Code Appendix

Erik Andersen

2024-05-07

```
# HW 3 Code
# Load packages
pacman::p_load(tidyverse, magrittr, kableExtra, nleqslv)

# Load data
# I saved the date in HW2 where I already calculated the div-price ratio and dividend growth rate, so I
returns_df = read_csv(here::here("HW3", "data", "returns.csv"))
```

```
## Rows: 98 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (6): date, vwretd, vwretx, t90ret, div_price, div_growth
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Rename the returns sequences to things that make sense
returns_df %<>%
  rename("dividends" = vwretd,
         "no_dividends" = vwretx)

#### Question 1 ######
# First we will replicate the VARs
# We can just run them equation by equation
reg_returns = returns_df %>% lm(log((dividends+1) /lag((dividends+1))) ~ lag(log(div_price)) ,.)
reg_div_growth = returns_df %>% lm(log(div_growth) ~ lag(log(div_price)),.)
reg_div_price = returns_df %>% lm(log(div_price) ~ lag(log(div_price)),.)

# Calculate correlations between all the various variables
return_div_growth = cor(reg_returns$residuals, reg_div_growth$residuals)
return_div_price = cor(reg_returns$residuals, reg_div_price$residuals)
growth_price = cor(reg_div_growth$residuals, reg_div_price$residuals)

# Calculate the standard deviations
returns_sd = sd(reg_returns$residuals)
div_growth_sd = sd(reg_div_growth$residuals)
div_price_sd = sd(reg_div_price$residuals)

# Make table object
coefs = c(coef(reg_returns)[2], coef(reg_div_growth)[2], coef(reg_div_price)[2])
names(coefs) = c("Returns", "Div Growth", "Div Price")

standard_errors = c(summary(reg_returns)$coefficients[2,2], summary(reg_div_growth)$coefficients[2,2], s
```

```r
var_cov = matrix(c(returns_sd, return_div_growth, return_div_price, return_div_growth,
                   div_growth_sd, growth_price, return_div_price, growth_price,
                   div_price_sd), byrow = T, nrow = 3)
colnames(var_cov) = c("r", "Div Growth", "Div Price")

var_table = cbind(coefs, standard_errors, var_cov)


################
# Impulse response function for dividend growth

# Define shocks
epsilon_r = 1
epsilon_d = 1
epsilon_dp = 0

# Initialize series
ds = c(0, coefs[2]*0 + epsilon_d)
dgs = c(0, coefs[2]*0 + epsilon_d)
dps = c(0, coefs[3]*0 + epsilon_dp)
ps = c(0, (1 - coefs[3])*0 + (epsilon_d - epsilon_dp))
rs = c(0, coefs[1] *0 + epsilon_r)

# Simulate VAR forward
for(i in 2:20){
  ds = c(ds,ds[i] + coefs[2] * dps[i])
  dgs = c(dgs, coefs[2] *dps[i])
  dps = c(dps, coefs[3] *dps[i])
  ps = c(ps,ps[i] + (1 - coefs[3]) *dps[i])
  rs = c(rs, coefs[1] *dps[i])
}

impulse_response_growth = data.frame(cbind(c(-1,0:19),dgs,dps,ps,rs,ds))
colnames(impulse_response_growth) = c('t','dg','dp','p','r','d')

# Plot
plot_div_growth =
  impulse_response_growth |>
  as_tibble() |>
  pivot_longer(cols = -t) |>
  ggplot(aes(x = t, y = value, color = name)) +
  geom_line() +
  labs(x = "", y = "", color = "") +
  scale_color_brewer(palette = "Dark2", labels = c("Dividend Price Level", "Dividend Growth",
                                                   "Dividend Price", "Prices", "Returns")) +
  ggtitle("Response to Dividend Growth Shock") +
  cowplot::theme_cowplot()

ggsave(here::here("HW3", "plots", "div_growth_irf.pdf"))
```

```
## Saving 6.5 x 4.5 in image
```

```r
# Same thing for dividend yield
# Define shocks
er2 = -0.96 # rho
ed2 = 0
edp2 = 1

# Initialize series
ds2 = c(0, coefs[2]*0 + ed2)
dgs2 = c(0, coefs[2]*0 + ed2)
dps2 = c(0, coefs[3]*0 + edp2)
ps2 = c(0, (1 - coefs[3])*0 + (ed2-edp2)) # See homework for why shocks work like this
