Asset-Price Bubbles in the Australian Market

Alcock, Jamie; Andrlikova, Petra; Aspris, Angelo; Foley, Sean; Satchell, Stephen; Segara, Reuben; Wright, Danika; Yao, Juan

CIFR
Centre for International Finance and Regulation
Towards Financial System Integrity

Research Working Paper Series

Asset price bubbles in the Australian market

Jamie Alcock
Petra Andrlikova
Angelo Aspris
Sean Foley
Stephen Satchell
Reuben Segara
Danika Wright
Juan Yao

The University of Sydney

JUNE 2016
WORKING PAPER NO.119/2016 / Project T006

This research was supported by the Centre for International Finance and Regulation , which is a Centre of Excellence for research and education in the financial sector, funded by the Commonwealth and NSW Governments www.cifr.edu.au).

Electronic copy available at: http://ssrn.com/abstract=2831806

Asset Price Bubbles in the Australian Market

Jamie Alcock, Petra Andrlikova, Angelo Aspris, Sean Foley,
Stephen Satchell, Reuben Segara, Danika Wright and Juan Yao
The University of Sydney

June 3, 2016

Executive Summary

We explore the prevalence of asset-price bubbles in Australian listed industrial equities and A-REIT markets. In contrast to the US listed stock markets, we find little evidence of asset-price bubbles in historical returns of Australian markets (1992-2016). Our findings are robust to the choice of econometric method and historical data range.

We also provide a review of the literature surrounding asset-pricing bubbles, as well as a review of the econometric identification of asset-price bubbles. In our analysis we note that significant future research is required in the econometric identification of asset-price bubbles. While the existence of asset price bubbles cannot be ruled out, significant advancements in the literature are required before academics and practitioners can gain any further insight.

Electronic copy available at: http://ssrn.com/abstract=2831806

Contents

1 Introduction

A study of market bubbles is generally considered a test of market efficiency (or inefficiency) since bubbles are concerned with rising prices that are detached from their fundamental values. Verifying the existence of such an inefficiency requires us to be able to appropriately formulate fundamental value, which typically assumes homogeneous and rational investors. Requiring additional attention is the issue of persistence. Cochrane (1991) and Chung and Lee (1998) suggest that deviations, which slowly return to fundamental values, are more indicative of a 'fad' as opposed to a bubble. As such, an additional dimension in this definition is associated with the duration of the inefficiency.

The literature has contributed a significant number of models to test for the presence of asset pricing bubbles or fads. A fad is commonly understood as a mean-reverting deviation from fair value, whereas a bubble is an explosive deviation from fair value. Among the methods most commonly adopted to detect asset-price bubbles are variance bound tests, which imply an upper bound on stock-price variability. The literature also provides for empirical tests, such as West's two-step approach which compares asset pricing models with and without bubbles. Unit root and cointegration test are also widely used to detect asset pricing bubbles. Most recently, tests in which bubbles are treated as unobservable variables have garnered significant attention. The state-space model with Markov-switching regimes is used to define the price, dividend and bubble process. The two regimes relate to bubble burst and bubble survival. The most important outcome of this model is the implied probability of bubble burst for any asset or asset class.

The presence of bubbles in asset prices has been a widely-discussed topic among theoretical economists and empirical researchers. Indeed, the two Nobel laureates in Economics for 2013 have widely differing opinions on the existence of asset-price bubbles. Eugene Fama's efficient markets hypothesis states that asset-price bubbles cannot exist because prices already include all the publicly available information.The key to the efficient markets hypothesis in rejecting the presence of asset-price bubbles is the prediction that investors can not exploit information, as the market responds immediately to new information as it arrives. If the market knew asset-prices were bubble-like it would be a profitable opportunity to take positions that would benefit from the correction, an outcome that is inconsistent with the efficient markets hypothesis.

Robert J. Shiller challenges the efficient markets hypothesis by analysing the US stock market and concludes that stock volatility is much greater than what can be explained by the rational expectations of future dividends. Shiller further extends

this observation to form the irrational exuberance argument. In this argument, the positive feedback loop drives high prices even higher as enthusiasm among investors spreads and supports the potential for bubble formation.

Asset-price bubbles assisted by substantial credit growth, very often precede financial crises (Phillips, Shi, and Yu, 2013). The global financial crisis of 2008-2009 is no exception. The creators of regulatory mechanisms are therefore focusing on controlling abnormal or excessive credit creation. The main aim of central bankers is to establish conditions for stability in financial markets and thus minimise the potential for asset-price bubbles. This can turn into an impossible task if central bankers fail to identify whether there really is a bubble in a given financial market or not. It is therefore crucial to develop an appropriate empirical framework for bubble identification.

Section 2 includes a literature review with a summary of the existing theoretical strands explaining the existence of asset-price bubbles and Section 3 discusses the existing detection techniques. Section 4 presents the results of the implementation of these bubble-detection tests in the Australian context. In Section 5, we provide a critical analysis of existing methods. Section 6 concludes.

2 Literature Review

We present the main theoretical strands explaining the presence of asset price bubbles. We start with the most standard model setting assuming rational expectations. We then focus on theories allowing for irrational market players to participate in financial markets. A short review of experimental evidence is then presented.

2.1 Rational expectations and symmetric information

In this setting, all agents are assumed to have rational expectations and the same perfect information. Tirole (1982) rules out the existence of bubbles in situations where it is known that the initial allocation is Pareto optimal. If everyone knows that the current allocation is Pareto optimal, no individual would be willing to buy an overpriced asset because it would make him worse off. No trade thus occurs at a price higher than the fundamental value and bubbles cannot emerge.

2.1.1 Gordon growth model

The Gordon growth model implies the following relation between the price and dividend values

$$\frac{p_t}{d_{t-1}} = \frac{1}{r-g}, \tag{1}$$

where r is the rate of return and g is the expected growth of earnings from dividends. In this model, bubbles can emerge with high expected growth of earnings or low returns. When applied to the Tech bubble on the US stock market, Cochrane (2002) concludes that earnings growth would have to be extremely large to justify the empirical evidence of large price to dividend ratios. Low return environments can also lead to bubble-type behaviour of asset prices. It is however hard to theoretically explain why the market risk premium would suddenly drop in times of bubbles occurrence to allow for low returns. This theory also does not explain other observed phenomena such as the high volumes traded during bubbles, the scarcity of shares or short-sales restrictions.

If we substitute d_{t-1} from equation (1) for $d_{t-1} = e_{t-1}(1+g)PR_{t-1}$, where e_{t-1} are earnings per share and PR_{t-1} is the payout ratio, we receive the price to earning (PE) relation:

$$\frac{p_t}{e_t} = PE_t = \frac{(1+g)PR_{t-1}}{r-g}. \tag{2}$$

Assuming that the retained earnings are reinvested at rate r, or that $g = (1 - PR_t)r$ and the payout ratio is stable over time, and substituting into equation (2)

leads to,

$$\frac{p_t}{e_t} = PE_t = \frac{(1+(1-PR)r)PR}{r-(1-PR)r} = \frac{1+(1-PR)r}{r} = \frac{1+g}{r}, \tag{3}$$

for $r \neq 0$.

It thus implies that

$$r\frac{p_t}{e_t} = 1+g. \tag{4}$$

By substituting $r = r_f + \lambda$, we get $(r_f+\lambda)\frac{p_t}{e_t} = 1+g$, where λ is the market price for risk and r_f is the risk free rate. Since $\lambda \geq 0$ and $g \geq 0$, it follows that

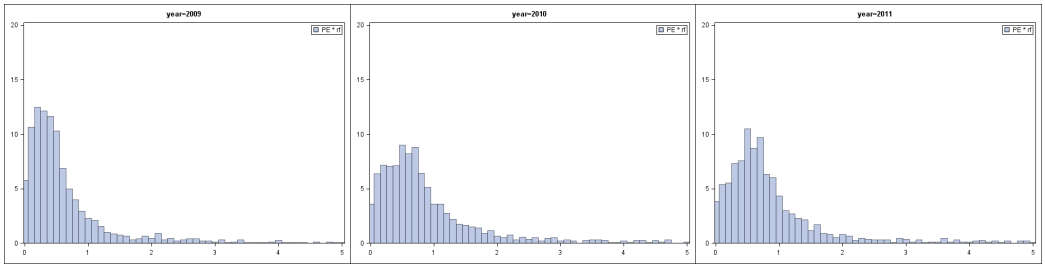
$$r_f\frac{p_t}{e_t} \leq 1. \tag{5}$$

If the inequality does not hold, it may be a sign of the existence of bubble. As we can see from Figure 1, a large portion of Australian listed firms do not satisfy this condition and have $r_f\frac{p_t}{e_t} > 1$.

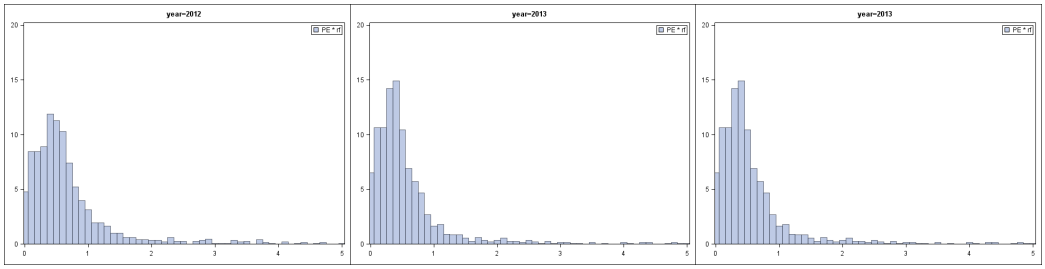
2.2 Rational expectations and asymmetric information

Grossman (1976) states that with any positive cost of obtaining information, and prices which fully reveal information, market equilibrium would never exist since there would be no additional benefit of paying for information, if value can be observed directly from prices. Individual traders do not personally benefit from having inside information in a fully revealing equilibrium. Individuals can only benefit from private information if price is only partly revealing this information. There has to be another source of uncertainty or noise so that market agents cannot infer the information from prices directly. This implies that theoretically bubbles can exist, since in equilibrium prices cannot include all the information about fundamental value and the presence of bubble does not have to be commonly known. Even if agents are aware of the presence of a bubble, they do not know whether the other agents know about the bubble or not. If agents have asymmetric information available, Allen et al. (1993) argue that bubbles can emerge under the following conditions:

1. prices do not fully reveal information
2. investors are constrained from selling in at least one asset
3. it is not common knowledge that the initial allocation is Pareto efficient.



(a) 2009 (b) 2010 (c) 2011



(d) 2012 (e) 2013 (f) 2014

Figure 1: Histogram of $PE_t * r_f$ of Australian listed companies in years 2009-2014. Price to earnings ratio (PE_t) is retrieved from DataStream, the 90-day bank accepted bill rate is used as the risk free rate (r_f) published by the Reserve Bank of Australia.

2.3 Heterogeneous beliefs and short-sales constraints

We continue with theoretical setting with rational expectations, in which agents start with different prior distributions of value. This is in contrast to the asymmetric information model, where agents have the same prior distribution but a noisy price signal. Investors with different prior distribution will agree to disagree about the asset value (Brunnermeier, 2008). In the presence of short selling constraints and heterogeneous beliefs, bubbles can emerge because pessimists are constrained from reducing the price to its fundamental level. This view is also supported by experimental evidence. Kirman and Teyssiere (2005) conduct an experiment, in which agents with heterogeneous beliefs are given different forecasting tools, which form their expectations. If one particular forecasting tool turns out to be more successful, more people will use it and if such tool is self-reinforcing, price will deviate from the fundamental value and a bubble will occur. They argue that bubbles can exist while traders are rational.

Scheinkman and Xiong (2003) provide evidence that heterogeneous belief bubbles are accompanied by large traded volumes and high stock price volatility. If only some advisors understand new technological advances and recognise heterogeneity in advisors, bubbles arise (Hong et al., 2008). Cheng et al. (2012) find that price efficiency is improved with increased short selling after removing short-sale constraints on IPO stocks in the Taiwanese market.

2.4 Rational and behavioural traders

2.4.1 Institutional herding

Rational institutional investors are expected to take advantage of arbitrage opportunities and drive the price to equilibrium, however, there are always limits of arbitrage.

This theoretical explanation of asset-price bubbles is also referred to as the limited arbitrage theory (Brunnermeier, 2008). In this model, rational sophisticated traders interact with behavioural traders, who can be influenced by certain biases. Under the efficient market hypothesis, bubbles cannot exist since rational traders will trade against the bubble before it is allowed to emerge. There are however limits to this arbitrage theory, which prevent the rational traders from correcting the price. Brunnermeier (2008) lists the following limitations.

1. fundamental risk emerges due to the fact that perfect substitutes rarely exist, which makes short positions more risky

2. noise trader risk, which might push the price in a different direction, widening the mispricing

3. synchronisation risk, as the coordination of rational traders is required to influence the price.

Due to these limitations, rational traders may not trade against the mispricings aggressively enough, allowing bubbles to form. Abreu and Brunnermeier (2002) reveal that rational traders may not aim to correct prices, rather preferring to "ride the bubble". Empirical evidence from hedge funds, which are assumed to represent rational traders, is in line with the "riding the bubble" hypothesis (Brunnermeier and Nagel, 2004). Brokers and banks tend to employ "chartists" as well as "fundamentalists". The question remains whether this "chartism" or other form of technical analysis has any effect on market efficiency (Bowden, 1990).

DeMarzo, Kaniel, and Kremer (2008) argue that when an asset is in limited supply the rational agents care about their relative wealth. They may not wish to

trade against the crowd. They may just invest in an asset that is commonly hold by others due to the concerns of taking risk alone.

Mitchell and Pulvino (2012) examine the arbitrage strategies of hedge funds over 2008 financial crisis in particular. They argue that due the difficulty of access to debt financing by hedge funds during the crisis period, instead of trading in the direction to force prices of similar securities to converge, arbitrageurs had to liquidate existing positions, thus causing increased divergence of prices in the market. As a result, mispricing in the market persisted for months.

There is extensive empirical evidence showing that money managers tend to trade excessively in the direction of the recent trades of other managers (see Grinblatt, Titman, and Wermers (1995); Lakonishok, Shleifer, and Vishny (1992)). Sias (2004) documents evidence that institutional investors are attracted to securities with the same characteristics. They tend to follow their own and other trades changes overtime. Moreover, institutional investors are more likely to follow similarly classified institutional investors than differently classified institutional investors.

Money managers mimic actions of each other in order to preserve their reputation and/or compensation. Herding behavioural of investors and financial professionals can be either rational or non-rational. Dass, Massa, and Patgiri (2008) argue that apart from the reputational concerns, mutual fund managers are driven to herd when their performance are evaluated relative to their peers. Thus asset price bubbles are caused by herding among traders.

Dasgupta, Prat, and Verardo (2011) further investigate the price impact of institutional herding. Based on model of Scharfstein and Stein (1990), they incorporate the interactions among three classes of traders: career-concerned fund managers, profit-motivated proprietary traders and security dealers endowed with market power. Their model shows that the reputational concerns of fund managers give rise to an endogenous tendency to imitate past trades, which impacts the prices of the assets they trade.

Brown, Wei, and Wermers (2013) argue that analyst recommendation changes induce mutual funds mangers herd into stock with consensus sell-side analyst updates, and out of stocks with consensus downgrades. The herding behaviour is stronger for managers with career concerns. Herding in institutional traders and money managers destabilised markets and cause asset prices to deviate from the equilibrium and thus partially explain bubble formation.

2.4.2 Herding and high-frequency traders

The emergence of high-frequency traders as modern market makers has caused considerable concern, for whilst they may bring additional liquidity, particularly during calm periods, their lack of explicit quoting obligations may increase volatility when the market experiences stressful periods. This can lead to situations such as the May 6th flash crash in 2010, which saw around 1 trillion USD wiped off the market value of the Dow Jones, which authors such as Dichev, Huang, and Zhou (2014) attribute to short-term herding combined with the withdrawal of high-frequency market makers. This withdrawal thinned the limit order book, resulting in extremely low liquidity and high volatility.

While the main argument for high-frequency traders interaction in the market is the additional liquidity that they provide, which studies such as Hendershott, Jones, and Menkveld (2011) and Menkveld (2013) have shown reduces transactions costs through lower bid-ask spreads. Generally, reduced transactions costs are associated with reduced volatility. However, liquidity above some threshold may not always be beneficial. Indeed, there is some evidence that increased liquidity - or trading volume - while appearing to reduce volatility at the level of an individual stock, may amplify tail risk on a larger scale.

Dichev et al. (2014) examine the relationship between higher trading volumes and stock-level volatility, finding that higher trading volumes can produce "its own volatility above and beyond that based on fundamentals". This is pertinent to the study of bubbles, as the thinning of liquidity in stressful periods may amplify the extremes to which financial markets move. Haldane (2011) emphasizes the danger of normalising deviance at the micro level, concluding that "thinner technological slices may make for fatter market tails. Flash Crashes, like car crashes, may be more severe the greater the velocity."

As modern capital markets have evolved, the level of high-frequency trading has grown substantially. This growth is heavily concentrated in a small number of firms. A 2015 report by ASIC[1] identified that over 27% of total equity trading volume was conducted by less than 0.5% of high-frequency traders. This level of concentration is one of the factors Sornette and Von der Becke (2011) cite as indicative of increased volatility. This change in the landscape of the Australian capital markets increases the importance of testing explicitly for the existence of asset price bubbles.

[1] Review of high-frequency trading and dark liquidity

2.5 Irrational valuation

Shiller (2000) suggests that this behaviour is irrational and suggests that irrational exuberance is the psychological basis of a speculative bubble. This is consistent with the evidence found by Coates (2012) in the neuroscience field, who finds that bull and bear markets affect the brain chemistry of traders, which can lead to the exacerbation of financial booms and busts.

Shiller (2015) defines speculative bubbles in the context of irrational valuation:

"I define a speculative bubble as a situation in which news of price increases spurs investor enthusiasm, which spreads by psychological contagion from person to person, in the process amplifying stories that might justify the price increases and bringing in a larger and larger class of investors, who, despite doubts about the real value of an investment, are drawn to it partly through envy of others successes and partly through a gamblers excitement."

It is however hard to believe that all the investors including experienced fund managers would trade without any rationality. If this was the case, it would imply that no asset pricing model would work. An asset pricing framework allowing for irrational patterns in human behaviour is yet to be introduced. Some of the cognitive biases observed by psychologists are discussed in Ritter (2003).

2.6 Convenience yield

Cochrane (2002) links the evidence of short-sales constraints, high dispersion of opinion, relative scarcity of shares, high traded volumes and high volatility that occurs during bubble periods in the US and concludes that bubbles existence can be explained by the convenience yield. Taking the example of 3Com and Palm share prices, he demonstrates that the bubble component or the price difference of fundamentally identical instruments can be explained by the liquidity premium. It was cheaper to buy Palm shares implicitly by buying 3Com than making a direct investment in Palm shares although both strategies would lead to the same outcome. The Palm shares were overpriced due to the high turnover, low-share supply, arbitrage and short-sales constraints and a poor correlation of the two share prices in short term.

Cochrane (2002) relates the argument of convenience yield to the money-market. People hold a smaller amount of cash for their own consumption despite the fact that they are losing the interest relative to holding government bills. It is convenient to hold some amounts of cash to be able to make transactions in the period between the trips to the banks to ensure liquidity. Palm shares provided this liquidity since the traded volumes and turnover each day were high relative to 3Com.

Hong et al. (2006) provide empirical evidence consistent with the convenience yield theory by analysing the relation between number of tradeable shares and speculative bubbles. They conclude that investors with heterogeneous beliefs who face short-sale constraints trade stocks with limited float because of insider lockup. The turnover and volatility decrease with asset float and price drop at the expiration date.

2.7 Experimental evidence

Investigating the price dynamics and how bubbles form has also been investigated in the experimental literature. A distinct advantage of investigating this issue in a laboratory setting is the ability to control the assets fundamental price. One of the first experimental studies in this area was conducted by Smith, Suchanek, and Williams (1988). In this experiment, participants are provided with both an asset and cash and are not allowed to short-sell. The traders are free to trade the asset that pays a dividend over the length of the experiment, which consist of 15 periods, each lasting a maximum of 240 seconds. The fundamental price of the asset at each period is equal to the present value of the future expected dividends. In the learning to optimise experimental design set up by Smith et al. (1988), traders are motivated to optimise their profits. The traders endowment is equal to the sum of the capital gains/losses from trading and dividends earned. The experiment reveals after the initial period, the asset price substantially increases above the fundamental value and crashes towards the end of the experiment. An interesting finding is that price bubbles appear frequently, especially if traders are less experienced. Using a similar experimental design, Dufwenberg, Lindqvist, and Moore (2005) address whether traders experience can help prevent the formation of bubbles. They find evidence to suggest that that when the fraction of experienced traders is only one-third, the incidence of bubbles can be eliminated or substantially reduced. Mixed evidence in the experimental literature is found when short sale constraints are relaxed. In contrast to Haruvy and Noussair (2006); Ackert et al. (2002) find that the ability short sell makes experimental markets more efficient, thereby preventing the formation of bubbles.

In a complementary learning to forecast experiment design to Smith et. al., subjects are rewarded according to their forecasting accuracy (see Marimon, Spear, and Sunder (1993)). Hommes et al. (2008) present an experiment focused on the expectation formation of participants. The task is to predict the next price of an asset when there is no knowledge about the underlying market equilibrium equation, but there is perfect knowledge of the dividend generating process. Participants are

therefore allowed to construct their rational expectations of fundamental price. In this experimental set up, bubbles emerge. This can be explained by the so-called positive-feedback expectations, when participants seem to extrapolate the trend in observed asset prices into the future. It demonstrates that even in situation with symmetric information, bubbles exist. With an increasing price, participants predict a further increase in price, which leads to further increase in prices and the emergence of a bubble.

Blanchard and Watson (1982) refer to this as the "deterministic bubble" (p.4). He points out, however, that to be rational the deterministic bubble would need to continue into perpetuity, making such an outcome implausible.

To summarise, there are several theoretical views that are able to explain the existence of asset-price bubbles on stock markets. While most of the theories focus on the question of whether bubbles can or cannot exist, few studies have sufficiently explained why and when bubbles emerge and burst (Brunnermeier, 2008).

3 Methodology

There are many existing empirical tests for asset-price bubbles. These include variance bounds tests, which imply an upper bound on stock-price variability. If this upper bound is exceeded, a bubble may be present. The next type of asset-price bubbles test are the West's two-step tests, which test for the presence of bubbles directly by comparing the empirical models with and without bubbles. If the estimates from the two models are equal, there is no bubble present. The unit root and cointegration tests are the most widely used tests for bubbles detection. If price and dividend process are cointegrated and thus include the same source of randomness, there is no bubble in asset prices. The last type of tests discussed in this paper are tests, in which bubbles are treated as unobservable variables. The state-space model with Markov-switching regimes is used to define the price, dividend and bubble process. The two regimes relate to bubble burst and bubble survival. The most important outcome of this model is the implied probability of bubble burst for any asset or asset class.

Before we describe each test in a greater detail, we set up the empirical design consistent across all testing procedures. We start by rearranging the formula for net returns to get the expression for current price, where p_t is the price at time t, d_t is the dividend paid at time t and r_t is the net return.

$$p_t = E_t \left[\frac{p_{t+1} + d_{t+1}}{1 + r_{t+1}} \right], \tag{6}$$

Moreover, using the law of iterated expectations and assuming constant rate of return, we can see that the current price equals the expected discount value of all future dividends and the expected discounted price of the asset at maturity date.

$$p_t = E_t \left[\sum_{i=1}^{T-i} \frac{d_{t+i}}{(1+r)^i} \right] + E_t \left[\frac{p_T}{(1+r)^{T-t}} \right] \tag{7}$$

For assets with a finite maturity, the price at maturity, p_T, is zero, as no further dividends are paid after maturity. For assets with infinite maturity, the present value of price in infinity is also equal to zero due to the discounting effect

$$\lim_{T \to \infty} \frac{P_T}{(1+r)^{T-t}} = 0, \tag{8}$$

which is also known as the transversality condition. If the transversality condition does not hold, the asset can include a bubble component and $p_t = p_t^f + b_t$, where p_t^f is

the fundamental value component of the asset price and b_t is the bubble component of the asset price. This implies that under certain conditions, bubbles can emerge under the rational expectations and symmetric information setting.

The empirical tests for the existence of asset-price bubbles in financial markets is based on the equation (2), which can be rewritten in the following manner, where B_t is the bubble component. The asset price includes two components, the market fundamental component, which is the discounted value of all future expected dividends, and the bubble component. In the no bubble case, the transversality condition is satisfied and $B_t = 0$.

$$p_t = \sum_{i=1}^{T-i} \frac{E_t(d_{t+i})}{(1+r)^i} + B_t \tag{9}$$

If not stated otherwise, the following assumptions are required in the empirical tests for bubble identification:

1. there are no informational asymmetries
2. the agents are risk neutral
3. the risk free rate is constant
4. the dividend generating process in not expected to change.

The next subsections describe the individual empirical tests used to detect asset-price bubbles on financial markets.

3.1 Variance Bounds Tests

The Shiller (1981b) variance bounds test for equity prices evaluates market efficiency by testing the null hypothesis of the market fundamental solution to equation (6). The ex post rational price can be expressed as the present value of actual dividends:

$$p_t^* = \sum_{i=1}^{T-i} \frac{d_{t+i}}{(1+r)^i} \tag{10}$$

Assuming rational expectations, the difference between expected and actual dividends is a mean-zero variable (Gürkaynak, 2008):

$$p_t^* = \sum_{i=1}^{T-i} \frac{E_t(d_{t+i}) + \epsilon_t}{(1+r)^i} = p_t + \sum_{i=1}^{T-i} \frac{\epsilon_t}{(1+r)^i}, \tag{11}$$

where ϵ_t is assumed to be uncorrelated with the information available at time t, the variance of the ex-post price p_t^* can therefore be defined as

$$V(p_t^*) = V(p_t) + \varphi V(\epsilon) \geq V(p_t), \tag{12}$$

where $\varphi = [1/(1+2)^2)]/[1 - 1/(1+r)^2]$. Equation (12) creates an upper bound on the variance of the asset prices. If the equity prices violate the variance bound, it implies that prices do not follow equation (10).

The usage of this technique to detect bubbles is however problematic since this test was originally constructed to test the present value model and not the presence of bubbles in prices. Kleidon (1986) shows that data simulated from the present value model violate the variance bound without the presence of a bubble if the time-series of variances is non-stationary. This is due to the fact that the variance should be a cross-sectional point-in-time measure but the variances from the equation (12) are time-series estimates, which creates a problem. The variance bounds tests are therefore not suited for detecting bubbles in asset prices but can serve as a good monitoring tool for a successful bubble identification.[2]

3.2 West (1987) two-step tests

The West two-step tests examine the presence of bubbles by testing the hypothesis of the absence of a bubble and the bubble existence sequentially. West's test is based on the observation that with the absence of bubbles in asset prices, the Euler equation of the no-arbitrage asset pricing can be estimated and implies values of the discount rate. Assuming that dividends follow an autoregressive (AR) process with known parameters and discount rate, the relationship between the market fundamental asset values and dividends can be obtained.

Under the null hypothesis, the estimates model coefficients for the no-bubble and the existence of a bubble hypothesis are equal. If the estimates differ, it leads to the conclusion of model misspecification or bubbles. Model misspecification can then be ruled out by applying a specification test to the Euler equation and the AR dividend process. If the model is not misspecified and the two estimates are not equal, the only possible explanation is the existence of a bubble.

The Euler equation is the rearranged formula for the net return given that expectations are based on an information set Ω_t:

$$p_t = \frac{1}{1+r} E_t \left(p_{t+1} + d_{t+1} | \, \Omega_t \right). \tag{13}$$

[2]Further discussion on the suitability of this method is provided in Gürkaynak (2008).

Equation (9) can be tested using the regression in the following form:

$$p_t = \frac{1}{1+r}\left(p_{t+1} + d_{t+1}\right) + u_t. \tag{14}$$

If dividends follow a stationary AR(1) process

$$d_t = \phi d_{t-1} + u_t^d, \tag{15}$$

the market fundamental price can be expressed as

$$p_t^f = \sum_{i=1}^{T-i} \frac{E_t(d_{t+i}|\Omega_t)}{(1+r)^i} = \overline{\beta} d_t, \tag{16}$$

where

$$\overline{\beta} = \frac{\frac{\phi}{1+r}}{1-\frac{\phi}{1+r}}.$$

The actual price can contain a bubble.

$$p_t = \beta d_t + B_t. \tag{17}$$

If there is a bubble present in the asset prices, the estimate of β from equation (17) will be biased. This type of test can only detect bubbles that are correlated with dividends. West (1987) applies this logic on level as well as differenced data to account for non-stationarity of dividend and price processes. He also assumes an $AR(q)$ price and dividend process, where q is determined to fit the data.

3.2.1 Estimation technique

West (1987) proposes to test the presence of speculative bubbles using level and differenced data. For the sake of simplicity, we will use the notation based on level of the price and dividend process.

Step 1: Determine order of the AR process

The first step of the West (1987) test involves the identification of the ARIMA order q of the dividend process. The identification is based either on arbitrarily selected $q = 4$ or identification based on e.g. the Akaike information criterion.

Step 2: Estimate regression coefficients using OLS and 2SLS

After we determine the order q, we get to the estimation of the parameters from the trivariate system of equations:

$$p_t = b(p_{t+1} + d_{t+1}) + u_t, \tag{18}$$

which follows from equation (14), where $b = 1/(1+r)$.

Dividends follow an AR(q) process

$$d_{t+1} = \mu + \phi_1 d_t + ... + \phi_q d_{t-q+1} + v_{t+1}, \tag{19}$$

where $\phi_1, ..., \phi_q$ are AR coefficients and v_{t+1} is the error term.

Prices also follow an AR(q) process

$$p_{t+1} = m + \delta_1 d_t + ... + \delta_q d_{t-q+1} + w_{t+1}, \tag{20}$$

where $\delta_1, ..., \delta_q$ are AR coefficients and w_{t+1} is the error term.

