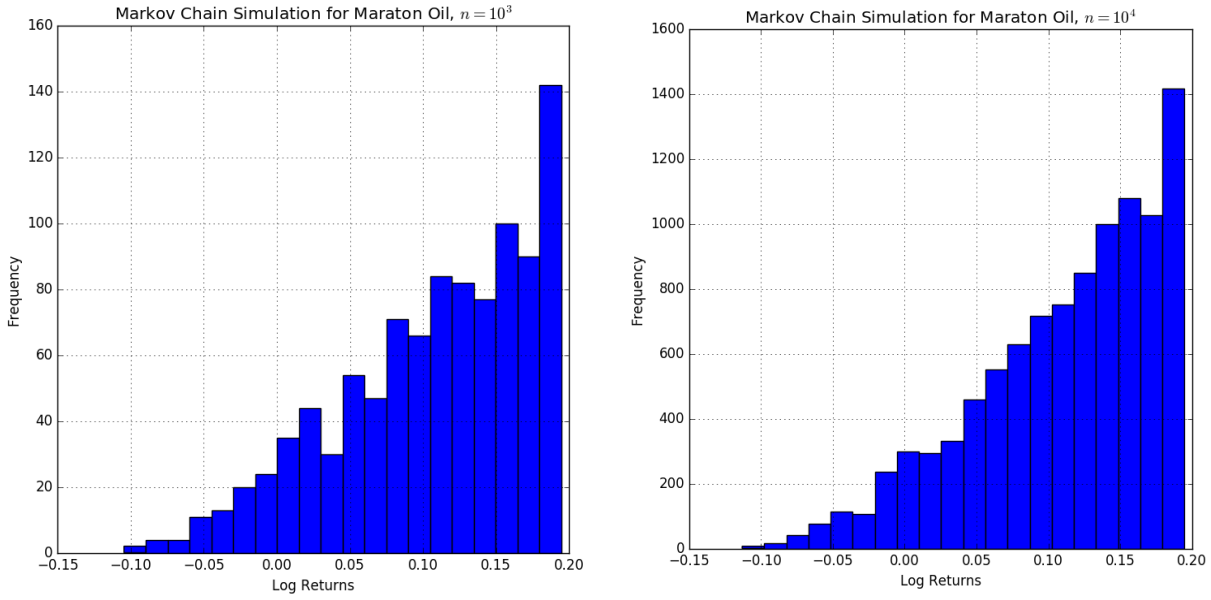
```
>> import ghmm
```

B Figures

As the simulation for the discrete state Markov Chain shows, the distribution of the returns stabilizes with $n = 10^4$ simulations. Using these histograms we can apply non-parametric methods

[49] https://github.com/moldovean/usm/tree/master/Thesis

[50] link: http://sourceforge.net/svn/?group_id=67094



(a) n=10^3

(b) n = 10^4

Figure 7: Markov Chain Simulation

to obtain the KDE or use the frequency approach to answer decision-making related questions. For example: *What is the probability that Maraton oil will lose money from investing in Kurdistan region of Iraq?*.

C Theoretical requirements

Bayesian Estimation

Bayesian estimation is simply a rearrangement of the conditional probability formula.

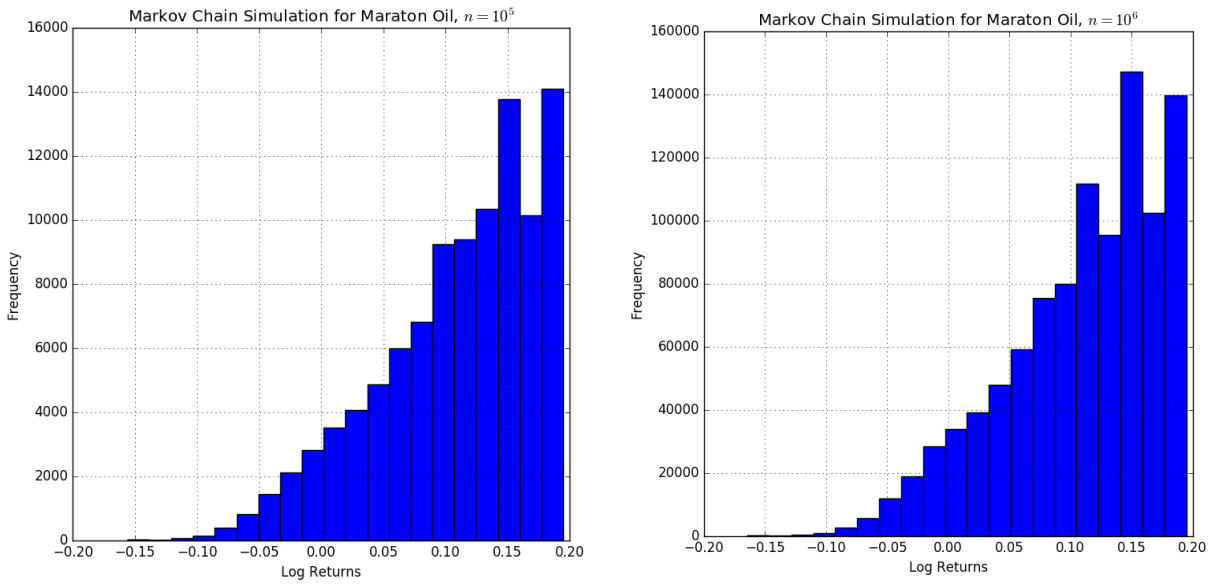
$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A,B)}{\mathbb{P}(B)} = \frac{\mathbb{P}(B|A) * \mathbb{P}(A)}{\mathbb{P}(B)} \tag{73}$$

