# *COSY_PAK* MANUAL OF FUNCTIONS

***by***

**C-K. Chen and N. Sreenath**

**Systems Engineering Department**
**Case School of Engineering**
**Case Western Reserve University**
**Cleveland OH 44106-7070**

**sree@veda.cwru.edu**

## Introduction

All transfer functions used in **COSY_PAK** are scalar unless specified. Some **COSY_PAK** functions return graphic or boolean output. Most other functions return alpha-numeric output in the form of a list. The desired output should be extracted from the returned value if more than one variable is returned. An example is given below.

### Example :

***In[ 11] :*** pout ={{k,l},{m,n,j}};

***In[12] :*** pout[[2]]
***Out[12] :*** {m,n,j}

***In[13] :*** pout[[2,1]]
***Out[13] :*** m

Presently we support only single input single output systems (SISO). Future versions of **COSY_PAK** would incorporate multi-input multi-output systems (MIMO). The general algorithms for these functions can be found in any standard control engineering textbook such as Ogata [1991] (see README file). Some of the functions we have used from the very fine Signal Processing Packages :Copyright: Copyright 1989-1991 by Brian L. Evans, Georgia Tech Research Corporation.

See **README** file for more information on **COSY_PAK.** The functions in this manual are listed according to the **COSY_Notes** notebook chapter.

## Symbol Definition

**gopts:** *Mathematica* graphics options.
**s**: Laplace variable.
**t**: time variable
**Transf(s)**: Transfer function.
**y(x)**: Function of time.

## Chapter 1. Introduction to Control Systems Analysis

**ChekAnal[Transf, s, r, w,showder]:** Checks the analyticity of the transfer function Transf(s) at the point s=r+jw using **Cauchy-Reimann** conditions. If 'showder=1' (optional) then the derivatives in the computation are shown**.**

**CPulse[l,t]:** Defines a pulse which begins at t=0 and ends at t = l. The **CPulse** has value 1 within the range (0,l), 0 outside this range, and 1/2 at the points t=0 and t=l. A continuous-pulse center at the origin is written as **CPulse[l, t + l/2]** or **Shift[-l/2,t][CPulse[l, t]]** (from Brian Evans' Signal Processing package).

**CStep[t]**, a.k.a. **Unit[-1][t]**: The unit step function, which is 1 for t > 0, 0 for t < 0, and 1/2 at t = 0. It is commonly used for continuous expressions t. See also Step and Unit (from Brian Evans' Signal Processing package).

**Delta[expr]:** The Dirac delta function. The area under this functions is 1 but it only has value at the origin. That is, **Integrate[ Delta[t] g[t], {t, t1, t2} ]** is g[0] if t1 <= 0 <= t2, 0 otherwise. It differs from the Kronecker delta function **Impulse[t]** (from Brian Evans' Signal Processing package).

**InvLaPlace[f, s]** and **InvLaPlace[f, s, t]:** Gives the multidimensional bilateral inverse Laplace transform of f. A region of convergence can be specified by using **InvLaPlace[{f, rm, rp}, s, t]**, where rm is **R-** and rp is **R+** in the region (strip) of convergence: **R- < Re(s) < R+**. Note that InvLaPlaceTransform is an alias for InvLaPlace (from Brian Evans' Signal Processing package).

**LaPlace[e, t]** or **LaPlace[e, t, s]:** Gives the two-sided Laplace transform of the expression e, which is a function of t, by returning an object of four slots tagged by **LTransData: <transform>, <rminus>, <rplus>,**

**<laplace_variables>**. The Region of Convergence **(ROC)** is defined as **<rminus> < Re{s} < <rplus>**. Note that the returned **ROC** is either the actual **ROC** or a subset of the actual **ROC**. In two dimensions, **LaPlace[e, {t1, t2}, {s1, s2}]** is the same as **LaPlace [ LaPlace[e, t1, s1], t2, s2 ]**. This notation extends naturally to higher dimensions. Note that the right-sided transform is specified by multiplying the expression by **CStep[t]**. Also, LaPlaceTransform is an alias for LaPlace (from Brian Evans' Signal Processing package).