The coefficients from equations (19) and (20) are estimated using OLS. The coefficient b from equation (18) is estimated using instrumental variable approach based on two-stage least-square (2SLS) method. Equation (18) can be also expressed as $p_t = bX_{t+1} + u_t$, where $X_{t+1} = p_{t+1} + d_{t+1}$. The vector X_{t+1} is estimated using information available at time t in the first stage of the 2SLS method

$$X_{t+1} = \alpha D_t + \epsilon_{t+1} \tag{21}$$

In the second stage, we use the fitted values of $\hat{X}_{t+1}$ and estimate

$$p_t = b\hat{X}_{t+1} + u_t. \tag{22}$$

Step 3: Apply GMM to get the variance-covariance matrix of the trivariate system

We now have all the estimates from the trivariate system of equations, which is in total $2q + 3$ coefficients. West (1987) further applies the Hansen (1982) method of moments estimator, $g_T(\theta) = \frac{1}{T}\sum h_t(\theta)$.

Let $D_t = (1, d_t, ..., d_{t-q+1})'$ be the vector of dividends. For the estimated vector of parameters $\hat{\theta}$, the moment equations satisfy the orthogonality conditions from the trivariate system of equations

$$0 = \frac{1}{T}\sum h_t(\hat{\theta}) = \frac{1}{T}\sum \begin{pmatrix} D_t(p_t - X_t\hat{b}) \\ D_t(d_{t+1} - D_t'\hat{\phi}) \\ D_{t+1}(p_{t+1} - D_{t+1}'\hat{\delta}) \end{pmatrix}. \tag{23}$$

Since $E(h_t(\theta^*))$, where θ^* is the true but unknown θ, Hansen (1982) show that $\sqrt{T}(\hat{\theta}) \sim N(0, (F_T' S^{-1} F_T)^{-1})$, where F_T is the $(3q+3) \times (2q+3)$ matrix of partial derivatives of the moment equations with respect to the vector of parameters θ, $F_T = \frac{\partial h_t(\theta)}{\partial \theta}$ and S is the $(3q+3) \times (3q+3)$ covariance-variance matrix of the orthogonality conditions,

$$S = E(h_t(\theta) h_t(\theta)') = \frac{1}{T} \sum \begin{pmatrix} E(D_t(p_t - X_t\hat{b})(p_t - X_t\hat{b})' D_t') \\ E(D_t(d_{t+1} - D_t'\hat{\phi})(d_{t+1} - D_t'\hat{\phi})' D_t') \\ E(D_{t+1}(p_{t+1} - D_{t+1}'\hat{\delta})(p_{t+1} - D_{t+1}'\hat{\delta})' D_t') \end{pmatrix}.$$

There is in total $3q+3$ equations in the orthogonality conditions matrix h_t and $2q+3$ coefficients to be estimated.

The first part of the orthogonal conditions $D_t(p_t - X_t\hat{b})$ can be expressed as $\begin{pmatrix} 1 \\ d_t \\ \vdots \\ d_{t-q+1} \end{pmatrix} \times (p_t - X_t\hat{b}) = \begin{pmatrix} p_t - X_t\hat{b} \\ d_t(p_t - X_t\hat{b}) \\ \vdots \\ d_{t-q+1}(p_t - X_t\hat{b}) \end{pmatrix}$, which is a $(q+1)$ vector with one coefficient $(\hat{b})$.

The second part of the orthogonal conditions $D_t(d_{t+1} - D_t'\hat{\phi})$ can be expressed as

$$\begin{pmatrix} 1 \\ d_t \\ \vdots \\ d_{t-q+1} \end{pmatrix} \times (d_{t+1} - \hat{\mu} - \phi_1 d_t + \cdots \phi_q d_{t-q+1}) = \begin{pmatrix} d_{t+1} - \hat{\mu} - \phi_1 d_t + \cdots \phi_q d_{t-q+1} \\ d_t(d_{t+1} - \hat{\mu} - \phi_1 d_t + \cdots \phi_q d_{t-q+1}) \\ \vdots \\ d_{t-q+1}(d_{t+1} - \hat{\mu} - \phi_1 d_t + \cdots \phi_q d_{t-q+1}) \end{pmatrix},$$

which is a $(q+1)$ vector with $(q+1)$ coefficients $(\hat{\mu}, \hat{\phi}_1, ..., \hat{\phi}_q)$.

Finally, the third part of the orthogonal conditions $D_{t+1}(p_{t+1} - D_{t+1}'\hat{\delta})$ can be expressed as

$$\begin{pmatrix} 1 \\ d_t \\ \vdots \\ d_{t-q+1} \end{pmatrix} \times (p_{t+1} - \hat{m} - \delta_1 d_t + \cdots \delta_q d_{t-q+1}) = \begin{pmatrix} p_{t+1} - \hat{m} - \delta_1 d_t + \cdots \delta_q d_{t-q+1} \\ d_t(p_{t+1} - \hat{m} - \delta_1 d_t + \cdots \delta_q d_{t-q+1}) \\ \vdots \\ d_{t-q+1}(p_{t+1} - \hat{m} - \delta_1 d_t + \cdots \delta_q d_{t-q+1}) \end{pmatrix},$$

which is a $(q+1)$ vector with $(q+1)$ coefficients $(\hat{m}, \hat{\delta}_1, ..., \hat{\delta}_q)$.

Step 4: Test the presence of bubbles

The test for the presence of bubbles is constructed based on the relationship between the estimated parameters $\hat{\theta} = (\hat{b}, \hat{\mu}, \hat{\phi}_1, ..., \hat{\phi}_q, \hat{m}, \hat{\delta}_1, ..., \hat{\delta}_q)$ and the parameters

from the following set of equations.

$$\begin{gathered} p_{t+1} = m + \delta_1 d_{t+1} + ... + \delta_q d_{t-q+2} + w_{t+1} \\ m + \delta_1 d_{t+1} + ... + \delta_q d_{t-q+2} = \sum_{i=1}^{\infty} b^i E d_{t+i+1}) | H_{t+1} \\ w_{t+1} = z_{t+1} + c_{t+1} \\ z_{t+1} = \sum_{i=1}^{\infty} b^i (E d_{t+i+1}) | I_{t+1} - E d_{t+i+1}) | H_{t+1}) \end{gathered} \tag{24}$$

Hansen and Sargent (1981) show that the system of equations from (24) can be simplified into the following set of constraints, $R(\theta)$:

$$\begin{gathered} 0 = m - b(1-b)^{-1} \Phi(b)^{-1} \mu \\ 0 = \delta_1 - (\Phi(b)^{-1} - 1) \\ 0 = \delta_j - \Phi(b)^{-1} \sum_{k=j}^{q} b^{k-j+1} \Phi_k, \text{ for } j = 2, ..., q, \end{gathered} \tag{25}$$

where $\Phi(b)^{-1} = (1 - \sum_{i=1}^{q} b^i \phi_i)^{-1}$.

The null hypothesis of no bubbles present in the data is given as $R(\theta) = 0$. The test statistic is calculated as

$$R(\hat{\theta})' \left[\left(\frac{\partial R(\theta)}{\partial \hat{\theta}} \right) (F_T' S^{-1} F_T)^{-1} \left(\frac{\partial R(\theta)}{\partial \hat{\theta}} \right)' \right]^{-1} R(\hat{\theta}) \tag{26}$$

This statistic (26) is χ^2-distributed with $q + 1$ degrees of freedom.

3.3 Unit root and cointegration tests

Another way to test for the existence of asset-bubbles is to apply unit root tests to the relationship between stock prices and dividend yields. If the asset price does not deviate from its fundamental value, the time-series of the ratio between the stock price and its dividend yield should be stationary. In other words, if the two time series are cointegrated, they share the same stochastic drift, which is counter the hypothesis of an asset price bubble. This approach has been widely used by the academic literature (Phillips et al., 2013, 2011; Chan and Woo, 2008; Jirasakuldech et al., 2008; Waters, 2008; Cuñado et al., 2005; Koustas and Serletis, 2005; Bohl, 2003; Chung and Lee, 1998; Lamont, 1998; Charemza and Deadman, 1995; Horvath and Watson, 1995; Ackert and Smith, 1993; Craine, 1993; Froot and Obstfeld, 1989; Diba and Grossman, 1988; Campbell and Shiller, 1986).

Diba and Grossman (1988) propose a test that allows for other factors o_t that influence the price and specify the market fundamental pricing relation as

$$p_t^f = \sum_{i=1}^{T-i} \frac{E_t(d_{t+i} + o_t)}{(1+r)^i}. \tag{27}$$

Assuming that the other factors are not more non-stationary than the dividend process, the market fundamental price will be of the same non-stationary order. The presence of bubbles can be thus tested by verifying whether stock prices are stationary after differencing the same number of times to make dividends stationary. Using the Dickey-Fuller test, Diba and Grossman (1988) conclude that after integrating both time series by an order of one, equation (27) holds, which rejects the presence of bubbles. Other authors have however obtained contradictory results using different data frequency or data sub-samples.

Campbell and Shiller (1986) identify persistent deviations of stock prices from present value of future dividends analysing the S&P 500 composite index and thus provide evidence of bubbles existence while using tests for cointegration between stock value and dividends. Similar conclusions are drawn by Froot and Obstfeld (1989) and Craine (1993), who test for a unit root presence in the price-dividend ratio on annual data of S&P 500 index. Bohl (2003) also obtains differing results of unit root tests with and without including in data the IT bubble observed since mid 90s. The following empirical papers were also not able to resolve the issue of existence of asset price bubbles as their findings are often contradictory. For instance, Horvath and Watson (1995) support the cointegration relationship between stock prices and dividends and suggest that no bubble exist in the US. On the other hand, Cuñado et al. (2005) test for presence of a bubble using fractional approach. When using monthly data, they cannot reject the unit root hypothesis, which implies presence of rational bubbles. Nevertheless, on daily and weekly data, they reject the hypothesis of unit root. The inability of existing literature to conclusively prove or disprove the existence of asset-price bubbles leaves this an open empirical question.

Evidence outside the US varies widely. For example, Jirasakuldech et al. (2008) use multivariate tests to examine the Thai equity market and conclude that there is no long-run relationship between prices, dividends and earnings, which indicates the presence of a bubble. Chung and Lee (1998) find that Hong Kong and Singaporean prices do not deviate from fundamentals, whereas Korean and Japanese markets are strongly influenced by non-fundamental and non-financial factors. Cheng et al. (2012) find that price efficiency is improved with increased short selling after removing short-sale constraints on IPO stocks in the Taiwanese market.

The existing literature identifies the following limitations of unit root tests. Ackert and Smith (1993) argue that the standard measures of dividend payments underestimate the total cash flows to shareholders, which impacts the cointegration between dividends and prices. Moreover, Waters (2008) notes that certain types of bubbles are not detectable by unit root tests. He concludes that unit root tests have difficulty differentiating between periodically collapsing bubbles and a persis-

tent, mean-reverting process. He also finds that bubbles generated by the stochastic explosive unit root model of Charemza and Deadman (1995) can be detected by properly specified unit root tests, but the periodically collapsing bubbles of Evans (1991) cannot.

Since standard unit root tests may erroneously reject evidence of bubbles when prices contain stochastic bubbles, Chan and Woo (2008) develop a new test to detect the existence of stochastic explosive root bubbles. Using Monte-Carlo simulation, they examine the power performance of this new test for the null hypothesis of cointegration in the presence of bubbles with a stochastic explosive root. Koustas and Serletis (2005) use autoregressive fractionally integrated moving average (ARFIMA) processes, since it allows for more flexible modelling of low frequency dynamics of stock prices, dividends, and their equilibrium relationship while allowing significant deviations from equilibrium in the short run. Moreover, it enables the authors to investigate the degree of persistence of log dividend yields for the S&P 500. They conclude that fractionally integrated dividend yield is consistent with rational bubbles in stock market prices.

Phillips et al. (2011) develop a new methodology to identify bubble behaviour with consistent dating of their origination and collapse. This allows them to create an early warning diagnostic of bubble activity. They investigate time series of financial asset prices, commodity prices and bond prices are investigated and propose a new procedure for the identification of bubble migration across markets. They find statistically significant bubble characteristics in all time series in the US financial markets. Phillips et al. (2011) employ an econometric method using forward recursive regression coupled with sequential right-sided unit root tests. This method enables early identification of mildly explosive behaviour in asset prices, or anticipatory evidence of a move away from martingale behaviour. The method proposed by Phillips et al. (2011) is effective in situations when there is a single bubble existing in the data. The identification of multiple bubbles, which periodically collapse is a more complex exercise (Phillips et al., 2013). This limits the use of standard econometric methods for bubble identification since in reality more bubbles can exist within the analysed period. Phillips et al. (2013) propose a procedure suitable for cases with multiple bubbles in the data set by allowing for a flexible window width, on which the tests for bubbles identification are conducted.

3.3.1 Phillips, Shi, and Yu (2013) rolling window approach

The model specification for asset prices is based on equation (17) including the bubble component:

$$p_t = \sum_{i=1}^{T-i} \frac{E_t(d_{t+i} + o_t)}{(1+r)^i} + B_t. \tag{28}$$

The bubble component satisfies the submartingale property:

$$E_t(B_{t+1}) = (1+r)B_t. \tag{29}$$

We can empirically test for the presence of an explosive process in the the price-dividend ratio by using the most standard test of unit root existence, the Augmented Dickey-Fuller (ADF) test (Said and Dickey, 1984), with a null hypothesis of unit root presence. We allow possibility of multiple bubbles in the data following Phillips et al. (2013) using flexible rolling windows. The empirical model is defined as

$$\Delta y_t = \alpha_{r_1,r_2} + \beta_{r_1,r_2} y_{t-1} + \sum_{i=1}^{k} \psi^i_{r_1,r_2} \Delta y_{t-i} + \epsilon_t, \tag{30}$$

where Δy_t is a change in price-dividend ratio, $\alpha_{r_1,r_2}, \beta_{r_1,r_2}$ and $\psi^i_{r_1,r_2}$ are regression coefficients, k is the lag order, $r_2 = r_1 + r_w$ is the window size and errors are i.i.d. random variables, $\epsilon_t \sim N(0, \sigma^2_\epsilon)$. The corresponding ADF test statistics based on this regression are defined as $ADF^{r_2}_{r_1}$.

The presence of bubbles is tested using the generalised supremum ADF (GSADF) test introduced by Phillips et al. (2013). The GSADF test is based on a sequence of recursive ADF unit root tests with flexible forward rolling window. The product of this method is a dating strategy, which identifies the points of bubble origination and termination. The advantage of using the GSADF test is the flexibility of windows with flexible starting and ending points. This enables the detecting of explosive behaviour more frequently than by using the method with fixed windows.

The idea of the GSADF test is to run the regression from equation (30) recursively on sub-samples of data. The GSADF test statistics is then the largest ADF statistic (supremum) over all feasible ranges of r_1 and r_2:

$$GSADF(r_0) = \sup_{r_2 \in [r_0,1], r_1 \in [0, r_2 - r_0]} \left\{ ADF^r_{r_1} r_2 \right\} \tag{31}$$

The asymptotic critical values necessary for any statistical inference about significance are provided in Phillips et al. (2013). The asymptotic distribution of GSADF

tests depends on the smallest window size, as the smallest window size decreases, critical values increase.

The most important feature of a successful asset-pricing detection tool is the ability of the method to assess whether the bubble exists or not with real time data. In other words, it is required that the mechanism correctly evaluates if a particular point in time belongs to a bubble phase. Phillips et al. (2013) propose a backward GSADF test to perform tests on a backward expanding sample. The origination date of a bubble is the first chronological observation, when GSADF test statistics value is below the critical value. The outcome of this strategy is a set of origination and ending dates of asset-price bubbles existence. The main advantage of this method relative to other approaches is that it can be applied on data samples that include more than one explosive feature.

3.4 Bubbles treated as unobservable variables

The empirical model used for speculative bubbles identification when bubbles are treated as unobservable variables follows the method from Al-Anaswah and Wilfling (2011). This technique is based on the state-space model with Markov switching. Al-Anaswah and Wilfling (2011) define a present value stock price model in state-space form estimated using Kalman filter. The unobservable bubble process is specified by two-regime Markov switching model, where the two regimes are bubble survival and bubble collapse. Al-Anaswah and Wilfling (2011) follow Wu (1997), who treats bubbles as unobservable variables and extend their scheme by allowing for switching regimes.

The model is specified as follows:

$$q = \kappa + \psi E_t(p_{t+1}) + (1 - \psi)d_t - p_t, \tag{32}$$

where q is the log return, expectations are conditional on all available information at time t, $p_t = ln(P_t)$ is the log stock price at time t, $d_t = ln(D_t)$ is the log dividend paid at time t and κ, ψ are linearisation parameters.[3]

Applying the transversality condition, $\lim_{i\to\infty} \psi^i E_t(p_{t+i} = 0$, leads to the no bubble solution to equation (19) and defines the market fundamental asset price:

$$p_t^f = \frac{\kappa - q}{1 - \psi} + (1 - \psi) \sum_{i=0}^{\infty} \psi^i E_t(d_{t+i}). \tag{33}$$

The general solution to equation (33) includes the bubble component, which satisfies the submartingale property specified in equation (29). In order to prevent

[3] ψ is the average log dividend-price ratio and κ is defined by $\kappa = -ln(\psi) - (1 - \psi)ln(1/\psi - 1)$.

the problems related to non-stationarity of time-series data, the price equation is expressed in first differences.

$$\Delta p_t = \Delta p_t^f + \Delta B_t = (1-\psi)\sum_{i=0}^{\infty}\psi^i E_t(d_{t+i} - E_{t-1}(d_{t+i-1})) + \Delta B_t. \tag{34}$$

Wu (1997) and Al-Anaswah and Wilfling (2011) assume that dividends follow an AR integrated moving average process, in particular the ARIMA(h,1,0) in the form

$$\Delta d_t = \mu + \sum_{j=1}^{h}\phi_j \Delta d_{t-j} + \delta_t, \tag{35}$$

where $\delta_t \sim N(0, \sigma_\delta^2)$. The autoregressive process from equation (35) can be expressed in terms of $(h \times 1)$ vectors $\mathbf{y_t} = (\Delta d_t, \Delta d_{t-1}, ..., \Delta d_{t-h+1})'$, $\mathbf{u} = (\mu, 0, 0, ..., 0)$, $\mathbf{v_t} = (\delta, 0, 0, ..., 0)$ and $(h \times h)$ matrix

$$A = \begin{pmatrix} \phi_1 & \phi_2 & \phi_3 & ... & \phi_{h-1} & \phi_h \\ 1 & 0 & 0 & ... & 0 & 0 \\ 0 & 1 & 0 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & 1 & 0 \end{pmatrix}, \tag{36}$$

which simplifies the equation (31) into the form

$$\mathbf{y_t} = \mathbf{u} + \mathbf{A}\mathbf{y_{t+1}} + \mathbf{v_t}. \tag{37}$$

The stock price model from equation (30) is represented by

$$\Delta p_t = \Delta d_t + \mathbf{m}\boldsymbol{\Delta}\mathbf{y_t} + \Delta B_t, \tag{38}$$

where $\mathbf{m} = \mathbf{gA}(\mathbf{I}-\mathbf{A})^{-\mathbf{1}}[\mathbf{I} - (1-\psi)(\mathbf{I}-\psi\mathbf{A})^{-1}]$ (Campbell and Shiller 1987). The linear bubble process follows from the submartingale property specified in equation (16)

$$B_t = (1/\psi)B_{t-1} + \eta_t, \tag{39}$$

where η_t is assumed to be i.i.d., $N(0, \sigma_\eta^2)$, $\psi = 1 + r$ and ψ_t is uncorrelated with dividend innovation δ_t. The bubble process is however unobservable, which requires the use of state-space representation and the Kalman filter (Al-Anaswah and Wilfling, 2011).

State space model

The state-space representation of the present-value model is expressed as follows. The model description is based on Al-Anaswah and Wilfling (2011) and Wu (1995).

$$\boldsymbol{\beta}_t = \mathbf{F}\boldsymbol{\beta}_{t-1} + \boldsymbol{\xi}_t, \tag{40}$$

$$\mathbf{z_t} = \mathbf{H}\boldsymbol{\beta}_{t-1} + \mathbf{D}\mathbf{g_t} + \boldsymbol{\zeta}_t, \tag{41}$$

In this system of equations, $\boldsymbol{\beta}_t$ is the $(n \times 1)$ vector of unobserved or state-space variables, $\mathbf{g_t}$ and $\mathbf{z_t}$ are $(m \times 1)$ and $(n \times 1)$ vectors of observed (input and output) variables, $\mathbf{F}, \mathbf{H}$ and $\mathbf{D}$ are constant real matrices and $\xi_\mathbf{t}$ and $\zeta_\mathbf{t}$ are $(n \times 1)$ and $(l \times 1)$ vectors of disturbances. We assume that the vectors of disturbances are serially uncorrelated, $E(\boldsymbol{\xi}_t) = \mathbf{0}$, $E(\boldsymbol{\xi}_t\boldsymbol{\xi}_t') = \boldsymbol{\Omega}$, $E(\boldsymbol{\zeta}_t) = \mathbf{0}$ and $E(\boldsymbol{\zeta}_t\boldsymbol{\zeta}_t') = \mathbf{R}$.

The asset-pricing model consists of three parts, the ARIMA dividend process from equation (35), the stock price process from equation (34) and the bubble process from equation (39). The model can be expressed in the state-space form defined in equations (40) and (41) when

$\boldsymbol{\beta}_t = (B_t, B_{t-1})', \mathbf{z_t} = (\Delta d_t, \Delta p_t)'$,
$\mathbf{g_t} = (1, \Delta d_t)', \Delta d_{t-1}, \Delta d_{t-2}, ..., \Delta d_{t-h})'$,
$\boldsymbol{\xi}_t = (\eta_t, 0)', \zeta_t = (\delta_t, 0)'$

$$\mathbf{F} = \begin{pmatrix} 1/\psi & 0 \\ 1 & 0 \end{pmatrix}, \mathbf{H} = \begin{pmatrix} 0 & 0 \\ 1 & -1 \end{pmatrix}, \tag{42}$$

and

$$\mathbf{D} = \begin{pmatrix} \mu & 0 & \phi_1 & \phi_2 & ... & \phi h - 1 & \phi_h \\ 0 & (1+m_1) & (m_2 - m_1) & (m_3 - m_2) & ... & (m_{h-2} - m_{h-1}) & -m_h \end{pmatrix}, \tag{43}$$

where m_i is the ith element of vector $\mathbf{m}$ from equation (25). The covariance matrices are specified as

$$\boldsymbol{\Omega} = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & 0 \end{pmatrix}, \mathbf{R} = \begin{pmatrix} \sigma_\delta^2 & 0 \\ 0 & 0 \end{pmatrix}. \tag{44}$$

The bubble process is treated as the unobservable part of the state space model. There are two measurement equations, including the dividends and asset price process and two transition equations defining the bubble process. The estimation of the stochastic bubble component using the Kalman filter is outlined in Al-Anaswah and Wilfling (2011), page 1075.

3.4.1 State-space model with Markov switching regimes

In the state-space model with Markov switching regimes, the parameters of matrices $\mathbf{F}, \mathbf{H}, \mathbf{D}, \mathbf{\Sigma}$ and $\mathbf{R}$ can switch between two regimes, which modifies the state space model into the following form

$$\boldsymbol{\beta}_t = \mathbf{F}_{\mathbf{S_t}} \boldsymbol{\beta}_{t-1} + \boldsymbol{\xi}_t, \tag{45}$$

$$\mathbf{z_t} = \mathbf{H}_{\mathbf{S_t}} \boldsymbol{\beta}_{t-1} + \mathbf{D}_{\mathbf{S_t}} \mathbf{g_t} + \boldsymbol{\zeta}_t, \tag{46}$$

$$\begin{pmatrix} \boldsymbol{\xi}_t \\ \boldsymbol{\zeta}_t \end{pmatrix} = N \left(0 \quad \begin{pmatrix} \mathbf{\Omega}_{S_t} & 0 \\ 0 & \mathbf{R}_{\mathbf{S_t}} \end{pmatrix} \right). \tag{47}$$

S_t denotes the unobservable two-states regime variable, $S_t = (1, 2)$, which determines the values of parameters at time t. The Markov transition probability matrix is given by

$$\mathbf{\Pi} = \begin{pmatrix} p_{11} & 1 - p_{22} \\ 1 - p_{11} & p_{22} \end{pmatrix}, \tag{48}$$

where $p_{ij} = P(S_t = j | S_{t-1} = i)$.

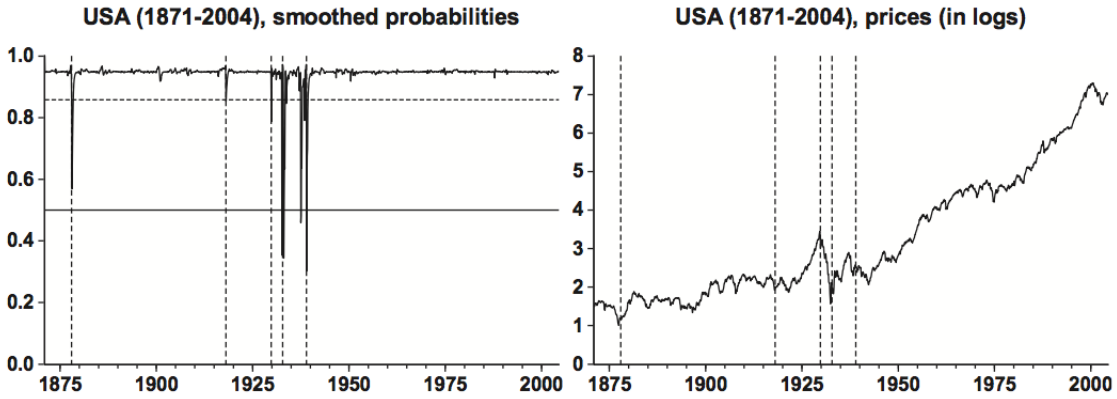
The procedure to estimate the parameters of the state-space model is alsi depicted in in Figure 5.1 in Kim and Nelson (1999, p. 105). In summary, after obtaining the initial values, the Kalman filter is run with while collapsing the posteriors $\boldsymbol{\beta}^{(i,j)}_{t|t-1}$ and $\mathbf{P}^{(i,j)}_{t|t-1}$ after the end of each iteration to get $\boldsymbol{\beta}^{j}_{t|t}$ and $\mathbf{P}^{j}_{t|t}$.

The aim of this regime-switching model is to estimate the two-state switching parameter matrices $\mathbf{F}_{\mathbf{S_t}}, \mathbf{H}_{\mathbf{S_t}}, \mathbf{D}_{\mathbf{S_t}}, \mathbf{\Omega}_{S_t}, \mathbf{R}_{\mathbf{S_t}}$. Al-Anaswah and Wilfling (2011) note that it is difficult to numerically maximise the likelihood function with so many parameters and reduce their model complexity by imposing the following restrictions on model parameters. They model the dividend process d_t as a pure random walk without drift, i.e. set $\mu = 0$ and $h = 0$ in equation (35) thereby contracting the matrix D from equation (43) correspondingly. Second, they only allow the autoregressive coefficient $1/\psi$ in the bubble process (39) to switch between the two regimes (in order to discriminate between moderately growing and explosive periods in the bubble process) while modeling all other parameters as non-switching between the Markov-regimes. Both restrictions imply that Al-Anaswah and Wilfling (2011) estimate the switching autoregressive bubble coefficients $1/\psi_1$ and $1/\psi_2$ in the matrices $\mathbf{F}_{\mathbf{S_t}}$, the non-switching variances σ^2_η and σ^2_δ in the matrices $\mathbf{\Omega}$ and $\mathbf{R}$ from equations (44) and both transition probabilities p_{11} and p_{22} from the matrix Π in equation (48).

This method is able to identify (ex-post) the dates of bubble burst, which are the points in time when the probability of bubble surviving is lower than 0.5. The

example of the bubble burst identification for the US market is shown in Figure 2. This method offers a real-time mechanism how to detect bubbles in financial markets and brings valuable estimates of bubble burst probabilities for each point of time of the analysed period.

Figure 2: State-space model with regime switching. Source: Al-Anaswah and Wilfling (2011), page 1083.



4 Implementation

We implement the main empirical tests for asset-price bubbles detection in the Australian market, using the S&P ASX 200 index as a proxy for the market and the A-REIT index for Real Estate Investment Trusts listed on ASX as a proxy for the property market. We provide results of the Shiller (1981a) variance bounds test, West (1987) two-step test, and both the Phillips, Shi, and Yu (2013) and Phillips, Wu, and Yu (2011) unit root tests.

4.1 Australian Equities: S&P ASX 200 index

4.1.1 Variance bounds test

We apply the Shiller (1981a) variance bounds test, which considers the implications of comparing the efficient market model against a perfect foresight model which is commonly referred to as the ex-post rational price.

The efficient market model is represented as:

$$p_t = \sum_{i=0}^{\infty} \frac{E_t(d_{t+i})}{(1+r)^i}$$

where p_t is the real price, d_t is the real dividend, r is the constant interest rate, and $E_t()$ is the expectation operator conditional on information available at time t.

The ex post rational price is defined as:

$$p_t^* = \sum_{i=0}^{\infty} \frac{d_{t+i}}{(1+r)^i}$$

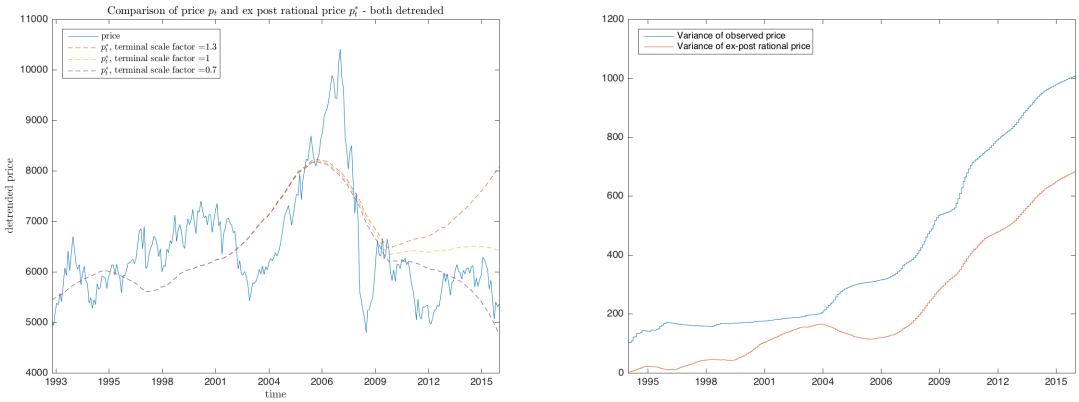
where p_t^* is the ex post rational price.