At the moment there is a dire lack of intuitive written books in probability and statistics, see for example stackexchange.com question[51] For more information about Bayesian rule and applications of Bayesian estimation on the parameters of a probability distribution, I recommend http://psu.edu/[52]. Some people recommend http://www.statlect.com/[53] because they also provide accessible proofs

[51] link: http://stats.stackexchange.com/questions/70545/looking-for-a-good-and-complete-probability-and-statistics-book

[52] link: https://onlinecourses.science.psu.edu/stat414/node/241

[53] link: https://www.statlect.com/fundamentals-of-probability/Bayes-rule



(a) n=10^5 (b) n=10^6

Figure 8: Markov Chain Simulation

of elementary and less elementary facts that are difficult to find in probability and statistics books. Another popular site for statistics is http://stattrek.com/[54]

Eigenvalues and Eigenvectors

Let A be an $n \times n$ matrix and $\mathbf{x}$ be an $n \times 1$ vector. If the product $A\mathbf{x}$ points in the same direction as the vector $\mathbf{x}$, then $\mathbf{x}$ is an eigenvector of A. Eigenvalues and eigenvectors describe what happens when a matrix is multiplied by a vector. For a rigorous introduction to eigenvalues and eigenvectors consider the open MIT course Strang (2011). For an interactive learning and building a sense of feel of what eigenvectors and eigenvalues are, consider http://setosa.io[55] which uses the http://threejs.org/ visualisation library.

D Code Implementation

Look Ahead Estimate Implementation

This code was implemented by Spencer Lyon and referenced by (Stachurski and Martin, 2008) in the Stachurski and Sargent. (2016) blog. It implements a class for the purpose of simulating

[54]link: http://stattrek.com/probability/bayes-theorem.aspx

[55]link: http://setosa.io/ev/eigenvectors-and-eigenvalues/

a continuous state Markov process. The implementation leveraged on the paradigm of object oriented programming and creates a class `LAE` (Look Ahead Estimate) which takes the stochastic kernel p and the observations vector $n \times 1$ vector.

```
#=
Creats and Object which will be used to compute a sequence of marginal
   densities for a continuous state space Markov chain where the
   transition probabilities can be represented as densities rather
   than a discrete distribution.
@author : Spencer Lyon <spencer.lyon@nyu.edu> @date: 2014-08-01
@modified: Adrian Vrabie <adrian@vrabie.net> @date: 2016-05-17
References: http://quant-econ.net/jl/stationary_densities.html
=#

"""
A look ahead estimator associated with a given stochastic kernel p and
   a vector
of observations X.

##### Fields
- `p::Function`: The stochastic kernel. Signature is `p(x, y)` and it
   should be
vectorized in both inputs
- `X::Matrix`: A vector containing observations. Note that this can be
   passed as
any kind of `AbstractArray` and will be coerced into an `n x 1` vector.

"""
type LAE
   p::Function
   X::Matrix

   function LAE(p::Function, X::AbstractArray)
      n = length(X)
      new(p, reshape(X, n, 1))
```

```
    end
end
```

The function `lae_est` takes as input a `LAE` object and a vector or points y which in our case represent quantities of capital. Then the function simply calculates the average of the probability kernel applied to these points y and returns the average without dimensions (the `squeeze` function).

```
"""
A vectorized function that returns the value of the look ahead
    estimate at the
values in the array y.
##### Arguments
- `l::LAE`: Instance of `LAE` type
- `y::Array`: Array that becomes the `y` in `l.p(l.x, y)`
##### Returns
- `psi_vals::Vector`: Density at `(x, y)`
"""

function lae_est{T}(l::LAE, y::AbstractArray{T})
    k = length(y)
    v = l.p(l.X, reshape(y, 1, k))
    psi_vals = mean(v, 1)
    return squeeze(psi_vals, 1)
end
```

