clear all

close all

restoredefaultpath

addpath(genpath('jsz\_library'))

clc

load('sample\_RY\_model\_jsz.mat')

load('sample\_zeros.mat')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute the loadings and rotated model with the yields as states

[BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings(W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt);

yields\_m = ones(length(dates),1)\*AcP + (yields\*W.')\*BcP;

figure(1)

plot(year(dates) + month(dates)/12, yields\_m)

xlabel('date')

ylabel('yields')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% We can also re-parameterize kinfQ with rho0\_cP

% This is beneficial since it will help make the likelihood continuous.

[BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings\_rho0cP(W, K1Q\_X, rho0\_cP, Sigma\_cP, mats, dt);

% We could also use rinfQ (the long run mean of the short rate under Q) instead of kinfQ provided that the model is Q-stationary.

% This will make the likelihood continuous and provides a proper bijective identification for the set of Q-stationary models

[BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings(W, K1Q\_X, -K1Q\_X(1,1)\*rinfQ, Sigma\_cP, mats, dt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Can compute the likelihood with the "without error assumption"

llk = jszLLK(yields, W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt);

% Or all the bells and whistles

[llk, AcP, BcP, AX, BX, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ] = ...

jszLLK(yields, W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt);

fprintf('The average (negative) log likelihood is %6.6g\n', mean(llk))

% With the without error assumption, we can concentrate out kinf (which

% makes the choice of rinf vs rho\_cP vs kinf irrelevant)

[llk, AcP, BcP, AX, BX, kinf, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] = ...

jszLLK\_kinf\_conc(yields, W, K1Q\_X, Sigma\_cP, mats, dt);

fprintf('The average (negative) log likelihood is (concentrate) %6.6g\n', mean(llk))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The model was estimated assuming the 6-month, 2-year, 10-year zeros were

% measured without error. We can compute the likelihood with the KF error

% assumption. (this function requires [K0P\_cP, K1P\_cP, sigma\_e])

[llk, AcP, BcP, AX, BX, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, yields\_filtered, cP\_filtered] = ...

jszLLK\_KF(yields, W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt, K0P\_cP, K1P\_cP, sigma\_e);

fprintf('The average (negative) log likelihood is %6.6g when using KF instead of assuming yields without error at these estimates\n', mean(llk))

figure(2)

plot(year(dates) + month(dates)/12, [yields\*W', cP\_filtered])

xlabel('date')

ylabel('yields')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

close all

restoredefaultpath

addpath(genpath('jsz\_library'))

clc

% This gives a sample estimation of an N-factor "without error" GDTSM.

% See "A New Perspective on Gaussian Dynamic Term Structure Models" by Joslin, Singleton and Zhu

% Load some data: mats (1\*J) and yields (T\*J)

load('sample\_zeros.mat')

% Setup format of model/data:

N = 2;

W = pcacov(cov(yields));

W = W(:,1:N)'; % N\*J

cP = yields\*W'; % T\*N

dt = 1/12;

% Estimate the model by ML.

help sample\_estimation\_fun

VERBOSE = true;

[llks, AcP, BcP, AX, BX, kinfQ, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] = ...

sample\_estimation\_fun(W, yields, mats, dt, VERBOSE);

function [llks, AcP, BcP, AX, BX, kinfQ, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] =

sample\_estimation\_fun(W, yields, mats, dt, VERBOSE)

% function [llks, AcP, BcP, AX, BX, kinfQ, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] = ...

% sample\_estimation(W, yields, mats, dt, VERBOSE)

%

% Estimates the model with the following setup:

% 1. Optimize over (lamQ, Sigma\_cP).

% 2. lamQ is assumed to be real, parameterized by the difference to maintain order.

% 3. Sigma\_cP parameterized by cholesky factorization in optimization

% 4. Take randomized lamQ as initial seeds. See lines 107-108 for the randomization

% 5. always use OLS estimate of Sigma\_cP to start (see Joslin, Singleton, Le)

% 6. Generate 50 random seeds and take best to start (avoid really bad areas -- no point in looking here)

% 7. (kinfQ, sigma\_e, K0P, K1P) are all concentrated out of the likelihood function. See JSZ and JLS.

% 8. Run fmincon and repeat 3 times. Repeating re-sets the iteratively computed Hessian.

%

%

%

% INPUTS:

% W : N\*J, weights for the yield portfolios measured without error

% yields : T\*J, annualized zero coupon yields

% mats : 1\*J, maturities, in years

% dt : scalar, time in years for each period

% VERBOSE : boolean, true prints more output

%

% OUTPUTS:

% K0Q\_X : N\*1, normalized latent-model matrix

% K1Q\_X : N\*N, normalized latent-model matrix

% Sigma\_cP : N\*N, positive definite matrix that is the covariance of innovations to cP

% K0P\_cP : N\*1,

% K1P\_cP : N\*N,

% sigma\_e : scalar, standard error of yield observation errors (errors are i.i.d)

%

% Compute likelihood conditioned on first observation!

%

% llk : T\*1 time series of -log likelihoods (includes 2-pi constants)

% AcP : 1\*J yt = AcP' + BcP'\*Xt (yt is J\*1 vector)

% BcP : N\*J AcP, BcP satisfy internal consistency condition that AcP\*W' = 0, BcP\*W' = I\_N

% AX : 1\*J yt = AX' + BX'\*Xt

% BX : N\*J Xt is the 'jordan-normalized' latent state

%

%

% The model takes the form:

% r(t) = rho0\_cP + rho1\_cP'\*cPt

% = rinfQ + 1'\*Xt (Xt is the 'jordan-normalized' state

% = 1 period discount rate (annualized)

% Under Q:

% X(t+1) - X(t) = K0Q\_X + K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% cP(t+1) - cP(t) = K0Q\_cP + K1Q\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

% where Sigma\_X is chosen to match Sigma\_cP

% and K0Q\_X(m1) = kinfQ where m1 is the multiplicity of the highest eigenvalue (typically 1)

%

% Under P:

% cP(t+1) - cP(t) = K0P\_cP + K1P\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

%

% Model yields are given by:

% yt^m = AcP' + BcP'\*cPt (J\*1)

% And observed yields are given by:

% yt^o = yt^m + epsilon\_e(t)

% where V\*epsilon\_e~N(0,sigma\_e^2 I\_(J-N))

% and V is an (J-N)\*J matrix which projects onto the span orthogonal to the

% row span of W. This means errors are orthogonal to cPt and cPt^o = cPt^m.

%

warning off all

if ~exist('VERBOSE','var') || isempty(VERBOSE), VERBOSE = true; end

nSeeds = 50; % Number of random starting points. We want to avoid really bad starting values.

mlam = .95; % most negative eigenvalue is greater than -mlam

nRepeats = 3; % We run fmincon this many times in a row. This is useful to reset the iterative computation of the Hessian

[N,J] = size(W);

cP = yields\*W'; % T\*N

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setup the likelihood function that input a P\*1 vector of parameters.

% Parameterize eigenvalues in terms of the difference to maintain order.

% We parametrize Sigma\_cP in terms of cholesky factorization.

% Also modify the likelihood function to return a default value for weird parameter values with numerical issues

llk\_fun = @(dlamQ, cholSigma\_cP) llk\_fun0(yields, W, dlamQ, cholSigma\_cP, mats, dt);

% dlamQ : N\*1

% cholSigma\_cP : [N\*(N+1)/2]\*1 vector of subdiagonal element of cholesky factorization

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% SETUP INITIAL CONDITIONS:

% STARTING POINT FOR Sigma\_cP:

% Always initialize Sigma\_cP at the VAR estimate. This should be accurate, see Joslin, Le, and Singleton.

