SUMMARY OF SIGNAL PROCESSING OBJECTS

1. `$ConvolutionDomain` (in "Convolution.m")
   $ConvolutionDomain is the default domain for all of the routines in the Convolution package (except for DTPiecewiseConvolution and CTPiecewiseConvolution which imply a domain). Use SetConvolutionDomain to reset its value.

2. `$DeltaFunctionScaling` (in "SigProc.m")
   The variable $DeltaFunctionScaling dictates how to scale the height of Delta functions for the purposes of plotting: None or Scaled.

3. `$PtolemyProlog` (in "Ptolemy.m")
   $PtolemyProlog is a global variable containing the definitions to make the generated Ptolemy code work properly. The default value will find the file "headerX.X.pt" on Mathematica's path, read it in, and write it out, where X.X is $PtolemyVersion. This is useful if you are running the Ptolemy interpreter on another file system. Otherwise, you can set this value to

   ```
   StringJoin[ "(load \"",
               FindFile["SignalProcessing`ObjectOriented`headerX.X.pt", $Path ],
               "\")" ]
   ```

   where X.X is the version of Ptolemy you are using, which will cause Ptolemy to load the header as the first step in running the simulation.

4. `$PtolemyVersion` (in "Ptolemy.m")
   $PtolemyVersion indicates the current version of Ptolemy being used. Currently, only version 0.3 of Ptolemy is supported.

5. `ADDITIVE` (in "System.m")
   ADDITIVE is a system property. An operator T is additive if T[x + y] = T[x] + T[y] for all x, y. It also serves as a function. ADDITIVE[system] returns True if the operator system is additive. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

6. `AKindOf` (in "Slot.m")
   AKindOf is a slot that describes the class to which the object belongs.

7. `ASPAnalyze` (in "ASPAnalyze.m")
   ASPAnalyze[f, t], ASPAnalyze[f, {t, start, end}], and ASPAnalyze[f, t, start, end] will plot f as a one-dimensional, continuous-time function of t (real part of f will be shown as solid lines, the imaginary part as dashed lines). The routine will also print the strip of convergence,

indicate stability criteria, display the pole-zero diagram, and plot the magnitude and phase responses. The two-dimensional version becomes ASPAnalyze[f, {t1, t2}], ASPAnalyze[f, {t1, start1, end1}, {t2, start2, end2}], or ASPAnalyze[f, {t1, t2}, {start1, start2}, {end1, end2}]. All non-variable symbols will be assigned a value of 1 when the analyzer needs to generate a plot (override this by using options like a -> 2).

8. `ASSOCIATIVE` (in "System.m")
   ASSOCIATIVE is a system property. It also serves as a function. ASSOCIATIVE[system] returns True if the system is associative. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

9. `AddChildToTree` (in "Tree.m")
   AddChildToTree[tree, parent, newchild] adds newchild under every parent in tree.

10. `AddSlot` (in "Slot.m")
    AddSlot[object, writeflag, slot, initvalue] adds the slot to object, where object (a symbol assigned to the object's value) is mutated; initvalue is the initial value assigned to this new slot and writeflag is True if the slot can be overwritten. The object is returned if the slot slot already exists. Note that initvalue can be a collection of values like 1,2,3.

11. `AddT` (in "TransSupport.m")
    AddT[transq, t1, t2] adds the two transforms t1 and t2 together and determines the new region of convergence. The new transform is returned as a list. AddT[transq, t1, t2, lowerlimit] uses lowerlimit as the lower limit on region of convergence values when combining ROC's (default is 0).

12. `AdditiveCosts` (in "CostFunctions.m")
    AdditiveCosts[oper[p1, p2, ..., pn][sig1, sig2, ..., sign]] returns the number of additions needed to implement the operator oper (having parameters p1, p2, ..., pn) in terms of the sampling rate of the input signals sig1, sig2, ..., sign.

13. `Adjoin` (in "Lisp.m")
    Adjoin[item, list] prepends item to list if it is not already in the list.

14. `AlgorithmCostFunction` (in "CostFunctions.m")
    AlgorithmCostFunction[expr, {w1, w2, ..., wn}, {f1, f2, ..., fn}] computes the cost of expr as a weighted sum of cost functions, i.e. sum (wi fi) over i. The cost functions should be able to measure the cost of a general expression.

15. `Aliasby` (in "SigProc.m")
    Aliasby[m,w][x] represents the aliasing of x, a continuous-frequency signal, by replicating it

every (2 Pi / m) in w and dividing the result by m. It is always rewritten in terms of the Periodic operator.

16. `AllEquivalentForms` (in "Heuristics.m")
    AllEquivalentForms[expr, rules] returns a list of all possible ways to rewrite expr according the list of rules. The Max options specifies an upper limit on the number of rewritten forms returned.

17. `AllSubsets` (in "SupCode.m")
    AllSubsets[set] returns a list of all subsets of set, including the null set {}. The original set must be a list.

18. `Analog` (in "FilterSupport.m")
    Analog is continuous-time signal processing.

19. `AnalogFilter` (in "FilterDesign.m")
    AnalogFilter[class, time-variable, order, epsilon, A, transition-ratio, cutoff-frequency] will return a filter expression containing one or more of a gain term, a CFIR section, and a CIIR section. Here, class is one of Bessel, Butterworth, ChebyshevI, ChebyshevII, Elliptic. The last four arguments are filter parameters. The value of the cutoff-frequency defaults to 1. The Bessel filter design procedure, however, only uses the order and the cutoff-frequency and ignores the other arguments. See also FilterParameters and FilterOrder.

20. `AppendSlot` (in "Slot.m")
    AppendSlot[object, slot, appendfun, addvalue] will set the value of the slot slot in object to the value returned by calling appendfun with two arguments: the old value of the slot and addvalue. Note that object is an unevaluated symbol and that addvalue can be a collection of values like 1,2,3.

21. `ApplyHomogeneityRule` (in "TransSupport.m")
    ApplyHomogeneityRule[fun, head, var, rest] applies the homogeneity rule (factor out constants) to function fun for the transform represented by head.

22. `Area` (in "Convolution.m")
    Area[ar] is an object representing the area under a Dirac delta function. A Dirac delta function is usually written as C Delta[t - t0], where C is the area under the delta function and t0 is the location of the delta function. In F-interval form, C Delta[t - t0] becomes { Area[C], t0, t0 }.

23. `ArraySignal` (in "Signal.m")
    ArraySignal[oldsig, start, end] returns a new signal which has slots Start and End equal to start and end, respectively, and is array-based (Implementation slot is Array).

24. `ArraySignalQ` (in "Signal.m")
ArraySignalQ[signal] returns True if the signal is array-based, which means that all signal values are computed at one time and stored.

25. `Arrow2D` (in "SupCode.m")
Arrow2D[tail, plotwidth, plotheight] returns a graphics object that is an arrow starting at tail, pointing upwards, of length plotheight. The length of the tail and the direction of the arrow are controlled by an optional fourth parameter.

26. `AssociateItem` (in "SupCode.m")
AssociateItem[item, lookuplist, newlist] finds the location of item in the lookuplist and returns the element of newlist in that position. If item is not is lookup list, Null is returned. If item is a list, then a list of associations is returned.

27. `Assuming` (in "SupCode.m")
Assuming[condition] keeps track of assumptions made during a calculation. Assuming[All] gives all of the current assumptions. Assuming[condition, True] prints the condition if it is not True. Assuming[] removes all current assumptions.

28. `AutoCorrelation` (in "Convolution.m")
AutoCorrelation[e, v] computes the autocorrelation of expression e (in the form of a list of F-intervals, an F-interval, or a function) with respect to v. The variable v is treated as either discrete and continuous according to the value of the Domain option. The default domain is the value of $ConvolutionDomain.

29. `Bandpass` (in "FilterSupport.m")
Bandpass describes a filter that passes a band of frequencies and rejects all others.

30. `Bandstop` (in "FilterSupport.m")
Bandstop describes a filter that rejects a band of frequencies and passes all others.

31. `Bessel` (in "FilterSupport.m")
Bessel is a type of infinite impulse response (IIR) filter. The step response of a lowpass Bessel function has low overshoot. The transfer function is of the form d0 / Bn(s), where d0 is a constant and Bn(s) is the nth-order Bessel polynomial. The magnitude response of a lowpass Butterworth filter is smooth (no ripples) and monotonically decreases with respect to frequency.

32. `BesselPolynomial` (in "FilterDesign.m")
BesselPolynomial[order, s] results from truncating a continued fraction expansion representation of the unit delay exp(-s). Bessel polynomials satisfy the recursion B[n, s] = (2n - 1) B[n-1, s] + s^2 B[n-2, s], with B[0, s] = 1 and B[1, s] = s + 1.

33. `BezoutNumbers` (in "LatticeTheory.m")
BezoutNumbers[a, b] gives the Bezout numbers of integers a and b. That is, it returns {lambda, mu} so that lambda a + mu b == gcd(a,b). In this case, BezoutNumbers simply calls ExtendedGCD. For the collection of integers a1, a2, ..., an, BezoutNumbers[a1, a2, ..., an] returns a list of integer coefficients k1, k2, ..., kn such that k1 a1 + k2 a2 + ... + kn an = d, where d is the greatest common divisor of a1, a2, ..., an. The solutions of the Bezout identity are periodic. This routine will always return those solutions smallest in absolute value. See also ExtendedGCD.

34. `BilinearTransformation` (in "FilterSupport.m")
BilinearTransformation[omega, w, k] maps a continuous frequency omega (in rad/sec) to a discrete frequency w (in rad) in the interval (0, Pi) by the relationship omega = k Tan[w/2]. BilinearTransformation assumes that two of three arguments have value and will set the value of the unknown parameter. It returns a list of the three arguments. BilinearTransformation[omega, w, k, h, s, z, t] will transform the analog transfer function h = f(s) into a digital transfer function g(z). Now, if h is instead a function of t and not of s, then the Bilinear transformation will be applied to the Laplace transform of h.

35. `BranchOperation` (in "Ptolemy.m")
BranchOperation[token, simpleOper, refs] represents one simple operation simpleOper in the computation of a larger algorithm with the output(s) referenced by the symbol token. The larger algorithm uses the output n times. BranchOperation[token, simpleOper, n, Constant] indicates that the token is an alias for a constant computed according to simpleOper. This object is displayed as token := simpleOper. See ExpressionToBranchOperations.

36. `ButLast` (in "Lisp.m")
ButLast[list] and ButLast[list, n] returns list but drops last n elements (by default, n = 1).

37. `Butterworth` (in "FilterSupport.m")
Butterworth is a type of infinite impulse response (IIR) filter whose magnitude squared response is of the form 1/(1 + (w/wc)^(2 n)) where w is the frequency variable, wc is the cutoff frequency, and n is the filter order. The magnitude response of a lowpass Butterworth filter is smooth (no ripples) and monotonically decreases with respect to frequency.

38. `CConvolve` (in "SigProc.m")
CConvolve[t][x1, x2, ...] represents the continuous convolution in t of functions x1, x2, ... CConvolve[{t1, t2, ...}][x1, x2, ...] represents the multidimensional continuous convolution in t1, t2, ... of functions x1, x2, ...

39. `CFIR` (in "FilterSupport.m")
CFIR[t, {h0, h1, h2, ...}] represents the finite impulse response of an all-zero analog filter.

Input to the filter is specified as FIR[t, h] [x[t]], where x[t] is some symbolic signal processing expression. For the generalized N-dimensional FIR structure, the argument t becomes a list of indices (like {t1, t2} for N=2) and the feedback coefficients are in an N x N matrix. Supports option Roots, which for 1-D filters is a list of zeroes.

40. `CIIR` (in "FilterSupport.m")
CIIR[t, {a0, a1, a2, ...}] represents an all-pole analog filter with gain 1 and DC gain 1/a0. The terms a1, a2, ... are the feedback coefficients. The Laplace transform of such an expression is 1 / (a0 + a1 s + a2 s^2 + ...). Such filters can alternatively be specified by their poles via CIIR[t, Roots -> {-1, -2, -3}] which yields the object CIIR[t, {6., 11., 6., 1}, Roots -> {-1, -2, -3}]. CIIR is also an operator whose default input is a Dirac delta. An alternate input h can be specified by CIIR[t, {a0, a1, a2, ...}][h]. For the generalized N-dimensional IIR structure, the argument t becomes a list of indices (like {t1, t2} for N=2) and the feedback coefficients are specified by an N x N matrix.

41. `COMMUTATIVE` (in "System.m")
COMMUTATIVE is a system property. It also serves as a function. COMMUTATIVE[system] returns True if the system is commutative. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

42. `CONTINUOUS` (in "System.m")
CONTINUOUS is a system property. It also serves as a function. CONTINUOUS[system] returns True if the operator system only takes CONTINUOUS systems as arguments. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

43. `CPulse` (in "SigProc.m")
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]].

44. `CStep` (in "SigProc.m")
CStep[t], a.k.a. Unit[-1][t], is 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. Under Mathematica 2.1 and higher, CStep is defined as UnitStep, but UnitStep will always be displayed as CStep so the change should be transparent to the user. See also Step and Unit.

45. `CTFTData` (in "Fourier.m")
CTFTData is the data-tag for the CTFT.

46. `CTFTransform` (in "Fourier.m")
CTFTransform[f, t] or CTFTransform[f, t, w] computes the continuous-time Fourier transform

of f. The time variable is t, and the frequency variable is w. In multiple dimensions, t and w become lists of time and frequency variables, respectively.

47. `CTPiecewiseConvolution` (in "Convolution.m")
CTPiecewiseConvolution[h, x, t] carries out the one-dimensional, continuous-time piecewise convolution of h and x with respect to t. See PiecewiseConvolution for valid representation of h and x. Using F-interval notation {function, left end point, right end point}, a Dirac delta is meant when function = Area[<area>] (where <area> equals the area of under the impulse function) and left end point = right end point (the location of the impulse). A Dialogue option of True or All will cause the graphs for an animation to be generated in which h is flipped in the variable t and then slid across x. See also PiecewiseConvolution.

48. `CenterOfSymmetry` (in "Signal.m")
CenterOfSymmetry[signal] returns the center-of-symmetry of the signal.

49. `ChebyshevI` (in "FilterSupport.m")
ChebyshevI is a type of infinite impulse response (IIR) filter whose magnitude squared response is of the form 1/(1 + e^2 Tn(w/wc)^2) where e controls passband ripple, w is the frequency variable, and wc is the cutoff frequency. The term Tn(s) is the nth-order Chebyshev polynomial written as ChebyshevT[n, s] in Mathematica. The magnitude response of the stopband monotonically decreases with respect to frequency.