Viterbi Algorithm

The goal of this subsection is to create a parsimonious implementation of the Viterbi algorithm for demonstration purposes. I used the default Python 2.7 and IPython Notebook, but it should work in Python 3.+ as well, though I did not test it.

```
def viterbi(obs, states, start_p, trans_p, emit_p):
```

```
  //obs - Observations vector

   V = [{}]
   for s in states:
      V[0][s] = start_p[s]*emit_p[s][obs[0]]
   for t in range(1,len(obs)):
      V.append({})
      for s in states:
         V[t][s] = max(V[t-1][s_i]*trans_p[s_i][s]*emit_p[s][obs[t]]
            for s_i in states)
   return V
```

Baum Welch Algorithm

The implementation of the algorithm described in section 3 is being implemented by the GHMM project[56]. The project is ongoing and the main libraries are written in C but it comes with Python wrappers which makes the implementation of HMM much easier. The leader of GHMM project is Alexander Schliep[57].

Simulating an HMM

Here we will try to replicate the unfair dice problem using the GHMM library in Python programming language.

First we need to install the ghmm library for python 2.7+ from the http://ghmm.org.

Then using Ipython or Jupyter we create the HMM model m:

```
import ghmm
A = [[0.9, 0.1], [0.3, 0.7]]
efair = [1.0 / 6] * 6
eloaded = [3.0 / 13, 3.0 / 13, 2.0 / 13, 2.0 / 13, 2.0 / 13, 1.0 / 13]
sigma = ghmm.IntegerRange(1,7)
B = [efair, eloaded]
```

[56] http://ghmm.org

[57] http://www.cs.rutgers.edu/ schliep/index.html

```
pi = [0.5]*2
m = ghmm.HMMFromMatrices(sigma, ghmm.DiscreteDistribution(sigma), A,
    B, pi)

print(m)

DiscreteEmissionHMM(N=2, M=6)
  state 0 (initial=0.50)
    Emissions: 0.17, 0.17, 0.17, 0.17, 0.17, 0.17
    Transitions: ->0 (0.90), ->1 (0.10)
  state 1 (initial=0.50)
    Emissions: 0.23, 0.23, 0.15, 0.15, 0.15, 0.08
    Transitions: ->0 (0.30), ->1 (0.70)
```

Then we generate a sequence of observations and print them.

```
obs_seq = m.sampleSingle(30)
print obs_seq
sigma = ghmm.IntegerRange(1,7)
obs = map(sigma.external, obs_seq)

print obs
[2, 1, 2, 1, 6, 3, 3, 5, 6, 4, 1, 3, 4, 3, 1, 2, 3, 1, 6, 3, 5, 2, 4,
    5, 4, 1, 4, 2, 2, 6]
```

Conclusions: even though there is a hidden Markov model generating these sequences, they are impossible to distinguish to the naked eye.

Training on nucleotide data with two states

First we create the alphabet of possible sequences:

```
import ghmm
from ghmm import *
dna = ['a','c','t','g']
sigma = Alphabet(dna)
```

We then randomly pick a model that we will train.

```
A = [[0.9, 0.1], [0.3, 0.7]]
normal = [.25,.15,.35,.25]
island = [.25,.25,.25,.25]
B=[normal,island]
pi = [0.5] * 2
m=HMMFromMatrices(sigma,DiscreteDistribution(sigma),A,B,pi)
print m
obs_seq = m.sampleSingle(50)
print obs_seq


Out:
DiscreteEmissionHMM(N=2, M=4)
  state 0 (initial=0.50)
    Emissions: 0.25, 0.15, 0.35, 0.25
    Transitions: ->0 (0.90), ->1 (0.10)
  state 1 (initial=0.50)
    Emissions: 0.25, 0.25, 0.25, 0.25
    Transitions: ->0 (0.30), ->1 (0.70)

gtgcggggcggaaccgatcatggtcatccttgttgtctattactatgcaa
```

```
# Baum Welch algorithm to training
## Train Data
train_seq = EmissionSequence(sigma, ['g', 'c', 'c', 'g', 'g', 'c',
   'g', 'c', 'g', 'c', 'g', 'c', 'c', 'g', 'c', 'g', 'c', 'g', 'c',
   'g', 'c', 'c', 'g', 'c', 'g', 'c', 'c', 'c','t', 't', 't', 't',
   't', 't', 'a', 't', 'a', 'a', 'a', 'a', 't', 't', 't', 'a', 't',
   'a', 't', 'a', 'a', 'a', 't', 'a', 't', 't', 't', 't', 'g', 'c',
   'c', 'g', 'g', 'c', 'g', 'c', 'g', 'c', 'g', 'c', 'c', 'g', 'c',
   'g', 'c', 'g', 'c', 'g', 'c', 'c', 'g', 'c', 'g', 'c', 'c', 'c',
   't', 't', 't', 't', 't', 't', 'a', 't', 'a', 'a', 'a', 'a', 't',
```

```
    't', 't', 'a', 't', 'a', 't', 'a', 'a', 'a', 't', 'a', 't', 't',
    't', 't'])

m.baumWelch(train_seq)

print m

DiscreteEmissionHMM(N=2, M=4)
  state 0 (initial=0.00)
    Emissions: 0.39, 0.00, 0.61, 0.00
    Transitions: ->0 (0.98), ->1 (0.02)
  state 1 (initial=1.00)
    Emissions: 0.00, 0.57, 0.00, 0.43
    Transitions: ->0 (0.04), ->1 (0.96)


print m.viterbi(train_seq)
([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0], -89.38982033967802)
```