The test compares the standard deviation of the two series. If the efficient markets model holds true, then it can be shown that:

$$\sigma(p_t) \leq \sigma(p_t^*).$$

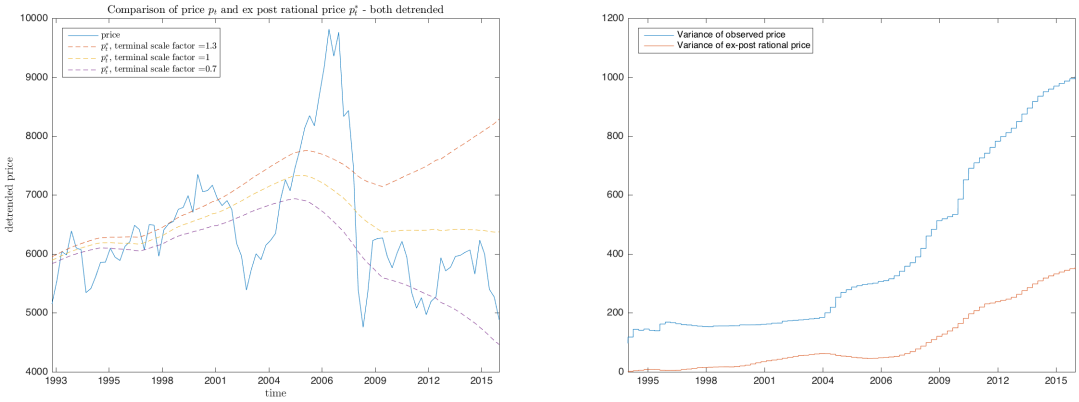
Figure 3 (Figure 4) displays the observed and ex-post rational price series on monthly (quarterly) basis as well as the corresponding variance bounds.

Table 1 contains the boundary test results using monthly and quarterly data. Columns of the results table correspond to a different end point scaling of the ex post rational price, with 1.0 representing the mean of the detrended price series. The standard deviation of the price series, std(p), clearly exceeds the standard deviation of each ex post rational price series, std(p*), both on monthly and quarterly basis.



(a) Prices (b) Variances

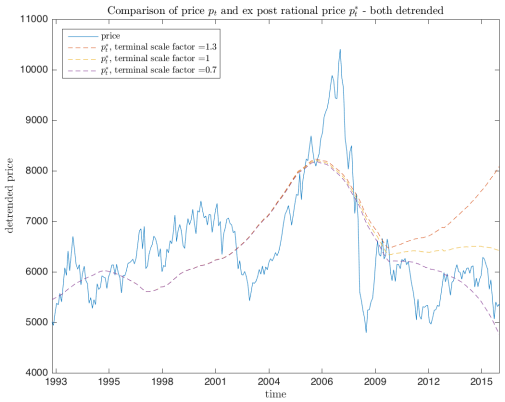
Figure 3: Variance Bounds Test on the S&P ASX 200 index. Comparison of observed price and ex-post rational price (both detrended) in panel a), and its variances in panel b), calculated using monthly data from May 1992 until April 2016



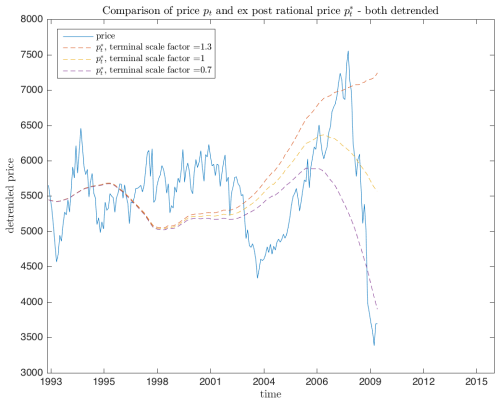
(a) Prices (b) Variances

Figure 4: Variance Bounds Test on the S&P ASX 200 index. Comparison of observed price and ex-post rational price (both detrended) in panel a), and its variances in panel b), calculated using quarterly data from May 1992 until April 2016

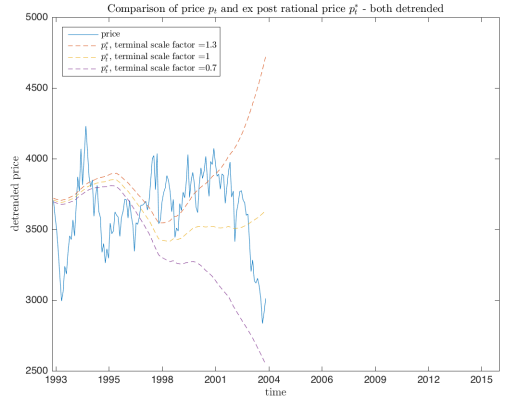
One interpretation of this result is that the price and dividend data contains bubbles. However, we should be cautious here as the failure of the test does not



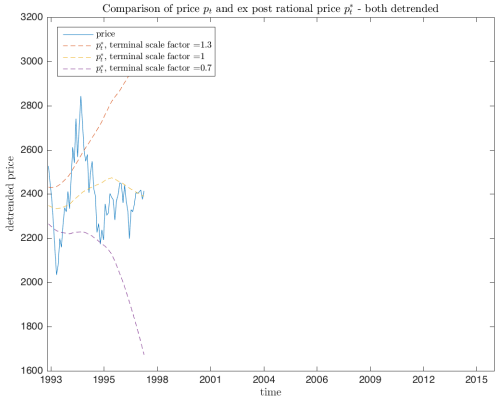
(a) 1992-2016



(b) 1992-2010



(c) 1992-2004



(d) 1992-1998

Figure 5: Variance Bounds Test on the S&P ASX 200 index. Comparison of observed price and ex-post rational price (both detrended) using monthly data from May 1992 until April 2016 in panel a), monthly data from May 1992 until December 2010 in panel b), monthly data from May 1992 until December 2004 in panel c) and monthly data from May 2000 until December 1998 in panel d).

directly imply the existence of bubbles. Essentially, the Shiller (1981a) variance bounds test is a validation test of the efficient markets model and is thus not a direct test for rational bubbles.

The Shiller (1981a) test assumes that the price process is stationary, which is unlikely to be satisfied in the real world. We therefore focus further on alternative

Table 1: Results of the Shiller (1981a) variance bounds tests conducted on monthly and quarterly data available between May 1992 and April 2016

	Monthly			Quarterly		
E(p)	6420.90	6420.90	6420.90	6375.70	6375.70	6375.70
E(d)	246.39	246.39	246.39	245.18	245.18	245.18
r	0.04	0.04	0.04	0.04	0.04	0.04
r2	0.08	0.08	0.08	0.08	0.08	0.08
b	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
cor(p,p*)	0.45	0.65	0.71	0.23	0.64	0.57
std(d)	41.84	41.84	41.84	41.92	41.92	41.92
std(p)	1079.60	1079.60	1079.60	1067.70	1067.70	1067.70
std(p*)	846.62	759.50	853.01	653.21	402.53	634.01

bubble detection strategies.

4.1.2 West (1987) two-step tests

Unlike the variance bounds test which doesn't directly include the presence of bubbles when rejecting the model, the West (1987) test specifically includes the ability to check for bubbles by testing the model and no-bubbles hypotheses sequentially.

The Euler equation relating prices and dividends is given by:

$$p_t = bE_t(p_{t+1} + d_{t+1})$$

where p_t is the real stock price in period t, b the constant real discount rate $0 < b = 1/(1+r) < 1$, r the constant expected return, and $E_t()$ is the expectation operator conditional on information available at time t.

In the absence of bubbles an estimate of the discount rate can be obtained by noting that

$$p_t = b(p_{t+1} + d_{t+1}) + u_{t+1}$$

where u_{t+1} is the error term. Since u_{t+1} is correlated with p_{t+1} and d_{t+1}, the estimation requires the use of an Instumental Variable.

Next, West (1987) assumes dividends follow a pure auto-regressive process, order q, of the form

$$d_{t+1} = \mu + \phi_1 d_t + \ldots + \phi_q d_{t-q+1} + v_{t+1}$$

Estimates for the dividend regression coefficients are readily obtained by OLS.

Finally, the usual asset price equation

$$p_t = \sum_{i=1}^{\infty} b^i E_{t+1}(d_{t+i})$$

is estimated by letting

$$p_{t+1} = m + \delta_1 d_{t+1} + \ldots + \delta_q d_{t-q+2} + w_{t+1}$$

As each estimation process above represents part of the relationship between prices and dividends it is possible to derive constraint equations relating all the estimated model parameters. Under the assumption that no bubbles are present, the constraint equations should all be satisfied. A statistical test is constructed to measure how closely the constraints are satisfied.

It is important to note that the West (1987) test hinges on being able to validate the discount rate and dividend ARIMA estimation processes. Provided these estimates are sound, model misspecification can be ruled out and any violation of the constraint equations can be directly attributed to the presence of bubbles. Consistent with West's (1987) original application, we set the AR order q to 2 and use both level and differenced data.

Table 2: Results of the West (1987) two-step test conducted on monthly and quarterly data available between May 1992 and April 2016

	Monthly		Quarterly	
Data Size:	286	285	94	93
Data Set:	05/92-04/16	05/92-04/16	05/92-04/16	05/92-04/16
Differenced:	no	yes	no	yes
q:	2	2	2	2
DoF:	3	3	3	3
Test Statistic:	0.025	0.003	0.064	0.018
P-value:	0.999	1.000	0.996	0.999

When we run the West test on full data sample from 1992 to 2016, we get a p-value close to 1 and cannot reject the hypothesis that there are no bubbles in the Australian data during this time period, see Table 2. The Test Statistic from Table 2 represents the West's test statistic defined in equation (26). The same conclusion holds if we divide the data into shorter periods (e.g. one-year or two-year periods), in all the sub-periods, the p-value is greater than 0.9, which suggests that based on

the West (1987) test, there is no evidence to suggest that bubbles exist in either Australian stock or real estate prices.

4.1.3 Unit root tests

Unit Root tests rely on testing the relationship between the price and dividend ratio series for stationarity. If the price and dividend series are cointegrated they share the same stochastic drift, which is counter to the hypothesis of bubbles existing. The testing procedures broadly fall under the term Unit Root testing.

The simplest test for a unit root is the augmented Dickey-Fuller (ADF) test. In this testing scenario, the entire sample is considered and a single test result is returned. The power of a simple ADF test to detect bubbles is weak in the sense that it doesn't yield useful information about the start and end dates of bubble episodes, and it can fail if more than one episode occurs.

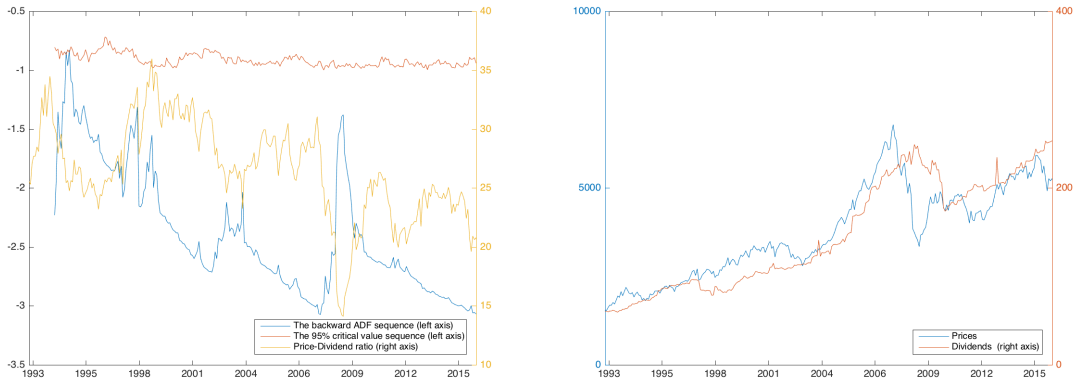
Phillips, Shi, and Yu (2013) rolling window approach

To get around the limitations of the simple ADF test Phillips, Wu, and Yu (2011) propose an extension called the Supremum ADF test (SADF) which considers an increasing sample window size, starting from a mimimum window and ending at the full sample. Their test then calculates the Supremum of all ADF tests conducted on all sub-sample windows and compares that to appropriate critical values derived by extensive Monte Carlo simulations. The SADF test also offers a date-stamping strategy whereby bubble start and end dates can be successfully identified. Although the SADF test is extremely useful it has some shortcomings, not least of which is that it can fail to find bubbles when considering the full sample size but can successfully find bubbles when considering each half of the full sample separately.

We perform an SADF test, with parameters specified exactly as in Phillips, Wu, and Yu (2011). Specifically, we test for:

- a unit root with drift
- a minimum window size of 36 months
- zero lag terms included in each ADF test
- 2000 Monte Carlo simulations when computing critical values
- Null model parameters d and η set to 1
- Critical value tests performed at 90%, 95%, and 99% confidence levels

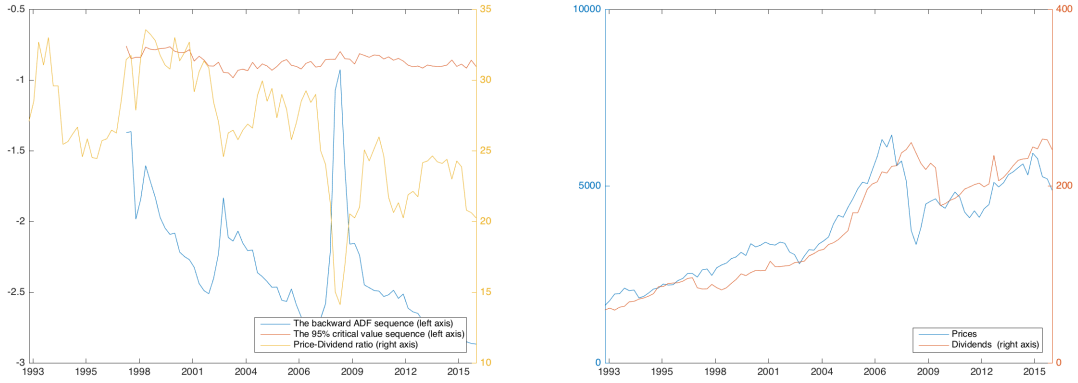
Phillips, Wu, and Yu (2011) Unit Root Test 1: Monthly S&P ASX 200



(a) Test Diagnostic (b) Prices and Dividends

Figure 6: S&P ASX 200, SADF test on monthly price-dividend ratio from May 1992 until April 2016 in panel a), with corresponding price and dividend levels in panel b).

Phillips, Wu, and Yu (2011) Unit Root Test 1: Quarterly S&P ASX 200



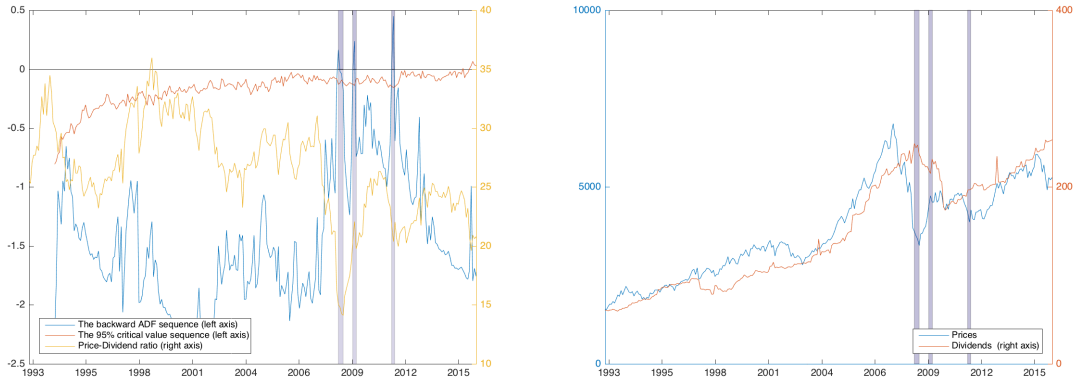
(a) Test Diagnostic (b) Prices and Dividends

Figure 7: S&P ASX 200, SADF test on quarterly price-dividend ratio from May 1992 until April 2016 in panel a), with corresponding price and dividend levels in panel b).

We also plot the sample sequence, the backwards ADF statistic sequence, and the 95% critical value sequence on monthly basis in Figure 6 and on quarterly basis in Figure 7. We cannot identify any bubbles in monthly or quarterly prices.

The SADF test, which is based on the backward sequence of ADF test statistic, shown in Figures 6 and 7, has some shortcomings and may not adequately identify bubbles present in the data. Phillips et al. (2013) improve this method and propose a new measure, the Generalized supremum ADF test (GSADF). The GSADF test extends the concept of the SADF test by allowing the window start point to also move, thereby covering many more sub-samples of the entire sample size. The GSADF test has been shown to be extremely effective in identify commonly accepted bubble episodes in historical data, and it also offers the same data-stamping strategies as the SADF test. Note that the data-stamping strategies for SADF and GSADF are typically run on backwards tests, rather than forwards, to provide test results at the most recent sample point (i.e. the windows start at the most recent points and extend back in time, rather than forwards in time).

Phillips, Shi, and Yu (2013) Unit Root Test 2: Monthly S&P ASX 200



(a) Test Diagnostic

(b) Prices and Dividends

Figure 8: A-REIT index, The backwards SADF statistic sequence: SADF test on monthly price-dividend ratio from May 1992 until April 2016 in panel a), with corresponding price and dividend levels in panel b). Shaded areas represent detected asset-price bubbles.

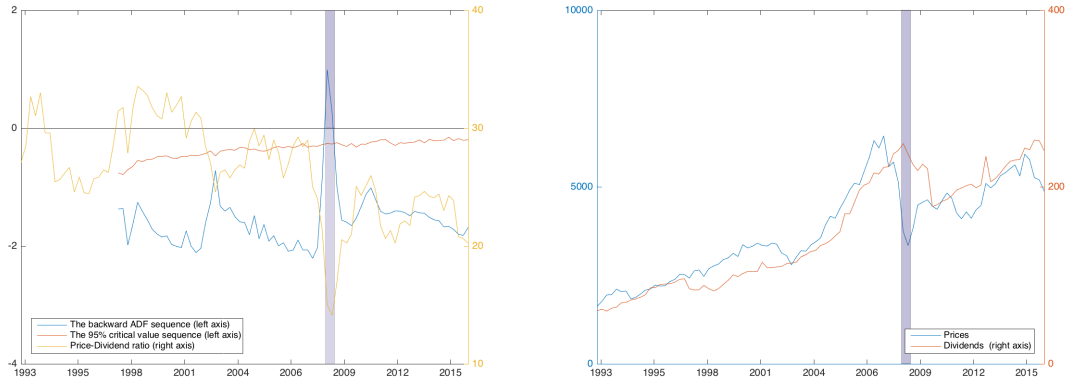
We plot the output of this improved unit root test statistic, and the 95% critical value sequence based on monthly data in Figure 8 and quarterly data in Figure 9. When we analyse monthly prices, we find that there are points at which the

backwards SADF sequence exceeds the critical value sequence, which suggests that there are bubbles present in data. Based on the SADF backward-dating strategy, there is evidence of three periods, when prices contain a bubble in the Australian market. The periods when the Phillips et al. (2013) procedure detected bubbles in prices are from November 2008 until February 2009, August 2009 until November 2009 and August 2011 until October 2011.

All of these three periods are rather short and represent a small deviation of prices from dividends. We therefore do not suspect that there is any systematic asset-price bubble present in the data.

Furthermore, we focus on quarterly data and conduct the same test to detect any bubbles in quarterly prices. Our results suggest that there is only one period (which is also relatively short) of a bubble between September 2008 and February 2009, Figure 9.

Phillips, Shi, and Yu (2013) Unit Root Test 2: Quarterly S&P ASX 200



(a) Test Diagnostic

(b) Prices and Dividends

Figure 9: S&P ASX 200, The backwards SADF statistic sequence: SADF test on quarterly price-dividend ratio from May 1992 until April 2016 in panel a), with corresponding price and dividend levels in panel b). Shaded areas represent detected asset-price bubbles.

Table 3 provides the value of the test statistic used to examine whether there are any asset-price bubbles using all the data available for the S&P ASX 200 index. Since the value of the test statistic is lower than most of the critical values, we conclude that there is little evidence of existence of bubbles in the Australian equity prices.

The results of no evident bubble found in the Australian equity market contrast

Table 3: Results of the unit-root tests conducted on monthly and quarterly data available between May 1992 and April 2016. We use the following test to identify existence of bubbles in the Australian market: the Supremum ADF test (SADF) proposed by Phillips, Wu, and Yu (2011) and the Generalized supremum ADF test (GSADF) proposed by Phillips, Shi, and Yu (2013). Test-Stat is the value of the test statistic of each test and CV represent critical values.

	Test-Stat	CV 90 pct	CV 95 pct	CV 99 pct
Monhtly				
SADF	-0.829	0.306	0.558	1.081
GSADF	0.450	1.222	1.451	1.897
Quarterly				
SADF	-0.927	0.044	0.313	0.783
GSADF	0.992	0.704	0.981	1.423

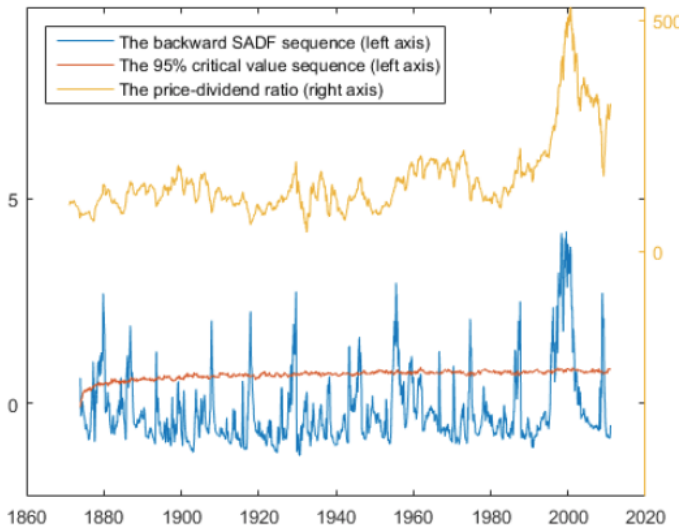


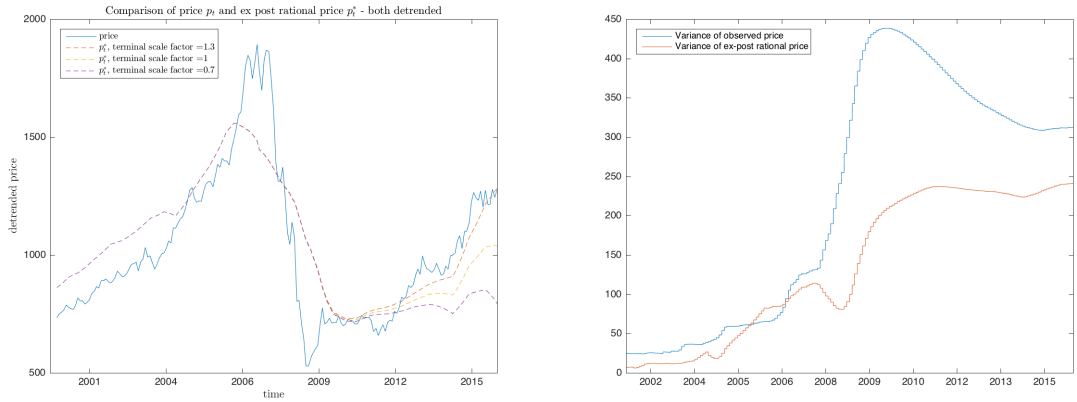
Figure 10: Results of the unit-root tests conducted on US equity data (price-to-dividend ratio) available between January 1871 and December 2010. We use the Phillips, Shi, and Yu (2013) Generalized supremum ADF test (GSADF) test to identify existence of bubbles in the US equity market: the Generalized supremum ADF test (GSADF).

those of the US equities. In the US, there is a strong evidence of the existence of an asset-price bubble, detected using the Phillips, Shi, and Yu (2013) unit root test, around the year 2000, when the test statistic exceeds the critical value for a considerable time period, which is associated with a sharp increase in prices, as shown in Figure 10.

4.2 Australian REIT market: A-REIT index

4.2.1 Variance bounds test

We apply the Shiller (1981a) variance bounds test on the A-REIT index to compare the standard deviation of the observed price and ex-post rational price, Figure 11 on monthly basis and Figure 12 on quarterly basis.



(a) Prices

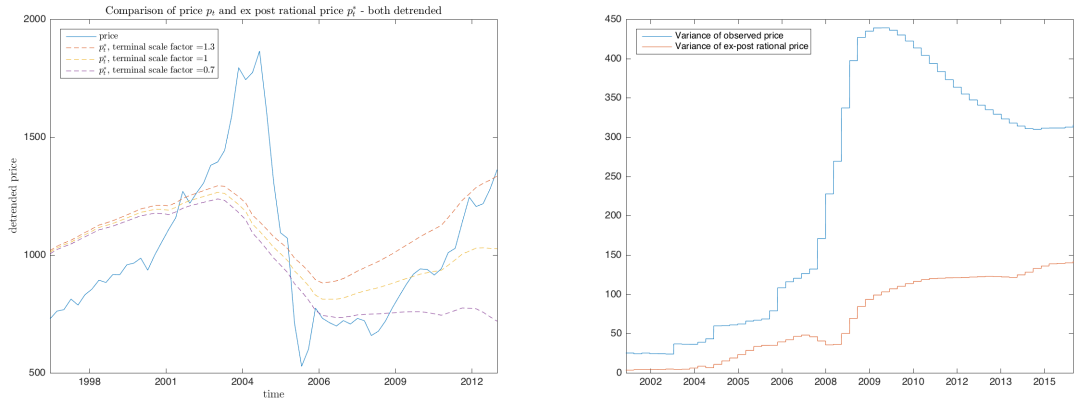
(b) Variances

Figure 11: Variance Bounds Test on the A-REIT index. Comparison of observed price and ex-post rational price (both detrended) in panel a), and its variances in panel b), calculated using monthly data from May 2000 until April 2016

Table 4 contains the boundary test result on a monthly and quarterly data. Columns of the results table correspond to a different end point scaling of the ex post rational price, with 1.0 representing the mean of the detrended price series. The standard deviation of the price series, std(p), clearly exceeds the standard deviation of each ex post rational price series, std(p*), both on monthly and quarterly basis.

4.2.2 West (1987) two-step tests

We apply the West test on full data sample of the A-REIT index from 2000 to 2016, and get a p-value close to 1, which implies that we cannot reject the hypothesis that there are no bubbles in the Australian REIT prices during this time period, see Table 5. The same conclusion holds if we divide the data into shorter periods (e.g. one-year or two-year periods), in all the sub-periods, the p-value is greater than 0.9,



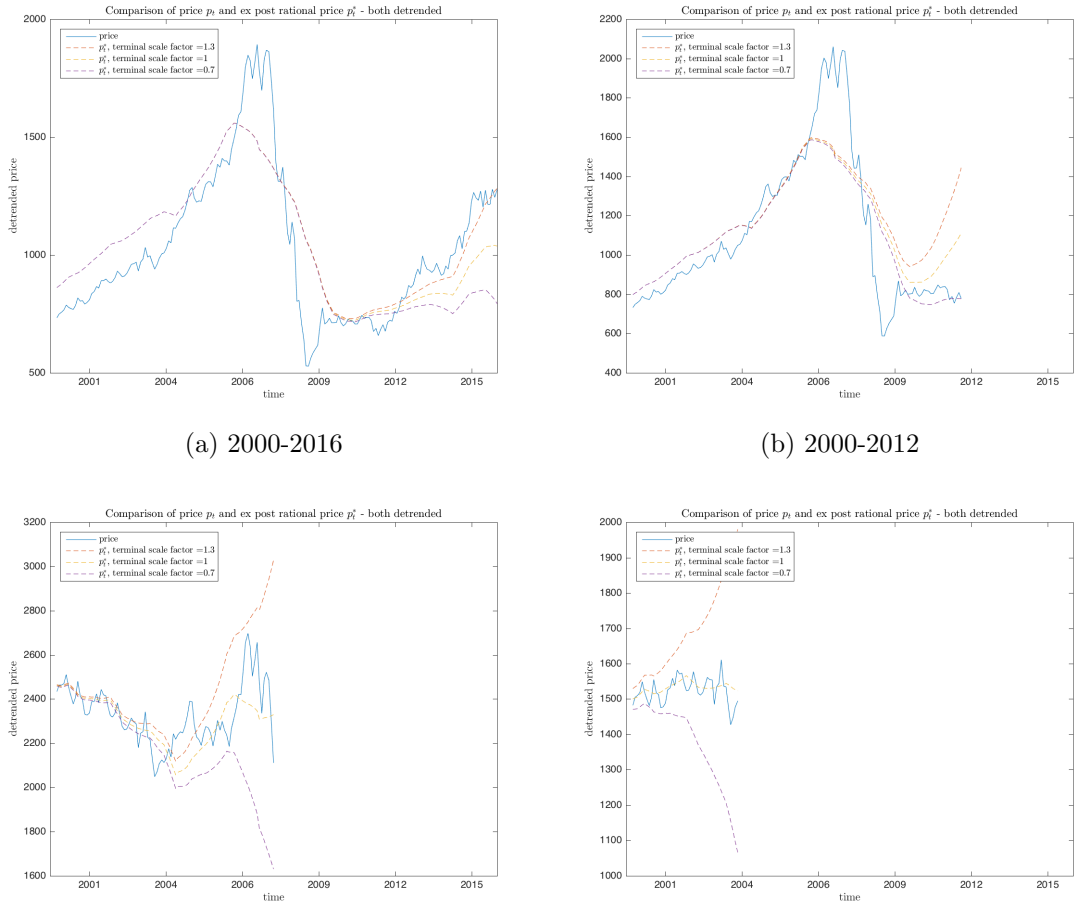
(a) Prices (b) Variances

Figure 12: Variance Bounds Test on the A-REIT index. Comparison of observed price and ex-post rational price (both detrended) in panel a), and its variances in panel b), calculated using quarterly data from May 2000 until April 2016

Table 4: Results of the Shiller (1981a) variance bounds tests conducted on monthly and quarterly data available between May 2000 and April 2016

	Monthly			Quarterly		
E(p)	1034.40	1034.40	1034.40	1027.00	1027.00	1027.00
E(d)	63.54	63.54	63.54	63.43	63.43	63.43
r	0.06	0.06	0.06	0.06	0.06	0.06
r2	0.13	0.13	0.13	0.13	0.13	0.13
b	-0.0001	-0.0001	-0.0001	-0.0001	-0.0001	-0.0001
cor(p,p*)	0.87	0.82	0.72	0.73	0.65	0.47
std(d)	20.37	20.37	20.37	21.08	21.08	21.08
std(p)	315.57	315.57	315.57	315.55	315.55	315.55
std(p*)	241.17	244.23	261.67	127.95	141.99	193.02

which suggests that based on the West (1987) test, we did not find any evidence of bubbles existing in Australian prices.



