```
% Draft: Analysis
```

The Diebold-Li Model in the State-Space System

A variant of the Nelson-Siegel model, the Dynamic Nelson Siegel (DNS) or the Diebold-Li model is a reparameterization of the original equation. Given the time to maturity τ, and date t when the yield is observed, the yield $y_t(\tau)$ from the Diebold-Li model is a function of four parameters:

$$y_t(\tau_m) = L_t + S_t\left(\frac{1-e^{-\lambda\tau_m}}{\lambda\tau_m}\right) + C_t\left(\frac{1-e^{-\lambda\tau_m}}{\lambda\tau_m} - e^{-\lambda\tau_m}\right) \tag{1}$$

Here, L_t is the level or the the long-term factor as it influences long-term bonds, S_t is the slope or short-term factor, and C_t is the curvature or the medium-term factor. λ determines the maturity at which the loading on the curvature is maximized, and governs the exponential decay rate of the model.

The level, slope and curvature in the Diebold-Li model follow a first order vector autoregressive process or VAR(1), which has a state-space representation as follows:

$$\begin{pmatrix} L_t - \mu_L \\ S_t - \mu_S \\ C_t - \mu_C \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} L_{t-1} - \mu_L \\ S_{t-1} - \mu_S \\ C_{t-1} - \mu_C \end{pmatrix} + \begin{pmatrix} \eta_t(L) \\ \eta_t(S) \\ \eta_t(C) \end{pmatrix} \tag{2}$$

The corresponding measurement equation is:

$$\begin{pmatrix} y_t(\tau_1) \\ y_t(\tau_2) \\ \vdots \\ y_t(\tau_M) \end{pmatrix} = \begin{pmatrix} 1 & \frac{1-e^{-\lambda\tau_1}}{\lambda\tau_1} & \frac{1-e^{-\lambda\tau_1}}{\lambda\tau_1} - e^{-\lambda\tau_1} \\ 1 & \frac{1-e^{-\lambda\tau_2}}{\lambda\tau_2} & \frac{1-e^{-\lambda\tau_2}}{\lambda\tau_2} - e^{-\lambda\tau_2} \\ & \vdots & \\ 1 & \frac{1-e^{-\lambda\tau_M}}{\lambda\tau_M} & \frac{1-e^{-\lambda\tau_M}}{\lambda\tau_M} - e^{-\lambda\tau_M} \end{pmatrix} \begin{pmatrix} L_t \\ S_t \\ C_t \end{pmatrix} + \begin{pmatrix} \epsilon_t(\tau_1) \\ \epsilon_t(\tau_2) \\ \vdots \\ \epsilon_t(\tau_M) \end{pmatrix} \tag{3}$$

In matrix notation, we can write the state-space system for the three-dimensional vector of the mean adjusted factors f_t and observed yields y_t:

$$(f_t - \mu) = A(f_{t-1} - \mu) + \eta_t \tag{4}$$

$$y_t = \Lambda f_t + \epsilon_t \tag{5}$$

where, η_t and ϵ_t are orthogonal to each other and follow white noise processes with the following distribution:

$$\begin{bmatrix} \eta_t \\ \epsilon_t \end{bmatrix} \sim \text{WN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma_\eta & 0 \\ 0 & \Sigma_\epsilon \end{bmatrix} \right)$$

The mean-adjusted factors describe the latent states:

$$x_t = f_t - \mu \tag{6}$$

Susbtituting equation (6) in (4), the state-space system of the Diebold-Li model is characterized as:

$$x_t = Ax_{t-1} + \eta_t \tag{7}$$

Diebold and Li (2006) converted the prices of the government bonds to unsmoothed Fama-Bliss US treasury zero-coupon yields to develop the Dynamic Nelson Siegel model to approximate the dynamic nature of the structure of factors. However, I have constructed the DNS by directly using the treasury yields from FRED. I have aggregated the monthly data on the treasury yields of various maturities, ranging from 3 month to 30 years. Depicting a snapshot of the current yield levels in the market, each of the ten series has both a cross-sectional and a temporal dimension.

```
% has data from October 31, 1993 to August 30, 2020 collected from FRED
% no missing values in the original data
tres = readtable("data_df.csv");

% extract the "Dates" column and store it as "dates" table
dates = table2array(tres(:, 'Date'));

% extract column names of table "tres"
tres.Properties.VariableNames;


% removes the "Date" column
tresc= removevars(tres,{'Date'});
data = table2array(tresc);

% DataTable: has columns' names of yields of varying ttm
% and row names for each date

% yields of 10 term to maturity (TTM) in months
ttm =[3; 6; 12; 24; 36; 60; 84; 120; 240; 360];

% yields = dataset on 10 different maturities for in-sample modeling
yields = data(1:end,:);      % in-sample yields for estimation
```

Principal Component Analysis

After aggregating the yields of various maturities, I reduced the dimensionality of the dataset by performing PCA. I have written the code to manually perform the PCA, and then I compared the results with those from MATLAB's built-in "pca" function. Following are the steps that I employed in doing the factor analysis:

1. First, I demeaned (or mean centered) the data to ensure that all the variables are in the same scale and don't have have large variances and range. This ensures that the range of continuous variables contribute equally to the analysis. As PCA is sensitive to variances, mean centering precludes the variables with larger ranges to dominate those with smaller ranges.
2. Secondly, I calculated a (10×10) dimension of covariance matrix of the ten demeaned yields to find which which variables of the demeaned yields vary from each other with respect to the means. The correlation matrix informs the variables that are highly correlated, and consequently, have more redundant information.
3. As the variables from the previous step are correlated with other, I have now rotated the coordinate system to create a new set of variables (principal components) which are uncorrelated with each other, i.e. have zero covariance. From the covariance matrix, I computed the eigenvalues and the eigenvectors. This step diagonalizes the covariance matrix, decorrelating the relationship between the new variables. So, the off-diagonal entries are 0, and the diagonal entries are the eigenvalues, which are the variances of theeir corresponding eigenvectors.
4. Now, I have sorted the eigenvalues in the new coordinate system in the decreasing order of variances. Thereby, I have arranged the PCs such that the $\mathrm{PC1}$ explains the maximum variation present in the demeaned yields. Each PC is a linear combination of the mixtures of demeaned yields, and each $\mathrm{PC}\,i, i > 1$ explains the remaining variation in the dataset.
5. Finally, I caliberated the proportion of variance explained by each $\mathrm{PC}\,i$.

Unlike the original and demeaned yields, each principal component (PC) is orthogonal to each other, i.e. each $\mathrm{PC}\,i$ has zero covariance with $\mathrm{PC}\,j$ where $i \neq j$.Thus, the off-diagonal entries of the correlation matrix is 0. The first three PCs or the latent factors – $\mathrm{PC1}, \mathrm{PC2}, \mathrm{PC3}$, are also refered to as the level, slope, and curvature factors, respectively. The scree plot shows that the level and slope factors explain 94.28 and 5.26 percent of variance, respectively.

```
% deman the data (subract the mean of each column with the column's
% respective observation
% zero-centers the columns or standardizes the range of continuous vars
demeaned_yields = yields - mean(yields);

% covariance matrix of the demeaned yields
% diagonal entries are the variances and off-diagonals are covariance
covariance = cov(demeaned_yields);

% % calculate eigenvectors, and eigenvalues of the covariance matrix
% eigenval_diag: diagonal matrix of eignevalues
% diagonalized matrix due to 0 covariances but non-0 variances
```

```
% here, covariance * eigvector = eigvector * eigval_diag
[eigvector, eigval_diag] = eig(covariance);

% eigval: extracts eigenvalues (diagonal entries of eigval_diag)
% and stores them as a column vector
% eigenvalues of the diagonalized matrix are the variance of the PCs in the
% rotated coordinate system
eigval = diag(eigval_diag);


% sort in decreasing order of eigenvalue (variance of the PCs)
% store the indices of which columns the sorted eigenvalues come from
[eigval, indices]=sort(eigval,'descend');

% arrange the eigenvectors (PCs) in the descreasing order their
% corresponding eigenvalues (variances)
eigvector = eigvector(:,indices);

% generate PCA component space (PCA scores)
PCs = demeaned_yields * eigvector;


% variance explained by each PC
var_explained = var(PCs)/sum(var(PCs));
var_explained = var_explained';

colNames = {'Variance Explained (%)'};
var_explained_tab = array2table(var_explained,'VariableNames',colNames)
```

var_explained_tab = 10×1 table

	Variance Explained (%)
1	0.9428
2	0.0527
3	0.0036
4	0.0005
5	0.0002
6	0.0002
7	0.0000
8	0.0000
9	0.0000
10	0.0000

```
% verify that the variance explaind with each PC matches with the results
% above

verified_explained = diag(eigval_diag)/trace(eigval_diag)*100 ;
```

```
verified_explained = sort(var_explained, 'descend');

disp_varexp = {"variance Explained by Each PC:";...
    "---------------------------------";...
    verified_explained};
cellfun(@disp,disp_varexp)
```