[Gamma\_hat, alpha\_hat, Omega\_hat] = regressVAR(cP);

Sigma\_cP0 = Omega\_hat;

L0 = chol(Sigma\_cP0, 'lower');

inds = find(tril(ones(N)));

cholSigma\_cP0 = L0(inds);

% STARTING POINT FOR lamQ:

% Generate some random seeds so we don't waste time searching with very weird parameters.

bestllk = inf;

for n=1:nSeeds

% To be sure the eigenvalues are ordered, we parameterize the difference in eigenvalues, dlamQ.

dlamQ(1,1) = .01\*randn; % When this is positive we'll have Q-non-stationary model

dlamQ(2:N,1) = -diff(sort([dlamQ(1); rand(N-1,1)]));

llk = llk\_fun(dlamQ, cholSigma\_cP0);

if llk<bestllk

if VERBOSE, fprintf('Improved seed llk to %5.5g\n',llk), end

bestllk = llk;

dlamQ0 = dlamQ;

end

end

X0 = [dlamQ0; cholSigma\_cP0]; % [N + N\*(N+1)/2]\*1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Let's constrain so the most negative eigenvalue (=sum(dlamQ)) is greater than -mlam

A = [-ones(1,N), zeros(1,N\*(N+1)/2)];

B = mlam;

Aeq = [];

Beq = [];

% Bounds for eigenvalues:

LB = [-2,-inf\*ones(1,N-1)];

UB = [.5,zeros(1,N-1)];

% Bounds for cholesky factorization of Sigma\_cP

A0 = ones(N);

inds\_diag = find(ismember(find(tril(A0)), find(diag(diag(A0))))) + N;

inds\_offdiag = find(~ismember(find(tril(A0)), find(diag(diag(A0))))) + N;

LB(inds\_diag) = 1e-7; % Avoid getting non-singular Sigma\_cP, should be positive to be identified

LB(inds\_offdiag) = -inf;

UB(N+1:N\*(N+1)/2+N) = inf;

options = optimset('display','off','TolX',1e-8,'TolFun',1e-8);

X = X0;

for i=1:nRepeats

[X, llk] = fmincon(@(Z) llk\_fun(Z(1:N), Z(N+1:end)),X,A,B,Aeq,Beq,LB,UB,[],options);

if VERBOSE

fprintf('Likelihood on step %d: %10.10g\tparameters:',i,llk)

fprintf('%3.3g\t',X)

fprintf('\n')

end

end

[llk, K1Q\_X, Sigma\_cP] = llk\_fun0(yields, W, X(1:N), X(N+1:end), mats, dt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[llks, AcP, BcP, AX, BX, kinfQ, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] = ...

jszLLK\_kinf\_conc(yields, W, K1Q\_X, Sigma\_cP, mats, dt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Likelihoood function. Very extreme parameters may have numerical

% problems since some intermediate matrices may be nearly non-singular. In

% this case set the likelihood to a "bad" default value.

function [llk, K1Q\_X, Sigma\_cP] = llk\_fun0(yields, W, dlamQ, cholSigma\_cP, mats, dt)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Extract the vector parameters:

N = length(dlamQ);

K1Q\_X = diag(cumsum(dlamQ));

inds = find(tril(ones(N)));

L(inds) = cholSigma\_cP;

L = reshape(L, [N,N]);

Sigma\_cP = L\*L';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

default\_llk = 1000;

try

llk = mean(jszLLK\_kinf\_conc(yields, W, K1Q\_X, Sigma\_cP, mats, dt));

if isnan(llk) || ~isreal(llk) || ~isfinite(llk)

llk = default\_llk;

end

catch

llk = default\_llk;

end

if llk<-100

[llks, AcP, BcP, AX, BX, kinfQ, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] = ...

jszLLK\_kinf\_conc(yields, W, K1Q\_X, Sigma\_cP, mats, dt);

keyboard

end

function [Minf, Vinf] = asymptoticMomentsGaussian(K0Pd, K1Pd, H0d)

% function [Minf, Vinf] = asymptoticMomentsGaussian(K0Pd, K1Pd, H0d)

%

% X(t+1) - X(t) = K0Pd + K1Pd\*X(t) + eps(t+1), cov(eps(t)) = H0d

%

% Compute the stationary distribution of X is N(Minf,Vinf)

% (Assuming negative real parts of the eigenvalues of K1Pd)

%

% K0Pd : N\*1

% K1Pd : N\*N

% H0d : N\*N

%

% Minf : N\*1

% Vinf : N\*N

N = length(K0Pd);

Minf = - K1Pd\K0Pd;

A = K1Pd + eye(N);

% vec(A\*B\*C) = kron(C'\*A)\*vec(B)

% Vinf = A\*Vinf\*A' + H0d

Vinf = reshape( (eye(N^2) - kron(A, A))\H0d(:),[N,N]);

function [By,Ay, dAyH0d] = gaussianDiscreteYieldLoadingsDiagonal(maturities, K0d, K1d\_diag, H0d, rho0d, rho1d, timestep)

%function [By,Ay, dAyH0d] = gaussianDiscreteYieldLoadingsDiagonal(maturities, K0d, K1d\_diag, H0d, rho0d, rho1d, timestep)

%

% DOESN'T HANDLE UNIT ROOTS!!

%

% THIS FUNCTION ASSUMES K1d is diagonal

% K0d : N\*1

% K1d\_diag : N\*1

% H0d : N\*N

% rho0d : scalar

% rho1d : N\*1

% timestep : optional argument.

%

% By : N\*M

% Ay : 1\*M (faster to not compute with only one output argument)

%

% r(t) = rho0d + rho1d'Xt

% = 1 period discount rate

% P(t) = price of t-period zero coupon bond

% = EQ0[exp(-r0 - r1 - ... - r(t-1)]

% = exp(A+B'X0)

% yields = Ay + By'\*X0

% yield is express on a per period basis unless timestep is provided.

% --For example, if the price of a two-year zero is exp(-2\*.06)=exp(-24\*.005),

% --and we have a monthly model, the function will return Ay+By\*X0=.005

% --unless timestep=1/12 is provided in which case it returns Ay+By\*X0=.06

%

% Where under Q:

% X(t+1) - X(t) = K0d + K1d\*X(t) + eps(t+1), cov(eps(t+1)) = H0d

%

% We can compute the loadings by recurrence relations:

% A1 = -rho0d

% B1 = -rho1d

% At = A(t-1) + K0d'\*B(t-1) .5\*B(t-1)'\*H0d\*B(t-1) - rho0d

% Bt = B(t-1) + K1d'\*B(t-1) - rho1d

%

% Or in closed form by noting that

% r0+r1+..+r(t-1) = c.X(0) + alpha0 + alpha1\*eps1 + ... + alpha(t-1)\*eps(t-1)

% ~ N(c.X(0) + alpha0, alpha1'H0d\*alph1 + ... + alpha(t-1)'\*H0d\*alpha(t-1))

%

% And then use the MGF of Y~N(mu,Sigma) is E[exp(a.Y)] = a'\*mu + .5\*a'\*Sigma\*a

% (or similarly use the partial geometric sum formulas repeatedly)

%

% Let G = K1+I

% X(0)

% X(1) = K0 + G\*X(0) + eps1

% X(2) = K0 + G\*K0 + G^2\*X(0) + G\*eps1 + eps2

% X(3) = K0 + G\*K0 + G^2\*K0 + G^3\*X(0) + G^2\*eps1 + G\*eps2 + eps3

% X(n) = sum(I+G+..+G^(n-1))\*K0 + G^n\*X(0) + sum(i=1..n,G^(n-i)\*epsi)

% = (I-G\(I-G^n)\*K0 + G^n\*X(0) + sum(i=1..n,G^(n-i)\*epsi)

%

% cov(G^n\*eps) = G^n\*cov(eps)\*(G^n)'

% vec(cov(G^n\*eps) = kron(G^n,G^n)\*vec(eps)

% = (kron(G,G)^n)\*vec(eps)

%

% sum(X(i),i=1..n) = mu0 + mu1\*X0 + u

% mu0 = (I-G)\(I - (I-G)\(G-G^(n+1))\*K0

% mu1 = (I-G)\(G-G^(n+1))

% vec(cov(u)) = see below.

% u = (I-G)\(I-G^n)\*eps1 +

% (I-G)\(I-G^(n-1))\*eps2 + ..

% (I-G)\(I-G)\*epsn

% cov(u) = (I-G)\Sig/(I-G)'

% Sig = sum(cov(eps)) + sum(i=1..n,G^i\*cov(eps)) +

% sum(i=1..n,cov(eps)G^i') + sum(i=1..n,G^i\*cov(eps)\*G^i')

% compute the last one using vec's. see below.