(a) 2000-2016

(b) 2000-2012

(c) 2000-2008

(d) 2000-2004

Figure 13: Variance Bounds Test on the A-REIT index. Comparison of observed price and ex-post rational price (both detrended) using monthly data from May 2000 until April 2016 in panel a), monthly data from May 2000 until December 2012 in panel b), monthly data from May 2000 until December 2008 in panel c) and monthly data from May 2000 until December 2004 in panel d).

4.2.3 Unit root tests

Phillips, Shi, and Yu (2013) rolling window approach

We perform the SADF test on A-REIT index, with parameters specified exactly as in Phillips et al. (2011). Specifically, we test for:

Table 5: Results of the West (1987) two-step test conducted on monthly and quarterly data available between May 2000 and April 2016

	Monthly		Monthly	
Data Size:	192	191	63	62
Data Set:	05/00-04/16	05/00-04/16	05/00-04/16	05/00-04/16
Differenced:	no	yes	no	yes
q:	2	2	2	2
DoF:	3	3	3	3
Test Statistic:	0.0101	0.001	0.1922	0.0713
P-value:	0.9997	1.000	0.9788	0.995

- a unit root with drift
- a minimum window size of 36 months
- zero lag terms included in each ADF test
- 2000 Monte Carlo simulations when computing critical values
- Null model parameters d and η set to 1
- Critical value tests performed at 90%, 95%, and 99% confidence levels

We identify one bubble in monthly A-REIT prices, observed between October 2014 and January 2015. We plot the sample sequence, the backwards ADF statistic sequence, and the 95% critical value sequence on monthly basis in Figure 14 and on quarterly basis in Figure 15.

This detected bubble is not caused by a sharp increase of price but rather by a sudden drop in dividend yield. This decrease in dividends of REITs was however not persistent and once dividends have increases to the original level, the detected ceased to exist. It is questionable that such situation can truly be referred to as an asset-price bubble in REIT prices since the cause of the detection of a bubble was not due to prices but dividends.

We also implement the Generalized supremum ADF (GSADF) test and plot the output of this improved unit root test statistic, and the 95% critical value sequence based on monthly data in Figure 16 and quarterly data in Figure 17. When we analyse monthly prices, we find that there are points at which the backwards SADF sequence exceeds the critical value sequence, which suggests that there are bubbles

Phillips, Wu, and Yu (2011) Unit Root Test 1: Monthly A-REIT index

(a) Test Diagnostic (b) Prices and Dividends

Figure 14: A-REIT index, SADF test on monthly price-dividend ratio from May 2000 until April 2016 in panel a) with corresponding price and dividend levels in panel b). Shaded areas represent detected asset-price bubbles.

Phillips, Wu, and Yu (2011) Unit Root Test 1: Quarterly A-REIT index

(a) Test Diagnostic (b) Prices and Dividends

Figure 15: A-REIT index, SADF test on quarterly price-dividend ratio from May 2000 until April 2016 in panel a) with corresponding price and dividend levels in panel b).

present in data. Based on the SADF backward-dating strategy, there is evidence of two periods, when the test detect an asset-price bubble in the Australian REIT market:

- Bubble 1: between March 2010 and July 2010
- Bubble 2: between September 2011 and November 2011.

The first bubble period is caused by the fact that dividends decreased by less than prices. The prices did not grow between March 2010 and July 2010 but the test still detects a bubble since dividends decrease at a higher rate. The second bubble period is relatively short and is also not associated with any sharp increase in price. We therefore conclude that REIT prices are relatively stable and follow the changes of dividends, which suggests that threre are no evident asset-price bubbles in REIT prices.

Phillips, Shi, and Yu (2013) Unit Root Test 2: Monthly A-REIT index

(a) Test Diagnostic

(b) Prices and Dividends

Figure 16: A-REIT index, The backwards SADF statistic sequence: SADF test on monthly price-dividend ratio from May 2000 until April 2016 in panel a) ,with corresponding price and dividend levels in panel b). Shaded areas represent detected asset-price bubbles.

Our findings using Phillips et al. (2013) on quarterly data Figure 17 and results from the two unit root tests on all data (Table 6) confirm our hypothesis that there is little evidence of asset-price bubbles in REIT prices.

Phillips, Shi, and Yu (2013) Unit Root Test 2: Quarterly A-REIT index

(a) Test Diagnostic

(b) Prices and Dividends

Figure 17: A-REIT index, The backwards SADF statistic sequence: SADF test on quarterly price-dividend ratio from May 2000 until April 2016 in panel a) ,with corresponding price and dividend levels in panel b).

Table 6: Results of the unit-root tests conducted on monthly and quarterly data available between May 2000 and April 2016. We use the following test to identify existence of bubbles in the Australian REIT market: the Supremum ADF test (SADF) proposed by Phillips, Wu, and Yu (2011) and the Generalized supremum ADF test (GSADF) proposed by Phillips, Shi, and Yu (2013). Test-Stat is the value of the test statistic of each test and CV represent critical values.

	Test-Stat	CV 90 pct	CV 95 pct	CV 99 pct
Monthly				
SADF	-0.776	0.262	0.479	1.089
GSADF	0.698	1.057	1.297	1.773
Quarterly				
SADF	-0.838	-0.030	0.249	0.778
GSADF	-0.662	0.445	0.730	1.252
GSADF	0.992	0.704	0.981	1.423

5 Critical Analysis

We discuss the performance of the existing empirical techniques to detect asset-price bubbles. In particular, we analyse the Shiller (1981a) variance bound test, West (1987) two-step test and Phillips, Shi, and Yu (2013); Phillips, Wu, and Yu (2011) unit root tests. We focus on the applicability of these tests in the Australian context.

5.1 Shiller (1981a) variance bounds test

Shiller (1981a) provides one of the first attempts to identify asset-price bubbles in US stock data. It compares observed prices with ex-post rational prices, calculated based on the expected value of dividend payments. With perfect foresight, the ex-post rational price is the present value of the sum of all future dividends. Without perfect foresight, the ex-post rational price contains an error, which increases its variance. The variance of the ex-post ration price should therefore be higher than the variance of observed prices.

Shiller (1981a) measures the variances based on a time series of prices and dividends and therefore assumes that the price process is stationary and ergodic. If this assumption is valid, the variance bound inequality is a good test to identify asset price bubbles in data. In reality, however we often see that prices are not stationary and are believed to follow a random walk non-stationary process. This implies that the variance of the non-stationary price process will be higher than the variance of ex-post rational prices measured using dividends, which is not due to the presence of asset-price bubbles. The variance bound test can be used only based on a cross sectional relationship, e.g. across economies but not time.

Because we have only one realisation of prices for a given time in reality, we do not recommend using the variance bound test to make any statements about the presence of bubbles.

5.2 West (1987) two-step test

West (1987) attempts to overcome the above mentioned problems with detection of asset-price bubbles and propose a different technique, which does not rely so heavily on unrealistic assumptions. He models two situations, with and without bubbles in prices. He then tests whether the two situations yield similar results, if yes, there is no bubble present in data.

West (1987) assumes that dividends follow an autoregressive process, which implies that the forecasting equations are stationary in either the levels or first differences of real dividends. Flood and Hodrick (1990) argue that the likelihood that

a constant process characterises dividends for over 100 years of data is rather low. Another aspect of the West (1987) test criticised by Flood and Hodrick (1990) is the specification of the the return generating model, which involves using only the one-period relation between current price and expected next period dividend and price.

In terms of the usability of this method for early detection of asset-price bubbles in data, this test does not directly identify the dates of beginning of bubbles and ends of bubbles. With a long time series of data and relatively short period of bubbles in data, this test is likely to be unable to correctly identify any bubble existence. Although, this test represents a valid instrument for ex-post detection of bubbles in data, it may under certain circumstances be unable to detect all price bubbles and we therefore do not recommend to use this test solely.

5.3 Unit root tests

Another alternative way to test for the presence of bubbles is to analyse the relationship between stock prices and dividend yields. This approach has become popular and widely-used among practitioners and researchers. We analyse the dating strategy methodology developed by Phillips, Wu, and Yu (2011) and updated by Phillips, Shi, and Yu (2013). This method aims to overcome some of the issues related to the usability of identification techniques that concerned e.g. the West (1987) two-step test, as discussed above. This dating strategy allows for an early-warning diagnostic of bubble activity. The original technique (Phillips, Wu, and Yu, 2011) is effective when there is a single bubble in data but may become unable to identify multiple, periodically collapsing bubbles. Phillips, Shi, and Yu (2013) therefore create an updated technique, which allows for a flexible window width and can detect more than one bubble in a given data set.

The unit root tests assess whether the price-dividend ratio is stationary. The Phillips, Shi, and Yu (2013) rolling-window approach uses the Augmented Dickey-Fuller test, with a null hypothesis of unit root presence and thus no bubble in data. Cochrane (1991) critises the use of unit root tests. He argues that time series with a unit root can be decomposed into a stationary series and a random walk. The random walk component can have a small variance so that the tests for unit roots will have arbitrarily low power in finite samples. This implies the identification of asset-prices bubbles using unit root test may have a large Type II error.

Despite some of the weaknesses, the Phillips, Shi, and Yu (2013) rolling-window strategy provides a strong test of bubble presence, which can be used as an early-warning mechanism since it also outputs the dates when the bubble in data started

and ended.

We have however identified a few situations when the Phillips, Shi, and Yu (2013) and Phillips, Wu, and Yu (2011) detected asset-price bubbles without having any significant increase in prices. The detection of bubbles was due to a decrease in dividends, which increased the price-to-dividend ratio. It is questionable whether such a situation can truly be defined as an asset-price bubble situation. We propose to be more conservative and do not recommend to make any statement about bubble presence based on the outcome of the unit root test only. We recommend to take into account all the available information, including the preceding changes of dividends and prices.

5.4 Real-time detection strategy

The Shiller (1981a) variance bounds test and the West (1987) two-step test provide merely an ex-post technique to determine whether there was an asset-price bubble present in data some years back in history. These tests cannot detect bubbles in real-time and are thus not suitable for an early-warning detection mechanism for an existing asset-price bubble.

The unit root tests developed by Phillips, Shi, and Yu (2013) and Phillips, Wu, and Yu (2011) on the other hand provide a mechanism to detect asset-price bubbles in real time, which increases their usability in practice. The output of the Phillips, Shi, and Yu (2013) and Phillips, Wu, and Yu (2011) tests is a timely dating strategy, which identifies the begginning date and ending data of each asset-price bubble present in data, using ex-post information available at each point in time only, and thus eliminating the forward-looking bias.

6 Conclusion

There are various theoretical views that take into account and explain the existence of asset-price bubbles on stock markets. While most of the theories focus on the question of whether bubbles can or cannot exist, few studies have sufficiently examined why and when bubbles emerge and burst (Brunnermeier, 2008).

This report first summarises the main theoretical arguments and market conditions that allow bubbles to exist. Second, it describes the most-widely used methods to detect bubbles in real time. Further, it implements the main methods for asset-price bubble identification in Australia, using the S&P ASX 200 index as a proxy for the Australian equity market and the A-REIT index as a proxy for the Australian REIT market. We find little evidence of asset-price bubbles in Australia. Australian prices are stable and consistent with dividend yields. Finally, we present a critical analysis of the method used and conclude that the unit root tests, represented by Phillips, Shi, and Yu (2013) and Phillips, Wu, and Yu (2011), provide the most suitable technique for bubble detection in prices.

Given the importance of detecting threats to the stability of financial markets, the limited understanding of asset-price bubbles in prior research is unsettling. The primary conflicts are both theoretically and methodologically based. The comprehensive and objective examination of asset-price bubble theory and detection models presented in this report make several significant contributions. Firstly, by synthesising the arguments from a large and growing strand of literature this report establishes a framework for comparison of the key existing techniques for detecting bubbles. Further, in identifying the limitations in the existing set of techniques this report prescribes a technique that is most suitable to the Australian market while also setting a foundation for future research that may develop enhanced detection methods. Finally, in finding little evidence of bubbles in Australian listed equities and property, this report contributes to optimal policy setting by regulatory bodies concerned with market stability and potential distortions.

References

Abreu, D., Brunnermeier, M. K., 2002. Synchronization risk and delayed arbitrage. Journal of Financial Economics 66 (2), 341–360.

Ackert, L. F., Charupat, N., Church, B. K., Deaves, R., 2002. Bubbles in experimental asset markets: Irrational exuberance no more. Federal Reserve Bank of Atlanta Working Paper 24.

Ackert, L. F., Smith, B. F., 1993. Stock price volatility, ordinary dividends, and other cash flows to shareholders. The Journal of Finance 48 (4), 1147–1160.

Al-Anaswah, N., Wilfling, B., 2011. Identification of speculative bubbles using state-space models with markov-switching. Journal of Banking & Finance 35 (5), 1073–1086.

Allen, F., Morris, S., Postlewaite, A., 1993. Finite bubbles with short sale constraints and asymmetric information. Journal of Economic Theory 61 (2), 206–229.

Blanchard, O. J., Watson, M. W., 1982. Bubbles, rational expectations and financial markets.

Bohl, M. T., 2003. Periodically collapsing bubbles in the us stock market? International Review of Economics & Finance 12 (3), 385–397.

Bowden, R. J., 1990. Predictive disequilibria and the short run dynamics of asset prices. Australian Journal of Management 15 (1), 65–87.

Brown, N. C., Wei, K. D., Wermers, R., 2013. Analyst recommendations, mutual fund herding, and overreaction in stock prices. Management Science 60 (1), 1–20.

Brunnermeier, M. K., 2008. Bubbles. The New Palgrave Dictionary of Economics 2.

Brunnermeier, M. K., Nagel, S., 2004. Hedge funds and the technology bubble. The Journal of Finance 59 (5), 2013–2040.

Campbell, J. Y., Shiller, R. J., 1986. Cointegration and tests of present value models. Tech. rep., National Bureau of Economic Research.

Chan, H. L., Woo, K.-Y., 2008. Testing for stochastic explosive root bubbles in asian emerging stock markets. Economics Letters 99 (1), 185–188.

Charemza, W. W., Deadman, D. F., 1995. Speculative bubbles with stochastic explosive roots: the failure of unit root testing. Journal of Empirical Finance 2 (2), 153–163.

Cheng, L.-Y., Yan, Z., Zhao, Y., Chang, W.-F., 2012. Short selling activity, price efficiency and fundamental value of ipo stocks. Pacific-Basin Finance Journal 20 (5), 809–824.

Chung, H., Lee, B.-S., 1998. Fundamental and nonfundamental components in stock prices of pacific-rim countries. Pacific-Basin Finance Journal 6 (3), 321–346.

Coates, J., 2012. The hour between dog and wolf: Risk-taking, gut feelings and the biology of boom and bust. HarperCollins UK.

Cochrane, J. H., 1991. A critique of the application of unit root tests. Journal of Economic Dynamics and Control 15 (2), 275–284.

Cochrane, J. H., 2002. Stocks as money: convenience yield and the tech-stock bubble. Tech. rep., National bureau of economic research.

Craine, R., 1993. Rational bubbles: A test. Journal of Economic Dynamics and Control 17 (5), 829–846.

Cuñado, J., Gil-Alana, L. A., De Gracia, F. P., 2005. A test for rational bubbles in the nasdaq stock index: a fractionally integrated approach. Journal of Banking & Finance 29 (10), 2633–2654.

Dasgupta, A., Prat, A., Verardo, M., 2011. The price impact of institutional herding. Review of Financial Studies 24 (3), 892–925.

Dass, N., Massa, M., Patgiri, R., 2008. Mutual funds and bubbles: The surprising role of contractual incentives. Review of Financial Studies 21 (1), 51–99.

DeMarzo, P. M., Kaniel, R., Kremer, I., 2008. Relative wealth concerns and financial bubbles. Review of Financial Studies 21 (1), 19–50.

Diba, B. T., Grossman, H. I., 1988. Explosive rational bubbles in stock prices? The American Economic Review, 520–530.

Dichev, I. D., Huang, K., Zhou, D., 2014. The dark side of trading. Journal of Accounting, Auditing & Finance 29 (4), 492–518.

Dufwenberg, M., Lindqvist, T., Moore, E., 2005. Bubbles and experience: An experiment. American Economic Review, 1731–1737.

Evans, G. W., 1991. Pitfalls in testing for explosive bubbles in asset prices. The American Economic Review, 922–930.

Flood, R. P., Hodrick, R. J., 1990. On testing for speculative bubbles. The Journal of Economic Perspectives 4 (2), 85–101.

Froot, K. A., Obstfeld, M., 1989. Intrinsic bubbles: The case of stock prices. Tech. rep., National Bureau of Economic Research.

Grinblatt, M., Titman, S., Wermers, R., 1995. Momentum investment strategies, portfolio performance, and herding: A study of mutual fund behavior. The American economic review, 1088–1105.

Grossman, S., 1976. On the efficiency of competitive stock markets where trades have diverse information. The Journal of finance 31 (2), 573–585.

Gürkaynak, R. S., 2008. Econometric tests of asset price bubbles: Taking stock. Journal of Economic Surveys 22 (1), 166–186.

Haldane, A., 2011. The race to zero, speech at international economic association sixteenth world congress. Beijing, China 8.

Hansen, L. P., 1982. Large sample properties of generalized method of moments estimators. Econometrica: Journal of the Econometric Society, 1029–1054.

Hansen, L. P., Sargent, T. J., 1981. Formulating and estimating dynamic linear rational expectations models. Rational Expectations and Econometric Practice 1, 91–126.

Haruvy, E., Noussair, C. N., 2006. The effect of short selling on bubbles and crashes in experimental spot asset markets. The Journal of Finance 61 (3), 1119–1157.

Hendershott, T., Jones, C. M., Menkveld, A. J., 2011. Does algorithmic trading improve liquidity? The Journal of Finance 66 (1), 1–33.

Hommes, C., Sonnemans, J., Tuinstra, J., Van De Velden, H., 2008. Expectations and bubbles in asset pricing experiments. Journal of Economic Behavior & Organization 67 (1), 116–133.

Hong, H., Scheinkman, J., Xiong, W., 2006. Asset float and speculative bubbles. The Journal of Finance 61 (3), 1073–1117.

Hong, H., Scheinkman, J., Xiong, W., 2008. Advisors and asset prices: A model of the origins of bubbles. Journal of Financial Economics 89 (2), 268–287.

Horvath, M. T., Watson, M. W., 1995. Testing for cointegration when some of the cointegrating vectors are prespecified. Econometric Theory 11 (05), 984–1014.

Jirasakuldech, B., Emekter, R., Rao, R. P., 2008. Do thai stock prices deviate from fundamental values? Pacific-Basin Finance Journal 16 (3), 298–315.

Kim, C.-J., 1994. Dynamic linear models with markov-switching. Journal of Econometrics 60 (1-2), 1–22.

Kim, C.-J., Nelson, 1999. State-space models with regime switching: classical and Gibbs-sampling approaches with applications. Vol. 2. MIT press Cambridge, MA.

Kirman, A., Teyssiere, G., 2005. Testing for bubbles and change-points. Journal of Economic dynamics and Control 29 (4), 765–799.

Kleidon, A. W., 1986. Variance bounds tests and stock price valuation models. The Journal of Political Economy, 953–1001.

Koustas, Z., Serletis, A., 2005. Rational bubbles or persistent deviations from market fundamentals? Journal of Banking & Finance 29 (10), 2523–2539.

Lakonishok, J., Shleifer, A., Vishny, R. W., 1992. The impact of institutional trading on stock prices. Journal of financial economics 32 (1), 23–43.

Lamont, O., 1998. Earnings and expected returns. The Journal of Finance 53 (5), 1563–1587.

Marimon, R., Spear, S. E., Sunder, S., 1993. Expectationally driven market volatility: an experimental study. Journal of Economic Theory 61 (1), 74–103.

Menkveld, A. J., 2013. High frequency trading and the new market makers. Journal of Financial Markets 16 (4), 712–740.

Mitchell, M., Pulvino, T., 2012. Arbitrage crashes and the speed of capital. Journal of Financial Economics 104 (3), 469–490.

Phillips, P. C., Shi, S.-P., Yu, J., 2013. Testing for multiple bubbles: limit theory of real time detectors.

Phillips, P. C., Wu, Y., Yu, J., 2011. Explosive behavior in the 1990s nasdaq: When did exuberance escalate asset values? International economic review 52 (1), 201–226.

Ritter, J. R., 2003. Behavioral finance. Pacific-Basin Finance Journal 11 (4), 429–437.

Said, S. E., Dickey, D. A., 1984. Testing for unit roots in autoregressive-moving average models of unknown order. Biometrika 71 (3), 599–607.

Scheinkman, J. A., Xiong, W., 2003. Overconfidence and speculative bubbles. Journal of political Economy 111 (6), 1183–1220.

Shiller, R. J., 1981a. Do stock prices move too much to be justified by subsequent changes in dividends? The American Economic Review 71 (3), 421–436.

Shiller, R. J., 1981b. The use of volatility measures in assessing market efficiency*. The Journal of Finance 36 (2), 291–304.

Shiller, R. J., 2000. Irrational exuberance.

Shiller, R. J., 2015. Irrational exuberance. Princeton University Press.

Sias, R. W., 2004. Institutional herding. Review of financial studies 17 (1), 165–206.

Smith, V. L., Suchanek, G. L., Williams, A. W., 1988. Bubbles, crashes, and endogenous expectations in experimental spot asset markets. Econometrica: Journal of the Econometric Society, 1119–1151.

Sornette, D., Von der Becke, S., 2011. Crashes and high frequency trading. Swiss Finance Institute Research Paper (11-63).

Tirole, J., 1982. On the possibility of speculation under rational expectations. Econometrica: Journal of the Econometric Society, 1163–1181.

Waters, G. A., 2008. Unit root testing for bubbles: a resurrection? Economics Letters 101 (3), 279–281.

West, K. D., 1987. A specification test for speculative bubbles. The Quarterly Journal of Economics 102 (3), 553–580.

Wu, Y., 1995. Are there rational bubbles in foreign exchange markets? evidence from an alternative test. Journal of International Money and Finance 14 (1), 27–46.

Wu, Y., 1997. Rational bubbles in the stock market: Accounting for the us stock-price volatility. Economic Inquiry 35 (2).

Appendix A: State-space model with regime switching - Estimation Set-Up

Step-by-step procedure of the Filter and the Approximate Maximum Likelihood Estimation of the Model

The filter for the state-space model with Markov switching is proposed by Kim (1994) and is a combination of the Kalman filter and the Hamilton filter. The Kim (1994) filter can be summarised into the following steps, which follow closely the summary provided by Kim and Nelson (1999) in Chapter 5.2.3 and Al-Anaswah and Wilfling (2011):

Step 1: Choose initial values for the Kalman filter

The Kalman filter treats the parameters from the state-space model as known. In reality, these parameters are unknown and we need to estimate the to get the initial parameters to start the Kalman filter.[4] We estimate the vector of parameter matrices, $\boldsymbol{\alpha}$, $\boldsymbol{\alpha} = (\mathbf{F}, \mathbf{H}, \mathbf{D}, \boldsymbol{\Omega}, \mathbf{R})$ using the following log likelihood function, Al-Anaswah and Wilfling (2011):

$$L(\boldsymbol{\alpha}|\mathbf{z}, \mathbf{g}) = const - 0.5 \sum_{t=1}^{T} (\ln[\det(\mathbf{H}\mathbf{P}_{\mathbf{t}|\mathbf{t-1}}\mathbf{H}' + \mathbf{R})] + \\ + \boldsymbol{\zeta}'_{t|t-1})(\mathbf{H}\mathbf{P}\mathbf{H}' + R)^{-1}\boldsymbol{\zeta}'_{t|t-1})'. \quad (49)$$

After we obtain the maximum-likelihood estimate of $\boldsymbol{\alpha}$, we can determine the smoothed estimates of the state vector and its error covariance matrix using the Kalman filter and the full-sample smoother as described in Step 2.

Step 2: Run the Kalman filter for states $i, j = 1, 2$

The estimates of $\beta^j_{t|\tau}$ and its covariance matrix can obtained using the Kalman filter defined by the following algorithm. Let $\beta^{(i,j)}_{t|\tau}$ denote the best linear mean-squared estimate of β_t in state j given the model and all observed information up to time τ and $S_t = j$ and $S_{t-1} = i$.

$$\boldsymbol{\beta}^{(i,j)}_{t|t-1} = \mathbf{F_j}\boldsymbol{\beta}^i_{t-1|t-1} \quad (50)$$

$$\mathbf{P}^{(i,j)}_{t|t-1} = \mathbf{F_j}\mathbf{P}^i_{t-1|t-1}\mathbf{F_j}' + \boldsymbol{\Omega}_j \quad (51)$$

[4] Matlab function "estimate", http://au.mathworks.com/help/econ/ssm.estimate.html

$$\boldsymbol{\zeta}_{t|t-1}^{(i,j)} = \mathbf{z_t} - \mathbf{H_j}\boldsymbol{\beta}_{t|t-1}^{(i,j)} - \mathbf{D_j}\mathbf{g_t} \tag{52}$$

$$\mathbf{f}_{t|t-1}^{(i,j)} = \mathbf{H_j}\mathbf{P}_{t|t-1}^{(i,j)}\mathbf{H_j}' + \mathbf{R_j} \tag{53}$$

$$\boldsymbol{\beta}_{t|t}^{(i,j)} = \boldsymbol{\beta}_{t|t-1}^{(i,j)} + \mathbf{P}_{t|t-1}^{(i,j)}\mathbf{H_j}'(\mathbf{f}_{t|t-1}^{(i,j)})^{-1}\boldsymbol{\zeta}_{t|t-1}^{(i,j)} \tag{54}$$

$$\mathbf{P}_{t|t}^{(i,j)} = [\mathbf{I} - \mathbf{P}_{t|t-1}^{(i,j)}\mathbf{H_j}'(\mathbf{f}_{t|t-1}^{(i,j)})^{-1}\mathbf{H_j}]\mathbf{P}_{t|t-1}^{(i,j)}, \tag{55}$$

where $\mathbf{P}_{t|\tau}^{(i,j)}$ is the mean-squared error matrix of the $\beta_{t|\tau}^{(i,j)}$ estimate, $\mathbf{f}_{t|t-1}^{(i,j)}$ is the conditional variance of the forecast error $\boldsymbol{\zeta}_{t|t-1}^{(i,j)}$. These equations are computed forward recursively.[5]

To get a more efficient estimates of the state vector and its error covariance matrix, Al-Anaswah and Wilfling (2011) propose using the full-sample smoother[6] computed backward recursively to get the vector β_t, given that $S_t = j$ and $S_{t+1} = k$:

$$\boldsymbol{\beta}_{t|T}^{(j,k)} = \boldsymbol{\beta}_{t|t}^{j} + \mathbf{J_t}^{(j,k)}(\boldsymbol{\beta}_{t+1|T}^{k} - \boldsymbol{\beta}_{t+1|t}^{(j,k)}) \tag{56}$$

$$\mathbf{P}_{t|T}^{(j,k)} = \mathbf{P}_{t|t}^{j} + \mathbf{J_t}^{(j,k)}(\mathbf{J}_{t+1|T}^{k} - \mathbf{J}_{t+1|t}^{(j,k)})\mathbf{J_t}^{(j,k)\text{`}}, \tag{57}$$

where $\mathbf{J_t}^{(j,k)} = \mathbf{P}_{t|t}^{j}\mathbf{F_k}'[\mathbf{P}_{t+1|t}^{(j,k)}]^{-1}$, $\boldsymbol{\beta}_{t|T}^{(j,k)}$ is an inference of β_t based on full sample, $\mathbf{P}_{t|T}^{(j,k)}$ is the mean-squared error matrix of $\boldsymbol{\beta}_{t|T}^{(j,k)}$.

Step 3: Calculate $Pr[S_t, S_{t-1}|\Psi_t]$ and $Pr[S_t|\Psi_t]$ using the Hamilton filter
This step is explained in detail in Chapter 5.2.2 in Kim and Nelson (1999).

Step 3.1
At the beginning of the t-th iteration, given the $Pr[S_{t-1} = i|\Psi_t]$, $i = 1, 2$, we calculate

$$Pr[S_t = j, S_{t-1} = i|\Psi_{t-1}] = Pr[S_t = j|S_{t-1} = i]Pr[S_{t-1} = i|\Psi_t], \tag{58}$$

where $Pr[S_t = j|S_{t-1} = i]$ is the transition probability.

Step 3.2
The joint density of z_t, S_t and S_{t-1} is given as

$$f(\mathbf{z_t}, S_t = j, S_{t-1} = i|\Psi_{t-1}) = f(\mathbf{z_t}|S_t = j, S_{t-1} = i, \Psi_{t-1}) \\ Pr[S_t = j, S_{t-1} = i|\Psi_{t-1}], \tag{59}$$

[5]Matlab function "filter", http://au.mathworks.com/help/econ/ssm.filter.html
[6]Matlab function "smooth", http://au.mathworks.com/help/econ/ssm.smooth.html

for $i, j = 1, 2$, and from which we can obtain the marginal density of $\mathbf{z_t}$ by

$$f(\mathbf{z_t}|\Psi_{t-1}) = \sum_{j=1}^{M}\sum_{i=1}^{M} f(\mathbf{z_t}|S_t = j, S_{t-1} = i, \Psi_{t-1}) Pr[S_t = j, S_{t-1} = i|\Psi_{t-1}], \quad (60)$$

where the conditional density $f(\mathbf{z_t}|S_t = j, S_{t-1} = i, \Psi_{t-1})$ can be obtained from the prediction error decomposition

$$f(\mathbf{z_t}|S_t = j, S_{t-1} = i, \psi_{t-1}) = (2\pi)^{-\frac{N}{2}} (\mathbf{f}_{t|t-1}^{(i,j)})^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{\zeta}_{t|t-1}^{(i,j)})'(\mathbf{f}_{t|t-1}^{(i,j)})^{-1}\boldsymbol{\zeta}_{t|t-1}^{(i,j)}\right\}, \quad (61)$$

for $i, j = 1, 2$, where $\boldsymbol{\zeta}_{t|t-1}^{(i,j)}$ and $\mathbf{f}_{t|t-1}^{(i,j)}$ are given by equations (52) and (53).

Step 3.3

After the end of time t, we can update the probability term $Pr[S_t = j, S_{t-1} = i|\Psi_{t-1}]$ and get

$$Pr[S_t = j, S_{t-1} = i|\Psi_t] = \frac{f(\mathbf{z_t}|S_t = j, S_{t-1} = i, \Psi_{t-1}) f(S_t = j, S_{t-1} = i|\Psi_{t-1})}{f(\mathbf{z_t}|\Psi_{t-1})}, \quad (62)$$

for $i, j = 1, 2$.

