Three Essays on Computational Approaches to Economics

Xunzhao Yin

Submitted to the graduate degree program in Department of Economics and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Committee members

Shigeru Iwata, Chairperson

Azadeh Jalali

John Keating

Xuemin Tu

Jianbo Zhang

Date defended: November 21, 2019

The Dissertation Committee for Xunzhao Yin certifies
that this is the approved version of the following dissertation :

Three Essays on Computational Approaches to Economics

Shigeru Iwata, Chairperson

Date approved: November 21, 2019

Abstract

The three essays included in this dissertation are on three different popular computational approaches that are widely applicable in economics. In Chapter 1, a state-space model is constructed which is linear in state variables and nonlinear in parameters. From the model, the time-varying level of natural interest rate is estimated using Kalman filter and Gibbs sampling algorithms. Chapter 2 proposes a new algorithm, called Implicit Particle Gibbs, to solve nonlinear state-space models. And Chapter 3 reviews recent development of deep learning and reinforcement learning algorithms and their applications in economics.

Acknowledgements

I cannot express enough thanks to my advisor and coauthor, Dr. Shigeru Iwata, and committee members, Dr. Jalali, Dr. Keating, Dr. Tu and Dr. Zhang, for their continued support and encouragement. I offer my sincere appreciation for the learning opportunities provided by my committee.

My completion of this project could not have been accomplished without the support of my family. My heart felt thanks.

Contents

List of Figures

List of Tables

Chapter 1

A Decline in the Natural Rate of Interest: Is it Real?

1.1 Introduction

The federal funds rate in the United States was at or near historic low in most of the years since the Great Recession. The persistence of low level of interest rates suggests that their long-run natural level is likely to have declined. The long-run level, known as the r^* or the *natural rate of interest* since Wicksell (1898), has taken the central stage in monetary policy discussions. Woodford (2003) defines r^* as the equilibrium real rate of return in an economy with full price flexibility. In empirical researches, most papers follow Laubach & Williams (2016) and define r^* as "the level of real interest rates expected to prevail, say, five to ten years in the future, after the economy has emerged from any cyclical fluctuations and is expanding at its trend rate." Monetary authorities focus on r^* for two reasons: not only does it provide a guidepost on how tight or loose the policy is, but also an persistently low r^* poses serious challenges on future ammunition of conventional policy tools.

Despite its merits, how useful the notional rate is in practice remains in question. r^* is the benchmark against which several popular policy rules are implemented. A policy rule cannot prescribe with confidence until r^* is pinned down. However, as a latent variable, r^* can only be inferred from observable data. Although many estimates have been proposed, policymakers still find difficulty in pinning down r^* with confidence. In the literature, estimates of r^* are usually reported with fairly large uncertainty band. The Laubach & Williams (2003) estimates have a standard error ranging from 97 basis points to 383 bp depending on specification, which translate into a 95% uncertainty band at least 400 bp wide. Most other researches also give estimates flanked by quite large uncertainty bands [1] Low precision hampers the use of r^* in policy discussion.

The literature has made many efforts to address the precision challenge, and most of these efforts have adopted a *direct* strategy. At the core of this strategy is an assessment of what accounts for the changes in r^*. Economic theory ties r^* to the growth rate of potential output and explains a lower level of r^* with a slower pace of potential growth. This connection is central to many researches including Laubach & Williams (2003), Holston et al. (2017) and Johannsen & Mertens (2018). However, since potential output is also unobservable, linking r^* to the potential growth is not likely to improve the estimates precision. More researches explain r^* with other variables. Hamilton et al. (2016) find evidence that the U.S. r^* is related with the long-run world real rate. Gagnon et al. (2016) demonstrate how demographic changes account for most of the decline in r^* since the 1980s. Summers (2014) discusses how savings and investment could have lowered r^*. Hakkio & Smith (2017) assume that the movement of r^* is partly explained by bond premiums. And Del Negro et al. (2017) find that the main drivers of the decline in r^* are rising premiums for safety and liquidity. Using different explaining factors, these researches produce various estimates of r^*, but none of them comes with satisfactory precision. This leads us to postulate that the movement of r^* is the result of a compound of factors and to conclude that the *direct* strategy is counterproductive for the purpose of estimating with precision.

In contrast to the *direct* strategy, we take an *indirect* approach. Instead of examining what accounts for r^*, we explore what r^* gives rise to. The ups and downs on the Treasury bonds market are partly engendered by economic variables including r^*. Following Johannsen & Mertens (2018), we decompose macroeconomic variables into long-run trends and short-run cyclical components. While short-run variables including the cyclical component of the federal funds rate impinge mainly on the short-end of the yield curve, long-run economic variables such as r^* are able to affect both ends. For that reason, we postulate the infomation

[1] For example, the 95% uncertainty bands of the Kiley (2015), Lubik & Matthes (2015) and Johannsen & Mertens (2018) estimates are respectively about 3%, 4.5% and 3.5% wide.

of r^* has been encoded into the *level* of the Treasury yield curve, and propose to decode with a macro-finance term-structure model. The dynamics of macroeconomic variables and Treasury yields constitute a state-space model wherein r^* together with other latent economic variables can be estimated using a Kalman filter.

Consistent with other researches, our r^* estimates exhibit a downward trend since the early 1990s. The decline accelerated since the beginning of the century. The uncertainty bands surrounding our estimates are considerably narrower than in most other researches. The decline in our r^* estimates is significant. We find the *level* of the Treasury yield curve exhibits a strong correlation with r^*, while the *slope* of the yield curve has a strong comovement with the federal funds rate gap.

1.2 The Model

Our whole model consists of two blocks: the macroeconomic block and the finance block. The macroeconomic block describes long-run and short-run dynamics of inflation, output and federal funds rate. The finance block connects macroeconomic variables with the Treasury yield curve rates. The two blocks together make up a state-space model, where the finance block provides extra measurement equations and reinforces the identification of latent state variables.

1.2.1 Macroeconomic Block

The macroeconomic variables are modeled in a fashion with an "independent trend/cycle decomposition". We assume that inflation, real GDP and effective federal funds rate vary around their respective trends. In general, for each of the three variables Y_t, we write

$$Y_t = Y_t^* + \tilde{Y}_t,$$

where Y_t^* denotes the trend; $\tilde{Y}_t$ denotes the gap between a variable from its trend. Similar to Johannsen & Mertens (2018), the defining feature of the cyclical component is that they are zero mean ergodic where

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \tilde{Y}_t \longrightarrow \mathrm{E}\left[\tilde{Y}_t\right] = 0.$$

As for the trend component, we take the long-run perspective as advocated by Laubach & Williams (2003) and define the trend as

$$Y_t^* \equiv \lim_{h\to\infty} \mathrm{E}_t\left[Y_{t+h}\right]. \tag{1.1}$$

Specifically,

1.2.1.1 Inflation

We decompose the inflation rate into a trend, a cyclical component and an error term

$$\pi_t = \pi_t^* + \tilde{\pi}_t + e_{\pi,t}, \tag{1.2}$$

where $e_{\pi,t} \sim \mathrm{N}(0, \sigma_\pi^2)$ denotes a one-off measurement error that does not have any impact on economic dynamics; the trend π_t^* is believed to be determined by long-run monetary factors such as the policymaker's inflation target. According to definition (1.1), π_t^* is a martingale . We assume a random walk process for π_t^*,

$$\pi_t^* = \pi_{t-1}^* + \varepsilon_{\pi^*,t}, \quad \varepsilon_{\pi^*,t} \sim \mathrm{N}(0, \sigma_{\pi^*}^2). \tag{1.3}$$

1.2.1.2 GDP Growth

Let X_t denote the U.S. real GDP and X_t^* denote its potential level.

Published each quarter by the Bureau of Economic Analysis, GDP growth rate g_t is, roughly speaking, computed by evaluating today's GDP against its level three months ago

$$g_t = \ln \frac{X_t}{X_{t-3}}.$$

By definition, real GDP deviates from its potential by the output gap $\tilde{x}_t$,

$$X_t = X_t^* \left(\tilde{x}_t + 1\right).$$

So g_t can be decomposed into two parts:

$$\ln \frac{X_t}{X_{t-3}} \approx \ln \left[\frac{X_t^*}{X_{t-3}^*}\right] + \ln \left(\frac{\tilde{x}_t + 1}{\tilde{x}_{t-3} + 1}\right).$$

We denote the growth rate of potential real GDP by g_t^*, and rewrite the above equation into

$$g_t = g_t^* + (\tilde{x}_t - \tilde{x}_{t-3}) + e_{g,t}, \tag{1.4}$$

where, again, we assume an error term $e_t \sim \mathrm{N}(0, \sigma_g^2)$ since $\ln(\tilde{x}_t + 1) \approx \tilde{x}_t$ for a small magnitude of $\tilde{x}_t$.

The growth rate of potential GDP is determined by exogenous variables such as technological progress or demographics. We assume it is a random walk process according to (1.1)

$$g_t^* = g_{t-1}^* + \varepsilon_{g^*,t}, \quad \varepsilon_{g^*,t} \sim \mathrm{N}(0, \sigma_{g^*}^2). \tag{1.5}$$

The output gap data are published by the Congressional Budget Office. We assume they are measured with noise in the manner that

$$\tilde{x}_t^{\mathrm{CBO}} = \tilde{x}_t + e_{\tilde{x},t}, \quad e_{\tilde{x},t} \sim \mathrm{N}(0, \sigma_{\tilde{x}}^2). \tag{1.6}$$

1.2.1.3 Federal Funds Rate

The federal funds rate is often described with a Taylor rule

$$i_t = r_t^* + \pi_t + \phi_\pi \left(\pi_t - \pi_t^*\right) + \phi_x \tilde{x}_t,$$

where r_t^* stands for the neural rate of real interest. Equivalently, we can rewrite the above equation into

$$\begin{aligned} i_t &= r_t^* + \pi_t^* + \left(\phi_\pi + 1\right)\left(\pi_t - \pi_t^*\right) + \phi_x \tilde{x}_t, \\ &\approx r_t^* + \pi_t^* + (\phi_\pi + 1)\tilde{\pi}_t + \phi_x \tilde{x}_t \end{aligned} \tag{1.7}$$

where $r_t^* + \pi_t^*$ is sometimes called the *neutral rate of nominal interest* and forms the benchmark of the policy rule. According to the Taylor rule, the gap from the benchmark, $i_t - r_t^* - \pi_t^*$, is chosen by policymakers from time to time based on the inflation gap $\tilde{\pi}_t$ and the output gap $\tilde{x}_t$. Denote the federal funds rate gap by $\tilde{i}_t$. Then we can write

$$i_t = r_t^* + \pi_t^* + \tilde{i}_t + e_{i,t}, \tag{1.8}$$

Note that a measurement error $e_{i,t} \sim \mathrm{N}(0, \sigma_i^2)$ is included here, although it might be small.

We assume r_t^* evolves according to a random walk process

$$r_t^* = r_{t-1}^* + \varepsilon_{r^*,t}, \quad \varepsilon_{r^*,t} \sim \mathrm{N}(0, \sigma_{r^*}^2). \tag{1.9}$$

1.2.1.4 Cyclical Components

In a basic New Keyesian model, the cyclical dynamics of economy are explained by three equations. The IS equation relates the federal funds rate gap $\tilde{i}_t$ with output gap $\tilde{x}_t$. The Phillips curve explains how output gap $\tilde{x}_t$ leads to the deviation $\tilde{\pi}_t$ of inflation from its trend. And a monetary policy rule such as (1.7) describes how fluctuations in output and inflation feed into monetary policy decisions $\tilde{i}_t$. Solved for rational expectation, a New Keynesian model admits a first-order VAR as its reduced form and, in practice, is usually fitted with quarterly or annual data. In order to work with a monthly data set, we generalize the New Keynesian idea into a VAR(p) model

$$\boldsymbol{\Phi}(L)\tilde{\boldsymbol{Y}}_t = \boldsymbol{\varepsilon}_{\mathrm{GAP},t}, \quad \boldsymbol{\varepsilon}_{\mathrm{GAP},t} \sim \mathrm{N}\left(\mathbf{0}, \boldsymbol{\Sigma}_{\mathrm{GAP}}\right). \tag{1.10}$$

where $\tilde{\boldsymbol{Y}}_t = \left(\tilde{\pi}_t, \tilde{x}_t, \tilde{i}_t\right)'$; the value of p is chosen to be four so that a policy action one quarter ago continues to impinge on today's economic activities.

All roots of the polynomial

$$\boldsymbol{\Phi}(z) = \boldsymbol{I} - \boldsymbol{\Phi}_1 z - \boldsymbol{\Phi}_2 z^2 \cdots$$

are assumed to lie outside of a unit circle so that the gaps $(\tilde{\pi}_t, \tilde{x}_t, \tilde{i}_t)'$ admit the value $\mathbf{0}$ as their unconditional mean. As a result, if free of any shocks, maintaining monetary policy at the natural level ($\tilde{i}_{t+h} = 0, h = 1, 2, \ldots$) would direct the economy toward its potential over time ($\lim_{h\to\infty} \tilde{x}_{t+h} \to 0$).

Note that under the above setup, we have

$$\lim_{h\to\infty} \mathrm{E}_t\left[\pi_{t+h}\right] = \pi_t^*,$$
$$\lim_{h\to\infty} \mathrm{E}_t\left[g_{t+h}\right] = g_t^*,$$
$$\lim_{h\to\infty} \mathrm{E}_t\left[i_{t+h}\right] = r_t^* + \pi_t^*,$$

as defined in (1.1). So the trends represent the market's expectation of each of the three headline data series in the far future.

$\varepsilon_{\pi^*,t}, \varepsilon_{g^*,t}, \varepsilon_{r^*,t}$ are each assumed to be i.i.d. and independent of short-run risks $\boldsymbol{\varepsilon}_{\mathrm{GAP},t}$ so that the forces which shape the long-run equilibrium are independent of the shocks which cause the business cycle.

1.2.1.5 r_t^* & g_t^*

In contrast to the economic theory, we do not assume a direct relation between the natural rate r_t^* and growth of potential output g_t^*. Some researchers including Hamilton et al. (2016) and Borio et al. (2018) find the link between these two variables empirically weak. Other researchers propose other factors that potentially contribute to the movement of r_t^* Summers (2014); Gagnon et al. (2016); Hakkio & Smith (2017). We remain skeptical of these explanations, and choose to keep maximum flexibility by assuming a random walk for either r_t^* or g_t^*.

1.2.1.6 State-Space Model

The macroeconomic block of our model constitutes a state-space model with six latent variables, $\{\tilde{\pi}_t, \tilde{x}_t, \tilde{i}_t, \pi_t^*, g_t^*, r_t^*\}$. Their dynamics are described by transition equations (1.3), (1.5), (1.9) and (1.10). They are related with four observed data series through measurement equations (1.2), (1.4), (1.6) and (1.8).

1.2.2 Finance Block

The finance block provides more measurement equations for the latent macroeconomic variables.

The yields of U.S. Treasury bonds are described in a no-arbitrage term-structure model. As advocated by Dai & Singleton (2000), the whole yield curve is assumed to be spanned by a small number of factors. Following the macro-finance literature, the factors are assumed to evolve under the influence of macroeconomic variables. The cross-sectional variations of the yields are regulated by no-arbitrage conditions.

1.2.2.1 Three Factors

One salient feature of the Treasury yield curve is that its cross-sectional variation is well explained by only a small number of factors. 99.92% of the variations of the nine yields in our data set are explained by their first three principal components - usually called the *level*, *slope* and *curvature*. We denote the three factors by

$$\boldsymbol{\mathcal{P}}_t \equiv (L_t, S_t, C_t)' .$$

1.2.2.2 Factors and the Economy

Following the macro-finance literature e.g. Creal & Wu (2017), we assume the macroeconomic variables have a contemporaneous impact on $\boldsymbol{\mathcal{P}}_t$

$$\boldsymbol{\mathcal{P}}_t = \boldsymbol{K}_{0\mathcal{P}} + \boldsymbol{K}_{1\mathcal{P}}\boldsymbol{\mathcal{P}}_{t-1} + \boldsymbol{\Psi}\boldsymbol{M}_t + \boldsymbol{\Sigma}_{\mathcal{P}}\boldsymbol{e}_{\mathcal{P},t}, \quad \boldsymbol{e}_{\mathcal{P},t} \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{1.11}$$

where $\boldsymbol{M}_t$ stands for the trends and gaps of the headline economic variables elaborated in section 1.2.1

$$\boldsymbol{M}_t \equiv \left(\tilde{\pi}, \tilde{x}_t, \tilde{i}_t, \pi_t^*, g_t^*, r_t^*\right)' ; \tag{1.12}$$

the coefficient matrix $\boldsymbol{\Psi}$ is specified in the following form

$$\boldsymbol{\Psi} = \begin{pmatrix} \psi_{11} & \psi_{12} & \psi_{13} & \psi_{14} & \psi_{15} & \psi_{16} \\ \psi_{21} & \psi_{22} & \psi_{23} & 0 & 0 & 0 \\ \psi_{31} & \psi_{32} & \psi_{33} & 0 & 0 & 0 \end{pmatrix}, \tag{1.13}$$

where the bottom-right block is assumed to be $\mathbf{0}$ so that the *slope* and *curvature* bear the imprints of only the gaps $(\tilde{\pi}_t, \tilde{x}_t, \tilde{i}_t)'$.

To see the reason, let $\mathcal{M}_{t+1}$ denote the stochastic discount factor of a representative agent, and

$$\mathfrak{m}_t \equiv \log \mathcal{M}_t.$$

Let $y_{m,t}$ denote the yield of a Treasury bond that matures in m months, $m = 1, 2, \ldots$ It can be shown that $y_{m,t}$ is determined by the expectation of the future path of 1-month Treasury yields, $y_{1,t+i}$, plus a term premium

$$y_{m,t} = \frac{1}{m}\mathrm{E}_t\left[\sum_{i=0}^{m-1} y_{1,t+i}\right] - \frac{1}{m}\sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right), \quad m = 1, 2, \ldots,$$

where the second term stands for the term premium. Historically 1-month Treasury yield and the federal funds rate tracked each other closely. If we assume these two are roughly the same,

$$y_{1,t} \approx i_t, \quad t = 1, 2, \ldots,$$

then, from (1.8), $y_{m,t}$ can be written in the following form

$$y_{m,t} = \pi_t^* + r_t^* + \frac{1}{m}\mathrm{E}_t\sum_{i=0}^{m-1} \tilde{i}_{t+1} - \frac{1}{m}\sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right), \quad m = 1, 2, \ldots \tag{1.14}$$

Note that, at time t, the trend components π_t^* and r_t^* anchor the yield of each and every maturity. Intuitively, trends such as π_t^* or r_t^* are the permanent components underlying headline economic variables. A permanent change tends to alter the market's expectation of short-term interest rate forever, and the Treasury yields of all maturities at time t will be affected as a result. For illustration purposes if we write the *slope* and *curvature* as

$$S_t = y_{m+1,t} - y_{1,t},$$
$$C_t = (y_{2m+1,t} - y_{m+1,t}) - (y_{m+1,t} - y_{1,t}),$$

then from (1.14) we can see that S_t and C_t will not change from a move of π_t^* or r_t^*. In this spirit, we postulate that the long-run trend variables (π_t^*, r_t^*) as well as g_t^* have little impact on the *slope* and *curvature.*

1.2.2.3 Pricing Kernel and the $\mathbb{Q}$-Measure

Following the literature, a representative agent values Treasury bonds using a stochastic discount factor of the following form

$$\mathcal{M}_{t+1} = \exp\left\{-y_{1,t} - \boldsymbol{\lambda}_t' \boldsymbol{\nu}_{t+1} - \frac{1}{2}\boldsymbol{\lambda}_t'\boldsymbol{\lambda}_t\right\},$$

where $y_{1,t}$ denotes the yield of a one-month Treasury bill; $\boldsymbol{\nu}_t$ denotes a three-dimensional risk vector that are priced on the bond market; and $\boldsymbol{\lambda}_t$ is the so-called *price of risks.* $\boldsymbol{\nu}_t$ is generated by both macroeconomic risks and non-economic risks, and the macroeconomic risks $\{\boldsymbol{\varepsilon}_{\mathrm{GAP},t}, \varepsilon_{\pi^*,t}, \varepsilon_{g^*,t}, \varepsilon_{r^*,t}\}$ impinge on the bond market through only three channels as made clear by (1.11).