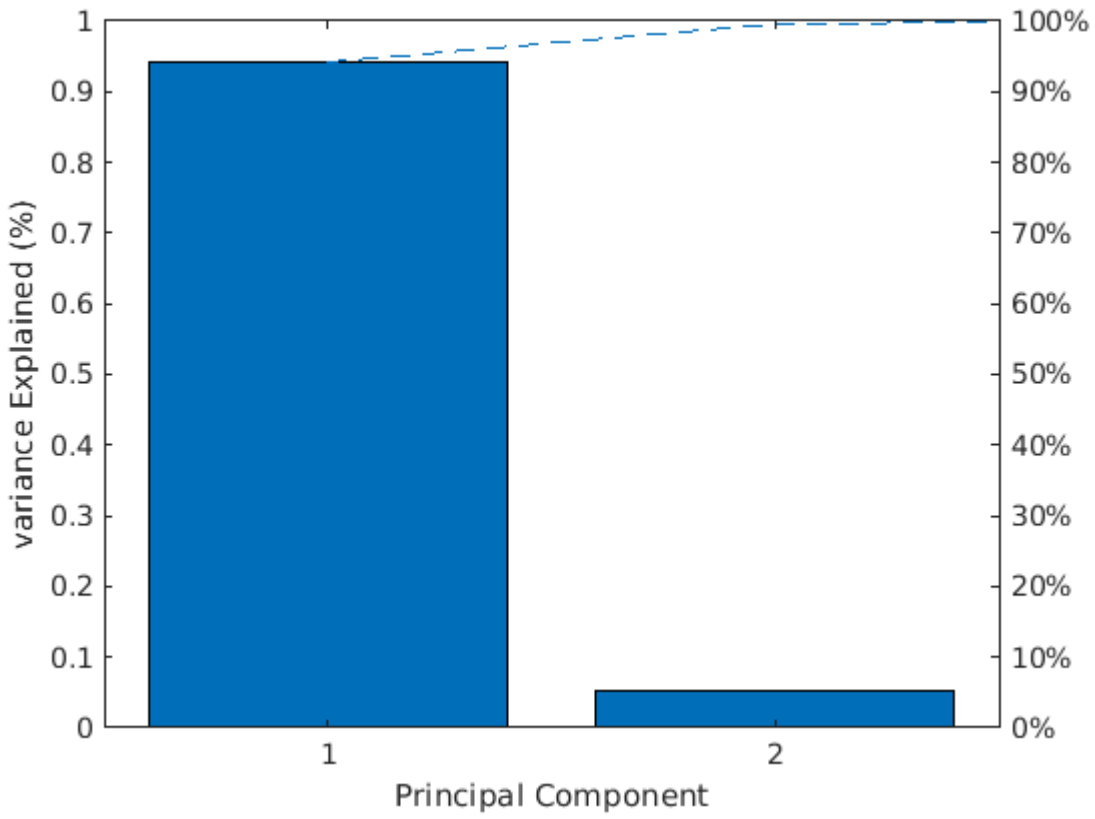
```
variance Explained by Each PC:
---------------------------------
    0.9428
    0.0527
    0.0036
    0.0005
    0.0002
    0.0002
    0.0000
    0.0000
    0.0000
    0.0000
```

Next, I have depicted a plot of variance explained by each PC, wherein the graph displays that PC1and PC2 explain majority of the variance, compared to the subsequent PCs.

```
% first three principal component score vectors are:
PCs3 = PCs(:,1:3);

% column names of the table with first 3 PCs "PCs3_table"
PCs3_heading = {'PC1' 'PC2'  'PC3'};
fig = figure('position',[200 200 400 180]);
PCs3_table=uitable('parent',fig,'data',PCs3,'columnname',PCs3_heading, ...
    'columnwidth',{100},'position',[10 10 350 170]);

% Scree plot that displays the percentage of variance explained by the
% first few PCs.
figure()
pareto(var_explained)
```

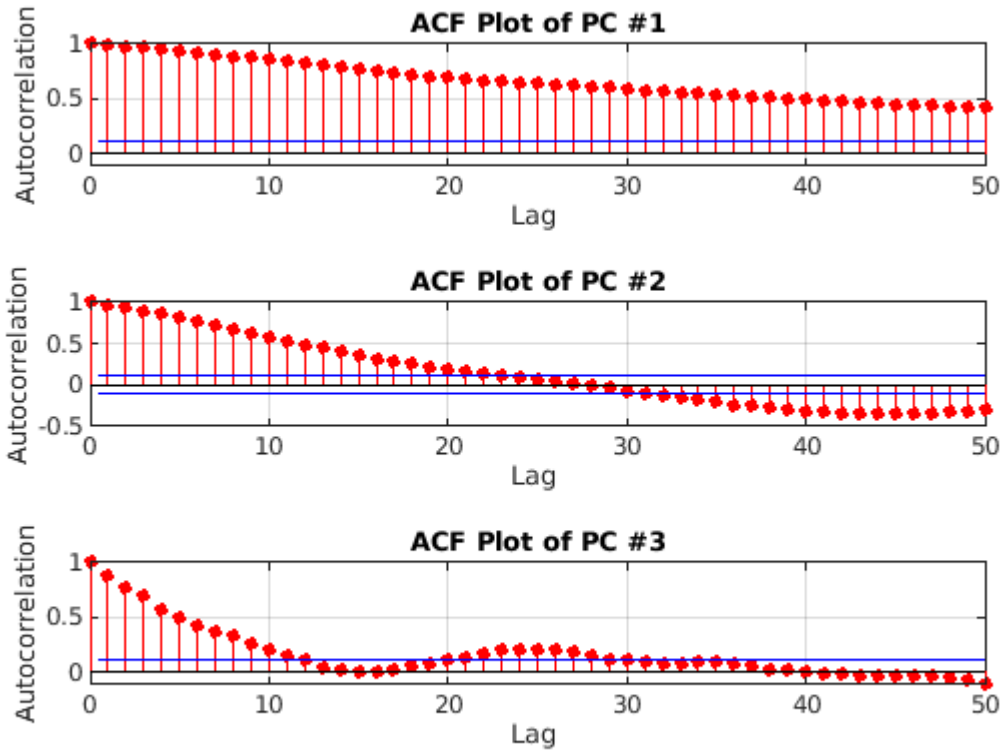
	PC1	PC2	PC3
1	3.6445	-1.4172	-0.016
2	4.4332	-1.8032	-0.050
3	4.4024	-1.9380	-0.161
4	4.0180	-1.7597	-0.087
5	5.6163	-1.9656	-0.186
6	7.0827	-2.4914	-0.544
7	8.3686	-2.2893	-0.629

```
xlabel('Principal Component')
ylabel('variance Explained (%)')
```



Now, I examined the autocorrelation present in the three PCs through the ACF plots. PC1 is heavily autocorrelated even after 50 lags, thereby showing high level of persistance. The two PCs are relatively less autocorrelated, with the curvature factor showing the least autocorrelation amongst the lags.

```
figure
hold on % retains current plot while adding new plots
for i=1:3
    subplot(3,1,i)
    autocorr(PCs3(:,i), 50)
    caption = sprintf('ACF Plot of PC #%d', i);
    title(caption, 'FontSize', 15);

    % label the x-axis and add extra space between two plots
    xlabel("Lag " + newline + "   ")
    ylabel("Autocorrelation")

end
hold off
```



Two-Step Diebold-Li Model with Fixed λ

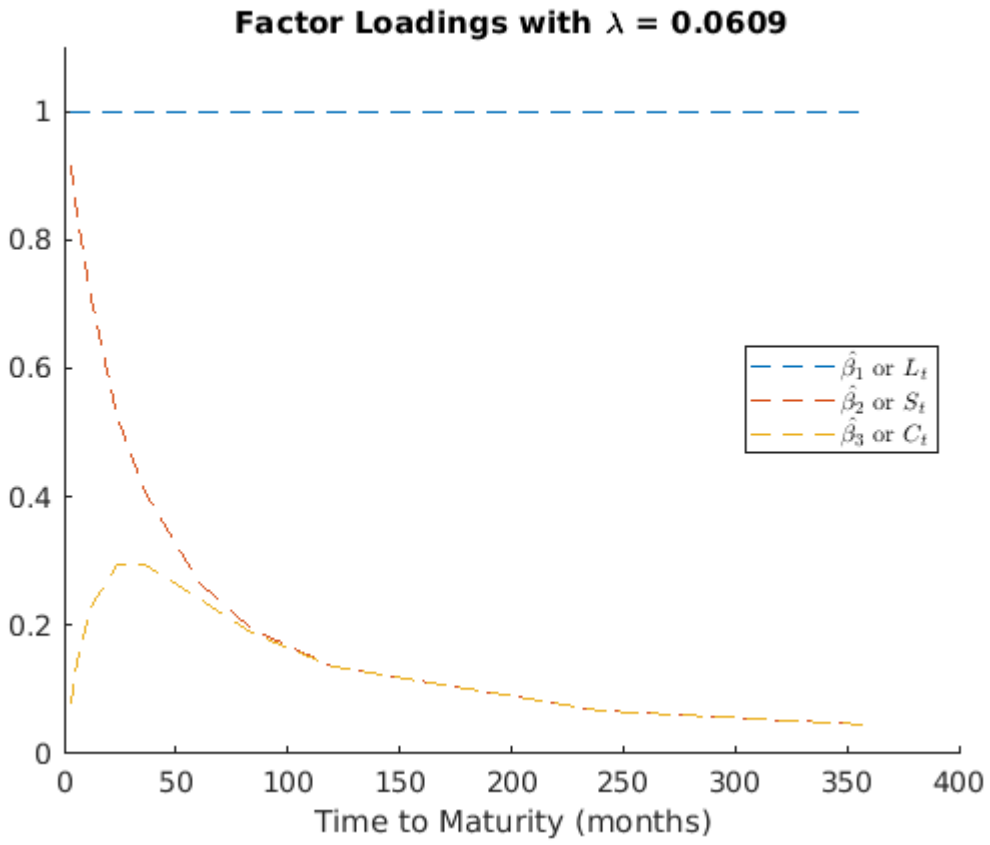
Finding the ex-ante value of the latent factors is tantamount to finding the ex-ante values of the yield curve as it is a function of the factors. Then, I have constructed the Diebold-Li model with the explanatory variables as factor forecasts. The two-step approach of fitting the Diebold and Li model and estimating the parameters of their yield curve is:

- Firstly, I have kept λ fixed and estimated the level, slope, and curvature parameters for each of the daily values of the yield curve. We can consider the regression coefficients from OLS as the three factors. Repeating this process for all observed yield curves yields a three-dimensional matrix of estimates of the three latent factors. Keeping λ fixed simplifies the estimation method from non-linear least squares to OLS, which creates a static Nelson Siegel model at each month.
- Secondly, I have fitted $\mathrm{VAR}(1)$ to the time series of factors derived in the first step.

In the Nelson-Siegel model, λ_t determines the time to maturity at which the loadings on the curvature, or the medium-term factor are maximized. These are usually yields that mature in 24 to 36 months. Dielbold and Li (2005) set $\lambda_t = 0.0609$ for all t. This is the value at which the loading on the curvature (medium-term factor) is maximized occurs at 30 months. Below, I have shown the first-step in the Diebold-Li model, and accumulated the coefficients (factor loadings) and the residuals from the OLS model. The factor loadings on β_1, β_2 and β_3 govern the level, slope and curvature, respectively, of the yield curve. Then, I plotted the factor loadings in one graphs.

```
lambda0 = 0.0609;
% X is a 10 by 3 matrix: as there is data on 10 different yields and 3 betas
% factors
X = [ones(size(ttm)) (1-exp(-lambda0*ttm))./(lambda0*ttm) ...
    ((1-exp(-lambda0*ttm))./(lambda0*ttm)-exp(-lambda0*ttm))];


% size(yields,1): queries the length of the 1st dim of 'yields" : 324
% betas = zeros(324, 3)
% stores the betas
betas = zeros(size(yields,1),3);

% numel : returns the number of elements in array "ttm" : 10
% resid = zeros(324, 10)
% stores the resid from the OLS model
resid = zeros(size(yields,1),numel(ttm));

% fix lambda and compute the 3 betas params for each monthly observation of the
% yield curve by OLS
% the betas coefficients from the OLS regression are equivalent to the 3
% factors: level, slope, and curvature
```

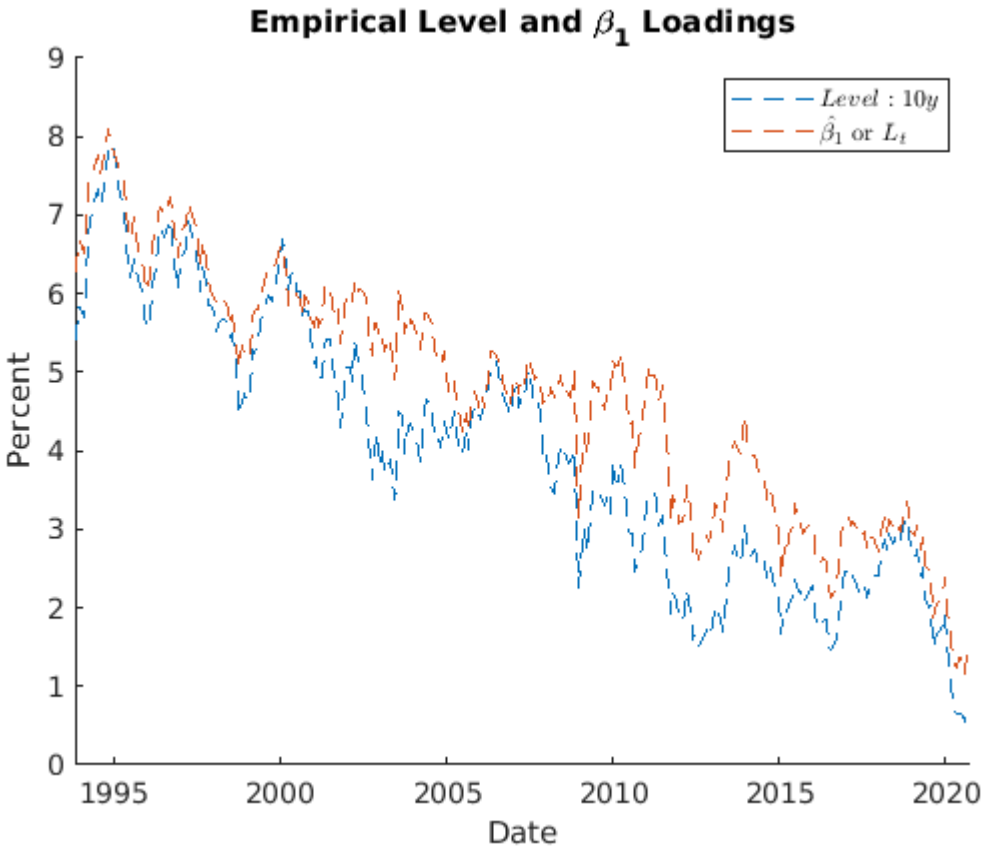
```
% collect the 3D time-series of the betass (estimated factors) by fitting
% the OLS model on each yield curve observed
% store the betas (regression coefficients) and resid

for i = 1:size(yields,1)
    OLSmod = fitlm(X, yields(i,:)', 'Intercept', false);
    betas(i,:) = OLSmod.Coefficients.Estimate';

    % raw residuals (observed - fitted values) from the OLS model
    resid(i,:) = OLSmod.Residuals.Raw';
end


% Plot the factor loadings
figure
hold on
plot(ttm,X)
title('Factor Loadings with \lambda = 0.0609')
xlabel('Time to Maturity (months)')
ylim([0 1.1])
legend({'$\hat{\beta_1}$ or $L_t$';'$\hat{\beta_2}$ or $S_t$'; ['$\hat{\beta_3}$ or' ..
    '  $C_t$']}, 'Interpreter', 'latex','location','east')
hold off
```