Step 4: Collapse $M \times M$ posteriors into $M \times 1$

We use the probability terms from Step 3.3 to collapse the $M \times M$ posteriors into $M \times 1$ equations. In particular, the term $\boldsymbol{\beta}_{t|t}^{j}$ from equation (54) and $\mathbf{P}_{t|t}^{j}$ from equation (55) can be collapsed into $M \times 1$ equations using the following approximations:

$$\boldsymbol{\beta}_{t|t}^{j} = \frac{\sum_{i=1}^{M} Pr[S_{t-1} = i, S_t = j|\psi_t]\boldsymbol{\beta}_{t|t}^{(i,j)}}{Pr[S_t = j|\Psi_t]} \quad (63)$$

$$\mathbf{P}_{t|t}^{j} = \frac{\sum_{i=1}^{M} Pr[S_{t-1} = i, S_t = j|\psi_t][\mathbf{P}_{t|t}^{(i,j)} + (\boldsymbol{\beta}_{t|t}^{j} - \boldsymbol{\beta}_{t|t}^{(i,j)})(\boldsymbol{\beta}_{t|t}^{j} - \boldsymbol{\beta}_{t|t}^{(i,j)})']}{Pr[S_t = j|\Psi_t]}. \quad (64)$$

The derivation of these approximation steps is provided by in Kim and Nelson (1999) on pages 100-102.

7 Appendix B: Matlab Toolbox User Guide

Asset Bubble

The Asset Bubble Toolbox showcases some of the different techniques for detecting rational bubbles in price and dividend data.

Many algorithms for detecting asset bubbles in price and dividend data have been proposed over the last 50 years. In this toolbox a subset of the the total literature is explored, prodiving an overview of how rational bubble detection methods have evolved over time.

The toolbox provides methods for executing bubble detection tests on real price and dividend data, and exposes useful summary results and plots to help make inference about the existence of bubbles in the data being analysed.

Quick Links

- **Loading Test Data**
- **Variance Bounds Test**
- **West's Test**
- **Unit Root Tests**
- **State-Space Markov Switching Test**

Function and Class Reference

- **Functions and classes provided in the Asset Bubble Toolbox**

Demonstrations

- **Robert Shiller's Variance Bounds test**
- **Keneth West's test**
- **Phillips, Shi, and Yu's SADF and GSADF tests**
- **Al-Anaswah and Wilfling's State-space Markov switching test**

Published with MATLAB® R2015b

Loading Test Data

Contents

Overview

Testing for bubbles in price and dividend data can be performed on any price and dividend time series and the Asset Bubble Toolbox makes no restrictions on assumed sample data.

Historically, tests have always been peformed on Robert Shiller's S&P 500 datasets which are freely available from http://aida.wss.yale.edu/~shiller/data.htm or http://www.econ.yale.edu/~shiller/data/chapt26.html.

As a convenience, the Asset Bubble Toolbox includes latest versions of these datasets at time of toolbox publication. In addition, convenience functions have been provided to allow quick loading of the data into MATLAB variables.

In all cases, Microsoft Excel is a prerequisite in order to use the parameterised loading functions (all data files are excel files). If Excel is unavailable, in some cases the data can be still be loaded but with pre-determined parameter settings.

Shiller's Chapter 26 data

This data is sourced from http://www.econ.yale.edu/~shiller/data/chapt26.html, and contains S&P 500 yearly composite price and dividend data for the years 1871 to 1999. The data has been modified slightly to include an additional PPI column with PPI 1979 = 100 (by default, the data has PPI 1982 = 100).

To load data, use the *LoadShillerChap26Data* function. *LoadShillerChap26Data* takes two parameters:

- maxYear - The maximum year to include data for
- scaleByPPI - Boolean (true/false) indicating if all prices should be made real by scaling by PPI relative to PPI in the final observation year

The following example shows how to load Shiller's data from 1871 to 1979 inclusive (as used in Shiller's original paper).

```
[Dates, Prices, Dividends, PPI] = LoadShillerChap26Data(1979, true);
```

If Excel is not available, use the following command to load the data variables into the workspace. In this case maxYear is fixed at 1979 and scaleByPPI is always true.

```
load VarianceBoundsDemoData
```

For a version of the data with maxYear fixed at 1980 and scaleByPPI set to true use

```
load WestsTwoStepDemoData
```

Shiller's S&P monthly data

This data is sourced from http://aida.wss.yale.edu/~shiller/data.htm, and contains S&P 500 composite monthly price and dividend data for the years 1871 to current.

To load data, use the *LoadShillerTestData* function. *LoadShillerTestData* automatically rolls monthly data up to yearly data which is the format primarily used in asset bubble detection research papers. *LoadShillerTestData* takes two parameters:

- maxYear - The maximum year to include data for
- scaleByCPI - Boolean (true/false) indicating if all prices should be made real by scaling by CPI relative to CPI in the final observation year.

The following example shows how to load Shiller's data from 1871 to 1980 inclusive (as used in West's original paper).

```
[Dates, Prices, Dividends, CPI] = LoadShillerTestData(1980, true);
```

Phillips, Shi, and Yu's price-dividend ratio data

This data is sourced from https://sites.google.com/site/shupingshi/home/research and contains sample data use by Phillips, Shi, and Yu (PSY) in their paper on SADF and GSADF testing procedures.

The following example shows how to load PSY's data which runs from 1871 to 2010 inclusive (as used in PSY's original paper).

```
[Dates, PriceOnDividend] = LoadUnitRootTestData();
```

If Excel is not available, use the following command to load the data variables into the workspace.

```
load UnitRootDemoData
```

Published with MATLAB® R2015b

Variance Bounds Test

The Asset Bubble Toolbox includes functions for performing variance bounds tests.

To conduct a test, first load some data and create an instance of the *VarianceBounds* class. Then call *PerformTest* to obtain an instance of the *VarianceBoundsResult* class, which can be used for easy plotting and table generation.

Try running the following example:

```
load VarianceBoundsDemoData.mat
VB = VarianceBounds(Dates, Prices, Dividends);
VBR = VB.PerformTest(true, [1.3 1 .7]);
VBR.PlotRealPriceAgainstExPostRationalPrice();
ResultsTable = VBR.GenerateResultsTable()
```

For more information see:

- Variance Bounds Test Demonstration
- Variance Bounds Test Documentation

Published with MATLAB® R2015b

West's Test

The Asset Bubble Toolbox includes functions for performing West's test for speculative bubbles.

To conduct a test, first load some data and create an instance of the *WestsTwoStep* class. Then call *PerformTest* to obtain the test result.

Try running the following example:

```
load WestsTwoStepDemoData.mat
WTS = WestsTwoStep(Dates, Prices, Dividends);
TestResults = WTS.PerformTest(2, false)
```

For more information see:

- West's Test Demonstration
- West's Test Documentation

Published with MATLAB® R2015b

Unit Root Tests

The Asset Bubble Toolbox includes functions for performing different types of unit root tests.

To conduct a test, first load some data and create an instance of the *UnitRootBubbleTest* class. Then call one of the unit root test methods, depending on which test type you would like to perform. Running a test returns and instance of *UnitRootBubbleTestResult* which can be used for easy plotting and table generation.

Try running the following example:

```
load UnitRootDemoData.mat
URBT = UnitRootBubbleTest(Dates, PriceOnDividend);

EqType = UnitRootNullTestType.UnitRootWithDrift;
Tw = 36;
k = 0;
M = 2000;
d = 1;
eta = 1;
quants = [.9 .95 .99];
UseMex = true;
URBTR = URBT.PerformSADFTest(EqType, Tw, k, M, d, eta, quants, UseMex);

URBTR.GenerateResultsTable()
URBTR.GeneratePlot(0.95);
URBTR.GenerateBubbleStartEndTimes(0.95)
```

For more information see:

- Unit Root Test Demonstration
- Unit Root Test Documentation

Published with MATLAB® R2015b

Variance Bounds test for bubbles

Robert Shiller's variance bounds test for asset bubbles in price and dividend data

Contents

Method

Robert Shiller's variance bounds test considers the implications of comparing the efficient markets model against a perfect foresight model which is commonly referred to as the *ex post* rational price.

The efficient martkets model is represented as:

$$p_t = \sum_{i=0}^{\infty} \frac{E_t(d_{t+i})}{(1+r)^i}$$

where p_t is the real price, d_t is the real dividend, r is the constant interest rate, and $E_t()$ is the expectation operator conditional on information available at time t.

The *ex post* rational price is defined as:

$$p_t^* = \sum_{i=0}^{\infty} \frac{d_{t+i}}{(1+r)^i}$$

where p_t^* is the *ex post* rational price.

The test compares the standard deviation of the two series. If the efficient markets model holds true, then it can be shown that:

$$\sigma(p_t) \leq \sigma(p_t^*)$$

Demonstration

Shiller's original application of the test was to S&P 500 annual price and dividend data from 1871 to 1979.

Load Shiller's test data and create an instance of the *VarianceBounds* class

```
load VarianceBoundsDemoData.mat
VB = VarianceBounds(Dates, Prices, Dividends);
```

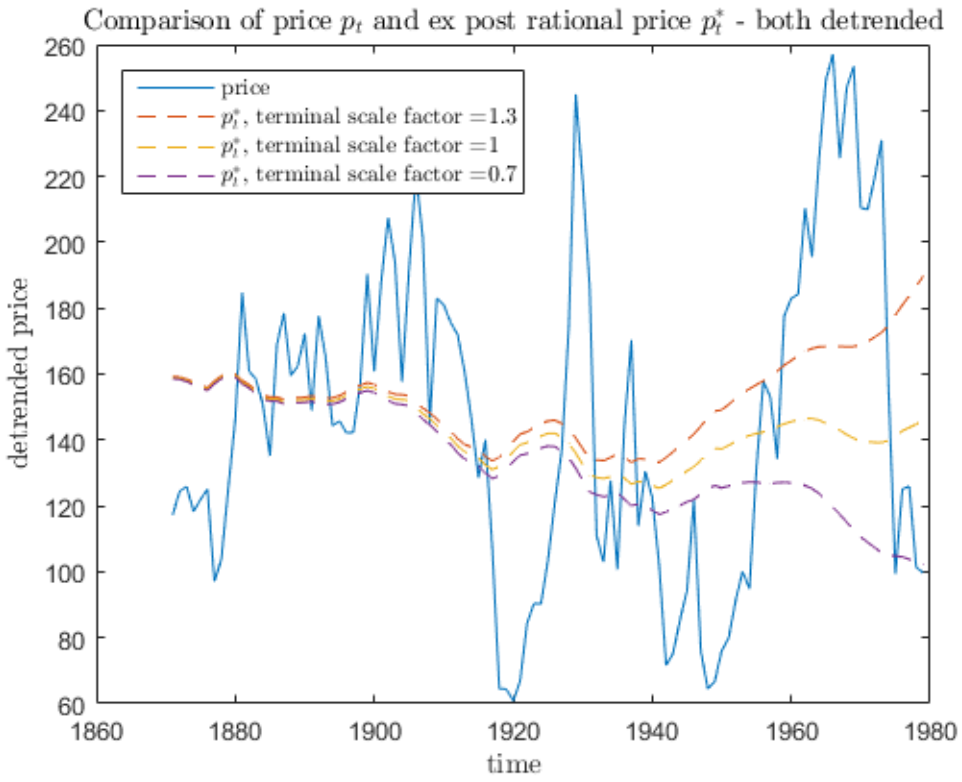
The *VarianceBounds* class has a method called *PerformTest* which takes two arguments called *shouldDetrend* and *terminalValueScales*. *shouldDetrend* is a boolean (true or false), which should be set to true if detrending of the data is required. *terminalValueScales* is a 1xN matrix of scalings around 1.0 for the terminal condition of the *ex post* rational price.

In Shiller's original application, the data was detrended so we will do the same here. Additionally, we will specify some scalings for the terminal condition of the *ex post* rational price to see how they affect the test result.

```
VBR = VB.PerformTest(true, [1.3 1 .7]);
```

PerformTest returns an instance of the *VarianceBoundsResult* class, which contains methods for plotting and displaying test results. Let's plot the real price and ex post rational price series together.

```
VBR.PlotRealPriceAgainstExPostRationalPrice();
```



The plot above is essentially the same plot that Shiller presented. For comparison we show the different *ex post* rational price series. Shiller's terminal condition on the *ex post* rational price is the mean of the detrended price series (*PerformTest* uses the same condition).

Now let's print the test result information. Each column of the results table corresponds to a different end point scaling of the *ex post* rational price, with 1.0 representing the mean of the detrended price series.

```
ResultsTable = VBR.GenerateResultsTable()
```

```
ResultsTable =

                        Results
                ____________________________

    E(p)          145.92      145.92      145.92
    E(d)          6.7374      6.7374      6.7374
    r bar       0.046172    0.046172    0.046172
    r2 bar      0.094475    0.094475    0.094475
    b           0.015796    0.015796    0.015796
    cor(p,p*)    0.35875     0.26876    0.031906
    std(d)         1.4116      1.4116      1.4116
    std(p)         48.756      48.756      48.756
```

```
std(p*)        12.949      9.6057      16.224
```

The standard deviation of the price series, std(p), clearly exceeds the standard deviation of each *ex post* rational price series, std(p*).

One interpretation of this result is that the price and dividend data contains bubbles. However, we should be cautious here as the failure of the test does not directly imply the existence of bubbles. Essentially Shiller's variance bounds test is a validation test of the efficient markets model and not a direct test for rational bubbles.

Published with MATLAB® R2015b

West's test for speculative bubbles

Kenneth West's specification test for speculative bubbles in price and dividend data.

Contents

Method

Unlike the Variance Bounds test which doesn't directly include the presence of bubbles when rejecting the model, West's test specifically includes the ability to check for bubbles by testing the model and no-bubbles hypotheses sequentially.

The Euler equation relating prices and dividends is given by:

$$p_t = bE_t(p_{t+1} + d_{t+1})$$

where p_t is the real stock price in period t, b the constant real discount rate $0 < b = 1/(1+r) < 1$, r the constant expected return, and $E_t()$ is the expectation operator conditional on information available at time t.

In the absence of bubbles an estimate of the discount rate can be obtained by noting that

$$p_t = b(p_{t+1} + d_{t+1}) + u_{t+1}$$

where u_{t+1} is the error term. Since u_{t+1} is correlated with p_{t+1} and d_{t+1}, the estimation requires use of Instumental Variables.

Next, West assumes dividends follow a pure auto-regressive process, order q, of the form

$$d_{t+1} = \mu + \phi_1 d_t + \ldots + \phi_q d_{t-q+1} + v_{t+1}$$

Estimates for the dividend regression coefficients are readily obtained by OLS.

Finally, the usual asset price equation

$$p_t = \sum_{i=1}^{\infty} b^i E_{t+1}(d_{t+i})$$

is estimated by letting

$$p_{t+1} = m + \delta_1 d_{t+1} + \ldots + \delta_q d_{t-q+2} + w_{t+1}$$

Because each estimation process above represents part of the relationship between prices and dividends it is possible to derive constraint equations relating all the estimated model parameters. Under the assumption that no bubbles are present, the constraint equations should all be satisfied. A statistical test is constructed to measure how closely the constraints are satisfied.

It is important to note that West's test hinges on being able to validate the discount rate and dividend ARIMA estimation processes. Provided these estimates are sound, model mispecification can be ruled out and any violation of the constraint equations can be directly attributed to the presence of bubbles.

Demonstration

West's original application of the test was to S&P 500 annual price and dividend data from 1871 to 1980.

Load West's test data and create an instance of the *WestsTwoStep* class

```
load WestsTwoStepDemoData.mat
WTS = WestsTwoStep(Dates, Prices, Dividends);
```

The *WestsTwoStep* class has a method called *PerformTest* which takes two arguments called *ARq* and *UseDifferencedData*. ARq is the auto-regressive order. *UseDifferencedData* is a boolean (true or false), which controls if the test is run on level (false) or differenced (true) data.

In West's original application, ARq was set at 2 and 4 and both level and differenced data were considered. For now, let's choose ARq = 2 and level data.

```
TestResults = WTS.PerformTest(2, false)
```

```
TestResults =

            DataSize: 108
             DataSet: '1873-1980'
         Differenced: 'no'
                   q: 2
                 DoF: 3
       TestStatistic: 0.5224
    TestStatisticSig: 0.9139
```

TestResults is a MATLAB structure array with metrics about the test process. We can see that the test was run on data covering 1873 to 1980 and produced a highly significant test statistic of 0.5224. In this case, we reject the hypothesis that bubbles exist in the data. We also need to validate each estimation process used.

Published with MATLAB® R2015b

Unit Root tests for bubbles

Contents

Method

Unit Root tests rely on testing the relationship between the price and dividend ratio series for stationarity. If the price and dividend series are cointegrated they share the same stochastic drift, which is counter to the hypothesis of bubbles existing. The testing procedures broadly fall under the term *Unit Root* testing.

The simplest test for a unit root is the *augmented Dickey-Fuller* (ADF) test. In this testing scenario, the entire sample is considered and a single test result is returned. The power of a simple ADF test to detect bubbles is weak in the sense that it doesn't yield useful information about the start and end dates of bubble episodes, and it can fail if more than one episode occurs.

To get around the limitations of the simple ADF test Phillips, Wu and Yu (2011, PWY) propose an extension called the Supremum ADF test (SADF) which considers an increasing sample window size, starting from a mimimum window and ending at the full sample. Their test then calculates the Supremum of all ADF tests conducted on all sub-sample windows and compares that to appropriate critical values derived by extensive Monte Carlo simulations. The SADF test also offers a date-stamping strategy whereby bubble start and end dates can be successfully identified. Although the SADF test is extremely useful it has some shortcomings, not least of which is that it can fail to find bubbles when considering the full sample size but can successfully find bubbles when considering each half of the full sample separately.

The answer to the failings of the SADF test is the *Generalized supremum ADF* test (GSADF) as proposed by Phillips, Shi, and Yu (2013, PSY). The GSADF test extends the concept of the SADF test by allowing the window start point to also move, thereby covering many more sub-samples of the entire sample size. The GSADF test has been shown to be extremely effective in identify commonly accepted bubble episodes in historical data, and it also offers the same data-stamping strategies as the SADF test. Note that the data-stamping strategies for SADF and GSADF are typically run on backwards tests, rather than forwards, to provide test results at the most recent sample point (i.e. the windows start at the most recent points and extend back in time, rather than forwards in time).

SADF Demonstration

Let's explore PSY's application of the SADF test to the S&P 500, sampled monthly from January 1871 to December 2010.

Load PSY's test data and create an instance of the *UnitRootBubbleTest* class

```
load UnitRootDemoData.mat
URBT = UnitRootBubbleTest(Dates, PriceOnDividend);
```

The *UnitRootBubbleTest* class has methods for performing ADF, SADF, and GSADF tests. We will perform an SADF test, with parameters specified exactly as in PSY's paper. Specifically, we will test for:

- a unit root with drift
- a minimum window size of 36 months
- zero lag terms included in each ADF test
- 2000 Monte Carlo simulations when computing critical values

- Null model parameters d and η set to 1
- Critical value tests performed at 90%, 95%, and 99%

In addition, we can supply a final boolean parameter *UseMex* to indicate if we want the software to use compiled code when performing tests. Since the simulations can take a long time to compute it is ideal to speed things up as much as possible. In general, always set UseMex to true unless you need to modify the code.

```
EqType = UnitRootNullTestType.UnitRootWithDrift;
Tw = 36;
k = 0;
M = 2000;
d = 1;
eta = 1;
quants = [.9 .95 .99];
UseMex = true;
URBTR = URBT.PerformSADFTest(EqType, Tw, k, M, d, eta, quants, UseMex);
```

PerformSADFTest returns an instance of the *UnitRootBubbleTestResult* class, which contains methods for plotting and displaying test results. Let's start by examining the test result by calling *GenerateResultsTable*

```
URBTR.GenerateResultsTable()
```

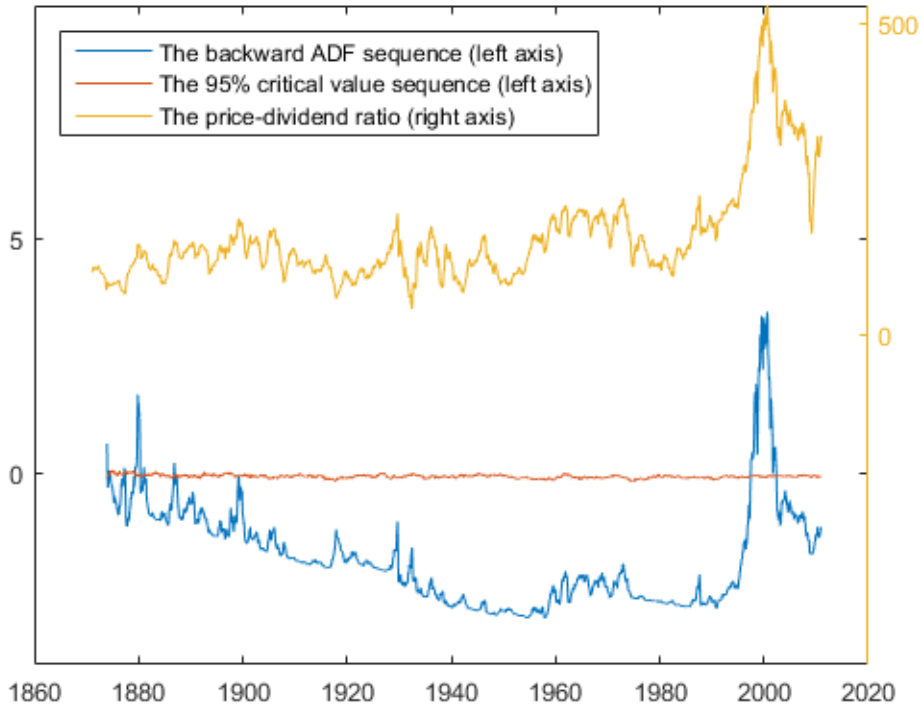
```
ans =

            Test_Stat    CV_90     CV_95    CV_99
            _________    ______    _____    ______

    SADF    3.4432       1.4218    1.659    2.0803
```

It is clear that the SADF statistic exceeds all critival values at 90%, 95% and 99%. On this basis, we conclude that the data contains bubbles. Now let's plot the sample sequence, the backwards ADF statistic sequence, and the 95% critical value sequence.

```
URBTR.GeneratePlot(0.95);
```



Here we can clearly see that there are points at which the backwards ADF sequence exceeds the critical value sequence. Finally, let's run the date-stamping strategy to get estimates for bubble start and end dates.

```
URBTR.GenerateBubbleStartEndTimes(0.95)
```

```
ans =

                StartDate    EndDate
                _________    _______

    Bubble1     1879.4       1880.4
    Bubble2     1997.5       2002.5
```

Two bubble episodes are identified. Note that the date-stamping strategy imposes a mimimum window size for an episode to be classed as a bubble, which is set at $\log_{10}(T)$ as per guidelines in PSY's paper.

GSADF Demonstration

Let's continue by exploring PSY's application of the GSADF test to the S&P 500, sampled monthly from January 1871 to December 2010. Parameters for the GSADF test are identical for the SADF test above.

Run the GSADF test and examine results

```
URBTR = URBT.PerformGSADFTest(EqType, Tw, k, M, d, eta, quants, UseMex);
URBTR.GenerateResultsTable()
```

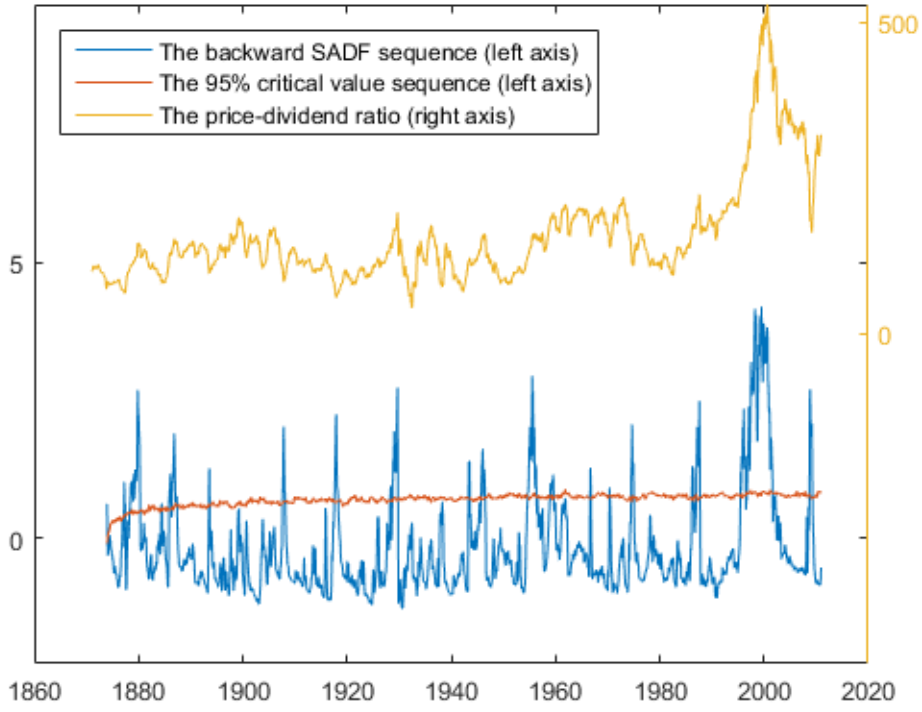
```
ans =

              Test_Stat    CV_90     CV_95     CV_99
              _________    ______    ______    ______

    GSADF     4.2069       2.5791    2.8091    3.1982
```

It is clear that the GSADF statistic exceeds all critival values at 90%, 95% and 99%. On this basis, we conclude that the data contains bubbles. Plot the sample sequence, the backwards SADF statistic sequence, and the 95% critical value sequence.

```
URBTR.GeneratePlot(0.95);
```



Here we can clearly see that there are points at which the backwards SADF sequence exceeds the critical value sequence. Finally, let's run the date-stamping strategy to get estimates for bubble start and end dates. When estimating dates using the GSADF test, *GenerateBubbleStartEndTimes* requires two additional parameters. *MinDurationInYears* controls the minimum size for a bubble episode to register as a bubbler, and *SamplesPerYear* represents the number of data points over a year in the sample. PSY look for bubbles greater than 0.5 years using monthly data.

```
MinDurationInYears = 0.5;
SamplesPerYear = 12;
URBTR.GenerateBubbleStartEndTimes(0.95, MinDurationInYears, SamplesPerYear)
```

```
ans =

                StartDate    EndDate
```

Bubble1	1878.3	1880.4
Bubble2	1886.5	1887.5
Bubble3	1907.6	1908.2
Bubble4	1917.6	1918.4
Bubble5	1928.7	1929.8
Bubble6	1945.8	1946.5
Bubble7	1954.7	1956.4
Bubble8	1987	1987.8
Bubble9	1995.5	1996.5
Bubble10	1996.8	2001.7
Bubble11	2008.8	2009.4

Eleven bubble episodes are identified. These episodes correspond to all the currently accepted major periods of exuberance and collapse in the S&P 500 data.

Published with MATLAB® R2015b

Functions and classes provided in the Asset Bubble Toolbox

Variance Bounds Test

- **VarianceBounds** class
- **VarianceBoundsResult** class

West's Test

- **AROrderPicker** class
- **DiscountRateEstimator** class
- **WestsCovarianceMatrix** class
- **WestsDataBuilder** class
- **WestsSymbolicComps** class
- **WestsTwoStep** class

Unit Root Tests

- **ADF_FL** function
- **CriticalValuesCache** class
- **UnitRootBubbleTest** class
- **UnitRootBubbleTestResult** class
- **UnitRootNullSimulator** class
- **UnitRootNullTestType** enumeration
- **UnitRootTests** class
- **UnitRootTestsCaller** function
- **UnitRootTestsCaller_mex** MEX file
- **UnitRootTestType** enumeration

State-space Markov Switching Tests

- Not currently implemented...

Published with MATLAB® R2015b

VarianceBounds class

Contents

Overview

Class for computing variance bounds test for price and dividend time series data using the method of Shiller (1981b).

Robert Shiller's variance bounds test considers the implications of comparing the efficient markets model against a perfect foresight model which is commonly referred to as the *ex post* rational price.

The efficient martkets model is represented as:

$$p_t = \sum_{i=0}^{\infty} \frac{E_t(d_{t+i})}{(1+r)^i}$$

where p_t is the real price, d_t is the real dividend, r is the constant interest rate, and $E_t()$ is the expectation operator conditional on information available at time t.

The *ex post* rational price is defined as:

$$p_t^* = \sum_{i=0}^{\infty} \frac{d_{t+i}}{(1+r)^i}$$

where p_t^* is the *ex post* rational price.

The test compares the standard deviation of the two series. If the efficient markets model holds true, then it can be shown that:

$$\sigma(p_t) \leq \sigma(p_t^*)$$

Constructor

```
VB = VarianceBounds(dates, prices, dividends);
```

Inputs:

- *dates* is an Nx1 vector of increasing dates for each point in the sample data
- *prices* is an Nx1 vector of prices for each point in the sample data
- *dividends* is an Nx1 vector of dividends for each point in the sample data

Outputs:

- *VB* is the constructed VarianceBounds class

PerformTest

```
VBR = VB.PerformTest(shouldDetrend, terminalValueScales);
```

Peforms the variance bounds test, optionally detrending data prior to running the test.

Inputs:

- *shouldDetrend* is a boolean, indicating if the data should be detrended via a log regression before running the test
- *terminalValueScales* is a 1xM array of scalings around 1.0 for the terminal condition of the *ex post* rational price

Outputs:

- *VBR* is an instance of VarianceBoundsResult which can be used to generate plots and tables

---

Published with MATLAB® R2015b

VarianceBoundsResult class

Contents

Overview

Class representing the results from running a variance bounds test. Contains useful methods for generating plots and tables of results.

Constructor

Users should not explicity construct instances of *VarianceBoundsResult*. Instead, they should call *PerformTest(...)* on an instance of *VarianceBounds* to return a constructed *VarianceBoundsResult*, referred to below as VBR.

GenerateResultsTable

```
[tab] = VBR.GenerateResultsTable();
```

Generates table of test results similar to that found in Shiller's paper (p431). The final rows in the table can be used to accept of reject the test for bubbles.

Outputs:

- *tab* is a MATLAB table object.