Under the risk-neutral measure $\mathbb{Q}$, we assume that $\boldsymbol{\mathcal{P}}_t$ follows a VAR(1) process as advocated by Joslin et al. (2014)

$$\boldsymbol{\mathcal{P}}_t = \boldsymbol{K}^{\mathbb{Q}}_{0\mathcal{P}} + \boldsymbol{K}^{\mathbb{Q}}_{1\mathcal{P}}\boldsymbol{\mathcal{P}}_{t-1} + \boldsymbol{\Sigma}_{\mathcal{P}}\boldsymbol{e}^{\mathbb{Q}}_t, \quad \boldsymbol{e}^{\mathbb{Q}}_t \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{1.15}$$

1.2.2.4 Yields

Under these assumptions, the yield of an m-month Treasury bond is an affine function of $\boldsymbol{\mathcal{P}}_t$,

$$y_{m,t} = \tilde{A}_m + \tilde{\boldsymbol{B}}_m\boldsymbol{\mathcal{P}}_t + e_{m,t}, \quad e_{m,t} \sim \mathrm{N}(0, \sigma_m^2), \quad m = 1, 2, \ldots, M, \tag{1.16}$$

where the loadings $\left\{\tilde{A}_m\right\}_{m=1}^{M}$ and $\left\{\tilde{\boldsymbol{B}}_m\right\}_{m=1}^{M}$ are regulated by no-arbitrage conditions along the maturity spectrum. They are known functions of the parameters governing the $\mathbb{Q}$ dynamics in (1.15) (see Joslin et al. (2011) for details).

1.2.2.5 State-Space Model

(1.11) and (1.16) form the second group of measurement equations in our model. The factors $\boldsymbol{\mathcal{P}}_t$ are observable from principal analysis of the yields data. Note that although (1.16) does not directly provide information of the state variables $\boldsymbol{M}_t$, it affect their inference indirectly through no-arbitrage conditions.

1.2.3 Summary

In summary, the macroeconomic block specified in Section 1.2.1 together with the finance block developed in Section 1.2.2 makes up our fully-fledged model. In the model, the economy is shaped by both long-run and short-run economic forces. These economic forces are assumed to have left varied traces on the Treasury bonds market. As a result, complementing headline economic data, the Treasury yield curve provides another angle of observing the latent economic factors.

From the perspective of a state-space model, the transition equations are (1.3), (1.5), (1.9) and (1.10), and the measurement equations include (1.2), (1.4), (1.6), (1.8), (1.11) and (1.16).

1.3 Empirical Results

1.3.1 Data

Our data set is at a monthly frequency starting from Jan. 1987 and ending in Sep. 2008. We focus on the relatively tranquil Great Moderation era to avoid the structural breaks such as time-varying volatility or zero-lower bound bought about by the Great Inflation and the Great Recession.

The data set is made up of two groups of series.

The macroeconomic group includes core PCE inflation rate, real GDP growth rate, output gap and effective federal funds rate. The output gap is constructed as the percentage deviation of the BEA's real GDP from the CBO's real potential GDP, both of which are available at a quarterly frequency. We construct the monthly series by interpolating the original quarterly observations.

The finance group is composed of Treasury yields of nine different maturities. All are available at a monthly frequency. 3-month and 6-month yields are published by the Federal Reserve Board of Governors. Longer-term yields including 1, 2, 3, 5, 7, 10 and 20-year ones are retrieved from the widely-used Gürkaynak et al. (2007) data set.

	Prior	***Shape***	***Scale***	***Posterior***
$\boldsymbol{\Phi}$, $\boldsymbol{K}_{0\mathcal{P}}$, $\boldsymbol{K}_{1\mathcal{P}}$, $\boldsymbol{\Psi}$	Flat	NA		Gaussian
$\left(\boldsymbol{\Sigma}_{\text{GAP}}, \sigma^2_{\pi^*}, \sigma^2_{g^*}, \sigma^2_{r^*}\right)$	Inv Gamma	$\mathbf{0}$	$\begin{pmatrix} 0.5 \times \boldsymbol{I}_{4\times4} & \mathbf{0} \\ \mathbf{0} & 0.15 \times \boldsymbol{I}_{2\times2} \end{pmatrix}$	Inv Gamma
$\left(\sigma^2_{\pi}, \sigma^2_{g}, \sigma^2_{i}, \sigma^2_{\tilde{x}}\right)$	Inv Gamma	$\mathbf{0}$	$0.01 \times \boldsymbol{I}_{4\times4}$	Inv Gamma
$\sigma^2_m,\ m = 1, \ldots, M$	Flat after log	NA		Inv Gamma
$\boldsymbol{\Sigma}_{\mathcal{P}}$, $k^{\mathbb{Q}}_{\infty}$, $\lambda_1, \lambda_2, \lambda_3$	Flat	NA. Simulated using independent M.H.		

Table 1.1: Priors of Parameters

1.3.2 Estimation Methodology

We estimate parameters and the latent economic variables $\boldsymbol{M}_t$ defined in (1.12) employing a Gibbs sampling. Given the parameters, the posterior distribution of $\boldsymbol{M}_t$ can be derived with a Kalman filter. And given $\boldsymbol{M}_t$, most parameters can be sampled from their conjugate posteriors. The parameters governing the loadings in (1.16) do not have posteriors conjugate to priors, and a random-walk Metropolis-Hastings algorithm is called in to generate a sample.

We assume uninformative priors for most of the parameters and all latent variables. Several variance parameters are assigned informative priors as detailed in Table 1.1 in order to avoid model degeneracy.

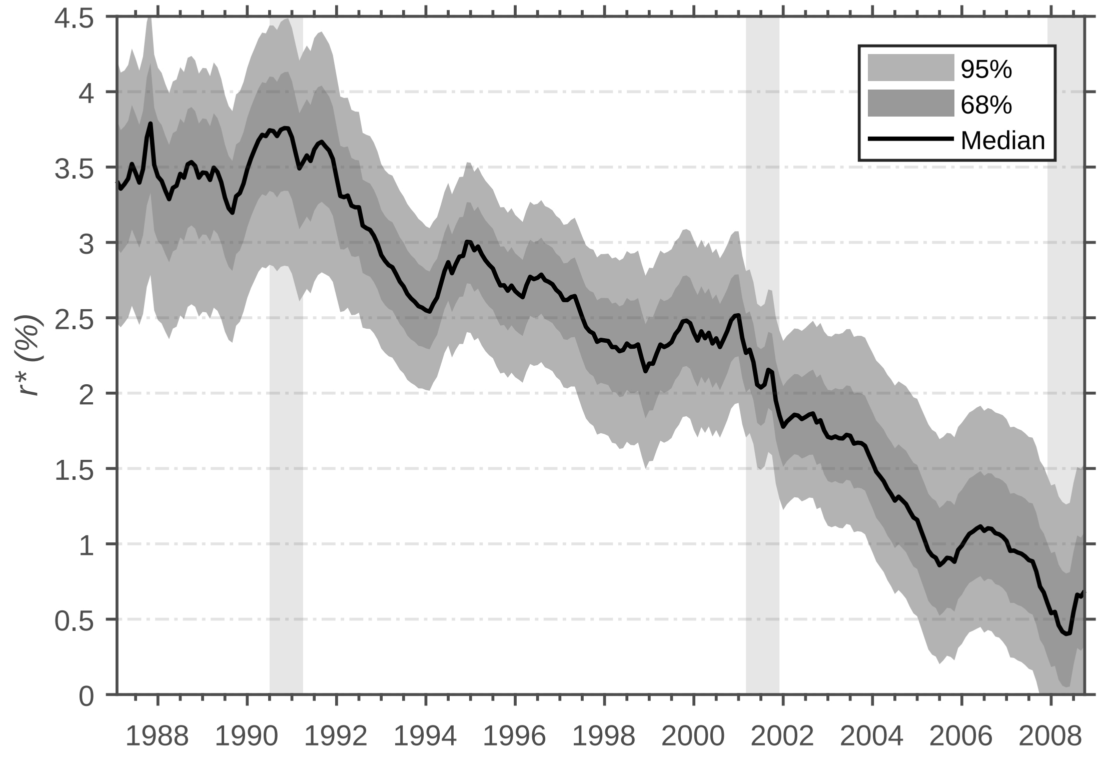


Figure 1.1: Posterior Distribution of r_t^*

1.3.3 Results

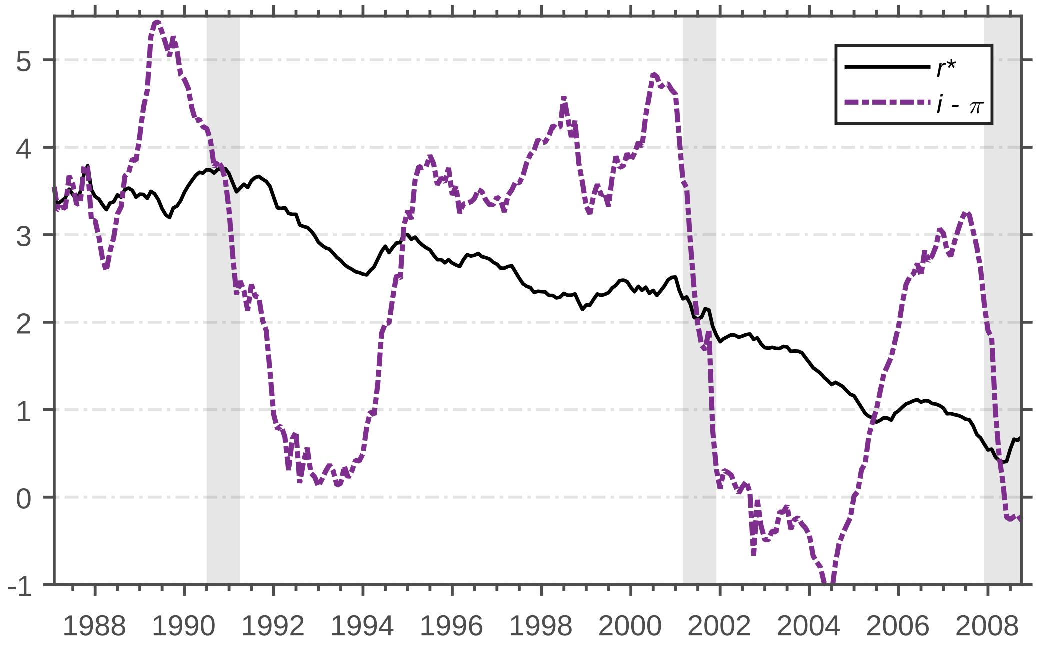
1.3.3.1 r_t^* Estimates

Figure 1.1 depicts our estimates of r_t^*. The solid line on the graph is the smoothed (or two-sided) posterior median, and the shaded bands flanking the median are the 68% and 95% confidence bands.

According to the estimates, we are 95% confident that r_t^* was somewhere between 2.5% and 4% in the late 1980s with median around 3.5%. It stayed roughly unchanged before 1991. Starting from 1992, the rate entered a downward trajectory and lost one percentage point in the next decade. After 2001, the decline accelerated, with another 1.5 percent lost in less than five years. By the end of our sample period, r_t^* was 95% likely to be between 0 and 1.5% in round number. Our estimates suggest that the year of 2007 saw the rate finally dropped below 1%. The 95% confidence band remains approximately 1.5% wide throughout the sample period.

Figure 1.2 illustrates how real federal funds rate moves from side to side around r^*. The solid line in the upper panel is our estimates (the posterior median) of r_t^*, and the dashed line stands for *ex post* real federal funds rate. The shaded rectangles illustrate the NBER's recession periods. Our data sample extends over three recessions, with r_t^* exhibiting little cyclical movement as mandated by its long-run nature.

r_t^* estimates crossed the real rate each time right in the middle of a recession, despite the limited width of each recession rectangle. That is, our estimates demonstrate that monetary policy historically tends to flip from the restrictive side to the accommodative side as the economy slips downhill. Real policy rate before the 2001 recession was as much as 2.5% above neutral. As the economy peaked, real policy rate quickly turned around, broke neutral in May 2001 and recorded a low of 2.5% below in late 2002. Similar pattern is recorded for the other two recessions. It demonstrates that historically the Federal Reserve never aimed the policy rate at neutral. Although r^* has been the anchor, monetary policy has been allowed



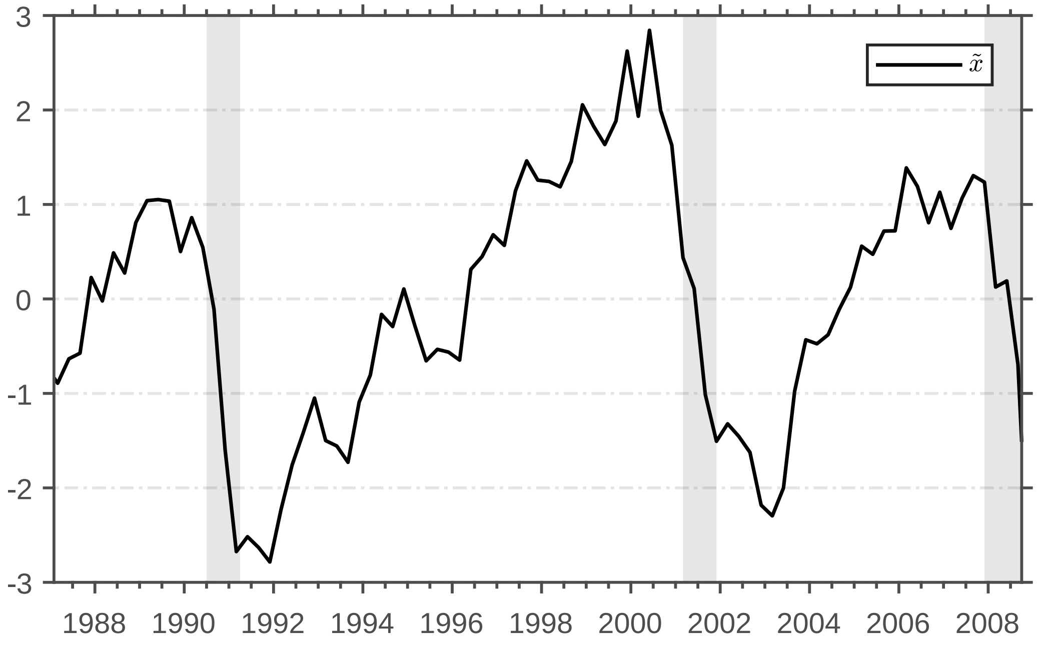


Figure 1.2: Natural Rate, Real Rate (Upper Panel) and Output Gap (Lower Panel)

plenty of leeway swinging around the anchor to lean against the wind. The distance of monetary policy from the neutral provides a gauge of how much accommodation/restriction the policy provides.

1.3.3.2 r_t^*, $\tilde{i}_t$, and Yield Curve

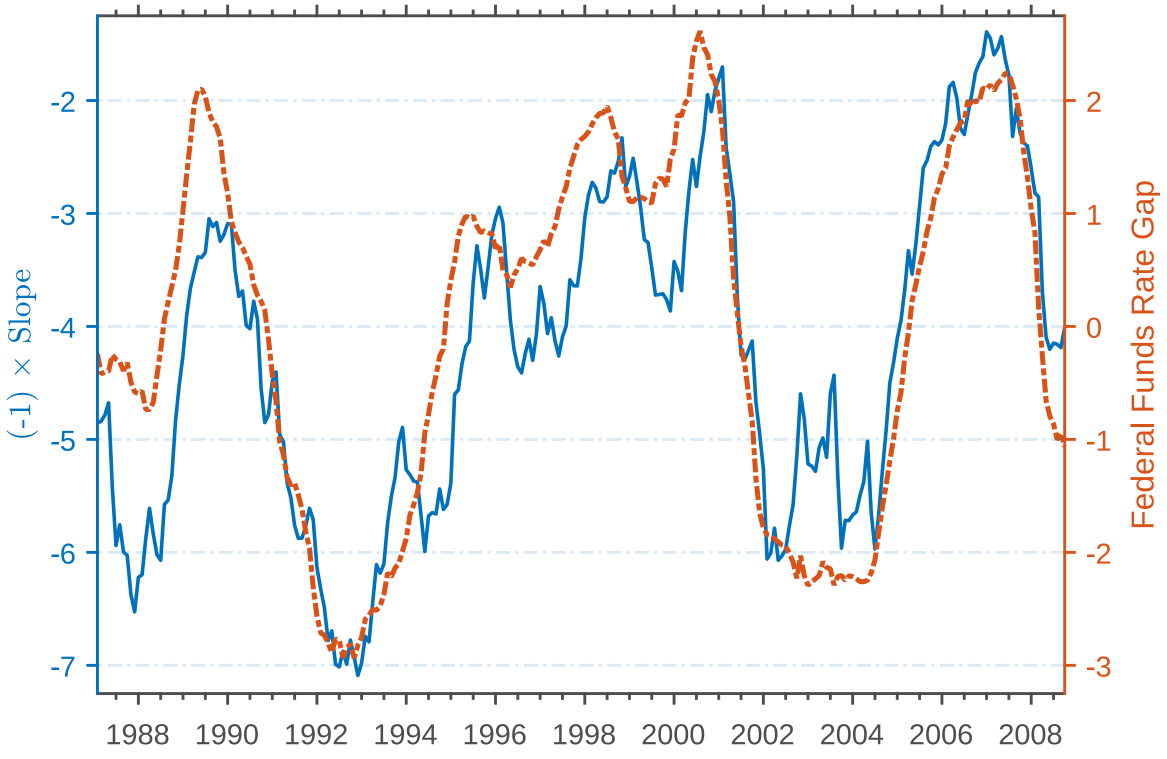


Figure 1.3: The *Slope* and Policy Rate Gap

We find natural rate r_t^* and rate gap $\tilde{i}_t$ are associated with different facets of the yield curve.

Figure 1.3 illustrates our estimates of federal funds rate gap $\tilde{i}_t$ (dashed line) defined in (1.8) together with the (negative of) yield curve *slope* (solid line). With a correlation coefficient of -0.87 between these two, our model provides direct evidence supporting the suggestion of using the *slope* as a quick proxy for the federal funds rate gap Laurent (1988); Bernanke & Blinder (1992); Bomfim (1997).