50. `ChebyshevII` (in "FilterSupport.m")
ChebyshevII is a type of infinite impulse response (IIR) filter whose magnitude squared response is of the form 1/(1 + e^2 Xn(w/wc)^2) where e controls stopband ripple, w is the frequency variable, and wc is the cutoff frequency. Here, Xn(w) = Tn(wr) / Tn(wr / w), where Tn(w) is the nth-order Chebyshev polynomial written as ChebyshevT[n, s] in Mathematica. The magnitude response of the passband monotonically decreases with respect to frequency.

51. `CheckSeparability` (in "System.m")
CheckSeparability[op[parameters]] checks returns a list of the variables in which op[parameters] is separable. CheckSeparability[op[parameters], varlist] checks to see in which of the variables in varlist the operator is separable.

52. `CirclePS` (in "SupCode.m")
CirclePS[r] and CirclePS[r, p] will return a graphics object, a circle of radius r with plotstyle p. CirclePS makes use of the Circle primitive.

53. `CircularShift` (in "SigProc.m")
CircularShift[n0, N, n] represents the circular shifting of a sequence of length N in index

variable n by n0. That is, the sequence is treated as a ring buffer of length N so each sample at index n is shifted to the right by (n + n0) mod N places. See also the SPP function Shift and the Mathematica function RotateRight.

54. `ClearQueue` (in “StackQueue.m”)
ClearQueue[q] sets q to be an empty queue.

55. `ClearStack` (in “StackQueue.m”)
ClearStack[s] sets s to be an empty stack.

56. `ColumnHermiteForm` (in “LatticeTheory.m”)
ColumnHermiteForm[F] returns {X, H} where X is the m x m regular unimodular matrix that puts the m x n integer matrix F into upper triangular (column Hermite) form H (which is m x n). That is, X . F = H.

57. `Combine` (in “SupCode.m”)
Combine[object, joindata] sets the value of object to the joining of object with joindata; however, if object has no value, then object is set to joindata.

58. `CommutableResamplingMatricesQ` (in “LatticeTheory.m”)
CommutableResamplingMatricesQ[downsamplingmat, upsamplingmat] returns True if the two resampling operations are commutable.

59. `ComplexQ` (in “Lisp.m”)
ComplexQ[z] returns True if z is a complex number, False otherwise.

60. `ComplexTo2DCoord` (in “SupCode.m”)
ComplexTo2DCoord[x] returns the two-dimensional coordinate corresponding to the complex form of x. That is, a pair of values in the form { Re[x], Im[x] } is returned.

61. `ComplexTo2DCoordList` (in “SupCode.m”)
ComplexTo2DCoordList[zlist] returns a list of coordinates corresponding to the complex form of each element in zlist. That is, zlist is mapped through ComplexTo2DCoord.

62. `ComputableSignalQ` (in “Signal.m”)
ComputableSignalQ[signal] returns True if computational methods like Fetch are supported for the signal. This is true if the signal has an Implementation slot.

63. `ComputePerOutputSamplingRate` (in “CostFunctions.m”)
ComputePerOutputSamplingRate[expr, fun1, fun2, vars] in 1-D simply returns the sampling rate for expr. In m-D, this function first applies fun1 to each column of the sampling matrix for expr in terms of variables vars and then applies fun1 to the resulting vector.

64. `ConjT` (in “TransSupport.m”)
    ConjT[transq, X(s), s] implements T{ x*(t) } –> X*(s*), where X(s) is the transform of x(t).

65. `Cons` (in “Lisp.m”)
    Cons[x, list] prepends x to the list. Cons[x, y] returns {x, y} if y is not a list.

66. `ConstantQ` (in “SupCode.m”)
    ConstantQ[x] returns True if x is always constant. If x is an atom, then x is constant if it is a number or if it x has a Constant attribute. If x is a function of the form f[arg1, arg2, ...], then x is considered to be constant if arg1, arg2, ..., are constant. ConstantQ[x, c1, c2, ...] and ConstantQ[x, {c1, c2, ...}] analyze the expression x assuming that c1, c2, ... are constant.

67. `ConstantTerm` (in “SupCode.m”)
    ConstantTerm[expr, x] returns the constant term of the expression expr.

68. `Continuous` (in “SigProc.m”)
    Continuous is a possible value for a signal’s domain.

69. `ConvertFromList` (in “Convolution.m”)
    FormulaToPiecewise[l, t] converts a list of F-intervals in the form {f, t1, t2} to a signal processing expression. The new expression will contain either Delta, CStep, and CPulse functions (continuous-time) or Impulse, Step, and Pulse functions (discrete-time) according to the value of the Domain option. The default domain is the value of $ConvolutionDomain. The old name for this function is ConvertFromList.

70. `ConvertToList` (in “Convolution.m”)
    PiecewiseToFormula[e, t] converts the expression e to a list of F-intervals in the form {fun, left, right}. The domain of the new expression is established according to the value of the Domain option. The default domain is the value of $ConvolutionDomain. The old name for this function is ConvertToList.

71. `ConvexPolygonIntersection` (in “Polygons.m”)
    ConvexPolygonIntersection[p1, p2, ...] returns the intersection of polygons p1, p2, ... The algorithm exhaustive enumerates all line segments connecting the vertices in the two polygons. This function returns an empty list if there are no intersections.

72. `ConvolutionOutputForm` (in “Convolution.m”)
    ConvolutionOutputForm is an option for CTPiecewiseConvolution PiecewiseConvolution, and DTPiecewiseConvolution. It can take a value of Automatic, Formula, or Piecewise. If the option is set to Automatic, then the output will depend on the form of the inputs: if either input was specified as a formula (or expression), then the output will also be a formula; otherwise, the output will be a list of F-intervals. If the option is set to Piecewise, the output

will be a list of F-intervals no matter the way the input is specified. If it is set to Formula, the output will be an expression no matter the input form.

73. `Convolve` (in "SigProc.m")
    Convolve[n][x1, x2, ...] represents the discrete convolution in n of functions x1, x2, ... Convolve[{n1, n2, ...}][x1, x2, ...] represents the multidimensional discrete convolution in n1, n2, ... of functions x1, x2, ...

74. `ConvolveT` (in "TransSupport.m")
    ConvolveT[transq, convop, t1, t2] implements one dimension of the transform of a convolution, Convolution[All, All, t][x1, x2], where convop is the convolution operator, t1 is the transform of x1, and t2 is the transform of x2.

75. `CorrelationSequence` (in "Correlation.m")
    CorrelationSequence[x,n,z] or CorrelationSequence[X,n,z] will return the correlation sequence as discrete-time function of n. Note that the first argument can be either a sequence x[n] or a z-transform X(z). Also note that the last two arguments are optional.

76. `CostRange` (in "CostFunctions.m")
    symbol/: CostRange[symbol] = range assigns a range of values to the symbol. You can remove the definition by evaluating symbol/: CostRange[symbol] = . .

77. `DFT` (in "SigProc.m")
    DFT[L, n, k] is the forward discrete Fourier transform operator. Here, L is the DFT length(s), n is the time-domain variable(s), and k is the frequency-domain variable(s). Multidimensional DFT's are supported. Applying TheFunction to the DFT operator will invoke the DFT rule base (object DFTransform).

78. `DFTData` (in "DFT.m")
    DFTData is a data tag for the symbolic DFT.

79. `DFTransform` (in "DFT.m")
    DFTransform[f, N, n, k] returns the DFT of sequence f which is a function of the time index n. Sequence f is defined over the interval n = 0 to n = N - 1. Only the first two arguments are required DFTransform calls DTFTransform when it cannot determine the DFT.

80. `DISCRETE` (in "System.m")
    DISCRETE is a system property. It also serves as a function. DISCRETE[system] returns True if the system only accepts DISCRETE signals as arguments. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

81. `DSPAnalyze` (in "DSPAnalyze.m")
DSPAnalyze[f, n] and DSPAnalyze[f, n, start, end] will plot the real values of f[n] from n = Floor[start] to n = Ceiling[end], print its z-transform, indicate conditions for stability, display the pole-zero diagram, and plot the magnitude and phase responses. DSPAnalyze[f, {n1,n2}] and DSPAnalyze[f, {n1,n2}, {s1,s2}, {e1,e2}] treat f as a function of two-variables n1 and n2. The domain of the time-domain graph is n1 from s1 to e1 and n2 from s2 to e2. All non-variable symbols will be assigned a value of 1 when the analyzer generates graphics (override this with options like a -> 2).

82. `DTFT` (in "SigProc.m")
DTFT[n, w] is the forward discrete-time Fourier transform operator. Here, n is the time-domain variable(s), and w is the frequency-domain variable(s). Multidimensional DTFT's are supported. Applying TheFunction to the DTFT operator will invoke the DTFT rule base (object DTFTransform).

83. `DTFTData` (in "DTFT.m")
DTFTData is the data tag for the discrete-time Fourier transform.

84. `DTFTransform` (in "DTFT.m")
DTFTransform[f, n, w] returns the discrete-time Fourier transform of the input function f with respect to n. The result is a function in w that is periodic with period 2 Pi in w. Note that this transform may rely on the z-transform and the substitution z = Exp[I w], which is not valid if the region of convergence does not include the unit circle. For the multidimensional DTFT, n and w becomes lists. Also note that only the first two arguments are required.

85. `DTPiecewiseConvolution` (in "Convolution.m")
DTPiecewiseConvolution[h, x, n] carries out the one-dimensional, discrete-time piecewise convolution of h and x with respect to n. See PiecewiseConvolution for the valid representations of h and x. Using F-interval notation {function, left end point, right end point}, an impulse (Kronecker delta) function is meant when left end point = right end point (the location of the impulse) and the strength of the impulse function is given by function. A Dialogue option of True or All will cause the graphs for an animation to be generated in which h is flipped in the variable n and then slid across x. Real-valued interval end points will be truncated to integers. See also PiecewiseConvolution.

86. `DefaultSlot` (in "Slot.m")
DefaultSlot[object tag, slot, value] will set up the value as the default for the slot in any object with a tag of object tag. For example, to set up 0 as the default value for the slot CurrentValue of a Signal, use DefaultSlot[Signal, CurrentValue, 0].

87. `DefineObjectType` (in "Slot.m")
DefineObjectType[object, parent object] clears the attributes of object and sets the HoldRest attribute.

88. `DefineSystem` (in "System.m")
DefineSystem[operator, property1, property2, ...] will install the operator as a system. The properties property1, property2, ... are considered to be True for the operator. For example, if DefineSystem[Upsample[l, n], DISCRETE, LINEAR] is evaluated, Upsample will be added to the list of system operators SPsystems, and one of the four rules generated is CONTINUOUS[Shift[l,n]] := True. Note that CONTINUOUS[Shift] will have no meaning.

89. `DeleteFromTree` (in "Tree.m")
DeleteFromTree[tree, node] deletes all nodes with info/name of node. If the node is a parent, then the entire subtree is pruned.

90. `DeleteSlot` (in "Slot.m")
DeleteSlot[object, slot] removes the slot slot from object and returns the new object. Note that object is an unevaluated symbol.

91. `Delta` (in "SigProc.m")
Delta[expr] is 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. Under Mathematica 2.1 and higher, Delta is defined as DiracDelta, but DiracDelta will always be displayed as Delta so the change should be transparent to the user. It differs from the Kronecker delta function Impulse[t].

92. `DeltaIntegrate` (in "SigProc.m")
DeltaIntegrate[fun, {x, xlower, xupper}] and DeltaIntegrate[fun, x] are special hooks for the built-in Integrate routine. Under Mathematica 2.1, the rules in Calculus‘DiracDelta‘ take precedence over this function.

93. `DeltaPlot` (in "SigProc.m")
DeltaPlot is a object that can generate a plot of only Dirac delta functions or a combination of Dirac delta functions and another plot. In the first case, either four or six arguments are passed: deltafuns, t, tmin, tmax, ymin, ymax. In the second case, either five or seven arguments are passed: deltafuns, t, tmin, tmax, plot, ymin, ymax. In both cases, ymin and ymax are the optional arguments. DeltaPlot returns a list of plots suitable for sending to Show. Note that 1-D Delta functions are plotted as arrows. The variable $DeltaFunctionScaling dictates how Delta functions are scaled when plotted: None or Scaled.

94. `Dequeue` (in "StackQueue.m")
Dequeue[queue, element] removes element from the queue.

95. `DerivativeT` (in "TransSupport.m")
DerivativeT[transq, expr, s, m] applies the derivative operator to the transform expr which is a function of s. The derivative is taken m times if and only if expr satisfies transq.

96. `DesignDecimationSystem` (in "Decimation.m")
DesignDecimationSystem[Polygon[pointlist]] returns the decimation system to resample the bandpass signal (which is a convex hull represented by the polygon) at the Nyquist rate. The complete decimation system consists of four parts in cascade: modulator, upsampler, filter, and downsampler. The modulator is defined as Exp[I shiftVector . timeIndices]. This routine returns a list: { shiftVector, upsampling matrix, downsampling matrix }. The Mod option controls the grid size and therefore the size of the entries of the resampling matrices. DesignDecimationSystem[ratmat] converts the rational resampling matrix into a decimation system.

97. `DesignFilter` (in "FilterSupport.m")
DesignFilter[domain, class, type, time-variable, deltap, deltas, w1, w2, w3, w4] returns a filter expression (involving IIR/FIR or CIIR/CFIR filter sections). Domain is either Analog or Digital. Possibilities for class include Butterworth, ChebyshevI, ChebyshevII, and Elliptic. Possible filter types are Lowpass, Highpass, Bandpass, and Bandstop. For lowpass filters, the magnitude response of the passband resides on [1, 1 - deltap] and the magnitude response of the stopband occupies [0, deltas]. The last two arguments, w3 and w4, are only necessary for bandpass and bandstop filters. Frequencies w1, w2, w3, and w4 are measured in rad/sec and should be given in ascending order.

98. `DiagonalMatrixQ` (in "LatticeTheory.m")
DiagonalMatrixQ[matrix] returns True if the matrix is diagonal, and gives False otherwise.

99. `Dialogue` (in "SupCode.m")
Dialogue is an option for all of the symbolic transforms, the differential/difference equations solvers, and more. Possible settings are False, True, or All, for no, partial, or full justification, respectively. In the case of symbolic transforms, a setting of True or All will cause the rule base to describe strategies being applied to compute the transform as well as the functions (if any) which it could not transform. If it set to All, then the rule base will also display the intermediate calculations involved in taking the transform.

