SMP Library

September 1982

XAbs

Abs simplification rules

absolute value – modulus – numerical value

S.Wolfram
Jul 1981

```
Abs[$x $$x] :: Abs[$x] Abs[$$x]
Abs[$x^($n_=Natp[$n])] : Abs[$x]^$n
D[Abs[$x],{$x,1,$y}] : Sign[$y]
Abs[Sign[$x_=($x~=0)]] : 1
```

~~Abs[$x] ^ ($n_ = Evenp[$n]) :~~

• Theta; Sign; Ramp; Delta

```
#I[1]::  <XAbs
#I[2]::  Abs[a b^2 c]

                        2
#O[2]:   Abs[a] Abs[b]   Abs[c]
```

Warning: Redefines Abs.

XAck

Ackermann function

generalized power – general recursion – recursive function theory

C.Cole and S.Wolfram
Jul 1981

Ack[x, y]

An Ackermann function.

```
Ack[0,$y] : Mod[$y+1,3]
Ack[$x,0] : $x+1
Ack[$x,1] : $x+2
Ack[$x,2] : 2$x
Ack[$x,3] : 2^$x
Ack[$x,4] :: Arrow[2,$x+1]
Ack[$x,$y] :: Ack[$x,$y] : Ack[Ack[$x-1,$y],$y-1]
```

Arrow[n, m]

Knuth arrow function n^(n^(n^...))) with m powers.

```
Arrow[$n,0] : 1
Arrow[$n,1] : $n
Arrow[$n,$m] :: $n^Arrow[$n,$m-1]
```

```
#I[1]::   <XAck

#I[2]::   Ar[10,Arrow[$i,2]]

#O[2]:    {1,4,27,256,3125,46656,823543,16777228,387428588,1.*^10}

#I[3]::   Ack[2,5]

#O[3]:    16

#I[4]::   Ack[3,5]
```

XAllbut

List element deletion

list element removal – complement – exclusion

S.Wolfram

Aug 1982

Allbut[*list*, *e1*, *e2*, . . .]
yields *list* with all occurrences of *e1*, *e2*, ... deleted.

```
Allbut[$list,$$e] :: (Lcl[t]; t:$list; Do[i,Len[List[$$e]], \
                      t:Del[List[$$e][i],t]]; t)
```

Misfeatures: untested

XAny

List element condition test

scan – any – test for any elements – existence test

S.Wolfram
Jul 1981

Any[*temp*, *list*]

tests whether any of the elements of *list* yield "true" on application of the template *temp*.

```
Any_:Tier
_Any[Smp] : {0,Inf}
Any[$temp,$list] :: In[$1_=Ap[$temp,{$1}],$list]
```

• Pos; XScan

```
#I[1]::  <XAny

#I[2]::  Any[Evenp,Ar[5,Prime]]

#O[2]:   1
```

XArperm

Permutation generation

reorderings – symmetries

S.Wolfram
Jul 1981

Arperm[*n*, (*spec* : (*all*))]

yields a list of the permuations of *n* elements which exhibit the symmetries *spec*.

```
Arperm[$n] :: Flat[Ar[Ar[$n,$n],List,Uneq],$n]

          <<XList8
          Arperm_:Tier
          Arperm[1]:{{1}}
          Arperm[$n_:Natp[$n]] :: Arperm[$n]: \
                  Flat[Map[Ar[$n,Ins[$n,$%1,$%2]],Arperm[$n-1]],1]
          Arperm[$n,Cyclic] :: Ar[$n,Cyc[Ar[$n],$%1]]
          Arperm[$n,Even] :: Cat[Ar[$n!,Arperm[$n],Evenp]]
          Arperm[$n,Odd] :: Cat[Ar[$n!,Arperm[$n],Oddp]]
```

```
#I[1]::  <<XArperm

#I[2]::  Arperm[3]

#O[2]:   {{3,2,1},{2,3,1},{2,1,3},{3,1,2},{1,3,2},{1,2,3}}

#I[3]::  Arperm[3,Cyclic]

#O[3]:   {{2,3,1},{3,1,2},{1,2,3}}

#I[4]::  Arperm[3,Even]

#O[4]:   {{2,3,1},{3,1,2},{1,2,3}}

#I[5]::  Arperm[3,Odd]

#O[5]:   {{3,2,1},{2,1,3},{1,3,2}}
```

Prerequisites: XList8

XBase

Number base conversion

radix arithmetic – positional notation – binary – ternary
octal – hexadecimal – scales of notation – integer conversion

S.Wolfram

Jul 1981

To10[*ccccc*, *n*]

converts the number *ccccc* from base *n* to base 10. *ccccc* represents an integer whose digits are characters in the symbol name *ccccc*. The "digit" 10 is represented by a, 11 by b and so on.

```
To10_:Tier
To10[$s_=Symbp[$s],$b_=Natp[$b]] :: (Lcl[%l]; %l:Expl[$s]; \
                    Sum[$b^(Len[%l]-%i)*%l[%i],{%i,1,Len[%l]}])
```

From10[*x*, *n*]

converts the decimal integer *x* to a Base projection in base *n*.

```
From10_:Tier
From10[$n_=Natp[$n],$b_=Natp[$b-1]] :: (Lcl[%tot,%res,%i]; \
        For[%i:1; %res:$n, %res~=0, Inc[%i], %tot[%i]:Mod[%res,$b]; \
        %res:Floor[%res/$b]]; Impl[Rev[%tot]])
```

• XTern

```
#I[1]::  <XBase
#I[2]::  From10[1452,2]
#O[2]:   "10110101100"
#I[3]::  To10[%,2]
#O[3]:   1452
#I[4]::  From10[%,16]
#O[4]:   "5ac"
#I[5]::  To10[%,16]
#O[5]:   1452
```

XBell

Bell numbers

Stirling numbers – number of equivalence relations
combinatorial functions

S.Wolfram

Sep 1982

Bell[*n*]

n th Bell number.

```
Bell[$n_/;Natp[$n]] :: Sum[Sti2[$n,%i],{%i,1,$n}]
```

[Sloane: Handbook of Integer Sequences, sect. 3.12]

XBer2V

Bernoulli polynomials

S.Wolfram
Feb 1982

Definition of values for integer index.

```
<XBerV
Ber[$n_=Natp[$n],$x] :: Sum[Ber[%k] Comb[$n,%k] $x^($n-%k),{%k,0,$n}]
```

[MOS sect. 1.5.1]

XBerV

Bernoulli numbers

S.Wolfram
Feb 1982

Definitions for special values of argument.

```
Ber[0] : 1

Ber[1] : -1/2

Ber[2] : 1/6

Ber[4] : -1/30

Ber[$m_=Oddp[$m]] : 0

Ber[$m_=Natp[$m/2]] :: (Lcl[%a,%c];%c:1;%a:0;Do[%i,0,$m-1,%a:%a+%c Ber[%i]; \
                %c:%c ($m+1-%i)/(%i+1)]; Ber[$m]:-%a/($m+1))
```

[AS 23.1.7]

XBit

Bitwise operations

binary – bitwise and – bitwise or – collate

S.Wolfram
Jul 1981

Bits[*n*]

yields a list of binary bits corresponding to the integer *n*.

```
Bits_:Tier
Bits[$n_=Natp[$n]] :: (Lcl[%tot,%res,%i]; %i:1;%res:$n; \
        Loop[%res~=0,%tot[%i]:Mod[%res,2];%res:Floor[%res/2];Inc[%i]]; \
                Rev[%tot])
```

Intbit[*list*]

finds the integer corresponding to a list of bits.

```
Intbit[$list_=Contp[$list]] ::  \
                Sum[2^(Len[$list]-%i) $list[%i],{%i,1,Len[$list]}]
```

Bitand[*n*, *m*]

yields the bitwise conjunction of *n* and *m*.

```
Bitand_:Comm
Bitand[$n_=Natp[$n],$m_=(Natp[$m] & $m>$n)] :: \
        (Lcl[%n,%m]; %n:Bits[$n]; %m:Bits[$m]; \
        Intbit[Rev[Ldist[Rev[%n] & Ar[Len[%n],Rev[%m]]]]])
 Bitand[$n_=Natp[$n],$n]::$n
```

Bitor[*n*, *m*]

yields the bitwise disjunction of *n* and *m*.

```
Bitor_:Comm
Bitor[$n_=Natp[$n],$m_=Natp[$m]] :: \
        Intbit[Rev[Ldist[Rev[Bits[$n]] | Rev[Bits[$m]]]]]
```

```
#I[1]::  <<XBit

#I[2]::  Bits[123]

#O[2]:   {1,1,1,1,0,1,1}

#I[3]::  Intbit[%]

#O[3]:   123

#I[4]::  Bitand[123,514]

#O[4]:   4

#I[5]::  Bitor[123,514]

#O[5]:   635
```

XBox

Additional graphical objects

geometrical figures – square – box – circle – plotting regions

S. Wolfram

Jul 1981

Box[{x, y}]
represents a unit box centred at the point x,y.

```
Box[{$x,$y}] :: Line[{Pt[{$x-0.5,$y-0.5}],Pt[{$x+0.5,$y-0.5}], \
                      Pt[{$x+0.5,$y+0.5}],Pt[{$x-0.5,$y+0.5}], \
                              Pt[{$x-0.5,$y-0.5}]}]
```

Circle[{x, y}]
represents a unit circle centred at the point x,y.

```
Circle[{$x,$y}] :: {Curve[Ar[20,Pt[{$x,$y}+{Sin[2Pi $1/20], \
                              Cos[2Pi $1/20]}]]]}
```

XCatalan

Catalan numbers

convex polygon dissection – rooted planar trees

S.Wolfram
Sep 1982

Catnum[*n*]
n th Catalan number.

```
Catnum[$n] :: Comb[2$n,$n]/($n+1)
```

[Sloane: Handbook of Integer Sequences, sect. 3.5]

XCf

Continued Fraction Numbers

continued fraction representation

S.Wolfram
Aug 1982

CfD[{*a1*, *a2*, . . .}]
converts the continued fraction number with coefficients *a1*, *a2*, ... to decimal form.

```
CfD[$list_=Contp[$list]] :: (Lcl[t];t:Last[$list]; \
          Do[i,Len[$list]-1,1,-1,t:N[1/t+$list[i]]];t)
```

DCf[*n*, *ord*]
converts the decimal number *n* to a continued fraction form to order *ord*.

```
DCf[[$n,$ord_=Natp[$ord]]] :: \
     (Lcl[n,ni,list]; n:$n; list:{}; Do[i,$ord,ni:Floor[n]; \
          list:Cat[list,{ni}]; n:N[1/(n-ni)]]; list)
```

```
#I[1]::  <XCf

#I[2]::  N[Phi]

#O[2]:   1.61803

#I[3]::  DCf[%,10]

#O[3]:   {1,1,1,1,1,1,1,1,1,1}

#I[4]::  CfD[%]

#O[4]:   1.61818

#I[5]::  DCf[N[Pi],10]

#O[5]:   {3,7,15,1,292,1,1,1,2,1}
```

Future enhancements: Treat arbitrary precision numbers; may be included in `Cf[]` internal code.