Intuitively, *ceteris paribus* if the Federal Reserve lowers the federal funds rate, the short end of the Treasury yield curve will dip simultaneously while the long-end tends to remain largely unchanged. As a result the *slope* heads upward. Specifically, from (1.8) and (1.14), the *slope* and the funds rate gap $\tilde{i}_t$ are related in the following way

$$S_t \approx -\tilde{i}_t + \frac{1}{m}\mathrm{E}_t \sum_{i=0}^{m-1} \tilde{i}_{t+1} - \frac{1}{m} \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right). \tag{1.17}$$

Since $\tilde{i}_t$ is stationary, any deviation of $\tilde{i}_t$ from zero is expected to die out in the far future. So, when m is large, the 2nd term on the right hand side will be dominated by the first one, and the equation suggests a strong negative correlation between S_t and $\tilde{i}_t$. Although in the model we postulate that the *slope* is shaped by all three cyclical components $(\tilde{\pi}_t, \tilde{x}_t, \tilde{i}_t)$, as made clear by (1.13), it turns out that $\tilde{i}_t$ alone explains 75.5% of the *slope* variation.

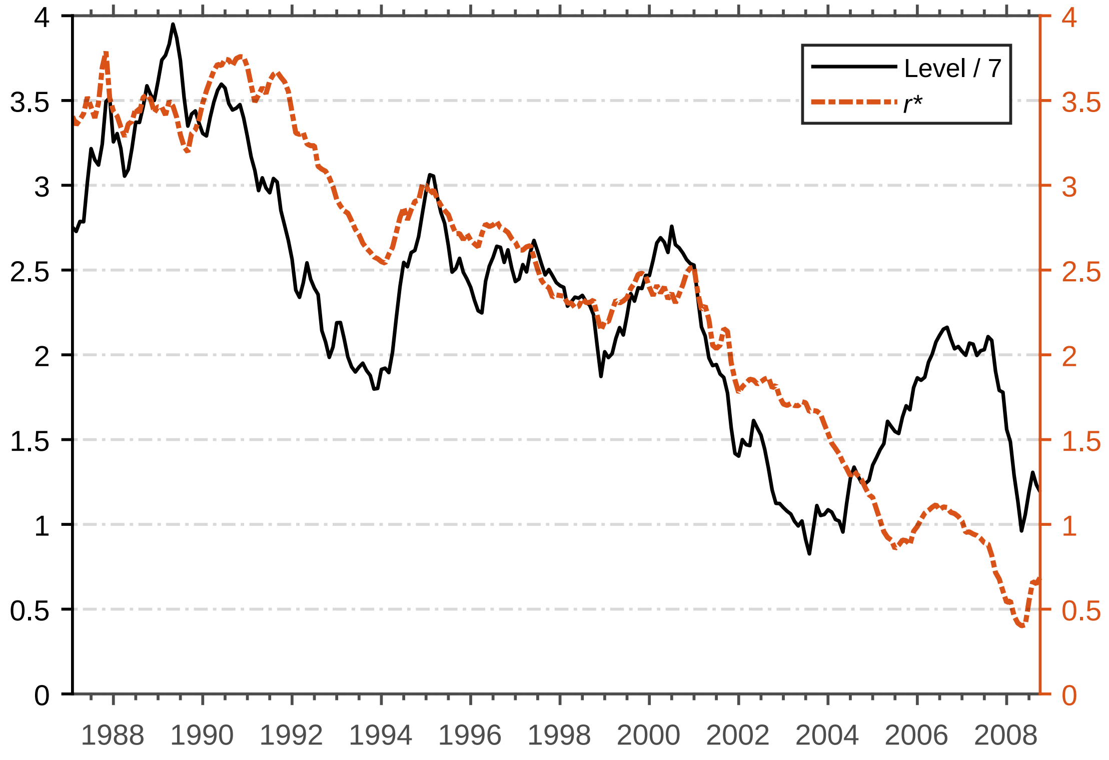


Figure 1.4: The *Level* and r_t^*

On the other hand, our r_t^* estimates exhibit a strong correlation ($\rho = 0.82$) with the *level* as illustrated by Figure 1.4, where the solid line stands for the *level* (scaled down by a factor of 7) and the dashed line is our estimates of r_t^*. In Equation (1.11), the *level* is assumed to be related with all six latent economic variables. But the result shows that r_t^* alone explains 66.4% of *level* variation. Intuitively, the reason lies in the dual-role r_t^* is playing: at the short-end of the yield curve, it forms the basis of federal funds rate and short-term yields through the Taylor rule; and at the long-end, it is the expected short rate and factored into longer-term yields. When r_t^* changes, both ends of the yield curve moves and the *level* changes in the same direction. [2]

1.4 Conclusion

This paper contributes to the discussion of r^* by significantly improving its estimates precision. We explore how the information of r^* is encoded into not only macroeconomic variables but also Treasury yield curve rates. We model economic variables and Treasury yields together using a no-arbitrage macro-finance framework, and estimate the model using Gibbs sampling and Kalman filter. We find r^* highly correlated with the yield curve *level*. The uncertainty band surrounding our r^* estimates is narrower compared with other researches. We find r^* has been declining since the early 1990s, and the hypothesis that r^* is a constant can be rejected with confidence.

[2] The macro-finance literature e.g. Dewachter & Lyrio (2006) and Rudebusch & Wu (2008) usually identifies the *level* with the long-run inflation trend π^* exclusively, assuming a constant natural r^*. Under the more general setup, we do find a high correlation ($\rho = 0.70$) between the *level* and π^*, but the correlation between the *level* and r^* ($\rho = 0.82$) turns out to be even stronger.

Chapter 2

Implicit Particle Gibbs

Hidden Markov models, or called state-space models, have wide applications in economics researches. With transition equations specifying the dynamics of state variables, which are usually from economic theories, and measurement equations connecting state variables to data, HMM are found extremely useful in finding latent economic variables. In state-space systems with linear measurement equations, Kalman filter is well known to provide the benchmark solution. But when the system is equipped with non-linear measurement equations, Kalman filter is not applicable and alternative methods must be resorted to.

Among these alternative algorithms of solving a hidden Markov model, particle filter methods haven been found to be highly flexible and efficient. A sequence of particles are sampled for state variables of each time period using sequential Monte Carlo methods. The particles are then weighted to approximate state variables' posterior distributions given the information available each period. The weights can then be further adjusted to approximate all states' joint distribution given all information available through the dataset. However, the computational efficiency of particle filter in practice faces several well-known challenges, including but not limited to the problems of weights degeneracy and path degeneracy. Many researches have been devoted to the mitigation of these computational challenges.

We contribute to the particle filter literature by proposing a algorithm that infusing two cutting-edge algorithms, with one devoted to addressing the weights degeneracy problem and the other focusing on path degeneracy. The new algorithm is tested in a simple non-linear HMM model and found to exhibit superior efficiency.

2.1 Inference in Hidden Markov Models

Let $(\mathsf{X}, \mathcal{X})$ and $(\mathsf{X}, \mathcal{Y})$ be two measurable spaces. Let M be a Markov kernel on $\mathsf{X} \times \mathcal{X}$, and G a Markov kernel on $\mathsf{X} \times \mathcal{Y}$. Then the map K defined as

$$K(x; C) \equiv \iint_C M(x, \mathrm{d}x')G(x', \mathrm{d}y'), \quad x, x' \in \mathsf{X}, \quad C \in \mathcal{X} \otimes \mathcal{Y} \tag{2.1}$$

is Markov kernel defined on $\mathsf{X} \times (\mathcal{X} \otimes \mathcal{Y})$.

Let $\{X_t\}_{t\in\mathbb{N}}$ and $\{Y_t\}_{t\in\mathbb{N}}$ be two discrete-time stochastic processes whose state spaces are given by $(\mathsf{X}, \mathcal{X})$ and $(\mathsf{Y}, \mathcal{Y})$ respectively and transition probability is given by K. If $\{X_t\}_{t\in\mathbb{N}}$ is not observable, then the Markov chain $\{(X_t, Y_t)\}_{t\in\mathbb{N}}$ following kernel K is called a Hidden Markov model (HMM).

Statistical inference for Hidden Markov models involves computing the posterior distribution, $\phi_{s:s'|t}$, of a collection of state variables $X_{s:s'} \equiv (X_s, \ldots, X_{s'})$, with $s < s'$ conditional on a batch of observations, $Y_{0:t} \equiv (Y_0, \ldots, Y_t)$.

2.1.1 filtering

Filtering is the inference of $\phi_{t|t}$, i.e. the posterior distribution of X_t given $Y_{0:t}$. We abbreviate $\phi_{t|t}$ to ϕ_t.

If there exists a probability measure ν on $(\mathsf{Y}, \mathcal{Y})$ such that for all $x \in \mathsf{X}$, the probability measure $G(x, \cdot)$ is absolutely continuous with respect to $\nu(\cdot)$, then there exist a density function $g(x, \cdot)$ such that for $A \in \mathcal{Y}$,

$$G(x, A) = \int_A g(x, y)\nu(\mathrm{d}y)$$

Let f be a measurable function defined on $(\mathsf{X}, \mathcal{X})$, and Ψ a probability measure define on $(\mathsf{X}, \mathcal{X})$. If we denote the kernel

$$Q_t(x_{t-1}, f) \equiv \int_A M(x_{t-1}, \mathrm{d}x_t) g(x_t, y_t) f(x_t), \quad A \in \mathcal{X}, \tag{2.2}$$

and denote

$$\gamma_t(f) \equiv \int \cdots \int_{\mathsf{X}^{t+1}} \Psi(\mathrm{d}x_0) g(x_0, y_0) Q_1(x_0, \mathrm{d}x_1) \ldots Q(x_{t-1}, \mathrm{d}x_t) f(x_t)$$

then

$$\gamma_t(f) = \iint_{\mathsf{X}^2} \gamma_{t-1}(\mathrm{d}x_{t-1}) Q_t(x_{t-1}, \mathrm{d}x_t) f(x_t) \tag{2.3}$$

The posterior distribution of X_t conditional on $Y_{0:t}$, be defition, is given by

$$\begin{aligned} \phi_t(f) &= \frac{\gamma_t(f)}{\gamma_t(1)} \\ &\propto \gamma_t(f) \end{aligned} \tag{2.4}$$

Therefore, the filtering distribution of time t can be derived using the filtering distribution of the previous period using Equation (2.3).

Algorithm 2.1.1 (Forward Filtering)**.** Compute forward in time the filtering distributions $\phi_0, \ldots, \phi_T$ using the recursion (2.3).

2.1.2 Smoothing

Smoothing is the inference of $\phi_{0:t|t}$, i.e. the joint posterior distribution of $X_{0:t}$ given $Y_{0:t}$.

2.1.2.1 Backward Kernel

If there exists a probability measure μ on $(\mathsf{X}, \mathcal{X})$ such that for all $x \in \mathsf{X}$, the probability measure $M(x, \cdot)$ is absolutely continuous with respect to $\mu(\cdot)$, then there exist a density function $m(x,\cdot)$ such that for $A \in \mathcal{X}$,

$$M(x,A) = \int_A m(x,x')\mu(\mathrm{d}x')$$

Let f be a measurable function defined on $(\mathsf{X}^2, \mathcal{X}^{\otimes 2})$. Define

$$B_{\phi_{t-1}}(x_t, f) \equiv \frac{\iint_{\mathsf{X}^2} \phi_{t-1}(\mathrm{d}x_{t-1}) m(x_{t-1}, \mathrm{d}x_t) f(x_{t-1}, x_t)}{\iint_{\mathsf{X}^2} \phi_{t-1}(\mathrm{d}x_{t-1}) m(x_{t-1}, \mathrm{d}x_t)}$$

Then $B_{\phi_{t-1}}$ is a backward kernel that when X_t and $Y_{0:(t-1)}$ are known, gives the probability of X_{t-1}.

2.1.2.2 Backward Recursion

Let $0 \leq s \leq t$. Let f be a measurable function defined on $(\mathsf{X}^{t-s+2}, \mathcal{X}^{\otimes(t-s+2)})$.

Given $\phi_{s:t|t}$, then

$$\phi_{(s-1):t|t} = \int \cdots \int_{\mathsf{X}^{t-s+2}} \phi_{s:t|t}(\mathrm{d}x_{s:t}) B_{\phi_{s-1}}(x_s, \mathrm{d}x_{s-1}) f(x_{(s-1):t}) \tag{2.5}$$

So the smoothing distribution of time s can be derived from the smoothing distribution of the later period using the above recursion.

Algorithm 2.1.2 (Backward Smoothing)**.** From ϕ_t, compute backward in time the smoothing distribution $\phi_t, \phi_{t-1:t|t}, \ldots, \phi_{0:t|t}$ using the recursion (2.5).

2.1.3 Forward Filtering Backward Smoothing

In summary, the inference of the distribution of the hidden state $X_{0:T}$ can be implemented in two steps:

Algorithm 2.1.3 (Forward Filtering Backward Smoothing)**.** For $t = 0, 1, \ldots, T$, $\phi_{0:t|t}$ can be derived by completing the following Filtering and Smoothing steps:

1. Compute forward in time the filtering distributions $\phi_0, \ldots, \phi_T$ using the recursion (2.3).

2. From ϕ_t, compute backward in time the smoothing distribution $\phi_t, \phi_{t-1:t|t}, \ldots, \phi_{0:t|t}$ using the recursion (2.5).

2.2 Particle Filters

2.2.1 Particle Filtering

The filtering state of Particle Filters are a combination of the sequential importance sampling method introduced in Handschin & Mayne (1969) and the sampling importance resampling algorithm proposed in Rubin (1987).

2.2.1.1 Sequential Importance Sampling

μ denotes a probability measure on a measurable space $(\mathsf{X}, \mathcal{X})$, which is referred to as the *target distribution*. The aim of importance sampling is to get a set of sample points that approximates the target distribution well. The plain Monte Carlo approach consists in drawing an i.i.d. sample $\{X^i\}_{i=1}^N$ of size N from the target distribution μ.

But in some situations, it is more appropriate to sample from another probability distribution ν on $(\mathsf{X}, \mathcal{X})$ called the *proposal distribution*. Assume that the target distribution μ is absolutely continuous with respect to ν, and denote by $w \equiv \mathrm{d}\mu/\mathrm{d}\nu$ the Radon-Nikodym derivative of μ with respect to ν, referred to in the sequel as the *weight function*. Then, for $f \in L^1(\mu)$,

$$\mu(f) = \int f(x)\mu(\mathrm{d}x) = \int f(x)w(x)\nu(\mathrm{d}x) = \nu(fw).$$

If $\{X^i\}_{i=1}^N$ is an i.i.d. sample from ν, the above equation suggests the following estimator of

$\mu(f)$:

$$N^{-1} \sum_{i=1}^{N} f(X^i) w(X^i),$$

which is called the *importance sampling estimator*. $\{(X^i, w(X^i))_{i=1}^N\}$ is called a *weighted sample*. So each sample points, or *particle* X^i is labeled by weight $w(X^i)$.

Importance sampling introduces little restrictions on the choice of the proposal distribution. The choice is typically guided by two requirements: the proposal distribution should be easy to simulate and should lead to efficient estimator.

According to Equations (2.3) and (2.4), the filtering distribution

$$\phi_t(f) = \iint_{\mathsf{X}^2} \phi_{t-1}(\mathrm{d}x_{t-1}) Q_t(x_{t-1}, \mathrm{d}x_t) f(x_t)$$

Suppose that the weighted sample $\{(X_{t-1}^i, \omega_{t-1}^i)\}_{i=1}^N$ is consistent for ϕ_{t-1}. We can construct a weighted sample $\{(X_t^i, \omega_t^i)\}_{i=1}^N$ that approximates ϕ_t as follows. In the proposal step, each particle X_{t-1}^i gives a single offspring X_t^i, $i \in \{1, \ldots, N\}$, of which the distribution is specified by the proposal kernel $R_t(X_{t-1}^i, \cdot)$. Next, we assign to the new particle X_t^i the importance weight

$$\omega_t^i = \omega_{t-1}^i w_t(X_{t-1}^i, X_t^i) \tag{2.6}$$

where, given x_{t-1}^i, w_t is the Radon-Nikodym density of $Q(x_{t-1}^i, \cdot)$ with respect to $R(X_{t-1}^i, \cdot)$.

The first obvious choice is to set $R_t = M$, which is called the *prior kernel*. The weight function w_t then, according to (2.2), simplifies to

$$w_t(X_{t-1}^i, X_t^i) = g(X_t^i, Y_t)$$

This kernel is often convenient: sampling particles from M is often straightforward, and computing the incremental weight amounts to evaluating the conditional likelihood of the new observation given the current particle.

Another obvious choice of the proposal is $Q(X_{t-1}^i, \cdot)$ with incremental weight 1, which is the optimal case we can get. The optimal kernel was introduced in Zaritskii et al. (1976) and Akashi & Kumamoto (1977). When the observation equation is linear, the optimal kernel is convenient.

The key problem of sequential importance sampling is that the weights will degenerate as the time index t goes forward, as can be seen from (2.6). Most of the normalized weights $\omega_t^i / \sum_i \omega_t^i$ close to 0 except for a few. Weight ω_t^i measures the adequacy of the particle X_t^i to the target distribution ϕ_t. A particle with small weight does not contribute to the approximation. If there are too many ineffective particles, the particle approximation is inefficient.

2.2.1.2 Resampling

To avoid the degeneracy of the importance weights, Gordon et al. (1993) proposed a solution based on *resampling* using the normalized weights as probabilities of selection. That resampling method is rooted in the *sampling importance resampling*, or SIR, method to sample a distribution μ, introduced by Rubin (1987, 1988).

In the setting of a single step importance estimator, the SIR process proceeds in two stages. In the *sampling stage*, an i.i.d. sample $\{X^i\}_{i=1}^N$ is drawn from the proposal distribution ν. The importance weights are then evaluated at particle positions, $\omega^i = w(X^i)$. In the *resampling* state, a sample of size N denoted by $\{\tilde{X}^i\}_{i=1}^N$ is drawn from the set of points $\{X^i\}_{i=1}^N$, with probability proportional to the weights ω^i. Doing so we obtain an equally weighted sample $\{\tilde{X}^i, 1\}_{i=1}^N$ also targeting μ.

Algorithm 2.2.1 (Sequential Importance Sampling with Resampling)**.**

Initial State Draw an i.i.d. sample $\{X_0^{N,i}\}_{i=1}^N$ from R_0, and set

$$\omega_0^{N,i} = g_0(X_0^{N,i})w_0(X_0^{N,i}) \quad \text{for } i = 1, \dots, N.$$

Recursion For $t = 1, 2, \dots, T$,

- Draw $\{X_t^{N,i}\}_{i=1}^N$ conditionally independently from the distribution $R_t(X_{t-1}^{N,i}, \cdot)$.
- Compute the updated importance weights

$$\omega_t^{N,i} = \omega_{t-1}^{N,i} w_t(X_{t-1}^{N,i}, X_t^{N,i}), \quad i = 1, \dots, N.$$

Resampling (Optional) Draw a multinomial trial $\{I_t^i\}_{i=1}^N$ with probabilities of success $\{\omega_t^i/\Omega_t^N\}_{i=1}^N$, and set $X_t^i = X_t^{N,I_t^i}$ and $\omega_t^i = 1$ for $i = 1, \dots, N$.

2.2.2 Particle Smoothing

2.2.2.1 Poor Man's Smoothing

From particle filtering, we generate the weighted samples $\{(X_t^i, \omega_t^i)\}_{i=1}^N$ targeting the filtering distribution ϕ_t.