100. `Difference` (in "SigProc.m")
Difference[k,n] represents the k-th backward difference with respect to the variable n. The first-order difference of f equals f[n] - f[n-1], the second-order difference is f[n] - 2 f[n-1] + f[n-2], and so on. Applying this operator to the sequence x[n] would be written as Difference[k,n][x[n]].

101. `Digital` (in "FilterSupport.m")
Digital is discrete-time signal processing.

102. `Dirichlet` (in "SigProc.m")
Dirichlet[N, w] is the normalized aliased sinc function. which is defined as Sin[N w / 2] / (N Sin [w / 2]). It is periodic in w with a period of 2 Pi for N odd and a period of 4 Pi for N even. Note that Dirichlet[N, 0] is 1. Aliases for Dirichlet are ASinc, AliasSinc, and AliasedSinc.

103. `Discrete` (in "SigProc.m")
Discrete is a possible value for a signal's domain.

104. `DiscreteGraphics` (in "SigProc.m")
DiscreteGraphics[{{x1,y1}, {x2,y2}, ... {xN,yN}}] will plot the points as lines drawn from the x-axis to the y-value. It takes two optional arguments: a line thickness (or plot style) and a point thickness. Each is a real value from 0 to 1 indicating a width that is a fraction of the size of the entire graph.

105. `DiscreteTimeSignalQ` (in "Signal.m")
DiscreteTimeSignalQ[signal] returns True if the signal is a discrete-time signal.

106. `DistinctCMYKColors` (in "Polygons.m")
DistinctCMYKColors[n] will generate a list of n distinct cyan, magenta, yellow, and black colors.

107. `DistinctCosetVectors` (in "LatticeTheory.m")
DistinctCosetVectors[resmat] returns a sorted list of all of the distinct coset vectors associated with the resampling matrix resmat. DistinctCosetVectors[resmat, {U, D, V}] finds all of the distinct coset vectors of resmat from its Smith Form U D V. This is carried by finding the distinct coset vectors of the diagonal matrix D, mapping them by U, and then taking each vector modulo the resampling matrix resmat. The returned coset vectors are not sorted.

108. `DistinctRGBColors` (in "Polygons.m")
DistinctRGBColors[n] will generate a list of n distinct reg, green, and blue colors as distinct form one another as possible but also with as much contrast to white and black as possible.

109. `Domain` (in "SigProc.m")
Domain is an option for many signal processing functions. It is either Discrete or Continuous.

110. `DomainScale` (in "TransSupport.m")
DomainScale is an option for MagPhasePlot indicating the scaling of the independent variable axis, Linear or Log, for the magnitude plot.

111. `Downsample` (in "SigProc.m")
Downsample[m,n] represents the downsampling operator. Downsampling resamples a function at the points n = 0, m, 2m, 3m, ... In N dimensions, n is a list of N indices and m is an N x N matrix. Applying TheFunction to f[n] which is downsampled in one dimension, (written as Downsample[m, n][f[n]]) will produce f[m n].

112. `DownsamplingAliasing` (in "Decimation.m")
DownsamplingAliasing[M, polylist] graphically illustrates the effect of downsampling by the integer matrix M on the frequency domain of the input signal represented as the union of polygons expressed as the list of polygons polylist. The input and output frequency domains, although representing different sampling grids, are both graphed in the fundamental tile -Pi < w1 < Pi and -Pi < w2 < Pi. The Dialogue option controls how much analysis will be performed: False for none, True for some, and All for all.

113. `DummyVariables` (in "SigProc.m")
DummyVariables[dimension, variable] generates dimension dummy variables with variable as the root. For example, n can be generated by DummyVariables[1, n] or DummyVariables[0, n]. As another example, DummyVariables[3, t] returns {t1, t2, t3}. In this cases, the generated variables have the same context as variable. DummyVariables[dimension, variable, context] can be used to specify another context/package.

114. `Element` (in "SupCode.m")
Element[expr, n] returns the nth element of expr if n is an integer or list of integers. Element[expr, n1, n2, ..., nm] returns expr[[n1, n2,..., nm]] if n1, n2, ... nm are integers. Element also works if the indices can be rationalized to be integers, so Element[{1,2,3}, 1.] returns 1.

115. `Elliptic` (in "FilterSupport.m")
Elliptic is a type of infinite impulse response (IIR) filter whose magnitude squared response is of the form 1/(1 + e^2 Rn(w,L)^2) where e controls ripple, w is the frequency variable, and wc is the cutoff frequency. Here, Rn(w, L) is called a Chebyshev rational function such that L is a parameter describing ripple properties. Elliptic filters have equiripple in the passbands and stopbands.

116. `Elt` (in "Lisp.m")
Elt[list, n] returns the nth element in the list with n=0 indicating the first element.

117. `EmptyQ` (in "SupCode.m")
EmptyQ[packet] returns True the packet of data is empty.

118. `EndBandwidth` (in "Signal.m")
EndBandwidth[signal] retuns the end location of a signal's bandwidth. In the case of signals that are not band-limited, this function will return Infinity (the default).

119. `Enqueue` (in "StackQueue.m")
Enqueue[queue, element] adds element to the queue.

120. `EuclidFactors` (in "LatticeTheory.m")
EuclidFactors[k, p, q] gives the factors a and b such that p a + q b == k for relatively prime p and q. When p and q are integers, this routine searches over a set of min(|p|, |q|) integers. When p and q are integer matrices, this routine searches over a set of min(|det(p)|, |det(q)|) integer vectors and the vector a or b corresponding to the integer matrix with the smallest determinant will always be a distinct coset of that matrix. See also RelativelyPrime.

121. `Every` (in "Lisp.m")
Every[query, expr] returns True if every element of expr satisfies the test query. Every[query, expr, levelspec] returns True if every part of the expression expr satisfies the test query.

122. `ExpressionToBranchOperations` (in "Ptolemy.m")
ExpressionToBranchOperations[expr] decomposes a nested Mathematica expression (tree) into a list of simpler (branch) operations. The full calling sequence is ExpressionToBranchOperations[expr, pullConstants, gainflag, plusflag, a1, a2, ...]. If pullConstants is True, then this routine will combine constants within a product (if gainflag is True) and a summation (if incflag is True) into one constant term. The arguments a1, a2, ... indicate symbols in expr to be treated as constants (see ConstantQ). Definitions of these constants are not included as branch operations because their values are unknown. See BranchOperation.

123. `Extent1D` (in "SigProc.m")
Extent1D[expression, var] returns an interval {lower-limit, upper-limit} outside of which the function is zero. This routine assumes that var is a real variable. The routine returns None if the function is zero-valued for all var.

124. `FIR` (in "FilterSupport.m")
FIR[n, {h0, h1, h2, ...}] represents the finite impulse response of an all-zero digital filter. As a function of n, the response is actually equal to h0, h1, h2, ..., for n = 0, 1, 2, ... Note that FIR[n, 2] will be simplified to 2 Impulse[n]. FIR can also represent an all-zero filter. In this case, the input x[n] to the filter is specified as FIR[n, h] [x[n]], where x[n] is some symbolic signal processing expression. For the generalized N-dimensional FIR structure, the argument n becomes a list of indices (like {n1, n2} for N=2) and the feedback coefficients are in an N x N matrix.

125. `FT` (in "SigProc.m")
FT[t, w] is the forward continuous Fourier transform operator. Here, t is the time-domain variable(s) and w is the frequency-domain variable(s). Applying TheFunction to this operator will invoke the continuous-time Fourier transform rule base (object CTFTransform).

126. `FVariables` (in “DSupport.m”)
FVariables is a data tag for the transform variables in a Fourier transform.

127. `FactorRationalResamplingMatrix` (in “Decimation.m”)
FactorRationalResamplingMatrix[ratmat] factors the non-singular rational matrix into two matrix factors L^-1 M where L and M are both non-singular integer matrices.

128. `Fetch` (in “Signal.m”)
Fetch[signal, index] returns the value of the signal at index, where index can be a number (for one-dimensional signals) or a list of numbers (for multidimensional signals). Fetch only works for computable signals which are arrayed, pointwise, and stream signals. Stream signals also have an associated method called FetchNext. If the signal is arrayed and if any one of the indices is an infinity, then the value of the signal's OutOfDomainValue slot will be returned.

129. `FetchArray` (in “Signal.m”)
FetchArray[array signal] ...

130. `FetchNext` (in “Signal.m”)
FetchNext[stream signal] will return the next value in the stream of data represented by stream signal.

131. `FilterOrder` (in “FilterSupport.m”)
FilterOrder[filter-type, transition-ratio, nu] returns the filter order for the lowpass filter that meets the transition ratio (for digital filters, see BilinearTransformation) and the filter parameter nu (which is a function of the passband and stopband ripple parameters– c.f. FilterParameters).

132. `FilterParameters` (in “FilterSupport.m”)
FilterParameters[delta1, delta2, epsilon, A, Eripple, ATT, nu] fills in the values of the missing parameters. It assumes that either the passband ripple (delta1) and the stopband ripple (delta2) or epsilon (controls ripple) and A (1 / delta2) are given. The values of Eripple, ATT, and nu will be computed and assigned to the symbols passed into those slots. Note that the passband takes on values between 1 and (1 - delta1) and that the stopband takes on values between 0 and delta2. The function returns a list of all seven parameters.

133. `FilterTransform` (in “FilterDesign.m”)
FilterTransform[filter-domain, lowpassfilter, s, new-filter-type, wc, wcnew, wcnewupper] transforms a lowpass filter with cutoff frequency wc into either a lowpass or highpass filter with a cutoff of wcnew or a bandpass or bandstop filter with lower/upper cutoff frequencies of wcnew and wcnewupper. Here, the new-filter-type can be Lowpass, Highpass, Bandpass, or Bandstop.

134. `FindBoundingRegion` (in "Decimation.m")
FindBoundingRegion[majorAxis, polygon] finds the smallest rectangle enclosing the convex hull represented by polygon with the line majorAxis as the rectangle's major axis.

135. `FindFile` (in "SupCode.m")
FindFile[filename, dirlist] looks at the directories in dirlist to find the file filename. The filename can be a context (i.e. one containing backquotes). It returns $Failed if it cannot find the file. A common use of this function is FindFile[filename, $Path].

136. `FindMinimumRectangle` (in "Decimation.m")
FindMinimumRectangle[Polygon[pointlist]] finds the rectangle of minimum area that circumscribes the convex hull represented as a polygon. The rectangle, which may be rotated about an axis, is returned as a polygon.

137. `FindRMinus` (in "ROC.m")
FindRMinus[rminus1, rminus2] and FindRMinus[rminus1, rminus2, lowerlimit] returns the greater of the two coordinates rminus1 and rminus2. This function and FindRPlus[] are used to find the intersection of two regions of convergence.

138. `FindRPlus` (in "ROC.m")
FindRPlus[rplus1, rplus2] and FindRPlus[rplus1, rplus2, lowerlimit] returns the lesser of the two coordinates rplus1 and rplus2. This function and FindRMinus[] are used to find the intersection of two regions of convergence.

139. `Finish` (in "DFT.m")
Finish is a data-tag for the DFT.

140. `Formula` (in "Convolution.m")
Formula is an option for the piecewise convolution routines. See ConvolutionOutputForm for more information.

141. `FourierSimplify` (in "DSupport.m")
FourierSimplify[x] applies exponential to trigonometric rewrite rules to expression x.

142. `FunctionToPtolemy` (in "Ptolemy.m")
FunctionToPtolemy[head] converts a Mathematica function like Cos to Ptolemy star.

143. `GCLD` (in "LatticeTheory.m")
GCLD[a, b] finds a greatest common left divisor of resampling matrices a and b which is unique to within the product of a regular unimodular matrix on the right. It forms F = {{a, b}, {0, 0}} and solves the equation F . X = H where X is a square regular unimodular matrix and H is the row Hermite (lower-triangular) form of the rectangular matrix F. Of the four

submatrices of H, only H11 is non-zero. The GCLD of a and b is H11. This routine supports a Dialogue option. See also LCLM, RowHermiteForm, and ResamplingMatrix.

144. `GCRD` (in "LatticeTheory.m")
GCRD[a, b] returns a greatest common right divisor of resampling matrices a and b which is unique to within the product of a regular unimodular matrix on the left. It forms F = {{a, 0}, {b, 0}} and solves the equation X . F = H where X is a square regular unimodular matrix and H is the column Hermite (upper-triangular) form of the rectangular matrix F. Of the four submatrices of H, only H11 is non-zero. H11 is the GCRD. This routine supports a Dialogue option. See also LCRM, ColumnHermiteForm, and ResamplingMatrix.

145. `GenerateCode` (in "SupCode.m")
GenerateCode[object] converts object to a string (if necessary) and then evaluates it (translates it to Mathematica code).

146. `GeneratePattern` (in "SupCode.m")
GeneratePattern[namestring] generates a symbol with the name equal to namestring followed by an underscore, which makes the object be a pattern which can be used for pattern matching. For example, GeneratePattern[a] yields the pattern (a).

147. `GenerateSymbol` (in "SupCode.m")
GenerateSymbol[namestring], GenerateSymbol[namestring, trailer], and GenerateSymbol[namestring, trailer, header], generate a symbol by concatenating header, namestring, and trailer.

148. `GetAllExponents` (in "SupCode.m")
GetAllExponents[expr, x] returns all exponents of the term x in the expression expr. GetAllExponents[z^3 + 2 z^6, z] returns {6, 3}.

149. `GetAllFactors` (in "SupCode.m")
GetAllFactors[expr, x] returns all factors of the term x in the expression expr. GetAllFactors[z^3 + 2 z^6, z] returns {1, 2}. GetAllFactors[z^-3 + 2 z^-6, z] returns {1/2, 1}.

150. `GetAllShiftFactors` (in "SupCode.m")
GetAllShiftFactors[expr, x] returns a list containing all shift factors in the variable x. The expression is traversed as is using a depth-first search. For example, GetAllShiftFactors[(2 s + 2 b + c) Exp[s + b + Pi], s] returns { b + Pi, (2 b + c)/2 }.

151. `GetArgs` (in "SupCode.m")
GetArgs[function] returns the argument(s) of the function. For example, GetArgs[Bogus[1,2,3]] would return 1,2,3. If the programmer only wishes to replace the head of a function with another, then use Apply instead of GetArgs. This function is similar to ToCollection.

152. `GetDeltaFunctions` (in "SigProc.m")
GetDeltaFunctions[function, variable(s)] will extract all delta functions from the argument function. See also DeltaPlot.

