MathWorks®
Accelerating the pace of engineering and science

MATLAB® Control Theory

Classical Control Theory

Transfer Functions (TF)

`s = tf('s')` `sys = (s-1)/(s+1);`	Create special variable s that you can use in a rational expression to create a continuous-time transfer function model.
`num = [5]; den = [1 1 3];` `sys = tf(num,den)`	Create a continuous-time transfer function (TF) model by specifying numerator (num) and denominator (den) properties.
`z = tf('z',ts)` `sys = (z-1)/(z+1);`	Create a special variable z that you can use in a rational expression to create a discrete-time transfer function with sample time ts.
`sys = tf(num,den,ts)`	Create a discrete-time transfer function model, setting the numerator (num), denominator (den), and sample time (ts) properties.

`L = 1/(S+5);`
`S_s = feedback(1, L);`
`T_s = feedback(L, 1);`
`% Or %= 1-S`
	Sensitivity TF.

`% T(jw) + S(jw) = 1`	Complementary Sensitivity or Co-Sensitivity TF. See functions: getSensitivity and getCompSensitivity for additional information, and https://www.mathworks.com/help/control/ug/using-feedback-to-close-feedback-loops.html for ways of closing feedback loops with the feedback command.
`C = pid(Kp,Ki,Kd,tf)`	PID Controller with time constant.
`sys = filt(num,den)`	Specify discrete transfer functions in DSP format. ts is unspecified.
`ts = 0.5;` `sys = filt(num,den,ts)`	With ts specified.

Responses

`stepplot(sys)`	Plots step response of dynamic system.

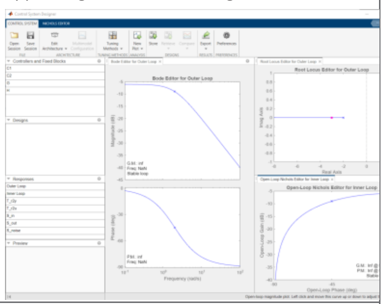
`[y,tOut] = step(sys)`
`[y,tOut] = step(sys,tFinal)`
`[y,tOut] = step(sys,[t0,tFinal])`
`[y,tOut] = step(sys,t)`
	Returns the step response (y) and time vector tOut.
	t0 is the initial time (when not specified assumed to be zero).
	tFinal the final time.
	t is a time vector.

`h = stepplot(sys)` `h = stepplot(sys1,...,sysN)`	Plot step response with additional plot customization options.
`S_info = stepinfo(sys)`	Gives step response data like rise time, undershoot, setting time, etc.
`S_info = stepinfo(y,t)`	Can either use a sym or specify vectors y and t [length(y) = length(t)].
`impulse(sys)`	Impulse response plot of dynamic system. Same syntax as step.
`impulseplot`	Same syntax as stepplot.
`lsim(sys,u,t)` `lsim(sys,u,t,x0)`	Plot simulated time response of dynamic system to arbitrary inputs. Allows you to choose the input u, time vector t, and initial state vector x0.
h = lsimplot(sys) **h = lsimplot(sys,u,t)** **h = lsimplot(All_sys,u,t)**	Very similar syntax as stepplot. Can plot multiple systems. sys1, LineSpec1, sys2, LineSpec2.