PlotRealPriceAgainstExPostRationalPrice

```
VBR.PlotRealPriceAgainstExPostRationalPrice();
```

Generates a new figure and plots an overlay of the *ex post* rational price on the real price. If different *ex post* rational price terminal scalings have been supplied, each is plotted as a different coloured series.

PlotExPostRationalPrice

```
VBR.PlotExPostRationalPrice();
```

Plots the *ex post* rational price series. If different *ex post* rational price terminal scalings have been supplied, each is plotted as a different coloured series.

This is useful for comparing how the terminal scalings affect the computed values for the *ex post* rational price.

PlotDetrendRegressionDetails

```
VBR.PlotDetrendRegressionDetails();
```

If the user requested detrending of the price and dividend data prior to performing the test, the regression process can be plotted via this function.

AROrderPicker - class

Contents

Overview

Determines the best pure AR order for a given time series. AROrderPicker uses Akaike and Bayesian information criteria to pick the best pure AR order (i.e. there is no MA component).

Constructor

```
AROP = AROrderPicker(Data);
```

Inputs:

- *Data* is the input time series data

Outputs:

- *AROP* is the constructed AROrderPicker class

SetData

```
AROP.SetData(Data)
```

Sets the data on the AROrderPicker class instance.

Inputs:

- *Data* is the input time series data

PickBestAROrder

```
[pAIC, pBIC, aic, bic] = AROP.PickBestAROrder(MaxP, d)
```

Picks the best AR order *p* for an *ARIMA(p,d,0)* model using Akaike and Bayesian Information Criteria.

Inputs:

- *MaxP* is the maximum number of AR lags to consider
- *d* is the degree of differencing in the ARIMA model

Outputs:

- *pAIC* is the best AR order using Akaike IC
- *pBIC* is the best AR order using Bayesian IC
- *aic* is the Akaike IC object returned from MATLAB's aicbic function
- *bic* is the Bayesian IC object returned from MATLAB's aicbic function

DiscountRateEstimator - class

NOTE: *DiscountRateEstimator* makes use of Mike Cliff's freely available GMM and MINZ libraries (see https://sites.google.com/site/mcliffweb/programs).

Contents

Overview

Estimates the discount rate b from the arbitrage (Euler) equation relating prices and dividends.

The arbitrage relationship is given by

$$p_t = b * E_t(p_{t+1} + d_{t+1})$$

This can be rewritten in the form

$$p_t = b * (p_{t+1} + d_{t+1}) + u_{t+1}$$

where u_{t+1} is potentially correlated with p_{t+1} and d_{t+1}. The esimtation uses Instrumental Variables, with d_t or $\Delta(d_t)$ as Instruments

Constructor

```
DRE = DiscountRateEstimator(Data);
```

Inputs:

- *Data* is a structure array instance of the data object returned by *WestsDataBuilder.GenerateData*.

Outputs:

- *DRE* is the constructed DiscountRateEstimator class

EstimateDiscountRate

```
EDR = DRE.EstimateDiscountRate(UseSimpleMethod);
```

Estimates the discount rate.

Inputs:

- *UseSimpleMethod* is boolean. When true, uses simple 2SLS IV estimation. When false uses two-stage GMM estimation, with the first stage being simple IV estimation and the second stage being application of GMM to yield the best heteroskedasticity consistent estimate.

Outputs:

- *EDR* is a structure array with fields for the results, as well as the data used. *EDR* varies depending on which estimation technique is used.

WestsCovarianceMatrix - class

Contents

Overview

Computes West's variance-covariance matrix *V* for the vector Θ of regression coefficients (West 1987, p563).

Assuming the order of the AR process is q, the coefficients vector Θ with dimenstions $(2q+3) \times 1$ is represented as

$$\Theta = (b, \mu, \phi_1, \ldots, \phi_q, m, \delta_1, \ldots, \delta_q)'$$

where

- b is the estimated discount rate from the arbitrage (Euler) equation
- μ is the d_t regression intercept
- $\phi_1, \ldots, \phi_q$ are the d_t regression coefficients
- m is the p_{t+1} regression intercept
- $\delta_1, \ldots, \delta_q$ are the p_{t+1} regression coefficients

Constructor

```
WCM = WestsCovarianceMatrix(Data,Sd_hat,b_hat,phi_hat,delta_hat,IsDifferenced);
```

Inputs:

- *Data* is a structure array instance of the data object returned by *WestsDataBuilder.GenerateData*.
- *Sd_hat* is an estimate of the spectral density matrix from estimating the arbitrage equation, size *(q+1)x(q+1)*
- *b_hat* is an estimate of b
- *phi_hat* is a vector containing estimates for μ and ϕ, size *(q+1)x1*
- *delta_hat* is a vector containing estimates for m and δ, size *(q+1)x1*
- *IsDifferenced* is boolean (true/false) and indicates if level or differenced data is being used.

Outputs:

- *WCM* is the constructed WestsCovarianceMatrix class

Computed Property *V*

The variance-covariance matrix is automatically computed during class construction and is available to access via dot(.) notation.

```
CovMat = WCM.V;
```

Published with MATLAB® R2015b

WestsDataBuilder - class

Contents

Overview

Class for building the different data matrices used in West's speculative test for bubbles.

Constructor

```
WDB = WestsDataBuilder(Prices, Dividends, ARq, UseDifferencedData);
```

Inputs:

- *Prices* is an Nx1 vector of prices for each point in the sample data
- *Dividends* is an Nx1 vector of dividends for each point in the sample data
- *ARq* is the auto-regressive order used to represent dividends in the model
- *UseDifferencedData* is boolean (true/false) and indicates if level or differenced data should be constructed.

Outputs:

- *WDB* is the constructed WestsDataBuilder class

GenerateData

```
Data = WDB.GenerateData();
```

Construct the data used in West's test process.

Outputs:

- *Data* is structure array with the following fields
- *Data.pt* is the price series
- *Data.dt* is the dividend series
- *Data.ptplus1* is the price series at time $t+1$ (or the 1st Δ of it)
- *Data.dtplus1* is the dividend series at time t+1 (or the 1st Δ of it)
- *Data.X* is (price + dividend) at time $t+1$
- *Data.D* is the instruments (lagged dividends) at time t (or the 1st Δ of it)
- *Data.Dplus1* is the instruments (lagged dividends) at time $t+1$ (or the 1st Δ of it)

Published with MATLAB® R2015b

WestsSymbolicComps - class

Contents

Overview

Class to compute symbolic math structures for West's coefficients

Assuming the order of the AR process is q, the coefficients vector Θ with dimenstions $(2q+3) \times 1$ is represented as

$$\Theta = (b, \mu, \phi_1, \ldots, \phi_q, m, \delta_1, \ldots, \delta_q)'$$

where

- b is the estimated discount rate
- μ is the d_t regression intercept
- $\phi_1, \ldots, \phi_q$ are the d_t regression coefficients
- m is the p_{t+1} regression intercept
- $\delta_1, \ldots, \delta_q$ are the p_{t+1} regression coefficients

Two constraint equations are constructed:

- R1 - For level data: $d_t \sim AR(q)$
- R2 - For differenced data: $\Delta(d_t) \sim AR(q)$

The jacobian matrix is also derived for each constraint vector

- dR1 = $\frac{\partial R1}{\partial \Theta}$
- dR2 = $\frac{\partial R2}{\partial \Theta}$

All structures are returned as symbolic math objects and can be evaluated with real values for the coefficients via subs.

For example, suppose the have the real calculated coefficients in a (2q+3)x1 vector Coeffs. We can evaluate the dR1 jacobian using

```
subs(dR1,Theta, Coeffs)
```

Constructor

```
WSC = WestsSymbolicComps(ARq);
```

Outputs:

- *WSC* is the constructed WestsSymbolicComps class

Computed Properties

The following properties are automatically computed during class construction and are available to access via dot(.) notation.

- *WSC.R1* is the level data symbolic constraint vector
- *WSC.R2* is the differenced data symbolic constraint vector
- *WSC.dR1* is the level data symbolic jacobian matrix
- *WSC.dR2* is the differenced data symbolic jacobian matrix

Published with MATLAB® R2015b

WestsTwoStep - class

Contents

Overview

Class for computing Kenneth West's specification test for speculative bubbles in price and dividend data.

Unlike the Variance Bounds test which doesn't directly include the presence of bubbles when rejecting the model, West's test specifically includes the ability to check for bubbles by testing the model and no-bubbles hypotheses sequentially.

The Euler equation relating prices and dividends is given by:

$$p_t = bE_t(p_{t+1} + d_{t+1})$$

where p_t is the real stock price in period t, b the constant real discount rate $0 < b = 1/(1+r) < 1$, r the constant expected return, and $E_t()$ is the expectation operator conditional on information available at time t.

In the absence of bubbles an estimate of the discount rate can be obtained by noting that

$$p_t = b(p_{t+1} + d_{t+1}) + u_{t+1}$$

where u_{t+1} is the error term. Since u_{t+1} is correlated with p_{t+1} and d_{t+1}, the estimation requires use of Instumental Variables.

Next, West assumes dividends follow a pure auto-regressive process, order q, of the form

$$d_{t+1} = \mu + \phi_1 d_t + \ldots + \phi_q d_{t-q+1} + v_{t+1}$$

Estimates for the dividend regression coefficients are readily obtained by OLS.

Finally, the usual asset price equation

$$p_t = \sum_{i=1}^{\infty} b^i E_{t+1}(d_{t+i})$$

is estimated by letting

$$p_{t+1} = m + \delta_1 d_{t+1} + \ldots + \delta_q d_{t-q+2} + w_{t+1}$$

Because each estimation process above represents part of the relationship between prices and dividends it is possible to derive constraint equations relating all the estimated model parameters. Under the assumption that no bubbles are present, the constraint equations should all be satisfied. A statistical test is constructed to measure how closely the constraints are satisfied.

Specifically, once estimates have been obtained for all coefficients, the constraint vector $R(\Theta)$ is constructed which relates the estimated coefficients $\hat{\Theta}$ to each other using analytical expressions. The vector $R(\Theta)$ has dimenstions *(q+1)x1*.

The null hypothesis is that $R(\hat{\Theta}) = 0$.

West's test statistic is

$$R(\hat{\Theta})^T \left[\left(\frac{\partial R}{\partial \Theta} \right) V \left(\frac{\partial R}{\partial \Theta} \right)^T \right] R(\hat{\Theta})$$

where:

- $\frac{\partial R}{\partial \Theta}$ is the *(q+1)x(2q+3)* jacobian matrix of first partial derivatives of $R(\Theta)$
- V is the *(2q+3)x(2q+3)* estimated variance-covariance matix of the coefficients Θ

The test statistic, under the null, is asymptotically distributed as a χ^2 distribution with *q+1* degrees of freedom.

Rejecting the null and validating the regression estimates provides evidence for speculative bubbles in the data.

It is important to note that West's test hinges on being able to validate the discount rate, dividend ARIMA, and price on distributed lag of dividends estimation processes. Provided these estimates are sound, model mispecification can be ruled out and any violation of the constraint equations can be directly attributed to the presence of bubbles.

Constructor

```
WTS = WestsTwoStep(Dates, Prices, Dividends);
```

Inputs:

- *Dates* is a time series vector of date data
- *Prices* is a time series vector of price data
- *Dividends* is a time series vector of dividend data

Outputs:

- *WTS* is the constructed WestsTwoStep class

PerformTest

```
Result = WTS.PerformTest(ARq, UseDifferencedData)
```

Performs West's test for speculative bubbles

Inputs:

- *ARq* is the auto-regressive order
- *UseDifferencedData* is boolean (true/false) and indicates if level or differenced data should be used when performing the test

Outputs:

- *Result* is a MATLAB structure array with the following fields:
- *Result.DataSize* is the total number of samples used
- *Result.DataSet* is a string representation of the date range used
- *Result.Differenced* is "true" or "false"

- *Result.q* is the auto-regressive order
- *Result.DoF* is the degrees of freedom for the χ^2 test
- *Result.TestStatistic* is the calculated test statistic value
- *Result.TestStatisticSig* is the significance of the calculated test statistic

---

Published with MATLAB® R2015b

ADF_FL

NOTE: This code has been sourced directly from Phillips, Shi, and Yu (see https://sites.google.com/site/shupingshi/home/research).

Contents

Overview

Computes a right-tailed augmented Dickey Fuller test with a fixed lag order for the regression model.

Usage

```
[estm] = ADF_FL(y,adflag,EqType)
```

where,

- *y* is the input time series data
- *adflag* is the lag order of the ADF test, *adflag* >= 0
- *EqType* is an instance of UnitRootNullTestType
- *estm* is the right-tailed test statistic that results from running the test on the data

---

Published with MATLAB® R2015b

CriticalValuesCache - class

Contents

Overview

Singleton class for managing a cache of Unit Root test critical values.

Unit root tests rely on simulating the null hypothesis model many times to determine the critical value sequences for a fixed set of model and test parameters. Since this process can be time consuming, it is preferable to cache the results of simulations for quick retrieval at a later time when conducting significance tests.

CriticalValuesCache is a single-instance class (singleton) which provides methods for interacting with the cached critical value sequences, as well as generating new entries for the cache when no existing match is found.

Typical usage is to get a handle to the cache and then query it for values, optionally asking the cache to generate the required values if they don't exist.

Constructor

Since *CriticalValuesCache* is a singleton, no public constructor is available. To use *CriticalValuesCache*, get a handle to the single instance by calling the *getInstance* method, as in the following example:

```
CVC = CriticalValuesCache.getInstance;
```

getCachePath

```
cachePath = CVC.getCachePath();
```

Retrieves the full system file path to the folder housing the cache files.

Outputs:

- *cachePath* is a MATLAB *fullfile* specification.

ClearCache

```
CVC.ClearCache();
```

Clears the contents of the critical values cache.

getCVs

```
[TestCVs, simResults, fileWasCreated] = CVC.getCVs(self, TType, T, M, EqType, Tw, d, eta, lag, rebuild, quants, useMex);
```

Gets the critical values sequence for the supplied parameters.

Unless rebuild is set to true, getCVs will attempt to locate a cached copy of the simulated results from a previous run. If no match is found in the cache, or if rebuild is set to true, getCVs will force-run the simulation using the supplied parameters and add the values to the cache upon completion.

The null model for data simulation is defined as:

$$y_t = d * T^{-\eta} + \theta * y_{t-1} + e_t$$

where e_t is i.i.d. (0, σ^2) and $\theta = 1$

Inputs:

- *TType* is an instance of UnitRootTestType

- *T* is the sample size
- *M* is the number of simulations to perform
- *EqType* is an instance of UnitRootNullTestType
- *Tw* is the minimum test window size
- *d* and *eta* are the null model parameters
- *lag* is the ADF lag order
- *rebuild* is a logical value which will cause the simulations to run again when set to true
- *quants* is a row vector of quantiles to use when calculating critical values
- *useMex* is a boolean value, set to true to use the compiled mex file when simulating

Outputs:

- *TestCVs* is a vector or critical values for the supplied quantiles *quants*
- *simResults* is the matrix of simulated critical values
- *fileWasCreated* returns true if the cache file was created or updated, false otherwise.

Published with MATLAB® R2015b

UnitRootBubbleTest - class

Contents

Overview

Performs various types of unit root bubble detection tests.

This is a helper class that encapsulates the process of testing for bubbles in a price-dividend ratio series.

Constructor

```
URBT = UnitRootBubbleTest(dates, priceOnDividend);
```

Inputs:

- *dates* is the date series (sorted, increasing)
- *priceOnDividend* is the price-dividend ratio series for the associated rows in *dates*

Outputs:

- *URBT* is the constructed UnitRootBubbleTest class

SetData

```
URBT.ResetData(dates, priceOnDividend);
```

Resets the data on the UnitRootBubbleTest class instance.

Inputs:

- *dates* is the date series (sorted, increasing)
- *priceOnDividend* is the price-dividend ratio series for the associated row in *dates*

PerformADFTest

```
URBTR = URBT.PerformADFTest(EqType, k, M, d, eta, quants, UseMex);
```

Runs an ADF test on the price-dividend ratio data.

Inputs:

- *EqType* is an instance of UnitRootNullTestType
- *k* is the ADF lag order
- *M* is the number of simulations to perform

- *d* and *eta* are the null model parameters
- *quants* is a row vector of quantiles to use when calculating critical values
- *useMex* is a boolean value, set to true to use the compiled mex file when simulating

Outputs:

- URBTR is an instance of UnitRootBubbleTestResult

PerformSADFTest

```
URBTR = URBT.PerformSADFTest(EqType, Tw, k, M, d, eta, quants, UseMex);
```

Runs a SADF test on the price-dividend ratio data.

Inputs:

- *EqType* is an instance of UnitRootNullTestType
- *Tw* is the minimum test window size
- *k* is the ADF lag order
- *M* is the number of simulations to perform
- *d* and *eta* are the null model parameters
- *quants* is a row vector of quantiles to use when calculating critical values
- *useMex* is a boolean value, set to true to use the compiled mex file when simulating

Outputs:

- URBTR is an instance of UnitRootBubbleTestResult

PerformGSADFTest

```
URBTR = URBT.PerformGSADFTest(EqType, Tw, k, M, d, eta, quants, UseMex);
```

Runs a GSADF test on the price-dividend ratio data.

Inputs:

- *EqType* is an instance of UnitRootNullTestType
- *Tw* is the minimum test window size
- *k* is the ADF lag order
- *M* is the number of simulations to perform
- *d* and *eta* are the null model parameters
- *quants* is a row vector of quantiles to use when calculating critical values
- *useMex* is a boolean value, set to true to use the compiled mex file when simulating

Outputs:

- URBTR is an instance of UnitRootBubbleTestResult

Published with MATLAB® R2015b

UnitRootBubbleTestResult class

Contents

Overview

Class representing the results from running a unit root test. Contains useful methods for generating plots and tables of results.

Constructor

Users should not explicity construct instances of *UnitRootBubbleTestResult*. Instead, they should call *PerformTest(...)* on an instance of *UnitRootBubbleTest* to return a constructed *UnitRootBubbleTestResult*, referred to below as URBTR.

GenerateResultsTable

```
[tab] = URBTR.GenerateResultsTable();
```

Generates table of test statistic and associated right-tailed critical values.

Note: Should only be called for SADF and GSADF test types (not useful for ADF test type).

Outputs:

- *tab* is a MATLAB table object.

GenerateBubbleStartEndTimes

```
[tab] = URBTR.GenerateBubbleStartEndTimes(Beta, MinDurationInYears, SamplesPerYear);
```

Generates a table of bubble start and end times using the test's date stamping algorithm.

Inputs:

- *Beta* is the critival value sequence level (typically one of 0.9, 0.95, 0.99)
- *MinDurationInYears* is only required for test type GSADF, and represents the minimum bubble size in years (can be fractional).
- *SamplesPerYear* is only required for test type GSADF, and represents the number of data samples in a full year.

Outputs:

- *tab* is a MATLAB table object.

GeneratePlot

```
[hAx,hLine1,hLine2] = URBTR.GeneratePlot(Beta);
```

Generates a plot of the unit root bubble test results. The plot contains lines for the real price series, the *Beta x 100* % critical value sequence, and the unit root test statistic sequence.

Inputs:

- *Beta* is the critival value sequence level (typically one of 0.9, 0.95, 0.99)

Outputs:

- *hAx* contains handles of the two axes created
- *hLine1* is the handle of the graphics object for line 1
- *hLine2* is the handle of the graphics object for line 2

Published with MATLAB® R2015b

UnitRootNullSimulator - class

Contents

Overview

Simulator for the various types of null tests for each unit root test type (ADF, SADF, GSADF). Simulations are typically used to derive asymptotic critical values for a given set of model parameters.

The null model for data simulation is defined as:

$$y_t = d * T^{-\eta} + \theta * y_{t-1} + e_t$$

where e_t is i.i.d. (0, σ^2) and $\theta = 1$

Constructor

```
URNS = UnitRootNullSimulator(TType, T, M, EqType, Tw, d, eta, lag, useMex);
```

Inputs:

- *TType* is an instance of UnitRootTestType
- *T* is the sample size
- *M* is the number of simulations to perform
- *EqType* is an instance of UnitRootNullTestType
- *Tw* is the minimum test window size
- *d* and *eta* are the null model parameters
- *lag* is the ADF lag order
- *useMex* is a boolean value, set to true to use the compiled mex file when simulating

Outputs:

- *URNS* is the constructed UnitRootNullSimulator class

PerformSimulation

```
simResults = URNS.PerformSimulation();
```

Performs the simulation steps for the specified unit root test.

WARNING: This function can take considerable time to execute if *T* and *M* are large (> 1000). Typical runtimes for large input parameters range from hours to days. *PerformSimulation* uses the default MATLAB parallel pool to distribute simulation calculations over multiple threads, thus reducing runtime considerably. *tic* and *toc* are also utilised, giving the user the total run time to perform the simulation.

To work out approximately how long this method will take to run, determine how many available CPU cores there are, say *C*, and call the method with *T* and setting *M* = *C*, which will give you the run time with full core utilisation, denoted t_C. The appoximate runtime in seconds when using *M* is then given by:

$$t_M = \frac{M \times t_C}{C}$$

Outputs:

- *simResults* is a matrix containing test results for each simulation. In the case of ADF tests *simResults* is *1 x M*. For SADF and GSADF tests, *simResults* is *(T-Tw+1) x M*.

Published with MATLAB® R2015b

UnitRootNullTestType - enumeration

Contents

Overview

Enumeration representing the different types of ADF null model tests.

Usage

Specify a value by using dot(.) syntax to select the required enumeration value. The following values are supported:

```
UnitRootNullTestType.UnitRoot
UnitRootNullTestType.UnitRootWithDrift
UnitRootNullTestType.UnitRootWithDriftAndTrend
```

Published with MATLAB® R2015b

UnitRootTests - class

Contents

Overview

Contains a static implementation for each different unit root test. The available test types are ADF, SADF, and GSADF.

Note: GSADF can be slow for large data sets and small minimum window size, therefore it is recommended to use the MEX file *UnitRootTestsCaller_mex* instead of this class when performing GSADF tests on large data sets.

Constructor

UnitRootTests only contains static methods, therefore no constructor is required.

UnitRootTests.ADF

```
Results = UnitRootTests.ADF(data, k, EqType);
```

Performs the basic ADF test on the supplied data.

Inputs:

- *data* is the input time series data
- *k* is the ADF lag order
- *EqType* is an instance of UnitRootNullTestType

Outputs:

- *Results* contains the right-tailed ADF test statistic value

UnitRootTests.SADF

```
Results = SADF(data, T, swindow0, k, EqType);
```

Performs the SADF test on the supplied data.

Inputs:

- *data* is the input time series data
- *T* is the sample size
- *swindow0* is the minimum test window size
- *k* is the ADF lag order
- *EqType* is an instance of UnitRootNullTestType

Outputs:

- *Results* contains the right-tailed ADF test statistic value

UnitRootTests.GSADF

```
Results = GSADF(data, T, swindow0, k, EqType);
```

Performs the GSADF test on the supplied data.

Inputs:

- *data* is the input time series data
- *T* is the sample size
- *swindow0* is the minimum test window size
- *k* is the ADF lag order
- *EqType* is an instance of UnitRootNullTestType

Outputs:

- *Results* contains the right-tailed ADF test statistic value

---

Published with MATLAB® R2015b

UnitRootTestsCaller - function

Contents

Overview

A wrapper function for calling the methods of the UnitRootTests class.

This function exists primarily for the purpose of the MATLAB Coder MEX generator, which requires the entry point to the code to be a function (not a class method).

Usage

```
[simResults] = UnitRootTestsCaller(TType, data, T, swindow0, k, EqType);
```

Inputs:

- *TType* is an instance of UnitRootTestType
- *data* is the time series data
- *T* is the sample size
- *swindow0* is the minimum test window size
- *k* is the ADF lag order
- *EqType* is an instance of UnitRootNullTestType

Outputs:

- *simResults* is the result of running the unit root test on the data

---

Published with MATLAB® R2015b

UnitRootTestType - enumeration

Contents

Overview

Enumeration representing the different types of unit root tests available in the Asset Bubble Toolbox.

Usage

Specify a value by using dot(.) syntax to select the required enumeration value. The following values are supported:

```
UnitRootTestType.ADF
UnitRootTestType.SADF
UnitRootTestType.GSADF
```