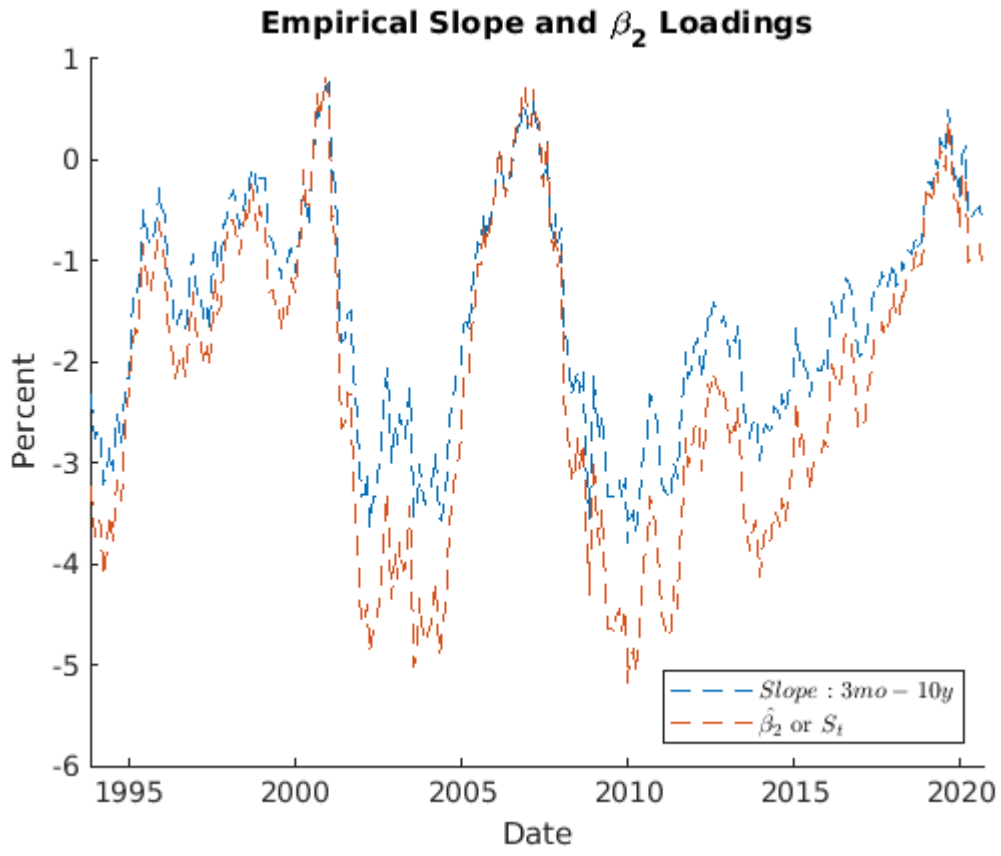
$\widehat{\beta}_1$ is a flat line parallel to the maturity axis, representing the constant L_t in the Diebold-Li equation (1). The loadings on $\widehat{\beta}_2$ or S_t is a function $\frac{1-e^{\lambda\tau}}{\lambda\tau}$, that starts at 1 approximately but monotonically decays to 0; therefore, it is a convex-shaped curve. Finally, the loadings on $\widehat{\beta}_3$ or C_t is a function $\frac{1-e^{\lambda\tau}}{\lambda\tau} - e^{-\lambda\tau}$ that starts at 0 and gradually converges to 0, affecting the medium-term yields. λ_t governs the pace of exponential decay wherein small values of λ_t slows the decay and better fits the curve at longer term maturities. This contrasts with large values of λ_t which fastens the rate of recay and better fits shorter-term maturities. Now, I have compared the path of the level, slope and curvature with their corresponding factor loadings from the first step (OLS regresson) in the Diebold-Li model. The 10-year Treasury yield is a measure of level. The slope or the yield spread is the difference between the 3-month and 10-year Treasury yield. Lastly, the curvature is measured as $2 \times 2y - 10y - 3\text{mo}$.

```
% actual level, slope and curvature from the dataset
% measure of level: L_t
tres_level = tres{:, 'tres10y'};

% measure of slope S_t = yield spread
tres_slope = tres{:, 'tres3mo'} - tres{:, 'tres10y'};

% measure of curvature C_t
tres_curvature = 2*tres{:, 'tres2y'} - tres{:, 'tres10y'} - tres{:, 'tres3mo'};

% stack them together in a matrix
compare_mat = [tres_level tres_slope tres_curvature betas];

% Convert the matrix into a table with variable names
compare_tab = array2table(compare_mat,...
    'VariableNames',{'level','slope','curvature', 'beta1', 'beta2', 'beta3'});

% convert the string of dates into datetime format
dates_dt = array2table(datetime(dates,'InputFormat','yyyy-MM-dd'), ...
    'VariableNames',{'Date'});

% stack the datetime dates with the level, slope and curvature in a table
compare_dt = [dates_dt, compare_tab];


% In each separate graph, plot the level, slope and curvature
% alsongside its corresponding factor loadings

figure
hold on
plot(dates, compare_mat(:,[1,4]))
title('Empirical Level and \beta_1 Loadings')
xlabel('Date')
ylabel('Percent')
```

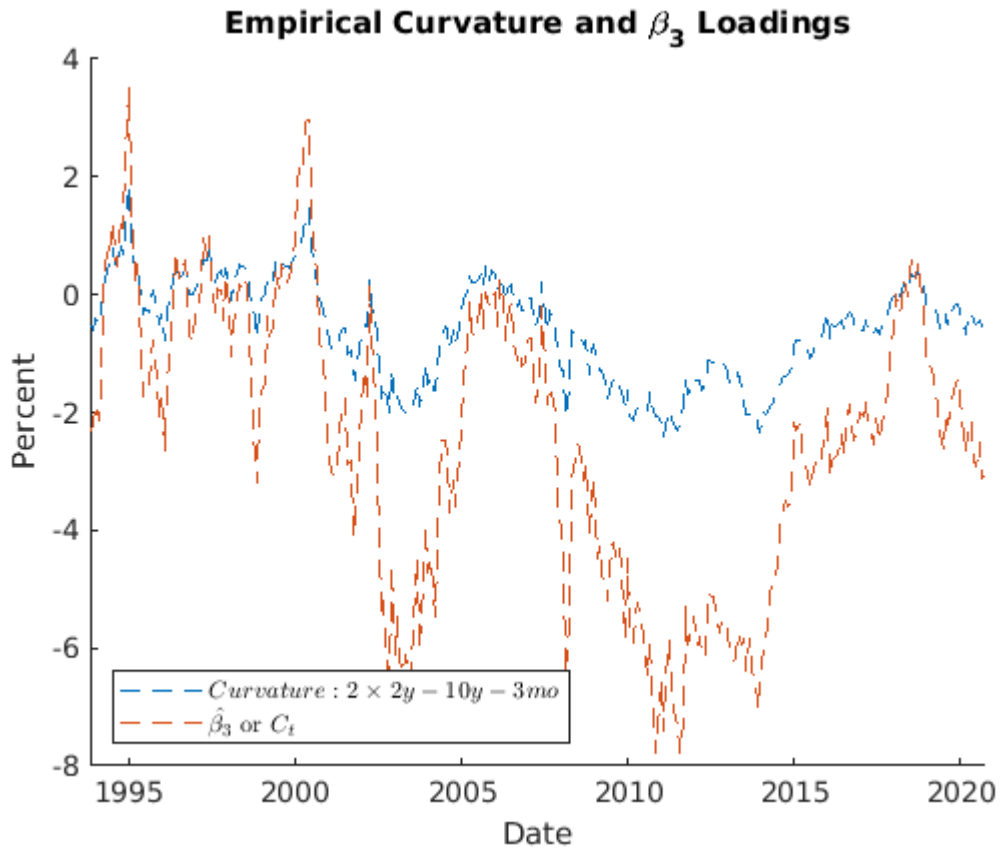
```
legend({'$Level: 10y$';'$\hat{\beta_1}$ or $L_t$'},'Interpreter', 'latex', ...
    'location','northeast')
hold off
```




```
figure
hold on

plot(dates, compare_mat(:,[2,5]))
title('Empirical Slope and \beta_2 Loadings')
xlabel('Date')
ylabel('Percent')
legend({'$Slope: 3mo-10y$';'$\hat{\beta_2}$ or $S_t$'},'Interpreter', 'latex', ...
    'location','southeast')
hold off
```




```
figure
hold on

plot(dates, compare_mat(:,[3,6]))
title('Empirical Curvature and \beta_3 Loadings')
xlabel('Date')
ylabel('Percent')
legend({'$Curvature: 2\times2y-10y-3mo$';'$\hat{\beta_3}$ or $C_t$'},'Interpreter', ...
    'latex','location','southwest')
hold off
```



The empirical level, slope and curvature factors, alonside their corresponding factor loadings closely follow each other, as also reflected by the high positive correlation with each other :

$\rho(10y, \text{level}) = 0.9615,\ \rho(3\text{mo} - 10y, \text{slope}) = 0.9939, \rho(2 \times 2y - 10y - 3\text{mo}, \text{curvature}) = 0.9781.$

```
corr_compare = corrcoef(compare_mat);
rowNames = {'10y','3mo-10y','2*2y-10y-3mo', 'beta 1', 'beta 2', 'beta 3'};
colNames = {'level','slope','curvature', 'beta 1', 'beta 2', 'beta 3'};
corr_compare_tab = array2table(corr_compare,'RowNames',rowNames, ...
    'VariableNames',colNames)
```

corr_compare_tab = 6×6 table

	level	slope	curvature	beta 1	beta 2	beta 3
1 10y	1.0000	0.0950	0.5617	0.9615	0.1405	0.5959
2 3mo-10y	0.0950	1.0000	0.5785	-0.1337	0.9939	0.5379
3 2*2y-10y-3mo	0.5617	0.5785	1.0000	0.3333	0.6413	0.9781
4 beta 1	0.9615	-0.1337	0.3333	1.0000	-0.1002	0.3673
5 beta 2	0.1405	0.9939	0.6413	-0.1002	1.0000	0.6033
6 beta 3	0.5959	0.5379	0.9781	0.3673	0.6033	1.0000

```
% Summary stats of the latent variables from the Diebold-Li model (DNS)
```

```
tab_betas=table();
tab_betas.Min=min(betas)';
tab_betas.Mean=mean(betas)';
tab_betas.Max=max(betas)';
tab_betas.Std_dev=std(betas)';
tab_betas.Properties.RowNames = {'beta 1', 'beta 2', 'beta 3'};

disp_sumstat_betas = {"Summary Statistics of the Latent Variables from DNS:";...
    "---------------------------------";...
    tab_betas};
cellfun(@disp,disp_sumstat_betas)
```

```
Summary Statistics of the Latent Variables from DNS:
---------------------------------
```

	Min	Mean	Max	Std_dev
beta 1	1.1602	4.7137	8.1034	1.5517
beta 2	-5.1804	-2.2325	0.81551	1.5339
beta 3	-7.7825	-2.491	3.5225	2.4178

The descriptive statistics indicates that the curvature factor is most volatile, followed by the level and slope factors, respectively.