Poles, Zeros, and s-domain

P = pole(sys)	Returns the poles of a given system.
Z = zero(sys) **[Z,gain] = zero(sys)**	Returns the zeros of a given system. Returns the system zero-pole-gain.
[p,z = pzmap(sys)	Return the system poles and zeros of the dynamic system model sys.
pzplot(sys)	Plot pole-zero map of dynamic system sys with customization options.
% z = zeros **% p = poles** **% k = gain** **sys = zpk(z,p,k)**	Creates a continuous-time zero-pole-gain model with zeros and poles specified as vectors and the scalar value of gain. The output sys is a zpk model object storing the model data. Set zeros or poles to [] for systems without zeros or poles. These two inputs need not have equal length, and the model need not be proper (that is, have an excess of poles).
sys = zpk(z,p,k,ts)	If you want a discrete zpk model.
sys = zpk(z,p,k,ltiSys)	If you want to specify properties inherited from the dynamic system model include ltiSys.
[z,p,k] = zpkdata(sys)	Access zero-pole-gain data
[z,p,k,Ts] = zpkdata(sys) **[z,p,k,Ts,covz,covp,covk] = zpkdata(sys)**	Returns the zeros z, poles p, and gain(s) k of the zero-pole-gain, sample time Ts, covariances of the zeros, poles and gainmodel of the identified model sys.
iopzmap(sys)	Plot pole-zero map for I/O pairs of models.
iopzplot(sys)	Plot pole-zero map for I/O pairs with additional plot customization options.
bode(sys)	Plot frequency response of dynamic system model sys.
[mag,phase,wout] = bode(sys)	Compute the frequency response of dynamic system model sys, and return the magnitude and phase of the response at each frequency in the vector wout.
bodeplot(sys)	Plot Bode frequency response with additional plot customization options.
margin(sys) **[Gm,Pm,Wcg,Wcp] = margin(sys)**	Lets you plot and find the Gain margin, phase margin, and crossover frequencies.
allmargin(sys) **% open loop system**	Gives relevant margin info like rise time, GainMargin, PhaseMargin, DelayMargin, system stability, etc.
rlocus(sys)	Root locus plot of dynamic system. Use grid on for the relevant grid.
[r,k] = rlocus(sys)	Calculate the root locus of SISO model sys and return the resulting vector of feedback gains k and corresponding complex root locations r.
rlocusplot(sys)	Plot root locus and return plot handle.
nyquist(sys) **[re,im,wout] = nyquist(sys)**	Nyquist plot of frequency response. Use grid on for the relevant grid.
nyquistplot(sys)	Nyquist plot with additional plot customization options.

MathWorks®
Accelerating the pace of engineering and science

nichols(sys)	Nichols chart of frequency response. Use grid on for the relevant grid.
[mag,phase,wout] = nichols(sys)	Compute the frequency response of dynamic system model sys and returns the magnitude and phase of the response at each frequency in the vector wout
nicholsplot(sys)	Plot Nichols frequency responses with additional plot customization options.
ltiview(sys)	Opens the Linear Time Invariant (LTI) Viewer GUI for a given system.
controlSystemDesigner	Interactively design and analyze SISO controllers with the Control System Designer app, using automated tuning methods. 
damp(sys)	Displays the damping ratio, natural frequency, and time constant of the poles of the linear model sys
[wn,zeta,p] = damp(sys)	Returns the natural frequencies wn, and damping ratios zeta of the poles of sys, and the sys poles, p.

Modern Control Theory

State Space (SS) Representation

Continuous-time (CT) SS $\dot{x} = Ax + Bu$ $y = Cx + Du$	*x* is the state vector. *u* is the input vector. *y* is the output vector.
Discrete-time (DT) SS $x[n+1] = Ax[n] + Bu[n]$ $y[n] = Cx[n] + Du[n]$	*A* is the system matrix. *B* is the input matrix. *C* is output Matrix. *D* is the feedforward or direct transmission matrix.
sys = ss(A,B,C,D)	Creates a continuous-time state-space model object.
sys = ss(A,B,C,D,ltiSys)	Creates a state-space model with properties such as input and output names, internal delays and sample time values inherited from the model ltisys.
sys = ss(ssSys,'minimal')	Returns the minimal state-space realization with no uncontrollable or unobservable states.
[a,b,c,d,Ts] = ssdata(sys)	Access state-space model data. For viewing TF data see help tfdata.
[num,den] = tfdata(tf(sys)) **syms s;** **I = eye(size(A));** **H_s = C*(s*I-A)^-1*B+D**	Converts the state space model represented by sys to transfer function and returns its numerator and denominator coefficients.Creates a symbol expression. Useful for checking SISO, SIMO, MISO, and MIMO hand calculations.
[A,B,C,D] **= ssdatass(sys));**	Converts the transfer function represented by sys to state space model and extracts its A, B,C and D matrices

`sys = ss(A,B,C,D,ts)`	Creates the discrete-time state-space model object with the sample time ts (in seconds). Set ts = -1 to leave the sample time to unspecified.
`lsimplot(sys,u,t,x0);` `[t, y] = lsim(sys,u,t,x0)`	Plot simulated time response of dynamic system with t (time samples) and x0 (initial state values).
`io = linio(block,port)lins ys = linearize(model,io)`	Create an input perturbation analysis point for the signal that originates from the specified output port of a Simulink® block. Linearize the Simulink model using the specified analysis point(s).