XChar

Character manipulation

text manipulation – character strings – ASCII – letters
words

S.Wolfram

Jul 1981

Upperp[*str*]

yields 1 if the first character of the string *str* is an upper case letter, and 0 otherwise.

```
Upperp[$str] :: (!Exp|[A][1]) <= Exp|[$str][1] <= (!Exp|[Z][1])
```

Lowerp[*str*]

yields 1 if the first character of the string *str* is a lower case letter, and 0 otherwise.

```
Lowerp[$str] :: (!Exp|[a][1]) <= Exp|[$str][1] < (!Exp|[z][1])
```

Tolower[*str*]

converts all upper case letters in *str* to lower case.

```
Tolower[$str] :: Imp|[Map[$1-(!Exp|[A][1]-Exp|[a][1]),Exp|[$str],1, \
                   (!Exp|[A][1]) <= $2 <= (!Exp|[Z][1])]]
```

```
#I[1]::  <XChar
#I[2]::  t1:"a message"
#O[2]:   "a message"
#I[3]::  t2:"A MESSAGE"
#O[3]:   "A MESSAGE"
#I[4]::  Lowerp[t1]
#O[4]:   1
#I[5]::  Lowerp[t2]
#O[5]:   0
#I[6]::  Tolower[t2]
#O[6]:   "a message"
```

XChi2

Chi squared distribution

probability functions – errors – statistics – normal distribution
Gaussian distribution

S.Wolfram
Aug 1982

QChi2[*chi2*, *nu*]
value of chi squared distribution at *chi2* with *nu* degrees of freedom.

```
QChi2[[$chi2,$nu]] :: N[Gamma[$nu/2,$chi2/2]/Gamma[$nu/2]]
```

[AS 26.4.1]

XClass

Classified data statistics

binned data – histograms

C.Feynman
Aug 1981

Class[*datum*, *cent*, *spacing*]

returns the number closest to *datum* which is equal to *cent* modulo *spacing*. This is the class mark (the location of the center of the class) of the class in which *datum* belongs, given an arrangement of classes of constant width (equal to *spacing*), one of which is centered on *cent*.

```
Class[$x,$cent,$space] : $cent + -1*($space*Floor[1/2 + ($cent + -$x)\
/$space])
```

Classify[*d*, *cent*, *space*]

returns a list, the indices of which are the class marks of a set of classes whose spacing is *space*, and one of which is centered on *cent*. The elements of the list are the frequencies with which the numbers in the list *d* fall into the respective classes. The returned list is the smallest which will include all of *d*.

```
Classify[$data,$c,$s] :: Proc[Lcl[rslt] ; rslt : Ar[{{Class[Ap[Min\
,$data],$c,$s],Class[Ap[Max,$data],$c,$s],$s}},0] ; Map[rslt[Class\
[$y,$c,$s]] : rslt[Class[$y,$c,$s]] + 1,$data]\
; rslt]
```

CMode[*d*]

returns a list consisting of the modes of the data in *d*, i. e. the class mark or marks of the most crowded class or classes.

```
CMode[$data] :: Ap[Cat,Pos[Ap[Max,$data],$data]]
```

CMean[*d*]

returns the arithmetic mean of the data in *d*.

```
CMean[$data] :: N[Sum[Elem[$data,{n}]*Ind[$data,n],{n,1,\
Len[$data]}]/Ap[Plus,$data]]
```

CSD[*d*]

returns the standard deviation of the data in *d*.

```
CSD[$d] :: N[Sqrt[CVar[$d]*Ap[Plus,$d]/(Ap[Plus,$d] + -1)]]
```

CVar[*d*]

returns the variance of the data in *d*.

```
CVar[$d] :: N[(Sum[Ind[$d,n]^2*Elem[$d,{n}],{n,1,\
Len[$d]}] + -1*(Ap[Plus,$d]*CMean[$d]^2))/Ap[Plus,$d]]
```

CApprox[*d*]

returns an Err projection whose mean and standard deviation are the same as those of *d*.

```
CApprox[$d] :: Err[CMean[$d],CSD[$d]]
```

CMD[*d*]

returns the mean absolute deviation of the data in *d*.

```
CMD[$d] :: (Lcl[mean] ; mean : CMean[$d] ; Sum[Abs[Ind[$d,n] + -mean]\
*Elem[$d,{n}],{n,1,Len[$d]}])/Ap[Plus,$d]
```

Unclassify[*d*]

returns a list of unclassified data, which, if classified, would produce *d*. This is the "inverse" of `Classify`. It is of course not perfect, since much information is lost in the classification process.

```
Unclassify[$d] :: Purify[Ap[List,Map[Repl[Ind[$d,$x],Elem[$d,{$x}\
]],List[1 .. Len[$d]]]]]
Purify[$list]::(Lcl[r];r:{};Do[i,Len[$list],If[Numbp[Elem[$list,{i}]],\
r:Cat[r,List[Elem[$list,{i}]]]]];r)
```

CFrac1[*d*,*f*]

is a projection which is called only by `CFract`. You should never need it.

```
CFrac1[$d,$frac] :: If[$frac > 0,Lcl[n,total,wanted] ; \
For[n : 0 ; total : 0; wanted : $frac*Ap[Plus,$d],wanted > total,\
n : n + 1; total : total + Elem[$d,{n}],n] + 1,Lcl[n] ; n\
: 0 ; Loop[Null,n : n + 1,Elem[$d,{n}] = 0]]
```

CFract[*d*,*f*]

returns the *f* fractile of the data in *d*.

```
CFract[$d _= (Len[$d] > 1),$frac _= ($frac >= 0 & 1 >= $frac)] :: \
N[(Lcl[where,c,bl,fm,fm1,nn] ; where : CFrac1[$d,$frac] ; c : Ind[$d,\
2] + -Ind[$d,1] ; bl : Ind[$d,where] + -1*c/2 ; fm\
: Elem[$d,{where}] ; fm1 : Ap[Plus,Map[Elem[$d,{$n}],\
{1 .. where + -1}]] ; nn : Ap[Plus,$d] ; \
bl + c*(nn*$frac + -fm1)/fm)]
```

CMed[*d*]

returns the median of the data in *d*.

```
CMed[$d] :: CFract[$d,0.5]
```

CQ1[*d*]

returns the first quartile of the data in *d*.

```
CQ1[$d] :: CFract[$d,0.25]
```

CQ3[*d*]

returns the third quartile of the data in *d*.

```
CQ3[$d] :: CFract[$d,0.75]
```

CMin[*d*]

returns the minimum of the data in *d*.

```
CMin[$d] :: (Lcl[n] ; n : 0 ; Ind[$d,Loop[Null,n : n + 1,Elem[$d,\
{n}] = 0]])
```

CMax[*d*]

returns the maximum of the data in *d*.

```
        CMax[$d] :: (Lcl[n] ; n : Len[$d] + 1 ; Ind[$d,Loop[Null,n : n + -1,\
        Elem[$d,{n}] = 0]])

#I[1]::  <xclass

#I[2]::  Class[2.43,0,1]

#O[2]:   2

#I[3]::  Class[2.43,.5,1]

#O[3]:   2.5

#I[4]::  Classify[{22,15,35,45,36,25,16,35,29,38,28,45,36,33,20,25,15,9,48,42,

#O[4]:   {[7.5]: 1, [12.5]: 2, [17.5]: 2, [22.5]: 3, [27.5]: 2, [32.5]: 3,

                [37.5]: 3, [42.5]: 4, [47.5]: 1}
```

At this point, due to a garbage collector bug, SMP went into an infinite loop. We thus have to start over:

```
#I[1]::  <exampl
```

This file contains an example of classified data. It is the value of the variable a.

```
#I[2]::  <xclass

#I[3]::  a

#O[3]:   {[-3]: 1, [-1]: 2, [1]: 0, [3]: 2, [5]: 4, [7]: 6, [9]: 5, [11]: 6,

                [13]: 4, [15]: 2, [17]: 1, [19]: 1}

#I[4]::  CMode[a]

#O[4]:   {7,11}

#I[5]::  CMean[a]

#O[5]:   8.588235

#I[6]::  CSD[a]

#O[6]:   4.991615

#I[7]::  Unclassify[a]

#O[7]:   {-3,-1,-1,3,3,5,5,5,5,7,7,7,7,7,7,9,9,9,9,9,11,11,11,11,11,11,13,13,

             13,13,15,15,17,19}
```

XCode

Mixed radix operations

APL encode/decode

S.Wolfram
Jan 1982

Encode[*radix*, *list*]
encodes *list* according to the specified mixed *radix*.

```
Encode[$radix_=Listp[$radix],$list_=(Listp[$list]   \
&Len[$list]=Len[$radix]+1)] :: \
        (Lcl[%tot,%i]; %tot:$list[1]; Do[%i,2,Len[$list], \
        %tot:%tot $radix[%i-1]+$list[%i]]; %tot)
```

Decode[*radix*, *n*]
yields the number *n* in the specified mixed *radix*.

```
_Decode[Init] :: <XTri
Decode[$radix_=Listp[$radix],$n_=Natp[$n]] :: \
    (Lcl[%tot,%dig,%rad,%i];%rad:Rev[Map[Ap[Mult,$1],Tri[$radix]]]; \
        %tot:$n; Do[%i,Len[$radix],%tot:%tot-(%dig[%i]: \
                  Gint[%tot/%rad[%i]])%rad[%i]]; Cat[Rev[%dig],{%tot}])
```

Misfeatures: Decode definition is incorrect

Decode definition not yet correct.
Example:

```
time:{365,24,60}
Encode[time,{4,2,1,5}]
Decode[time,%]
```

XCon

Tensor contraction

explicit tensors – inner products – generalized traces

S.Wolfram

Jul 1981

Con[*list*, *ni*, *nj*]

forms the contraction of the tensor list over its *nith* and *njth* indices.

```
Con[$list,$ni_=Natp[$ni],$nj_=Natp[$nj]] :: \
        (Lcl[%t]; %t:Trans[Trans[$list,{1,$ni}],{2,$nj}]; \
                Ap[Plus,Ar[Len[%t],%t[$1,$1]]])
```

```
#I[1]::  <XCon

#I[2]::  w3:Ar[{2,2,2},f]

#O[2]:   {{{f[1,1,1],f[1,1,2]},{f[1,2,1],f[1,2,2]}},
          {{f[2,1,1],f[2,1,2]},{f[2,2,1],f[2,2,2]}}}

#I[3]::  Con[%,1,2]

#O[3]:   {f[1,1,1] + f[2,2,1],f[1,1,2] + f[2,2,2]}

#I[4]::  Con[w3,2,3]

#O[4]:   {f[1,1,1] + f[1,2,2],f[2,1,1] + f[2,2,2]}

#I[5]::  w4:Ar[{2,2,2,2},f];

#I[6]::  Con[w4,2,4]

#O[6]:   {{f[1,1,1,1] + f[1,2,1,2],f[2,1,1,1] + f[2,2,1,2]},
          {f[1,1,2,1] + f[1,2,2,2],f[2,1,2,1] + f[2,2,2,2]}}
```

• Inner; Outer

XConfus

Confusing function

S.Wolfram
Jul 1981

Conf[*x*]
is a confusing function.

```
Conf[$x_;Numbp[$x]] :: Conf[$x]:Rand[]
```

XConsSol

Numerical solution of equations

Newton's method – numerical inversion

T.Shaw
Feb 1982

ConsSol[*f*, *e1*=*e2*, *v*, {$$*args*}]

creates a top level smp function *f*[$$*args*,*guess1*,*guess2*,*acc*] which returns the value of *v* (a variable in the *e1* or *e2*) which solves the equation. The $$*args* are parameters to the equation. They must be generic symbols. When given to the consed function, all parameters must be numeric. This function may then be consed, for increased efficiency.

```
_ConsSol[Smp] : 0
ConsSol _: Tier

ConsSol[$f_=Symbp[$f],$e1=$e2,$v_=Symbp[$v]] :: \
          ConsSol[$f,$e1=$e2,$v,{}]
ConsSol[$f_=Symbp[$f],$e1=$e2,$v_=Symbp[$v],$args_=Listp[$args]] :: \
          (Ap[Set,{Proj[$f,Cat[$args,{$g1,$g2,$acc}]], \
           Ap[' ( Lcl[x1,x2,y1,y2] ; x1 : $g1 ; y1 : $g2 ; \
                   y1 : $0 ; y2 : $1 ; \
                   Loop[ fabs[y2 - y1] > fabs[$acc y2], Lcl[tmp] ; \
                              tmp : x2 - y2 * ((x2-x1)/(y2-y1)) ; \
                              x1:x2 ; y1:y2 ; x2:tmp ; y2:$2 ] ; \
                   x2 ), \
          {S[$e1-$e2,$v->$g1],S[$e1-$e2,$v->$g2],S[$e1-$e2,$v->tmp]}]}];$f)
```

```
#I[1]::  <XConsSol

#I[2]::  ConsSol[f,Cos[x] = $y x,x,{$y}]

#O[2]:   ' f

#I[3]::  Cons[f]

#O[3]:   {' f}

#I[4]::  f[1,.5,1.5,.0001]

#O[4]:   0.739085

#I[5]::  N[Cos[X]]

#O[5]:   0.739085

#I[6]::  f[5,.1,.9,.0001]

#O[6]:   0.196164

#I[7]::  N[Cos[X]]

#O[7]:   0.980821

#I[8]::  5*@6

#O[8]:   0.980821
```

XContig

Contiguous list generation

pad – fill – make rectangular – make square – make cubical
fill holes

S.Wolfram
Jul 1981

Contig[*list*, {*n1*, *n2*, . . .}, *elem*]
renders list contiguous with *ni* entries at level *i* by inserting *elem* where necessary.

```
Contig[$list_↙Listp[$list],$n_↙Listp[$n],$elem] :: \
          Contig0[$list,$n,$elem,1]
Contig0[$list,$n,$elem,$lev_↙$lev>Len[$n]] : $list
Contig0[$list_↙~Listp[$list],$n,$elem,$lev] : $list
Contig0[$list_↙Listp[$list],$n,$elem,$lev] :: \
          Ar[$n[$lev],If[P[Proj[$list[$1],{0}]=Proj],   \
                    $elem,Contig0[$list[$1],$n,$elem,$lev+1]]]
```

```
#I[1]::  <XContig

#I[2]::  t:{[3]:a,[2]:b}

#O[2]:   {[3]: a,  [2]: b}

#I[3]::  Contig[t,{4},0]

#O[3]:   {0,b,a,0}

#I[4]::  Contig[t,{8},x]

#O[4]:   {x,b,a,x,x,x,x,x}

#I[5]::  Ar[{{2,3},{3,4}},f]

#O[5]:   {[2]: {[3]: f[2,3], [4]: f[2,4]}, [3]: {[3]: f[3,3], [4]: f[3,4]}}

#I[6]::  Contig[%,{4,4},0]

#O[6]:   {0,{0,0,f[2,3],f[2,4]},{0,0,f[3,3],f[3,4]},0}
```