% K1d\_diag is N\*1 -- the diagonal of K1d

M = length(maturities);

N = length(K0d);

Ay = zeros(1,M);

By = zeros(N, M);

dAyH0d = zeros(N,N,M);

I = ones(N,1);

G = K1d\_diag+ones(N,1); % N\*1

if nargout>1

GG = G\*G'; % N\*N

GpG = G\*ones(1,N) + ones(N,1)\*G'; % N\*N (i,j) entry is G(i)+G(j)

end

for m=1:M

mat = maturities(m);

if mat==1

By(:,m) = rho1d;

Ay(:,m) = rho0d;

mu0 = zeros(N,1);

mu1 = ones(N,1);

Sigma0 = zeros(N);

continue

end

i = mat-1; % # of random innovations X(0) + X(1) + ... + X(mat)=X(0)+...+X(i)

% X(0) + ... + X(i)~N(mu0 + mu1\*X(0), Sigma0)

mu1 = I - K1d\_diag.\(G - G.^(i+1)); % N\*1

mu0 = -K1d\_diag.\((i+1)\*I - mu1).\*K0d;

By(:,m) = rho1d.\*mu1/mat;

if nargout>1

Sigma\_term1 = i\*H0d; % N\*N

Sigma\_term2 = ((mu1 - 1)\*ones(1,N)).\*H0d; % N\*N

Sigma\_term3 = Sigma\_term2.';

Sigma\_term4 = (1 - GG(:)).\(GG(:) - GG(:).^(i+1)).\*H0d(:);

Sigma\_term4 = reshape(Sigma\_term4, [N,N]);

Sigma0 = (K1d\_diag\*K1d\_diag.').\(Sigma\_term1 - Sigma\_term2 - Sigma\_term3 + Sigma\_term4);

Ay(:,m) = rho0d + (rho1d.'\*mu0 - .5\*rho1d.'\*Sigma0\*rho1d)/mat;

end

if nargout>2

dAyH0d(:,:,m) = -.5\*i\*rho1d\*rho1d' ...

+.5\*((mu1-1)\*ones(1,N)).\*(rho1d\*rho1d.') ...

+.5\*(ones(N,1)\*(mu1-1).').\*(rho1d\*rho1d.') ...

- .5\*reshape((1 - GG(:)).\(GG(:) - GG(:).^(i+1)), [N,N]).\*(rho1d\*rho1d.');

dAyH0d(:,:,m) = (K1d\_diag\*K1d\_diag.').\dAyH0d(:,:,m)/mat;

end

end

if nargin==7

By = By/timestep;

if nargout>1

Ay = Ay/timestep;

if nargout>2

dAyH0d = dAyH0d/timestep;

end

end

end

function [By, Ay] = gaussianDiscreteYieldLoadingsRecurrence(maturities, K0d, K1d, H0d, rho0d, rho1d, timestep)

% function [By, Ay] = gaussianDiscreteYieldLoadingsRecurrence(maturities, K0d, K1d, H0d, rho0d, rho1d, timestep)

%

% K0d : N\*1

% K1d : N\*1

% H0d : N\*N

% rho0d : scalar

% rho1d : N\*1

% timestep : optional argument.

% By : N\*M

% Ay : 1\*M (faster to not compute with only one output argument)

%

% r(t) = rho0d + rho1d'Xt

% = 1 period discount rate

% P(t) = price of t-period zero coupon bond

% = EQ0[exp(-r0 - r1 - ... - r(t-1)]

% = exp(A+B'X0)

% yields = Ay + By'\*X0

% yield is express on a per period basis unless timestep is provided.

% --For example, if the price of a two-year zero is exp(-2\*.06)=exp(-24\*.005),

% --and we have a monthly model, the function will return Ay+By\*X0=.005

% --unless timestep=1/12 is provided in which case it returns Ay+By\*X0=.06

%

% Where under Q:

% X(t+1) - X(t) = K0d + K1d\*X(t) + eps(t+1), cov(eps(t+1)) = H0d

%

% A1 = -rho0d

% B1 = -rho1d

% At = A(t-1) + K0d'\*B(t-1) .5\*B(t-1)'\*H0d\*B(t-1) - rho0d

% Bt = B(t-1) + K1d'\*B(t-1) - rho1d

%

% mautirities: 1\*M # of periods

M = length(maturities);

N = length(K0d);

Atemp = 0;

Btemp = zeros(N,1);

A = nan(1,M);

B = nan(N,M);

curr\_mat = 1;

for i=1:maturities(M)

Atemp = Atemp + K0d'\*Btemp +.5\*Btemp'\*H0d\*Btemp - rho0d;

Btemp = Btemp + K1d'\*Btemp - rho1d;

if i==maturities(curr\_mat)

Ay(1,curr\_mat) = -Atemp/maturities(curr\_mat);

By(:,curr\_mat) = -Btemp/maturities(curr\_mat);

curr\_mat = curr\_mat + 1;

end

end

if nargin==7

Ay = Ay/timestep;

By = By/timestep;

end

function [x\_tm1t, P\_tm1t, x\_tt, P\_tt, K\_t, llks] = kf(y, Phi, alpha, A, b, Q, R, x00, P00)

% function [x\_tm1t, P\_tm1t, x\_tt, P\_tt, K\_t, llks] = kf(y, Phi, alpha, A, b, Q, R, x00, P00)

%

% Notation is as in Time Series Analysis and Its Applications by Shumway and Stoffer

% MODIFIED TO HAVE INTERCEPTS

%

% x\_t = Phi\*x\_{t-1} + alpha + w\_t, cov(w)=Q, x is p-dimensional

% y\_t = A\*x\_t + b + v\_t, cov(v)=R, y is q-dimensional

%

% y : q\*n

% Phi : p\*1

% alpha : p\*1

% A : q\*p

% b : q\*1

% Q : p\*p

% R : q\*q

% x00 : p\*1 x0 is assued to be normal N(x00,P00)

% P00 : p\*p

%

% x\_tm1t : p\*n

% P\_tm1t : p\*p\*n

% x\_tt : p\*n

% P\_tt : p\*p\*n

% K\_t : p\*q\*n

% llks : n\*1

%

% P\_tm1t(tau) = P\_{tau}^{tau-1}, tau=1,..,n

% x\_tm1t(tau) = x\_{tau}^{tau-1}, tau=1,..,n

% P\_tt(tau) = P\_{tau}^{tau}, tau=1,..,n

% x\_tt(tau) = x\_{tau}^{tau}, tau=1,..,n

% K\_t(tau) = K\_{tau}, tau=1,..,n Kalman Gain

% llks(t) = log(likelihood(y(t)|y(1),...y(t-1))) (includes 2\*pi terms).

% it is likelihood not minus log likelihood

p = size(Q,1);

[q,n] = size(y);

P\_tm1t = nan(p,p,n);

x\_tm1t = nan(p,n);

P\_tt = nan(p,p,n);

x\_tt = nan(p,n);

K\_t = nan(p,q,n);

llks = nan(n,1);

% Initialize the recursion and do first step:

Ip = eye(p);

x\_tm1t(:,1) = Phi\*x00 + alpha; % 4.35

P\_tm1t(:,:,1) = Phi\*P00\*Phi.' + Q; % 4.36

for t=1:n

if t==1

x\_tm1t(:,1) = Phi\*x00 + alpha; % 4.35

P\_tm1t(:,:,1) = Phi\*P00\*Phi.' + Q; % 4.36

else

x\_tm1t(:,t) = Phi\*x\_tt(:,t-1) + alpha; % 4.35

P\_tm1t(:,:,t) = Phi\*P\_tt(:,:,t-1)\*Phi.' + Q; % 4.36

end

epst = (y(:,t) - (A\*x\_tm1t(:,t) + b)); % 4.40 (modified)

Sigmat = A\*P\_tm1t(:,:,t)\*A.' + R; % 4.41;

K\_t(:,:,t) = P\_tm1t(:,:,t)\*A.'/Sigmat; % 4.39

x\_tt(:,t) = x\_tm1t(:,t) + K\_t(:,:,t)\*epst; % 4.37

P\_tt(:,:,t) = (Ip - K\_t(:,:,t)\*A)\*P\_tm1t(:,:,t); % 4.38

% NOTE: For strange parameters, we might have non-psd Sigmat which would be a problem

term2 = log(det(Sigmat));

term3 = max(epst.'\*(Sigmat\epst),0);

% if ~isreal(term3) || ~isreal(term2), keyboard, end

llks(t) = -q/2\*log(2\*pi) - .5\*term2 -.5\*term3; % as in 4.67

end

function [beta, alpha, Omega, log\_llk] = reducedRankFreeInterceptRegress(Y, X, r, Omega)

% function [beta, alpha, Omega, log\_llk] = reducedRankFreeInterceptRegress(Y, X, r, Omega)

%

% Y : T\*N

% X : T\*M

% r : scalar, rankd of beta

% Omega : N\*N

%

% beta : M\*N

% alpha : 1\*N

% Omega : N\*N

% log\_llk : T\*1 minus log of the likelihood (including 2pi's)

%

% Regress Y(t,:) = X(t,:)\*beta + alpha + eps(t), cov(eps) = Omega

%

% under the assumption that beta has rank r, and alpha is unconstrained.