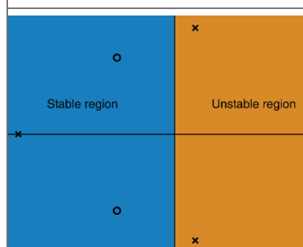
Controllability, Observability, and Minimal Realizations

`Co = ctrb(sys)`
`= ctrb(sys.A,sys.B);`
`Co = ctrb(A,B)`
`rank(Co)`
	Returns the Controllability matrix Co. If full rank or $det(Co) \neq 0$ then the system is Controllable.
	Co = [B A*B A^2*B … A^(n-1)*B].

`Ob = obsv(sys)`
`= obsv(sys.A,sys.C);`
`Ob = obsv(A,C)`
`rank(Ob)`
	Returns the Observability matrix Ob. If full rank or $det(Ob) \neq 0$ then the system is Observable.
	Ob = [C; C*A; C*A^2; … C*A^(n-1)].

`H = Ob*Co % note the order. It matters.`	Hankel Matrix. If full rank or $det(H) \neq 0$ then the system is a minimal realization.
`H = hankel(c)`	Returns a square Hankel Matrix where c defines the first column of the matrix, and the elements are zero below the main anti-diagonal.
`H = hankel(c,r)`	Returns a Hankel matrix with c as its first column and r as its last row.
`sysr = minreal(sys)` `[sysr,U] = minreal(sys,tol)`	Minimal realization or pole-zero cancellation. U such that (U*A*U', U*B, C*U') is a Kalman decomposition of (A,B,C).

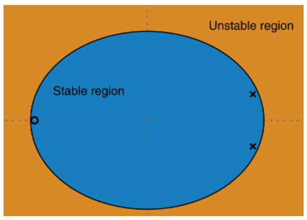
Stability (CT)

	**When plotting (s-plane) the poles (x's) and zeros (o's) of a system you want the poles (s) to be in the light half plane (LHP).** $s = \sigma + j\omega = Re(s) + Im(s)$. $s = \sigma + j\omega$ ***is complex.*** $s = \sigma$ ***is simple.*** $s = j\omega$ ***is purely imaginary.***

`sys = tf([2 5 1],[1 3 5]);`
`h = pzplot(sys);`
`grid on % Use when you want the angle of poles. Theta = atan(Imag(s)/Real(s))`
`pzmap(sys)`
`[p,z] = pzmap(sys)`
	Asymptoticly Stable / Stable $Re[s_i] = \sigma_i < 0$ for all $i = 1, \ldots, n$. (All poles need to be in the LHP).
	Marginally Stable if $\sigma_i = 0$ for any simple pole and poles $\sigma_i > 0$.
	Unstable if one or both of these conditions is satisfied: 1) $\sigma_i > 0$ for any pole and 2) $\sigma_i = 0$ for any multiple order pole s_i(poles that repeat).

`eig(A) % Check eigenvalues for stability.`
`% Or use`
`s_Poles = abs(pole(sys))`
`Stable = isstable(sys)`
	A is **Hurwitz** if the eigenvalues of A (nxn) have strictly negative real parts ($Re[\lambda_i] < 0$).
	Asymptoticly Stable / Stable when all λs are $Re[\lambda] < 0$.
	Marginally stable if any λs are $Re[\lambda] = 0$.
	Unstable if any λs are $Re[\lambda] > 0$.

Stability (DT)



(plot above)	**Discrete Time (DT)** shares similar properties as **CT**. Plot is of the **z-plane.** However, we now have a unit circle instead of LHP and RHP. $s = \sigma + j\omega$ $z = e^{sT}$ (Standard Z-transform)

`A = [0.1 0; 0.2 0.9];`
`B = [0.1; 0.1];`
`C = [10 5];`
`D = [0];`
`sys = ss(A,B,C,D,'Ts',0.1)`
`pzmap(sys) % DT`
	Asymptoticly Stable / Stable if all poles lie within the unit circle.
	Marginally Stable if there are any poles the edge of the circle $\lvert z\rvert$. Poles cannot be outside of this circle.
	Unstable if any poles lie outside of the circle.

`A_k = sys.A;`
`eig(A_k) % Check DT eigenvalues for stability.`
`% Or use`
`s_Poles = abs(pole(sys))`
`Stable = isstable(sys)`
	Asymptoticly Stable / Stable if $\lvert\lambda_i\rvert < 1$ for all $i = 1,\ldots,n$.
	Marginally stable if $\lvert\lambda_i\rvert \leq 1$ for all $i = 1,\ldots,n$.
	Unstable if $\lvert\lambda_i\rvert > 1$ for any or all $i = 1,\ldots,n$.