Published with MATLAB® R2015b

8 Appendix C: Matlab Code

```
function CreateDemoData()
%CREATEDEMODATA Creates demo data
%   Each bubble test has traditionally been applied to a specific dataset.
%   This function loads the relevant dataset and then saves the variables
%   into *.mat files for easy loading in demo files

% Get the current working folder
folder = fileparts(which(mfilename));

% Shiller's variance bounds test uses S&P 500 data from 1871 to 1979
[Dates, Prices, Dividends, PPI] = LoadShillerChap26Data(1979, true);
filename = fullfile(folder, 'VarianceBoundsDemoData.mat');
save(filename,'Dates','Prices','Dividends','PPI');

% West's test uses S&P 500 data from 1871 to 1980
[Dates, Prices, Dividends, PPI] = LoadShillerChap26Data(1980, true);
filename = fullfile(folder, 'WestsTwoStepDemoData.mat');
save(filename,'Dates','Prices','Dividends','PPI');

% The unit root tests of Phillips, Shi, and Yu use S&P 500 data from 1871 to 2010
[Dates,PriceOnDividend] = LoadUnitRootTestData();
filename = fullfile(folder, 'UnitRootDemoData.mat');
save(filename,'Dates','PriceOnDividend');

end
```

```
function [Dates, Prices, Dividends, PPI] = LoadShillerChap26Data(maxYear, scaleByPPI)
%LOADSHILLERCHAP26DATA Loads data from Robert Shiller's Chapter 26 appendix
%   Loads data from Robert Shiller's Chapter 26 appendix (obtained from http://www.econ.yale.edu/~shiller/data/chapt26.html)
%   The test data file is located in this script's directory in Shiller_Chap26_Data.xlsx
%   maxYear represents the max year to include data for (from 1871 to 1999)
%   scaleByPPI is a boolean.  If true, the prices and dividends are made real by scaling using PPI

folder = fileparts(which(mfilename));
filename = fullfile(folder, 'Shiller_Chap26_Data.xlsx');
sheet='Sheet1';
xlRange='A6:G134'; % Note: If an updated data file is obtain, this range will need to be adjusted
shillerData = xlsread(filename, sheet, xlRange);

% Filter by max year (find largest index)
filterSize = length(find(shillerData(:,1) < maxYear + 1));

Dates = shillerData(1:filterSize,1);
PPI = shillerData(1:filterSize,7);
Prices = shillerData(1:filterSize,2);
Dividends = shillerData(1:filterSize,3);

% Store the final PPI value, in case we are scaling by PPI
% Note that this data originally has PPI 1982 = 100 (which is slightly different than
% PPI 1979 = 100 as used by Shiller in his original paper).
% However, a new column has been added to the data to scale PPI to 1979 = 100
PPIFinalValue = PPI(filterSize);

% Adjust for inflation by scaling relative to PPI in the final year for consideration
% to convert all monetary values to real values relative to the final year
if (scaleByPPI)
    PPIScalingRatio = PPIFinalValue./PPI;
    Prices = Prices.*PPIScalingRatio;
    Dividends = Dividends.*PPIScalingRatio;
end

end
```

```
function [Dates, Prices, Dividends, CPI] = LoadShillerTestData(maxYear, scaleByCPI)
%LOADSHILLERDATA Loads data from Robert Shiller's excel data file
%   Loads data from Robert Shiller's excel data file (obtained from http://www.econ.↙
yale.edu/~shiller/data/ie_data.xls)
%   The test data file is located in this script's directory in shiller_ie_data.xls
%   maxYear represents the max year to include data for (from 1871 to 2016)
%   scaleByCPI is a boolean.  If true, the prices and dividends are made real by↙
scaling using CPI
%   The data is rolled up to yearly data before being returned.

folder = fileparts(which(mfilename));
filename = fullfile(folder, 'shiller_ie_data.xls');
sheet='Data';
xlRange='A9:F1748'; % Note: If an updated data file is obtain, this range will need to↙
be adjusted
shillerData = xlsread(filename, sheet, xlRange);

% Filter by max year (find largest index)
filterSize = length(find(shillerData(:,6) < maxYear + 1));

Dates = shillerData(1:filterSize,6);
CPI = shillerData(1:filterSize,5);
Prices = shillerData(1:filterSize,2);
Dividends = shillerData(1:filterSize,3);

% Now roll up the monthly data to yearly data.
startDate = floor(Dates(1));
endDate = ceil(Dates(end));
rolledUpSize = endDate - startDate;

DatesRolledUp = zeros(rolledUpSize,1);
PricesRolledUp = zeros(rolledUpSize,1);
DividendsRolledUp = zeros(rolledUpSize,1);
CPIRolledUp = zeros(rolledUpSize,1);

% Store the final CPI value, in case we are scaling by CPI
CPIFinalValue = CPI(filterSize);

% Adjust for inflation by scaling relative to CPI in the final year for consideration
% to convert all monetary values to real values relative to the final year
if (scaleByCPI)
    CPIScalingRatio = CPIFinalValue./CPI;
    Prices = Prices.*CPIScalingRatio;
    Dividends = Dividends.*CPIScalingRatio;
end

% We use a basic roll-up strategy, taking the Jan record for all but the dividend↙
stream
for i=1:rolledUpSize
    % Construct a filter for the current year's data
    filter = Dates > (startDate + i - 1) & Dates < (startDate + i);

    % Filter each set down to just the data for the year
    SubPrices = Prices(filter);
    SubDividends = Dividends(filter);
```

```
    SubCPI = CPI(filter);

    %Shiller uses the January prices so we do the same
    DatesRolledUp(i) = startDate + i - 1;
    PricesRolledUp(i) = SubPrices(1);
    CPIRolledUp(i) = mean(SubCPI);

    % We (arbitrarily) use an average of the dividends over the entire year
    DividendsRolledUp(i) = mean(SubDividends);

%     if (scaleByCPI)
%         % Deflate by CPI for the start of the year
%         PricesRolledUp(i) = PricesRolledUp(i)*CPIFinalValue/SubCPI(1);
%         % Deflate by the average CPI value of the year
%         DividendsRolledUp(i) = DividendsRolledUp(i)*CPIFinalValue/CPIRolledUp(i);
%     end
end

Dates = DatesRolledUp;
Prices = PricesRolledUp;
Dividends = DividendsRolledUp;
CPI = CPIRolledUp;

end
```

```
function [ Dates, PriceOnDividend ] = LoadUnitRootTestData()
%LOADUNITROOTTESTDATA Loads data from PSY's sample excel data file
%   Loads data from Phillips, Shi, and Yu's sample excel data file (obtained from
https://sites.google.com/site/shupingshi/PrgGSADF.zip?attredirects=0&d=1)
%   The test data file is located in this script's directory in SP_DV.xlsx

folder = fileparts(which(mfilename));
filename = fullfile(folder, 'SP_DV.xlsx');

SPDV = xlsread(filename);
Dates = SPDV(1:1680,2);
PriceOnDividend = SPDV(1:1680,6);

end
```

```
classdef VarianceBounds < handle
    %VARIANCEBOUNDS Shiller's Variance bounds test for financial time series
    %   Class for computing variance bounds test for two financial time series using the method of Shiller (1981b).
    %   Usage:
    %       ---Constructor---
    %       VB = VarianceBounds(dates, prices, dividends, priceIndex), where
    %         dates is an Nx1 vector of increasing dates for each point in the sample data
    %         prices is an Nx1 vector of prices for each point in the sample data
    %         dividends is an Nxq1 vector of dividends for each point in the sample data
    %         It is assumed that prices and dividends have been made real by scaling using a price index (e.g. CPI)
    %       ---Methods---
    %       VBR = PerformTest(shouldDetrend)
    %           Performs the variance bounds test and returns a VarianceBoundsResult object

    properties (SetAccess = private)
        Dates
        Prices
        Dividends
        PricesDetrended
        DividendsDetrended
    end

    methods
        % Constructor
        function VBT = VarianceBounds(dates, prices, dividends)
            VBT.Dates = dates;
            VBT.Prices = prices;
            VBT.Dividends = dividends;
            VBT.PricesDetrended = prices;
            VBT.DividendsDetrended = dividends;
        end

        function VBR = PerformTest(self, shouldDetrend, terminalValueScales)
            % Peforms the variance bounds test, optionally detrending data prior to running the test
            % terminalValueScales is a 1xM array of scalings around 1.0 for the terminal condition of the ex post rational price
            % Returns an instance of VarianceBoundsResult which can be used to generate plots and tables

            % Detrend data (if required)
            Beta = [];
            if shouldDetrend
                [Beta] = self.DetrendData();
            end

            % Compute the ex post rational price series
            [ExPostRationalPrice, r_bar] = ComputeExPostRationalPrice(self, terminalValueScales);
```

```
            % Construct the results object
            VBR = VarianceBoundsResult(self.Dates, self.Prices, self.Dividends,↙
shouldDetrend, ...
                self.PricesDetrended, self.DividendsDetrended, Beta,↙
ExPostRationalPrice, ...
                terminalValueScales, r_bar);
        end
    end

    methods (Access = private)
        function [ExPostRationalPrice, r_bar] = ComputeExPostRationalPrice(self,↙
terminalValueScales)
            % Computes the ex post ration price series from real prices and dividends

            % Compute the approximation to the discount rate
            r_bar = mean(self.DividendsDetrended)/mean(self.PricesDetrended);
            gamma_bar = 1/(1+r_bar);

            % Allocate memory and initialise the final values
            ExPostRationalPrice = zeros(length(self.PricesDetrended),length↙
(terminalValueScales));
            ExPostRationalPrice(length(ExPostRationalPrice),:) = terminalValueScales.↙
*mean(self.PricesDetrended);

            % Work backwards recursively to populate the series
            for t=length(ExPostRationalPrice)-1:-1:1
                ExPostRationalPrice(t,:) = gamma_bar*(ExPostRationalPrice(t+1,:) +↙
self.DividendsDetrended(t+1,1));
            end
        end

        function [Beta] = DetrendData(self)
            % AssumeS an exponential fit, then regresses natural log of prices on↙
dates

            [Beta,~,~,~,~] = regress(log(self.Prices), [ones(size(self.Dates)) self.↙
Dates]);

            lambda = exp(Beta(2,1));
            T = self.Dates(end,1);

            self.PricesDetrended = zeros(size(self.Prices));
            self.DividendsDetrended = zeros(size(self.Dividends));

            for t=1:length(self.Dates)
                self.PricesDetrended(t,1) = self.Prices(t,1)/(lambda^(self.Dates(t,1)↙
- T));
                self.DividendsDetrended(t,1) = self.Dividends(t,1)/(lambda^(self.Dates↙
(t,1) + 1 - T));
            end
        end
    end
end
```

```
classdef VarianceBoundsResult < handle
    %VARIANCEBOUNDSRESULT Result container class for variance bounds test results
    %   Class representing the result of running a variance bounds test.
    %   Can be used to generate plots and tables of results

    properties (SetAccess = private)
        Dates
        Prices
        Dividends
        ShouldDetrend
        PricesDetrended
        DividendsDetrended
        Beta
        ExPostRationalPrice
        TerminalValueScales
        r_bar
    end

    methods
        function VBTR = VarianceBoundsResult(Dates, Prices, Dividends, ShouldDetrend,↙
PricesDetrended, ...
                DividendsDetrended, Beta, ExPostRationalPrice, TerminalValueScales,↙
r_bar)
            VBTR.Dates = Dates;
            VBTR.Prices = Prices;
            VBTR.Dividends = Dividends;
            VBTR.ShouldDetrend = ShouldDetrend;
            VBTR.PricesDetrended = PricesDetrended;
            VBTR.DividendsDetrended = DividendsDetrended;
            VBTR.Beta = Beta;
            VBTR.ExPostRationalPrice = ExPostRationalPrice;
            VBTR.TerminalValueScales = TerminalValueScales;
            VBTR.r_bar = r_bar;
        end

        function [tab] = GenerateResultsTable(self)
            % Generates table of results according to Shiller's paper (p431)

            ResultsRows = 9;
            ResultsCols = length(self.TerminalValueScales);

            Results = zeros(ResultsRows,ResultsCols);
            RowNames = cell(ResultsRows,1);

            for j=1:ResultsCols
                Index = 1;
                RowNames{Index} = 'E(p)';
                Results(Index,j) = mean(self.PricesDetrended);

                Index = Index + 1;
                RowNames{Index} = 'E(d)';
                Results(Index,j) = mean(self.DividendsDetrended);

                Index = Index + 1;
                RowNames{Index} = 'r bar';
```

```
                Results(Index,j) = self.r_bar;

                Index = Index + 1;
                RowNames{Index} = 'r2 bar';
                Results(Index,j) = (1 + self.r_bar)^2 - 1;

                Index = Index + 1;
                RowNames{Index} = 'b';
                Results(Index,j) = self.Beta(2,1);

                Index = Index + 1;
                RowNames{Index} = 'cor(p,p*)';
                Results(Index,j) = corr(self.PricesDetrended, self.ExPostRationalPrice↙
(:,j));

                Index = Index + 1;
                RowNames{Index} = 'std(d)';
                Results(Index,j) = std(self.DividendsDetrended);

                Index = Index + 1;
                RowNames{Index} = 'std(p)';
                Results(Index,j) = std(self.PricesDetrended);

                Index = Index + 1;
                RowNames{Index} = 'std(p*)';
                Results(Index,j) = std(self.ExPostRationalPrice(:,j));
            end

            tab = table(Results,'RowNames',RowNames);
        end

        function PlotRealPriceAgainstExPostRationalPrice(self)
            fig1 = figure;
            set(fig1,'defaulttextinterpreter','latex');
            plot(self.Dates, self.PricesDetrended)
            hold on
            plot(self.Dates, self.ExPostRationalPrice, '--')
            xlabel('time')
            ylabelText = 'price';


            titleText = 'Comparison of price $p_t$ and ex post rational price↙
$p_t^*$';
            if self.ShouldDetrend
                titleText = [titleText ' - both detrended'];
                ylabelText = ['detrended ' ylabelText];
            end
            ylabel(ylabelText)
            title(titleText)

            legendStrings = cell(length(self.TerminalValueScales) + 1,1);
            legendStrings(1) = {'price'};
            for i=1:length(self.TerminalValueScales)
                legendStrings(i+1) = {strcat('$p_t^*$, terminal scale factor = ',↙
num2str(self.TerminalValueScales(i)))};
```

```
            end
            h = legend(legendStrings, 'Location','northwest');
            set(h,'Interpreter','latex')
            %legend('Location','northwest')
        end

        function PlotExPostRationalPrice(self)
            figure
            hold on
            legendStrings = cell(length(self.TerminalValueScales),1);
            for i=1:length(self.TerminalValueScales)
                plot(self.Dates, self.ExPostRationalPrice(:,i));
                legendStrings(i) = {strcat('Scale factor = ', num2str(self.↙
TerminalValueScales(i)))};
            end
            xlabel('time')
            ylabel('ex post rational price')
            title('ex post rational price - different terminal scalings')
            legend(legendStrings);

            dateMin = fix(min(self.Dates)) - 1;
            dateMax = fix(max(self.Dates)) + 1;

            axis([dateMin dateMax 0 300])
        end

        function PlotDetrendRegressionDetails(self)
            % Plots the regression details of the detrending process

            % Check to make sure the data was detrended before running this
            if ~self.ShouldDetrend
                error('No detrending was performed on the data. Unable to generate↙
plot.')
            end
            figure

            subplot(1,2,1)
            plot(self.Dates, log(self.Prices));
            hold on
            plot(self.Dates, [ones(size(self.Dates)) self.Dates]*self.Beta, 'r-')
            xlabel('time')
            ylabel('ln(price)')
            title('Linear regression of ln(price)')
            legend('ln(price)','regression')
            legend('Location','northwest')

            subplot(1,2,2)
            plot(self.Dates, self.Prices);
            hold on
            Y = self.Beta(1,1) + self.Beta(2,1)*self.Dates;
            plot(self.Dates, exp(Y), 'r-');
            xlabel('time')
            ylabel('price')
            title('Exponential fit of price data')
            legend('price','fit')
```

```
            legend('Location','northwest')
        end
    end
end
```

```
classdef AROrderPicker < handle
    %ARORDERPICKER Determines the best pure AR order for a given time series
    %   AROrderPicker uses Akaike and Bayesian information criteria
    %   to pick the best pure AR order (i.e. there is no MA component).

    properties (SetAccess = private)
        Data
    end

    methods
        function AROP = AROrderPicker(Data)
            AROP.Data = Data;
        end

        function SetData(self, Data)
            self.Data = Data;
        end

        function [pAIC, pBIC, aic, bic] = PickBestAROrder(self, MaxP, d)
            % Picks the best AR order p using an ARIMA(p,d,0) model and
            % both Akaike and Bayesian Information Criteria.
            % MaxP is the maximum number of AR lags to consider
            % d is the degree of differencing
            % Returns the best p for both AIC and BIC

            LOGL = zeros(MaxP,1);
            PQ = zeros(MaxP,1);
            for p = 1:MaxP
                mod = arima(p,d,0);
                [~,~,logL] = estimate(mod,self.Data,'print',false);
                LOGL(p,1) = logL;
                PQ(p,1) = p; % Normally p+q but q (MA order) is assumed zero
            end

            % Total number of params is p+q+1, but q=0
            [aic,bic] = aicbic(LOGL,PQ+1,length(self.Data));
            [~, pAIC] = min(aic);
            [~, pBIC] = min(bic);
        end
    end
end
```

```
classdef DiscountRateEstimator
    %DISCOUNTRATEESTIMATOR Estimates the discount rate b
    %   The arbitrage relationship is given by
    %       p_t = b*E(p_(t+1) + d_(t+1)|I_t)
    %   This can be rewritten in the form
    %       p_t = b*(p_(t+1) + d_(t+1)) + u_(t+1)
    %   where u_(t+1) is potentially correlated with p_(t+1) and d_(t+1)
    %   The esimtation uses Instrumental Variables, with d_t or diff(d_t) as Instruments

    properties (SetAccess = private)
        Data
    end

    methods
        function DRE = DiscountRateEstimator(Data)
            % Data is an instance of the structure array returned by WestsDataBuilder.GenerateData()
            DRE.Data = Data;
        end

        function EDR = EstimateDiscountRate(self, UseSimpleMethod)
            % Estimate the discount rate.
            % UseSimpleMethod is boolean.  When true, uses simple 2SLS IV estimation. When false uses GMM estimation.
            % EDR is a structure array with fields for the results, as well as the data used.

            if UseSimpleMethod
                EDR = self.SimpleIVEstimation();
            else
                EDR = self.GMMEstimation();
            end
        end
    end

    methods (Access = private)
        function EDR = SimpleIVEstimation(self)
            % Performs simple 2SLS IV estimation

            % Get the data
            X = self.Data.X;
            Y = self.Data.pt;
            Z = self.Data.D;

            % Perform 2SLS
            % Stage 1 - Regress preditors on instruments
            % Stage 2 - Regress response on stage 1 fitted values

            %Pz = Z*inv(Z'*Z)*Z';
            %b_hat = inv(X'*Pz*X)*X'*Pz*Y;
            Pz = Z*((Z'*Z)\Z');
            b_hat = (X'*Pz*X)\X'*Pz*Y;
            r_hat = (1-b_hat)/b_hat;
```

```
            % Build up the stucture array result
            EDR(1).b_hat = b_hat;
            EDR(1).r_hat = r_hat;
        end

        function EDR = GMMEstimation(self)
            % Performs GMM estimation on the data using Mike Cliff's GMM library
            % West's IV estimation is effectively GMM assuming
            % heteroskedasticity in the errors (i.e. it is NOT simple 2SLS)
            % The first step obtains an estimate of the discount rate using 2SLS
            % The second step uses this estimate to find the optimal
            % estimate by application of GMM.

            % Get the data
            X = self.Data.X;
            Y = self.Data.pt;
            Z = self.Data.D;

            % Step 1 - Initial estimate of b_hat using 2SLS
            EDR_2SLS = self.SimpleIVEstimation();
            b_hat = EDR_2SLS.b_hat;

            % Step 2 - Optimal estimate of b_hat using GMM
            gmmopt.infoz.momt='lingmmm';
            gmmopt.plot = 0;
            gmmopt.prt = 0;
            [out, opts] = gmm(b_hat,gmmopt,Y,X,Z);

            b_hat = out.b;
            r_hat = (1-b_hat)/b_hat;

            % Build up the stucture array result
            EDR(1).b_hat = b_hat;
            EDR(1).r_hat = r_hat;
            EDR(1).gmmOut = out;
            EDR(1).gmmOpts = opts;
        end
    end
end
```

```
classdef WestsCovarianceMatrix < handle
    %WESTSCOVARIANCEMATRIX Summary of this class goes here
    %   Detailed explanation goes here

    properties (SetAccess = private)
        D
        Dplus1
        X
        Sd_hat
        pt
        ptplus1
        dtplus1
        b_hat
        phi_hat
        delta_hat
        IsDifferenced
        T
        q
        A_hat
        m
        V
    end

    methods
        function WCM = WestsCovarianceMatrix(Data,Sd_hat,b_hat,...
                phi_hat,delta_hat,IsDifferenced)
            % Computes West's variance-covariance matrix V (West 1987, p563)
            % Data is a structure array holding the different data matrices
            % Sd_hat is an estimate of the spectral density matrix from estimating the↙
arbitrage equation (q+1)x(q+1)
            % b_hat is an estimate of the discount rate 1x1
            % phi_hat is the coefficient estimates from the dividend ARIMA process↙
(q+1)x1
            % delta_hat is the coefficent estimates from the regression of prices on↙
lagged dividends (q+1)x1
            % IsDifferenced is true if using differenced data, false otherwise

            WCM.D = Data.D;
            WCM.Dplus1 = Data.Dplus1;
            WCM.X = Data.X;
            WCM.Sd_hat = Sd_hat;
            WCM.pt = Data.pt;
            WCM.ptplus1 = Data.ptplus1;
            WCM.dtplus1 = Data.dtplus1;
            WCM.b_hat = b_hat;
            WCM.phi_hat = phi_hat;
            WCM.delta_hat = delta_hat;
            WCM.IsDifferenced = IsDifferenced;

            WCM.ComputeCovarianceMatrix();
        end
    end

    methods (Access = private)
        function ComputeCovarianceMatrix(self)
```

```
            % Performs the computational steps to approximate the covariance matrix V

            % Determine the integer constants from the supplied data
            self.T = length(self.pt);
            self.q = length(self.phi_hat) - 1;

            % Compute Hansen and Singleton's optimal weighting matrix
            %self.A_hat = self.X'*self.D*inv(self.T*self.Sd_hat);
            self.A_hat = self.X'*self.D/(self.T*self.Sd_hat);

            % Compute S_hat
            S_hat = self.S_hat();

            % Compute the normalising matrix
            % FT is diag(T^1/2,...,T^1/2)
            FT = diag(sqrt(self.T)*ones(2*self.q+3,1));
            if self.IsDifferenced
                % FT is diag(T,T^1/2,...,T^1/2)
                FT(1,1) = self.T;
            end

            % Compute normalising matrix CT
            % CT is diag(T^1/2,...,T^1/2)
            CT = diag(sqrt(self.T)*ones(2*self.q+3,1));
            if self.IsDifferenced
                % CT is diag(T^3/2,T^1/2,...,T^1/2)
                CT(1,1) = self.T^(3/2);
            end

            % Compute sum of h_t_theta terms
            h_t_theta = self.h_t_theta(1);
            for t = 2:self.T
                h_t_theta = h_t_theta + self.h_t_theta(t);
            end

            h_t_theta_trans = h_t_theta';

            % Finally, compute the covariance matrix approximation
%             FTinv = inv(FT);
%             Term1 = inv(FTinv*h_t_theta*FTinv);
%             Term2 = inv(FTinv*h_t_theta_trans*FTinv);
%             self.V = Term1*S_hat*Term2;

            Term1 = FT\h_t_theta/FT;
            Term2 = FT\h_t_theta_trans/FT;
            self.V = Term1\S_hat/Term2;

            % Scale by CT
            %self.V = inv(CT)*self.V;
        end

        function [S_hat] = S_hat(self)
            % Computes S_hat

            % Approximate variable m, used in the weighting process
```

```
            % m ~ sqrt(T)
            self.m = ceil(sqrt(self.T));

            S_hat = self.Omega_i(0);

            for i=1:self.m
                Omega_i = self.Omega_i(i);
                S_hat = S_hat + self.K(i)*(Omega_i + Omega_i');
            end
        end

        function [Omega_i] = Omega_i(self, i)
            % Computes Omega_i, used in equation for S_hat

            % Initialise storage for Omega_i
            Omega_i = zeros(2*self.q+3);

            for t=i+1:self.T
                h_t = self.h_t(t);
                h_t_minus_i = self.h_t(t-i);
                Omega_i = Omega_i + h_t*h_t_minus_i';
            end

            Omega_i = Omega_i/self.T;
        end

        function [K] = K(self, i)
            % Computes weight elements used in S_hat calculation
            K = 1 - i/(self.m + 1);
        end

        function [h_t] = h_t(self, t)
            % Computes the orthogonality vector h_t

            h_t = zeros(2*self.q+3,1);

            % Arbitrage equation
            h_t(1,1) = self.A_hat*self.D(t,:)'*(self.pt(t) - self.X(t)*self.b_hat);

            % ARIMA process for dividends
            h_t(2:self.q+2,1) = self.D(t,:)'*(self.dtplus1(t)-self.D(t,:)*self.↙
phi_hat);

            % OLS process for prices on distributed lag of dividends
            if ~self.IsDifferenced
                h_t(self.q+3:end,1) = self.Dplus1(t,:)'*(self.ptplus1(t)-self.Dplus1↙
(t,:)*self.delta_hat);
            else
                h_t(self.q+3:end,1) = self.Dplus1(t,:)'*(self.ptplus1(t)-self.D(t,:)↙
*self.delta_hat);
            end
        end

        function [h_t_theta] = h_t_theta(self, t)
            % Computes the matrix of partial derivatives (w.r.t. theta) of
```

```
            % the orthogonality vector h_t

            % Initialise to zero (as is the case for many of the partial derivatives)
            h_t_theta = zeros(2*self.q+3);

            % Arbitrage equation (partial derivatives all zero except w.r.t. b_hat )
            h_t_theta(1,1) = self.A_hat*self.D(t,:)'*(-self.X(t));

            % ARIMA process for dividends
            dhdphi = -self.D(t,:);
            h_t_theta(2:self.q+2,2:self.q+2) = self.D(t,:)'*dhdphi;

            % OLS process for prices on distributed lag of dividends
            if ~self.IsDifferenced
                dhddelta = -self.Dplus1(t,:);
            else
                dhddelta = -self.D(t,:);
            end
            h_t_theta(self.q+3:end,self.q+3:end) = self.Dplus1(t,:)'*dhddelta;
        end
    end

end
```

```
classdef WestsDataBuilder < handle
    %WESTSDATABUILDER Class for building the different data matrices used in West's speculative test for bubbles

    properties (SetAccess = private)
        Prices
        Dividends
        q
        UseDifferencedData
    end

    methods
        function WDB = WestsDataBuilder(Prices, Dividends, ARq, UseDifferencedData)
            WDB.Prices = Prices;
            WDB.Dividends = Dividends;
            WDB.q = ARq;
            WDB.UseDifferencedData = UseDifferencedData;
        end

        function Data = GenerateData(self)
            % Construct the data used in West's test process
            % Results are returned in a structure array with the following
            % fields
            %   pt is the price series
            %   dt is the dividend series
            %   ptplus1 is the price series at time t+1 (or the diff of it)
            %   dtplus1 is the dividend series at time t+1 (or the diff of it)
            %   X is (price + dividend) at time t+1
            %   D is the instruments (lagged dividends) at time t (or the diff of it)
            %   Dplus1 is the instruments (lagged dividends) at time t+1 (or the diff of it)

            Lags = self.q - 1;

            % All vectors are initially from t=1:T
            if ~self.UseDifferencedData
                % Not using differencing
                pt = self.Prices(1:end-1); % t=1:T-1
                dt = self.Dividends(1:end-1); % t=1:T-1
                ptplus1 = self.Prices(2:end); % t=2:T
                dtplus1 = self.Dividends(2:end); % t=2:T
                X = self.Prices(2:end) + self.Dividends(2:end); % t=2:T

                % Get the instruments D and Dplus1, from level dividends
                D = [ones(length(dt),1) lagmatrix(dt,0:Lags)]; % t=1:T-1
                Dplus1 = [ones(length(dtplus1),1) lagmatrix(dtplus1,0:Lags)]; % t=2:T

                % Reduce data lengths considering lags, since D and Dplus1
                % will contain NaN for Lags>0
                pt = pt(Lags+1:end,1); % t=1+Lags:T-1
                dt = dt(Lags+1:end,1); % t=1+Lags:T-1
                ptplus1 = ptplus1(Lags+1:end,1); % t=2+Lags:T
                dtplus1 = dtplus1(Lags+1:end,1); % t=2+Lags:T
                X = X(Lags+1:end,1); % t=2+Lags:T
                D = D(Lags+1:end,:); % t=1+Lags:T-1
```

```
                Dplus1 = Dplus1(Lags+1:end,:); % t=2+Lags:T
            else
                % Using differencing
                pt = self.Prices(1:end-1); % t=1:T-1
                dt = self.Dividends(1:end-1); % t=1:T-1
                ptplus1 = diff(self.Prices); % t=2:T
                dtplus1 = diff(self.Dividends); % t=2:T
                X = self.Prices(2:end) + self.Dividends(2:end); % t=2:T

                % Get the instruments D and Dplus1, from differenced dividends
                D = [ones(length(dtplus1)-1,1) lagmatrix(dtplus1(1:end-1,1),0:Lags)];↙
% t=2:T-1
                Dplus1 = [ones(length(dtplus1)-1,1) lagmatrix(dtplus1(2:end,1),0:↙
Lags)]; % t=3:T

                % Reduce data lengths considering lags and differencing,
                % since D and Dplus1 will contain NaN for Lags>0
                pt = pt(Lags+2:end,1); % t=2+Lags:T-1
                dt = dt(Lags+2:end,1); % t=2+Lags:T-1
                ptplus1 = ptplus1(Lags+2:end,1); % t=3+Lags:T
                dtplus1 = dtplus1(Lags+2:end,1); % t=3+Lags:T
                X = X(Lags+2:end,1); % t=3+Lags:T
                D = D(Lags+1:end,:); % t=2+Lags:T-1
                Dplus1 = Dplus1(Lags+1:end,:); % t=3+Lags:T
            end

            % Package up into structure array
            Data(1).pt = pt;
            Data(1).dt = dt;
            Data(1).ptplus1 = ptplus1;
            Data(1).dtplus1 = dtplus1;
            Data(1).X = X;
            Data(1).D = D;
            Data(1).Dplus1 = Dplus1;
        end
    end

end
```

```
classdef WestsSymbolicComps
    %WESTSSYMBOLICCOMPS Class to compute symbolic math structures for West's coefficients
    %   Assuming the order of the AR process is q, the coefficients vector Theta
    %   with dimenstions (2q+3)x1 is represented as
    %
    %   Theta = (b,mu,phi1,...,phiq,m,delta1,...,deltaq)'
    %       where
    %           b is the estimated discount rate
    %           mu is the d_t regression intercept
    %           phi1,...,phiq are the d_t regression coefficients
    %           m is the p_t regression intercept
    %           delta1,...,deltaq are the p_t regression coefficients
    %
    %   Two constraint equations are constructed:
    %       R1 - For level data: d_t ~ AR(q)
    %       R2 - For differenced data: d_t - d_(t-1) ~ AR(q)
    %
    %   The jacobian matrix is also derived for each constraint vector
    %       dR1 = dR1/dTheta
    %       dR2 = dR2/dTheta
    %
    %   All structures are returned as symbolic math objects and can be evaluated with
    %   real values for the coefficients via subs.
    %   For example, suppose the have the real calculated coefficients in a (2q+3)x1 vector Coeffs.
    %   We can evaluate the dR1 jacobian using
    %       subs(dR1,Theta, Coeffs)

    properties (SetAccess = private)
        q
        Theta
        R1
        R2
        dR1
        dR2
    end

    methods
        function WSC = WestsSymbolicComps(ARq)
        % Constructs a new WestsSymbolicComps object
        % ARq is the auto-regressive order
            WSC.q = ARq;

            % Create symbolic variables for each element of the coefficient vector
            % Note: mu is handled differently since mu() is also a MATLAB function
            syms b m;
            C = ones(ARq,1);
            phi = sym('phi%d', size(C));
            delta = sym('delta%d', size(C));

            % Construct the coefficient vector
            WSC.Theta = [b; sym('mu'); phi; m; delta];

            % Construct the constraint vector R1 for level data
```

```
            % The formulae are taken from West's paper, equation 13a
            PHI = 1 - b*phi(1);
            for i=2:ARq
                PHI = PHI - (b^i)*phi(i);
            end

            PHIINV = PHI^-1;

            WSC.R1 = m - b*((1-b)^-1)*PHIINV*sym('mu');
            WSC.R1(2,1) = delta(1)-(PHIINV-1);

            for j=2:ARq
                WSC.R1(j+1,1) = delta(j);
                for k=j:ARq
                    WSC.R1(j+1,1) = WSC.R1(j+1,1) - PHIINV*((b^(k-j+1))*phi(k));
                end
            end

            % Construct the jacobian of the constraint vector R1
            WSC.dR1 = jacobian(WSC.R1, WSC.Theta);

            % Construct the constraint vector R2 for difference data
            % The formulae are taken from West's paper, equation 13b
            WSC.R2 = m - (b*((1-b)^-1)*PHIINV + PHIINV - 1)*sym('mu');
            for j=1:ARq-1
                WSC.R2(j+1,1) = delta(j) - (PHIINV-1)*phi(j);
                for k=j+1:ARq
                    WSC.R2(j+1,1) = WSC.R2(j+1,1) - PHIINV*((b^(k-j))*phi(k));
                end
            end
            WSC.R2(end+1,1) = delta(end) - (PHIINV - 1)*phi(end);

            % Construct the jacobian of the constraint vector R2
            WSC.dR2 = jacobian(WSC.R2, WSC.Theta);
        end
    end
end
```

```
classdef WestsTwoStep < handle
    %WESTSTWOSTEP Class to perform West's two-step test for speculative bubbles
    %   West's two-step test compares regression coefficients from different
    %   regression mechanisms to determine if price and dividend series contain speculative bubbles.
    %
    %   The test involves three estimation procedures:
    %   1) Estimating the discount rate b = 1/(1+r) from the Euler equation using Instrumental Variables
    %   2) Estimating the AR(q) regression coefficients for the dividend generating process using OLS
    %   3) Estimating the regression coefficients for the price vs dividend generating process using OLS
    %
    %   The order of the AR process is q, and is supplied by the user (West uses q=2 and q=4)
    %   The unknown coefficients vector Theta has dimensions (2q+3)x1
    %
    %   Once estimates have been obtained for all unknown coefficients, a constraint vector R(Theta) is