% Note Letting Yhat(t,:) = Y(t,:) - Ybar

% Xhat(t,:) = X(t,:) - Xbat

% for any Omega, beta, we can concentrate out alpha by rewriting the

% likelihood as

% -N/2\*log(2\*pi) - .5\*log(det(Omega))

% - .5\*sum((Yhat(t,:) - Xhat(t,:)\*beta)\*Omega^-1\*(Yhat(t,:) - Xhat(t,:)\*beta)

% - T/2\*(alpha - Ybar - beta'\*Xbar).'(alpha - Ybar - beta'\*Xbar)

% (cross terms cancel)

%

% The MLE of alpha is then alpha = Ybar = beta'\*Xbar

%

% If Omega is not provided, likelihood is maximized over beta and Omega

%

[T N] = size(Y);

M = size(X,2);

Y0 = Y; % T\*N

X0 = X; % T\*M

Ybar = mean(Y); % 1\*N

Xbar = mean(X); % 1\*M

Y = Y - ones(T,1)\*Ybar;

X = X - ones(T,1)\*Xbar;

%M = size(X,2);

if nargin==2

r = N;

end

if r==N

beta = (X'\*X)\(X'\*Y); % M\*N

else

if nargin<4

M00 = 1/(T) \* Y'\*Y;

M0k = 1/(T) \* Y'\*X;

Mkk = 1/(T) \* X'\*X;

[V D] = eig(M0k'\*(M00\M0k),Mkk);

[A B] = sort(diag(D),'descend');

b = V(:,B(1:r));

% b = b/b(1:r,1:r);

a = M0k\*b/(b'\*Mkk\*b);

beta = (a\*b')';

else

P = chol(X'\*X);

L = chol(Omega);

betaols = (X'\*X)\(X'\*Y);

[U S V] = svd(P\*betaols/L);

S(r+1:end,r+1:end) = 0;

beta = P\U\*S\*V'\*L;

end

end

if nargout>2 && nargin<4

Omega = 1/T\*(Y-X\*beta).'\*(Y-X\*beta);

end

if nargout>3

log\_llk = +N/2\*log(2\*pi) + .5\*log(det(Omega)) + (sum((Y-X\*beta).'.\*(Omega\(Y-X\*beta).'),1)).'; % T\*1

end

alpha = Ybar - Xbar\*beta;

function [beta, Omega, log\_llk] = reducedRankRegress(Y, X, r, Omega)

%function [beta, Omega, log\_llk] = reducedRankRegress(Y, X, r, Omega)

%

% Y : T\*N

% X : T\*M

% r : scalar, rankd of beta

% Omega : N\*N

%

% beta : M\*N

% Omega : N\*N

% log\_llk : T\*1 minus log of the likelihood (including 2pi's)

%

% Regress Y = X\*beta + eps, cov(eps) = Omega

%

% under the assumption that beta has rank r

% If Omega is not provided, likelihood is maximized over beta and Omega

%

[T N] = size(Y);

%M = size(X,2);

if nargin==2

r = N;

end

if r==N

beta = (X'\*X)\(X'\*Y); % M\*N

else

if nargin<4 || isempty(Omega)

M00 = 1/(T) \* Y'\*Y;

M0k = 1/(T) \* Y'\*X;

Mkk = 1/(T) \* X'\*X;

[V D] = eig(M0k'\*(M00\M0k),Mkk);

[A B] = sort(diag(D),'descend');

b = V(:,B(1:r));

% b = b/b(1:r,1:r);

a = M0k\*b/(b'\*Mkk\*b);

beta = (a\*b')';

else

P = chol(X'\*X);

try

L = chol(Omega);

catch

[U0 D0 V0] = svd(Omega);

L = U0\*sqrt(D0);

end

betaols = (X'\*X)\(X'\*Y);

[U S V] = svd(P\*betaols/L);

S(r+1:end,r+1:end) = 0;

beta = P\U\*S\*V'\*L;

end

end

if nargout>1 && (nargin<4 || isempty(Omega))

%disp('compute omega')

Omega = 1/T\*(Y-X\*beta).'\*(Y-X\*beta);

end

if nargout>2

log\_llk = +N/2\*log(2\*pi) + .5\*log(det(Omega)) + .5\*(sum((Y-X\*beta).'.\*(Omega\(Y-X\*beta).'),1)).'; % T\*1

end

function [Gamma\_hat, alpha\_hat, Omega\_hat] = regressVAR(X)

% function [Gamma\_hat, alpha\_hat, Omega\_hat] = regressVAR(X)

% X: T\*N

%

% Gamma\_hat : N\*N

% alpha\_hat : N\*1

% Omega\_hat : N\*N

%

% X(t+1) = alpha + Gamma\*X(t) + eps(t+1), cov(eps(t+1)) = Omega

%

% Compute the maximum likelihood estimates of Gamma, alpha, Omega

%

% NOTE: The MLE estimates of Gamma, alpha do not depend on Omega.

% That is, the argmax\_{Gamma,alpha} [L(X|Gamma,alpha,Omega)] = f(X)

% So this function provides MLE of Gamma, alpha for a fixed Omega.

[T,N] = size(X);

Yt = X(1:end-1,:); % (T-1)\*N

Ytp1 = X(2:end,:); % (T-1)\*N

Y = Ytp1.'; % N\*(T-1)

Z = [ones(T-1,1), Yt].'; % (N+1)\*(T-1)

A = Y\*Z.'\*inv(Z\*Z.'); % N\*(N+1)

alpha\_hat = A(:,1);

Gamma\_hat = A(:,2:end);

if nargout==3

residuals = Ytp1 - (A\*Z).'; % (T-1)\*N

Omega\_hat = 1/(T-1)\*residuals.'\*residuals;

end

function [K1Q\_X, isTypicalDiagonal, m1] = jszAdjustK1QX(K1Q\_X, eps1)

% function [K1Q\_X, isTypicalDiagonal, m1] = jszAdjustK1QX(K1Q\_X, eps1);

%

% This function adjusts diagonal K1Q\_X to give a non-diagonal but more

% computationally tractable K1Q\_X.

%

%

% K1Q\_X can fall into a few cases:

% 0. diagonal

% 1. not diagonal

% 2. zero eigenvalue

% 3. near repeated roots

% In cases 1-3, the diagonal closed form solver doesn't work, so compute differently.

% In case 1-2, we will use the recursive solver, though there are more efficient methods.

% In case 3, we will add a positive number above the diagonal. this gives a different member of the set of observationally equivalent models.

% So for example:

% [lambda1, 0; 0, lambda2] is replaced by [lambda1, f(lambda2-lambda1); 0, lambda2] when abs(lambda1 - lambda2)<eps0

% By making f not just 0/1, it will help by making the likelihood

% continuous if we parameterize by kinf. (not an issue in some cases.)

%

% We also order the diagonal of diagonal K1Q.

%

if nargin==1,

eps1 = 1e-3;

end

diag\_K1Q\_X = diag(K1Q\_X);

isDiagonal = all(all((K1Q\_X==diag(diag\_K1Q\_X))));

% For diagonal matrix, sort the diagonal and check to see if we have near repeated roots.

if isDiagonal

diag\_K1Q\_X = -sort(-diag\_K1Q\_X);

K1Q\_X = diag(diag\_K1Q\_X);

hasNearUnitRoot = ~all(abs(diag\_K1Q\_X)>eps1); % Only applicable for diagonal

hasNearRepeatedRoot = ~all(abs(diff(diag\_K1Q\_X))>eps1); % Only applicable for diagonal

isTypicalDiagonal = isDiagonal & ~hasNearRepeatedRoot & ~hasNearUnitRoot;

else

isTypicalDiagonal = false;

end

% If we have a near repeated root, add a constnat above the diagonal. This

% representative of the equivalence class gives easier inversion for latent

% states vs. yields. By varying the constant

if isDiagonal && ~isTypicalDiagonal

diff\_diag = abs(diff(diag\_K1Q\_X));

super\_diag = cutoff\_fun(diff\_diag);

K1Q\_X(1:end-1,2:end) = K1Q\_X(1:end-1,2:end) + diag(super\_diag);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N = size(K1Q\_X,1);

super\_diagonal = K1Q\_X(N+1:N+1:end);

m1 = max(find(cumprod([1,super\_diagonal])));

% Cutoff function sets the super diagonal entry to something between 0 and

% 1, depending on how close the eigenvalues are.

function xc = cutoff\_fun(x, eps1)

eps1 = 1e-3;

eps0 = 1e-5;

xc = 1.\*(x<eps0) + ...