E Curiosities

Who was Andrey Markov?

Although your marginal benefit from what I know about Andrey Markov will not greatly exceed what you can already read from wikipedia[58], I will avail myself to mentioning that Andrey Markov was first of all a rebellious student, and in his academics he performed poorly. His past academic performances, in line with the Markovian property, don't matter.

[58] https://en.wikipedia.org/wiki/Andrey_Markov

Comparing programming languages for HMM implementation

There are significant gains to implementing the Baum-Welch algorithm in a compiled language when performing 10 EM iterations. Moreover, there are significant gains when comparing the results from the C language to the C++ with Armadilo library. See the academic blog on: Tulouse University It would also be interesting to compare Java versus C. It is not known to me if the implementation of C from Tulouse University takes advantages of parallelism.

References

A. P. Dempster, N. M. Laird, D. B. R. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38. http://www.jstor.org/stable/2984875.

Barbu, V. and Limnios, N. (2008). *Semi-Markov Chains and Hidden Semi-Markov Models toward Applications: Their Use in Reliability and DNA Analysis*. Lecture Notes in Statistics. Springer New York.

Baum, L. E. and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bull. Amer. Math. Soc.*, 73(3):360–363. http://projecteuclid.org/euclid.bams/1183528841.

Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Statist.*, 37(6):1554–1563.

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171.

Benesch, T. (2001). The baum–welch algorithm for parameter estimation of gaussian autoregressive mixture models. *Journal of Mathematical Sciences*, 105(6):2515–2518.

Fraser, A. (2008). *Hidden Markov Models and Dynamical Systems*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104). https://books.google.md/books?id=DMEWrB-2gGYC.

Hamilton, J. D. (2005). What's real about the business cycle? (11161).

Hamilton, J. D. (2016). Macroeconomic regimes and regime shifts. Technical report, National Bureau of Economic Research.

Hamilton, J. D. and Raj, B. (2002). *Advances in Markov-Switching Models*. Springer Science Business Media.

Häggström, O. (2002). *Finite Markov Chains and Algorithmic Applications (London Mathematical Society Student Texts)*. Cambridge University Press, 1 edition.

Langville, A. N. and Meyer, C. D. (2011). *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press.

Lin, D. and Stamp, M. (2011). Hunting for undetectable metamorphic viruses. *Journal in computer virology*, 7(3):201–214.

Lozovanu, D. and Pickl, S. (2015). *Optimization of Stochastic Discrete Systems and Control on Complex Networks*. Springer International Publishing.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. doi: 10.1109/5.18626.

Romer, D. (2006). *Advanced macroeconomics*. McGraw-Hill higher education. McGraw-Hill.

Stachurski, J. and Martin, V. (2008). Computing the distributions of economic models via simulation. *Econometrica*, 76(2):443–450.

Stachurski, J. and Sargent., T. J. (2016). Quant Econ quantative economics. http://quant-econ.net/.

Strang, G. (Fall 2011). 18.06sc linear algebra. http://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/index.htm.

Sundberg, R. (1972). *Maximum likelihood theory and applications for distributions generated when observing a function of an exponential variable*. PhD thesis, PhD Thesis. Institute of Mathematics and Statistics, Stockholm University, Stockholm.

Tauchen, G. (1986). Finite state markov-chain approximations to univariate and vector autoregressions. *Economics Letters*, 20(2):177–181.