After fitting the three-dimensional data of factors, I have fit a first – order VAR model on this estimated time series of factors. Using the multivariate least squares (MLS) method, which is a multivariate analog to the OLS done equation-by-equation, I estimated the coefficeint matrix A. The state transition matrix in equation (7) is $X_t = AX_{t-1} + \eta_t$

Assuming that $E\left[\eta_t X_{t-1}^T\right] = 0$, and the error from equation (7) is $\eta_t = X_t - \mathrm{AX}_{t-1}$ (8)

Substituting equation (8) in endogeneity assumption equation, $E\left[(X_t - \mathrm{AX}_{t-1})X_{t-1}^T\right] = 0$

$$\Rightarrow E\left[X_t X_{t-1}^T\right] = A\,E\left[X_{t-1} X_{t-1}^T\right] \Rightarrow A = E\left[X_t X_{t-1}^T\right] E\left[X_{t-1} X_{t-1}^T\right]^{-1}$$

As the data has T time periods of observations, let X_0 and X_1 be matrices of length $(T-1)\times n$, then the estimator of A is given by: $\hat{A} = X_0 X_1^T \left(X_1 X_1^T\right)^{-1}$

```
 % current betas values with one less observation
betas0 = betas(1:end-1,:);

% (t-1) lagged betas values
betas1 = betas(2:end,:);

% coefficient matrix of the lagged betass
A = (betas1'*betas0)/(betas0'*betas0)
```

```
A = 3×3
    0.9985    0.0098   -0.0053
   -0.0104    0.9163    0.0552
   -0.0097    0.0162    0.9617
```

Now, I have calculated the variance covariance matrix of the residuals from the VAR(1) and OLS model from the 2-step Diebold-Li method. These matrices are Σ_η and Σ_ϵ, respectively from the distribution of the errors discussed earlier:

$$\begin{bmatrix} \eta_t \\ \epsilon_t \end{bmatrix} \sim \text{WN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Sigma_\eta & 0 \\ 0 & \Sigma_\epsilon \end{bmatrix} \right)$$

```
% Innovations of VAR and its variance-cov matrix
var_error = betas0' - A*betas1'; var_error = var_error';
sigma_eta = cov(var_error);

disp_sigma_eta = {"variance Covariance Matrix of the VAR(1) Innovations:";...
    "-----------------------------------";...
    sigma_eta};
cellfun(@disp,disp_sigma_eta)
```

```
variance Covariance Matrix of the VAR(1) Innovations:
---------------------------------
    0.0592   -0.0589    0.0167
   -0.0589    0.1267   -0.0413
    0.0167   -0.0413    0.4656
```

```
% H0: 10-by-10 covariance matrix of the residuals from the OLS model
sigma_epsilon = cov(resid);
disp_sigma_epsilon = {"variance Covariance Matrix of the Residuals from OLS:";...
    "-----------------------------------";...
    sigma_epsilon};
cellfun(@disp,disp_sigma_epsilon)
```

```
variance Covariance Matrix of the Residuals from OLS:
---------------------------------
    0.0048   -0.0019   -0.0050   -0.0020    0.0009    0.0034    0.0040   -0.0001   -0.0003   -0.0037
   -0.0019    0.0019    0.0009   -0.0003   -0.0007   -0.0004   -0.0002    0.0007    0.0005   -0.0004
   -0.0050    0.0009    0.0067    0.0025   -0.0010   -0.0040   -0.0050    0.0001   -0.0004    0.0052
   -0.0020   -0.0003    0.0025    0.0033    0.0005   -0.0033   -0.0036   -0.0015    0.0008    0.0038
    0.0009   -0.0007   -0.0010    0.0005    0.0012   -0.0001   -0.0004   -0.0012   -0.0001    0.0010
    0.0034   -0.0004   -0.0040   -0.0033   -0.0001    0.0043    0.0041    0.0010   -0.0008   -0.0043
    0.0040   -0.0002   -0.0050   -0.0036   -0.0004    0.0041    0.0062    0.0011   -0.0003   -0.0059
   -0.0001    0.0007    0.0001   -0.0015   -0.0012    0.0010    0.0011    0.0030   -0.0013   -0.0017
   -0.0003    0.0005   -0.0004    0.0008   -0.0001   -0.0008   -0.0003   -0.0013    0.0076   -0.0057
   -0.0037   -0.0004    0.0052    0.0038    0.0010   -0.0043   -0.0059   -0.0017   -0.0057    0.0117
```

VAR(p) is:

$f_{t+h} = a_0 + \sum_{i=1}^{p} A_j f_{t-j} + \epsilon_t \ , \ \forall\, t = 1, 2, \ldots, T \,,$

where ϵ_t is a $(K \times 1)$ vector of errors, A_j is a $(K \times K)$ matrix of coefficients with *p* lags of *y.* As there are 3 latent factors in this model, $K = 3.$ In the multivariate normal representation, f is a $(\mathrm{RT} \times 1)$ vector, wherein all the observations of the level variable (for all time periods), followed by those of the slope and curvature are stacked in one column vector.

In matric-variate normal distribution terms, the VAR is $F = \mathrm{XA} + E,$ where *E* and ϵ are vectorization of errors similar to those of y and F. $A = [a_0\,, A_1, \ldots, A_p]^T \Rightarrow \alpha = \mathrm{vec}(A).$ Thus α is a $(\mathrm{RK} \times 1)$ vector.

$y = (I_K \otimes X)\, \alpha + \epsilon,$ where $\epsilon \sim N(0, \Sigma \otimes I_M)$

From among the four prior available, I will use Monte Carlo integration when analytical results are present in diffuse and Minnesota prior. Alternatively, I will use Gibbs Sampling when employing the Independent Normal-Wishart prior. Here, I will take "burn-in" draws and store only those draws that have converged.

replaced:

K with R:

F with F

M with K

Z and Z1 with G and G1

```
% Originally written by Koop and Korobilis (2009),I have modified it.


% Direct h-step ahead forecasts where there are p lags of F:
% f(t+h) = A0 + f(t) x A1 + ... + f(t-p+1) x Ap + e(t+h)



% VARcoef is the (R x K) matrix of VAR coefficients,
% alpha is the RM x 1 column vector of vectorized VAR coefficients, i.e.
% alpha = vec(VARcoef), and sigma_eps is a covariance matrix of dimension (K x K).


randn('seed',2);
rand('seed',2);


% ================== Introductory Specifications ==================
```

```
% Specify the VAR model

% 1: if there is an intercept; 0 otherwise
intercept = 1;

% Number of lags on dependent variables
p = 1;

% 1: forecast h-step ahead, 0: no forecasts
% where h = forecast horizon
forecast = 1;

% number of times to draw from the predictive density
% for each generated draw of the parameters
repeat = 50;

% forecast horizon or periods
horizon = 1;

% 1: Calculate impulse resp, 0: no impulse resp
impulses = 1;

% Compute impulse resp for h_irf number of periods or horizons
h_irf = 12;

% Set prior_dist for the BVAR model:

% prior_dist = 1 is Diffuse or'Jeffreys' prior
% prior_dist = 2 is Minnesota
% prior_dist = 3 is Normal-Wishart
% prior_dist = 4 is Independent Normal-Wishart

% use MC integration when prior_dist = 1,2,3, and
% use Gibbs sampler when prior_dist = 4


prior_dist = 2;


% Setting the Gibbs Sampler

% Number of saved draws
num_draw_save = 12000;

% Number of discarded or (burn-in) draws
num_draw_discard = 3000;


% Parameter values converge to the true values when we use analytical results
if prior_dist ==1 || prior_dist == 2 || prior_dist == 3
    num_draw_discard = 0*num_draw_discard;
end

% Total number of draws is the sum of saved and discarded draws
```

```
num_total_draws = num_draw_save + num_draw_discard;

% Prints every "iterations"-th iteration
iterations = 3000;
```

```
%========================== Organizing the Data ==========================

% the data is "betas" i.e the latent factors: level, slope and curvature
% betas is a (T by K) matrix

% nobs = number of observations in the dataset
[nobs, K] = size(betas);

% Model specification for direct forecasts
if forecast==1

    % Check if the input is incompatable
    if horizon <=0
        error('The forecasting horizon number is incorrect')
    end

    % VAR specification based on the forecasting method

        F1 = betas(horizon + 1:end,:);
        F2 = betas(2:end-horizon,:);
        nobs = nobs - horizon - 1;

else
   F1 = betas;
   F2 = betas;
end

% Generate lagged values of F matrix,
% which will constitute a variable of the X matrix
% F is [T x K]. ylag is [T x (Mp)]

Flag = lagf(F2,p);

% Define X matrix that consists of the intercept, and
% the lags of the latent variables


if intercept

    % ones(nobs-p,1): create a vector of 1s of dim(nobs-p)
    % X1 matrix consists  a vector of constant 1s and lagged betas

    X1 = [ones(nobs-p,1) Flag(p+1:nobs,:)];
else
    X1 = Flag(p+1:nobs,:);
end
```

```
% size of final matrix X after taking the lags
% X1 is the name of the final matrix
% R is (4 by 3) as it includes a constant
% nobs3 and R: display the number of rows and columns in X1, respectively
[nobs3, R] = size(X1);


% Create the block diagonal matrix G such that y = beta * G + error
% eye(K) returns an K -by- K identity matrix with ones on the main diagonal
% and zeros elsewhere
% returns the Kronecker tensor product of matrices eye(K) and X1

G1 = kron(eye(K),X1);


% F matrix: First row of Flag has 0s
% Delete the first row from Flag to match the dimensions of X matrix
% final F matrix inputted in the VAR

F1 = F1(p+1:nobs,:);

% T is the number of actual time series observations of F and X
T = nobs - p;
```

```
%============================ Set up of the Forecasting Model ============================

% Keep the last "horizon" num of observations to evaluate (pseudo-)forecasts
if forecast==1

    % matrix F_pred stores 500,000 forecasted draws of each latent factor
    F_pred = zeros(num_draw_save*repeat,K);

    % vector pred_likhood stores the predictive likelihood
    pred_likhood = zeros(num_draw_save,1);

    % Direct forecasts: store the last observation to evaluate the model's
    % performance
        F = F1(1:end-1,:);
        X = X1(1:end-1,:);
        G = kron(eye(K),X);
        T = T - 1;

% No forecasts: keep all observations
else
    F = F1;
    X = X1;
    G = G1;
end
```

```
%============================ Set up of Impulse Response Function ========================
```

```
% Create matrices to store forecasts
if impulses == 1;

    % impulse resp to a shock in level
    imp_level = zeros(num_draw_save,K,h_irf);

    % impulse resp to a shock in slope
    imp_slope = zeros(num_draw_save,K,h_irf);

    % impulse resp to a shock in curvature
    imp_curv = zeros(num_draw_save,K,h_irf);

    bigj = zeros(K,K*p);
    bigj(1:K,1:K) = eye(K);
end
```

```
%========================== Initialize the Parameters ==============================

% Obtain the Maximum Likelihood estimators

% estimate a matrix of regression coefficients  (A_ols is named as phi in my thesis)
% dim(A_ols) = 4 x 3 = (r x k)

A_ols = inv(X'*X)*(X'*F);

% A_ols_vec = vec(A_ols): stack the matrix cols into a vector of params
% each column is stacked one below the other
% creates an (rk x 1) = (12 x 1) vector of coefficients from A_ols

A_ols_vec = A_ols(:);

% Sum of squared errors of the VAR
% dim(sse) = 3 x 3
sse = (F - X*A_ols)'*(F - X*A_ols);
sigma_ols = sse./(T-R+1);

% Initialize Bayesian posterior parameters using the OLS values

% single draw from alpha's posterior
% alpha is (12 x 1) vector of the 9 VAR coefficients and 3 constants
% (alpha is named as A in my thesis)

alpha = A_ols_vec;

% single draw from the posterior of VARcoef of dim 4 x 3
VARcoef = A_ols;

% SSE based on each draw of VARcoef
sse_gibbs = sse;

%  single draw from the posterior of sigma_eps
```

```
sigma_eps = sigma_ols;

% eye(K): identity matrix of dim 3
IXF =  kron(eye(K),(X'*F));


% Store the posterior draws


% initially, create a matrix of zeros of dim(12,000 x 12)
% then save posterior draws of alpha
alpha_draws = zeros(num_draw_save,R*K);

% initially, create a matrix of zeros of dim(12,000 x 4 x 3)
% then save posterior draws of VARcoef
VARcoef_draws = zeros(num_draw_save,R,K);

% initially, create a matrix of zeros of dim(12,000 x 3 x 3)
% then save posterior draws of sigma_eps
sigma_draws = zeros(num_draw_save,K,K);
```