(1 - (x - eps0)/(eps1 - eps0)).\*(x>=eps0 & x<eps1) + ...

0.\*(x>eps1);

xc = 1.\*(log(x)<log(eps0)) + ...

(1 - (log(x) - log(eps0))/(log(eps1) - log(eps0))).\*(log(x)>=log(eps0) & log(x)<log(eps1)) + ...

0.\*(log(x)>log(eps1));

xc(x==0) = 1;

function [llk, AcP, BcP, AX, BX, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ] = ...

jszLLK(yields\_o, W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt, K0P\_cP, K1P\_cP, sigma\_e, rankRP)

%function [llk, AcP, BcP, AX, BX, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ] = ...

% jszLLK(yields\_o, W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt, K0P\_cP, K1P\_cP, sigma\_e, rankRP)

%

%

% This function computest the likelihood for a Gaussian term structure.

% See "A New Perspective on Gaussian Dynamic Term Structure Models" by Joslin, Singleton and Zhu

%

% INPUTS:

% yields\_o : (T+1)\*J, matrix of observed yields (first row are t=0 observations, which likelihood conditions on)

% mats : 1\*J, maturities in years

% dt : scalar, length of period in years

%

% W : N\*J, vector of portfolio weights to fit without error.

% K1Q\_X : N\*N, normalized latent-model matrix (does not have to be diagonal, see form below)

% kinfQ : scalar, when the model is stationary, the long run mean of the annualized short rate under Q is -kinfQ/K1(m1,m1)

% Sigma\_cP : N\*N, positive definite matrix that is the covariance of innovations to cP

%

% OPTIONAL INPUTS:

% K0P\_cP : N\*1, OPTIONAL (supply [] to omit)

% K1P\_cP : N\*N, OPTIONAL (supply [] to omit)

% sigma\_e : scalar, standard error of yield observation errors

%

%

% Compute likelihood conditioned on first observation!

%

% llk : T\*1 time series of -log likelihoods (includes 2-pi constants)

% AcP : 1\*J yt = AcP' + BcP'\*Xt (yt is J\*1 vector)

% BcP : N\*J AcP, BcP satisfy internal consistency condition that AcP\*W' = 0, BcP\*W' = I\_N

% AX : 1\*J yt = AX' + BX'\*Xt

% BX : N\*J Xt is the 'jordan-normalized' latent state

%

%

% The model takes the form:

% r(t) = rho0\_cP + rho1\_cP'\*cPt

% = 1'\*Xt (Xt is the 'jordan-normalized' state

% = 1 period discount rate (annualized)

%

% Under Q:

% X(t+1) - X(t) = K0Q\_X + K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% cP(t+1) - cP(t) = K0Q\_cP + K1Q\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

% where Sigma\_X is chosen to match Sigma\_cP

% and K0Q\_X(m1) = kinfQ where m1 is the multiplicity of the highest eigenvalue (typically 1)

% Under P:

% cP(t+1) - cP(t) = K0P\_cP + K1P\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

%

% Model yields are given by:

% yt^m = AcP' + BcP'\*cPt (J\*1)

% And observed yields are given by:

% yt^o = yt^m + epsilon\_e(t)

% where V\*epsilon\_e~N(0,sigma\_e^2 I\_(J-N))

% and V is an (J-N)\*J matrix which projects onto the span orthogonal to the

% row span of W (i.e. V = null(W)'). This means errors are orthogonal to cPt and cPt^o = cPt^m.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setup

[T,J] = size(yields\_o(2:end,:));

% Setup W if we are using individual yields without error:

N = size(W,1);

cP = yields\_o\*W'; % (T+1)\*N, cP stands for math caligraphic P.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setup default parameters:

concentrateSigmaE = true;

concentrateK0PK1P = true;

if exist('K0P\_cP','var') && ~isempty(K0P\_cP), concentrateK0PK1P = false; end

if ~exist('rankRP','var') || isempty(rankRP), rankRP = N; end

if exist('sigma\_e','var') && ~isempty(sigma\_e), concentrateSigmaE = false; end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% COMPUTE THE Q-LIKELIHOOD:

% First find the loadings for the model:

% yt = AcP' + BcP'\*cPt, AcP is 1\*J, BcP is N\*J

[BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings(W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt);

yields\_m = ones(T+1,1)\*AcP + cP\*BcP; % (T+1)\*J, model-implied yields

yield\_errors = yields\_o(2:end,:) - yields\_m(2:end,:); % T\*J

square\_orthogonal\_yield\_errors = (yield\_errors).^2; % T\*J, but N-dimensional projection onto W is always 0, so effectively (J-N) dimensional

% Compute optimal sigma\_e if it is not supplied

if concentrateSigmaE

sigma\_e = sqrt( sum(square\_orthogonal\_yield\_errors(:))/(T\*(J-N)) );

end

llkQ = .5\*sum(square\_orthogonal\_yield\_errors.')/sigma\_e^2 + (J-N)\*.5\*log(2\*pi) + .5\*(J-N)\*log(sigma\_e^2); % 1\*T

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% COMPUTE THE P-LIKELIHOOD:

% We have three case: (1) unconstrained, (2) reduced rank risk premia and (3) kalman filter

if concentrateK0PK1P

if rankRP==N

% Run OLS to obtain maximum likelihood estimates of K0P, K1P

[K1PplusI, K0P\_cP] = regressVAR(cP);

K1P\_cP = K1PplusI - eye(N);

else

% Run reduced rank regression to compute ml estimates of K0P, K1P

Y1 = diff(cP) - cP(1:end-1,:)\*K1Q\_cP.';

X = cP(1:end-1,:);

[beta1, alpha1] = reducedRankFreeInterceptRegress(Y1, X, rankRP, Sigma\_cP);

K1P\_cP = beta1.' + K1Q\_cP;

K0P\_cP = alpha1.';

% THIS WOULD DO REDUCED RANK of [K0P, K1P] - [K0Q, K1Q] (only rankRP are priced instead of only rankRP have time-varying price)

% Y1 = diff(cP) - cP(1:end-1,:)\*K1Q\_cP.' - ones(T,1)\*K0Q\_cP.';

% X = [ones(T,1), cP(1:end-1,:)];

% K1P\_cP = beta1(2:end,:).' + K1Q\_cP;

% K0P\_cP = beta1(1,:).' + K0Q\_cP;

end

end

innovations = cP(2:end,:).' - (K0P\_cP\*ones(1,T) + (eye(N)+K1P\_cP)\*cP(1:end-1,:).'); % N\*T

llkP = .5\*N\*log(2\*pi) + .5\*log(det(Sigma\_cP)) + .5\*sum(innovations.\*(Sigma\_cP\innovations),1); % 1\*T

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

llk = (llkQ + llkP).'; % T\*1 series

function [llk, AcP, BcP, AX, BX, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, yields\_filtered, cP\_filtered] = ...

jszLLK\_KF(yields\_o, W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt, K0P\_cP, K1P\_cP, sigma\_e)

% function [llk, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP] = llkJSZ(yields\_o, mats, dt, W, yields\_woe, dt, K0P\_cP, K1P\_cP, Sigma\_cP, K1Q\_X, kinfQ, sigma\_e)

%

% This function computest the likelihood for a Gaussian term structure.

% See "A New Perspective on Gaussian Dynamic Term Structure Models" by Joslin, Singleton and Zhu

%

%

% INPUTS:

% yields\_o : T\*J, matrix of observed yields (first row are t=0 observations, which likelihood conditions on)

% mats : 1\*J, maturities in years

% dt : scalar, length of period in years

% W : N\*J, vector of portfolio weights to fit without error.

% K1Q\_X : N\*N, normalized latent-model matrix (does not have to be diagonal, see form below)

% kinfQ : scalar, when the model is stationary, the long run mean of the annualized short rate under Q is -kinfQ/K1(m1,m1)

% Sigma\_cP : N\*N, positive definite matrix that is the covariance of innovations to cP

% K0P\_cP : N\*1,

% K1P\_cP : N\*N,

% sigma\_e : scalar, standard error of yield observation errors

%

%

% Compute likelihood conditioned on first observation!