Let I_t^i denote the index of the ancestral particle from which the particle X_t^i originates. We call I_t^i the *one-step ancestor index* of the particle X_t^i. Then from these one-step ancestor indexes, we can construct the *ancestral path* of X_t^i. Let

$$B_t^i = i,$$
$$B_s^i = I_{s+1}^{B_{s+1}^i}, \quad s = 0, \dots, t-1$$

be the *ancestor indexes* or X_t^i. Then the *ancestral path*, $X_{0:t}^i$, of X_t^i, is simply

$$X_{0:t}^i = (X_0^{B_0^i}, \dots, X_t^{B_t^i}).$$

Then the weighted sample $\{(X_{0:T}^i, \omega_T^i)\}_{i=1}^N$ is consistent for the joint smoothing distribution $\phi_{0:T|T}$. That is, if we sample a full time 0 to time T path from $\{(X_{0:T}^i, \omega_T^i)\}_{i=1}^N$, the path will be distributed according to $\phi_{0:T|T}$.

Despite this favorable theoretical result, approximating the joint smoothing distribution with the ancestral path of $\{X_t^i\}_{i=1}^N$ is doomed to failure. The later particles tend to be all generated by a couple of particles from the previous time step. This is a problem known as *path degeneracy.* If the time steps are sensibly long, the approximation of smoothing distribution near the beginning of the time always fails.

The problem of path degeneracy is solved in two step. Andrieu et al. (2010) proposed a algorithm called *conditional particle Gibbs.* And then Lindsten et al. (2012, 2014) augmented this algorithm with a *ancestor sampling* step.

2.2.2.2 Conditional Particle Gibbs

The smoothing distribution $\phi_{0:T|T}$ can be viewed as a marginal distribution of a bigger distribution $\pi(X_{0:T}, I_{0:T}, J)$, where J is an index variable following the multinomial distribution $\{\omega_T^i / \sum_i \omega_T^i\}_{i=1}^N$; $I_{0:T}$ are the one-step ancestor indexes as defined in the section 2.2.2.1. And then $\pi(X_{0:T}, I_{0:T}, J)$ can be viewed as a distribution of compound variables $\{X_{0:n}^{-B_{0:n}^J}, I_{0:n}^{-B_{0:n}^J}, X_{0:n}^{B_{0:n}^J}, B_{0:n}^J\}$.

Condition particle Gibbs sampling from $\pi(X_{0:n}^{-B_{0:n}^J}, I_{0:n}^{-B_{0:n}^J}, X_{0:n}^{B_{0:n}^J}, B_{0:n}^J)$ consists of sampling iteratively from $\pi(X_{0:n}^{-B_{0:n}^J}, I_{0:n}^{-B_{0:n}^J} | X_{0:n}^{B_{0:n}^J}, B_{0:n}^J)$ and $\pi(X_{0:n}^{B_{0:n}^J}, B_{0:n}^J | X_{0:n}^{-B_{0:n}^J}, I_{0:n}^{-B_{0:n}^J})$. To sample the first part,

Algorithm 2.2.2. [Particle Gibbs Part 1]

For $t = 0$**:**

- For $\ell \neq b_0^j$, draw $X_0^\ell \sim r_0(\cdot)$;
- set $X_0^{b_0^j} = x_0^{b_0^j}$.
- For $\ell = 1, \ldots, N$, compute $\omega_0^\ell = g_0(X_0^\ell) w_0(X_0^\ell)$.

For $t = 1, \ldots, n$**:** Given $\{\boldsymbol{X}_{t-1}, \boldsymbol{\omega}_{t-1}\}$,

- for $\ell \neq b_t^j$, draw $(I_t^\ell, X_t^\ell) \sim \mathrm{p}_t(\cdot, \cdot)$;
- set $X_t^{b_t^j} = x_t^{b_t^j}$. Set $I_t^{b_t^j} = b_{t-1}^j$.
- For $\ell = 1, \ldots, N$, compute $\omega_t^\ell = w_t(X_{t-1}^{I_t^\ell}, X_t^\ell)/\vartheta_t(X_{t-1}^{I_t^\ell})$.

And to sample the second part,

Algorithm 2.2.3. [Particle Gibbs Part 2]

For $t = n$**,**

- sample

$$J \sim \left\{ \frac{\omega_n^1}{\Omega_n^N}, \ldots, \frac{\omega_n^N}{\Omega_n^N} \right\}.$$

- Set $B_n^J = J$.
- Set $X_n^J = x_n^J$.

For $t = n-1, \ldots, 0$**,**

- Set $B_t^J = i_{t+1}^{B_{t+1}^J}$.
- Set $X_t^J = x_t^{B_t^J}$.

We get the conditional Particle Gibbs algorithm by combining the two parts

Algorithm 2.2.4. [Particle Gibbs]

1. Initialize. Set $\{X_{0:n}[0], B_{0:n}[0]\}$ arbitrarily.
2. For Iteration $k \geq 1$,
   (a) run a CPF algorithm 2.2.2 conditional on $\{X_{0:n}[k-1], B_{0:n}[k-1]\}$, and

(b) run Algorithm 2.2.3 and set $\{X_{0:n}[k], B_{0:n}[k]\} = \{X_{0:n}^{B_{0:N}^J}, B_{0:n}^J\}$.

This Gibbs algorithm, as introduced in the seminal paper by Andrieu et al. (2010), is the most straightforward. However, this method is known to suffer from path degeneracy Douc et al. (2014). Hence, for this method to worker properly, the number of particles N needs to be large enough. For many problems, this is unrealistic from a computational point of view. This issue can be addressed by modifying the Gibbs sweep.

2.2.2.3 Conditional Particle Filter with Ancestor Sampling

This method, which aims to address the degeneracy problem of conditional particle Gibbs sampler, was introduced by Lindsten et al. (2012, 2014) and Lindsten & Schön (2013). The idea is to sample new values of the ancestor indexes $B_{0:n-1}^J$ as part of procedure 2.2.2. For $t = 1, \ldots, n$, after having sampled $(I_t^{-b_t^j}, X_t^{-b_t^j})$, we add a step in which we sample a new value for $I_t^{b_t^j}$, resulting in the following sweep:

Algorithm 2.2.5. [Particle Gibbs with Ancestor Sampling Part 1]
Input:

- Conditioned particles $(x_{0:n}^{b_{0:n}^j}, j)$.

For $t = 0$**:**

- For $\ell \neq b_0^j$, draw $X_0^\ell \sim r_0(\cdot)$;
- set $X_0^{b_0^j} = x_0^{b_0^j}$.
- For $\ell = 1, \ldots, N$, compute $\omega_0^\ell = g_0(X_0^\ell) w_0(X_0^\ell)$.

For $t = 1, \ldots, n$**:** Given $\{\boldsymbol{X}_{t-1}, \boldsymbol{\omega}_{t-1}\}$,

- for $\ell \neq b_t^j$, draw $(I_t^\ell, X_t^\ell) \sim \mathrm{p}_t(\cdot, \cdot)$;
- set $X_t^{b_t^j} = x_t^{b_t^j}$.

- Draw $I_t^{b_t^j}$ with conditional probability of $I_t^{b_t^j} = \ell$ given by

$$\frac{\omega_{t-1}^{\ell} m(X_{t-1}^{\ell}, x_t^{b_t^j})}{\sum_{\ell=1}^{N} \omega_{t-1}^{\ell} m(X_{t-1}^{\ell}, x_t^{b_t^j})}.$$

- For $\ell = 1, \ldots, N$, compute $\omega_t^{\ell} = w_t(X_{t-1}^{I_t^{\ell}}, X_t^{\ell}) / \vartheta_t(X_{t-1}^{I_t^{\ell}})$.

Output:

- $(\boldsymbol{X}_{0:n}, \boldsymbol{I}_{1:n-1}, \boldsymbol{I}_n^{-j})$.

2.3 Implicit Particle Gibbs

The choice of a good proposal is vital to solving the problem of weights inbalance. As discussed in Section 2.2.1.1, the proposal distribution of generating X_t^i has many options. The optimal proposal is $Q_t(X_{t-1}^i, \cdot)$, and yet direct sampling from it is in most cases infeasible. Actually, the optimal proposal is feasible only when the measurement equations of the model are linear. Chorin et al. (2010) proposed the *implicit particle filter* method to find an approximation of the optimal proposal.

2.3.1 Implicit Particle filter

Although direct sampling is difficult, we can sample the optimal proposal $Q_t(X_{t-1}^i, \cdot)$ indirectly using importance sampling techniques from another proposal distribution. To get the particle positions, the optimal proposal $Q_t(X_{t-1}^i, \cdot)$, in its density function, can be explicitly written into

$$q_t(X_{t-1}^i, x_t) \propto \exp\left\{-F_t^i(x_t)\right\}. \tag{2.7}$$

Suppose x_t is an m-dimensional random variable. We sample φ from an m-dimensional standard normal distribution

$$\varphi \sim \frac{1}{(2\pi)^{m/2}} \exp\left\{-\frac{\varphi^T\varphi}{2}\right\},$$

and solve the equation

$$F_t^i(x_t) - C_t^i = \frac{\varphi^T\varphi}{2}, \tag{2.8}$$

where C_t^i is a constant. The value of φ is most likely in the neighborhood of 0. Therefore, if the constant is chosen in a way such that $C_t^i = \min_{x_t} F_t^i(x_t)$, then Equation (2.8) ensures that the solution x_t is a high probability position for $Q_t(X_{t-1}^i, \cdot)$.

To derive the particle weights, suppose (2.8) is solved by $x_t^i(\varphi)$. Then, each value x_t of X_t^i appears with probability density

$$\frac{1}{(2\pi)^{m/2}} \exp\left\{-\frac{\varphi^T(x_t)\varphi(x_t)}{2}\right\} |J|^{-1},$$

where J is the Jacobian of $X_t^i(\varphi)$, while the target density $q_t(X_{t-1}^i, \cdot)$, from (2.7) and (2.8), is proportional to

$$\exp\left\{-C_t^i\right\} \exp\left\{-\frac{\varphi^T(x_t)\varphi(x_t)}{2}\right\}.$$

Therefore, the weight of the particle should be

$$\exp\left\{-C_t^i\right\} \cdot |J|(2\pi)^{m/2}.$$

2.3.1.1 Implementation

As proposed by Morzfeld et al. (2012), we could implement the importance sampling in three steps.

Firstly, we need to solve the minimization problem, $\min_{x_t} F_t^i(x_t)$, using standard tools, e.g. Newton's method, a quasi-Newton method, or more sophisticated minimization strategies. Denote

$$m_t^i = \arg\min_{x_t} F_t^i(x_t),$$
$$C_t^i = F_t^i(m_t^i).$$

And if we use Newton's method, then in the neighborhood of m_t^i, we have

$$F_t^i(x_t) \approx C_t^i + \frac{1}{2}(x_t - m_t^i)^T H_t^i (x_t - m_t^i),$$

where H_t^i denotes the Hessian evaluated at the minimum. Cholesky decompose the Hessian $H_t^i = (U_t^i)^T U_t^i$.

Secondly, we solve (2.8). As we noted before, (2.8) has multiple solutions. One of them can be found by letting

$$x_t = m_t^i + \lambda_t^i U_t^i \varphi, \tag{2.9}$$

where U_t^i is some $m \times m$ matrix under our control and remains to be chosen. Substitute (2.9) into (2.8), and we get an algebraic equation with one single variable λ_t^i

$$F_t^i(m_t^i + \lambda_t^i U_t^i \varphi) - C_t^i = \frac{\varphi^T \varphi}{2}. \tag{2.10}$$

To initialize the numerical computation, choose $\lambda_t^{i0} = 1$. Intuitively, Equation (2.8) has a solution in various directions geometrically. We pick a direction, $U_t^i \varphi$, and determine how far we need to walk along the direction to reach the solution.

What remains to be done is to determine the Jacobian. From (2.9), we know

$$\frac{\partial x_t}{\partial \varphi} = U_t^i \left(\varphi \frac{\partial \lambda_t^i}{\partial \varphi} + \lambda_t^i I \right),$$

where $\partial \lambda_t^i / \partial \varphi$ is a row vector, which can be derived implicitly from (2.10) as follows:

$$(\nabla F_t^i) \left(\lambda_t^i U_t^i + U_t^i \varphi \frac{\partial \lambda_t^i}{\partial \varphi} \right) = \varphi^T,$$
$$\frac{\partial \lambda_t^i}{\partial \varphi} = \frac{\varphi^T - \lambda_t^i (\nabla F_t^i) U_t^i}{(\nabla F_t^i) U_t^i \varphi}$$

with ∇F_t^i being a row vector denoting the gradient of F_t^i. So

$$\frac{\partial x_t}{\partial \varphi} = U_t^i \left[\frac{1}{(\nabla F_t^i) U_t^i \varphi} \varphi \left(\varphi^T - \lambda_t^i (\nabla F_t^i) U_t^i \right) + \lambda_t^i I \right].$$

We the derive the Jacobian:

$$J = \det(U_t^i) \cdot (\lambda_t^i)^{m-1} \cdot \frac{\varphi^T \varphi}{(\nabla F_t^i) U_t^i \varphi}.$$

2.3.2 Implicit Particle Gibbs

We propose to incorporate the implicit particle filter step in a Gibbs sampler as part of procedure 2.2.5. The idea is to always sample new particles from a near-optimal proposal distribution as discussed in the previous section. For $t = 1, \ldots, n$, when we draw $(I_t^{-b_t^j}, X_t^{-b_t^j})$, we add a step of implicit filtering, solving the maximization problem of the optimal proposal and sampling a new value around its maximizer.

We implemented this algorithm in the following example, and compared its performance with Lindsten et al. (2012)'s particle Gibbs with ancestor sampling.

Example 2.3.1 (NGM Model).

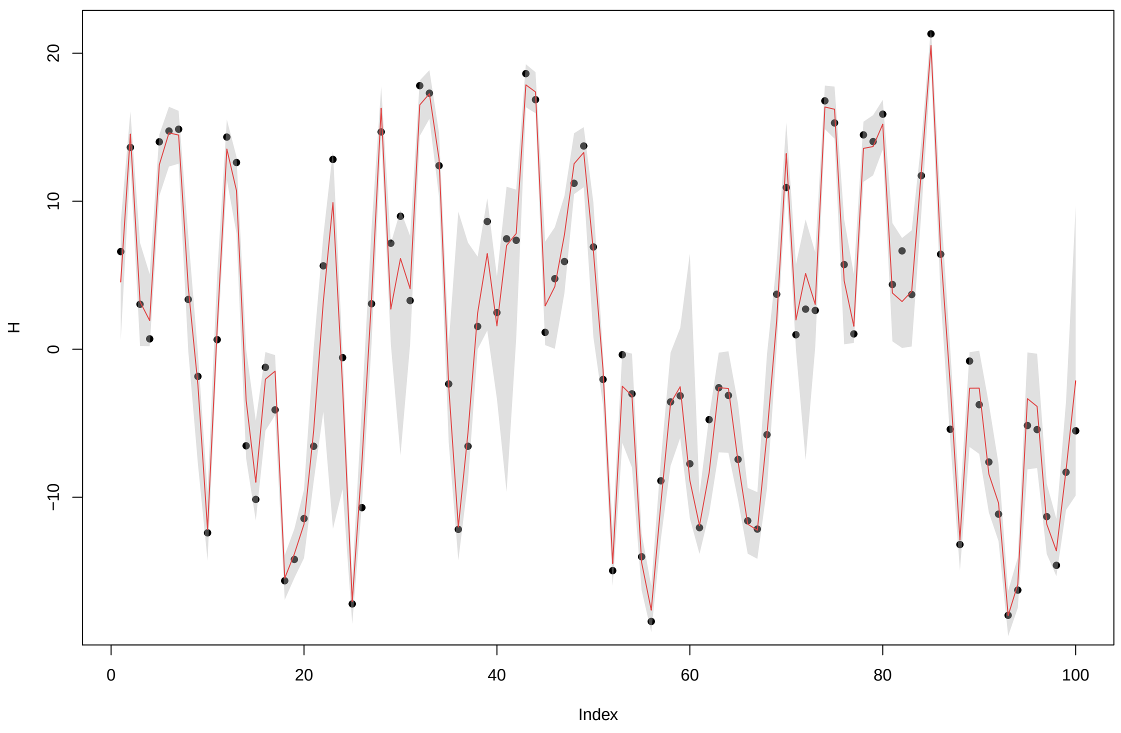
$$x_t = \alpha x_{t-1} + \beta \frac{x_{t-1}}{1 + x_{t-1}^2} + \gamma \cos[1.2(t-1)] + \eta_t,$$
$$y_t = \frac{x_t^2}{20} + \epsilon_t,$$

where y_t is a function of x_t^2.

The result is shown in Figure 2.1. We find that both algorithms filtered out the true states reasonably well. But the confidence band around the implicit particle Gibbs results is usually smaller than that of the other algorithm, demonstrating an improved efficiency gained from the implicit particle filtering step.

2.4 Conclusion

Particle Filter methods, or known as Sequential Monte Carlo methods, are a powerful class of algorithms in solving non-linear state-space models. Although widely used, particle filter methods are known to suffer two challenges in practice. The optimal proposal distribution is vital to mitigate the problems of both weights degeneracy and path degeneracy. But direct sampling from the optimal proposal is usually not possible. Implicit Particle Filtering, proposed by Chorin et al. (2010) and Morzfeld et al. (2012), gives an algorithm to sample indirectly from the optimal proposal using importance sampling techniques. A Gibbs sampler can then be constructed by modifying the Particle Gibbs with Ancestor Sampling algorithm proposed by Lindsten et al. (2012), replacing the particle filtering step in that algorithm by implicit particle filtering. We tested the socalled Implicit Partile Gibbs algorithm in a small non-linear state-space model and found superior inference performance.



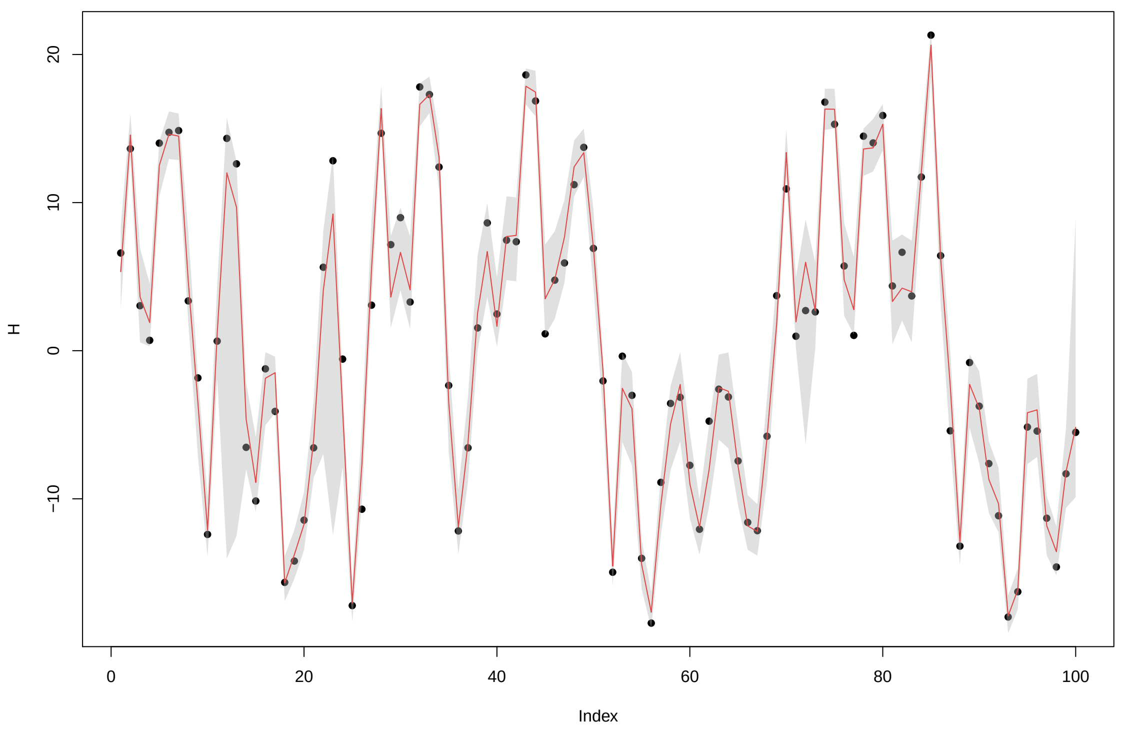


Figure 2.1: NGM Model - Particle Gibbs v.s. Implicit Particle Gibbs

Chapter 3

Deep Reinforcement Learning

The recent wave of advancement in artificial intelligence is driven mainly by the breakthrough in one subfield of machine learning - deep neural networks. The technological breakthroughs in DNN since late 2000s greatly improve the performance of machine learning problems. Researchers could now delegate one most important job - feature engineering - to an automated algorithm. As result, reliance on domain knowledge is greatly weakened and performance has been improved significantly.