153. `GetOperatorVariables` (in "SupCode.m")
GetOperatorVariables[op] returns the variable(s) in the parameterized operator op. By default, GetOperatorVariables[op[par1, par2, ..., parn]] returns the last parameter, parn.

154. `GetRMinus` (in "ROC.m")
GetRMinus returns the value of the Rminus component of the region of convergence contained in the passed argument. So, GetRMinus[ZTransData[x,rm,rp,z]] returns the value of rm; GetRMinus[Rminus[x]] returns x; and GetRMinus[x] returns x.

155. `GetRPlus` (in "ROC.m")
GetRPlus returns the value of the Rplus component of the region of convergence contained in the passed argument. GetRPlus[ZTransData[x,rm,rp,z]] returns the value of rp; GetRPlus[Rplus[x]] returns x; and GetRPlus[x] returns x.

156. `GetRoot` (in "SupCode.m")
GetRoot[rule] extracts the value from an expression like {z -> 0.}, which is 0. in this case.

157. `GetRootList` (in "SupCode.m")
GetRootList[p, x] returns a list of the approximate numerical roots of expression p, a function of x, with infinite roots removed. GetRootList[p, x, filter] applies filter to the list of roots returned by the Solve function (defaults to N). When GetRootList cannot find a set of roots, it will search for numeric roots.

158. `GetShiftFactor` (in "SupCode.m")
GetShiftFactor[expr, x] returns a list containing the common shift factor in the variable x and a normalized version of expr. For example, GetShiftFactor[(2 s + 2 b + c) Exp[s + b + Pi], s] returns {b, 2 Exp[Pi + b + s] (b + c/2 + s)}.

159. `GetStateField` (in "SupCode.m")
GetStateField[state, field] returns the value of the slot field n the list state.

160. `GetValue` (in "SupCode.m")
GetValue[f[n], n, n0] finds the numeric value of f[n] at n = n0 and GetValue[f[n1,n2], {n1,n2}, {n01, n02}] finds the numeric value of f[n1,n2] at n1 = n01 and n2 = n02. When the first argument has the variables embedded in it, two arguments are sometimes enough: GetValue[object, n0]. This is true when the object is an abstract signal.

161. `GetVariables` (in "SupCode.m")
GetVariables[expr] returns a list of all of the variables in the expression expr. See VariableQ for the definition of a variable.

162. `HOMOGENEOUS` (in "System.m")
HOMOGENEOUS is a system property. An operator T is homogeneous if T[a x] = a T[x] if a is a scalar. It also serves as a function. HOMOGENEOUS[system] returns True if the system is homogeneous. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

163. `HasAttributes` (in "SupCode.m")
HasAttributes[symbol, attribute1, attribute2, ...] returns True if the evaluation of symbol is another symbol and the attributes to be checked are a subset of the attributes of this other symbol. HasAttributes[Plus, {Listable, Orderless}] would return True.

164. `HasSlotQ` (in "Slot.m")
HasSlotQ[object, slot] returns True if the slot slot exists in object.

165. `Highpass` (in "FilterSupport.m")
Highpass describes a filter that passes all frequencies above a cutoff frequency and rejects all others.

166. `IIR` (in "FilterSupport.m")
IIR[n, {a0, a1, a2, ...}] represents the infinite impulse response of an all-pole filter with gain 1, where a1, a2, ..., are the negated feedback coefficients and a0 is the scaling factor of the output. In discrete time, the output of IIR is the solution to y[n] in the difference equation a0 y[n] + a1 y[n-1] + ... + aN y[n - N] = impulse[n], with y[n] = 0 for n < 0. By default, the input into the filter (a.k.a. the forcing function) is an impulse. An alternate input h can be specified by the code IIR[n, {a0, a1, a2, ...}][h]. For the generalized N-dimensional IIR structure, the argument n becomes a list of indices (like {n1, n2} for N=2) and the feedback coefficients are specified by an N x N matrix.

167. `IIRCoefficients` (in "Signal.m")
IIRCoefficients is a special slot for infinite-impulse response digital filters. It simply contains a list of feedback coefficients.

168. `IIRFilter` (in "Signal.m")
IIRFilter[p,a0,a][x] calculates the output of the IIR filter at discrete time n, where the input at time n is x. The feedback coefficients are in the a vector and each is scaled by a0. The list p is a table of the last L outputs, where L is the number of feedback coefficients.

169. `IIRFunction` (in "FilterSupport.m")
IIRFunction[n, coeffs] and IIRFunction[n, coeffs, input] generate the object IIRFunction[n, coeffs, input, TheFunction -> hiir] that represents a digital IIR filter with feedback coefficients coeffs (see IIR for meaning) and input function input. When n is an integer or real-valued integer, IIRFunction will return the value of the filter computed at that sample. For negative n, the value of IIRFunction is zero but the value of hiir is the value of the input function (this allows initial conditions to be specified in the input function). Here, hiir is the actual recursive object that computes values of the IIR filter only as needed. To see what values have already been computed, type ??<hiir slot>.

170. `IIRPostUpdate` (in "Signal.m")
IIRPostUpdate is a special method just for IIR signals.

171. `ImaginaryQ` (in "SupCode.m")
ImaginaryQ[z] returns True if z is a number whose real part is zero.

172. `Impulse` (in "SigProc.m")
Impulse[n] is the unit Kronecker delta function. At n = 0, the function evaluates to 1. Elsewhere, it evaluates to 0.

173. `ImpulseTrain` (in "LatticeTheory.m")
ImpulseTrain[D, n] represents an infinite collection of Kronecker impulse functions. In one-dimension, D and n are integers. The multidimensional separable form is when both D and n are integer vectors of the same length. In the non-separable case, D is a resampling matrix and n is an integer vector. This function gives 1 whenever D^-1 n is an integer (vector), and gives 0 otherwise. See also ResamplingMatrix and Impulse.

174. `InRange` (in "SupCode.m")
InRange[a, b, c, amin, cmax, leftcompare, rightcompare] returns True if b in between a and c. The inclusiveness of the interval a to c is determined by the arguments leftcompare and rightcompare, each of which defaults to LessEqual. So, InRange[a, b, c] returns True if a <= b <= c. Non-numeric values, like Infinity, can be used for amin and cmax, which default to -Infinity and +Infinity, respectively.

175. `IncList` (in "LatticeTheory.m")
IncList[list, start, end] will increment the first element of list. If this becomes greater than the first element of end, then the next element of list will be incremented, and so forth. For example, IncList[{9,0,0}, {0,0,0}, {10,10,10}] would return {0,1,0} and IncList[{9,9,9}, {0,0,0}, {10,10,10}] would return {10,10,10}. This is useful when enumerating values over an n-dimensional rectangular prism.

176. `IndependentVariable` (in "Signal.m")
IndependentVariable[signal] returns the independent variable(s) involved in the signal's (functional) definition.

177. `InfCheck` (in "ROC.m")
InfCheck[expr] returns Infinity if expr is any Infinity form; otherwise, expr is returned.

178. `InfinityQ` (in "SupCode.m")
InfinityQ[a] will return True if a is Infinity, -Infinity, ComplexInfinity, DirectedInfinity[], or DirectedInfinity[r].

179. `InformUserQ` (in "SupCode.m")
InformUserQ[x] returns True if the options in object x contain Dialogue -> All or Dialogue -> True. It also returns True if x is All or True.

180. `InitTree` (in "Tree.m")
InitTree[root] returns an empty tree with a root of root.

181. `InputPortName` (in "Ptolemy.m")
InputPortName[op] indicates the name of the signal input port in Ptolemy for the operator op. By default, the function returns input .

182. `InsidePolygonQ` (in "Polygons.m")
InsidePolygonQ[Polygon[vertices], Point[coordinates]] returns true if the point is inside the polygon, and False otherwise. A third argument is a flag, which defaults to False, indicating what to return if the point lies on an edge of the polygon.

183. `IntegerVectorQ` (in "LatticeTheory.m")
IntegerVectorQ[arg] returns True if arg is a vector whose elements are integers.

184. `IntegrateT` (in "TransSupport.m")
IntegrateT[transq, t, variable, lower-limit, upper-limit] integrates the transform t (with respect to variable) from lower-limit to upper-limit. The resulting transform is returned as a list.

185. `Interleave` (in "SigProc.m")
Interleave[n][x0, x1, ...] interleaves samples of signals x0, x1, ..., which are functions of n. This is only applicable to discrete signals.

186. `Intern` (in "Lisp.m")
Intern[string] and Intern[string, package] creates a new symbol with the same name as string in the package given (the current package is used in no package is given).

187. `IntersectsROC` (in "ROC.m")
Intersects[rm1,rp1,rm2,rp2] returns True if the interval (rm1,rp1) overlaps with the interval (rm2,rp2) in any way. Therefore, IntersectsROC will return False when the two intervals only overlap at endpoints.

188. `IntervalQ` (in "Convolution.m")
IntervalQ[x] returns True if the argument x is an interval, and gives False otherwise. An interval has the form {f, t1, t2} such that none of f, t1, or t2 are lists. See also ValidIntervalQ.

189. `IntervalsToFunction` (in "SigProc.m")
IntervalsToFunction[interlist, n, step, pulse] will translate the list of functions and intervals (interlist) into a signal expression as a function of n. Each element in the interlist is a list of three elements: <function of n>, <n minus>, and <n plus>. The function <function of n> will be defined for <n minus> <= n <= <n plus>. Note that <n minus> can be an integer or -Infinity and that <n plus> can be an integer or Infinity. The default values are n = Global`n, step = Step, and pulse = Pulse. This function also works for when n represents a continuous variable (step = CStep and pulse = CPulse).

190. `InvCTFTransform` (in "Fourier.m")
InvCTFTransform[f, w, t] computes the inverse continuous time Fourier transform of f with frequency variable w and time variable t. In multiple dimensions, w and t become lists of frequency and time variables, respectively.

191. `InvDFT` (in "SigProc.m")
InvDFT[L, k, n] is the inverse discrete Fourier transform operator. Here, L is the DFT length(s), n is the time-domain variable(s), and k is the frequency-domain variable(s). Multidimensional inverse DFT's are supported. Applying TheFunction to the InvDFT operator will invoke the inverse DFT rule base (object InvDFTransform).

192. `InvDFTransform` (in "DFT.m")
InvDFTransform[f, N, k, n] takes the inverse DFT of sequence f which is a function of n that is defined over the interval n = 0 to n = N - 1. Only the first two arguments are required. InvDFTransform calls InvDTFTransform as a last resort.

193. `InvDTFT` (in "SigProc.m")
InvDTFT[w, n] is the inverse discrete-time Fourier transform operator. Here, n is the time-domain variable(s), and w is the frequency-domain variable(s). Multidimensional inverse DTFT's are supported. Applying TheFunction to the InvDTFT operator will invoke the inverse DTFT rule base InvDTFTransform.

194. `InvDTFTransform` (in "DTFT.m")
InvDTFTransform[f, w, n] returns the inverse discrete-time Fourier transform with respect

to w of the f. The function f is assumed to be periodic in w. The first two arguments are required. For the multidimensional version, w and n are lists. By default, the inverse DTFT rule base will apply the definition of the inverse DTFT when all other attempts to find the inverse have failed.

195. `InvFT` (in "SigProc.m")
InvFT[w, t] is the inverse discrete Fourier transform operator. Here, t is the time-domain variable(s) and w is the frequency-domain variable(s). Multidimensional Fourier transforms are supported. Applying TheFunction to this operator will invoke the inverse continuous-time Fourier transform rule base InvCTFTransform.

196. `InvL` (in "SigProc.m")
InvL[s, t] is the inverse Laplace transform operator. Applying TheFunction to the InvL operator will invoke the bilateral inverse Laplace rule base InvLaPlace.

197. `InvLaPlace` (in "InvLaPlace.m")
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+. In multiple dimensions, s and t become lists of Laplace and time variables, respectively. Note that InvLaPlaceTransform is an alias for InvLaPlace.

198. `InvZ` (in "SigProc.m")
InvZ[z, n] is the inverse z-transform operator. Applying TheFunction to the InvZ operator will invoke the inverse z-transform rule base InvZTransform.

199. `InvZTransform` (in "InvZTransform.m")
InvZTransform[e,z] or InvZTransform[e,z,n] gives the inverse z-transform of e. Here, e can be a function of z, a list, or a z-transform object (see ZTransData). As a list, e should contain three elements: F[z], rminus, and rplus, such that the region of convergence is rminus < |z| < rplus. If e is a z-transform object, then the inverse z-transform is simply InvZTransform[e] because e contains all necessary information. This routine returns signal processing expressions. Convert them to mathematical formulas by using TheFunction. The inverse transform of large expressions will be found much faster if automatic simplification is disabled by using the option Simplify -> False.

200. `InvalidInvLTransformQ` (in "LSupport.m")
InvalidInvLTransformQ[trans] returns True if the inverse Laplace transform rule base did not successfully finish the transform.

201. `InvalidInvZTransformQ` (in "ZSupport.m")

InvalidInvZTransformQ[x] returns True if the result from a call to InvZTransform was not a valid inverse z-transform. This occurs when the object myinvz exists in expression x.

202. `InvalidLTransformQ` (in "LSupport.m")
InvalidLTransformQ[trans] returns True if the forward Laplace transform rule base did not successfully finish the transform.

203. `InvalidZTransformQ` (in "ZSupport.m")
InvalidZTransformQ[x] returns True if a call exists to the function MyZTransform in expression x; that is, True is returned if the result from a call to Z or ZTransform was not a valid Z transform.

204. `KVariables` (in "DFT.m")
KVariables is a slot in the DFTData data structure which indicates the DFT frequency variables.

205. `KeepNormalized` (in "ZSupport.m")
KeepNormalized[zexpr, fun, z, args] keeps the z-transform expression zexpr in negative powers of z after the function fun is applied to zexpr and args as fun[zexpr, args].

206. `L` (in "SigProc.m")
L[t, s] is the forward Laplace transform operator. Applying TheFunction to the L operator will invoke the bilateral Laplace rule base.

207. `LCLM` (in "LatticeTheory.m")
LCLM[a, b] returns a least common left multiple of resampling matrices a and b which is unique to within the product of a regular unimodular matrix on the left. It forms F = {{a, 0}, {b, 0}} and solves the equation X . F = H where X is a regular unimodular matrix and H is the column Hermite (upper-triangular) form of the rectangular matrix F. Of the four submatrices of H, only H11 is non-zero. The LCLM of a and b is (X21 . a) = -(X22 . b). This routine supports a Dialogue option. See also GCRD, ColumnHermiteForm, and ResamplingMatrix.