%

% llk : T\*1 time series of -log likelihoods (includes 2-pi constants)

% AcP : 1\*J yt = AcP' + BcP'\*cPt (yt is J\*1 vector)

% BcP : N\*J AcP, BcP satisfy internal consistency condition that AcP\*W' = 0, BcP\*W' = I\_N

% AX : 1\*J yt = AX' + BX'\*Xt

% BX : N\*J Xt is the 'jordan-normalized' latent state

% yields\_filtered : T\*J E[cPt|y^o(t), y^o(t-1), .., y^o(1)]

% cP\_filtered : T\*N E[yt|y^o(t), y^o(t-1), .., y^o(1)]

%

% The model takes the form:

% r(t) = rho0\_cP + rho1\_cP'\*cPt

% = rinfQ + 1'\*Xt (Xt is the 'jordan-normalized' state

% = 1 period discount rate (annualized)

%

% Under Q:

% X(t+1) - X(t) = K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% cP(t+1) - cP(t) = K0Q\_cP + K1Q\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

% where Sigma\_X is chosen to match Sigma\_cP

%

% Under P:

% cP(t+1) - cP(t) = K0P\_cP + K1P\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

%

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setup

[T,J] = size(yields\_o);

% Setup W if we are using individual yields without error:

if isempty(W)

W = eye(J);

W = W(ismember(mats, yields\_woe),:); % N\*J

end

N = size(W,1);

cP = yields\_o\*W'; % (T+1)\*N, cP stands for math caligraphic P.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% COMPUTE THE JSZ-Normalized version of the model:

% yt = AcP' + BcP'\*cPt, AcP is 1\*J, BcP is N\*J

[BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings(W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now setup Kalman filter.

y = yields\_o; % T\*J

Phi = eye(N) + K1P\_cP; % N\*N

alpha = K0P\_cP; % N\*1

Q = Sigma\_cP; % N\*N

R = eye(J)\*sigma\_e^2;

% Assume that the t=0 states (the time before any yields are observed)

% are distributed N(mu, Sigma) with the model stationary distribution,

[x00, P00] = asymptoticMomentsGaussian(K0P\_cP, K1P\_cP, Sigma\_cP);

A = BcP.';

b = AcP.';

% If K1P\_cP is non-stationary, we will have a problem with this assumption,

% so use something else assuming LLK will let us evaluate the likelihood

eigP00 = eig(P00);

if any(~isreal(eigP00)) || any(eigP00<0)

x00 = mean(cP).';

P00 = cov(cP);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Run Kalman Filter:

[x\_tm1t, P\_tm1t, x\_tt, P\_tt, K\_t, llk] = kf( y.', Phi, alpha, A, b, Q, R, x00, P00);

llk = -llk; % we return negative of the llk

cP\_filtered = x\_tt.'; % T\*N E[cPt|y^o(t), y^o(t-1), .., y^o(1)]

yields\_filtered = cP\_filtered\*BcP + ones(T,1)\*AcP; % T\*J

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [llk, AcP, BcP, AX, BX, kinf, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ, K0Q\_X, K1Q\_X, rho0\_X, rho1\_X] = ...

jszLLK\_kinf\_conc(yields\_o, W, K1Q\_X, Sigma\_cP, mats, dt, K0P\_cP, K1P\_cP, sigma\_e, rankRP)

%function [llk, AcP, BcP, AX, BX, K0P\_cP, K1P\_cP, sigma\_e, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, cP, llkP, llkQ] = ...

% jszLLK\_kinf(yields\_o, W, yields\_woe, K1Q\_X, rinfQ, Sigma\_cP, mats, dt, K0P\_cP, K1P\_cP, sigma\_e, rankRP)

%

%

% This function computes the likelihood for a Gaussian term structure.

% See "A New Perspective on Gaussian Dynamic Term Structure Models" by Joslin, Singleton and Zhu

%

% INPUTS:

% yields\_o : (T+1)\*J, matrix of observed yields (first row are t=0 observations, which likelihood conditions on)

% mats : 1\*J, maturities in years

% dt : scalar, length of period in years

%

% W : N\*J, vector of portfolio weights to fit without error.

% K1Q\_X : N\*N, normalized latent-model matrix (does not have to be diagonal, see form below)

% Sigma\_cP : N\*N, positive definite matrix that is the covariance of innovations to cP

%

% OPTIONAL INPUTS:

% K0P\_cP : N\*1, OPTIONAL (supply [] to omit)

% K1P\_cP : N\*N, OPTIONAL (supply [] to omit)

% sigma\_e : scalar, standard error of yield observation errors

%

%

% Compute likelihood conditioned on first observation!

%

% llk : T\*1 time series of -log likelihoods (includes 2-pi constants)

% AcP : 1\*J yt = AcP' + BcP'\*Xt (yt is J\*1 vector)

% BcP : N\*J AcP, BcP satisfy internal consistency condition that AcP\*W' = 0, BcP\*W' = I\_N

% AX : 1\*J yt = AX' + BX'\*Xt

% BX : N\*J Xt is the 'jordan-normalized' latent state

%

%

% The model takes the form:

% r(t) = rho0\_cP + rho1\_cP'\*cPt

% = rinfQ + 1'\*Xt (Xt is the 'jordan-normalized' state

% = 1 period discount rate (annualized)

%

% Under Q:

% X(t+1) - X(t) = K0Q\_X + K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% cP(t+1) - cP(t) = K0Q\_cP + K1Q\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

% where Sigma\_X is chosen to match Sigma\_cP

% and K0Q\_X(m1) = kinfQ where m1 is the multiplicity of the highest eigenvalue (typically 1)

%

% Under P:

% cP(t+1) - cP(t) = K0P\_cP + K1P\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

%

% Model yields are given by:

% yt^m = AcP' + BcP'\*cPt (J\*1)

% And observed yields are given by:

% yt^o = yt^m + epsilon\_e(t)

% where V\*epsilon\_e~N(0,sigma\_e^2 I\_(J-N))

% and V is an (J-N)\*J matrix which projects onto the span orthogonal to the

% row span of W. This means errors are orthogonal to cPt and cPt^o = cPt^m.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setup

[T,J] = size(yields\_o(2:end,:));

% Setup W if we are using individual yields without error:

N = size(W,1);

cP = yields\_o\*W'; % (T+1)\*N, cP stands for math caligraphic P.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setup default parameters:

concentrateSigmaE = true;

concentrateK0PK1P = true;

if exist('K0P\_cP','var') && ~isempty(K0P\_cP), concentrateK0PK1P = false; end

if ~exist('rankRP','var') || isempty(rankRP), rankRP = N; end

if exist('sigma\_e','var') && ~isempty(sigma\_e), concentrateSigmaE = false; end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% COMPUTE THE Q-LIKELIHOOD:

% First find the loadings for the model:

% yt = AcP' + BcP'\*cPt, AcP is 1\*J, BcP is N\*J

% AcP = alpha0\_cP\*kinf + alpha1\_cP

rho0\_cP = 0;

% AcP0, AX0 will be the loadings with rho0\_cP = 0, which won't be correct

[BcP, AcP0, K0Q\_cPx, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX0, BX, Sigma\_X, alpha0\_cP, alpha1\_cP, alpha0\_X, alpha1\_X, m1] = jszLoadings\_rho0cP(W, K1Q\_X, rho0\_cP, Sigma\_cP, mats, dt);

V = null(W)';

kinf = ((mean(yields\_o(2:end,:))' - alpha1\_cP' - BcP'\*mean(cP(2:end,:)).')'\*(V'\*V\*alpha0\_cP')) / (alpha0\_cP\*V'\*V\*alpha0\_cP');

AX = alpha0\_X\*kinf + alpha1\_X;

AcP = alpha0\_cP\*kinf + alpha1\_cP;

K0Q\_X = zeros(N,1);

K0Q\_X(m1) = kinf;

rho0\_X = 0;

rho1\_X = ones(N,1);

[K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP] = jszRotation(W, K1Q\_X, K0Q\_X, rho1\_X, rho0\_X, BX, AX);

yields\_m = ones(T+1,1)\*AcP + cP\*BcP; % (T+1)\*J, model-implied yields

yield\_errors = yields\_o(2:end,:) - yields\_m(2:end,:); % T\*J

square\_orthogonal\_yield\_errors = (yield\_errors).^2; % T\*J, but N-dimensional projection onto W is always 0, so effectively (J-N) dimensional

% Compute optimal sigma\_e if it is not supplied

if concentrateSigmaE

sigma\_e = sqrt( sum(square\_orthogonal\_yield\_errors(:))/(T\*(J-N)) );

end

llkQ = .5\*sum(square\_orthogonal\_yield\_errors.')/sigma\_e^2 + (J-N)\*.5\*log(2\*pi) + .5\*(J-N)\*log(sigma\_e^2); % 1\*T