**LSolve[ diffequ == drivingfun, y[t] ]:** Solves the differential equation **diffequ = drivingfun**, where **diffequ** is a linear constant coefficient differential equation and **drivingfun** is the driving function (a function of t). Thus, **diffequ** has the form **a0 y[t] + a1 y'[t] + ....** One can specify initial values; e.g., **LSolve[ y''[t] + 3/2 y'[t] + 1/2 y[t] == Exp[a t], y[t], y[0] -> 4, y'[0] -> 10 ]**. A differential equation of N terms needs N-1 initial conditions. All unspecified conditions are considered to be zero. LSolve can justify its answers (from Brian Evans' Signal Processing package).

**PoleZeros[Transf, s]:** Computes finite poles and zeros of the transfer function **Transf .** Returns **{list of poles, list of zeros}.**

**SignalPlot[f, {t, start, end}]:** Plots **f(t)** as an one-dimensional, continuous-time function. It will show the real part as solid lines, and the imaginary part as dashed lines. **Delta** functions are plotted as upward pointing arrows.

**SignalPlot[f, {t1, start1, end1}, {t2, start2, end2}]** treats f as a function of two variables **t1** and **t2**. **SignalPlot** supports the same options as Plot for 1-D signals (functions) and **Plot3D** for 2-D signals (functions) (from Brian Evans' Signal Processing package).

## Chapter 2. Mathematical Modeling of Dynamic Systems

**Linearize[ f , zvars , zpoint , vvars , vpoint]:** Gives the linearization of vector function **f[zvars, vvars] = {$f_1$[zvars, vvars], ... ,$f_n$[zvars, vvars]}** around the operating point **zpoint** and **vpoint**. The length of the state variables **zvars** = **[$x_1$,...,$x_n$]** must be equal to the length of the operating point **zpoint** = **[$x_{10}$,...,$x_{n0}$]** and the length of the control variables **vvars** =

**$[v_1,...,v_r]$** must be equal to the length of the operating point **vpoint** = **$[v_{10},...,v_{r0}]$.** Returns the linearized system matrices **A** and **B** as **{A,B}**.

**Ode2SS[lhscoeff, rhscoeff]**: Converts linear ordinary differential equation (ODE) to *state space eqn*. The ODE is in the format:

$$y^{(n)} + a_1 y^{(n-1)} + ... + a_{n-1} y^{(1)} + a_n y = b_0 u^{(n)} + b_1 u^{(n-1)} + ... b_{n-1} u^{(1)} + b_n u$$

where **lhscoeff** = $[1, a_1,...,a_n]$, **rhscoeff** = $[b_0,b_1,...,b_n]$. If $a_i$ or $b_i$ don't exist, use $a_i=0$ or $b_i=0$. The output are Matrix A and Vector B in the state equation: dx/dt = A x + B u. Returns the state space matrices **A** and **B** as **{A,B}**.

**SS2Transf[A,B,C,s]:** This function transforms the state space representation of system **(A, B, C)** to its transfer function representation. Returns the transfer function **transf** as a function of the Laplace variable 's'.

## Chapter 3. Transient Response Analysis

**Response[ transf, Input, s, {TimeVar, StartTime, EndTime}, gopts]:** Plots the output of transfer function '**transf**' with Laplace input signal '**Input**'. The Laplace variable is '**s**'. The output graph uses the variable '**TimeVar**' and starts at '**StartTime**' and ends at '**EndTime**'. Returns the output variable as a function of time '**t**'.

**SecOrder[zeta, wn, t]:** Gives the unit step response value at time instant **t** for a standard second order system $\omega_n^2/(s^2 + 2\zeta\ \omega_n\ s + \omega_n^2)$ with the damping ratio $\zeta$=**zeta** and natural frequency $\omega_n$ =**wn**. Returns instantaneous value of the step response output variable at time instant '**t**'.

## Chapter 4. Steady-State Response Analysis

**Routh[Charpoly, s, z]:** Gives the Routh's table for application of Routh's stability criterion. The parameter **Charpoly** is the characteristic polynomial with variable **s**. The parameter **z** define the symbol to replace the zero value of first column terms if any. The characteristic polynomial is the denominator of the transfer function. Returns Routh's table as an array.