Now, I have loaded the file that contains the hyperparameters inserted in the prior distributions. These hyperparameters are used in the BVAR model. The file comprises of a function that stores priors for each of the aforementioned four priors.

In the Minnesota prior, I have estimated the VAR coefficient matrix A, where $A \sim N(A_{\text{mp}}, \Sigma_{\text{mp}})$

$$A_{\text{mp}} = \begin{cases} 0.95, & \text{for } a_{\text{ii}}\ i.e. \text{ the coefficient associated with the first lag} \\ 0, & \text{for } a_{\text{ij}}, \text{ where } i \neq j \end{cases}$$

The Minensota prior presupposes the prior covariance matrix Σ_{mp} to be diagonal, determined by the three hyperparameters – d_1, d_2, d_3 through the indicator funtion as follows:

$$\Sigma_{i,\text{jj}} = \begin{cases} \dfrac{d_1}{l^2}, & \text{when the coefficient is the own lag itself} \\ \dfrac{d_2\,\sigma_{\text{ii}}}{l^2\sigma_{\text{jj}}}, & \text{when the coefficients are the lags of different variables } i \neq j \end{cases}$$

Since, there is one lag for the variables, $l = 1$:

$$\Sigma_{i,\text{jj}} = \begin{cases} d_1, & \text{when the coefficient is the own lag itself} \\ \dfrac{d_2\,\sigma_{\text{ii}}}{\sigma_{\text{jj}}}, & \text{when the coefficients are the lags of different variables } i \neq j \end{cases}$$

```
% Hyperparameters in the prior distribution used in the BVAR model
```

```
prior_dist_hyper;
```

I have displayed the analytical posterior results for the first three priors - Diffuse, Minnesota, and Normal-Wishart. I implemented Markov Chain Monte Carlod simulations when using the Independent Normal-Wishart prior. In this case, the parameter values are contingent upon 50,000 Markov Chain Monte Carlo (MCMC) draws, out of which I have discarded the first 2000 draws. As impulse response functions are non-linear functions of the VAR coefficients, and Σ, I have simulated to get the posterior results from all four priors.

```
%========================= Sampling Begins Here ================================


% Begin the for loop for Gibbs Sampling

% repeat the iterations 15,000 times i.e. take 15,000 draws
for iter_rep = 1:num_total_draws



    % ---------  From Diffuse prior_dist, draw VARcoef and sigma_eps ---------
    if prior_dist == 1

        % Posterior of (alpha|sigma_eps,Data) ~ Normal
        % kronecker product of sigma_eps and inverse of (X'X)
        V_posterior = kron(sigma_eps,inv(X'*X));

        % Draw alpha

        % chol(V_posterior)': factorizes symmetric positive definite matrix
        % V_posterior into a lower triangle
        % randn(R*K, 1): creates a normally distributed vector of
        % R*K = 4 x 3 = 12 rows

        alpha = A_ols_vec + chol(V_posterior)'*randn(R*K,1);

        % Draw VARcoef

        % alpha is a (12 x 1) vector. reshape it into (R x K) vector
        VARcoef = reshape(alpha,R,K);

        % Draw sigma_eps

        % Posterior dist.:(sigma_eps|data) ~ inverse Wishart(sse_gibbs,T-K)
        % check 2.2.1 in pp 5 of the manual
        sigma_eps = inv(wishart(inv(sse_gibbs),T-R));



    % --------- From Minnesota prior_dist, draw VARcoef and sigma_eps ---------
    elseif prior_dist == 2
```

```
    % Here, we know the matrix sigma_eps
    % the prior_dist_hyper function contains the form of sigma_eps


      % Draw VARcoef

      % for all K = 3 number of factors
      for i = 1:K
          V_posterior = inv(inv(V_prior_dist((i-1)*R+1:i*R,(i-1)*R+1:i*R)) + ...
              inv(sigma_eps(i,i))*X'*X );

          a_posterior = V_posterior*(inv(V_prior_dist((i-1)*R+1:i*R, ...
              (i-1)*R+1:i*R))*a_prior_dist((i-1)*R+1:i*R,1) + ...
              inv(sigma_eps(i,i))*X'*F(:,i));

          % Draw alpha
          alpha((i-1)*R+1:i*R,1) = a_posterior + chol(V_posterior)'*randn(R,1);
      end

      % In terms of VARcoef, create a draw
      VARcoef = reshape(alpha,R,K);




  % --------- From the Normal-Wishart prior_dist, draw VARcoef and sigma_eps
  elseif prior_dist == 3


      % Relevent posterior params from pp 8 Koop and Korobilis (2009)


      V_posterior = inv( inv(V_prior_dist) + X'*X );
      A_posterior = V_posterior*(inv(V_prior_dist)*A_prior_dist + X'*X*A_ols);
      a_posterior = A_posterior(:);

      S_posterior = sse + S_prior_dist + A_ols'*X'*X*A_ols +...
      A_prior_dist'*inv(V_prior_dist)*A_prior_dist - ...
      A_posterior'*(inv(V_prior_dist) + X'*X)*A_posterior;

      v_posterior = T + v_prior_dist;

      % Covariance matrix of alpha's posterior dist. density
      cov_matrix = kron(sigma_eps,V_posterior);



      % Posterior dist.: (alpha|sigma_eps,data) ~ Normal

      % Draw alpha = vec(VARcoef)
      alpha = a_posterior + chol(cov_matrix)'*randn(R*K,1);

      % Draw the VAR coefficients
```

```
        VARcoef = reshape(alpha,R,K);

        % Posterior dist.: (sigma_eps|VARcoef,data) ~ iW(inv(S_posterior),v_posterior)
        % Draw sigma_eps of dim = 3 x 3
        % check eq (18) in pp 13
        sigma_eps = inv(wishart(inv(S_posterior),v_posterior));



        %--- From the Independent Normal-Wishart prior_dist, draw VARcoef and sigma_eps


    elseif prior_dist == 4
        variance = kron(inv(sigma_eps),eye(T));


        V_posterior = inv(V_prior_dist + G'*variance*G);  % below eq (19) in pp 13
        a_posterior = V_posterior*(V_prior_dist*a_prior_dist + G'*variance*F(:));

        % Draw alpha
        alpha = a_posterior + chol(V_posterior)'*randn(n,1);

        % Draw VARcoef
        VARcoef = reshape(alpha,R,K);

        % Posterior dist. of (sigma_eps|VARcoef,data) ~ iW(inv(S_posterior),v_posterior
        v_posterior = T + v_prior_dist;
        S_posterior = S_prior_dist + (F - X*VARcoef)'*(F - X*VARcoef);

        % Draw sigma_eps
        % below eq (20) in pp 13
        sigma_eps = inv(wishart(inv(S_posterior),v_posterior));

    end



    %================== Begin forecasting ======================

    if iter_rep > num_draw_discard



        if forecast==1

            % F_temp: create an matrix of 0s of dim (repeat-by-K)
            % later on, this F_temp will be filled with forecasts
                F_temp = zeros(repeat,K);

                % For each draw of sigma_eps and VAR coefficients,
                % caliberate 'repeat' forecasts

                for ii = 1:repeat
                    X_forecast = [1 F(T,:) X(T,2:K*(p-1)+1)];
```