Reinforcement learning is one field of machine learning that has benefited from DNN. In a reinforcement learning problem, an agent is trained to make automatic decisions through interactions with the environment. Commercial AI applications, e.g. self-driving cars, needs the agent to achieve human-level performance on a specific task. Therefore, the algorithm underlying the agent is required to be able to handle large and complex state space and to evolve quickly in real time as interactions with environment are underway. The state space of a self-driving system can be conceived as a collection of data, ranging from camera videos to radar signals, collected by various types of sensors. Traditional RL algorithms have been successfully applied to state spaces of moderate dimensions. But applicability to a high dimensional state space like that of a self-driving system remained impractical until reinforcement learning is combined with deep neural networks. Deep reinforcement learning enables a reinforcement learning agent to deal with large batches of data quickly, and hence gives rise to advanced algorithms that defeated world champions in a game called Go. And thus we are seeing the ever growing popularity in AI in almost every field of studies.

In addition, many commercial applications like a self-driving system cannot afford to wait for a big batch of accidents data before updating its algorithm. Quickly evolving to fix problems that have newly emerged is everything to a self-driving agent's success. Reinforcement leaning techniques including temporal difference (TD) and dyna-TD cater to this need.

In this chapter, we review the core ideas of both reinforcement learning and deep learning from the vantage of economics research. We point out the connection between RL and dynamic programming in Section 3.1, and then explains how RL caters to the two goals of a practical AI project in late Section 3.1 through Section 3.4. Section 3.5 introduces the architecture of deep neural network, especially one architecture is interesting particular to time series modeling. In Section 3.6, we review the applications of deep reinforcement learning in financial economics, game theory and macroeconomics.

3.1 Temporal Difference

Suppose we are playing a game that involves multiple time steps, and, across the time steps of the game, the state S_t follows a Markov process. A policy function π prescribes the action A_t a player could take under each state. By taking a certain action under a certain state, the player can earn some reward R_t each time step.

At time t, suppose the environment is in state s and the player plays action a. He will play in the subsequent time steps according to the policy function π. Then the value function q of following a policy function π can be defined as the expectation of the sum of all the following rewards

$$q_\pi(s,a) \equiv \mathrm{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T | S_t = s, A_t = a\right], \quad \text{for } s \in \mathbb{S} \tag{3.1}$$

where the subscript π denotes that the expectation is taken with respect to the probability distribution defined by policy function π.

The goal of solving the Markov decision problem is to find the optimal policy function

π_* that prescribes the best action each and every state. Denote the value function under π_* as q_*. Then by definition

$$\pi_*(s) = \arg\max_{a \in \mathbb{A}} q_*(s, a).$$

3.1.1 Dynamic Programming

Dynamic programming methods point out that the optimal value function and optimal policy can be easily found using a loop of a two-step iteration

1. $q^{(k+1)}(s, a) \leftarrow \mathrm{E}_{\pi^{(k)}}\left[R_{t+1} + \gamma q^{(k)}(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a\right]$, $s \in \mathbb{S}$

2. $\pi^{(k+1)}(s) \leftarrow \arg\max_{a \in \mathbb{A}} q^{(k+1)}(s, a)$

Step 1 illustrates that the value function following a certain policy function could be found at the fixed point of its Bellman equation, and Step 2 points to the direction at which the rewards is improving. Therefore, the iteration constantly improves on the policy function towards the optimal.

However, in dynamic programming , we assume the best situation in a sense. By taking expectation with respect to the distribution defined by the policy π, we assume the exact knowledge of how the Markov decision process works, i.e. $p(S', R|S, A)$. That is, dynamic programming requires a complete and accurate model of the environment (Sutton & Barto, 2018, p 89). This is an assumption that is often too strong in practice when the state space alone can be so enormously big that a computation of the transition probabilities become prohibitively expensive.

In contrast, reinforcement learning methods do not require a complete and accurate model of the environment. Actually, most reinforcement learning algorithms can be seen as an approximation to dynamic programming without knowing exactly the transition probabilities.

3.1.2 Monte Carlo Method

In many cases, deriving a full picture of transition probabilities is not necessary. In most real world games, we play without knowing exactly the probabilities of how an action moves the environment from one state to another. In other words, in many cases simulating many episodes of a game is computationally more feasible than modeling the Markov decision process.

By defition, the value function following a policy is the expectation of the sum of all future rewards. If we can simulate arbitrarily many episodes of games following a policy, the value function can simply be estimated as the sample averages. Therefore, as a substitute for dynamic programming methods, Monte Carlo methods propose to find the optimal policy by looping over the following three-step iteration

1. Play one episode of the game following policy $\pi(\cdot)$. Record the sequence of realized states, actions and rewards S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, …, R_T, S_T.

2. For S_t, A_t, compute $G \equiv R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$. Update value function as a sample average

$$q^{k+1}(S_t, A_t) \leftarrow q^k(S_t, A_t) + \alpha(G - q^k(S_t, A_t))$$

3. Update policy function $\pi^{(k+1)}(S_t) \leftarrow \arg\max_{a \in \mathbb{A}} q^{(k+1)}(S_t, a)$

Monte Carlo methods demonstrates that there is no need to compute the transition probabilities if we can simulate a large sample of episodes of a game. Actually, any realized sample can be viewed as a noisy estimate of the expectation. As we accumulate more samples, noises cancel out and expectation emerges as the sample's arithmetic average.

While Monte Carlo methods offers a way to bypass the need for computing the transition probabilities, the player has to wait until the end of one game before he can update value function and improve on his policy. In many real-world and academic applications, a game seems to continue forever with no clear ending. The longer a game episode lasts, the slower the convergence is. Computational cost increases quickly in the length of a game that it prohibits the application of Monte Carlo methods in many situations.

3.1.3 Temporal Difference

Note that in dynamic programming, an expectation is evaluated at Step 1 of the iteration. As we discussed above, the expectation actually can be approximated by a body of samples. At each time step of a game, the player plays following π, and then observes a reward and the next state. Given the old value function $q^{(k)}(\cdot, \cdot)$, that is to say that he gets a new sample of $R_{t+1} + q^{(k)}(S_{t+1}, A_{t+1})|S_t, A_t$.

The new sample then can be averaged with the old samples, i.e. $q^{(k)}$, to derive a new estimate of the expectation $q^{(k+1)}$. Specifically, the algorithm can be written in the following loop. At time step t and in state S_t,

1. take action A_t following policy $\pi^{(k)}(S_t)$. Observe reward R_{t+1} and next state S_{t+1}

2. Update value function

$$q^{(k+1)}(S_t, A_t) \leftarrow q^{(k)}(S_t, A_t) + \alpha \left[R_{t+1} + \gamma q^{(k)}(S_{t+1}, A_{t+1}) - q^{(k)}(S_t, A_t)\right]$$

3. Update policy function $\pi^{(k+1)}(S_t) \leftarrow \arg\max_{a \in \mathbb{A}} q^{(k+1)}(S_t, a)$

In Step 2, our estimates of the value function is guided towards the fixed point of its corresponding Bellman equation. And in Step 3, policy function is improved one step towards the optimal policy.

So temporal difference methods draw on both Monte Carlo and dynamic programming methods. On one hand, TD methods learns from samples, instead of exact knowledge of transition probabilities, to update value and policy functions just as Monte Carlo methods do. And on the other hand, TD updates fully utilizes its old estimates of value function just as dynamic programming does. As result, updates happen in real time at each time step, without the need to wait until the game episode comes to an end. Therefore, TD methods applies to problems with infinite time horizons.

The TD class of algorithms, e.g. SARSA, Q-learning, and expected SARSA, are today the most widely used reinforcement learning methods Sutton & Barto (2018). They can be implemented online in real time with minimal computational cost. And these two attributes are extremely important in business applications. For example, a self-driving car cannot afford to wait until a whole batch of accidents data floods in before it updates. It is by nature required to be able to evolve timely. And TD reinforcement learning methods with online updates and minimal amount of computation is what make self-driving system a prospect.

3.2 Value Function Approximation

Note that in all the above algorithms, our estimate of value function $q(S, A)$ is updated pointwisely. The value of each state-action pair is computed and recorded one by one and none. Specifically, the value function estimate is in a form of an linear combination of each and every state-action pair's indicator functions. It approaches the true value function at each point as the iteration goes into the limit

$$\sum_{s \in \mathbb{S}, a \in \mathbb{A}} (\hat{q}_{s,a} \cdot \mathbf{1}_{S=s, A=a}) \longrightarrow q(S, A).$$

Although well applicable to many small games, the task of learning the values pointwisely would become intractably difficult if the state-action space $\{\mathbb{S}, \mathbb{A}\}$ is so big that we could not list all state-action pair exhaustively. For example, the number of states in a chess game is roughly in the order of 10^{46}.

More practically, the value function can be approximated using a more general functional form. The approximation does not approach the true value function at each and every point.

Let $\hat{v}(S, \boldsymbol{w})$ denote the parameterized approximation of the value function, i.e.

$$\hat{q}(S, A, \boldsymbol{w}) \approx q(S, A)$$

where $\boldsymbol{w}$ denotes a weight vector; $\hat{q}$ can be seen as an estimate of q and therefore is capped with a hat symbol by convention. We may want $\hat{q}$ approximates the value function at better accuracy in certain part of the state space than in the rest. For example, we may be more interested in the area where the states are visited most frequently. Let $u(S)$ denote a probability distribution defined on the state space that specifies which states we care more than the others. Then we can try to find (i.) a functional form $\hat{q}$ and (ii.) a vector of weights $\boldsymbol{w}$ that minimize the expected mean squared error

$$\sum_{s \in \mathbb{S}, a \in \mathbb{A}} \mu(s) \left(q(s, a) - \hat{q}(s, a, \boldsymbol{w})\right)^2$$

3.2.1 Updating Weights $\boldsymbol{w}$

Let's discuss the functional form problem in the next section. Now suppose we've already decided on what functional form we want in value function approximation. Again, the issue we want to solve is a control problem: we have a robot player that learns how to play a game. But the state space of the game is so large that the simple update of the policy function π on each and every state is not possible.

With a function approximation, the update of the value function is equivalent to the update of the weights vector $\boldsymbol{w}$. As discussed before, we want to update the weights in a direction that minimized the distance between the true and approximate value functions

$$\min_{\boldsymbol{w}} \sum_{s \in \mathbb{S}, a \in \mathbb{A}} \mu(s) \left(q(s, a) - \hat{q}(s, a, \boldsymbol{w})\right)^2 .$$

Here μ is a probability distribution that describes which part of the state space we need to be approximated more accurately. Since usually we care more about the states that are visited the most frequently, a natural choice for μ is the stationary distribution of environment state under policy π. In that case, the target we want to minimize, the mean squared error, can be viewed as an expectation under the stationary probability distribution, and therefore can be estimated by a sample average.

At time t, suppose the robot player follows a policy function π and soon observes a reward R_{t+1} and the next state S_{t+1}. According to what we discussed in Section 3.1.3, given the old estimates $\boldsymbol{w}^{(k)}$,

$$R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}^{(k)}) \tag{3.2}$$

could be treated as a noisy estimate of $q(S_t, A_t)$.

The direction where $\boldsymbol{w}$ should be updated to is the opposite of the gradient of the target

$$\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k+1)} + \alpha \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}^{(k)}) - \hat{q}(S_t, A_t, \boldsymbol{w}^{(k)}) \right) \nabla \hat{q}(S_t, A_t, \boldsymbol{w}^{(k)})$$

where $\nabla \hat{q}(S_t, A_t, \boldsymbol{w}^{(k)})$ denotes the gradient of $\hat{q}$ with respect to $\boldsymbol{w}$. Given the functional form of $\hat{q}$, the gradient is easily got.

So the robot player can learn how to best play the game in a similar loop as in Section 3.1.3:

1. take action A_t following policy $\pi^{(k)}(S_t)$. Observe reward R_{t+1} and next state S_{t+1}

2. Update value function

$$\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k+1)} + \alpha \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}^{(k)}) - \hat{q}(S_t, A_t, \boldsymbol{w}^{(k)}) \right) \nabla \hat{q}(S_t, A_t, \boldsymbol{w}^{(k)})$$

3. Update policy function $\pi^{(k+1)}(S_t) \leftarrow \arg\max_{a \in \mathbb{A}} \hat{q}(S_t, a, \boldsymbol{w}^{(k+1)})$

The robot player still learns in a real-time fashion. But note that, different from in the old TD methods, the update at time t changes not only the value function at state S_t. Since an change in $\boldsymbol{w}$ affects the value function at numerous states, the knowledge it learns at time t could be generalized to the decision under many different environment states.

Generalization is a very important attribute of human intelligence. A person who knows how to use a Windows system usually learns how to operate under MacOS quickly. Be able to generalize knowledge from one state to similar states greatly lowers the computational cost of training a robot player. With function approximation, robot playing a game that was considered too complicated for machine, e.g. the game called Go, becomes a possibility.

3.2.2 Functional Form: Supervised Machine Learning

Now we come back to the problem of how to find a good approximation functional form. This is the task of another group of machine learning method called supervised learning. Supervised learning is very different from reinforcement learning. In reinforcement learning, the core task is to find a good policy function, while in supervise learning the core task is to approximate a function.

Supervised learning learns from a set of input-target examples. For example, imagine we have a dataset of pictures where some pictures are of cat and the rest are not. We draw some examples, label each example whether it has a cat in it or not, and presented the labeled pictures to a robot. Then the robot can learn what a cat looks like using supervised learning methods and tell if a new picture contains a cat. Or suppose we have a dataset of prices and attributes of many houses. Then supervised learning can take the attributes as inputs and prices as target and estimate the price of a new house. In other words, the function from house attributes to house price can be approximated using supervised learning methods.

Linear regression models, for example, are a subset of supervised learning. Suppose given the environment state S and the robot's action A, we researchers can devise a set of features $\boldsymbol{x}(S, A)$. Each feature captures a certain attribute of the state-action pair. The robot learns how to fit these attributes in a linear regression model to predict the value function $q(S, A)$

$$\hat{q}(S, A, \boldsymbol{w}) = \boldsymbol{w}'\boldsymbol{x}(S, A)$$

The weights vector $\boldsymbol{w}$ can be learnt with the TD methods we discussed in Section 3.2.1.

The goodness of fit of the regression comes from the relevance of the features. In the example of predicting housing prices, an house appraisal expert might come up with many features that an specialist of other profession cannot, and, other things equal, the function approximation built on his features would outperform the rest. That is why expertise or called domain knowledge is extremely important for the performance of supervised learning.

Besides linear regression models, a function can be approximated by any other supervised learning methods as well. For example, trees type of models are also widely used in many applications. But again, on top of a supervised learning algorithm, feature construction is another equally if not more important job that takes a lot effort. In many applications, top performance comes from an advanced algorithm with superior theoretic properties and a set of informative features.

State S can be thought of as a collection of raw data, and features can be considered as some linear/non-linear function of those raw data. Suppose we are building a learning system that helps to predict the probability that a loan might default. State S of a loan in question might be the amount, borrow's annual income, and borrow's credit card balance. Then the borrower's debt/income ratio can be constructed from the raw data as a very informative feature. Since a linear regression does not model the nonlinear interactions between the features, features that captures the features' non-linear relations can be particularly important.

Traditionally, researcher spend huge amount of effort on feature construction. But the situation has changed dramatically since 2012 when deep neural network was introduced. Deep neural network is a new architecture of supervise learning methods. Since non-linear relationships between features can be automatically constructed by a deep neural network, emphasis shifts back towards the quality of raw data away from feature construction. For more information on deep neural network, please refer to Section 3.5 where this topic is discussed in detail.

3.3 Policy Gradient Methods

The goal of solving a Markov decision problem is to find a policy function that maximizes rewards of a game. As dynamic programming pointed out, the goal can be achieved by first finding the optimal value function and then picking the best response under the optimal value function. Temporal difference type of learning methods follows the two-step scheme laid out by dynamic programming. But note that the optimal policy found under such a strategy is always deterministic. Given a value function, there is always one action that's is the best under a state. And therefore the policy learnt this way is always deterministic.

However, in many cases a deterministic policy might not be optimal. Suppose robot player is learning to play the following game in which it moves left or right to try to reach the terminal T as soon as possible. Suppose the state space is too large that the algorithm needs to divides the space using a grid, and approximate the value function on each grid. As a result, states S_1, S_2, S_3 look all the same to the robot. He cannot tell which state he is in if the game happens to be in that state grid. It turns out that S_2 is special. In state S_2, if the robot chooses to move right, it actually moves left and vice versa. States S_1 and S_3 are normal.

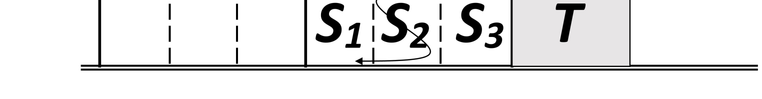


Figure 3.1: Corridor Game

Suppose the game in state S_1 right now. But again, the robot cannot tell if he is in S_1, S_2

or S_3 because of value function approximation. A deterministic policy would be a disaster in this case. A deterministic policy in state S_1 leaves us only two options: always move right or always move left. If the robot chooses to always go right, he will move to S_2 and then bounce back to S_1. So he will be trapped in that grid forever. But if he chooses to go left, he will move away from the terminal. So a deterministic policy would leave the robot trapped in his loop forever and he could never finish this game. You can imagine a cleaning robot is trapped at the corner of a room forever.

That is, with imperfect information, a deterministic policy is often not the optimal. In many applications, we cannot tell for sure in what situation we are right now at the moment, and best reaction is often to explore as many options as possible instead of sticking to one action. In the above example, the optimal policy in S_1 grid is to try right 59% of the time and left 41%. Actually any stochastic policy is better than a deterministic one.

Actually the policy function does not have to be learnt after learning the value function. Policy gradient provides a methods where the robot can directly learn the best decision from experience. Because it is not derived as the best action under a value function, the policy function learnt this way does not have to be deterministic.