208. `LCRM` (in "LatticeTheory.m")
LCRM[a, b] finds a least common right multiple of resampling matrices a and b which is unique to within the product of a regular unimodular matrix on the right. It forms F = {{a, b}, {0, 0}} and solves the equation F . X = H where X is a regular unimodular matrix and H is the row Hermite (lower-triangular) form of the rectangular matrix F. Of the four submatrices of H, only H11 is non-zero. The LCRM of a and b is (a . X12) = -(b . X22). This routine supports a Dialogue option. See also GCLD, RowHermiteForm, and ResamplingMatrix.

209. `LForm` (in "LSupport.m")

LForm[transform] returns True if the transform is a valid Laplace transform with a head of Transform or List.

210. `LINEAR` (in "System.m")

LINEAR is a system property. A system is linear if it is both HOMOGENEOUS and ADDITIVE. It also serves as a function. LINEAR[system] returns True if the operator system is linear. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

211. `LINEARPHASE` (in "System.m")

LINEARPHASE is a system property. A system exhibits linear phase if the phase component of its frequency response is an affine function of the frequency. It also serves a a function. LINEARPHASE[system] returns True if the system is memoryless. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

212. `LMultiDROC` (in "LSupport.m")

LMultiDROC[old-Laplace-transform, new-Laplace-transform] combines the transform functions, regions of convergence, and the transform variables in order to produce a multidimensional transform.

213. `LSolve` (in "LSolve.m")

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.

214. `LTransData` (in "ROC.m")

LTransData is a data tag for a valid Laplace transform object.

215. `LTransformQ` (in "LSupport.m")

LTransformQ[x] returns True if the datum x is of the form LTransData[X, rm, rp, s], where X is the LaPlace transform, rm is the Rminus component of the region of convergence, rp is the Rplus component of the region of convergence, and s is the variable(s) in the LaPlace transform.

216. `LVariables` (in "ROC.m")

LVariables is a slot in a Laplace transform object indicating the Laplace variables used in the transformation.

217. `LaPlace` (in “LaPlace.m”)
    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.

218. `LineImpulse` (in “SigProc.m”)
    LineImpulse[nlist, coefflist] represents a line impulse where nlist is a list of variables like {n1,n2,n3} and coefflist is a corresponding list of coefficients like {1,2,3}. For the previous lists, the line impulse is a set of impulse along the line n1 = 2 n2 = 3 n3.

219. `LineImpulsemDT` (in “TransSupport.m”)
    LineImpulsemDT[transq, expr, s, slist, nleft, fun] applies rules for evaluating multidimensional transforms which involve line impulses, like the z-transform of f[n1,n2] Impulse[n1 - n2] or the Laplace transform of f[t1,t2] Delta[t1 - t2]. In either case, the impulse can be removed by setting n1=n2 or t1=t2, which yields a one-dimensional function. The resulting 1-D transform is then altered by the rule z = z1 z2 or s = s1 + s2, respectively. Here, the argument fun is Times for the z-transform and Plus for the Laplace transform. Note that these rules are only applied if expr is a valid transform, which occurs if transq[expr] evaluates to True.

220. `Linear` (in “TransSupport.m”)
    Linear is a possible value for axis scaling. See MagPhasePlot.

221. `ListQ` (in “Lisp.m”)
    ListQ[expr] gives True if expr is a List, and False otherwise.

222. `ListReduce` (in “Lisp.m”)
    ListReduce[f, list], ListReduce[f, list, fromEndFlag], and ListReduce[f, list, fromEndFlag, initialValue] mimic the Lisp reduce routine. For example, ListReduce[Plus, {1, 2, 3, 4}, False, 0] gives (((1 + 2) + 3) + 4) or 10. Mathematica’s Fold command is used to implement ListReduce.

223. `Lowpass` (in “FilterSupport.m”)
    Lowpass describes a filter that passes all frequencies below a cutoff frequency and rejects all others.

224. `MEMORYLESS` (in “System.m”)
    MEMORYLESS is a system property. A system is memoryless if each output corresponding

to a given input only depends on the input (and not past values of the input). It also serves a a function. MEMORYLESS[system] returns True if the system is memoryless. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. A MEMORYLESS system with a single input is also SHIFTINVARIANT. See also DefineSystem and SystemQ.

225. `MagPhasePlot` (in "TransSupport.m")
MagPhasePlot[freqresp, {w, wmin, wmax }] plots the magnitude and phase response over the specified range of frequencies. It returns a list of two elements: the magnitude plot and the phase plot. The two-dimensional version has the form MagPhasePlot[freqresp, {w1, wmin1, wmax1}, {w2, wmin2, wmax2}]. The default options are initially biased for continuous-time frequency responses of digital signals. MagnitudePhasePlot is an alias for MagPhasePlot.

226. `MagRangeScale` (in "TransSupport.m")
MagRangeScale is an option for MagPhasePlot indicating the scaling of the dependent variable axis, Linear or Log, for the magnitude plot. A setting of Null disables the generation of the magnitude plot.

227. `MakeContinuousSignal` (in "Signal.m")
MakeContinuousSignal[] returns a signal object with the slot CONTINUOUS set to True.

228. `MakeDigitalSignal` (in "Signal.m")
MakeDigitalSignal[] returns a signal object with the slot DISCRETE set to True.

229. `MakeLObject` (in "LSupport.m")
MakeLObject[l] and MakeLObject[l, slist] convert the LaPlace transform information in list l into the standard object form LTransData[x, Rminus[rm], Rplus[rp], LVariables[slist]]. Here x is the LaPlace transform, rm is the lower bound on the strip of convergence, rp is the upper bound on the strip of convergence, and slist is the list of LaPlace variables used in the transform.

230. `MakeObject` (in "Slot.m")
MakeObject[object] returns a packet of data tagged with object. Note that an object must at most have the attribute of HoldRest. See DefineObjectType.

231. `MakeSeparable` (in "System.m")
MakeSeparable[operator[parameters][f]] will assume that the operator is separable in the parameters and decompose the operation into a series of nested one-dimensional operators on f. For parameters that are matrices, only their diagonal elements will be extracted. See SEPARABLE.

232. `MakeSignal` (in "Signal.m")
MakeSignal[] returns a signal object with one slot for system use. MakeSignal[var] returns a signal object with slot IndependentVariable set to var and with one slot defined for system use. MakeSignal[signal, n, property1, value, property2, value, ...] can be used to define a signal as a function of n (the properties are optional). If the signal is multidimensional, n will be a list of variables (one for each dimension, like {n1, n2, n3} in the three-dimensional case). Possible signals include Impulse[n], Step[n], FIR[n, {h0, h1, ...}], and IIR[n, {a0, a1, ...}]. See MyMakeSignal.

233. `MakeZObject` (in "ZSupport.m")
MakeZObject[l] and MakeZObject[l, zlist] converts the z-transform information in list l into the standard object form ZTransData[x, Rminus[rm], Rplus[rp], ZVariables[zlist]]. Here, x is the z-transform, rm is the lower bound on the region of convergence, rp is the upper bound on the region of convergence, and zlist is the list of z variables used in the transform.

234. `MemoryCosts` (in "CostFunctions.m")
MemoryCosts[oper[p1, p2, ..., pN][sig1, sig2, ..., sigN]] returns the number of memory elements needed to implement the operator oper (having parameters p1, p2, ..., pN) in terms of the sampling rate of the input signals sig1, sig2, ..., sigN.

235. `MixedPolynomialQ` (in "SupCode.m")
MixedPolynomialQ[p] and MixedPolynomialQ[p,x] return return True if p is a polynomial in negative and positive (mixed) powers of x. Note that rational numbers like 5/6 and 1 are polynomials. MixedPolynomialQ[x + x^-1, x] is True.

236. `MoveOperatorsToFront` (in "System.m")
MoveOperatorsToFront[head operator, signals, signals to move, operator to move] will move all operator to move operators contained in signals to move to the front of the expression formed by the head operator operating on signals. For example, MoveOperatorsToFront[Convolve[n], { x[n], Shift[2,n][y[n]] }, { Shift[2,n][y[n]] }, Shift] yields Shift[2,n] [Convolve[n] [x[n], y[n]]]. This is a valid rewriting of the original expression. In this case, Convolve[n] [x[n], Shift[2,n][y[n]]]. This object is used in the system rewrite rules to rewrite properly an operator operating on more than one signal.

237. `MultT` (in "TransSupport.m")
MultT[transq, t1, t2] multiplies the transforms t1 and t2 together. MultT[transq, t1, t2, lowerlimit] uses lowerlimit as the lower limit on the region of convergence (ROC) when combining the ROC's of t1 and t2.

238. `MultiDInvTransform` (in "TransSupport.m")
MultiDInvTransform[expression, transvar, timevar, options, transq, transform, makeobject,

tdefault] finds the multidimensional inverse transform of expression by one call to transform per dimension. Here, transq, transform, and makeobject are function heads.

239. `MultiDLookup` (in "TransSupport.m")
MultiDLookup[options, timevars, transformvars] expands any multidimensional transform pairs into a set of one-dimensional transform pairs.

240. `MultiDTransform` (in "TransSupport.m")
MultiDTransform[makefun, transform, transtest, expr, expr-vars, def-trans-var, trans-vars] finds the multidimensional transform by calling transform once per dimension. If a call to transform produces an incomplete (invalid) transform, then the multidimensional transform does not exist and this routine is exited. The dimension of the transform is determined from the number of expr-vars (which equals the number of trans-vars). Here, expr is transformed.

241. `MultiplicationCosts` (in "CostFunctions.m")
MultiplicationCosts[oper[p1, p2, ..., pN][sig1, sig2, ..., sigN]] returns the number of multiplications needed to implement the operator oper (having parameters p1, p2, ..., pN) in terms of the sampling rate of the input signals sig1, sig2, ..., sigN.

242. `MyApart` (in "SupCode.m")
MyApart[rational polynomial, x] decomposes the rational polynomial into a sum of fractions whose numerators are of the form (x + b)^n where b is a constant and n is an integer. MyApart uses GetRootList to find the roots and then calls Apart. MyApart[rational polynomial, x, filter] specifies a filter for GetRootList: Identity for rational and N for approximate roots. In Mathematica 1.2, MyApart is about 25 times slower than Apart.

243. `MyCollectAll` (in "SupCode.m")
MyCollectAll[expression, var] attempts to collect all subexpressions of expression in terms of var.

244. `MyFreeQ` (in "SupCode.m")
MyFreeQ[expr, form], when form is not a list, yields True if no subexpression in expr matches form. If form is a list, then True is returned if expr is free of each element of form. This is similar to MyFreeQ[expr, form1, form2, ...] which expands to MyFreeQ[expr, form1] and MyFreeQ[expr, form2] and ....

245. `MyMakeSignal` (in "Signal.m")
MyMakeSignal[signal, var] extends the possibilities of MakeSignal. It allows the definition of standard signals as a function of var. Signals defined this way include FIR[n, h0, h1, ...] and IIR[n, a0, a1, ...].

246. `MyMessage` (in "SupCode.m")
MyMessage[message-label, return-value, arg1, arg2, ...] first calls Message[message-label, arg1, arg2, ...] and then returns return-value.

247. `NextFun` (in "Signal.m")
NextFun is a method for stream-based signals describing how to compute the next value.

248. `NormalizeSamplingMatrix` (in "LatticeTheory.m")
NormalizeSamplingMatrix[sampmat] will decompose the sampling matrix sampmat into a diagonal matrix D and a normalized sampling matrix V. The results are returned as { D, V }.

249. `Nth` (in "Lisp.m")
Nth[n, list] returns the nth element in the list with n=0 indicating the first element.

250. `NthCdr` (in "Lisp.m")
NthCdr[n, list] returns the part of the list from the nth element to the end of the list, with n=0 indicating the first element.

251. `NullPlot` (in "SupCode.m")
NullPlot is a 2-d graphics object which only contains the origin.

252. `OperatorInOperExpr` (in "SigProc.m")
OperatorInOperExpr[operator expression] returns the full operator in operator expression. For example, Shift[2,n] would be returned from OperatorInOperExpr[Shift[2,n][x[n]]].

253. `OperatorName` (in "SigProc.m")
OperatorName[operator expression] returns the head of the expression. For example, OperatorName[Shift[2,n][x[n]]] will return Shift.

254. `OperatorToPtolemy` (in "Ptolemy.m")
OperatorToPtolemy[op[parameters]] converts an operator to a Ptolemy star.

255. `OutOfDomainValue` (in "Signal.m")
OutOfDomainValue[signal] is the value to return if the signal is accessed outside of its region of support. This value defaults to zero.

256. `OutputPortName` (in "Ptolemy.m")
OutputPortName[op] indicates the name of the signal output port in Ptolemy for the operator op. By default, the function returns output .

257. `OverlappingRegions` (in "Polygons.m")
OverlappingRegions[r] enumerates each pair of non-identical regions in r and returns the union

of all overlapping regions as a list of polygons (an empty list means no overlap). OverlappingRegions[region1, region2] returns a list of polygons that represents the overlap between region1 and region2. Multiple overlapping regions may be returned. Remove redundant regions with the Union command.

258. `ParametersInOperExpr` (in "SigProc.m")
ParametersInOperExpr[operator expression] returns the parameters of the operator in operator expression. For example, (2,n) would be returned from ParametersInOperExpr[Shift[2,n][x[n]]].

259. `PartitionIntoSubsets` (in "SupCode.m")
PartitionIntoSubsets[s, n] returns all of the ways set s can be partitioned into n subsets as a tree.

260. `PastValues` (in "Signal.m")
PastValues is a slot that remembers previously computed signal values.

261. `PatternQ` (in "SupCode.m")
PatternQ[expr] returns True if the head of expr is Pattern.

262. `PerOutputSamplingRate` (in "CostFunctions.m")
PerOutputSamplingRate[inputrate, blocksize(s), numsamples, var(s)] computes the sampling rate per output sample for block operations given the input sampling rate (matrix) inputrate, the size of blocks in each dimension blocksize(s), the total number of samples processed numsamples, and the variable(s) var(s) on which the block operation is performed.

263. `Periodic` (in "SigProc.m")
Periodic[k,n][f] indicates that f is a function of n which is periodic with period of k.

264. `PhaseRangeScale` (in "TransSupport.m")
PhaseRangeScale is an option for MagPhasePlot indicating the scaling of the dependent variable axis, Linear or Log, for the phase plot. A setting of Null disables the generation of the phase plot.