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% COMPUTE THE P-LIKELIHOOD:

% We have three case: (1) unconstrained, (2) reduced rank risk premia and (3) kalman filter

if concentrateK0PK1P

if rankRP==N

% Run OLS to obtain maximum likelihood estimates of K0P, K1P

[K1PplusI, K0P\_cP] = regressVAR(cP);

K1P\_cP = K1PplusI - eye(N);

else

% Run reduced rank regression to compute ml estimates of K0P, K1P

Y1 = diff(cP) - cP(1:end-1,:)\*K1Q\_cP.';

X = cP(1:end-1,:);

[beta1, alpha1] = reducedRankFreeInterceptRegress(Y1, X, rankRP, Sigma\_cP);

K1P\_cP = beta1.' + K1Q\_cP;

K0P\_cP = alpha1.';

% THIS WOULD DO REDUCED RANK of [K0P, K1P] - [K0Q, K1Q] (only rankRP are priced instead of only rankRP have time-varying price)

% Y1 = diff(cP) - cP(1:end-1,:)\*K1Q\_cP.' - ones(T,1)\*K0Q\_cP.';

% X = [ones(T,1), cP(1:end-1,:)];

% K1P\_cP = beta1(2:end,:).' + K1Q\_cP;

% K0P\_cP = beta1(1,:).' + K0Q\_cP;

end

end

innovations = cP(2:end,:).' - (K0P\_cP\*ones(1,T) + (eye(N)+K1P\_cP)\*cP(1:end-1,:).'); % N\*T

llkP = .5\*N\*log(2\*pi) + .5\*log(det(Sigma\_cP)) + .5\*sum(innovations.\*(Sigma\_cP\innovations),1); % 1\*T

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

llk = (llkQ + llkP).'; % T\*1 series

function [BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings(W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt, Sigma\_X)

% function [BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X] = jszLoadings(W, K1Q\_X, kinfQ, Sigma\_cP, mats, dt, Sigma\_X)

%

% Inputs:

% mats : 1\*J, maturities in years

% dt : scalar, length of period in years

% W : N\*J, vector of portfolio weights to fit without error.

% K1Q\_X : N\*N

% kinfQ : scalar, when the model is stationary, the long run mean under Q is -kinfQ/K1

% Sigma\_cP, Sigma\_X : N\*N covariance of innovations. PROVIDE ONE OR THE OTHER

%

% Returns:

% AcP : 1\*J

% BcP : N\*J

% AX : 1\*J

% BX : N\*J

% Sigma\_X : N\*N

%

%

% This function:

% 1. Compute the loadings for the normalized model:

% X(t+1) - X(t) = K0Q\_X + K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% and r(t) = 1.X(t)

% where r(t) is the annualized short rate, (i.e. price of 1-period zero coupon bond at time t is exp(-r(t)\*dt))

% and K0Q\_X(m1) = kinf, and K0Q\_X is 0 in all other entries.

% m1 is the multiplicity of the first eigenvalue.

%

% If Sigma\_X is not provided, it is solved for so that Sigma\_cP (below) is matched.

% yt = AX' + BX'\*Xt

%

% 2. For cPt = W\*yt and the model above for Xt, find AcP, BcP so that

% yt = AcP' + BcP'\*cPt

%

%

J = length(mats);

N = size(K1Q\_X,1);

rho0d = 0;

rho1d = ones(N,1);

mats\_periods = round(mats/dt);

M = max(mats\_periods);

[K1Q\_X, isTypicalDiagonal, m1] = jszAdjustK1QX(K1Q\_X);

K0Q\_X = zeros(N,1);

K0Q\_X(m1) = kinfQ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If Sigma\_cP is provided, we need to compute Sigma\_X by

% first computing BX

%

if nargin<7 || isempty(Sigma\_X)

% First compute the loadings ignoring the convexity term -- BX will be correct

% yt = AX' + BX'\*Xt

% yt is J\*1, AX is 1\*J

% BX is N\*J

% Xt is N\*1

%

% cPt = W\*yt (cPt N\*1, W is N\*J)

% = W\*AX' + W\*BX'\*Xt

% = WAXp + WBXp\*Xt

%

% Substituting:

% yt = AX' + BX'\*(WBXp\(cPt - WAXp))

% = (I - BX'\*(WBXp\WAXp))\*AX' + BX'\*WBXp\cPt

% = AcP' + BcP'\*cPt

% where AcP = AX\*(I - BX'\*(WBXp\WAXp))'

% BcP = (WBXp)'\BX

%

% Sigma\_cP = W\*BX'\*Sigma\_X\*(W\*BX')'

% Sigma\_X = (W\*BX')\Sigma\_cP/(W\*BX')'

%

% If K1d isn't diagonal, we should use the Recurrence solver:.

if isTypicalDiagonal

BX = gaussianDiscreteYieldLoadingsDiagonal(mats\_periods, K0Q\_X, diag(K1Q\_X), zeros(N,N), rho0d\*dt, rho1d\*dt, dt); % N\*J

else

BX = gaussianDiscreteYieldLoadingsRecurrence(mats\_periods, K0Q\_X, K1Q\_X, zeros(N,N), rho0d\*dt, rho1d\*dt, dt); % N\*J

end

WBXp = W\*BX.'; % N\*N

Sigma\_X = (W\*BX')\Sigma\_cP/(BX\*W');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now with Sigma\_X in hand, compute loadings for AX

if isTypicalDiagonal

[BX, AX] = gaussianDiscreteYieldLoadingsDiagonal(mats\_periods, K0Q\_X, diag(K1Q\_X), Sigma\_X, rho0d\*dt, rho1d\*dt, dt);

else

[BX, AX] = gaussianDiscreteYieldLoadingsRecurrence(mats\_periods, K0Q\_X, K1Q\_X, Sigma\_X, rho0d\*dt, rho1d\*dt, dt);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Rotate the model to obtain the AcP, BcP loadings.

% (See above for calculation)

BcP = (W\*BX.').'\BX;

AcP = AX\*(eye(J) - BX'\*((W\*BX')\W))'; % 1\*J

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now compute the rotated model parameters:

WBXp = W\*BX';

WAXp = W\*AX';

K1Q\_cP = WBXp\*K1Q\_X/WBXp;

K0Q\_cP = WBXp\*K0Q\_X - K1Q\_cP\*WAXp;

rho0\_cP = - ones(1,N)\*(WBXp\WAXp);

rho1\_cP = (WBXp)'\ones(N,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X, alpha0\_cP, alpha1\_cP, alpha0\_X, alpha1\_X, m1] = jszLoadings\_rho0cP(W, K1Q\_X, rho0\_cP, Sigma\_cP, mats, dt, Sigma\_X)

%function [BcP, AcP, K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP, K0Q\_X, K1Q\_X, AX, BX, Sigma\_X, alpha0\_cP, alpha1\_cP, alpha0\_X, alpha1\_X, m1] = jszLoadings\_rho0cP(W, K1Q\_X, rho0\_cP, Sigma\_cP, mats, dt, Sigma\_X)

%

%

% This gives a slight variant of the JSZ normalization.

%

% There is one "intercept" parameter governing the risk-neutral

% distribution. This can be the long run mean of the short rate under Q

% (assuming stationarity) or the drift of the most persistent factor.

%

% Here we parameterize the intercept parameter through the relationship:

% r\_t = rho0\_cP + rho1\_cP.Xt

%

% Given a fixed set of eigenvalues for K1Q, there is a one-to-one

% mapping between rho0\_cP and kinf.

%

%

% Inputs:

% mats : 1\*J, maturities in years

% dt : scalar, length of period in years

% W : N\*J, vector of portfolio weights to fit without error.

% K1Q\_X : N\*N

% rho0\_cP : scalar, the short rate will be rt = rho0\_cP + rho1\_cP.cP\_t

% Sigma\_cP, Sigma\_X : N\*N covariance of innovations. PROVIDE ONE OR THE OTHER

%

% Returns:

% AcP : 1\*J

% BcP : N\*J

% AX : 1\*J

% BX : N\*J

% Sigma\_X : N\*N

% AcP = alpha0\_cP\*kinf + alpha1\_cP

% AX = alpha0\_X\*kinf + alpha1\_X

%

%

% This function:

% 1. Compute the loadings for the normalized model:

% X(t+1) - X(t) = [kinfQ;0] + K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% and r(t) = 1.X(t)

% where r(t) is the annualized short rate, (i.e. price of 1-period zero coupon bond at time t is exp(-r(t)\*dt))

% If Sigma\_X is not provided, it is solved for so that Sigma\_cP (below) is matched.