3.3.1 Objective

Let $\pi(a|s, \boldsymbol{\theta})$ denote a policy function, where $\boldsymbol{\theta}$ denotes a weights vector. Suppose the functional form is known. Then different values of $\boldsymbol{\theta}$ determines the probabilities of different actions under each state. The robot wants to learn the optimal weights $\boldsymbol{\theta}^*$ such that the expected reward following policy $\pi(a|s, \boldsymbol{\theta}^*)$ can be maximized. Let μ denote the stationary distribution under policy π. Then the learning problem can be formalized as

$$\max_{\boldsymbol{\theta}} \sum_{s \in \mathbb{S}} \mu(s) \sum_{a \in \mathbb{A}} \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$$

where q_π is the value function under policy π as before. In a infinite time horizon setting, the robot's reward in the long run is maximized when the expected reward at each time step is maximized.

To maximize the objective, the direction to which $\boldsymbol{\theta}$ should be changed is given by the gradient. The derivation of the gradient seems tricky, because μ is the stationary distribution under π and is a complicated function of the weights vector $\boldsymbol{\theta}$. But Policy Gradient Theorem gives the convenient result that the gradient is simply

$$\sum_{s \in \mathbb{S}} \mu(s) \sum_{a \in \mathbb{A}} \nabla \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$$

or equivalently

$$\sum_{s \in \mathbb{S}} \mu(s) \sum_{a \in \mathbb{A}} \pi(a|s, \boldsymbol{\theta}) \left[\nabla \ln \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \right].$$

So Policy Gradient Theorem guarantees that we can just ignore the complicated dependence of μ on $\boldsymbol{\theta}$.

The gradient can be viewed as the expectation of $\nabla \ln \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$ under policy pi, and therefore can be noisily estimated using samples directly generated by playing the game following π.

3.3.2 Actor-Critic Algorithm

At each time step, the robot player plays action a following policy π. Then he receives a reward and observes the next state. He computes $\nabla \ln \pi(a|s, \boldsymbol{\theta})$ using the new data. At this point, he gets an estimate of the gradient of the objective function. Then he can update his policy function by moving the weights vector $\boldsymbol{w}$ in the direction of the estimated gradient

$$\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + \alpha^\theta \nabla \ln \pi(a|s, \boldsymbol{\theta}^\theta) q_\pi(s, a)$$

Note that the gradient of the policy function $\nabla\pi(a|s,\boldsymbol{\theta})$ gives the direction where the probability of taking action a can be increased. And $q_\pi(s,a)$ gives is the value of taking action a. Therefore, $\nabla \ln \pi(a|s,\boldsymbol{\theta}^\theta)q_\pi(s,a)$ guarantees that the actions that has higher values will be assigned higher probability after the update. So the update always moves in the direction that increases the reward in the long run.

But the problem of implementing the update is that we do not have the value function q. So actually the value functions needs to be approximated using the function approximation methods discussed in Section 3.2. With $\hat{q}$ denote the approximate value function, at each time step,

$$q_\pi(s,a) \approx \hat{q}(s,a,\boldsymbol{w}^{(k)})$$

So in a shell, the algorithm goes as follows. Each time step after receiving the reward and new state, the robot update his value function estimates and policy function

1. update value function

$$\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k+1)} + \alpha^w \left(r + \gamma\hat{q}(s',a,\boldsymbol{w}^{(k)}) - \hat{q}(s,a,\boldsymbol{w}^{(k)})\right)\nabla\hat{q}(s,a,\boldsymbol{w}^{(k)})$$

2. update policy function

$$\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + \alpha^\theta \nabla \ln \pi(a|s,\boldsymbol{\theta}^\theta)\hat{q}_\pi(s,a,\boldsymbol{w}^{(k+1)})$$

This algorithm is called the actor-critic algorithm. Here policy function is the actor and value function is the critic. The critic points out which actions' probabilities should be increased in a new policy, and the actor improves following the direction laid out by the critic.

Note that in this policy gradient method, a value function approximation is necessary just as before. But the role the value function plays in the derivation of policy function is quite different than in the past. The policy is not derived as the deterministic optimizer of the approximate value function. Instead, policy updates the probabilities of those actions that value function identifies as high-value ones. Policy is not required to be deterministic. If the functional form allows, the optimal policy is free to stay stochastic.

3.3.3 Policy Parameterization

Now we come back to the functional form of policy function $\pi(s, a, \boldsymbol{\theta})$. Similar to the problem in value function approximation, the parameterization of policy function can take various forms as long as the policy function is differential be the weights vector. Generally, the state space in an application is much larger than the action space. For example, in a self-driving car problem, the state space involves multiple types of data ranging from radar signals to videos. And different combinations of the state might calls for quite different actions. Complicated non-linear functional forms like deep neural network might have its merits. But typically the action space tends to be much smaller than the state space, and choices of actions are usually quite limited. So policy function often takes a quite simple form.

Two functional forms are frequently used. In discrete action space problems, the choice of softmax function is natural. Similar to in Section 3.2.2, we use $\boldsymbol{x}(s, a)$ denote a vector of features at state s and action a. Let $h(s, a, \boldsymbol{\theta})$ denote the preference over the actions at state s, then the softmax function

$$\pi(a|s, \boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{a \in \mathbb{A}} e^{h(s,a,\boldsymbol{\theta})}}$$

defines a probability distribution over the action space $\mathbb{A}$. Here $h(s, a, \boldsymbol{\theta})$ can take any functional form that is differentiable to $\boldsymbol{\theta}$. In contrast to the dynamic programming strategy, the distribution define this way allows for the possibility of stochastic policies. Of course, the distribution could converge to a deterministic policy in limit if the optimal policy is deterministic by nature.

In continuous action space problems, a Gaussian distribution is a natural and common choice

$$\pi(a|s,\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma(s,a,\boldsymbol{\theta}^{\sigma})} \exp\left\{-\frac{(a-\mu(s,a,\boldsymbol{\theta}^{\mu})^2}{2[\sigma(s,a,\boldsymbol{\theta}^{\sigma})]^2}\right\}$$

where the mean μ is often taken as a linear combination of features and standard deviation σ is often the exponential of another linear combination

$$\mu(s,a,\boldsymbol{\theta}^{\mu}) = \boldsymbol{\theta}^{\mu,T} \cdot \boldsymbol{x}(s,a),$$
$$\sigma(s,a,\boldsymbol{\theta}^{\sigma}) = \exp\left\{\boldsymbol{\theta}^{\sigma,T} \cdot \boldsymbol{x}(s,a)\right\}.$$

The Gaussian distribution also allows the policy function to be deterministic by degenerating and concentrating the probability mass towards a certain action.

3.4 Planning and Learning

In TD type of reinforcement learning algorithm, the robot player plays a game following a policy function. One time step later, he receives a reward and observes the new state in the environment. Then he applies the new data he collects during the time step to update his value function and policy function

$$q^{(k+1)}(s,a) \leftarrow q^{(k)}(s,a) + \alpha\left[r + \gamma q(k+1)(s',r) - q^{(k)}(s,a)\right]$$

where s, a denote the state-action pair at time t. So only the state and action that he experienced will be updated. All the other state-action pairs will stay untouched. As a result, after playing one episode of the game, only the situations he has experienced during the game will be updated. In order to master the game, the robot player needs to play many episodes.

Intuitively we would think that we could improve on the algorithm using the following scheme. Once the value estimate in state s is updated, the values of all states that could transition to state s should be updated. Clearly there is a connection between the states that are adjacent to each other, as pointed out by dynamic programming (Step 1). If the knowledge we gain at one state can be propagated to the related states, the learning processing would become much faster. The whole state space could be swept quickly, and the robot might be able to master the game in a much short period of time.

However, propagation is difficult in this environment, because the transition probabilities are unknown. Without a complete knowledge of the transition probabilities, we could not know which state-action pair leads the environment to which state, and therefore could not propagate knowledge from one state to a more general set of states - except for the states the robot has experienced, e.g. s, s'.

So in order to make propagation of knowledge practical, we need to know the transition probabilities, or at least an approximation of these probabilities

$$\hat{\mathrm{p}}(s', r|s, a) \approx \mathrm{p}(s', r|s, a)$$

Note that the robot learns part of the transition probabilities each game. During each episode of the game, he would experience many state-action pairs and observe where these state-actio pair leads the environment to. As he plays more episodes, he knows more and more of the transitions. When he looks back after say 10 games, he might be able to see that historically state s and action a leads to state s'_1 four times and s'_2 six times. That is, he will be able to build a model of the transition probabilities that get closer and closer to the true probabilities as he gains more experiences.

With $\hat{p}$, knowledge he learnt at time (s, a) in a game could be propagated back to an upper level of states, and then the further upper level, following the roadmap laid out by dynamic programming. So the robot player could learn much faster than without $\hat{p}$.

3.4.1 Dyna-TD

The name dyna-TD might come from the fact that the algorithm is a TD reinforced with dynamic programming. As we mentioned before, TD class of methods has multiple implementation, e.g. Q-learning algorithm. The dynamic programming reinforced version of Q-learning is called dyna-Q, and is widely used in many applications.

Dyna-TD, although a little bit more complicated, proceeds in a similar way to TD. At each time step t, the robot plays the game following the policy function π. Soon he receives a reward r and observes what new state s' the environment is going into. He then updates his value function and policy function as the TD algorithm describes in Section 3.1.3.

As the game goes, he has been building a model of the game at the same time. He has recorded all the transition pairs (s, a, s', r) that he has experience during this game and the games that have happened in the past. After step t, he adds the newly observed transition pair to the model. Since the model records all historic experiences, the collection of historic transition pairs will approximate the transition probabilities reasonably well eventually as he gains more experiences from playing the game.

He then simulate the game using the model he has built. He could simulate as many episodes of the game as he would like. At each time step of a simulated game, he plays under policy pi, and then arbitrarily simulate the reward and next state based on the model, or historic experiences. And after that he use the simulated data to update his value estimate and policy function. This is the dynamic programming past implemented in a sampling method.

In a sense, the dyna part of the algorithm is like a replay and post-game analysis in a basketball game. A player grows through playing. Each game the player gains experiences about the game, and eventually he will master the game after a reasonable amount of games. But if he replays and studies a game many times after each game experience, thinks of all the possibilities that could have played the game differently, he learns much more from one game experience. And therefore he might master the game with a much smaller number of game experiences.

3.4.2 Inaccurate Model

The dyna part of dyna-TD methods helps to optimize the policy function with respect to the model instead of to the real game environment. However, the model and the real environment usually will not be the same. So sometimes dyna-TD might orient the policy towards a direction that is suboptimal with respect to the environment. Since the model can only simulate the transitions it has recorded before, the policy function might be optimized in a way that the robot will only take actions he is already familiar with. Under such a policy, new territories of the state-action space will never be visited and, as a result, new information stops to feed into the model. The model and the policy function will stay suboptimal.

The problem is particularly important when the game environment is changing quickly. Feed of new information in this case is extremely important to ensure that the model stays updated and accurate. Otherwise, new features emerged in the environment might render the policy function irrelevant quickly.

The key to solving this problem is to prompts the policy function to keep exploring. One way to achieve this is to attach a bonus reward to the territory of the state-action space that has not been visited for a long time. For example, Dyna-Q+ method keep track for each state of how long has elapsed since the state has been visited in a real interaction with the real game environment. The longer time that has elapsed, the bigger the bonus reward. Under such an arrangement, the policy function will be twisted more towards environment exploration. The model will keep updating and iterating.

As new exploration from real game experiences feeds into the model, the dyna part will propagate the new information through the state space quickly. This way, the dyna part contributes the exploitation of new information, and a dyna-TD robot could quickly discover and correct and modeling error.

3.5 Deep Neural Network

The current wave of popularity of artificial intelligence is mainly driven by advancements in the fields of building up deep neural networks. Although proposed decades ago, the applicability of neural network models has been limited by the large size of parameters that need to be trained, and limit availability of data in an application.

Different architectures of deep neural networks have been proposed and applied. In its most basic form is the feedforward artificial neural network. In a feedforward ANN, data is feed into the model as inputs, propagated through the layers of neurons, and finally produces the output at approximates a target value. The layers between the input layer and the output layer are called hidden layers. Viewed as a function of the inputs, an ANN is a non-linear function of the independent variables. In theory, an ANN with a single hidden layer can approximate any continuous function on a compact region of input space to any degree of accuracy, if the size of the hidden layer is sufficiently large, known as the university approximation property Cybenko (1989).

But in practice, ANN with multiple hidden layers is usually adopted, with shallower layers capturing lower-level features and deeper layers capturing higher-level features. So adding more layers, not only helps to approximate more complex functions, but also allows for higher levels of abstraction. For example, in a typical neural network that is trained to recognize cat, shallower layers are found to be devoted to the recognition of lines of different angles while deeper layers tend to be able to compose lines into various shapes. In a sense, features are engineered automatically without human domain expertise in a neural network. And this has been proven to be particular useful in various fields of machine learning.

Other popular architectures of deep neural networks include convolutionary neural network and recurrent neural network. CNN is good at abstracting information from image data and RNN has proven to work well with time series data.

3.5.1 Recurrent Neural Network

Of particular interest to economics research is the recurrent type of architecture. As illustrated in Figure 3.2, different from feedforward neural network, RNN encodes data into neurons and then feed the neurons back into the model to derive the next batch of neurons together with a new batch of data. As a result, the information of time t will be stored in the form of neuron activations in a and affect future predictions $\hat{y}_{t+s}, s = 0, 1, \ldots$.

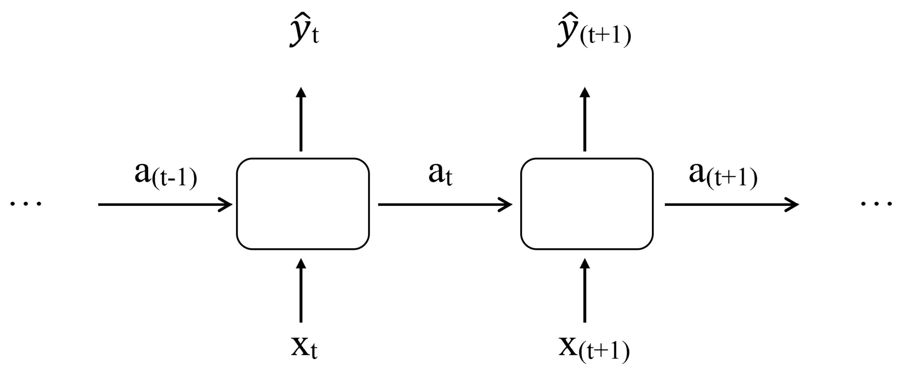


Figure 3.2: Recurrent Neural Network

The two boxes in Figure 3.2 stand for a function that takes old information and new data as inputs and outputs prediction $\hat{y}$. In conventional time series models, the function usually takes the form of a linear function. But in RNN models, the function is usually non-linear, as illustrated in Figure 3.3. In Figure 3.3, old information a_{t-1} and contemporary data x_t are encoded using a non-linear function, and then decoded to generate prediction $\hat{y}_t$. The encoder could be a tahn function or a rectified linear function. And the decoder could be a linear, logistic or a softmax function, depending on the type of output data. Overall, the function from input x_t to output y_t represented by the box in Figure 3.2 could be highly nonlinear and is highly flexible. The values of parameters in each encoder-decoder unit is determined by an optimization problem to minimize the prediction error

$$\sum_t (y_t - \hat{y}_t)^2 \tag{3.3}$$

just as the familiar OLS minimization problem. Because the function form is highly flexible

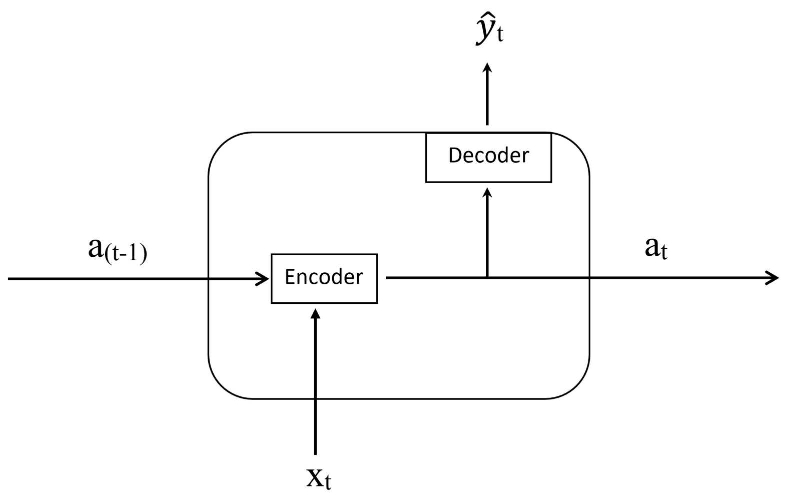


Figure 3.3: Recurrent Unit

and basically model free (not from economic theory), the input x_t is free to include any data that might help with model prediction.

The lag effect of x_t on y_t can be customized by altering the RNN function unit (the box in Figure 3.2). A standard RNN unit updates the stored information a_t at each time step and therefore tends to have short memory. Information from long in the past is lost through the information flow. Long memory can be realized by adding an update gate to the RNN unit. The update gate, as illustrated in Figure 3.4, can be a logistic function that chooses whether to update a_t using new data x_t or to keep a_t the same as a_{t-1}. Suppose a_t is n-dimensional, then the gate can be made n-dimensional so that each individual element of a_t can choose to update or not independently. As a result, the lag effect of each data series can be staggered. The RNN unit in Figure 3.4 is known as the Gated Recurrent Unit proposed in Cho et al.

(2014). Another popular RNN unit widely used is called Long-Short Memory Unit proposed by Hochreiter & Schmidhuber (1997), which can be viewed as a reinforced version of a gated recurrent unit.

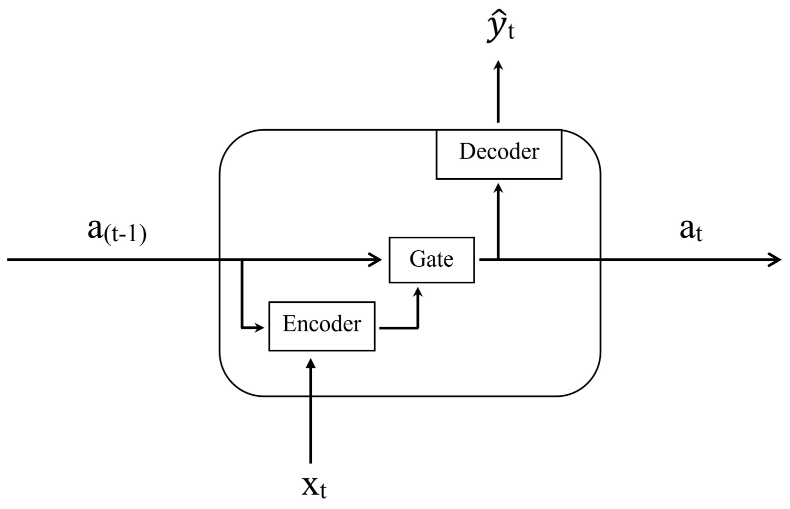


Figure 3.4: Gated Recurrent Unit

The RNN architecture as illustrated in Figure 3.2 has only one layer of recurrent units and in that sense is shallow. It can be made deeper by adding more layers of RNN units as illustrated in Figure 3.5. Additional layers help to identify more complicated relationships between x_t and y_t and in theory could improve on prediction accuracy.