265. `Piecewise` (in "Convolution.m")
Piecewise is an option for the piecewise convolution routines. See ConvolutionOutputForm for more information.

266. `PiecewiseConvolution` (in "Convolution.m")
PiecewiseConvolution[f, g, v] convolves the piecewise functions f and g with respect to the variable v. In this context, a function is represented in a piecewise fashion: (1) as F-intervals of the form {fun, left, right}, (2) as a list of F-intervals, or (3) as an expression. An F-interval

has the form {function, left end point, right end point}. The F-interval notation represents a finite-extent function or sequence when the end points do not equal infinity. The result of convolution is always returned as a list of F-intervals. The variable v is treated as either discrete and continuous according to the value of the Domain option. The default domain is the value of $ConvolutionDomain, which can be reset by SetConvolutionDomain. See also CTPiecewiseConvolution and DTPiecewiseConvolution.

267. `PlotList` (in "Convolution.m")
PiecewisePlot[l, {v, vstart, vend}] plots the piecewise function l vs. v. The variable v is treated as either discrete and continuous according to the value of the Domain option. The default domain is the value of $ConvolutionDomain. The old name for this function is PlotList.

268. `PlusByConstant` (in "Ptolemy.m")
PlusByConstant[a, const] indicates the sum of a and the constant const.

269. `PointwisePlot` (in "SupCode.m")
PointwisePlot[coordlist, text] and PointwisePlot[coordlist, text, multiplicitytext] will plot the coordinates in coordlist as text for 2-D and 3-D graphics. An optional fourth argument specifies the size of the font to use. For multiple occurrences of the same coordinate, the object multiplicitytext is displayed. The last two arguments are usually symbols, numbers, or FontForm objects.

270. `PointwiseSignal` (in "Signal.m")
PointwiseSignal is a type of signal in which signal values are computed point-by-point. Such a signal does not depend on previous outputs or inputs.

271. `PointwiseSignalQ` (in "Signal.m")
PointwiseSignalQ[signal] returns True if the values of the signal are computed point-by-point, and the value at each point is independent of other values of the signal.

272. `PoleZeroPlot` (in "TransSupport.m")
PoleZeroPlot[transform-object] plots the poles and zeros of the transform function as well as the region of convergence. For the z-transform, this function will also plot the unit circle and shade the region of convergence (whenever the ROC does not cover the entire z plane). To plot a transfer function f, use PoleZeroPlot[f, transform-variable, rminus, rplus, zdomainflag]. In the two-dimensional case, the transform-variable, rminus, and rplus are two-element lists. The two-dimensional pole-zero plot has three cases: separable transform, symmetric transform, and root map. The function returns the poles of the transform if the Return option is Roots or the pole-zero plot(s) if the Return option is Graphics.

273. `PolygonArea` (in "Polygons.m")

PolygonArea[Polygon[{p0, p1, ...}]] will compute the area of the polygon with vertices p0, p1, ...

274. `PolygonToTriangles` (in "Polygons.m")
PolygonToTriangles[polygon] will slice the polygon into triangles by connecting each vertex with the centroid.

275. `PolyphaseDownsample` (in "SigProc.m")
PolyphaseDownsample[m, h, n][x] is equivalent to downsampling x by m (w/r to n) and then filtering (w/r to n) by a polyphase form of the FIR filter h. Hence, the filtering is carried out at the lower sampling rate.

276. `PolyphaseResample` (in "SigProc.m")
PolyphaseResample[l, h, m, n][x] represents an efficient polyphase structure to perform sampling rate changes by a rational factor. The structure is equivalent to first upsampling by l, then filtering by the FIR filer h, and finally downsampling by m. In the efficient structure, the filtering is carried out at the lower sampling rate. The resampling factors (matrices) l and m must be relatively prime.

277. `PolyphaseUpsample` (in "SigProc.m")
PolyphaseUpsample[l, h, n][x] is equivalent to filtering (w/r to n) x by a polyphase form of the FIR filter h and then upsampling the result by l (w/r to n). Hence, the filtering is carried out at the lower sampling rate.

278. `Pop` (in "StackQueue.m")
Pop[stack] removes and returns the value at the top of the stack.

279. `PostUpdateFunction` (in "Signal.m")
PostUpdateFunction is a method which is applied to signal after each computation of a new signal value.

280. `PreUpdateFunction` (in "Signal.m")
PreUpdateFunction is a method which is applied to signal before each computation of a new signal value.

281. `PrintIt` (in "SupCode.m")
PrintIt[graphics, printer] will print out graphics on a printer. If the printer is not specified, the default printer is used.

282. `PrintName` (in "Signal.m")
PrintName[signal] is the name of the signal which is what is output on the screen when the signal is to displayed.

283. `ProtectIt` (in "SupCode.m")

ProtectIt[expr] evaluates expr. If it evaluates to a symbol, that symbol will be write protected. Rules can be written for that symbol, but values can no longer be assigned to it.

284. `Ptolemy` (in "Ptolemy.m")

Ptolemy is a block diagram simulation environment developed at the Department of Electrical Engineering and Computer Science at the University of California at Berkeley. It features the ability to simulate algorithms that mix synchronous data flow, dynamic data flow, and discrete event methods of computation. The environment can also generate C and assembly language programs. See also PtolemyProgram and PtolemyForm.

285. `PtolemyForm` (in "Ptolemy.m")

PtolemyForm[expr] generates the Ptolemy code that is equivalent to the signal processing expression expr. It is called by PtolemyProgram to generate self-contained Ptolemy simulations. See also Ptolemy and PtolemyProgram.

286. `PtolemyProgram` (in "Ptolemy.m")

PtolemyProgram[expr, {v1, v1min, v1max}, {v2, v2min, v2max}, ...] generates a complete Ptolemy program for simulation based on the processing of in variables v1, v2, ... over the specified range of values. The following options are supported: Dialogue (see Dialogue), Prolog (see $PtolemyProlog), and Plot (True or Automatic will plot the result of the algorithm expr). For example, the user can add aa -> 2 to specify that the constant aa in expr be assigned the value 2 in the generated Ptolemy code. See also Ptolemy and PtolemyForm.

287. `Pulse` (in "SigProc.m")

Pulse[len] and Pulse[len, n] define a pulse which begins at n = 0 and ends at n = len - 1. A discrete pulse centered at the origin is written as Pulse[len, n + (len - 1)/2] or Shift[-(len - 1)/2, n][Pulse[len, n]]. Sometime, a pulse will be automatically simplified; e.g., Pulse[0, n] is zero and Pulse[1, n] is really Impulse[n].

288. `Push` (in "StackQueue.m")

Push[stack, element] adds the element to the top of the stack.

289. `QueryTheUser` (in "SupCode.m")

QueryTheUser[expression, head, function] asks the user to provide a value for the function applied to expression (where the head of the expression is head). The information will be entered permanently in the Mathematica environment because the following piece of code will be generated: head/: function[expression] := value.

290. `Queue` (in "StackQueue.m")

Object head for a queue structure.

291. `ROCPlot` (in “TransSupport.m”)
ROCPlot[{rm, rp}] will plot the region of convergence as a shaded annular region. Possible options for ROCPlot are the same as those for the Show (or Plot) command.

292. `Radian` (in “TransSupport.m”)
Radian is a possible value for axis scaling. See MagPhasePlot.

293. `RationalChebyshev` (in “FilterDesign.m”)
RationalChebyshev[n, x, xL, L] returns a rational polynomial which is the nth order rational Chebyshev function evaluated at x for parameters xL and L.

294. `RationalFunctionQ` (in “SupCode.m”)
RationalFunctionQ[f,x] returns True if expression f is of the form f = g(x) / h(x), where h(x) depends on x but g(x) does not have to depend on x. For example, 1 / (x + 1) is a rational function in x but x^3 + x^2 + x / (x + 1) is not.

295. `RationalGCD` (in “SigProc.m”)
RationalGCD[e1, e2, ...] returns the greatest common divisor of the numerators of elements ei divided by the greatest common divisor of the denominators of elements ei. Every element must be a polynomial or a number. Each real-valued and complex-valued element is rationalized before GCD's are computed.

296. `RationalPolynomialQ` (in “SupCode.m”)
RationalPolynomialQ[p] and RationalPolynomialQ[p,x] return True if p is a rational polynomial in x. Note that rational numbers like 5/6 and 1 are also rational polynomials.

297. `RationalQ` (in “Lisp.m”)
RationalQ[m] returns True if m is a rational number. If m is an integer, then this function also return True, since the set of integers are a subset of rationals.

298. `ReadOnlyList` (in “Slot.m”)
ReadOnlyList is the slot which keeps track of the slots/methods which can not be altered.

299. `ReadOnlySlotQ` (in “Slot.m”)
ReadOnlySlotQ[object, slot or method] returns True if the slot or method can not be altered in the object.

300. `ReadSlot` (in “Slot.m”)
ReadSlot[object, slot] returns the contents of the slot slot in object. Note that object is an unevaluated symbol.

301. `ReadTextFile` (in “SupCode.m”)
ReadTextFile[filename] reads a string that is the entire text file. To write it out in the same form in which it was read, use OutputForm[ReadTextFile[filename]] >> filename.copy .

302. `RealOrComplex` (in "Signal.m")
RealOrComplex[signal] returns True if the signal's output values are real-valued or complex-valued.

303. `RealQ` (in "Lisp.m")
RealQ[z] returns True if z is a floating-point number (has a head of Real), False otherwise. See RealValuedQ.

304. `RealValuedQ` (in "SupCode.m")
RealValuedQ[z] gives True if z is a number whose imaginary component is 0, and gives False otherwise. See RealQ.

305. `RearrangePolygon` (in "Polygons.m")
RearrangePolygon[polygon] rearranges the listing of vertices in order to place the vertex that has the smallest x-coordinate among those that have the smallest y-coordinates as the first vertex.

306. `RegUnimodularResMatrixQ` (in "LatticeTheory.m")
RegUnimodularResMatrixQ[mat] gives True if mat is a resampling matrix with a determinant of +1 or -1, and gives False otherwise. See also ResamplingMatrix.

307. `RelativelyPrime` (in "LatticeTheory.m")
RelativelyPrime[a, b] returns True if a and b are relatively prime, and gives False otherwise. The data types of a and b must be the same. Possible data types are integers, polynomials, and square integer matrices (a.k.a. resampling matrices). For non-diagonal resampling matrices, a third argument of Right or Left is necessary to direct the matrix factorization to occur on the right (using GCLD) or left (using GCRD). See also GCD, PolynomialGCD, and ResamplingMatrix.

308. `RemoveOptions` (in "SupCode.m")
RemoveOptions[optionlist, options] removes the options from optionlist.

309. `ReorderResampling` (in "LatticeTheory.m")
ReorderResampling[resmat, vars, allvars] rewrites the resampling matrix resmat and the variables vars associated with it so that the resampling is carried out by the order of variables specified by allvars. Therefore, vars must be a subset of allvars.

310. `ReplaceWith` (in "SupCode.m")
ReplaceWith[oldexpr, newexpr] is a generalized way to specify a substitution when the substitution may be either atomic and a list of substitutions.

311. `ResamplingMatrix` (in "LatticeTheory.m")
ResamplingMatrix[arg] returns True if arg is a square matrix whose elements are integers

and whose determinant is non-zero. It returns False if arg is not a square matrix. It also returns False if arg is a square matrix with one element being a number that is not an integer. Otherwise, it returns the conditions for which the square matrix would be a resampling matrix.

312. `ResamplingMatrixMod` (in "LatticeTheory.m")
ResamplingMatrixMod[n, N] implements the integer vector n modulo a sampling matrix N which is defined as n - N . Floor[N^-1 . n]. The operation will be carried out if and only if n is an integer vector, N is (or could be) a sampling matrix, and N . n is a valid operation.

313. `Rev` (in "SigProc.m")
Rev[n][x] represents the operation of reversing the direction of x, with respect to the variables(s) n.

314. `RewriteSummations` (in "SigProc.m")
RewriteSummations[f, t, tlower, tupper] rewrites Summation and Periodic operators that appear in f so that the domain of the new expression includes t in [tlower, tupper]. It relies on the Extent1D function to compute the domain of the arguments to the Periodic and Summation operators.

315. `RightSided` (in "ZSolve.m")
RightSided is an option for ZSolve. If True, then the solution to the difference equation will be right-sided, left-sided otherwise.

316. `Rminus` (in "ROC.m")
Rminus[<r-minus>] is the head for the R- value in a transform.

317. `RootLocus` (in "TransSupport.m")
RootLocus[f(z), z, {freeparam, start, end, step}] plots the root locus of f(z) with respect to freeparam over the range of start to end at increments of step. In order to connect the points, the function f(z) will be factored symbolically with respect to z. All closed-form factors of reasonable size will be graphed as parametric plots. RootLocus will generate three plots: the pole zero plot for freeparam = start, the root locus, and the pole-zero plot for freeparam = end. The function returns a list of all three plots. RootLocus supports the same options as does Show. RootLocusPlot is an alias of RootLocus.

318. `RotateAxes` (in "SigProc.m")
RotateAxes[a, {x,y}][f] represents the rotation of the multi-dimensional function f in the (x,y) plane about the origin by a radians. Applying the object TheFunction to the RotateAxes operator will generate a mathematical expression for the rotated function.

319. `RowHermiteForm` (in "LatticeTheory.m")
RowHermiteForm[F] returns {X, H} where X is the n x n regular unimodular matrix that

puts the m x n integer matrix F into lower triangular (row Hermite) form H (which is m x n). That is, F . X = H.

320. `Rplus` (in "ROC.m")
Rplus[<r-plus>] is the head for the R+ value in a transform.

321. `RuleAppliesQ` (in "SupCode.m")
RuleAppliesQ[expr, rule] returns True if rule applies to expr. RuleAppliesQ[head[expr1, expr2, ..., exprn], rule, head] returns True if the rule applies to each expression expr1, expr2, ..., exprn.

322. `SEPARABLE` (in "System.m")
SEPARABLE[system] is a system property. It identifies the components of multidimensional systems that can decomposed into cascades of one-dimensional operations. SEPARABLE[system] returns True if the operator system is separable in all variables, False is separable in no two variables, or a list of variables in which it is separable. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

323. `SHIFTINVARIANT` (in "System.m")
SHIFTINVARIANT is a system property. It means that a shift in the input causes the same shift in the output. It also serves as a function. SHIFTINVARIANT[system] returns True if the operator system is shift-invariant. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

324. `SPHeuristicRewrite` (in "Heuristics.m")
SPHeuristicRewrite[expr, rules] tries to rewrite expressions on the same level of expr together using the list of rules.