XD

Derivatives

S.Wolfram
Jul 1981

```
D[Ber[$n,$z],{$z,$m,$x}]              :: $n!/($n-$m)!*Ber[$n-$m,$x]

D[Beta[$x,$y,1],{$x,1,$z}]            :: Beta[$z,$y,1] (Psi[$z] - Psi[$z+$y])
D[Beta[$x,$y,1],{$y,1,$z}]            :: Beta[$x,$z,1] (Psi[$z] - Psi[$x+$z])

D[CheT[$n,$z],{$z,1,$x}]              :: $n CheU[$n-1,$x]
D[CheT[$n,$z],{$z,$m,$x}]             :: 2^($m-1) Gamma[$m] $n Geg[$n-$m,$m,$x]

D[CheU[$n,$z],{$z,$m,$x}]             :: 2^$m $m!*Geg[$n-$m,$m+1,$x]

D[Chg[$a,$c,$z],{$z,$m,$x}]           ::\
        Poch[$a,$m] Chg[$a+$m,$c+$m,$x] / Poch[$c,$m]

D[Coshi[$z],{$z,1,$x}]                :: Cosh[$x]/$x
D[Cosi[$z],{$z,1,$x}]                 :: Cos[$x]/$x

D[Ei[$z],{$z,1,$x}]                   :: Exp[$x]/$x

D[EllK[$k,$t],{$t,1,$u}]              :: (1-$k^2 Sin[$u]^2) ^ (-1/2)
D[EllE[$k,$t],{$t,1,$u}]              :: Sqrt[1-$k Sin[$u]^2]

D[Erf[$z],{$z,1,$x}]                  :: 2 Exp[-$x^2] / Sqrt[Pi]
D[Erf[$z],{$z,$m,$x}]                 :: -2 (-1)^$m Exp[-$x^2] Her[$m-1,$x]

D[Eul[$n,$z],{$z,$m,$x}]              :: $n!/($n-$m)!*Eul[$n-$m,$x]

D[Expi[1,$z],{$z,$m,$x}]              :: (-1)^$m Exp[-$x] Chg[1,1+$m,$x]
D[Expi[$n,$z],{$z,1,$x}]              :: -Expi[$n-1,$x]

D[FreC[$z],{$z,1,$x}]                 :: Cos[Pi $x^2 / 2]
D[FreS[$z],{$z,1,$x}]                 :: Sin[Pi $x^2 / 2]

D[Gamma[$z],{$z,1,$x}]                :: Gamma[$x] Psi[$x,1]
D[Gamma[$z,$a],{$z,1,$x}]             :: -$a^($x-1) Exp[-$a]

D[Geg[$n,$l,$z],{$z,$m,$x}]           :: 2^$m Poch[$l,$m] Geg[$n-$m,$l+$m,$x]

D[Her[$n,$z],{$z,$m,$x}]              :: 2^$m $n!/($n-$m)!*Her[$n-$m,$x]

D[Hg[$a,$b,$c,$z],{$z,$m,$x}]         ::\
        Poch[$a,$m] Poch[$b,$m] Hg[$a+$m,$b+$m,$c+$m,$x] / Poch[$c,$m]

D[JacP[$n,$a,$b,$x],{$z,$m,$x}]       ::\
        Poch[$a+$b+$c+1,$m] JacP[$n-$m,$a+$m,$b+$m,$x] / 2^$m

D[JacCd[$z,$m],{$z,1,$x}]             :: ($m-1) JacSd[$x,$m] JacNd[$x,$m]
D[JacCn[$z,$m],{$z,1,$x}]             :: -JacSn[$x,$m] JacDn[$x,$m]
D[JacCs[$z,$m],{$z,1,$x}]             :: -JacNs[$x,$m] JacDs[$x,$m]
D[JacDc[$z,$m],{$z,1,$x}]             :: (1-$m) JacSc[$x,$m] JacNc[$x,$m]
D[JacDn[$z,$m],{$z,1,$x}]             :: -$m JacSn[$x,$m] JacCn[$x,$m]
D[JacDs[$z,$m],{$z,1,$x}]             :: -JacCs[$x,$m] JacNs[$x,$m]
D[JacNc[$z,$m],{$z,1,$x}]             :: JacSc[$x,$m] JacDc[$x,$m]
D[JacNd[$z,$m],{$z,1,$x}]             :: $m JacSd[$x,$m] JacCd[$x,$m]
D[JacNs[$z,$m],{$z,1,$x}]             :: -JacDs[$x,$m] JacCs[$x,$m]
D[JacSc[$z,$m],{$z,1,$x}]             :: JacDc[$x,$m] JacNc[$x,$m]
D[JacSd[$z,$m],{$z,1,$x}]             :: JacCd[$x,$m] JacNd[$x,$m]
D[JacSn[$z,$m],{$z,1,$x}]             :: JacCn[$x,$m] JacDn[$x,$m]

D[JacZ[$z,$m],{$z,1,$x}]              :: JacDn[$x,$m]^2
```

```
D[KumU[$a,$c,$z],{$z,$m,$x}]   :: (-1)^$m Poch[$a,$m] KumU[$a+$m,$c+$m,$x]

D[Lag[$n,$a,$z],{$z,1,$x}]     :: -Lag[1-$n,1+$a,$x]

D[LegP[$n,$z],{$z,1,$x}]       :: $n (LegP[$n-1,$x]-$x LegP[$n,$x]) / (1-$x^2)

D[Li[$n,$z],{$z,1,$x}]         :: Li[$n-1,$x]/$x

D[Lob[$z],{$z,1,$x}]           :: Log[Sec[$x]]

D[Logi[$z],{$z,1,$x}]          :: 1/Log[$x]

D[Psi[$z],{$z,1,$x}]           :: Psi[$x,2]
D[Psi[$z,$n],{$z,1,$x}]        :: Psi[$x,$n+1]

D[Sinhi[$z],{$z,1,$x}]         :: Sinh[$x]/$x
D[Sini[$z],{$z,1,$x}]          :: Sin[$x]/$x

D[Zeta[$z,$a],{$a,1,$b}]       :: -$z Zeta[$z+1,$b]
```

XDSol

Series solution of differential equations

power series – Frobenius method

J.Greif and S.Wolfram
Oct 1981

DSol[*eqn*, *y*, *x*, *ord*, {*y*[0], *y'*[0], ...}]
gives a series solution to the ordinary differential equation *eqn* with dependent variable *y* and independent variable *x* and specified boundary conditions, accurate to order *x*^*ord*.

• XSerSol

```
DSol[$eqn1=$eqn2,$f,$x,$ord,$init] :: \
(Lcl[%a,%eqn,%list,%f]; %f:Sum[%a[i] $x^i,{i,0,$ord}];\
%eqn:S[Ex[S[$eqn1-$eqn2,$f->%f]],$x^$i_=($i>$ord)->0];\
%list:Union[Cat[Ar[$ord-1,Coef[$x^$1,%eqn]],{S[%eqn,$x->0]},\
        Ar[Len[$init],D[%f,{$x,$1-1,0}]-$init[$1]]]]];\
%list:Map[$2=0,%list];\
%list:Sol[%list,Ar[{{0,$ord}},%a[$1]]];\
S[%f,%list])
```

```
#I[1]::  <XDSol

#I[2]::  eq:Dt[y,x]+a y=0

#O[2]:   Dt[y,x] + a y = 0

#I[3]::  DSol[%,y,x,2,{1}]

                    2  2
                   a  x
#O[3]:   1 - a x + -----
                     2
```

XDap

Directional application

mapping – rotate – APL – column operations

S.Wolfram
Jul 1981

Dap [*f*, *list*, *n*]

applies the template *f* to "columns" of elements at level *n* in *list*.

```
Dap[$f,$list_/Listp[$list],$n_/(Natp[$n] & $n>1)] :: \
        Trans[Map[Ap[$f,$%1],Trans[$list,$n]],$n-1]
Dap[$1,$list_/Listp[$list],1] :: Ap[$f,$list]
```

XData

Data input

columnated input – data

S.Wolfram

Jul 1981

Data[*file*, (*nline*)]

reads data from *file* for *nline* lines or until the end of file is encountered, taking columns to be separated by tabs, single or double spaces.

```
Data_:Tier
Data[$file] :: Data[$file,100000]
Data[$file,$nline_=Natp[$nline]] :: \
        (Lcl[%l,%n,%r]; Sxset[" ",List,3];Sxset["  ",List,3]; \
        Sxset["  ",List,3]; For[%l:{};%n:1,%r:Rd[,{$file,%n}];\
        (Numbp[%r] | Listp[%r]) \
        & %n<=$nline,Inc[%n],%l:Cat[%l,{%r}]]; "        ":= ; \
        " ":= ; "  ":= ; Ret[%l])
```

XDatal

Data input with conversion

columnated input – data conversion

J.Greif
Aug 1982

Cdata[*file*, (*cols* : *Inf*), (*temp*)]

reads data from file *file* using the SMP input filter `smpin` to convert Fortran E-format numbers, if present, to SMP *^ notation, and collect the data into a list of lists, each containing *cols* elements taken sequentially from the input. The template *temp* is applied to each sublist.

```
Cdata_:Tier
_Cdata[Smp]:{Inf,Inf,0}
_Cdata[Init]::(<XUnFlat; <XStr0)
Cdata[$file]::Cdata[$file,1000000]
Cdata[$file,,$temp]::Cdata[$file,100000,$temp]
Cdata[$file,$cols]::UnFlat[Run[CJoin["smpin -mklist >&4 <",$file]],\
        $cols]
Cdata[$file,$cols,$temp]::Map[$temp,Cdata[$file,$cols]]
```

• Cdat [$file,($cols:$Inf),($#temp)]

XDiff

Finite differences

Forward differences – difference equations – finite elements

S.Wolfram
Jul 1981

Diff[*f*, *x*, *x0*, *n*]
yields the *nth* forward finite difference of *f* with respect to *x* at the point *x*=*x0*.

```
Diff[$f,$x,$x0,$n] :: Sum[S[$f,$x->$x0+$n-%r] \
                      (-1)^%r Comb[$n,%r],{%r,0,$n}]
```

```
#I[1]::  <<XDiff

#I[2]::  Diff[f[x],x,0,3]

#O[2]:   -f[0] + 3f[1] - 3f[2] + f[3]

#I[3]::  Ar[5,Ex[Diff[(x+a)^3,x,1,$1]]]

                     2
#O[3]:   {7 + 9a + 3 a ,12 + 6a,6,0,0}
```

XDig

Digit manipulation

text manipulation – positional notation – digit extraction
number construction

S.Wolfram

Jul 1981

Updated Aug 1982

Dig[*n*, *i*]

yields the coefficient of 10^*i* in the number *n*.

```
Dig[$n_»Numbp[$n],$i_»Intp[$i]] :: Mod[Floor[$n/10^$i],10]
```

LDig[*n*]

yields a list of the digits in the integer *n*.

```
LDig[$n_»Natp[$n]] :: Expl[$n]
_F[Extr,'LDig]::FLDig
_FLDig[Init] :: <XPad
FLDig['F[$$x,$i1,$i2]] :: Flat[Map[LPad[Expl[$1],0,4],List[$$x]]]
```

NMake[*list*]

converts a list of digits to an integer.

```
NMake[$list] :: Make[Impl[$list]]
```

```
#I[1]::  <XDig
#I[2]::  Dig[456123.321,4]
#O[2]:   5
#I[3]::  Dig[114.00425,-3]
#O[3]:   4
#I[4]::  LDig[1467120]
#O[4]:   {1,4,6,7,1,2,0}
#I[5]::  NMake[%]
#O[5]:   1467120
#I[6]::  N[Pi,20]
#O[6]:*  (3.14159265358979323850)
#I[7]::  LDig[%]
#O[7]:   {0,0,0,3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3,8,5,0}
```

Prerequisites: XPad

XDim

Dimensional analysis

units – physical quantities – similarity

S.Wolfram
Jul 1981

Fundamental dimensions:

- length
- mass
- time
- current
- temperature
- (luminous) intensity
- amount (of substance)

Derived dimensions

SDim[1] :	area -> length^2	
SDim[2] :	volume -> length^3	
SDim[3] :	frequency -> time^-1	
SDim[4] :	density -> mass/volume	
SDim[5] :	concentration -> amount/volume	
SDim[6] :	velocity -> length/time	
SDim[7] :	acceleration -> length/time^2	
SDim[8] :	force -> mass length time^-2	
SDim[9] :	pressure -> force/area	
SDim[10] :	stress -> force/area	
SDim[11] :	viscosity -> pressure time	dynamic viscosity
SDim[12] :	energy -> force length	
SDim[13] :	work -> force length	
SDim[14] :	heat -> force length	
SDim[15] :	power -> energy/time	
SDim[16] :	charge -> current time	
SDim[17] :	voltage -> power/current	
SDim[18] :	emf -> voltage	
SDim[19] :	field -> voltage/length	electric field strength
SDim[20] :	resistance -> voltage/current	
SDim[21] :	conductance -> resistance^-1	
SDim[22] :	capacitance -> charge/voltage	

```
SDim[23] :      flux -> voltage time                  magnetic flux
SDim[24] :      inductance -> resistance time
SDim[25] :      dose -> energy/mass
SDim[26] :      activity -> time^-1
SDim[27] :      illuminance -> intensity steradian/area
SDim[28] :      irradiance -> power/area
SDim[29] :      entropy -> energy/temperature
SDim[30] :      conductivity -> power temperature/length
                                                      thermal conductivity
```