Conversion from CT to DT

`sysd = c2d(sysc,Ts)`	Discretizes the continuous-time dynamic system model sysc using a chosen method [when not specified zero-order hold (zoh) will be used] on the inputs and a sample time of Ts.
`sysd = c2d(sysc,Ts,method)`	Method = 'zoh' (default) | 'foh' | 'impulse' | 'tustin' % (aka bilinear transform) | 'matched' | 'least-squares' | 'damped'

`H = tf([1 -1],[1 4 5],'InputDelay', 0.5);`
`Hd = c2d(H,0.1,'foh');`
`step(H,'-',Hd,'--')`
	Plots the CT and DT TFs with an input delay of 0.5 seconds.
	Note: to convert from DT to CT use d2c.

`TF = 1/(10*s+5)`
`% 1st order method written in terms of z (s = ?).`
`% z = exp(s*Ts).`
	Discretization for hand calculations.

`s_FE = (z-1)/Ts;`
`s_BE = (z-1)/(Ts*z);`
`s_Tustin = (2/Ts)*(z-1)/(z+1);`
	Forward Euler (FE) Discretization.
	Backward Euler (BE) Discretization.
	Tustin (bilinear transform) approximation Discretization.

MISC Useful Functions

`Y_rad = atan(X)` `Y_deg = atand(X)`	Inverse tangent in radians. Inverse tangent in degrees. Note: same syntax for the other trigonometric functions.
`P_deg = atan2(Y,X)` `P_deg = atan2d(Y,X)`	Four-quadrant inverse tangent in radians. Four-quadrant inverse tangent in degrees.
`length(A)`	Length of largest array dimension.
`size(A)`	Array dimensions.
`numel(A)`	Number of elements in array.
`zeros(m,n)`	Create m x n matrix of zeros.
`ones(m,n)`	Creates m x n matrix of ones.
`eye(n)`	Creates a n x n square identity matrix.
`trace(A)`	Returns the sum of diagonal elements (a_{ii}) of a matrix. The sum of a_{ii} is equal to the sum of eigenvalues (λ).
`det(A)`	Determinant of matrix. The determinant is equal to the product of eigenvalues (λ).

`poly(A)`
`charpoly(A)`
`% Or if you want the equation.`
`I = eye(size(A))`
`syms S; det(S*I-A)`
	Polynomial with specified roots or characteristic polynomial
	Characteristic polynomial of matrix.
	Use expand(det(S*I-A)) if not displayed as a characteristic polynomial.

`null(A)`	Null space of matrix.
`orth(A)`	Orthonormal basis for matrix range.
`eig(A), eigs`	Eigenvalues and vectors of matrix (subset).
`D = diag(v)`	Returns a square diagonal matrix with the elements of vector v on the main diagonal.
`v = diag(D)`	Returns the diagonal elements of matrix in vector form.
`R = rref(A)`	Produces the reduced row echelon form of A
`rank(A)` `rank(A,TOL)`	Provides an estimate of the number of linearly independent rows or columns of a matrix A. The number of singular values of A that are larger than TOL. By default, TOL = max(size(A)) * eps(norm(A)).
`S = svd(A)` `[U,S,V] = svd(A)`	Singular value decomposition (SVD) [useful for MIMO Controls]. Returns the singular values of matrix A in descending order. Performs a singular value decomposition of matrix A, such that A = U*S*V'.
`s = svds(A)`	Subset of singular values and vectors Returns a vector of the six largest singular values of matrix A. This is useful when computing all of the singular values with svd is computationally expensive, such as with large sparse matrices.

`s = svds(A,k)` `s = svds(A,k,Sigma)`	Returns the k largest singular values. Returns k singular values based on the value of sigma. For example, svds(A,k,'smallest') returns the k smallest singular values.
`gsvd(A,B);`	Generalized SVD.

`sigma(sys)`
`sigma(sys1,...,sysN)`
 	
`[sv,wout] = sigma(sys)`
 	
`[sv,wout] = sigma(sys,w)`
	Singular value plot of dynamic system
	Plots the system(s).
	
	Each frequency in the vector wout. The output sv is a matrix, and the value sv(:,k) gives the singular values in descending order at the frequency wout(k). The function automatically determines frequencies in wout based on system dynamics.
	Returns the singular values sv at the frequencies specified by w.
	See sigmaplot for additional plot customization options.