    %   constructed which relates the coefficients to each other using analytical expressions.
    %   The vector R(Theta) has dimenstions (q+1)x1
    %
    %   The null hypothesis is that R(Theta) = 0.
    %   West's test statistic is
    %     R(Theta)'[(dR/dTheta) V (dR/dTheta)']^-1 R(Theta)
    %   where:
    %     dR/dTheta is the (q+1)x(2q+3) jacobian matrix of first partial derivatives of R(Theta)
    %     V is the (2q+3)x(2q+3) estimated variance-covariance matix of the coefficients Theta
    %
    %   The test statistic, under the null, is asymptotically distributed as
    %   a chi-squared distribution with q+1 degrees of freedom.
    %
    %   Rejecting the null and validating the estimates of 1), 2) and 3)
    %   provides evidence for speculative bubbles in the data.
    properties (SetAccess = private)
        Dates %Input: Date series data
        Prices %Input: Price series data
        Dividends %Input: Dividend series data
        q %Input: auto-regressive order
        IsDifferenced %Input: Wether or not to first-difference supplied data
        Data %Output: Structure array holding all the data vectors and matrices used in the calculations
        EDR %Output: Structure array holding output from the discount rate estimation process
        b %Output: The discount rate
        DividendCoeffs %Output: Estimated coefficients from dividend regression process
        PriceCoeffs %Output: Estimated coefficients from price regression process
        CovarianceMatrix %Output: Estimated asymptotic variance-covariance matrix of coefficients
        Coeffs %Output: The vector of estimated coefficients
```

```
        WSC %Output: Object holding symbolic representations of the constraint vector↙
and jacobian
        R %Output: Vector of evaluated constraint equations
        dR %Output: Matrix of evaluated Jacobian elements
        TestStatistic %Output: West's test statistic
        TestStatisticSig %Output: The statistical significance of the test statistic
    end

    methods
        function WTS = WestsTwoStep(Dates, Prices, Dividends)
            % Constructs a new WestsTwoStep object
            % Prices is a level price series
            % Dividends is the corresponding level dividend series for the price↙
series

            WTS.Dates = Dates;
            WTS.Prices = Prices;
            WTS.Dividends = Dividends;
        end

        function Result = PerformTest(self, ARq, UseDifferencedData)
            % Performs West's test for speculative bubbles
            % ARq is the auto-regressive order
            % UseDifferencedData should be true if the test should be run on↙
differenced data, false otherwise

            self.q = ARq;
            self.IsDifferenced = UseDifferencedData;
            WDB = WestsDataBuilder(self.Prices, self.Dividends, self.q, self.↙
IsDifferenced);
            self.Data = WDB.GenerateData();
            self.ComputeDiscoutRate();
            self.RegressDividends();
            self.RegressPrices();
            self.ComputeCovarianceMatrix();
            self.ComputeConstraintsAndJacobian();
            self.ComputeTestStatistic();

            % Pack results into structure array
            DataSize = length(self.Data.pt);
            Result(1).DataSize = DataSize;
            Result(1).DataSet = strcat(num2str(self.Dates(end-DataSize+1)),'-',num2str↙
(self.Dates(end)));
            if self.IsDifferenced
                Result(1).Differenced = 'yes';
            else
                Result(1).Differenced = 'no';
            end
            Result(1).q = self.q;
            Result(1).DoF = self.q+1;
            Result(1).TestStatistic = self.TestStatistic;
            Result(1).TestStatisticSig = self.TestStatisticSig;
        end
    end
```

```
    methods (Access = private)
        function ComputeDiscoutRate(self)
            % IV estimation of Euler equation

            % Compute the discount rate using GMM
            DRE = DiscountRateEstimator(self.Data);
            self.EDR = DRE.EstimateDiscountRate(false);
            self.b = self.EDR.b_hat;
        end

        function RegressDividends(self)
            % ARIMA estimation of dividend process

            LM = fitlm(self.Data.D(:,2:end),self.Data.dtplus1,'linear');
            % TODO: Store LM away for later use
            self.DividendCoeffs = LM.Coefficients.Estimate;
        end

        function RegressPrices(self)
            % OLS estimation of price process on lagged dividends

            % Regress price on distributed lag of dividends
            if (~self.IsDifferenced)
                LM = fitlm(self.Data.Dplus1(:,2:end),self.Data.ptplus1,'linear');
            else
                LM = fitlm(self.Data.D(:,2:end),self.Data.ptplus1,'linear');
            end

            % TODO: Store LM away for later use
            self.PriceCoeffs = LM.Coefficients.Estimate;
        end

        function ComputeCovarianceMatrix(self)
            % Compute variance-covariance matrix V

            WCM = WestsCovarianceMatrix(self.Data,self.EDR.gmmOut.S,...
                self.b,self.DividendCoeffs,self.PriceCoeffs,self.IsDifferenced);
            self.CovarianceMatrix = WCM.V;

%             WCMV2 = WestsCovarianceMatrixV2(self.Data,self.EDR.gmmOut.S,...
%                 self.b,self.DividendCoeffs,self.PriceCoeffs,self.IsDifferenced);
%             self.CovarianceMatrix = WCMV2.V;

%             PCM = PetrasCovarianceMatrix(self.Data,self.b,...
%                 self.DividendCoeffs,self.PriceCoeffs,self.IsDifferenced);
%             self.CovarianceMatrix = PCM.V;
        end

        function ComputeConstraintsAndJacobian(self)
            % Get the symbolic representation for the coefficients, constraints, and↙
associated Jacobian
            self.WSC = WestsSymbolicComps(self.q);

            % Construct the coefficients vector of estimated coefficients
            self.Coeffs = [self.b; self.DividendCoeffs; self.PriceCoeffs];
```

```
            % Evaluate the symbolic constraint vector and Jacobian matrix with the values in Coeffs
            if (~self.IsDifferenced)
                self.R = eval(subs(self.WSC.R1, self.WSC.Theta, self.Coeffs));
                self.dR = eval(subs(self.WSC.dR1, self.WSC.Theta, self.Coeffs));
            else
                self.R = eval(subs(self.WSC.R2, self.WSC.Theta, self.Coeffs));
                self.dR = eval(subs(self.WSC.dR2, self.WSC.Theta, self.Coeffs));
            end
        end

        function ComputeTestStatistic(self)
            % Compute West's test statistic
            %self.TestStatistic = self.R'*inv(self.dR*self.CovarianceMatrix*self.dR')*self.R;
            self.TestStatistic = self.R'*((self.dR*self.CovarianceMatrix*self.dR')\self.R);
            self.TestStatisticSig = 1 - chi2cdf(self.TestStatistic, self.q + 1);
        end
    end
end
```

```
classdef WestsTwoStepResult < handle
    %WESTSTWOSTEPRESULT Summary of this class goes here
    %   Detailed explanation goes here

    properties
    end

    methods
        function WTSR = WestsTwoStepResult()
        end
    end

end
```

```
% ------------------------------------------------------------------------
%  "Testing for Multiple Bubbles" by Phillips, Shi and Yu (2011)
%   Note: This function has been modified from the original.
%   Specifically, the input mflag has been changed to an enumerated
%   value EqType to reduce the complexity of the function.

%    In this program, we calculate the ADF statistic with a fixed
%    lag order.
%       y is the time series being tested, with N observations
%       adflag is the lag order of the regression model (k >= 0)
%       EqType is the value of UnitRootNullTestType defining the null hypothesis
%------------------------------------------------------------------------

function [estm]=ADF_FL(y,adflag,EqType)
    t1=size(y,1)-1; % N-1
    const=ones(t1,1); % (N-1)x1
    trend=1:1:t1; % 1x(N-1)
    trend=trend'; % (N-1)x1

    % Initialise the data vectors (level and first difference)
    y1  = y(size(y,1)-t1:size(y,1)-1); % y(1:N-1) is (N-1)x1
    dy  = y(2:size(y,1)) - y(1:size(y,1)-1); %y(2:end) - y(1:end-1) is (N-1)x1
    dy0 = dy(size(dy,1)-t1+1:size(dy,1)); % dy(1:N-1) is (N-1)x1, just dy
    x=y1; % (N-1)x1

    % Set up the regression model based on EqType
    switch(EqType)
        case UnitRootNullTestType.UnitRootWithDrift;
            x=[x const]; % Include constant
        case UnitRootNullTestType.UnitRootWithDriftAndTrend
            x=[x const trend]; % Include constant and trend
    end

    x1=x; % One of (N-1)x2, (N-1)x3, (N-1)x1 depending on EqType

    % If we are including lag terms in the regression, reduce the
    % working set size by the number of lags
    t2=t1-adflag; % N-1-k
    x2=x1(size(x1,1)-t2+1:size(x1,1),:);         % from k+1 to (N-1) (including y1 and↙
x) is (N-1-k)x(size(x1,2)
    dy01=dy0(size(dy0,1)-t2+1:size(dy0,1));      % from k+1 to (N-1) (including dy0)↙
is (N-1-k)x1

    % If we are including lag terms, append to the regressor data
    % with as many lag terms as we are including
    % i.e. include k lag variables of dy in x2
    if adflag>0;
        szx2 = size(x2);
        x2 = [x2 zeros(szx2(1),adflag)];
        j=1;
        while j<=adflag;
            % Insert the k-th lag of dy into x2
            x2(:,szx2(2) + j) = dy(size(dy,1)-t2+1-j:size(dy,1)-j); % dy(k+1-j:N-1-j)↙
is (N-1-k)x1
            j=j+1;
```

```
        end;
    end;

    % Perform the regression process (i.e. regress dy01 on x2)
    % beta is (M+k)x1 where M = EqType.AsInt (1,2,or 3)
    beta =(x2'*x2)^(-1)*(x2'*dy01);
    % Compute the residuals of the regression
    eps  = dy01 - x2*beta; % is (N-1-k)x1

    % Compute the significance
    dof = t2-adflag-double(EqType);
    sig = sqrt(diag(eps'*eps/dof*(x2'*x2)^(-1)));

    % Compute the ADF test statistic
    tvalue=beta./sig;

    % Extract the test statistic corresponding to the Beta*y_(t-1) term
    % i.e. the test statistic for the coeff calculated from the first column of x2
    % which is the ADF test statistic of interest
    estm=tvalue(1);
end
```

```
classdef UnitRootBubbleTest < handle
    %UNITROOTBUBBLETEST Performs various types of unit root bubble detection tests
    %   This is a helper class that encapsulates the process of testing for bubbles
    %   in a price-dividend ratio series.
    %   The user supplies a Tx2 matrix where the first column corresponds to a date
    %   series (sorted, increasing) and the second column is the price-dividend ratio
    %   for the associated date for the same row.

    properties (SetAccess = private)
        dates
        priceOnDividend
        T
        CVC
    end

    methods
        function URBT = UnitRootBubbleTest(dates, priceOnDividend)
            % Constructor for UnitRootBubbleTest
            URBT.CVC = CriticalValuesCache.getInstance;
            URBT.ResetData(dates, priceOnDividend);
        end

        function ResetData(self, dates, priceOnDividend)
            % Call this function to reset the data
            self.Initialise(dates, priceOnDividend);
        end

        function URBTR = PerformADFTest(self, EqType, k, M, d, eta, quants, UseMex)
            % Run an ADF test on the price-dividend data
            % EqType is an instance of the enumeration UnitRootNullTestType
            % k is the ADF lag order
            % M is the number of simulations to run when estimating critical values ↙
(set to 0 to disable simulations)
            % d and eta are the null model parameters
            % quants is a row vector of quantiles to use when calculating critical ↙
values
            % UseMex is a boolean value, set to true to use the compiled mex file when ↙
running tests

            % Perform the ADF test on the data
            Tw = self.T; % Test window is complete data set
            URBTR = self.PerformTest(UnitRootTestType.ADF, EqType, Tw, k, M, d, eta, ↙
quants, UseMex);
        end

        function URBTR = PerformSADFTest(self, EqType, Tw, k, M, d, eta, quants, ↙
UseMex)
            % Run a SADF test on the price-dividend data
            % EqType is an instance of the enumeration UnitRootNullTestType
            % Tw is the minimum test window size
            % k is the ADF lag order
            % M is the number of simulations to run when estimating critical values ↙
(set to 0 to disable simulations)
            % d and eta are the null model parameters
            % quants is a row vector of quantiles to use when calculating critical ↙
```

```
values
            % UseMex is a boolean value, set to true to use the compiled mex file when running tests

            % Perform the SADF test on the data
            URBTR = self.PerformTest(UnitRootTestType.SADF, EqType, Tw, k, M, d, eta, quants, UseMex);
        end

        function URBTR = PerformGSADFTest(self, EqType, Tw, k, M, d, eta, quants, UseMex)
            % Run a GSADF test on the price-dividend data
            % EqType is an instance of the enumeration UnitRootNullTestType
            % Tw is the minimum test window size
            % k is the ADF lag order
            % M is the number of simulations to run when estimating critical values (set to 0 to disable simulations)
            % d and eta are the null model parameters
            % quants is a row vector of quantiles to use when calculating critical values
            % UseMex is a boolean value, set to true to use the compiled mex file when running tests

            % Perform the GSADF test on the data
            URBTR = self.PerformTest(UnitRootTestType.GSADF, EqType, Tw, k, M, d, eta, quants, UseMex);
        end
    end

    methods (Access = private)
        function Initialise(self, dates, priceOnDividend)
            % Initialise the class properties
            self.dates = dates;
            self.priceOnDividend = priceOnDividend;
            self.T = length(dates);
        end

        function URBTR = PerformTest(self, TType, EqType, Tw, k, M, d, eta, quants, UseMex)
            if UseMex
                Results = UnitRootTestsCaller_mex(TType, self.priceOnDividend, self.T, Tw, k, EqType);
            else
                Results = UnitRootTestsCaller(TType, self.priceOnDividend, self.T, Tw, k, EqType);
            end

            URBTR = UnitRootBubbleTestResult(self.dates, self.priceOnDividend, self.T, TType, EqType, Tw, k, M, d, eta, quants, Results);

            % Perform simulations, if the user has requested it
            if M > 0
                [TestCVs, simResults, ~] = self.CVC.getCVs(TType, self.T, M, EqType, Tw, d, eta, k, false, quants, UseMex);
                URBTR.SetCVSimulationResults(TestCVs, simResults);
```

```
            end
        end
    end

end
```

```
classdef UnitRootBubbleTestResult < handle
    %UNITROOTBUBBLETESTRESULT Container class for results from running a unit root bubble test
    %   Encapsulates the result data from running a unit root test of a given type (ADF, SADF, GSADF)
    %   Used by UnitRootBubbleTest class when returning the result of a bubble test.
    %   Can be used to plot the results, determine if the null hypothesis is accepted/rejected,
    %   and generate date stamps for bubble start-end dates if the test type supports it.

    properties (SetAccess = private)
        dates
        priceOnDividend
        T
        TestType
        EqType
        Tw
        k
        M
        d
        eta
        quants
        TestResults
        TestCVs
        simResults
    end

    methods
        function URBTR = UnitRootBubbleTestResult(dates, priceOnDividend, T, TType, EqType, Tw, k, M, d, eta, quants, Results)
            URBTR.dates = dates;
            URBTR.priceOnDividend = priceOnDividend;
            URBTR.T = T;
            URBTR.TestType = TType;
            URBTR.EqType = EqType;
            URBTR.Tw = Tw;
            URBTR.k = k;
            URBTR.M = M;
            URBTR.d = d;
            URBTR.eta = eta;
            URBTR.quants = quants;
            URBTR.TestResults = Results;
        end

        function SetCVSimulationResults(self, TestCVs, simResults)
            self.TestCVs = TestCVs;
            self.simResults = simResults;
        end

        function [tab] = GenerateResultsTable(self)
            % Generates table of test statistic and associated right-tailed critical values

            if isempty(self.simResults)
```

```
                % Critical values have not bee supplied, so we can't do anything about this.
                error('Critical values have not been supplied.  Unable to generate table.')
            end

            TType = self.TestType.char;
            TestStat = max(self.TestResults);

            Vars = cell(1,1+length(self.quants));
            Vars{1} = TestStat;
            VarNames = cell(1,1+length(self.quants));
            VarNames{1} = 'Test_Stat';

            QuantsAsPercent = floor(100*self.quants);
            for i = 1:length(QuantsAsPercent)
                VarNames{i+1} = strcat('CV_',int2str(QuantsAsPercent(i)));
                Vars{i+1} = self.TestCVs(i);
            end

            tab = table(Vars{:},'RowNames',{TType},'VariableNames',VarNames);
        end

        function [tab]  = GenerateBubbleStartEndTimes(self, Beta, MinDurationInYears, SamplesPerYear)
            % Generates a table of bubble start and end times using the test's date stamping algorithm
            % Beta is the critival value sequence level (typically one of 0.9, 0.95, 0.99)

            if isempty(self.simResults)
                % Critical values have not bee supplied, so we can't do anything about this.
                error('Critical values have not been supplied.  Unable to generate table.')
            end

            if self.TestType == UnitRootTestType.ADF
                error('Test type ADF does not support start-end date detection.  Unable to generate table.')
            end

            if self.TestType == UnitRootTestType.GSADF && nargin < 4
                error('Test type GSADF requires MinDurationInYears and SamplesPerYear as the final parameters.  Unable to generate table.')
            end

            CVSequence = quantile(self.simResults,Beta,2);

            BubbleStartIndices = [];
            BubbleEndIndices = [];
            InBubble = false;
            DateIndexOffset = self.Tw-1;
            for i = 1:length(CVSequence)
                if self.TestResults(i) > CVSequence(i)
```

```
                    if ~InBubble
                        InBubble = true;
                        BubbleStartIndices = [BubbleStartIndices; DateIndexOffset +↙
i];
                    end
                else
                    if InBubble
                        InBubble = false;
                        BubbleEndIndices = [BubbleEndIndices; DateIndexOffset + i];
                    end
                end
            end

            if length(BubbleEndIndices) ~= length(BubbleStartIndices)
                BubbleEndIndices = [BubbleEndIndices; self.T];
            end

            % Filter out bubbles smaller than the minimum duration
            MinDuration = 0;
            switch (self.TestType)
                case UnitRootTestType.SADF
                    MinDuration = log10(self.T);
                case UnitRootTestType.GSADF
                    % Min duration is delta*log10(T) where delta is
                    % frequency dependent.
                    MinDuration = SamplesPerYear*MinDurationInYears;
            end

            DurationFilter = (BubbleEndIndices - BubbleStartIndices) > MinDuration;

            BubbleStartDates = self.dates(BubbleStartIndices(DurationFilter));
            BubbleEndDates = self.dates(BubbleEndIndices(DurationFilter));

            BubbleNames = cell(length(BubbleStartDates),1);
            for j = 1:length(BubbleNames)
                BubbleNames{j} = strcat('Bubble',int2str(j));
            end

            tab = table(BubbleStartDates, BubbleEndDates, 'RowNames', BubbleNames,↙
'VariableNames',{'StartDate','EndDate'});
        end

        function [hAx,hLine1,hLine2] = GeneratePlot(self, Beta)
            % Generates a plot of the unit root bubble test results
            % Beta is the critival value sequence level (typically one of 0.9, 0.95,↙
0.99)

            if isempty(self.simResults)
                % Critical values have not bee supplied, so we can't do anything about↙
this.
                error('Critical values have not been supplied.  Unable to generate↙
plot.')
            end

            if self.TestType == UnitRootTestType.ADF
```

```
                error('Test type ADF does not support plotting.  Unable to generate plot.')
            end

            Results = [NaN(self.Tw-1,1);self.TestResults];
            CVSequence = [NaN(self.Tw-1,1); quantile(self.simResults,Beta,2)];

            figure
            [hAx,hLine1,hLine2] = plotyy([self.dates,self.dates],[Results,CVSequence], self.dates,self.priceOnDividend);
            hAx(2).YLim = [-max(self.priceOnDividend) max(self.priceOnDividend)];
            range = max(Results) - min(Results);
            hAx(1).YLim = [min(Results)-1 max(Results) + range];

            ResultsLegenedItem = '';
            switch (self.TestType)
                case UnitRootTestType.SADF
                    ResultsLegenedItem = 'The backward ADF sequence (left axis)';
                case UnitRootTestType.GSADF
                    ResultsLegenedItem = 'The backward SADF sequence (left axis)';
            end

            CVLegenedItem = ['The ' int2str(floor(100*Beta)) '% critical value sequence (left axis)'];

            legend(ResultsLegenedItem, CVLegenedItem, 'The price-dividend ratio (right axis)')
            legend('Location','northwest')
        end
    end
end
```

```
classdef UnitRootTests
    %UNITROOTTESTS Contains an implementation for each different unit root test
    %   Uses ADF test with a fixed lag order and unit root equation type supplied by the caller.
    %   The available tests are ADF, SADF, and GSADF.
    %   Note: GSADF can be slow for large data sets and small minimum window size,
    %   therefore it is recommended to use the MEX file UnitRootTestsCaller_mex
    %   instead of this class when performing GSADF tests on large data sets.

    methods (Static)
        function Results = ADF(data,k,EqType)
            Results = ADF_FL(data(:,1),k,EqType);
        end

        function Results = SADF(data, T, swindow0, k, EqType)
            % The Supremum ADF test.
            % Uses an increasing window size starting from swindow0
            % and ending at T. k is the ADF lag order and EqType
            % is the type of unit root process being tested
            dim = T-swindow0+1;
            Results = zeros(dim,1);
            for i=swindow0:1:T;
                Results(i-swindow0+1)= ADF_FL(data(1:i,1),k,EqType);
            end;
        end

        function Results = GSADF(data, T, swindow0, k, EqType)
            % The Generalised Supremum ADF test.
            % Uses an increasing window size that ranges from r1 to r2
            % k is the ADF lag order and
            % EqType is the type of unit root process being tested
            dim = T-swindow0+1;
            Results=zeros(dim,1);
            for r2=swindow0:1:T;
                dim0=r2-swindow0+1;
                rwadft=zeros(dim0,1);
                for r1=1:1:dim0;
                    rwadft(r1)= ADF_FL(data(r1:r2,1),k,EqType);
                end;
                Results(r2-swindow0+1) = max(rwadft);
            end;
        end
    end
end
```

```
classdef UnitRootNullSimulator
    %UNITROOTNULLSIMULATOR Simulator for the various types of null tests for each unit root test type
    %   For unit root test of type TType, conducts the given unit root test using null equation type EqType
    %   on M simulations of the null model specification
    %
    %   The null model is defined as:
    %       y_t = d*T^-eta + theta*y_(t-1) + e_t, where e_t is i.i.d. (0,sigma^2) and theta = 1
    %
    %   TType is an instance of the UnitRootTestType enumeration
    %   NumSamps is the sample size
    %   NumSims is the number of simulations to perform
    %   EqType is an instance of the UnitRootNullTestType enumeration
    %   Tw is the minimum test window size
    %   d, and eta are the null model parameters
    %   lag is the ADF lag order

    properties (SetAccess = private)
        TType    % The type of unit root test being simulated
        NumSamps % The number of observations in the sample
        NumSims  % The number of simulations to run
        EqType   % The equation type defining the null hypothesis
        Tw       % The minimum test window size
        d        % Null model parameter
        eta      % Null model parameters
        lag      % The ADF lag order
        useMex   % Set to true to use the compiled MEX file when running tests
    end

    methods
        function URNS = UnitRootNullSimulator(TType, T, M, EqType, Tw, d, eta, lag, useMex)
            % Constructor for UnitRootNullSimulator
            URNS.TType = TType;
            URNS.NumSamps = T;
            URNS.NumSims = M;
            URNS.EqType = EqType;
            URNS.Tw = Tw;
            URNS.d = d;
            URNS.eta = eta;
            URNS.lag = lag;
            URNS.useMex = useMex;
        end

        function simResults = PerformSimulation(self)
            % Performs the simulation steps for the specified unit root test

            tic
            % Generate the data
            data = self.GenerateData;

            % Store local variables to avoid parfor broadcast var warnings
            T = self.NumSamps;
```

```
            M = self.NumSims;
            swindow0 = self.Tw;
            k = self.lag;
            Eq = self.EqType;
            UseMex = self.useMex;

            switch(self.TType)
                case UnitRootTestType.ADF
                   simResults =  self.SimADF(data, M, k, Eq, UseMex);
                case UnitRootTestType.SADF;
                    simResults = self.SimSADF(data, T, M, swindow0, k, Eq, UseMex);
                case UnitRootTestType.GSADF
                    simResults = self.SimGSADF(data, T, M, swindow0, k, Eq, UseMex);
            end
            toc
        end
    end

    methods (Access = private)
        function simResults = SimADF(~,data, M, k, Eq, UseMex)
            simResults = zeros(1,M);
            if UseMex
                parfor j=1:M;
                    simResults(1,j) = UnitRootTestsCaller_mex(UnitRootTestType.ADF,↙
data(:,j), 0, 0, k, Eq);
                end;
            else
                parfor j=1:M;
                    simResults(1,j) = UnitRootTests.ADF(data(:,j), k, Eq);
                end;
            end
        end

        function simResults = SimSADF(~, data, T, M, swindow0, k, Eq, UseMex)
            simResults = zeros(T-swindow0+1,M);
            if UseMex
                parfor j=1:M;
                    simResults(:,j) = UnitRootTestsCaller_mex(UnitRootTestType.SADF,↙
data(:,j), T, swindow0, k, Eq);
                end;
            else
                parfor j=1:M;
                    simResults(:,j) = UnitRootTests.SADF(data(:,j), T, swindow0, k,↙
Eq);
                end;
            end
        end

        function simResults = SimGSADF(~, data, T, M, swindow0, k, Eq, UseMex)
            simResults = zeros(T-swindow0+1,M);
            if UseMex
                parfor j=1:M;
                    simResults(:,j) = UnitRootTestsCaller_mex(UnitRootTestType.GSADF,↙
data(:,j), T, swindow0, k, Eq);
                end;
```

```
            else
                parfor j=1:M;
                    simResults(:,j) = UnitRootTests.GSADF(data(:,j), T, swindow0, k,↙
Eq);
                end;
            end
        end

        function data = GenerateData(self)
            % Generate M lots of data samples of length T
            % according to the null specification equation (see eqn. 3 in PSY)
            %   y_t = d*T^-eta + theta*y_(t-1) + e_t, where e_t is i.i.d. (0,sigma^2)↙
and theta = 1
            T = self.NumSamps;
            M = self.NumSims;

            rng('default');
            rng(M);
            e=randn(T,M);
            a=self.d*T^(-self.eta);
            data=cumsum(e+a);
        end
    end
end
```

```
classdef UnitRootNullTestType < uint16
    enumeration
        UnitRoot (1)
        UnitRootWithDrift (2)
        UnitRootWithDriftAndTrend (3)
    end

    methods
        function val = AsInt(self)
            val = uint16(self);
        end

        function val = AsDouble(self)
            val = double(self);
        end
    end
end
```

```
classdef (Sealed) CriticalValuesCache < handle
    %CRITICALVALUESCACHE Singleton class for managing the cache of test critical values
    %   Unit root tests rely on simulating the null hypothesis model many times
    %   to determine the critical value sequences for a fixed set of model and test parameters.
    %   Since this process can be time consuming, it is preferable to cache the results
    %   of simulations for quick retrieval at a later time when conducting significance tests.
    %
    %   CriticalValuesCache is a single-instance class (singleton) which provides methods for
    %   interacting with the cached critical value sequences, as well as generating new
    %   entries for the cache when no existing match is found.
    %
    %   Typical usage is to get a handle to the cache and then query it for values, optionally
    %   asking the cache to generate the required values if they don't exist.
    properties (Access = private)
        cachePath % The location of the cache in the file system
    end

    properties (Constant)
        CacheFolder = 'CriticalValues'; % The cache folder name
    end

    methods (Access = private)
        function obj = CriticalValuesCache
            % Private constructor for CriticalValuesCache
            obj.InitialiseCacheFolder;
        end

        function InitialiseCacheFolder(self)
            % Initialises the cache folder location in the file system

            % Find the filesystem path to the cache
            folder = fileparts(which(mfilename));
            self.cachePath = fullfile(folder, self.CacheFolder);

            % If the cache directory doesn't exist, create it
            if ~isdir(self.cachePath)
                mkdir(self.cachePath);
            end
        end
    end

    methods (Static)
        function CVCache = getInstance
            persistent localCVCache
            if isempty(localCVCache) || ~isvalid(localCVCache)
                localCVCache = CriticalValuesCache;
            end
            CVCache = localCVCache;
```

```
        end
    end

    methods
        function cachePath = getCachePath(self)
            % Returns the location of the cache in the file system
             cachePath = self.cachePath;
        end

        function ClearCache(self)
            % Clears the cached files from the cache directory
            cacheFiles = fullfile(self.cachePath, '*.mat');
            delete(cacheFiles);
        end

        function [TestCVs, simResults, fileWasCreated] = getCVs(self, TType, T, M,↙
EqType, Tw, d, eta, lag, rebuild, quants, useMex)
            % Gets the critical values sequence for the supplied parameters
            %   Conducts M simulations of the null model specification.
            %   The null model is defined as:
            %       y_t = d*T^-eta + theta*y_(t-1) + e_t, where e_t is i.i.d. (0,↙
sigma^2) and theta = 1
            %
            %   TType is an instance of the UnitRootTestType enumeration
            %   T is the sample size
            %   M is the number of simulations to perform
            %   EqType is an instance of the UnitRootNullModelType enumeration
            %   Tw is the minimum test window size
            %   d and eta are the null model parameters
            %   lag is the ADF lag order
            %   rebuild is a logical value which will cause the simulations to run↙
again when set to true
            %   quants is a row vector of quantiles to use when calculating critical↙
values
            %   useMex is a boolean value, set to true to use the compiled mex file↙
when simulating
            %
            %   Unless rebuild is set to true, getCVs will attempt to locate a cached↙
copy of the simulated
            %   results from a previous run.  If no match is found in the cache, or if↙
rebuild is set to true,
            %   getCVs will force-run the simulation using the supplied parameters and↙
add the values to the
            %   cache upon completion

            % Construct the results file name from the supplied parameters
            % The filename format is;
            %   TType_T_M_EqType_Tw_d_eta_lag.mat
            resultsFile = strcat(TType.char,'_',int2str(T),'_',int2str(M),'_',int2str↙
(EqType.AsInt),'_',int2str(Tw));
            resultsFile = strcat(resultsFile,'_',num2str(d),'_',num2str(eta),'_',↙
int2str(lag),'.mat');
            resultsFile = fullfile(self.cachePath, resultsFile);

            % If the file exists, and we are not forcing a rebuild, load the results↙
```

```
from file
            % Otherwise, compute the results and store in the cache.
            if ~rebuild && exist(resultsFile, 'file') == 2
                load(resultsFile, 'simResults');
                fileWasCreated = false;
            else
                URNS = UnitRootNullSimulator(TType, T, M, EqType, Tw, d, eta, lag, ↙
useMex);
                simResults = URNS.PerformSimulation();
                save(resultsFile, 'simResults');
                fileWasCreated = true;
            end

            % Results are now stored in simResults.
            % Compute the critical values for the supplied quantiles
            switch (TType)
                case UnitRootTestType.ADF
                    TestCVs = quantile(simResults,quants);
                case UnitRootTestType.SADF
                    sadf = max(simResults,[],1);
                    TestCVs = quantile(sadf,quants);
                case UnitRootTestType.GSADF
                    gsadf = max(simResults);
                    TestCVs = quantile(gsadf,quants);
            end
        end
    end
end
```

```
function [ simResults ] = UnitRootTestsCaller(TType, data, T, swindow0, k, EqType)
%UNITROOTTESTSCALLER A wrapper function for calling the methods of the UnitRootTests class
%   This function exists primarily for the purpose of the MATLAB Coder MEX generator, which
%   requires the entry point to the code to be a function (not a class method).

simResults = [];

switch(TType)
    case UnitRootTestType.ADF
       simResults =  UnitRootTests.ADF(data(:,1), k, EqType);
    case UnitRootTestType.SADF;
        simResults = UnitRootTests.SADF(data(:,1), T, swindow0, k, EqType);
    case UnitRootTestType.GSADF
        simResults = UnitRootTests.GSADF(data(:,1), T, swindow0, k, EqType);
end

end
```

```
classdef UnitRootTestType < uint16
    enumeration
        ADF (1)
        SADF (2)
        GSADF (3)
    end
end
```