## Chapter 5. Root-Locus Analysis

**RootLocus[Transf, s, {k,kmin,kmax}, gopts]:** Plots the root-locus plot of the transfer function **Transf(s)** with the Parameter **k** varying from **k=kmin** to **k=kmax**. Returns graphics.

## Chapter 6. Frequency-Response Analysis

**MagPlot[Transf, s, {w,wmin,wmax}, gopts]:** Plots the magnitude part of Bode plot of transfer function **Transf(s)** in decibel(dB), from frequency **w**=**wmin** to **w=wmax**, both > 0 and in radians per second. Returns graphics.

**MagvsPhase[Transf, s, {w,wmin,wmax}, gopts]:** Plots the magnitude vs. phase plot of transfer function **Transf(s)** with **s=jw**. The frequency **w** varies from **w=wmin** to **w=wmax** both > 0 and in radians per second. Returns graphics.

**NyquistPlot[Transf, s, {w,wmin,wmax}, gopts]:** Plots the Nyquist plot of the transfer function **Transf(s)**. The plot is composed of three parts. The first part corresponds to **s=jw** with **w=-wmin** to **w=-wmax**. The second part corresponds to **s=jw**, **w** varying from **w=+wmin** to **w=+wmax**. Encirclement information of (-1+j0) is provided by the third part which corresponds to s with theta=-pi/2 to pi/2. **wmin** and **wmax** should be > 0 and radians per second. Returns graphics.

**PhasePlot[Transf, s, {w,wmin,wmax}, gopts]:** Plots the phase part of Bode plot of transfer function **Transf(s)** in degree, from frequency **w**=**wmin** to **w=wmax**; **w** is in radian per second.Returns graphics.

**Polar[Transf, s, {w,wmin,wmax}, gopts]:** Plots the polar plot of the transfer function **Transf(s)** with **s=jw** , **w** varying from **w=wmin** to **w=wmax**, both > 0 and in radians per second. Returns graphics.

# Chapter 7. State Space Analysis Methods

**Controllable[A, B]:** Returns a logic (boolean) value representing the complete state controllability of system **A, B**.

**ExpAt[A]:** Returns exponential matrix of square matrix **A**.

**Observable[A, C]:** Returns a logic (boolean) value representing the complete state observability of system **A, C**.

**OutControllable[A, B, C, D]:** Returns a logic (boolean) value representing the output controllability of system **A, B, C, D**.

**ObsPolePlace[A, C, newpoles]:** Determines state observer gain matrix using Ackermann's formula.

**PolePlaceGain[A, B, newpoles]:** Returns gain matirix **K** to place poles of system **A - BK** at locations specified by **newpoles**. Uses Ackermann's formula.

**SysResponse[A, B, C, x0, input, s, {t, tmin, tmax}, gopts]:** Graphs system output y as a function of time for the system with matrices **A**, **B**, **C**, at initial state **x0** from time **tmin** to **tmax**. Returns the time domain solutions for state x and output y.

# Miscellaneous Linear Algebra Functions

**Note : Some of the functions that we give here have equivalents in *Mathematica* 2.0 and higher.**

**matrixpower[A, n]**: Returns the matrix $A^n$, where n is a positive integer. This function is used by the COSY_PAK functions controllable, observable, placepolegain, and obspoleplace functions.

**rank[A]**: Returns the rank of matrix **A**. The function rank returns the integer value corresponding to the number of linearly independent rows in the matrix A. The rank function is used by the functions observable, controllable, and outcont.

**sspace[a,b]**: This function is equivalent to the other **COSY_PAK** function **ODE2SS**. The function **sspace** returns the single input, single output state space form of the ordinary differential equation such as y''' + a2 y'' + a3 y' + a4 y = b1 u' + b2 u with the coefficients of y and its derivatives given by the list a and the coefficients of u and its derivatives given by the list b. The list a must start with a 1 and the list b should be padded with leading zeroes to make it the same length as a. The A, B, C,and D matrices of the equations

x'= Ax + Bu

y = Cy + Du

are returned as the global variables AOUT, BOUT, COUT, and DOUT, representing a system with scalar input u and output y.

**tpose[A]**: Returns the transpose of matrix **A**.