When combined with reinforcement learning, the optimization problem (3.3) can be twisted slightly as discussed in Section 3.2.1. The target y_t in a reinforcement learning problem is the value function of a state-action pair and can be estimated using the temporal different method as illustrated at Equation (3.2).

3.6 Applications of Deep Reinforcement Learning

The application of reinforcement learning has been gaining momentum in the domains in financial economics, microeconomics and macroeconomics since the introduction of deep neural network methods in 2012.

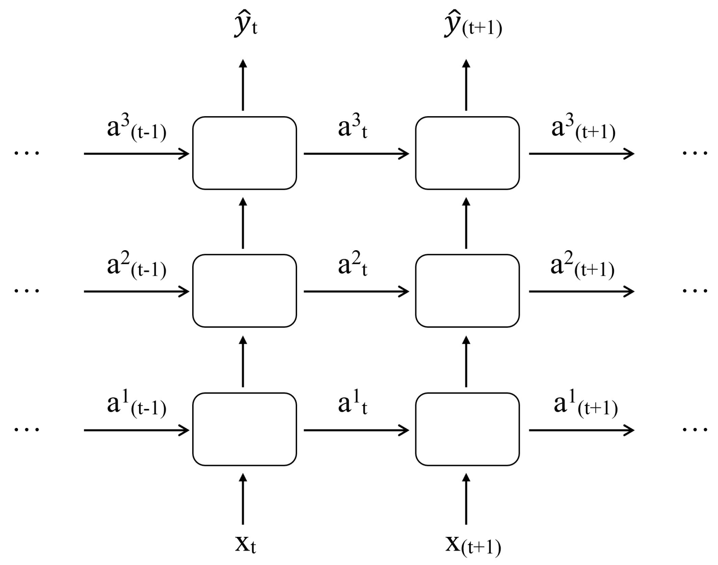


Figure 3.5: Deep Recurrent Neural Network

3.6.1 Finanial Economics

Training AI agents for automated financial asset treating is a topic that is received broad attention. Deng et al. (2016) builds a robot trader by combining recurrent neural network architecture with reinforcement learning framework. The RNN part automatically senses dynamic market conditions and creates features, while the RL part make trading decisions based on the featured constructed by the RNN model. They tested their robot on futures markets, including stock-index future markets and commodity future markets, and found its performance robust to various market conditions. They claim their robot could make reliable profits on multiple future markets.

Lu (2017) constructs their robot trader for the foreign exchange market using a similar scheme. Their build their decision making agent using a policy gradient method. Function approximation is accomplished using a long-short term memory recurrent neural network architecture. They tested the performance of the robot on GBPUSD trading.

Kanwar et al. (2019) applies deep reinforcement learning methods to explore how to optimally manage a portfolio of a given set of stocks. The robot is trained to maximize long term wealth of the portfolio. Their workhorse model is the Actor-Critic method.

Buehler et al. (2019) applies deep RL methods to a more complicated trading environment. The robot is trained to hedge a portfolio of derivatives in the presence of market frictions, e.g. transaction costs, market imperfections, liquidity constraints, or risk limits. Reward structure in their model, therefore, is nonlinear. They claim their algorithm can be implemented efficiently in high-dimensional situations. The model structure does not depend on market dynamics and can be generalized across various instruments. They find the model's computational performance is largely invariant in the size of portfolios.

In addition to trading and hedging, deep learning methods have also been applied to energy price forecasting Zhao et al. (2017), mortgage risk management Sirignano et al. (2016), stock market predictions Dixon et al. (2017), and limit order books predictions Sirignano (2019); Dixon (2018).

3.6.2 Game Theory

The connection between reinforcement learning and game theory has been noted since decades ago. Reinforcement learning features a decision-making agent in a Markov decision process. Stochastic multi-agent games, on the other hand, are a natural extension of Markov decision process into multi-agent environment. And learning behavior is crucial for multi-agent systems. Littman (1994) generalize reinforcement learning from a single agent framework into a multi-agent one. He demonstrated its application to a single two-agent game, each of which adopts a Q-learning type algorithm, and found stochastic optimal policies. Bowling & Veloso (2000) examined a number of algorithms for solving stochastic games from both the game theory and reinforcement learning communities. Tuyls & Nowé (2005) showed the remarkable similarities between reinforcement learning and evolutionary game theory, and pointed out some problems that are interesting in both fields and yet have not been solved.

Leibo et al. (2017) applied deep reinforcement learning to the study of social dilemmas. Generalized from Prisoner's Dilemma, they introduced sequential social dilemmas, where cooperativeness is a property that applies to policies instead of individual actions. Agents learn policies that implement their strategies. Each agent adopts its own deep q-network. They characterized two agents' behavior changes as a function of environment factors including resource abundance. And showed how the sequential nature of social dilemmas affects cooperation.

Tampuu et al. (2017) worked on the study of how cooperation and competition emerge between agents that learn using deep q-networks. They simulated the interaction in the well-known video game Pong. The demonstrated the evolution from competition to cooperation when the reward to cooperation is increased. They also showed that learning by playing against another agent results in more robust strategies.

3.6.3 Macroeconomics

Did neural network architecture takes a flexible model free approach that can be trained only on a large amount of data. Researches on macroeconomics using deep learning methods have been focused on variable forecasting so far. Önder et al. (2013) explored the applicability of artificial neural network for forecasting economic variables in the long run. Cook & Hall (2017) employed four different neural network architectures on predicting civilian unemployment rate. Their best performing modal build on an encoder-decoder architecture outperforms the Survey of Professional Forecasters (SPF) at every forecast horizon. They claim their approach provides good single serious performance and can incorporate novel data series.

3.7 Conclusion

In a reinforcement learning problem, an agent learns how to make automatic decisions to maximize its reward through interactions with the environment. Given the high dimensionality of the state space of many potential reinforcement learning applications, practical implementation of reinforcement learning algorithm in many cases is not possible without a deep neural network approximation. Deep reinforcement learning, which incorporates a deep neural network in a reinforcement learning algorithm, has empowered the development of advanced algorithms that have been deployed in many economic researches, ranging from financial economics to game theory. We are looking forward to seeing more applications of recurrent neural network, a particular architecture of deep neural network specializing in time-series data, in macroeconomic researches.

References

Akashi, H. & Kumamoto, H. (1977). Random sampling approach to state estimation in switching environments. *Automatica*, 13(4), 429–434.

Andrieu, C., Doucet, A., & Holenstein, R. (2010). Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3), 269–342.

Bernanke, B. S. & Blinder, A. S. (1992). The federal funds rate and the channels of monetary transmission. *American Economic Review*, (pp. 901 – 921).

Bomfim, A. N. (1997). The equilibrium fed funds rate and the indicator properties of term-structure spreads. *Economic Inquiry*, 35(4), 830–846.

Borio, C., Disyatat, P., & Rungcharoenkitkul, P. (2018). *What Anchors for the Natural Rate of Interest?* Technical report, BIS Working Paper.

Bowling, M. & Veloso, M. (2000). *An analysis of stochastic game theory for multiagent reinforcement learning.* Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science.

Buehler, H., Gonon, L., Teichmann, J., & Wood, B. (2019). Deep hedging. *Quantitative Finance*, (pp. 1–21).

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259.*

Chorin, A. J., Morzfeld, M., & Tu, X. (2010). Implicit particle filters for data assimilation. *Communications in Applied Mathematics and Computational Science*, 5(2), 221–240.

Cook, T. & Hall, A. S. (2017). Macroeconomic indicator forecasting with deep neural networks. *Federal Reserve Bank of Kansas City, Research Working Paper 17-11, September.*

Creal, D. D. & Wu, J. C. (2017). Monetary policy uncertainty and economic fluctuations. *International Economic Review.*

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4), 303–314.

Dai, Q. & Singleton, K. J. (2000). Specification analysis of affine term structure models. *The Journal of Finance*, 55(5), 1943–1978.

Del Negro, M., Giannone, D., Giannoni, M. P., & Tambalotti, A. (2017). Safety, liquidity, and the natural rate of interest. *Brookings Papers on Economic Activity*, 2017(1), 235–316.

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3), 653–664.

Dewachter, H. & Lyrio, M. (2006). Macro factors and the term structure of interest rates. *Journal of Money, Credit and Banking*, (pp. 119–140).

Dixon, M. (2018). Sequence classification of the limit order book using recurrent neural networks. *Journal of computational science*, 24, 277–286.

Dixon, M., Klabjan, D., & Bang, J. H. (2017). Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, 6(3-4), 67–77.

Douc, R., Moulines, E., & Stoffer, D. (2014). *Nonlinear time series: theory, methods and applications with R examples.* CRC Press.

Gagnon, E., Johannsen, B. K., & López-Salido, D. (2016). Family composition, life expectancy, and the equilibrium real interest rate.

Gordon, N. J., Salmond, D. J., & Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140 (pp. 107–113).: IET.

Gürkaynak, R. S., Sack, B., & Wright, J. H. (2007). The us treasury yield curve: 1961 to the present. *Journal of monetary Economics*, 54(8), 2291–2304.

Hakkio, C. S. & Smith, A. L. (2017). Bond premiums and the natural real rate of interest. *Economic Review*, 102, 5–39.

Hamilton, J. D., Harris, E. S., Hatzius, J., & West, K. D. (2016). The equilibrium real funds rate: Past, present, and future. *IMF Economic Review*, 64(4), 660–707.

Handschin, J. & Mayne, D. Q. (1969). Monte carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering. *International journal of control*, 9(5), 547–559.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.

Holston, K., Laubach, T., & Williams, J. C. (2017). Measuring the natural rate of interest: International trends and determinants. *Journal of International Economics*.

Johannsen, B. K. & Mertens, E. (2018). A time series model of interest rates with the effective lower bound. *Board of Governors of the Federal Reserve System*.

Joslin, S., Priebsch, M., & Singleton, K. J. (2014). Risk premiums in dynamic term structure models with unspanned macro risks. *The Journal of Finance*, 69(3), 1197–1233.

Joslin, S., Singleton, K. J., & Zhu, H. (2011). A new perspective on gaussian dynamic term structure models. *The Review of Financial Studies*, 24(3), 926–970.

Kanwar, N. et al. (2019). *Deep Reinforcement Learning-based Portfolio Management*. PhD thesis.

Kiley, M. T. (2015). What can the data tell us about the equilibrium real interest rate? *Board of Governors of the Federal Reserve System*.

Laubach, T. & Williams, J. C. (2003). Measuring the natural rate of interest. *Review of Economics and Statistics*, 85(4), 1063–1070.

Laubach, T. & Williams, J. C. (2016). Measuring the natural rate of interest redux. *Business Economics*, 51(2), 57–67.

Laurent, R. D. (1988). An interest rate-based indicator of monetary policy. *Economic Perspectives*, (Jan), 3–14.

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., & Graepel, T. (2017). Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (pp. 464–473).: International Foundation for Autonomous Agents and Multiagent Systems.

Lindsten, F., Jordan, M. I., & Schön, T. B. (2012). Ancestor sampling for particle gibbs. In *Advances in Neural Information Processing Systems* (pp. 2591–2599).

Lindsten, F., Jordan, M. I., & Schön, T. B. (2014). Particle gibbs with ancestor sampling. *Journal of Machine Learning Research*, 15(1), 2145–2184.

Lindsten, F. & Schön, T. B. (2013). Backward simulation methods for monte carlo statistical inference. *Foundations and Trends® in Machine Learning*, 6(1), 1–143.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.

Lu, D. W. (2017). Agent inspired trading using recurrent reinforcement learning and lstm neural networks. *arXiv preprint arXiv:1707.07338*.

Lubik, T. A. & Matthes, C. (2015). Calculating the natural rate of interest: A comparison of two alternative approaches. *Richmond Fed Economic Brief*, (Oct), 1–6.

Morzfeld, M., Tu, X., Atkins, E., & Chorin, A. J. (2012). A random map implementation of implicit filters. *Journal of Computational Physics*, 231(4), 2049–2066.

Önder, E., Bayır, F., & Hepsen, A. (2013). Forecasting macroeconomic variables using artificial neural network and traditional smoothing techniques. *Journal of Applied Finance and Banking*, 3(4), 73–104.

Rubin, D. B. (1987). The calculation of posterior distributions by data augmentation: Comment: A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The sir algorithm. *Journal of the American Statistical Association*, 82(398), 543–546.

Rubin, D. B. (1988). Using the sir algorithm to simulate posterior distributions. *Bayesian statistics*, 3(1), 395–402.

Rudebusch, G. D. & Wu, T. (2008). A macro-finance model of the term structure, monetary policy and the economy. *The Economic Journal*, 118(530), 906–926.

Sirignano, J., Sadhwani, A., & Giesecke, K. (2016). Deep learning for mortgage risk.

Sirignano, J. A. (2019). Deep learning for limit order books. *Quantitative Finance*, 19(4), 549–570.

Summers, L. H. (2014). U.s. economic prospects: Secular stagnation, hysteresis, and the zero lower bound. *Business Economics*, 49(2), 65–73.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., & Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4), e0172395.

Tuyls, K. & Nowé, A. (2005). Evolutionary game theory and multi-agent reinforcement learning. *The Knowledge Engineering Review*, 20(1), 63–90.

Wicksell, K. (1898). *Interest and Prices (Geldzins and Güterpreise).* English translation by R. F. Kahn, London: Macmillan, for the Royal Economic Society, 1936.

Woodford, M. (2003). *Interest and prices: Foundations of a theory of monetary policy.* princeton university press.

Zaritskii, V., Svetnik, V., & Shimelevich, L. (1976). Monte-carlo technique in problems of optimal information processing. *Automation and Remote Control*, 36(12), 2015–2022.

Zhao, Y., Li, J., & Yu, L. (2017). A deep learning ensemble approach for crude oil price forecasting. *Energy Economics*, 66, 9–16.

Appendix A

Appendix to Chapter 1

A.1 Macro-Finance Term Structure Model

A.1.1 Dynamics under the Real-World Probability Measure

Let $\boldsymbol{M}_t$ denote some macroeconomic variables, and $\boldsymbol{F}_t$ denote three latent variables which are priced on the Treasury securities market. Consider a macro term-structure model where the variables evolve in the following way

$$\boldsymbol{M}_t = \boldsymbol{\Phi}\boldsymbol{M}_{t-1} + \boldsymbol{\Sigma}_M\boldsymbol{\varepsilon}_{M,t}, \quad \boldsymbol{\varepsilon}_{M,t} \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{A.1}$$

$$\boldsymbol{F}_t = \boldsymbol{K}_0 + \boldsymbol{K}_1\boldsymbol{F}_{t-1} + \boldsymbol{\Gamma}\boldsymbol{M}_t + \boldsymbol{\Sigma}_F\boldsymbol{\varepsilon}_{F,t}, \quad \boldsymbol{\varepsilon}_{F,t} \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{A.2}$$

where $\boldsymbol{\varepsilon}_{M,t}$ and $\boldsymbol{\varepsilon}_{F,t}$ are independent. Equivalently, the second equation can be re-written into the following form

$$\boldsymbol{F}_t = \boldsymbol{K}_0 + \boldsymbol{K}_1\boldsymbol{F}_{t-1} + \boldsymbol{\Gamma}\boldsymbol{\Phi}\boldsymbol{M}_{t-1} + \boldsymbol{\Gamma}\boldsymbol{\Sigma}_M\boldsymbol{\varepsilon}_{M,t} + \boldsymbol{\Sigma}_F\boldsymbol{\varepsilon}_{F,t}.$$

Denote

$$\boldsymbol{\Sigma}_\nu\boldsymbol{\nu}_t \equiv \boldsymbol{\Gamma}\boldsymbol{\Sigma}_M\boldsymbol{\varepsilon}_{M,t} + \boldsymbol{\Sigma}_F\boldsymbol{\varepsilon}_{F,t}, \tag{A.3}$$

with

$$\boldsymbol{\nu}_t \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I}),$$
$$\boldsymbol{\Sigma}_\nu \boldsymbol{\Sigma}_\nu' = \boldsymbol{\Gamma}\boldsymbol{\Sigma}_M \boldsymbol{\Sigma}_M' \boldsymbol{\Gamma}' + \boldsymbol{\Sigma}_F \boldsymbol{\Sigma}_F'.$$

Then we have

$$\boldsymbol{F}_t = \boldsymbol{K}_0 + \boldsymbol{K}_1 \boldsymbol{F}_{t-1} + \boldsymbol{\Gamma}\boldsymbol{\Phi}\boldsymbol{M}_{t-1} + \boldsymbol{\Sigma}_\nu \boldsymbol{\nu}_t. \tag{A.4}$$

A.1.2 Pricing Kernel and Prices of Risks

Following Joslin et al. (2014), we assume the pricing kernel

$$\mathcal{M}_{t+1} = \exp\left\{-y_{1,t} - \boldsymbol{\lambda}_t' \boldsymbol{\nu}_{t+1} - \frac{1}{2}\boldsymbol{\lambda}_t' \boldsymbol{\lambda}_t\right\},$$

according to which only a small number of risk factors $\boldsymbol{\nu}_{t+1}$ are priced on the bond market. Note that the macroeconomic risk factors $\boldsymbol{\varepsilon}_{M,t+1}$, albeit not directly priced, span the bond-market risk factors together with some other risks $\boldsymbol{\varepsilon}_{F,t+1}$, as made clear by (A.3). The prices of risk factors $\boldsymbol{\lambda}_t$ are related to the market's risk attitude which in turn is assumed to be determined by the state of the economy and the state of the bond market

$$\boldsymbol{\lambda}_t = \boldsymbol{\lambda}_0 + \boldsymbol{\lambda}_{1M}' \boldsymbol{M}_t + \boldsymbol{\lambda}_{1F}' \boldsymbol{F}_t.$$

With $\mathcal{M}_{t+1}$ as the Radon-Nikodym derivative, we can define a new probability measure, denoted as $\mathbb{Q}$, under which the probability density function of $\boldsymbol{\nu}_{t+1}$ conditional on the current state variables $(\boldsymbol{M}_t, \boldsymbol{F}_t)$

$$\begin{aligned}
f^{\mathbb{Q}}_{\boldsymbol{\varepsilon}_{t+1}|\boldsymbol{M}_t,\boldsymbol{F}_t}(\boldsymbol{\nu}_{t+1}) &\equiv \mathcal{M}_{t+1} f(\boldsymbol{\nu}_{t+1}) \\
&\propto \exp\left\{-y_{1,t} - \boldsymbol{\lambda}_t' \boldsymbol{\nu}_{t+1} - \frac{1}{2}\boldsymbol{\lambda}_t' \boldsymbol{\lambda} t\right\} \exp\left\{-\frac{1}{2}\boldsymbol{\nu}_{t+1}' \boldsymbol{\nu}_{t+1}\right\} \\
&\propto \exp\left\{-\boldsymbol{\lambda}_t' \boldsymbol{\nu}_{t+1} - \frac{1}{2}\boldsymbol{\lambda}_t' \boldsymbol{\lambda} t - \frac{1}{2}\boldsymbol{\nu}_{t+1}' \boldsymbol{\nu}_{t+1}\right\} \\
&= \exp\left\{-\frac{1}{2}\left(\boldsymbol{\nu}_{t+1} + \boldsymbol{\lambda}_t\right)' \left(\boldsymbol{\nu}_{t+1} + \boldsymbol{\lambda}_t\right)\right\}.
\end{aligned}$$