XDios

Solution of Diophantine equations

integer equations

S.Wolfram
Jul 1981

PDios[*eqn*, {*n1*, {*n1max*, (*n1min*:0), (*n1step*:1)}}, ...]
tests all specified values for the *ni*, printing those sets found to satisfy the equation or condition eqn.

```
PDios[$eqn,$$r] :: (Lcl[%v]; %v:Ar[Len[%v:List[$$r]],%v[$1,1]]; \
        Flat[Ar[Ar[Len[%v],List[$$r][$1,2]],Pr[$$1];List[$$1], \
        P[N[S[$eqn,Ldist[%v->List[$$2]]]]]],Len[%v]-1])
```

```
#I[1]::  <XDios

#I[2]::  PDios[x^2+y^2=z^2, {x,10}, {y,10}, {z,10}]

3        4        5
4        3        5
6        8        10
8        6        10

#O[2]:   {{3,4,5}, {4,3,5}, {6,8,10}, {8,6,10}}
```

XDisc

Polynomial discriminants

polynomial roots

J.Greif
Jul 1981

Disc[*a*, *x*]
forms the discriminant of the polynomial a in x, which must be zero if a has multiple roots.

```
Disc[$a,$x_=Symbp[$x]] :: Rslt[Expt[$x,$a]$a-$x D[$a,$x],D[$a,$x],$x]
```

XEqn

Conversion to eqn format

text-formatting – pretty-printing

J.Greif
May 1982

Eqn[*expr*, *definitions*, *eqnum*]

writes eqn *definitions* and *expr* in eqn format. If number is given, the output is between .EQ .EN delimiters, with equation number *eqnum*. (If *eqnum*<0, no equation number is produced.) Otherwise, the output is placed between delimiters as an in-line expression. It restores the old print properties of functions read in from XEqnPr when finished with output. The print forms are read in as EQN properties from XEqnPr, and are locally converted to Pr properties. The user adds his own eqn constructs by giving any function an EQN property as in XEqnPr, and adding the name of the function to the list Prlist. The user adds his own eqn macros by adding them to the list Def, or adding the list Def to his own list.

```
_Eqn[Init]::<XEqnPr
Eqn_:Tier
Eqn[$expr,$defs_=Listp[$defs],$num]::\
        (Lcl[%p,%d]; %p:LPropsav[Prlist,Pr]; \
        LProprest[Prlist,Pr,Map[_$1[EQN],Prlist]];\
        %d:Cat[$defs];\
```

remove symbolic indices

```
        If[Len[%d]>0,\
        Pr[".EQ "]; Do[i,1,Len[%d],Pr[%d[i]]]; Pr[".EN "]];\
        If[$num<0,Pr[".EQ"],Pr[".EQ ",$num],Pr[".EQ ",$num]];\
        Pr[$expr]; Pr[".EN"]; LProprest[Prlist,Pr,%p];   )

Eqn[$expr,$defs_=Listp[$defs]]::\
        (Lcl[%p,%d]; %p:LPropsav[Prlist,Pr];\
        LProprest[Prlist,Pr,Map[_$1[EQN],Prlist]];\
        %d:Cat[$defs];\
        If[Len[%d]>0,\
        Pr[".EQ "]; Do[i,1,Len[%d],Pr[%d[i]]]; Pr[".EN "]];\
        Pr["@ ", $expr, " @"]; LProprest[Prlist,Pr,%p];   )
```

initialize things

```
Eqn[]
```

utilities, perhaps should go in XLProp

```
        LPropsav_:LProprest_:Tier
```

save existing property, or Null if not set

```
        LPropsav[$list,$prop]::(Lcl[%l]; %l:Map[_$1[$prop],$list];\
                Map[If[P[Match[_$%x[$prop],$1]=0],$1,Null],%l])
```

restore prop property to all items in list from oldist

```
          LProprest[$list,$prop,$oldlst_:(Len[$list]=Len[$oldlst])]::\
                  Ap[Set,{Map['_$1[$prop],$list],$oldlst}]

#I[1]::   <XEqn

#I[2]::   Rex[38]

                 2  2
#O[2]:    2 x   z   (16 + x) (216 + 324x + 188y - 648x z

                                                 2
                              + 12y z (2 + x) (58 + y ))

#I[3]::   Eqn[%,Def]

.EQ
delim @@
define cint #"\o'\s+6\(is\s0\(ci'"#
.EN
@         2 { x } sup { 2 }  { z } sup { 2 }  (16 + x)

                      * (216 + 324x + 188y - 648x z

                              + 12y z (2 + x) (58 +  { y } sup  { 2 } ))     @

#I[4]::   Eqn[@2,Def,3]

.EQ
delim @@
define cint #"\o'\s+6\(is\s0\(ci'"#
.EN
.EQ   3
2 { x }  sup  { 2 }   { z }  sup  { 2 }  (16 + x)

           * (216 + 324x + 188y - 648x z

                   + 12y z (2 + x) (58 +  { y }  sup  { 2 } ))
.EN

#I[5]::   <end>
```

Warning: This is a prototype only. It is incomplete and preliminary.

XEqnPr

Printing forms for eqn output

typesetting – two-dimensional output – UNIX – TROFF
text processing – word processing

J.Greif

Jun 1982

```
Lb:" { "
Rb:" } "
Nl:""""""""
_Pow[EQN][[$a,$b]]:Fmt[,Lb,$a,Rb," sup ",Lb,$b,Rb]
_Pow[EQN][[$a,1/2]]:Fmt[," sqrt ",Lb,$a,Rb]
_Div[EQN][[$a,$b]]:Fmt[,Lb,$a,Rb," over ",Lb,$b,Rb]
_Sum[EQN][[$ex,{$v,$strt,$end}]]:\
        Fmt[," sum from ",Lb,$v=$strt,Rb," to ",Lb,$end,Rb,$ex]
_Prod[EQN][[$ex,{$v,$strt,$end}]]:\
        Fmt[," prod from ",Lb,$v=$strt,Rb," to ",Lb,$end,Rb,$ex]
_Int[EQN][[$ex,{$v,$strt,$end}]]:\
        Fmt[," int from ",Lb,$strt,Rb," to ",Lb,$end,Rb,$ex,,"d",$v]
_Uneq[EQN][[$$x]]:Sx[" != ",{$$x}]
_Err[EQN][[$a,$b]]:Sx[" +- ",{$a,$b}]
_Union[EQN][[$$x]]:Fmt[," union ",{$$x}]
_Inter[EQN][[$$x]]:Fmt[," inter ",{$$x}]
_D[EQN][[$y,{$x,$n,$z}]]:Fmt[," left ",Nl,Lb," partial "^$n,$y,Rb," over ",Lb,\
         "partial ",$x^$n,Rb," right ","," | sub ",Lb,$x=$z,Rb]
```

examples of user functions

```
_Dsum[EQN][$$x]:Sx[" ciplus ",{$$x}]               direct sum
_Cint[EQN][$a,$x]:Fmt[," cint ",$a,,"d",$x]        contour integral
```

example of user definitions

```
Def[cint]:"define cint #""\o'\s+6\(is\s8\(ci'""#"
                                                   a contour integral sign
Def[delim]:"delim @@"
```

list of functions whose `Pr` properties will be saved

```
Prlist:{'Pow,'Div,'Sum,'Prod,'Uneq,'Err,'Union,'Inter,'D,'Dsum,'Cint}
```

XEuIV

Euler numbers and polynomials

S.Wolfram
Feb 1982

Explicit forms for integer values of index.

```
Eul[$n_=Oddp[$n]] : 0

Eul[$n_=Oddp[$n],$x] : 0

Eul[$n_=Evenp[$n]] : 2^$n Eul[$n,1/2]

Eul[$n_=Natp[$n/2],$x] : 2^($n+1)/($n+1) (Ber[$n+1,($x+1)/2] - \
                                 Ber[$n+1,$x/2])
```

[MOS sect. 1.5.2]

XEulgam

Generalized Euler-Mascheroni constants

S.Wolfram
Feb 1982

Eulgam[*n*]

represents the *n* th generalized Euler-Mascheroni constant.

```
Eulgam[0] : Euler
N[Eulgam[$n_,$n>0]] ::\
          N[Sum[Log[i]^$n/i,{i,1,20}]-Log[20]^($n+1)/($n+1)]
```

[AS sect. 23.2]

• Euler

Misfeatures: Numerical accuracy of sum should be investigated.

XExDot

Dot product expansion

scalar products – distribution – reduction – vectors scalars

S.Wolfram
Jul 1981

x_Scal
declares x to be a scalar.

Scalp[*expr*]
tests whether *expr* contains only scalar objects.

```
Scalp[$expr] :: Ap[And,Map[P[_$1[Type]=Scal],Cont[$expr]]]
```

ExDot[*expr*]
factors all declared scalars out of dot products.

```
ExDot[$expr] :: S[$expr,$$x.($a_/Scalp[$a]).$$y-->$a $$x.$$y, \
        ($a_/Scalp[$a]).$$x-->$a $$x, $$x.($a_/Scalp[$a])-->$a $$x,\
        $$x.(($a_/Scalp[$a]) $$b).$$y-->$a $$x.$$b.$$y, \
        (($a_/Scalp[$a]) $$b).$$x-->$a $$b.$$x, \
        $$x.(($a_/Scalp[$a]) $$b)-->$a $$x.$$b, Inf]
```

ExMDot[*expr*]
sets all dot products of a matrix with its inverse to the identity.

```
ExMDot[$expr] :: S[$expr,Minv[$m].$m->1,$m.Minv[$m]->1,Inf]
```

```
#I[1]::  <XExDot

#I[2]::  x_y_Scal

#O[2]:   Scal

#I[3]::  Scalp[x+y^2+1]

#O[3]:   1

#I[4]::  Scalp[a+x]

#O[4]:   0

#I[5]::  ExDot[a.(x b).(1+y).c]

#O[5]:   x a.b.c (1 + y)

#I[6]::  ExMDot[a.Minv[b].b.Minv[a].c]

#O[6]:   c
```

XFPow

Functionals

Functional powers — iterated functions — nested functions

S.Wolfram
Jul 1981

FPow[f, n, x]
yields n nested applications of f to x.

```
FPow_:Tier
_FPow[Smp]:{0,Inf,Inf}
FPow[$f,$n_=Natp[$n],$x] :: (Lcl[%e];%e:$x;Rpt[%e:Ap[$f,{%e}],$n])
FPow[$f,0,$x] : $x
```

```
#I[1]::  <XFPow

#I[2]::  FPow[f,10,x^2]

                                   2
#O[2]:   f[f[f[f[f[f[f[f[f[f[x ]]]]]]]]]]

#I[3]::  Ex[FPow[a $x(1-$x),3,x]]

          3       3  2     4  2       4  3     4  4     5  2       5  3     5  4
#O[3]:   a  x - a  x  + a  x  - 2 a  x  + a  x  + a  x  - 2 a  x  + a  x

              6  3       6  4       6  5       6  6     7  4       7  5
          + 2 a  x  - 6 a  x  + 6 a  x  - 2 a  x  + a  x  - 4 a  x

              7  6       7  7     7  8
          + 6 a  x  - 4 a  x  + a  x
```

XFib

Fibonacci numbers

Golden ratio

S.Wolfram
Jul 1981

Fib[*n*]

yields the *nth* Fibonacci number.

```
Fib[$n_Natp[$n]] :: Floor[N[!N[Phi^$n/Sqrt[5]+1/2]]]
Fib[$n] :: N[(Phi^$n-(-Phi)^(-$n))/Sqrt[5]]
```

XFierz

Fierz transformations

Dirac bilinear covariants – Dirac gamma matrices – fermion factors
Clifford algebra – completeness relations

S.Wolfram

Jul 1981

DIM

denotes number of dimensions (default 4).

```
DIM : 4
```

Fz[*k*, *l*]

yields elements of the Fierz rearrangement matrix.

```
Fz[$k,$l] :: (-1)^$l (DIM-2$l)/$k Fz[$k-1,$l] - \
                     (DIM-$k+2)/$k Fz[$k-2,$l]
Fz[0,$l] :: -1/2^Floor[DIM/2]
Fz[1,$l] :: (-1)^($l+1) (DIM-2$l)/2^Floor[DIM/2]
```