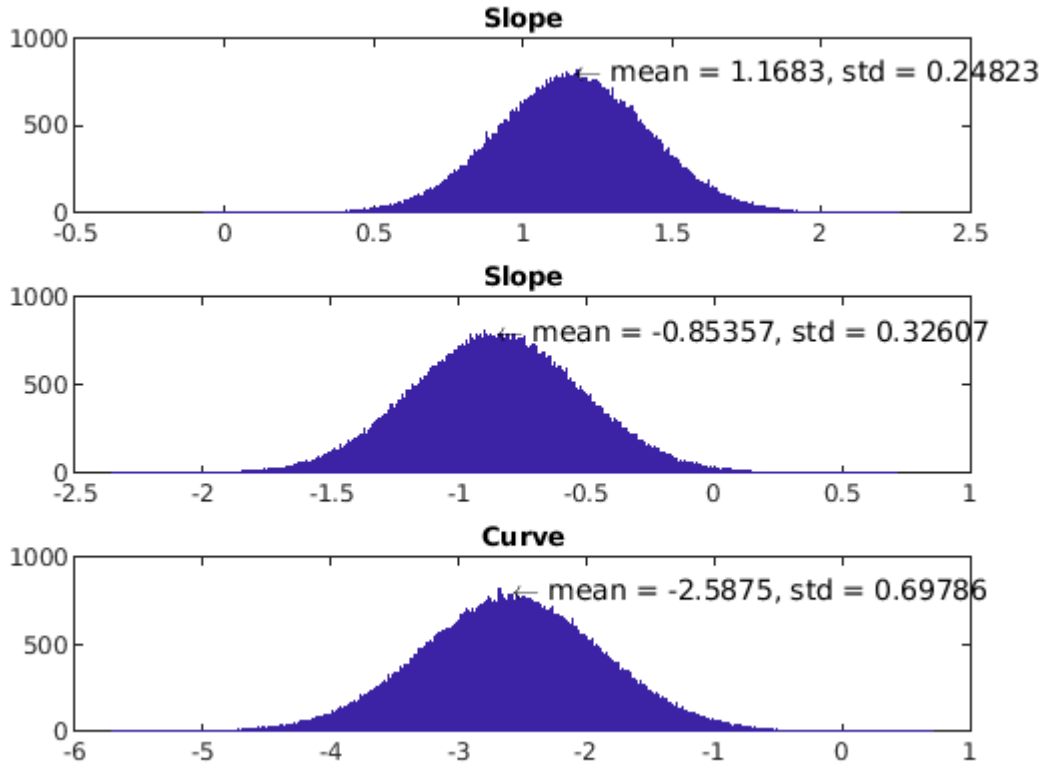
```
            % Forecast of T+1 conditional on data at time T
            F_temp(ii,:) = X_forecast*VARcoef + randn(1,K)*chol(sigma_eps);
        end

        % Forecasts stored in the matrix
        F_pred(((iter_rep-num_draw_discard)-1)*repeat+1:(iter_rep- ...
            num_draw_discard)*repeat,:) = F_temp;

        % Predictive likelihood
        pred_likhood(iter_rep-num_draw_discard,:) = mvnpdf(F1(T+1,:), ...
            X(T,:)*VARcoef,sigma_eps);

        if pred_likhood(iter_rep-num_draw_discard,:) == 0
            pred_likhood(iter_rep-num_draw_discard,:) = 1;
        end


% end forecasting
end


%============================ Impulse Response Functions ===============
if impulses==1

    % ------------------------ Identification code I ------------------

    Bv = zeros(K,K,p);
    for i_1=1:p
        Bv(:,:,i_1) = VARcoef(1+((i_1-1)*K + 1):i_1*K+1,:);
    end

    % Matrix of standard deviation for structural VAR
    shock = chol(sigma_eps)';
    d = diag(diag(shock));
    shock = inv(d)*shock;

    [resp]=impulse(Bv,shock,h_irf);

    % Restrict to policy shocks
    resp1 = squeeze(resp(:,1,:));
    resp2 = squeeze(resp(:,2,:));
    resp3 = squeeze(resp(:,3,:));

    imp_level(iter_rep-num_draw_discard,:,:) = resp1;
    imp_slope(iter_rep-num_draw_discard,:,:) = resp2;
    imp_curv(iter_rep-num_draw_discard,:,:) = resp3;



end

% ---------- Save the parameters' draws --------------------
alpha_draws(iter_rep-num_draw_discard,:) = alpha;
```

```
        VARcoef_draws(iter_rep-num_draw_discard,:,:) = VARcoef;
        sigma_draws(iter_rep-num_draw_discard,:,:) = sigma_eps;

    % stop saving results
    end

% end the main Gibbs for loop
end
```

```
%====================== End Sampling Posteriors ============================

%Posterior mean and variance/standard deviations of parameters:

% posterior mean of VARcoef
VARcoef_mean = squeeze(mean(VARcoef_draws,1));

% posterior mean of sigma_eps
sigma_mean = squeeze(mean(sigma_draws,1));

% Posterior standard deviations of parameters:

% posterior std of VARcoef
VARcoef_std = squeeze(std(VARcoef_draws,1));

% posterior std of sigma_eps
sigma_std = squeeze(std(sigma_draws,1));

% covariance matrix of the posterior of alpha (dim =[KM x KM] )
VARcoef_cov = cov(alpha_draws,1);
```

```
% average forecast and log predictive likelihood
if forecast == 1

    % mean forecast
    F_pred_mean = mean(F_pred,1);

    % standard deviation of the forecast
    F_pred_std = std(F_pred,1);
    log_pred_likhood = mean((log(pred_likhood)),1);

    % actual values of F at time T+h:

        true_value = F1(T+1,:);


    % ======= Plot the distribution of posterior predictive latent factors =====

    figure
    bars = 3000;
```

```
    subplot(3,1,1)
    hist(F_pred(:,1),bars);
    title('Slope')
    text(F_pred_mean(:,1),max(hist(F_pred(:,1),bars)),['\leftarrow mean = '...
        num2str(F_pred_mean(:,1)) ', std = ' num2str(F_pred_std(:,1))],...
        'HorizontalAlignment','left')

    subplot(3,1,2)
    hist(F_pred(:,2),bars);
    title('Slope')
    text(F_pred_mean(:,2),max(hist(F_pred(:,2),bars)),['\leftarrow mean = '...
        num2str(F_pred_mean(:,2)) ', std = ' num2str(F_pred_std(:,2))],...
        'HorizontalAlignment','left')

    subplot(3,1,3)
    hist(F_pred(:,3),bars);
    title('Curve')
    text(F_pred_mean(:,3),max(hist(F_pred(:,3),bars)),['\leftarrow mean = '...
        num2str(F_pred_mean(:,3)) ', std = ' num2str(F_pred_std(:,3))],...
        'HorizontalAlignment','left')
end
```