% yt = AX' + BX'\*Xt

%

% 2. For cPt = W\*yt and the model above for Xt, find AcP, BcP so that

% yt = AcP' + BcP'\*cPt

%

%

J = length(mats);

N = size(K1Q\_X,1);

rho0d = 0;

rho1d = ones(N,1);

mats\_periods = round(mats/dt);

M = max(mats\_periods);

[K1Q\_X, isTypicalDiagonal, m1] = jszAdjustK1QX(K1Q\_X);

K0Q\_X = zeros(N,1);

K0Q\_X(m1) = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% First compute the loadings ignoring the convexity term -- BX will be correct

% yt = AX' + BX'\*Xt

% yt is J\*1

% AX is 1\*J

% BX is N\*J

% Xt is N\*1

%

% cPt = W\*yt (cPt N\*1, W is N\*J)

% = W\*AX' + W\*BX'\*Xt

% = WAXp + WBXp\*Xt

%

% Substituting:

% yt = AX' + BX'\*(WBXp\(cPt - WAXp))

% = (I - BX'\*(WBXp\WAXp))\*AX' + BX'\*WBXp\cPt

% = AcP' + BcP'\*cPt

% where AcP = AX\*(I - BX'\*(WBXp\WAXp))'

% BcP = (WBXp)'\BX

%

% Sigma\_cP = W\*BX'\*Sigma\_X\*(W\*BX')'

% Sigma\_X = (W\*BX')\Sigma\_cP/(W\*BX')'

%

% If K1d isn't diagonal, we should use the Recurrence solver:.

% Since we are setting covariance to zero and K0Q\_X = [1;0;0..;0], the "A" loadings will be the loading on kinf

if isTypicalDiagonal

[BX, alpha0\_X] = gaussianDiscreteYieldLoadingsDiagonal(mats\_periods, K0Q\_X, diag(K1Q\_X), zeros(N,N), rho0d\*dt, rho1d\*dt, dt); % N\*J

else

[BX, alpha0\_X] = gaussianDiscreteYieldLoadingsRecurrence(mats\_periods, K0Q\_X, K1Q\_X, zeros(N,N), rho0d\*dt, rho1d\*dt, dt); % N\*J

end

WBXp = W\*BX.'; % N\*N

if nargin<7 || isempty(Sigma\_X)

Sigma\_X = (W\*BX')\Sigma\_cP/(BX\*W');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now with Sigma\_X in hand, compute loadings for AX

if isTypicalDiagonal

[BX, AX1] = gaussianDiscreteYieldLoadingsDiagonal(mats\_periods, K0Q\_X, diag(K1Q\_X), Sigma\_X, rho0d\*dt, rho1d\*dt, dt);

else

[BX, AX1] = gaussianDiscreteYieldLoadingsRecurrence(mats\_periods, K0Q\_X, K1Q\_X, Sigma\_X, rho0d\*dt, rho1d\*dt, dt);

end

% AX1 gives the intercept with K0Q\_X all zeros except 1 in the m1-th entry.

% So AX = alpha0\_X\*kinf + alpha1\_X which alpha1\_X = AX1 - alpha0\_X

alpha1\_X = AX1 - alpha0\_X;

WBXp = W\*BX';

% Need to find what kinf should be to get the desired rho0\_cP:

% rt = 1'\*Xt

% cPt = (W\*alpha0\_X')\*kinf + (W\*alpha1\_X') + (W\*BX')\*Xt

% rt = 1'\*(W\*BX')^(-1)\*[cPt -(W\*alpha0\_X')\*kinf - (W\*alpha1\_X')]

% --> rho0\_cP = -1'\*(W\*BX')^(-1)\*(W\*alpha0\_X')\*kinf - 1'\*(W\*BX')^(-1)\*(W\*alpha1\_X')

a0 = ones(1,N)\*(WBXp\(W\*alpha0\_X'));

a1 = ones(1,N)\*(WBXp\(W\*alpha1\_X'));

kinf = -(rho0\_cP + a1)/a0;

K0Q\_X(m1) = kinf;

AX = alpha0\_X\*kinf + alpha1\_X;

% AcP = alpha0\_cP\*kinf + alpha1\_cP

alpha0\_cP = ((eye(J) - (BX'/(W\*BX'))\*W)\*alpha0\_X')';

alpha1\_cP = ((eye(J) - (BX'/(W\*BX'))\*W)\*alpha1\_X')';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finally, rotate the model to obtain the AcP, BcP loadings.

% (See above for calculation)

BcP = (W\*BX.').'\BX;

AcP = AX\*(eye(J) - BX'\*((W\*BX')\W))'; % 1\*J

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now compute the rotated model parameters:

WBXp = W\*BX';

WAXp = W\*AX';

K1Q\_cP = WBXp\*K1Q\_X/WBXp;

K0Q\_cP = WBXp\*K0Q\_X - K1Q\_cP\*WAXp;

rho0\_cP = - ones(1,N)\*(WBXp\WAXp);

rho1\_cP = (WBXp)'\ones(N,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP] = jszRotation(W, K1Q\_X, K0Q\_X, rho1\_X, rho0\_X, BX, AX)

% function [K0Q\_cP, K1Q\_cP, rho0\_cP, rho1\_cP] = jszRotation(W, K1Q\_X, K0Q\_X, rho1\_X, rho0\_X, BX, AX)

%

%

% Inputs:

% W : N\*J, vector of portfolio weights to fit without error.

% K1Q\_X : N\*N

% K0Q\_X : N\*1

% rho1\_X : N\*1

% rho0\_X : 1\*1

% BX : N\*J

% AX : 1\*J

%

% Returns:

% K0Q\_cP : N\*1

% K1Q\_cP : N\*N

% rho0\_cP : scalar

% rho1\_cP : N\*1

%

%

% r(t) = rho0\_cP + rho1\_cP'\*cPt

% = 1'\*Xt

% = 1 period discount rate (annualized)

%

% Under Q:

% X(t+1) - X(t) = K0Q\_C + K1Q\_X\*X(t) + eps\_X(t+1), cov(eps\_X(t+1)) = Sigma\_X

% cP(t+1) - cP(t) = K0Q\_cP + K1Q\_cP\*X(t) + eps\_cP(t+1), cov(eps\_cP(t+1)) = Sigma\_cP

%

% Where Sigma\_X is chosen to match Sigma\_cP

%

% cPt = W\*yt (cPt N\*1, W is N\*J)

% = W\*AX' + W\*BX'\*Xt

% = WAXp + WBXp\*Xt

%

% Delta(cP) = WBXp\*Delta(Xt)

% = WBXp\*(K0\_X + K1Q\_X\*Xt + sqrt(Sigma\_X)\*eps(t+1))

% = WBXp\*K0Q\_X + WBXp\*(K1Q\_X)\*(WBXp\(cPt - WAXp)) + sqrt(Sigma\_cP)\*eps(t+1)

% = WBXp\*(K1Q\_X)/WBXp\*cPt +[WBXp\*K0Q\_X - WBXp\*(K1Q\_X)/WBXp\*WAXp] + sqrt(Sigma\_cP)\*eps(t+1)

%

% rt = rho0\_X + rho1\_X'\*Xt [annualized 1-period rate]

% = rho0\_X + rho1\_X'\*(WBXp\(cPt - WAXp))

% = [rho0\_X - rho1\_X'\*(WBXp\WAXp)] + ((WBXp)'\rho1\_X)'\*cPt

WBXp = W\*BX';

WAXp = W\*AX';

K1Q\_cP = WBXp\*K1Q\_X/WBXp;

K0Q\_cP = WBXp\*K0Q\_X - K1Q\_cP\*WAXp;

rho0\_cP = rho0\_X - rho1\_X'\*(WBXp\WAXp);

rho1\_cP = (WBXp)'\rho1\_X;

function W = jszWeightsFromMats(mats\_woe, mats)

% function W = jszWeightsFromMats(mats\_woe, mats);

%

% This is an easy way to generate the weighting matrix when we measure

% specific yields without error.

%

% For example:

% mats\_woe = [1,5];

% mats = [1:5}

% gives:

% [1,0,0,0,0;

% 0,0,0,0,1]

%

inds = ismember(mats, mats\_woe);

if ~(sum(inds)==length(mats\_woe))

error('Some of the maturities without error arent in the original maturities');

end

J = length(mats);

W = eye(J);

W = W(inds,:);