Fierz[*dim*]

yields the complete Fierz transformation matrix for any natural number of dimensions *dim*.

```
Fierz[$dim_=Natp[$dim]] :: (Lcl[DIM]; DIM:$dim; \
         If[Evenp[DIM],Ar[{{0,DIM},{0,DIM}},Fz], \
                  Ar[{{0,(DIM+1)/2},{0,(DIM+1)/2}},Fz]])
```

```
#I[1]::  <XFierz

#I[2]::  Fierz[3]

#O[2]:   {[0]: {[0]: -1/2, [1]: -1/2, [2]: -1/2},

          [1]: {[0]: -3/2, [1]: 1/2, [2]: 1/2},

          [2]: {[0]: -3/2, [1]: 1/2, [2]: 1/2}}
```

[T.Curtright (Univ. Florida)]

XFit

Curve fitting

linear fit – power fit – exponential fit – correlation coefficient
least squares fit – parameter determination – data analysis
smoothing – functional form

S.Wolfram
Jul 1981

Fit[{{*x1,y1*},{*x2,y2*},...},{*x,y*}]
obtains a linear fit for the relation between x and y.

```
Fit[$list,{$x,$y}] :: Ap[$y=$1 + $2 $x,Ap[Fit0,Trans[$list]]]

Fit0[$x,$y] :: (Lcl[%b0,%b1]; %b1:(Len[$x] Ap[Plus,$x $y] - \
Ap[Plus,$x] Ap[Plus,$y]) / (Len[$x] Ap[Plus,$x^2] - \
Ap[Plus,$x]^2); %b0:(Ap[Plus,$y] - %b1 Ap[Plus,$x])/Len[$x]; \
                {%b0,%b1} )
```

FitExp[{{*x1,y1*},{*x2,y2*},...},{*x,y*}]
obtains an exponential fit of the form $y = a\ b\hat{}x$ for the relation between x and y.

```
FitExp[$list,{$x,$y}] :: Ap[N[$y = Exp[$1] Exp[$2]^$x], \
                    Ap[Fit0[$%1,N[Log[$%2]]],Trans[$list]]]
```

FitPow[{{*x1,y1*},{*x2,y2*},...},{*x,y*}]
obtains a power fit of the form $y = a\ x\hat{}b$ for the relation between x and y.

```
FitPow[$list,{$x,$y}] :: Ap[N[$y = Exp[$1] $x^$2], \
                    Ap[Fit0,N[Log[Trans[$list]]]]]
```

Corr[{{*x1,y1*},{*x2,y2*},...}]
yields the population correlation coefficient between the xi and yi.

```
_Corr[Init] :: <XStat
Corr[$list] :: N[Lcl[%x,%y]; {%x,%y}:Trans[$list]; \
Ap[Plus,(%x - Mean[%x])(%y - Mean[%y]) / Sqrt[Var[%x] Var[%y]]]]
```

```
#I[1]::  <XFit

#I[2]::  t:Ar[5,N[{$,2Exp[$]}]]

#O[2]:   {{1,5.436564},{2,14.77811},{3,40.17107},{4,109.1963},{5,296.8263}}

#I[3]::  Fit[%,{x,y}]

#O[3]:   109.8776 + y = 67.71977x

#I[4]::  S[%,x->3]

#O[4]:   y = 93.28167

#I[5]::  FitPow[t,{x,y}]

                        2.421572
#O[5]:   y = 3.953884 x

#I[6]::  FitExp[t,{x,y}]

                        x
#O[6]:   y = 2 2.718282
```

```
#I[7]::  Corr[t]
#O[7]:   .3278786
```

Future enhancements: Should give error bars on fitted parameters.

XFun

Function generation

lambda expression – make generic – create function
create pattern

S.Wolfram
Jul 1981

Fun[*expr*, {*a*, *b*, *c*, . . .}]

yields a list to be used as the value of a symbol whose projections give values for *expr* with filters corresponding to *a*,*b*,... in that order.

```
Fun_:Tier
Fun[$expr,$list_←Listp[$list]] :: (Lcl[%l,%a]; %l:Map[Dummy,$list]; \
        Ap[Set,{Ap[%a,%l],S[$expr,Ldist[$list->%l]]}]; Ret[%a])

Dummy[$s_←Symbp[$s]] :: Make["$",$s]
```

XG

Dirac algebra

gamma matrix algebra — Chisholm identities — trace identities
Feynman diagrams — electrodynamics — quantum field theory

S.Wolfram
Jul 1981

Ginds

is a list of all symbols assigned type Gind.

```
Ginds :: Rel[Cont[,_$1[Type]=Gind]]
```

VCon[*expr*]

contracts any repeated pairs of indices in dot products in *expr*.

```
Con[$expr] :: S[Ex[$expr,,,,,In['Vdot,$1]], \
          Vdot[($%mu_/ _$%mu[Type]=Gind),$%mu] -> Ndim, \
          Vdot[$%mu_/ _$%mu[Type]=Gind,$%v1] Vdot[$%mu,$%v2] -> \
                        Vdot[$%v1,$%v2], Inf]
```

Transformations of products of Dirac gamma matrices.

```
SG[1] : ('G[$$x,$p,$q_/(Ord[$p,$q]<0),$$y]) -> 2('G[$$x,$$y]) $p[1].$q[1] - \
                                                  'G[$$x,$q,$p,$$y]

SG[2] : ('G[$p,$q_/(Ord[$q,$p]<0)]) --> 2 $p[1].$q[1] - G[$q,$p]

SG[3] : ('G[$$x,$p,$q_/(Ord[$q,$p]<0)]) --> 2 $p[1].$q[1] - G[$$x,$q,$p]

SG[4] : ('G[$p,$q_/(Ord[$q,$p]<0),$$x]) --> 2 $p[1].$q[1] - G[$q,$p,$$x]
```

Chisholm's identity in four dimensions.

```
SG[5] : ('G[$mu_/ _$mu[Type]=Gind,$$x_/Oddp[Len[$$x]],$mu]) --> -2 Rev[$$x]
```

XGFit

General least squares fitting

Regression – statistics – curve fitting – data analysis
parameter fitting – function fitting

S.Wolfram and P.Leyland

Feb 1982

GFit[{{*x1,y1*},{*x2,y2*},...},*form*,{*par1,par2,...*}]
finds values of the parameters *par1*, *par2*, .. which yield the least square deviation of the template *form* from the curve specified by the points {*x1,y1*}, {*x2,y2*}, ...

```
GFit_:Tier
GFit[$list_=Listp[$list],$f,$pars_=Listp[$pars]] :: (Lcl[%d]; \
        %d:Ap[Plus,(Map[$f,Trans[$list][1]]-Trans[$list][2])^2]; \
        Sol[Ldist[Ar[Len[$pars],D[%d,$pars[$1]]]=0],$pars][1])
```

```
#I[1]::  <XGFit

#I[2]::  t:Cat[Ar[{{0,1,0.1}},{$1,N[Exp[$1]]}]]

#O[2]:   {{0,1},{0.1,1.10517},{0.2,1.2214},{0.3,1.34986},{0.4,1.49182},
            {0.5,1.64872},{0.6,1.82212},{0.7,2.01375},{0.8,2.22554},
            {0.9,2.4596}}

#I[3]::  form:a0+a1 $1+a2 $1^2

                          2
#O[3]:   a0 + a1 $1 + a2 $1

#I[4]::  pars:{a0,a1,a2}

#O[4]:   {a0,a1,a2}

#I[5]::  GFit[t,form,pars]

#O[5]:   {a0 -> 1.0064,a1 -> 0.88906,a2 -> 0.79764}

#I[6]::  GFit[t,a0+a1 $1+a2 $1^3,{a0,a1,a2}]

#O[6]:   {a0 -> 0.998118,a1 -> 1.16909,a2 -> 0.588704}
```

Future enhancements: Errors etc.

XGQInt

Gaussian quadrature integration

numerical integration – approximate integration – numerical quadrature
symbolic-numeric interface

GQInt[*f*, {*x*, *lo*, *hi*}, *npt*]
forms the numerical integral of *f* with respect to *x* between *lo* and *hi* using *npt*-point Gaussian quadrature.

```
GQInt[$f,{$x,$lo,$hi},$npt_=In[$npt,{6,12,24}]] :: \
                N[($hi-$lo)/2 Sum[S[$f,$x->      \
                (($hi-$lo)GQx[$npt,i]+($hi+$lo))/2] GQw[$npt,i], \
                                 {i,-$npt/2,$npt/2}]]
GQx[6,0]:GQw[6,0]:0
GQx[6,1]:0.2386191860
GQx[6,2]:0.6612093864
GQx[6,3]:0.9324695142
GQx[6,-1]:-GQx[6,1]
GQx[6,-2]:-GQx[6,2]
GQx[6,-3]:-GQx[6,3]
GQw[6,1]:GQw[6,-1]:0.4679139345
GQw[6,2]:GQw[6,-2]:0.3607615730
GQw[6,3]:GQw[6,-3]:0.1713244923
```

```
#I[1]::   <XGQInt;GQInt:;<XGQInt

#I[2]::   GQInt[f[x],{x,-1,1},6]

#O[2]:    .1713245f[-.9324695] + .3607616f[-.6612094] + .4679139f[-.2386192]

                  + .4679139f[.2386192] + .3607616f[.6612094]

                  + .1713245f[.9324695]

#I[3]::   GQInt[x^2,{x,0,1},6]

#O[3]:    .3333333
```

XGammaS

Gamma function

Euler Gamma function – Euler integral of first kind

S.Wolfram
Feb 1982

Functional equations

Recurrence relations

```
SGamma[1,1] : Gamma[$z] -> ($z-1) Gamma[$z-1]

SGamma[1,2] : Gamma[$z] -> Gamma[$z+1]/$z
```

CanGam[*expr*, *reps*]
applies recurrence relations until the arguments of all Gamma functions in *expr* are canonical, and vanish when the replacements *reps* are applied.

```
CanGam[$expr,$$reps] :: S[$expr,Gamma[$1_Natp[Ex[S[$1,$$reps]]]] --> \
        (Lcl[%z,%n]; %n:Ex[S[$1,$$reps]]; %z:Ex[$1-%n]; \
        Prod[%z+%i,{%i,0,%n-1}] Gamma[%z]))

#I[1]::  <XGammaS

#I[2]::  t:Gamma[2-e/2]+Gamma[4+e]Gamma[1-e]

#O[2]:   Gamma[2 - e/2] + Gamma[1 - e] Gamma[4 + e]

#I[3]::  CanGam[t,e->0]

         -e Gamma[-e/2] (1 - e/2)
#O[3]:   ------------------------
                    2

                    2
              - e  Gamma[-e] Gamma[e] (1 + e) (2 + e) (3 + e)

#I[4]::  CanGam[t,e->1]

#O[4]:   Gamma[2 - e/2] + e Gamma[-1 + e] Gamma[1 - e] (-1 + e) (1 + e) (2 + e)

                             * (3 + e)
```

• XGammaV

XGammaV

Gamma function

Euler Gamma function – Euler integral of first kind

S.Wolfram
Feb 1982

Definitions for special values

```
Gamma[1] : 1

Gamma[2] : 1

Gamma[$n_,Natp[$n]] : ($n-1)!

Gamma[1/2] : Sqrt[Pi]

Gamma[$n_,Natp[$n-1/2]] : Pi^(1/2) 2^(1/2-$n) (2$n-2)!!

Gamma[-1/2] : -2 Sqrt[Pi]

Gamma[$n_,Natp[1/2-$n]] : (-2)^(1/2-$n) Sqrt[Pi]/(-2$n)!!
```

Derivatives

```
D[Gamma[$1],{$1,1,$x}] : Gamma[$x] Psi[$x]
```

[AS sect. 6.1; GR sect. 8.3; MOS sect. 1.1]

XGenocchi

Genocchi numbers

S.Wolfram
Sep 1982

Genocchi[*n*]
n th Genocchi number.

```
Genocchi[$n] :: 2^(2-2$n) $n Eul[2$n-1]
```

[Sloane: Handbook of Integer Sequences, sect. 3.13]

XGenp

Generic symbol test

dummy symbol test – generic predicate

S.Wolfram
Jul 1981

Genp[*expr*]
yields 1 if *expr* contains generic symbols, and 0 otherwise.

```
Genp[$expr] :: P[Len[Cont[$expr,_$%1[Gen]]]]
```

XGr

Basic graph theory

network theory – nodes – arcs – incidence matrix
adjacency matrix – graph representation – graph equivalence
graph isomorphism – Euler circuits – graph traversibility
Hamilton circuits

S.Wolfram

Jul 1981

A non-directed graph is represented by its incidence matrix, which specifies the number of arcs (edges) connecting each pair of nodes.