325. `SPIntegrationRules` (in "SigProc.m")
SPIntegrationRules is a collection of integral rules that augment the abilities of Mathematica's Integrate command.

326. `SPLessGreaterRules` (in "SupCode.m")
SPLessGreaterRules are a collection of rules for rewriting expressions involving inequalities. They are not included in the SPSimplificationRules.

327. `SPRecursiveRewrite` (in "Heuristics.m")
SPRecursiveRewrite[expr, rules] only returns a fully rewritten version of the signal processing expression expr. The Rewrite knowledge base is let loose on the expression expr without any guidance, but an upper limit on the number of iterations can be set through the MaxIterations

option. The user can see the intermediate expressions by setting the Dialogue option to True or All.

328. `SPSimplificationRules` (in “SupCode.m”)
SPSimplificationRules are a collection of simplification rules that are not carried out by Simplify. These rules require too much overhead to encode them directly into Mathematica. See also SPSimplify.

329. `SPSimplify` (in “SigProc.m”)
SPSimplify[e] will simplify expression e according to Mathematica’s simplification rules and the rules in SPSimplificationRules. At each iteration, Simplify is called and SPSimplificationRules is applied to the result. Unlike Simplify, SPSimplify supports a Dialogue option. You can specify which symbols to treat as real variables by using the Variables option.

330. `SPfunctions` (in “SupCode.m”)
SPfunctions maintains a current list of those new routines that have been loaded from the signal processing packages.

331. `SPoperators` (in “SupCode.m”)
SPoperators maintains a current list of the new mathematical operators that have been loaded from the signal processing packages.

332. `SPproperties` (in “System.m”)
SPproperties contains a list of currently recognized system properties like LINEAR.

333. `SPsignals` (in “SupCode.m”)
SPsignals maintains a current list of those new signals (mathematical functions) that have been loaded from the signal processing packages.

334. `SPsystems` (in “System.m”)
SPsystems is a list of all recognized systems. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. New ones can be defined by using DefineSystem.

335. `SameFormQ` (in “SupCode.m”)
SameFormQ[pattern, expr1, expr2, ...] returns True if every expression matches pattern via MatchQ. Once an expression does not match, this function immediately returns False.

336. `SamplingRate` (in “CostFunctions.m”)
SamplingRate[expr, var] returns the sampling rate (matrix) of the expression expr with respect to the discrete variable (variables) var. SamplingRate assumes that the input signals for multi-input operators have been sampled at the same rate.

337. `ScaleAxis` (in “SigProc.m”)
ScaleAxis[l,w][x] represents the compression by a factor of l (an integer) of the continuous w axis of x, which is usually a continuous-frequency signal.

338. `ScaleT` (in “TransSupport.m”)
ScaleT[transq, t, c] multiplies the transform t by c while leaving the region of convergence unaltered. The resulting transform is returned as a list.

339. `ScalingFactor` (in “SigProc.m”)
ScalingFactor[expr, variable] returns the greatest common divisor of all coefficients of powers of z in expr. The powers of z that are functions of z are not considered. For example, ScalingFactor[a z + a^2 z^2, z] returns a. This function enables the implementation of the similarity property for transforms.

340. `Second` (in “Lisp.m”)
Second[list] returns the second element of list.

341. `Self` (in “Signal.m”)
Self is a pre-defined slot that is bound to the entire object of which it is a member. It can be used as a variable in methods and slots.

342. `SequencePlot` (in “SigProc.m”)
SequencePlot[f, {n, start, end}] plots real-valued 1-D sequences and has the same options as Show. You will have to apply Re or Im to f to plot the real or imaginary values of the sequence, respectively. SequencePlot[f, {n1, start1, end1}, {n2, start2, end2}] plots 2-D sequences and supports the same options as Plot3D.

343. `SequenceToFunction` (in “SigProc.m”)
SequenceToFunction[seq], SequenceToFunction[seq, n], and SequenceToFunction[seq, n, noffset] return the sequence seq, {x[noffset], ..., x[noffset + N - 1]}, in symbolic form as a sum of impulse functions. For example, SequenceToFunction[{1, 2, 3}] becomes Impulse[n] + 2 Impulse[n - 1] + 3 Impulse[n - 2]. For multidimensional sequences, the argument n must be a list of variables of appropriate length. For example, SequenceToFunction[{{1,2},{3,4}}, {n1, n2}] returns Impulse[n1,n2] + 2 Impulse[n1,n2-1] + 3 Impulse[n1-1,n2] + 4 Impulse[n1-1, n2-2].

344. `SetConvolutionDomain` (in “Convolution.m”)
SetConvolutionDomain[domain] establishes the default convolution domain as either Continuous or Discrete. It resets the global variable $ConvolutionDomain. It returns domain on success, and Null on error.

345. `SetExclusion` (in "SupCode.m")
SetExclusion[set1, set2, ...] returns a set equal to the union of the sets minus the intersection of the sets.

346. `SetExclusiveOr` (in "Lisp.m")
SetExclusiveOr[set1, set2, ...] returns a set equal to the union of the sets minus the intersection of the sets.

347. `SetStateField` (in "SupCode.m")
SetStateField[state, field, value] will return a new state, which is a copy of the list state except that the value of the slot field will be equal to value.

348. `ShadedAnnulus` (in "TransSupport.m")
ShadedAnnulus[rm, rp] and ShadedAnnulus[rm, rp, angulartilt] will create a 2-D graphics object that is an annulus (rm < radius < rp) filled with uniformly spaced horizontal lines rotated by angulartilt degrees.

349. `Shift` (in "SigProc.m")
Shift[m,v][x] represents the shifting of the expression x by m samples to the right in the v direction, i.e. x[v] –> x[v - m].

350. `Signal` (in "Signal.m")
A signal is an object with head Signal. From the work of Cory Myers, possible slots include: Bandwidth, CenterOfSymmetry, End, EndBandwidth, Period, Start, StartBandwidth, Support, and Symmetry. Possible additional slots are: CurrentValue, DataType, Dimensions, FrequencyVariables, Implementation, Length, OutOfDomainValue, TheFunction, TimeVariables, ZTransData, ZVariables, CurrentIndex, NextFun, PreUpdateFunction, and PostUpdateFunction. Multidimensional signals are supported. See MakeSignal.

351. `SignalCleanup` (in "SigProc.m")
SignalCleanup[expr, t, nstart, nend] converts the signal expression expr in variables t into a normal Mathematica expression that can be plotted over the interval [nstart, nend]. It returns a list of two expressions: the first consists of the Delta functions extracted from expr, and the second is the rest of the expression.

352. `SignalPlot` (in "SigProc.m")
SignalPlot[f, {t, start, end}] will plot 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).

353. `SignalQ` (in "Signal.m")
SignalQ[x] returns True if the head of x is Signal.

354. `SignalsInOperExpr` (in "SigProc.m")
SignalsInOperExpr[operator expression] returns the signal(s) of the operator in operator expression. For example, { x[n] } would be returned from SignalsInOperExpr[Shift[2,n][x[n]]]. Another version SignalsInOperExpr[operator expression, operator] behaves as the first version if operator is the head of operator expression; otherwise, operator expression is returned.

355. `SimplifyList` (in "Convolution.m")
SimplifyList[interval list, v] takes a list of F-intervals each having the form {f, t1, t2} and returns a simplified sorted list of F-intervals. The interval list is treated as either a continuous-time or a discrete-time function according the value of the Domain option. The variable v is treated as either discrete and continuous according to the value of the Domain option. The default domain is the value of $ConvolutionDomain.

356. `Sinc` (in "SigProc.m")
Sinc[t] is the unit sinc function which is the ratio of Sin[t] over t. Note that Sinc[0] = 1.

357. `SmithChart` (in "SmithChart.m")
SmithChart displays a Smith Chart by issuing the command Show[SmithChartGraphics, DisplayFunction:>$DisplayFunction]. Function written by William Anklam.

358. `SmithChartGraphics` (in "SmithChart.m")
SmithChartGraphics returns the -Graphics- primitives required to plot a Smith Chart. SmithChartGraphics is meant to be used within the Show[] command function so that the Smith Chart can be displayed along with data. Function written by William Anklam.

359. `SmithChartPlot` (in "SmithChart.m")
SmithChartPlot[y, {f, fmin, fmax}] plots the impedance function y on a Smith chart. Function written by William Anklam.

360. `SmithEqualizedForm` (in "LatticeTheory.m")
SmithEqualizedForm[A] returns a list of three matrices U, D, and V. The m x n resampling matrix A equals U D V where U (m x m) and V (n x n) are integer matrices with determinants of +1 or -1 and D is a diagonal integer matrix such that |det(D)| = |det(A)|. The elements of D will be approximately equal to one another. With one argument, this routine will return a U matrix whose columns are approximately orthogonal to one another (by calling SmithLLLForm to perform the initial Smith form decomposition). SmithEqualizedForm[A, {U, D, V}] equalizes the diagonal elements of the diagonal matrix D.

361. `SmithFormMinimizeU` (in "LatticeTheory.m")
SmithFormMinimizeU[U, D, V] will make the regular unimodular U matrix as close to an identity matrix as possible. SmithFormMinimizeU[U, D, V, k, l] will make the kth and lth columns of the regular unimodular matrix U as close to the kth and lth columns of an identity matrix. In both cases, the minimization occurs in a least-squares sense without altering the diagonal matrix D. This routine supports a Dialogue option. This routine directly computes and then returns the new {U, D, V}.

362. `SmithFormMinimizeV` (in "LatticeTheory.m")
SmithFormMinimizeV[U, D, V] will make the regular unimodular V matrix as close to an identity matrix as possible. SmithFormMinimizeU[U, D, V, k, l] will make the kth and lth columns of the regular unimodular matrix V as close to the kth and lth rows of an identity matrix. In both cases, the minimization occurs in a least-squares sense without altering the diagonal matrix D. This routine supports a Dialogue option. This routine directly computes and then returns the new {U, D, V}.

363. `SmithFormPivot` (in "LatticeTheory.m")
SmithFormPivot[U, D, V, k, l, f] pivots a factor f from the lth diagonal element of D to the kth diagonal element of D. The function returns a list of the new U, D, and V. The pivot can only take place if f is a factor of the lth element of D and if f is relatively prime to Dkk Dll' / gcd(Dkk, Dll')^2 where Dkk is the kth diagonal element of D and Dll' is the lth diagonal element of D divided by f. An optional seventh argument specifies the minimization in a Frobenius norm sense— Left for U, Right for V, and None for none. The default is None.

364. `SmithFormSameU` (in "LatticeTheory.m")
SmithFormSameU[smithfun, s1, s2] computes the Smith Form of the integer matrix m1 so that m1 = u1 d1 v1. The routine then uses u2 = u1 to decompose m2. The first argument specifies which decomposition routine should be used to decompose m1 (e.g., SmithNormalForm). The routine returns {cond, {u1, d1, v1}, {u2, d2, v2}}, where cond is True of False depending on whether or not matrices m1 and m2 can be decomposed using the same U.

365. `SmithFormSameV` (in "LatticeTheory.m")
SmithFormSameV[smithfun, m1, m2] computes the Smith Form of the integer matrix m1 so that m1 = u1 d1 v1. The routine then uses v2 = v1 to decompose m2. The first argument specifies which decomposition routine should be used to decompose m1 (e.g., SmithNormal-Form). The routine returns {cond, {u1, d1, v1}, {u2, d2, v2}}, where cond is True of False depending on whether or not matrices m1 and m2 can be decomposed using the same V.

366. `SmithLLLForm` (in "LatticeTheory.m")
SmithLLLForm[A] returns a list of three matrices U, D, and V such that the m x n resampling

matrix A equals U D V where U (m x m) and V (n x n) are integer matrices with determinants of +1 or -1 and D is a diagonal matrix of integer components such that |det(D)| = |det(A)|. SmithLLLForm[A, smithfun] uses smithfun (which is the name of a routine) instead of the default routine SmithOrderedForm. The algorithm uses Mathematica's LatticeReduce command which is the Lenstra-Lenstra-Lovasz (LLL) lattice reduction algorithm. The Min option tells SmithLLLForm to return either a U or a V with minimum Frobenius norm: Left means U (default) and Right means V. SmithLLLForm supports the same options as does SmithOrderedForm.

367. `SmithMcMillanForm` (in "LatticeTheory.m")
SmithMcMillanForm[ratmat] factors the rational matrix ratmat into the product of U D V, where U and V are integer matrices having a determinant of +1 or -1 and D is a diagonal matrix having rational integers along the diagonal. The routine factors out the LCM of the denominators of ratmat decomposes the resulting integer matrix into Smith form, and multiplies the diagonal matrix by the rational factor. SmithMcMillanForm[smithfun, ratmat] will determine the Smith-McMillan form from the Smith form returned by smithfun.

368. `SmithNormalForm` (in "LatticeTheory.m")
SmithNormalForm[A] returns a list of three matrices U, D, and V such that the m x n resampling matrix A equals U D V where U (m x m) and V (n x n) are integer matrices with determinants of +1 or -1 and D is a diagonal matrix of integer components such that |det(D)| = |det(A)|. The algorithm follows Kaufmann's {Integer and Mixed Programming}. Note that this factorization is not unique. SmithNormalForm supports a Dialogue option. If Dialogue is True or All, SmithNormalForm will show immediate steps. See also SmithOrderedForm and SmithReducedForm.

369. `SmithOrderedForm` (in "LatticeTheory.m")
SmithOrderedForm[A] returns a list of three matrices U, D, and V such that the m x n resampling matrix A equals U D V where U (m x m) and V (n x n) are integer matrices with a determinant of +1 or -1 and D is a diagonal matrix of integer components such that |det(D)| = |det(A)|. The components along the diagonal of D are positive and sorted. The starting point is the result of the SmithNormalForm routine. SmithOrderedForm[A, {U, D, V}] adjusts D as described previously and then returns the new {U, D, V}. Like SmithNormalForm, SmithOrderedForm supports a Dialogue option. If True or All, SmithNormalForm will show immediate steps. See also SmithNormalForm and SmithReducedForm.