`H = tf([1 -1],[1 1 1],-1);`
`z = 1+j;`
`frsp = evalfr(H,z)`
	Evaluate system response at a specified frequency.

`A = [0.1 0; 0.2 0.9];`
`B = [0.1; 0.1];`
`C = [10 5]; D = [0];`
`sys = ss(A,B,C,D);`
`[H,wout]=freqresp(sys)`
 	
`w = 3; % Hz`
`H = freqresp(sys,w)`
`units = 'Hz';`
`H = freqresp(sys,w,units)`
	Returns the frequency response of the dynamic system model system (sys) at frequencies wout. freqresp automatically determines the frequencies based on the dynamics of system (sys).
	
	w is the real frequency grid.
	Use units to specify the frequency units.

`issiso(sys)`	Returns a logical value of 1 (true) if the dynamic system model sys is SISO and a logical value of 0 (false) otherwise.
`isempty(sys)`	Returns a logical value of 1 (true) if the dynamic system model sys has no input or no output, and a logical value of 0 (false) otherwise. Where sys is a frd model, isempty(sys) returns 1 when the frequency vector is empty.

Properties, Data, Forms and State Space (SS) Realizations

`J = jordan(A)` `[V,J] = jordan(A)`	Jordan normal form (Jordan canonical form) Computes the Jordan normal form of the matrix A. Due to Jordan form being sensitive to numerical errors convert numeric input to exact symbolic form.

`Schur decomposition`
`T = schur(A)`
`[U,T] = schur(A)`
	Returns T, the Schur matrix of A.
	Return T and a unitary matrix U such that A = U*T*U'.

`scaledsys = prescale(sys)`	Scales the entries of the state vector of a state-space model sys to maximize the accuracy of subsequent frequency-domain analysis. The scaled model scaledsys is equivalent to sys.

`A = [0.1 0; 0.2 0.9];`
`[S, P, A_bal] = balance(A)`
 	
`A_bal = balance(A,'noperm')`
	Returns the scaling vector S and the permutation vector P separately.
	The transformation T and balanced matrix A_bal are obtained from A, S, and P by T(:,P) = diag(S) and A_bal(P,P) = diag(1./S)*A*diag(S).
	Scales A without permuting its rows and columns.

`A = [0.1 0; 0.2 0.9];`
`[P,R,C] = equilibrate(A)`
 	
`[P,R,C] = equilibrate(A,outputForm)`
	Permutes and rescales matrix A such that the new matrix B = R*P*A*C has a diagonal with entries of magnitude 1, and its off-diagonal entries are not greater than 1 in magnitude.
	
	outputForm = "matrix" or "vector".

`% Requires R2023b or higher.`
 	
`rng(0)`
`% clear rng`
`sys = drss(40);`
`method = "balanced"`
`R = reducespec(sys,method)`
	Creates a model order reduction (MOR) specification object for a dense or sparse linear time-invariant (LTI) model sys.
	Method options:
	"balanced"
	"ncf" (Requires Robust Control Toolbox software)
	"modal"

`% Requires R2023b or higher.`
`A = [0.1 0; 0.2 0.9];`
`B = [0.1; 0.1];`
`C = [10 5];`
`D = [0];`
`sys = ss(A,B,C,D);`
`[msys,blks] = modalreal(sys)`
 	
`[msys,blks,TL,TR] = modalreal(sys)`
	Returns a modal realization msys of an LTI model sys. This is a realization where A or (A,E) are block diagonal and each block corresponds to a real pole, a complex pair, or a cluster of repeated poles. blks is a vector containing block sizes down the diagonal.
	
	If you also want the block-diagonalizing transformations TL and TR.

`% Requires R2023b or higher.`
`A = [0.1 0; 0.2 0.9];`
`B = [0.1; 0.1];`
`C = [10 5];`
`D = [0];`
`sys = ss(A,B,C,D);`
`csys = compreal(sys)`
 	
`csys = compreal(sys,type)`
 	
`[csys,T] = compreal(___)`
	Returns the controllable companion realization of the single-input LTI model sys.
	Realization of the model sys defined by type.
	"c" for controllable companion form.
	$A_c = T^{-1}AT, B_c = T^{-1}B, C_c = CT$
	"o" for observable companion form.
	$A_o = TAT^{-1}, B_o = TB, C_o = CT^{-1}$
	
	Returns the transformation matrix as well T for explicit models with matrices A, B, C.

`sysT = ss2ss(sys,T)`	Performs the state-coordinate transformation of sys using the specified transformation matrix T (must be invertible).