Nodes[*list*]
gives the number of nodes in the graph represented by *list*.

```
Nodes[$list] :: Len[$list]
```

Arcs[*list*]
gives the number of arcs in the graph represented by *list*.

```
Arcs[$list] :: Ap[Plus,Flat[$list]]/2
```

Regs[*list*]
gives the number of regions (faces) for planar graphs represented by *list*.

```
Regs[$list] :: Arcs[$list] - Nodes[$list] + 2
```

Degree[*list*, *n*]
yields the degree of node n in the graph represented by *list*.

```
Degree[$list,$n] :: Ap[Plus,$list[$n]]
```

ToArc[*list*]
converts the incidence matrix *list* to a list of `Arc` projections representing arcs.

```
ToArc[$list] :: Map[Arc,Flat[Ar[Dim[$list], \
                              Repl[{$1,$2},$list[$1,$2]]],1]]
```

ToNode[*list*]
converts a list of arcs to an incidence matrix.

```
ToNode[$list] :: Ar[Ar[2,`Ap[Max,Flat[S[$list,Arc->List]]]], \
                    Len[Pos[Arc[List[$$x]],$list]]]
```

Eulerp[*list*]
tests whether the graph represented by the incidence matrix *list* is Euler traversable.

```
Eulerp[$list] :: Ap[Plus,Map[Oddp,Map[Ap[Plus,$1],$list]]] <= 2
```

Hamp[*list*]
tests whether the graph represented by *list* is Hamilton traversable.

```
Hamp[$list] :: (Lcl[%u]; %u:Ar[Len[$list],0]; %u[1]:1; Hamp0[$list,1])
Hamp0[$list,$n] :: (Lcl[%t,%i]; %t:%i:0; Loop[%i<=Len[$list], \
        If[%u[$list[$n,%i]],0,%u[%i]:1;If[~(%t:Hamp0[$list,%i]), \
                %u[%i]:0]];Inc[%i],~%t]; %t)
```

Relab [*list*, *perm*]

relabs the nodes in the graph represented by the incidence matrix *list* according to the permutation *perm*.

```
_Relab[Init] :: <<XPerm0
Relab[$list,$perm] :: Apper[$perm,Map[Apper[$perm,$1],$list]]
```

Isop [*gr1*, *gr2*]

tests for isomorphism of the graphs represented by incidence matrices *gr1* and *gr2*.

```
Isop[$gr1,$gr1] : 1
Isop[$gr1,$gr2_/Nodes[$gr1]~=Nodes[$gr2]] : 0
Isop[$gr1,$gr2_/Arcs[$gr1]~=Arcs[$gr2]] : 0
Isop[$gr1,$gr2_/Isop1[$gr1]~=Isop1[$gr2]] : 0
        Isop1[$list] :: Sort[Map[Ap[Plus,$1],$list]]
Isop[$gr1,$gr2_/~P[Isop2[$gr1]=Isop2[$gr2]]] : 0
        Isop2[$list] :: Ex[Det[$list-%lam Ar[Dim[$list]]]]
```

```
#I[1]::  <<XGr

#I[2]::  g:{{0,2,0,0},{2,0,1,1},{0,1,0,1},{0,1,1,0}}

#O[2]:   {{0,2,0,0},{2,0,1,1},{0,1,0,1},{0,1,1,0}}

#I[3]::  Nodes[g]

#O[3]:   4

#I[4]::  Arcs[g]

#O[4]:   5

#I[5]::  Regs[g]

#O[5]:   3

#I[6]::  Degree[g,2]

#O[6]:   4

#I[7]::  ToArc[g]

#O[7]:   {Arc[{1,2}],Arc[{1,2}],Arc[{2,1}],Arc[{2,1}],Arc[{2,3}],Arc[{2,4}],

                Arc[{3,2}],Arc[{3,4}],Arc[{4,2}],Arc[{4,3}]}

#I[8]::  ToNode[%]

#O[8]:   {{0,2,0,0},{2,0,1,1},{0,1,0,1},{0,1,1,0}}

#I[9]::  Eulerp[g]

#O[9]:   1

#I[10]::  Hamp[g]

#O[10]:   0

#I[11]::  Relab[g,{3,1,2,4}]

#O[11]:   {{0,0,1,1},{0,0,2,0},{1,2,0,1},{1,0,1,0}}
```

```
#I[15]::  Isop[g,g]
#O[15]:   1
#I[16]::  Isop[g,Sort[g]]
#O[16]:   8
```

XHalt

Solution to halting problem

insoluble problem – oracle – Church's thesis

S.Wolfram
Jul 1981

Haltp[*prog*]
yields 1 if *prog* halts after a finite time.

```
Haltp[$prog] :: ($prog;1)
```

[Turing: A model for computation with applications to the ...]

XHarm

Harmonic sequence

S.Wolfram
Jul 1981

Harm[*n*]

represents the *nth* partial sum of the harmonic sequence.

```
Harm[$n] :: Sum[1/%i,{%i,1,$n}]
```

XHist

Histogram generation

data presentation – binning – discretization – quantization

S.Wolfram
Jul 1981

Hist[*list*, *nbin*]

forms a histogram of the contents of *list* with *nbin* bins.

```
Hist[$list_=Listp[$list],$nbin_=Natp[$nbin]] :: \
(Lcl[%min,%max,%step,%hist,%ind]; %step:N[((%max:Ap[Max,$list])-\
(%min:Ap[Min,$list]))/$nbin]; %hist:Ar[{{%min,%max,%step}},0];\
Do[%i,Len[$list],%ind:%min+%step Floor[$list[%i]/%step]; \
                %hist[%ind]:%hist[%ind]+1];%hist)
```

```
#I[1]::  <XHist

#I[2]::  t:Ar[10,Prime]

#O[2]:   {2,3,5,7,11,13,17,19,23,29}

#I[3]::  Hist[t,3]

#O[3]:   {[2]: 4, [11]: 3, [20]: 2, [29]: 1}
```

XHof

Hofstadter's recursive function

double recursion – generalized Fibonacci sequence – recursion testing

S.Wolfram
Jul 1981

Hof[*n*]

gives a recursive function defined by Hofstadter, whose values have several properties of randomness.

```
Hof[1]:Hof[2]:1
Hof[$n_=Natp[$n]] :: Hof[$n]:Hof[$n-Hof[$n-1]]+Hof[$n-Hof[$n-2]]
```

XHorn

Horner representation

numerical evaluation of polynomials – polynomial rearrangement

S.Wolfram
Jul 1981

Horn[*poly*, *x*]

constructs a Horner representation of the polynomial *poly* with respect to *x*.

```
Horn[$poly,$x] :: (Lcl[%p,%n]; If[(%n:Expt[$x,$poly])<=1,Ret[$poly]]; \
%p:Coef[$x^%n,$poly]; Do[%1,%n-1,%p:%p $x + Coef[$x^(%n-%1),$poly]]; \
$x %p + S[$poly,$x->0])
```

```
#I[1]::  <XHorn

#I[2]::  t:x^3+4a x^2+2x+c

                        2     3
#O[2]:   c + 2x + 4a x  + x

#I[3]::  Horn[%,x]

#O[3]:   c + x (2 + x (4a + x))
```

XInd

List index manipulation

S.Wolfram
Jul 1981

LInd[*list*]
yields a list of the indices in *list*.

```
LInd[$list] :: Ar[Len[$list],Ind[$list,$1]]
```

ToL[*list*]
writes entries in *list* as {index,value}.

```
ToL[$list] :: Ar[Len[$list],{Ind[$list,$1],Elem[$list,{$1}]}]
```

Maxind[*list*]
yields the maximal index in *list*.

```
Maxind[$list_Contp[$list]] :: Len[$list]
Maxind[$list] :: Ap[Max,LInd[$list]]
```

ToInd[*list*]
takes sublists {index,value} in *list*, and forms an indexed list [index]:value.

```
ToInd[$list] :: (Lcl[%l]; Map[%l[$1[1]]:$1[2],$list]; %l)
```

```
#I[1]::   <XInd

#I[2]::   t:{[a]:x^2,[b]:x^3,[c]:x+y}

                   2        3
#O[2]:    {[a]: x , [b]: x , [c]: x + y}

#I[3]::   LInd[%]

#O[3]:    {a,b,c}

#I[4]::   ToL[t]

              2      3
#O[4]:    {{a,x },{b,x },{c,x + y}}

#I[5]::   ToInd[%]

                                 3         2
#O[5]:    {[c]: x + y, [b]: x , [a]: x }
```

XIndep

Independent variable declaration

total derivatives – partial derivatives – differentials
functional independence

S.Wolfram
Jul 1981

Indep[{*y1*, *y2*, . . .}, {*x1*, *x2*, . . .}]
defines the *yi* to be independent of the *xj* so that Dt[$yi,$xj]:0.

```
Indep[$y,$x] :: Map[Map[Dt[$1,$2]:0,$x],$y]

#I[1]::  <XIndep
#I[2]::  Dt[{y1,y2,y3},x]
#O[2]:   {Dt[y1,x],Dt[y2,x],Dt[y3,x]}
#I[3]::  Indep[{y1,y2},{x,xp}]
#O[3]:   {{0,0},{0,0}}
#I[4]::  Dt[{y1,y2,y3},x]
#O[4]:   {0,0,Dt[y3,x]}
```

XInfo

Basic information theory

Shannon entropy – frequency – language analysis

S.Wolfram
Jul 1981

Shan[*prob*]

represents the Shannon entropy for a language whose symbols have relative frequencies as given in the list *prob*.

```
Shan[$prob_=Listp[$prob]] :: -N[Ap[Plus,Map[$x Log[$x,2],$prob]]/   \
                                     Ap[Plus,$prob]]
```

Freq[*list*]

yields a *list* of the relative frequencies of symbols appearing as elements of *list*.

```
Freq[$list_=Listp[$list]] :: (Lcl[%f]; Map[%f[$1[1]]:$1[2], \
                  Aex[$list][2]]; Rel[%f])
```

```
#I[1]::  <XInfo

#I[2]::  Ar[10,$x^2]

#O[2]:   {1,4,9,16,25,36,49,64,81,100}

#I[3]::  Shan[%]

#O[3]:   -5.817603

#I[4]::  {a,a,b,c,a,c,d,e,a,b,c,b,b,e}

#O[4]:   {a,a,b,c,a,c,d,e,a,b,c,b,b,e}

#I[5]::  Freq[%]

#O[5]:   {[e]: 2, [d]: 1, [c]: 3, [b]: 4, [a]: 4}
```

XIntp

Integer testing simplification

S.Wolfram
Jul 1981

```
Intp[($$x_=Intp[$$x])+($y_=Intp[$y])] : 1
Intp[($$x_=Intp[$$x])($y_=Intp[$y])] : 1

Intp[($n_=Intp[$n])^($m_=Natp[$m])] : 1
Intp[($n_=Abs[$n]>1)^($m_=Natp[-$m])] : 0

#I[1]::  <XIntp

#I[2]::  Intp[x]:1

#O[2]:   1

#I[3]::  Intp[x^2-2]

#O[3]:   1
```

XIter

General iterated forms

replication – iteration – generalized sum – generalized product

S.Wolfram

Jul 1981

Iter[*f*, *expr*, *var*, *lo*, *hi*]
applies *f* to the set of values of *expr* attained when *var* takes on values *lo*, *lo*+1, *lo*+2, ... , *hi*.

```
Iter_:Tier
_Iter[Smp]:{0,0,0,Inf,Inf}
Iter[$f,$expr,$var,$lo,$hi_=Intp[$hi-$lo]] :: \
        Ap[$f,Ar[{{$lo,$hi}},S[$expr,$var->$1]]]
```

```
#I[1]::  <XIter

#I[2]::  Iter[f,x+i^2,i,1,6]

#O[2]:   f[1 + x,4 + x,9 + x,16 + x,25 + x,36 + x]
```

XItp

Lagrange interpolation of list values

interpolation – smoothing – extrapolation

S.Wolfram

Jul 1981

Itp[*list*, *x*]

uses all the values given in *list* to yield an optimal estimate for the value corresponding to an index *x*.

```
Itp_:Tier
Itp[$list,$x] :: (Lcl[%n]; %n:Len[$list]; \
        Sum[%x:Ind[$list,i];Prod[$x-Ind[$list,j],{j,1,%n}]/ \
        (($x-%x) Prod[%x-Ind[$list,j],{j,1,%n,1,j~=i}]) \
                Elem[$list,{i}],{i, 1, Len[$list]}])
```

```
#I[1]::  <XItp

#I[2]::  t:Ar[{{0,1,0.2}},N[Sin[$x]]]

#O[2]:   {[0]: 0, [.2]: .1986693, [.4]: .3894183, [.6]: .5646425,

               [.8]: .7173561, [1]: .841471}

#I[3]::  Itp[%,x]

#O[3]:   25.8684x (-1 + x) (-4/5 + x) (-3/5 + x) (-2/5 + x)

                - 101.411x (-1 + x) (-4/5 + x) (-3/5 + x) (-1/5 + x)

                + 147.0423x (-1 + x) (-4/5 + x) (-2/5 + x) (-1/5 + x)

                - 93.48574x (-1 + x) (-3/5 + x) (-2/5 + x) (-1/5 + x)

                + 21.91331x (-4/5 + x) (-3/5 + x) (-2/5 + x) (-1/5 + x)

#I[4]::  Ex[%]

                                   2              3                4
#O[4]:   .999978x + .0002439154 x  - .1676164 x  + .001612961 x

                              5
                + .007252478 x

#I[5]::  N[S[%,x->0.8]]

#O[5]:   .7173561

#I[6]::  N[Sin[0.8]]

#O[6]:   .7173561
```