370. `SmithReducedForm` (in "LatticeTheory.m")
SmithReducedForm[A] returns a list of three matrices U, D, and V such that the m x n resampling matrix A equals U D V where U (m x m) and V (n x n) are integer matrices with a determinant of +1 or -1 and D is a diagonal matrix of positive integers such that |det(D)| =

|det(A)|. The diagonal elements are sorted down the diagonal and each one is a factor of the next one. Note that the starting point for this algorithm is the SmithOrderedForm routine. SmithReducedForm[A, {U, D, V}] adjust D as described previously and returns the new {U, D, V}. SmithReducedForm supports a Dialogue option. If True or All, SmithNormalForm will show immediate steps. See also SmithNormalForm and SmithOrderedForm.

371. `Some` (in "Lisp.m")
Some[query, expr] returns True if at least one element of expr satisfies the test query. Some[query, expr, levelspec] returns True if every part of the expression expr satisfies the test query.

372. `Stable` (in "TransSupport.m")
Stable[transexpr] returns True if the transform transexpr represents a stable time-domain signal.

373. `Start` (in "DFT.m")
Start is a slot in the DFTData data structure which indicates when starting point of the DFT sequence.

374. `StartBandwith` (in "Signal.m")
StartBandwith[signal] returns the starting location of the bandwidth of the signal. For signals that are not bandlimited, this method will return -Infinity which is the default value.

375. `Step` (in "SigProc.m")
Step[n] is the unit step function which is 1 for n >= 0 and 0 otherwise. It is commonly used for discrete n. This function differs from CStep only at the origin.

376. `StreamSignal` (in "Signal.m")
StreamSignal is a type of signal that computes its outputs one after another usually without reference to past values.

377. `StreamSignalQ` (in "Signal.m")
StreamSignalQ[signal] returns True if the values of signal are computed one at a time in some order. The value at each point can be dependent on previously-computed values, state information, etc. Signals generated by finite state machines are stream signals.

378. `StripPackage` (in "SupCode.m")
StripPackage[symbol] returns the symbol (as a string) after its context has been removed. To remove the package definition from every symbol in expression, use MapAll[StripPackage, expression].

379. `SubT` (in "TransSupport.m")
SubT[transq, t1, t2] subtracts the transforms t2 from t1 and determines the new region of

convergence. The resulting transform is returned as a list. SubT[transq, t1, t2, lowerlimit] uses lowerlimit as the lower limit on region of convergence values when combining ROC's (default is 0).

380. `SubTree` (in "Tree.m")
SubTree[tree, head] returns the subtree with root head.

381. `SubsetQ` (in "Lisp.m")
SubsetQ[set1, set2, set3, ...] returns True if set1 is a subset of set2 and set2 is a subset of set3, etc.

382. `SubstituteForT` (in "TransSupport.m")
SubstituteForT[transq, t, s, news] substitutes news at every occurrence of s in the transform t. The resulting transform is returned as a list.

383. `Summation` (in "SigProc.m")
Summation[i][f], Summation[i, iend][f], and Summation[i, istart, iend, inc][f] represents the summation of f with respect to variable i: i = (istart) to (iend) step (inc). The Summation operator is an abstraction of Mathematica's Sum object. Applying the object TheFunction to a Summation operator will invoke the Sum object if istart, iend, and inc are numbers.

384. `SymmetricPulseConvolutions` (in "Convolution.m")
SymmetricPulseConvolutions[c, a, t, n] convolves a symmetric pulse with itself n - 1 times. The symmetric pulse has amplitude c and is defined for the continuous real variable t over the interval [-a, a]. The function returns a formula. Therefore, SymmetricPulseConvolutions[c, a, t, 1] simply returns the symmetric pulse formula c CPulse[2 a, t + a].

385. `SystemProperties` (in "System.m")
SystemProperties[system] will show the properties attached to of an operator or list of operators. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. See also DefineSystem and SystemQ.

386. `SystemQ` (in "System.m")
SystemQ[system] will give True if system is in the SPsystems list, and False otherwise. Note that a system is a (parameterized) operator to which DefineSystem has attached properties. For example, by default, Shift[L, n] has been installed as a system, but Shift without the parameters has not.

387. `SystemRewriteRules` (in "RewriteRules.m")
SystemRewriteRules contains a list of over 50 rewrite rules for operators (systems) . This rule is composed of three other rule bases: (1) rules based on operator properties, (2) rules for single rate digital signal processing, and (3) rules for multirate digital signal processing.

The rules in (1) are patterned after those found in Myers' Signal Processing Language and Interactive Computing Environment (SPLICE) and Covell's Algorithm Design Environment (ADE).

388. `TableLookup` (in "SupCode.m")
TableLookup[index, hlist, len, val] returns hlist[[index]] if index is between 1 and len, inclusive; otherwise, val is returned.

389. `TheFunction` (in "SigProc.m")
TheFunction[object] returns the mathematical function embedded in object. For list and transform objects, this function returns the first element after the head of the object; otherwise, the contents of the slot TheFunction are returned. For signal processing operations, the equivalent mathematical operation is returned.

390. `Third` (in "Lisp.m")
Third[list] returns the third element of list.

391. `TimesByConstant` (in "Ptolemy.m")
TimesByConstant[a, const] indicates the product of a and the constant const.

392. `ToCollection` (in "SupCode.m")
ToCollection[expr] strips the head off of arg and returns the argument of expr as a collection. ToCollection returns an object that is a sequence, which is represented in Mathematica 1.2 as (a1, a2, ...) and in Mathematica 2.0 as Sequence[a1, a2, ...]. So, it provides a unified way of generating collections (sequences).

393. `ToContinuous` (in "SigProc.m")
ToContinuous[expr] replaces any discrete operator or signal in expr with its continuous equivalent.

394. `ToDiscrete` (in "SigProc.m")
ToDiscrete[expr] replaces any continuous operator or signal in expr with its discrete equivalent.

395. `ToList` (in "SupCode.m")
ToList[arg] returns arg if arg is a list. Otherwise, List[arg] is returned. ToList[arg1, arg2, ...] returns List[arg1, arg2, ...].

396. `TopElement` (in "StackQueue.m")
TopElement[packet] returns the first element in the data packet. This is the top of the stack if packet is a stack. This is the next in the queue if packet is a queue. The packet remains unaltered.

397. `Transform` (in "ROC.m")

Transform[transform, rminus, rplus] represents a transform. Transform is also a tag for exporting errors messages like Transform::twosided. This object is used exclusively by the various transform rule bases.

398. `TransformDialogue` (in "TransSupport.m")

TransformDialogue[strategy, options, arg1, arg2, ...] is the routine that displays textual dialogue about the strategy being used by a particular transform.

399. `TransformFixUp` (in "TransSupport.m")

TransformFixUp[time var, transform var, options, transform head, triplet flag, transform name, lower, upper] attempts to resolve incomplete transforms according to the transform pairs provided by the user via the TransformLookup option. If the transform is a triplet of information (funct, Rminus, Rplus), e.g. the forward z- and Laplace transforms, then triplet flag is True and the lower and upper arguments are used for the second and third fields of the triplet.

400. `TransformLookup` (in "TransSupport.m")

TransformLookup is an option for the transform rule bases. It allows the user to specify a list of transform pairs to augment those that already exist in the transform rule bases. For example, ZTransform[Shift[L, n][x[n]], n, z, TransformLookup -> { x[n] :> X[z] }]. To specify a region of convergence, use TransformLookup -> { x[n] :> { X[z], rm, rp } }. This example allows the user to work x[n] without every defining it. The multidimensional case is more tricky: ZTransform[x[n1, n2], n, z, TransformLookup -> { x[n1, n2] :> X[z1, n2], X[z1, n2] :> X[z1, z2] }]. A region of convergence can be specified for each dimension. For the z- and Laplace transform rule bases, one can also specify a region of convergence associated with the transform pairs.

401. `TriangleModWithSquare` (in "Polygons.m")

TriangleModWithSquare[Polygon[{v1, v2, v3}], {ll, ur}] will compute the triangle with vertices v1, v2, and v3 modulo the square with lower-left corner coordinates ll and upper-right corner coordinates ur.

402. `Unit` (in "SigProc.m")

Unit[n][t] is the nth order continuous unit function in t. Unit[0][t] is the Dirac delta function (see Delta). Unit[-1][t] is the continuous step function (see CStep).

403. `UnprotectIt` (in "SupCode.m")

UnprotectIt[expr] evaluates expr. If it evaluates to a symbol, write protection will be removed for that symbol.

404. `Upsample` (in "SigProc.m")
Upsample[k,n][f] represents the upsampling operation on f, which is either an expression of n or a sequence (list) of values. Upsampling inserts k-1 zeroes between every two samples. In N dimensions, n is a list of N indices and k is an N x N matrix.

405. `UpsampleFactor` (in "SigProc.m")
UpsampleFactor[f, n] returns the upsampling index which can be a negative or positive number. If this routine returns 1 or -1, then no upsampling has occurred; if it returns 0, then f is constant with respect to n.

406. `UpsampleSequence` (in "SigProc.m")
UpsampleSequence[seq,m,fill] inserts m-1 fill values between every two elements of seq.

407. `UpsampledFunction` (in "SigProc.m")
UpsampledFunction[m, fill, n, upf] represents the upsampling of f. In one-dimension, the upsampled form of f, upf, is obtained by evaluating n -> n / m.

408. `UpsampledSystem` (in "LatticeTheory.m")
UpsampledSystem[expr, vars] returns a resampling matrix and a set of linearly coupled variables taken from the master list vars. For example, UpsampledSystem[Cos[w1 + w2] Sin[w1 - w2], {w1, w2}] will return { {{1,1},{1,-1}}, {w1,w2} }. In this case, the dot product of the resampling matrix and the variable list give {w1 + w2, w1 - w2}. A third optional argument would be a function to extract coefficients that takes an expression, a list of variables, and a value to return on failure.

409. `ValidDecimatorBaseband` (in "Decimation.m")
ValidDecimatorBaseband[Polygon[pointlist]] determines whether or not the baseband represented by the polygon is a valid passband for a integer decimator structure. The routine returns { valid, passband, matrix } where valid is True or False, passband is the baseband whose centroid has been shifted to the origin, and matrix is the downsampling matrix of the decimator.

410. `ValidIntervalQ` (in "Convolution.m")
ValidIntervalQ[i] returns True if i is a valid F-interval, and gives False otherwise. An F-interval is an interval of the form {f, leftendp, rightendp} where f is a function and leftendp < rightendp unless f is an impulse function in which case leftendp must equal rightendp. In a valid F-interval, leftendp and rightendp must be numbers. See also IntervalQ.

411. `ValidOperatorQ` (in "System.m")
ValidOperatorQ[system, signal1, signal2, ...] will return True if every signal can be processed by the system. For example, False would be returned if system was only a discrete system and signal1 was a continuous signal.

412. `VariableQ` (in “SupCode.m”)
VariableQ[x] returns True if x is a symbol that (1) does not have a numerical value associated with it and (2) does not have its Constant attribute enabled. Pi fails the first test, so it is not considered a variable. A variable can also have the form of C[n] where n is an integer and C is a symbol whose Constant attribute is enabled.

413. `VectorConvolution` (in “Convolution.m”)
VectorConvolution[vector1, vector2] convolves two vectors that represent samples of function.

414. `WithinROC` (in “ROC.m”)
WithinROC[rm1,rp1,rm2,rp2] returns True if the interval [rm1,rp1] is a subset of the interval [rm2,rp2].

415. `WriteSlot` (in “Slot.m”)
WriteSlot[object, slot, value] attempts to write over the slot slot in object. If the slot slot is read-only, then object is returned unaltered. If the slot is not write protected, then the slot slot in object is overwritten with value. Note that object is an unevaluated symbol and that a slot value cannot be a collection of values like (1, 2, 3).

416. `Z` (in “SigProc.m”)
Z[n, z] is the forward z-transform operator. Applying TheFunction to the Z operator will invoke the z-transform rule base ZTransform.

417. `ZForm` (in “ZSupport.m”)
ZForm[x] return True if the data packet x is tagged with either List or Transform and contains three elements.

418. `ZPolynomial` (in “SupCode.m”)
ZPolynomial[m, n] is an mth order polynomial in the discrete variable n defined by the product of (-n - k) for k = 0 ... m-1. The z-transform of the product of this polynomial and some function f[n] gives z^m times the mth derivative of F(z).

419. `ZSolve` (in “ZSolve.m”)
ZSolve[diffequ == drivingfun, y[n]] solves the difference equation diffequ = drivingfun, where diffequ is a linear constant coefficient difference equation and drivingfun is the driving function (a function of n). Thus, diffequ has the form a0 y[n] + a1 y[n - 1] + ... By using options, one can specify initial values; e.g., ZSolve[y[n] + 3/2 y[n-1] + 1/2 y[n-2] == (1/4)^n Step[n], y[n], y[-1] -> 4, y[-2] -> 10]. A difference equations of N terms needs N-1 initial values. All unspecified initial values are considered to be zero. Right-sided and left-sided solutions are possible. ZSolve can justify its answers.

420. `ZTransData` (in "ROC.m")
ZTransData is a data tag for a valid z-transform object.

421. `ZTransform` (in "ZTransform.m")
ZTransform[e,n] or ZTransform[e,n,z] gives the z-transform of the expression e, which is a function of n, by returning an object of three slots which is tagged by ZTransData: <z-transform>, <rminus>, <rplus>, and <z-variables>. The Region of Convergence (ROC) is <rminus> < |z| < <rplus>. The routine returns the actual ROC or a subset of the actual ROC. In two dimensions, ZTransform[e, {n1, n2}, {z1, z2}], is the same as ZTransform [ZTransform [e, n1, z1], n2, z2]. This notation extends naturally to higher dimensions. The forward transform of large expressions will be found much faster if automatic simplification is disabled by using Simplify -> False.

422. `ZTransformQ` (in "ZSupport.m")
ZTransformQ[x] returns True if the datum x is of the form ZTransData[X, rm, rp, z], where X is the z-transform, rm is the Rminus component of the region of convergence, rp is the Rplus component of the region of convergence, and z is the variable(s) in the z-transform.

423. `ZUpsampledQ` (in "ZSupport.m")
ZUpsampledQ[f, n] returns True if the sequence f, as a function of n, has been upsampled. See UpsampleFactor.

424. `ZVariables` (in "ROC.m")
ZVariables is a data tag for the variables in a z-transform object. When applied to an object x containing z-transform information, ZVariables[x] returns the z-variables contained in the z-transform. For example, if x is a z-transform object of the form ZTransData[zexpr, Rminus[r-], Rplus[r+], ZVariables[z]], then ZVariables would return z.

425. `ZeroQ` (in "SupCode.m")
ZeroQ[x] returns True if x is 0 or 0.0