rs2 = c(0, coefs[1]*0 + er2)

# Simulate forward
for(i in 2:20){
  ds2 = c(ds2, ds2[i] + coefs[2]*dps2[i])
  dgs2 = c(dgs2, coefs[2]*dps[i])
  dps2 = c(dps2, coefs[3]*dps2[i])
  ps2 = c(ps2, ps2[i] + (1 - coefs[3])*dps2[i])
  rs2 = c(rs2, coefs[1]*dps2[i])
}

impulse_response_yield = data.frame(cbind(c(-1,0:19),dgs2,dps2,ps2,rs2,ds2))
colnames(impulse_response_yield) = c('t','dg','dp','p','r','d')

# Plot
plot_div_yield =
  impulse_response_yield |>
  as_tibble() |>
  pivot_longer(cols = -t) |>
  ggplot(aes(x = t, y = value, color = name)) +
  geom_line() +
  labs(x = "", y = "", color = "") +
  scale_color_brewer(palette = "Dark2", labels = c("Dividend Price Level", "Dividend Growth",
                                                   "Dividend Price", "Prices", "Returns")) +
  ggtitle("Response to Dividend Yield Shock") +
  cowplot::theme_cowplot()

ggsave(here::here("HW3", "plots", "div_yield_irf.pdf"))
```

```
## Saving 6.5 x 4.5 in image
```

```r
############ Question 2 ####################


### Part d
# Function to match the moments from parts b and c
# Define eta
eta = 0.0001

mpmoment = function(initial) {
  # Unpack argument. We have to do it this clunky way because the solver takes a vector
  mu = initial[1]
  delta = initial[2]
```

```
  phi = initial[3]

  # Calculate moments
  m1 = (1 + mu) * (2 + eta) / (2 + 2 * eta)
  m2 = (4 + 4 * mu^2 + 8 * mu + 4 * delta^2 + eta + 2 * eta + eta * mu^2) /
    (4 * (1 + eta))
  m3 = (delta^2 * (2 * phi + eta - 1) + (1 + mu)^2) / (1 + eta)
  return(c(m1 - 1.018, m2 - 1.03762, m3 - 1.03614256)) # These values from the calculated moments given
}

# Define parameters as
mu = 0.018
delta = 0.036
phi = -0.14

# Find values
moments = nleqslv(c(mu, delta, phi),mpmoment)$x |> as.data.frame()

colnames(moments) = c("")
rownames(moments) = c("Mu", "Delta", "Phi")


### Part e
# Function to calculate risk premium and risk free rate given different values of free parameters
risk_premia = function(eta, beta, alpha) {
  # Calculate stationary probabilities and transition matrix
  p = c(1 / (2 + 2 * eta), 1 / (2 + 2 * eta), eta / (1 + eta))
  Q = matrix(c(as.numeric(moments[3,]), 1 - as.numeric(moments[3,]) - eta, 0.5, 1 - as.numeric(moments[3
             nrow = 3)

  # Calculate endowment levels
  x = c(1 + as.numeric(moments[1,]) + as.numeric(moments[2,]), 1 + as.numeric(moments[1,]) - as.numeric

  # Calculate other parameters
  xf = x^-alpha
  rf = p %*% (1 / (beta * Q %*% as.matrix(xf)) - 1)
  xd = diag(x^(1 - alpha))
  one = matrix(c(1, 1, 1), nrow = 3)
  a = diag(3) - beta * Q %*% xd
  b = beta * Q %*% xd %*% one
  w = solve(a, b)

  # Initialize returns vectors
  r = matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0), nrow = 3)
  R = matrix(c(0, 0, 0), ncol = 3)

  # Calculate return rates
  for (i in 1:3) {
    for (j in 1:3) {
      r[i, j] = x[j] * (w[j] + 1) / w[i] - 1
    }
    R[, i] = Q[i, ] %*% r[i, ]
  }
```

```
  re = p %*% t(R)
  return(c(rf, re, re - rf))
}

### PArt f


# Initialize vector for simulations
sim = c()

# Loop over parameter space
for (alp in c(2, 10)) {
  for (bet in c(0.97, 0.997, 0.999)) {
    for (et in c(0.0001, 0.0002, 0.0005)) {
      epp = round(risk_premia(et, bet, alp) * 100, 4)
      epp = c(et, alp, bet, epp)
      sim = c(sim, epp)
    }
  }
}

sim = data.frame(t(matrix(sim, nrow = 6)))
colnames(sim) = c('Eta',
                  'Alpha',
                  'Beta',
                  'Risk-Free Rate',
                  'Expected Return',
                  'Risk Premium')
```