• XLItp

XKillIO

Input/Output removal

reclaim memory – conserve memory – save memory – forget past
destroy input – destroy output – kill input – kill output
kill labels – kill lines

S.Wolfram

Jul 1981

Updated Aug 1982

KillIO[(*n1*:0),(*n2*:Len[#I])]
removes values assigned to #I and #O lines numbered *n1* through *n2*.

```
KillIO_:Tier
KillIO[$n1_=Natp[$n1],$n2_=Natp[$n2]] ::   \
                   (Ar[{{$n1,$n2}},#I[$%1]:#O[$%1]:];)
KillIO[] :: (#I:#O:)
```

```
#I[1]::  <XKillIO

#I[2]::  t:x;

#I[3]::  Rpt[t:t(1+t),2]

#O[3]:   x (1 + x) (1 + x (1 + x))

#I[4]::  Ex[%]

                2       3    4
#O[4]:   x + 2 x  + 2 x  + x

#I[5]::  Fac[%]

                                2
#O[5]:   x (1 + x) (1 + x + x )

#I[6]::  KillIO[1,3]

#I[7]::  #O

                                                                 2
#O[7]:   {[6]: {[1]: , [2]: , [3]: }, [5]: x (1 + x) (1 + x + x ),

                        2       3    4
              [4]: x + 2 x  + 2 x  + x , [0]: {"/u1/smp/ALL/smp.init"}}

#I[8]::  KillIO[]

#I[9]::  #O

#O[9]:   {[8]: }

#I[10]::  #I

#O[10]:   {[9]:: #O, [8]:: KillIO[]}
```

XLArith

List arithmetic

S.Wolfram
Jul 1981

LSum[*list*]
yields the sum of all elements in the list or set of nested lists *list*.

```
LSum[$list] :: Ap[Plus,Flat[$list]]
```

LProd[*list*]
yields the product of elements in *list*.

```
LProd[$list] :: Ap[Mult,Flat[$list]]
```

```
#I[1]::  <XLArith
#I[2]::  {{a,b},{1,2,3},d}
#O[2]:   {{a,b},{1,2,3},d}
#I[3]::  LSum[%]
#O[3]:   6 + a + b + d
#I[4]::  LProd[@2]
#O[4]:   6a b d
```

XLCM

Lowest common multiple

S.Wolfram
Jul 1981

LCM[*n1*, *n2*, ...]
yields the lowest common multiple of *n1*,*n2*....

```
LCM_:Flat
LCM[$n1,$n2] :: ($n1 $n2)/Gcd[$n1,$n2]
```

XLChi2

List chi squared evaluation

goodness of fit – function fitting – curve fitting
model comparison – model fitting

S.Wolfram
Jan 1982

XLChi2[*list*, *form*, *i*]
forms the chi squared between elements of *list* and *form* as a function of *i*.

```
LChi2[$list_=Listp[$list],$form,$i] :: \
        Sum[N[($list[i]-S[$form,$i->i])^2],{i,1,Len[$list]}]
```

```
#I[1]::  <XLChi2

#I[2]::  Ar[5,N[Exp[$1]]]

#O[2]:   {2.71828,7.38906,20.0855,54.5982,148.413}

#I[3]::  LChi2[%,1+x+x^2/2,x]

#O[3]:   18747.8
```

XLDiff

Differences of list elements

forward differences – pairwise differences – pairwise subtraction

S.Wolfram
Jan 1982
Updated Aug 1982

LDiff[*list*, (n:1)]
yields a list of n th rank forward differences between successive entries of *list*.

```
LDiff[[$list_=Listp[$list]]] :: Ar[Len[$list]-1,$list[$1+1]-$list[$1]]
LDiff[[$list,1]] :: LDiff[$list]
LDiff[[$list,$n_=Natp[$n]]] :: LDiff[LDiff[$list,$n-1]]
```

```
#I[1]::  <XLDiff

#I[2]::  Ar[5,f]

#O[2]:   {f[1],f[2],f[3],f[4],f[5]}

#I[3]::  LDiff[%]

#O[3]:   {-f[1] + f[2],-f[2] + f[3],-f[3] + f[4],-f[4] + f[5]}

#I[4]::  LDiff[@2,2]

#O[4]:   {f[1] - 2f[2] + f[3],f[2] - 2f[3] + f[4],f[3] - 2f[4] + f[5]}

#I[5]::  Ar[10,$1^3-4$1^2+7]

#O[5]:   {4,-1,-2,7,32,79,154,263,412,607}

#I[6]::  Ar[3,LDiff[%,$1]]

#O[6]:   {{-5,-1,9,25,47,75,109,149,195},{4,10,16,22,28,34,40,46},
                 {6,6,6,6,6,6,6}}
```

XLItp

Interpolation of contiguous list values

Lagrangian interpolation — function evaluation

S.Wolfram
Jul 1981

LItp[*list*, *x*]
: uses all the values given in *list* to find an interpolated value for an element with index *x*.

```
LItp[$list_→Contp[$list],$x] :: \
        (Lcl[%n]; %n:Len[$list];   \
        Sum[$list[k+Floor[(%n+1)/2]] (-1)^(Floor[%n/2]+k) \
    /((k+Floor[(%n-1)/2])!*(Floor[%n/2]-k)!*($x-Floor[(%n+1)/2]-k)) \
                Prod[$x-Oddp[%n]-t, {t,Evenp[%n],2Floor[%n/2]}],\
                {k,-Floor[(%n-1)/2],Floor[%n/2]}])
```

LItp2[*list*, *x*]
: uses two-point (linear) interpolation to yield an estimate for an element of *list* with index *x*.

```
LItp2[$list,$x_→(1 <= $x <= Len[$list])] :: \
      (Lcl[%x]; %x:Floor[$x]; (1-$x+%x) $list[%x]+($x-%x) $list[%x+1])
```

LItp3[*list*, *x*]
: uses three-point Lagrange interpolation to estimate the value of an element of list with index *x*.

```
LItp3[$list,$x_→(2 <= $x <= Len[$list]-1)] :: \
        (Lcl[%x,%p]; %p:$x-(%x:Floor[$x]); \
        %p(%p-1)/2 $list[%x-1] + (1-%p^2) $list[%x] + \
                %p(%p+1)/2 $list[%x+1])
```

```
#I[1]::  <XLItp

#I[2]::  t:Ar[6,$x^4]

#O[2]:   {1,16,81,256,625,1296}

#I[3]::  LItp[t,3.5]

#O[3]:   150.0625

#I[4]::  3.5^4

#O[4]:   150.0625

#I[5]::  LItp2[t,3.5]

#O[5]:   168.5

#I[6]::  LItp3[t,3.5]

#O[6]:   154.75

#I[7]::  LItp[Ar[3],x]

         (-3 + x) (-2 + x)                          3(-2 + x) (-1 + x)
#O[7]:   ----------------- - 2(-3 + x) (-1 + x) + ------------------
                 2                                          2
```

```
#I[8]::  Ex[%]
#O[8]:   x
```

XLPart

Subpart position lists

power set

S.Wolfram
Jul 1981

LPart[*expr*]
yields a list of the positions of all subparts of *expr*.

```
LPart[$expr] :: Pos[$1,$expr]
```

APart[*expr*]
yields a list of all subparts of *expr*, together with the positions at which they appear.

```
APart[$expr] :: Pos[$1,$expr,List[$expr[$$x],List[$$x]]]
```

```
#I[1]::  <XLPart

#I[2]::  t:Rex[]

                             2              3
#O[2]:   2x (2 + 2x) + (-1 - 2 x ) (3 + y + -)
                                            x

#I[3]::  LPart[t]

#O[3]:   {{1,1},{1,2,1},{1,2,2},{1,2},{1},{2,1,1},{2,1,2,1},{2,1,2,2},{2,1,2},
                 {2,1},{2,2,1},{2,2,2},{2,2,3,1},{2,2,3,2},{2,2,3},{2,2},{2},
                 {0}}

#I[4]::  APart[t]

#O[4]:   {{x,{1,1}},{2,{1,2,1}},{2x,{1,2,2}},{2 + 2x,{1,2}},{2x (2 + 2x),{1}},

                                                                 2
                 {-1,{2,1,1}},{x,{2,1,2,1}},{2,{2,1,2,2}},{-2 x ,{2,1,2}},

                         2
                 {-1 - 2 x ,{2,1}},{3,{2,2,1}},{y,{2,2,2}},{1,{2,2,3,1}},

                                   3                           3
                 {x,{2,2,3,2}},{-,{2,2,3}},{3 + y + -,{2,2}},
                                x                           x

                           2              3
                 {(-1 - 2 x ) (3 + y + -),{2}},{' Plus,{0}}}
                                        x
```

XLProp

List property assignment

multiple assignment – property – distribution over lists

J.Greif
Jul 1982

LProp[*list*, *prop*, (*value*:1)]
assigns *value* to the *prop* property of each element of *list*.

```
LProp_:Tier
LProp[$list_=Listp[$list],$prop]::Map[Prset[$1,$prop],$list]
LProp[$list_=Listp[$list],$prop,$val_=Numbp[$val]]:: \
        Ap[Set,{Map[Prop[$1,$prop],$list],$val}]
LProp[$list_=Listp[$list],$prop,$val]::Ap[Setd,{Map[Prop[$1,$prop],\
        $list],$val}]
```

```
#I[1]::  <XLProp

#I[2]::  LProp[{s,t,u},Trace,Lpr]

#O[2]:   ' Lpr

#I[3]::  s[a+b I]/t[u[a+Ps[Exp[-x^2],x,0,1]] - 7*^456]

s[Cx[a,b]]
u[1 + a]
u[1 + a]
u[1 + a]
u[1 + a]
t[u[Ps[1,x,0,{0,3},{[0]: 1 + a,[1]: 0,[2]: -1,[3]: 0}]] + A[-7,456]]

                    s[a + b I]
#O[3]:*  ----------------------------
                             2
         t[u[(1 + a) - x ] + -7*^456]

#I[4]::  LProp[{s,t,u},Ldist]

#O[4]:   {Ldist,Ldist,Ldist}

#I[5]::  s[{x,y,z}]

{s[x],s[y],s[z]}
s[x]
s[y]
s[z]

#O[5]:   {s[x],s[y],s[z]}

#I[6]::  <end>
```

XLUP

LUP matrix methods

linear algebra – linear equation – determinant – matrix
matrix inverse

J.Greif

Jul 1982

```
<XMat3; <XPerm0; <XPerm1
```

Ldet[*m*]

finds determinant of matrix *m* by the LUP decomposition.

```
Ldet[$m_/Sqmatp[$m]] :: (Lcl[%l]; If[(%l:Lup[$m])=0,%l, \
        Tr[%l[2],Mult]*Sig[%l[3]],Tr[%l[2],Mult]*Sig[%l[3]]])
```

Lminv[*m*]

finds inverse of matrix *m* by the LUP decomposition.

```
Lminv[$m_/Sqmatp[$m]] :: (Lcl[%l]; If[P[(%l:Lup[$m])=0],Lminv[$m], \
        Trunc[Pmat[Pinv[%l[3]]].Tinv[%l[2]].Tinv[%l[1]],Len[$m]]])
```

Lmdiv[*mat*, *rhs*]

solves the matrix equation *mat*.x = *rhs* for x by the LUP decomposition.

```
Lmdiv[$mat_/Sqmatp[$mat],$rhs_/(Contp[$rhs]&Len[$rhs]=Len[$mat])] ::\
        (Lcl[%l,%d,%y]; If[P[(%l:Lup[$mat])=0],Lmdiv[$mat,$rhs],\
        %d=Embed[Trans[$rhs]]; \
        %y:Bsub[%l[1],%d]; Trunc[Bsub[%l[2].Pmat[%l[3]],%y],Len[$mat]]]
Lmdiv[$mat_/Sqmatp[$mat],$rhs_/(Matp[$rhs]&Len[$rhs]=Len[$mat])] ::\
        (Lcl[%l,%d,%y]; If[P[(%l:Lup[$mat])=0],Lmdiv[$mat,$rhs],\
        %d=Embed[$rhs]; \
        %y:Bsub[%l[1],%d]; Trunc[Bsub[%l[2].Pmat[%l[3]],%y],Dim[$rhs]]]
```

Pmat[*perm*]

converts a list denoting a permutation into the appropriate matrix.

```
Pmat[$p_/Permp[$p]]::Ar[{Len[$p],Len[$p]},$j=$p[$i]]
```