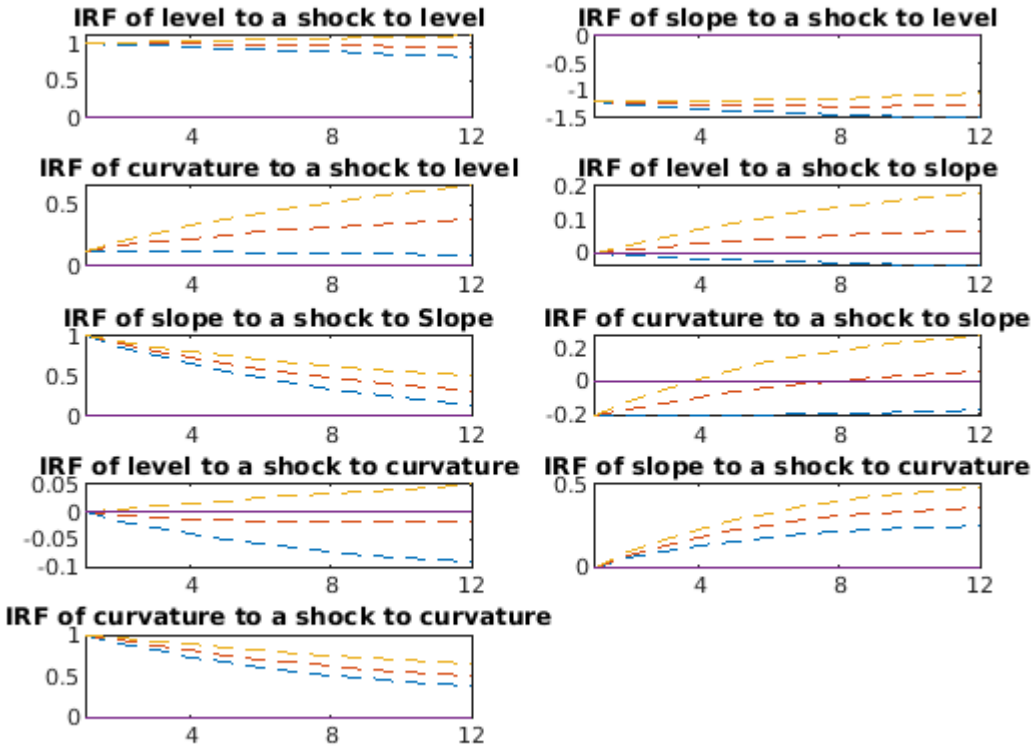

```
% Obtain the graphs of impulse response functions
if impulses==1;
```

```
% Affix quantiles from the posterior density of the impulse responses
qus = [.1, .5, .90];
imp_resp_infl = squeeze(quantile(imp_level,qus));
imp_resp_une = squeeze(quantile(imp_slope,qus));
imp_resp_int = squeeze(quantile(imp_curv,qus));



% ============== Plot the impulse response functions =====================


figure
set(0,'DefaultAxesColorOrder','factory',...
    'DefaultAxesLineStyleOrder','--|-|--')
subplot(5,2,1)
plot(squeeze(imp_resp_infl(:,1,:))')
hold;


% Shocks imposed on the level factor

plot(zeros(1,h_irf),'-')
title('IRF of level to a shock to level')
xlim([1 h_irf])
set(gca,'XTick',0:4:h_irf)
subplot(5,2,2)
plot(squeeze(imp_resp_une(:,1,:))')
hold;


plot(zeros(1,h_irf),'-')
title('IRF of slope to a shock to level')
xlim([1 h_irf])
set(gca,'XTick',0:4:h_irf)
subplot(5,2,3)
plot(squeeze(imp_resp_int(:,1,:))')
hold;


plot(zeros(1,h_irf),'-')
title('IRF of curvature to a shock to level')
xlim([1 h_irf])
set(gca,'XTick',0:4:h_irf)
subplot(5,2,4)
plot(squeeze(imp_resp_infl(:,2,:))')
hold;

% Shocks imposed on the slope factor

plot(zeros(1,h_irf),'-')
title('IRF of level to a shock to slope')
xlim([1 h_irf])
set(gca,'XTick',0:4:h_irf)
```

```
    subplot(5,2,5)
    plot(squeeze(imp_resp_une(:,2,:))')


    hold;
    plot(zeros(1,h_irf),'-')
    title('IRF of slope to a shock to Slope')
    xlim([1 h_irf])
    set(gca,'XTick',0:4:h_irf)
    subplot(5,2,6)
    plot(squeeze(imp_resp_int(:,2,:))')



    hold;
    plot(zeros(1,h_irf),'-')
    title('IRF of curvature to a shock to slope')
    xlim([1 h_irf])
    set(gca,'XTick',0:4:h_irf)
    subplot(5,2,7)
    plot(squeeze(imp_resp_infl(:,3,:))')


    % Shocks imposed on the slope factor

    hold;
    plot(zeros(1,h_irf),'-')
    title('IRF of level to a shock to curvature')
    xlim([1 h_irf])
    set(gca,'XTick',0:4:h_irf)
    subplot(5,2,8)
    plot(squeeze(imp_resp_une(:,3,:))')

    hold;
    plot(zeros(1,h_irf),'-')
    title('IRF of slope to a shock to curvature')
    xlim([1 h_irf])
    set(gca,'XTick',0:4:h_irf)
    subplot(5,2,9)
    plot(squeeze(imp_resp_int(:,3,:))')


    hold;
    plot(zeros(1,h_irf),'-')
    title('IRF of curvature to a shock to curvature')
    xlim([1 h_irf])
    set(gca,'XTick',0:4:h_irf)
end
```

```
Current plot held
Current plot held
Current plot held
Current plot held
Current plot held
Current plot held
```

```
Current plot held
Current plot held
Current plot held
```




```
% Mean Squared Forecast error

MSFE = (F_pred_mean - true_value).^2; MSFE = MSFE';


rowNames = {'Level','Slope','Curvature'};
colNames = {'MSFE'};
msfe_table = array2table(MSFE,'RowNames',rowNames, ...
    'VariableNames',colNames);

disp_mfse = {"The mean squared forecast errors for the latent factors are:";...
    "---------------------------------";...
    msfe_table};
cellfun(@disp,disp_mfse)
```

```
The mean squared forecast errors for the latent factors are:
---------------------------------
                MSFE
              ________

    Level     0.091137
    Slope     0.048578
    Curvature  0.30322
```

```
% Print important results

disp_VARcoef_mean = {"The mean matrix of the VAR regression coefficients is:";...
    "---------------------------------";...
    VARcoef_mean};
cellfun(@disp,disp_VARcoef_mean)
```

```
The mean matrix of the VAR regression coefficients is:
---------------------------------
    0.0081    0.1003   -0.2349
    0.9969   -0.0275    0.0309
    0.0099    0.9125    0.0246
   -0.0047    0.0632    0.9426
```

```
disp_VARcoef_variance = {"The variance matrix of the VAR regression coefficients is:";...
    "---------------------------------";...
    VARcoef_std};
cellfun(@disp,disp_VARcoef_variance)
```

```
The variance matrix of the VAR regression coefficients is:
---------------------------------
    0.0554    0.0733    0.1559
    0.0107    0.0141    0.0301
    0.0124    0.0164    0.0348
    0.0084    0.0110    0.0237
```

```
disp_sigma_mean = {"The mean of the VAR covariance matrix is:";...
    "---------------------------------";...
    sigma_mean};
cellfun(@disp,disp_sigma_mean)
```

```
The mean of the VAR covariance matrix is:
---------------------------------
    0.0602   -0.0609    0.0207
   -0.0609    0.1040   -0.0508
    0.0207   -0.0508    0.4756
```

```
disp_sigma_std = {"The variance of the VAR covariance matrix is:";...
    "---------------------------------";...
    sigma_std};
cellfun(@disp,disp_sigma_std)
```

```
The variance of the VAR covariance matrix is:
---------------------------------
   1.0e-13 *

    0.0164    0.0400    0.0091
    0.0400    0.0754    0.0061
    0.0091    0.0061    0.1277
```

```
disp_pred_mean = {"The predictive mean and standard deviation is:";...
    "-----------------------------------";...
    F_pred_mean};
cellfun(@disp,disp_pred_mean)
```

```
The predictive mean and standard deviation is:
----------------------------------
    1.1683   -0.8536   -2.5875
```

```
disp_pred_std = {"The predictive standard deviation is:";...
    "-----------------------------------";...
    F_pred_std};
cellfun(@disp,disp_pred_std)
```

```
The predictive standard deviation is:
----------------------------------
    0.2482    0.3261    0.6979
```

```
disp_pred_likhood = {"The log predictive likelihood is:";...
    "-----------------------------------";...
    log_pred_likhood};
cellfun(@disp,disp_pred_likhood)
```

```
The log predictive likelihood is:
----------------------------------
    0.4556
```

```
disp_true_value = {"The true value of y(t+h) is:";...
    "-----------------------------------";...
    true_value};
cellfun(@disp,disp_true_value)
```

```
The true value of y(t+h) is:
----------------------------------
    1.4701   -1.0740   -3.1382
```

```
if prior_dist == 1; name = 'diffuse';
elseif prior_dist == 2; name = 'minnesota';
elseif prior_dist == 3; name = 'normal_wishart' ;
elseif prior_dist == 4; name = 'indep_normal_wishart';

end
```

References

Diebold, F. X., & Li, C. (2006). Forecasting the term structure of government bond yields. *Journal of Econometrics, 130*(2), 337-364. doi:10.1016/j.jeconom.2005.03.005

Diebold, F. X., Rudebusch, G. D., & Aruoba, S. B. (2006). The macroeconomy and the yield curve: A dynamic latent factor approach. *Journal of Econometrics, 131*(1-2), 309-338. doi:10.1016/j.jeconom.2005.01.011

Diebold, Francis X., and Glenn D. Rudebusch. Yield Curve Modeling and Forecasting: The Dynamic Nelson-Siegel Approach. *Princeton University Press,* 2013.

https://www.mathworks.com/help/fininst/fitting-the-diebold-li-model_example-ex10300997.html

https://www.mathworks.com/help/econ/using-the-kalman-filter-to-estimate-and-forecast-the-diebold-li-model.html