MathWorks®
Accelerating the pace of engineering and science

`load wtankData.mat`
`sys = frd(response,frequency)`
`ts = 0.5; sys = frd(response,frequency,ts)`
 	
`num = [2,0]; den = [1,8,0];`
`ltiSys = tf(num,den,'TimeUnit','minutes','InputDelay',3)`
 	
`sys = frd(response,frequency,ltiSys)`
	Frequency-response data model
	Creates a continuous-time frequency-response data (frd) model, setting the ResponseData and Frequency properties.
	
	ts is sampling time and properties inherited from the dynamic system model is ltisys.

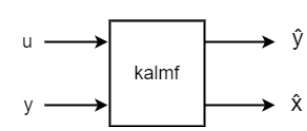
`Struct = get(sys)`
 	
`Value = get(sys,'PropertyName')`
	Access model property value.
	
	Specify the desired properties. For example, 'Numerator'.

`sys=set(sys,'Property',Value)` `sys=set(sys,'Property1',Value1, 'Property2',Value2,...)`	Set or modify model properties
`help and doc function`	Documentation on a given function.
`feedback, series, parallel, append, connect, tunablePID, getLoopTransfer, AnalysisPoint`	For block diagram related functions operations look into the documentation.
Simulation and Controller Design	

`K = place(A,B,p)`
`[K,prec] = place(A,B,p)`
 	
`L = place(A',C',p)'`
	Pole placement is a method of calculating the optimum gain matrix.
	p are your desired closed-loop poles. K is the state-feedack gain matrix.
	prec returns accuracy estimate of how closely the eignevalues of $A - BK$ match your desired pole locations.
	L are your observer gains.

`[K,S,P] = lqr(sys,Q,R,N)` `[K,S,P] = lqr(A,B,Q,R,N)`	Linear-Quadratic Regulator (LQR) design. Both Q and R need to be symmetric positive define matrices. **"Expensive" control strategy** tr(R) > tr(Q). **"Cheap" control strategy** Q is larger than R. Typical choices: 1) $Q = I$ with tr(R) > tr(Q) 2) $Q = I$ with tr(R) < tr(Q) 3) $Q = C^T C$ and $R = rI$ where r controls closed loop bandwidth. 4) Tune Q and R to match your use case.

`[K,S,e] = dlqr(A,B,Q,R,N)`	Linear-quadratic (LQ) state-feedback regulator for discrete-time state-space system.
`[X,K,L] = icare(A,B,Q,R,S,E,G)`	Implicit solver for continuous-time algebraic Riccati equations.

`[kalmf,L,P] = kalman(sys,Q,R,N)`
`[kalmf,L,P] = kalman(sys,Q,R,N,sensors,known)`
 	
`A = [1 -0.5 0.12;`
`     1  0    0;`
`     0  1    0];`
`B = [-0.383; 0.592; 0.512];`
`C = [1 0 0];`
`D = 0;`
`Ts = -1;`
`sys = ss(A,[B B],C,D,Ts, ...`
`'InputName',{'u' 'w'}, ...`
`'OutputName','y');`
`% Plant dynamics and additive input noise w`
`Q = 2.3;`
`R = 1;`
 	
`[kalmf,L,~,Mx,Z]`
`= kalman(sys,Q,R);`

	Design Kalman filter for state estimation. See the following resources:
	1) MATLAB series on Understanding Kalman Filtering.
	https://www.mathworks.com/videos/series/understanding-kalman-filters.html
	
	2) Kalman Filtering
	https://www.mathworks.com/help/control/ug/kalman-filtering.html
	
	3) State Estimation Using Time-Varying Kalman Filter
	https://www.mathworks.com/help/control/ug/state-estimation-using-time-varying-kalman-filter.html
	
	4) Nonlinear State Estimation Using Unscented Kalman Filter and Particle Filter
	https://www.mathworks.com/help/control/ug/nonlinear-state-estimation-using-unscented-kalman-filter.html

`[K,CL,GAM] = h2syn(P,NMEAS,NCON)`
 	
`[K,CL,GAM] = hinfsyn(P,NMEAS,NCON)`
 	
`[K,CLperf] = musyn(P,nmeas,ncont)`
	H2-optimal controller synthesis.
	
	H-infinity controller synthesis.
	
	Robust controller design via mu-synthesis.

 Special thanks to Tolemy Nibi our MATLAB Student Ambassador at Arizona State University for authoring this cheat sheet.