That is, under the $\mathbb{Q}$-measure,

$$\boldsymbol{\nu}_{t+1}|\boldsymbol{M}_t, \boldsymbol{F}_t \overset{\mathbb{Q}}{\sim} \mathrm{N}\left(-\boldsymbol{\lambda}_t, \boldsymbol{I}\right).$$

Define

$$\boldsymbol{\nu}^{\mathbb{Q}}_{t+1} \equiv \boldsymbol{\nu}_{t+1} + \boldsymbol{\lambda}_t.$$

Then (A.4) can be written as

$$\begin{aligned}
\boldsymbol{F}_t &= \boldsymbol{K}_0 + \boldsymbol{K}_1 \boldsymbol{F}_{t-1} + \boldsymbol{\Gamma\Phi M}_{t-1} + \boldsymbol{\Sigma}_\nu(\boldsymbol{\nu}^{\mathbb{Q}}_t - \boldsymbol{\lambda}_{t-1}) \\
&= (\boldsymbol{K}_0 - \boldsymbol{\Sigma}_\nu \boldsymbol{\lambda}_0) + (\boldsymbol{K}_1 - \boldsymbol{\Sigma}_\nu \boldsymbol{\lambda}_{1F}')\boldsymbol{F}_{t-1} + (\boldsymbol{\Gamma\Phi} - \boldsymbol{\Sigma}_\nu \boldsymbol{\lambda}_{1M}')\boldsymbol{M}_{t-1} + \boldsymbol{\Sigma}_\nu \boldsymbol{\nu}^{\mathbb{Q}}_t.
\end{aligned}$$

Let's denote

$$\begin{aligned}
\boldsymbol{K}^{\mathbb{Q}}_0 &\equiv \boldsymbol{K}_0 - \boldsymbol{\Sigma}_\nu \boldsymbol{\lambda}_0, \\
\boldsymbol{K}^{\mathbb{Q}}_1 &\equiv \boldsymbol{K}_1 - \boldsymbol{\Sigma}_\nu \boldsymbol{\lambda}_{1F}'.
\end{aligned}$$

Suppose $\boldsymbol{F}_t$ is identified under the normalization conditions of Joslin et al. (2011) as detailed in Section A.1.3, and suppose all parameters (including $\boldsymbol{K}^{\mathbb{Q}}_0$ and $\boldsymbol{K}^{\mathbb{Q}}_1$) are given except those in $\boldsymbol{\lambda}_t$. We parameterize $\boldsymbol{\lambda}_t$ as Joslin et al. (2014) did

1. $\boldsymbol{\lambda}_0 = \boldsymbol{\Sigma}_\nu^{-1}\left[\boldsymbol{K}_0 - \boldsymbol{K}_0^{\mathbb{Q}}\right]$,

2. $\boldsymbol{\lambda}_{1F} = \left[\boldsymbol{\Sigma}_\nu^{-1}(\boldsymbol{K}_1 - \boldsymbol{K}_1^{\mathbb{Q}})\right]'$,

3. $\boldsymbol{\lambda}_{1M} = \left(\boldsymbol{\Sigma}_\nu^{-1}\boldsymbol{\Gamma}\boldsymbol{\Phi}\right)'$,

such that the dynamics of $\boldsymbol{F}_t$ under the $\mathbb{Q}$-measure does not depends on the macroeconomic variables $\boldsymbol{M}_t$

$$\boldsymbol{F}_t = \boldsymbol{K}_0^{\mathbb{Q}} + \boldsymbol{K}_1^{\mathbb{Q}}\boldsymbol{F}_{t-1} + \boldsymbol{\Sigma}_\nu\boldsymbol{\nu}_t^{\mathbb{Q}}.$$

A.1.3 Normalization

We take the normalization conditions of Joslin et al. (2011) and assume

1. the yield of a 1-month Treasury bill

$$y_{1,t} = \boldsymbol{\iota}'\boldsymbol{F}_t,$$

2. $\boldsymbol{K}_0^{\mathbb{Q}} = \left(k_\infty^{\mathbb{Q}}, 0, 0\right)'$,

3. $\boldsymbol{K}_1^{\mathbb{Q}}$ is diagonal with its eigenvalues ordered increasingly.

A.1.4 Yields

The yield and price of a zero-coupon Treasury bond that matures in m-periods, $m = 1, 2, \dots, J$.

$$y_{m,t} = \frac{1}{m}\left(A_m + \boldsymbol{B}_m'\boldsymbol{F}_t\right), \tag{A.5}$$

$$P_{m,t} = e^{-m \cdot y_{mt}},$$

where A_m and $\boldsymbol{B}_m$ are given as follows

$$A_m = A_{m-1} + \boldsymbol{B}'_{m-1}\boldsymbol{K}_0^{\mathbb{Q}} - \frac{1}{2}\boldsymbol{B}'_{m-1}\boldsymbol{\Sigma}_F\boldsymbol{\Sigma}'_F\boldsymbol{B}_{m-1}, \tag{A.6}$$

$$\boldsymbol{B}_m = \begin{pmatrix} \frac{1-\lambda_1^m}{1-\lambda_1} \\ \frac{1-\lambda_2^m}{1-\lambda_2} \\ \frac{1-\lambda_3^m}{1-\lambda_3} \end{pmatrix},$$

where $(\lambda_1, \lambda_2, \lambda_3)$ denote the three eigenvalues of $\boldsymbol{K}_1^{\mathbb{Q}}$.

Let $\boldsymbol{y}_t$ denote the set of Treasury yields of all maturities at time t. According to (A.5), the yields are an affine function of $\boldsymbol{F}_t$

$$\boldsymbol{y}_t = \boldsymbol{A} + \boldsymbol{B}\boldsymbol{F}_t, \tag{A.7}$$

$$\boldsymbol{A} = \left(\frac{A_{m_1}}{m_1}, \frac{A_{m_2}}{m_2}, \ldots, \frac{A_{m_J}}{m_J}\right)', \tag{A.8}$$

$$\boldsymbol{B} = \left(\frac{\boldsymbol{B}_{m_1}}{m_1}, \frac{\boldsymbol{B}_{m_2}}{m_2}, \ldots, \frac{\boldsymbol{B}_{m_J}}{m_J}\right)', \tag{A.9}$$

A.1.5 Observationally-Equivalent JSZ Canonical Form

Suppose the yields of three portfolios are observed with no error,

$$\boldsymbol{\mathcal{P}}_t = \boldsymbol{W}\boldsymbol{y}_t.$$

Then, according to (A.7), we have

$$\begin{aligned}\boldsymbol{\mathcal{P}}_t &= \boldsymbol{W}\boldsymbol{A} + \boldsymbol{W}\boldsymbol{B}\boldsymbol{F}_t \\ &\equiv \boldsymbol{A}_W + \boldsymbol{B}_W\boldsymbol{F}_t. \end{aligned} \tag{A.10}$$

So the factors

$$\boldsymbol{F}_t = \boldsymbol{B}_W^{-1}(\boldsymbol{\mathcal{P}}_t - \boldsymbol{A}_W). \tag{A.11}$$

Suppose $(\boldsymbol{\mathcal{P}}_t, \boldsymbol{A}_W, \boldsymbol{B}_W)$ are given, $\boldsymbol{F}_t$ can be derived accordingly. Plug (A.11) into (A.7), and we get

$$\boldsymbol{y}_t = \left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{B}_W^{-1}\boldsymbol{A}_W\right) + \boldsymbol{B}\boldsymbol{B}_W^{-1}\boldsymbol{\mathcal{P}}_t, \tag{A.12}$$

where $\boldsymbol{A}$ was given by (A.8), $\boldsymbol{B}$ was given by (A.9), and $\boldsymbol{A}_W$ and $\boldsymbol{B}_W$ were defined in (A.10).

Plug (A.11) into (A.2), and we get

$$\boldsymbol{\mathcal{P}}_t = \left(\boldsymbol{B}_W\boldsymbol{K}_0 + \boldsymbol{A}_W - \boldsymbol{B}_W\boldsymbol{K}_1\boldsymbol{B}_W^{-1}\boldsymbol{A}_W\right) + \boldsymbol{B}_W\boldsymbol{K}_1\boldsymbol{B}_W^{-1}\boldsymbol{P}_{t-1} + \boldsymbol{B}_W\boldsymbol{\Gamma}\boldsymbol{M}_t + \boldsymbol{B}_W\boldsymbol{\Sigma}_F\boldsymbol{\varepsilon}_{F,t}.$$

Let's denote

$$\begin{aligned}
\boldsymbol{K}_{0\mathcal{P}} &\equiv \boldsymbol{B}_W\boldsymbol{K}_0 + \boldsymbol{A}_W - \boldsymbol{B}_W\boldsymbol{K}_1\boldsymbol{B}_W^{-1}\boldsymbol{A}_W,\\
\boldsymbol{K}_{1\mathcal{P}} &\equiv \boldsymbol{B}_W\boldsymbol{K}_1\boldsymbol{B}_W^{-1},\\
\boldsymbol{\Psi} &\equiv \boldsymbol{B}_W\boldsymbol{\Gamma},\\
\boldsymbol{\Sigma}_{\mathcal{P}}\boldsymbol{\varepsilon}_{\mathcal{P},t} &\equiv \boldsymbol{B}_W\boldsymbol{\Sigma}_F\boldsymbol{\varepsilon}_{F,t}, \quad \text{with } \boldsymbol{\Sigma}_{\mathcal{P}}\boldsymbol{\Sigma}_{\mathcal{P}} \equiv \boldsymbol{B}_W\boldsymbol{\Sigma}_F\boldsymbol{\Sigma}_F'\boldsymbol{B}_W'.
\end{aligned} \tag{A.13}$$

And we write

$$\boldsymbol{\mathcal{P}}_t = \boldsymbol{K}_{0\mathcal{P}} + \boldsymbol{K}_{1\mathcal{P}}\boldsymbol{P}_{t-1} + \boldsymbol{\Psi}\boldsymbol{M}_t + \boldsymbol{\Sigma}_{\mathcal{P}}\boldsymbol{\varepsilon}_{\mathcal{P},t}.$$

Suppose $(\boldsymbol{K}_{0\mathcal{P}}, \boldsymbol{K}_{1\mathcal{P}}, \boldsymbol{\Psi}, \boldsymbol{\Sigma}_{\mathcal{P}}, \boldsymbol{A}_W, \boldsymbol{B}_W)$ are given, then $(\boldsymbol{K}_0, \boldsymbol{K}_1, \boldsymbol{\Gamma}, \boldsymbol{\Sigma}_F)$ can be derived according to the above four equations. Plug (A.13) into (A.6), and we get

$$A_m = A_{m-1} + \boldsymbol{B}'_{m-1}\boldsymbol{K}_0^{\mathbb{Q}} - \frac{1}{2}\boldsymbol{B}'_{m-1}\boldsymbol{B}_W^{-1}\boldsymbol{\Sigma}_{\mathcal{P}}\boldsymbol{\Sigma}'_{\mathcal{P}}\boldsymbol{B}_W^{-1\prime}\boldsymbol{B}_{m-1}. \tag{A.14}$$

A.1.6 Empirical Implementation

Firstly, we estimate the JSZ canonical model as detailed in Section A.1.5. Secondly, the latent factors model (including the pricing kernel and prices of risks) constructed in Section A.1.1 and A.1.2 can then be derived as discussed in Section A.1.5.

A.2 Bayesian Estimation

Denote the observed macroeconomic data at time t by

$$\boldsymbol{M}_t^O \equiv \left(\boldsymbol{\pi}_t, \boldsymbol{g}_t, \boldsymbol{i}_t, \tilde{\boldsymbol{x}}_t^O\right)',$$

and denote the whole macroeconomic data sample by

$$\boldsymbol{M}_{1:T}^O \equiv \left(\boldsymbol{M}_1^O, \ldots, \boldsymbol{M}_T^O\right)'.$$

Denote the principal components data sample by

$$\boldsymbol{\mathcal{P}}_{1:T} \equiv \left(\boldsymbol{\mathcal{P}}_1, \ldots, \boldsymbol{\mathcal{P}}_T\right)'.$$

Denote the Treasury yields data sample by

$$\boldsymbol{y}_{1:T} \equiv \left(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T\right)'.$$

Let $f(\cdot)$ denote a probability density function. The posterior distribution of the latent macroeconomic variables $\boldsymbol{M}_t$ and the parameters

$$f\left(\boldsymbol{M}_{1:T}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}_M, \boldsymbol{\Sigma}_{M^O}, \boldsymbol{K}_{0\mathcal{P}}, \boldsymbol{K}_{1\mathcal{P}}, \boldsymbol{\Psi}, \boldsymbol{\Sigma}_{\mathcal{P}}, k_\infty^{\mathbb{Q}}, \lambda_1, \lambda_2, \lambda_3, \boldsymbol{\Sigma}_y \left| \boldsymbol{M}_{1:T}^O, \boldsymbol{\mathcal{P}}_{1:T}, \boldsymbol{y}_{1:T}\right.\right)$$

can be simulated with the following Gibbs Sampler:

1. sample from $f(\boldsymbol{M}_{1:T}|\boldsymbol{\Phi}, \boldsymbol{\Sigma}_M, \boldsymbol{\Sigma}_{1:T}^O, \boldsymbol{K}_{0\mathcal{P}}, \boldsymbol{K}_{1\mathcal{P}}, \boldsymbol{\Psi}, \boldsymbol{\Sigma}_{\mathcal{P}})$, which is derived from a Kalman filter;

2. sample from $f(\boldsymbol{\Phi}, \boldsymbol{\Sigma}_M, \boldsymbol{\Sigma}_M^O, \boldsymbol{K}_{0\mathcal{P}}, \boldsymbol{K}_{1\mathcal{P}}, \boldsymbol{\Psi}|\boldsymbol{M}_{1:T}, \boldsymbol{M}_{1:T}^O, \boldsymbol{\mathcal{P}}_{1:T}, \boldsymbol{\Sigma}_{\mathcal{P}})$, which is a Gaussian-Inverse Wishart distribution;

3. sample from $f(\boldsymbol{\Sigma}_{\mathcal{P}}|\boldsymbol{M}_{1:T}, \boldsymbol{\mathcal{P}}_{1:T}, \boldsymbol{y}_{1:T}, \boldsymbol{K}_{0\mathcal{P}}, \boldsymbol{K}_{1\mathcal{P}}, \boldsymbol{\Psi}, k_\infty^{\mathbb{Q}}, \lambda_1, \lambda_2, \lambda_3, \boldsymbol{\Sigma}_y)$ with a Random-Walk Metropolis-Hastings algorithm;

4. sample from $f(k_\infty^{\mathbb{Q}}, \lambda_1, \lambda_2, \lambda_3|\boldsymbol{M}_{1:T}, \boldsymbol{\mathcal{P}}_{1:T}, \boldsymbol{y}_{1:T}, \boldsymbol{K}_{0\mathcal{P}}, \boldsymbol{K}_{1\mathcal{P}}, \boldsymbol{\Psi}, \boldsymbol{\Sigma}_{\mathcal{P}}, \boldsymbol{\Sigma}_y)$ with a Random-Walk Metropolis-Hastings algorithm;

5. sample from $f(\boldsymbol{\Sigma}_y|\boldsymbol{\mathcal{P}}_{1:T}, \boldsymbol{y}_{1:T}, \boldsymbol{\Sigma}_{\mathcal{P}}, k_\infty^{\mathbb{Q}}, \lambda_1, \lambda_2, \lambda_3)$, which is an Inverse-Wishart distribution.

The prior distributions of parameters are detailed in Table 1.1.

A.3 The Yield of an m-Period Bond

Let $\mathcal{M}_t$ denote a representative consumer's stochastic discount factor, and $y_{1,t}$ denote the yield of a bond that matures in one period. From the Euler equation, we have

$$1 = \mathrm{E}_t\left[\mathcal{M}_{t+1}\exp\left(y_{1,t}\right)\right].$$

Similarly, let $y_{m,t}$ denote the annualized yield of a bond that matures in m periods. Then from the Euler equation, we have

$$1 = \mathrm{E}_t \left[\prod_{i=1}^{m} \mathcal{M}_{t+i} \exp\left(m y_{m,t}\right) \right]. \tag{A.15}$$

Denote

$$\mathfrak{m}_t \equiv \log \mathcal{M}_t.$$

Then (A.15) can be rewritten into the following form

$$\exp\left(-m y_{m,t}\right) = \mathrm{E}_t \left[\exp\left(\sum_{i=1}^{n} \mathfrak{m}_{t+i} \right) \right].$$

Suppose $\mathfrak{m}_{t+i}, i = 1, \ldots, n$ has a Gaussian distribution, then we have

$$\exp\left(-m y_{m,t}\right) = \exp\left(\mathrm{E}_t \sum_{i=1}^{n} \mathfrak{m}_{t+i} + \frac{1}{2} \mathrm{Var}_t \left(\sum_{i=1}^{n} \mathfrak{m}_{t+i} \right) \right).$$

Take log on both sides, and we get

$$y_{m,t} = -\frac{1}{m} \left[\mathrm{E}_t \sum_{i=1}^{m} \mathfrak{m}_{t+i} + \frac{1}{2} \mathrm{Var}_t \left(\sum_{i=1}^{m} \mathfrak{m}_{t+i} \right) \right].$$

To see how the m-period yield is related with the future 1-period yields, rewrite the above equation in the following form

$$\begin{aligned} y_{m,t} &= -\frac{1}{m} \left[\mathrm{E}_t \sum_{i=1}^{m} \left(\mathfrak{m}_{t+i} + \frac{1}{2} \mathrm{Var}_t(\mathfrak{m}_{t+i}) \right) + \frac{1}{2} \mathrm{Var}_t \left(\sum_{i=1}^{m} \mathfrak{m}_{t+i} \right) - \frac{1}{2} \sum_{i=1}^{m} \mathrm{Var}_t(\mathfrak{m}_{t+i}) \right] \\ &= \frac{1}{m} \left[\mathrm{E}_t \sum_{i=0}^{m-1} y_{1,t+i} - \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right) \right] \end{aligned}$$

Because

$$y_{1,t} \approx i_t,$$

the yield of an m-period bond can be re-written into the following form

$$\begin{aligned}
y_{m,t} &\approx \frac{1}{m}\left[\mathrm{E}_t \sum_{i=0}^{m-1} i_{t+1} - \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right)\right] \\
&= \frac{1}{m}\left[\mathrm{E}_t \sum_{i=0}^{m-1} \left(\pi^*_{t+1} + r^*_{t+1} + \tilde{i}_{t+1}\right) - \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right)\right] \\
&= \pi^*_t + r^*_t + \frac{1}{m}\mathrm{E}_t \sum_{i=0}^{m-1} \tilde{i}_{t+1} - \frac{1}{m} \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right), \\
&= i_t - \tilde{i}_t + \frac{1}{m}\mathrm{E}_t \sum_{i=0}^{m-1} \tilde{i}_{t+1} - \frac{1}{m} \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right).
\end{aligned}$$

Therefore, we get the *slope*

$$y_{m,t} - y_{1,t} \approx -\tilde{i}_t + \frac{1}{m}\mathrm{E}_t \sum_{i=0}^{m-1} \tilde{i}_{t+1} - \frac{1}{m} \sum_{i>j;i,j=1}^{m} \mathrm{Cov}_t\left(\mathfrak{m}_{t+i}, \mathfrak{m}_{t+j}\right).$$