Embed[*mat*]

embeds an n x m matrix into an identity matrix of dimension l x Max[m,l] where l is the first power of two not less than n.

```
Embed[$m_/Matp[$m]] :: (Lcl[%i,%d]; %d:Dim[$m]; %i:1; \
        Loop[%i<%d[1],%i:2*%i,];If[P[%i=%d[1]],$m,\
        Ar[{%i,Max[%i,%d[2]]},If[$1<=%d[1]&$2<=%d[2],$m[$1,$2],$1=$2]]]
```

Trunc[*mat*, *dim*]

extract from a matrix *mat* the entries within the bounds given by *dim*. If *dim* is a number, extract the corresponding square matrix, otherwise extract according to the dimension list.

```
trunc_:Tier
trunc[$m_/Matp[$m], $d_/Natp[$d]] :: Ar[{$d,$d},m[$i,$j]]
trunc[$m_/Matp[$m], $d_/Listp[$d]] :: Ar[{$d[1],$d[2]},$m[$i,$j]]
```

Tinv[*m*]

finds inverse of upper or lower triangular matrix *m*. See lemmas 6.5, 6.6 of reference below.

```
Tinv[$m_=Sqmatp[$m]] :: (Lcl[%m,%d,%p,%lp]; \
        If[P[Len[$m]=1],{{1/$m[1,1]}},\
        %m:Qtr[embed[$m]]; %d:Tinv[%m[4]]; %p:Tinv[%m[1]]; %lp:Len[%p]\
       Ar[{2*%lp,2*%lp},Sel[$i<=%lp & $j<=%lp,%p[$i,$j],$i>%lp&$j>%lp,\
        %d[$i-%lp,$j-%lp],$i>%lp,(-%d.%m[3].%p)[$i-%lp,$j],\
        %j>%lp,(-%p.%m[2].%d)[$i,$j-%lp]]]]);
```

Lup[*m*]

finds LUP decomposition of matrix *m*, returned as {L,U,P} where P is in the form of a permutation (contiguous list).

```
Lup_:Tier
Lup[$m_=Matp[$m]]::Lup[$m,Len[$m],Len[$m[1]]]
Lup[$m,$n_=Natp[$n],$p_=Natp[$p]] :: (Lcl[%b,%d,%g,%h,%l,\
        %n,%p,%p3,%fe,%lup1,%lup2,%u]; %n:$n/2;\
        %b:Ar[{%n,$p},$m]; If[P[(%lup1:lup[%b,%n,$p])=0],0,\
        %d:Ar[{%n,$p},$m[%n+$1,$2]].Pmat[Pinv[%lup1[3]]];\
        B and D in notation of reference. F.(E^-1) follows\
        %fe:Ar[{%n,%n},%d].Tinv[Ar[{%n,%n},%lup1[2,$i,$j]]];\
        %g:%d-%fe.%lup1[2];\
        decompose G' and build up answer\
        If[P[(%lup2:Lup[Ar[{%n,$p-%n},%g[$i,%n+$j]],%n,$p-%n])=0],\
                Ret[0],\
        find P3, H, L, U, P as matrix and convert latter to perm\
        %p3:Ar[$p,If[$1<=%n,$1,%n+%lup2[3,$1-%n]]];\
        %h:%lup1[2].Pmat[Pinv[%p3]]; %p:Pmat[%p3].Pmat[%lup1[3]];\
        %l:Cat[Ar[{%n,$p},If[$2<=%n,%lup1[1,$1,$2],0]],\
        Ar[{%n,$p},If[$2<=%n,%fe[$1,$2],%lup2[1,$1,$2-%n]]]];\
        %u:Cat[%h,Ar[{%n,$p},If[$2<=%n,0,%lup2[2,$1,$2-%n]]]];\
        {%l,%u,%p.Ar[Len[%p]]}]])
Lup[$m,1,$p_=Natp[$p]] :: (Lcl[%j,%p];\
        find non-zero column and perm making it first\
        %p:Ar[$p]; For[%j:1,$m[1,%j]=0 & %j<=$p,,Inc[%j]];\
        Sel[%j>$p,0,%j=1,{{{1}},$m,%p},1,%p[1]:%j;%p[%j]:1;\
        {{{1}},$m.Pmat[%p],%p}])
```

Bsub[*m1*, *m2*]

backsubstitutes to find solution x of matrix equation *m1*.x=*m2* where *m1* is upper or lower triangular. TEMPORARY — do in ordinary way

```
Bsub[$m_=Sqmatp[$m],$r_=Matp[$r]] :: Mdiv[$r,$m]
```

Qtr[*m*]

partition a 2^n by 2^n matrix *m* into quarters, returning {*m11*,*m12*,*m21*,*m22*} where each element is one of the quarters.

```
Qtr[$m_=Sqmatp[$m]] :: (Lcl[%n]; %n:Len[$m]/2; \
{Ar[{%n,%n},$m],Ar[{%n,%n},$m[$1+%n,$2]],Ar[{%n,%n},$m[$1,$2+%n]],\
Ar[{%n,%n},$m[$1+%n,$2+%n]]})
```

[Aho, Hopcraft and Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA., 1974]

XLap

Laplace transforms

operational methods – differential equations

S.Wolfram
Jul 1981

Lap[*expr*, *t*, *s*]
represents the Laplace transform of expr from the *t* domain to *s* domain.

```
Gamma[$n_=Natp[$n]] : ($n-1)!

Lap_:Tier

Lap[$n_=Numbp[$n],$t,$s] : $n/$s

Lap[$t,$t,$s] : 1/$s^2

Lap[$t^$p,$t,$s] : Gamma[$p+1]/$s^($p+1)

Lap[$x + $$x,$t,$s] :: Lap[$x,$t,$s] + Lap[$$x,$t,$s]

Lap[$x $$x,$t_=~In[$t,$x],$s] :: $x Lap[$$x,$t,$s]

Lap[$x/$y,$t_=~In[$t,$y],$s] :: Lap[$x,$t,$s]/$y

Lap[$x=$y,$t,$s] :: Lap[$x,$t,$s]=Lap[$y,$t,$s]

Lap['D[$y,{$x,$n,$t}],$t,$s] ::  \
        $s^$n Lap[S[$y,$x->$t],$t,$s] -  \
                         Sum[D[$y,{$x,i-1,0}] $s^($n-i),{i,1,$n}]

Lap[Exp[$$a t],t_=~In[t,$$a],s] :: 1/(s-$$a)ns

Lap[Exp[t],t_=~In[t,$$a],s] : 1/(s-1)$$a

Lap[Exp[$$a $t] $$x,$t_=~In[$t,$$a],$s] :: Lap[$$x,$t,$s-$$a]

Lap[($t+$$a)^$p,$t_=~In[$t,{$$a,$p}],$s] ::  \
        Gamma[$p+1,$$a $s] Exp[$$a $s]/$s^($p+1)

Lap[Log[$t],$t,$s] :: -Log[Euler $s]/$s

Lap[$t^$p Log[$t],$t,$s] :: Gamma[$p+1] (Psi[$p+1]-Log[$s])/$s^($p+1)

Lap[Log[$t]^2,$t,$s] : (Zeta[2] + Log[Euler $s]^2)/$s

Lap[Sin[$$a t],t_=~In[t,{$$a}],s] : $$a/(s^2 + $$a^2) e
```

XLatsum

Lattice sums

Madelung sums – crystal energies – zeta functions

S.Wolfram

Jul 1981

Latsum[*f*, *spec*]
sums the results of applying the template *f* to the sets of lattice points specified by *spec*.

```
⊥Latsum[Smp]:{8,Inf}
Latsum[$f,$spec] :: Ap[Plus,Flat[Ar[$spec,$f]]]
```

```
#I[1]::  <XLatsum

#I[2]::  Latsum[1/($1^2+$2^3),{5,4}]

#O[2]:   1.428285

#I[3]::  Sum[Sum[1/(i^2+j^3),{j,1,4}],{i,1,5}]

#O[3]:   1.428285
```

XLdEq

Equation list distribution

matrix equations – sets of equations – simultaneous equations

S.Wolfram
Jul 1981

LdEq[*expr*]
distributes Eq over lists in *expr*, thus converting equations involving lists into lists of equations.

```
LdEq[$expr] :: Ldist[$expr,'Eq]
```

```
#I[1]::  <XLdEq
#I[2]::  {{a,b},{c,d}}.{1,2}={5,-3}
#O[2]:   {a + 2b,c + 2d} = {5,-3}
#I[3]::  LdEq[%]
#O[3]:   {a + 2b = 5,3 + c + 2d = 0}
```

XLenex

Expanded length estimation

expansion – estimated size – total length – memory requirement

S.Wolfram
Jul 1981

Lenex[*expr*]
yields an upper limit on the length of the expression resulting from expansion of *expr*.

```
Lenex[$x+$$x] :: Lenex[$x]+Lenex[$$x]
Lenex[$x $$x] :: Lenex[$x] Lenex[$$x]
Lenex[$x^$n_=Natp[$n]] :: Comb[Lenex[$x]+$n-1,$n]
Lenex[($x $$x)/$y] :: Lenex[$x] Lenex[$$x]
Lenex[Log[$x $$x]] :: Lenex[$x]+Lenex[$$x]
Lenex[$x] : 1
```

* Size; Len

#I[1]:: <XLenex

#I[2]:: t:(1+x+x^2)(a+b+1)^3

#O[2]: $(1 + a + b)^3 \; (1 + x + x^2)$

#I[3]:: Lenex[%]

#O[3]: 38

#I[4]:: Ex[t]

#O[4]: $1 + 3a + 3b + x + 6a\,b + 3a\,x + 3a\,b^2 + 3a\,x^2 + 3b\,x + 3b\,x^2 + 3\,a^2\,b$
$+ 3\,a^2\,x + 3\,a^2\,x^2 + a^3\,x + a^3\,x^2 + 3\,b^2\,x + 3\,b^2\,x^2 + b^3\,x + b^3\,x^2$
$+ 6a\,b\,x + 6a\,b\,x^2 + 3a\,b^2\,x + 3a\,b^2\,x^2 + 3\,a^2\,b\,x + 3\,a^2\,b\,x^2$
$+ 3\,a^2 + a^3 + 3\,b^2 + b^3 + x^2$

#I[5]:: Len[%]

#O[5]: 38

#I[6]:: Lenex[t^5]

#O[6]: 2856

XLer

Lerch transcendent

S.Wolfram
Jul 1981

```
SLer[1]:    Ler[1,$s,1] -> Zeta[$s]

Sler[2]:    Ler[1,$s,$a] -> Zeta[$s,$a]

SLer[3]:    Ler[$z,$s,1] -> Li[$s,$z]/$z

SLer[4]:    Ler[$z,$s,$a] -> I $z^-$a Gamma[1-$s](2Pi)^($s-1)\
                 (Exp[-I Pi $s/2]Ler[Exp[-2 Pi I $a],1-$s,Log[$z]/(2Pi I)] \
                 -Exp[I Pi($s/2+2$a)]Ler[Exp[2Pi I $a],1-$s,1-Log[$z]/(2Pi I)])
```

[MOS pp. 33-4]

XLev

Level isolation

part extraction – structural operation – domains – depth

S.Wolfram
Jul 1981

Lev[*expr*, *n*]
yields a list of parts of *expr* on level *n*.

```
Lev_:Tier
Lev[$expr,$n] :: (Lcl[%l]; %l:{}; Map[%l:Cat[%l,{$1}],$expr,{$n}]; %l)
```

Lenlev[*expr*, *n*]
finds the number of parts of *expr* at level *n*.

```
Lenlev_:Tier
Lenlev[$expr,$n] :: (Lcl[%n]; %n:0; Map[`Inc[%n],$expr,{$n}]; %n)
```

```
#I[1]::  <XLev

#I[2]::  t:Rex[]

                2
#O[2]:   56 x    (x + z)

#I[3]::  Lev[%,1]

           2
#O[3]:   {x ,x + z}

#I[4]::  Lev[@2,2]

#O[4]:   {x,2,x,z}

#I[5]::  Lenlev[@2,2]

#O[5]:   4
```

XLevi

Levi-Civita tensor generation

totally antisymmetric tensor – basis tensor – epsilon tensor

S.Wolfram

Jul 1981

Levi[*n*]

yields the Levi-Civita total antisymmetric epsilon tensor in *n* dimensions.

```
Levi[$n_=Natp[$n]] :: Ar[Ar[$n,$n],Sig]
```

XList0

Basic list manipulations

head – LISP CAR – beginning – prepend – catentate
append – tail – take – select – part – insert – remove
replicate – repeat

S.Wolfram
Jul 1981

First[*list*]
yields the first entry in *list*.

```
First⊣Tier
First[$list_⊸Listp[$list]] :: Elem[$list,{1}]
```

Prep[*elem*, *list*]
prepends elem to the beginning of list.

```
Prep⊣Tier
Prep[$elem,$list_⊸Listp[$list]] :: Cat[{$elem},$list]
```

• Cat

App[*elem*, *list*]
appends *elem* to the end of *list*.

```
App⊣Tier
App[$elem,$list_⊸Listp[$list]] :: Cat[$list,{$elem}]
```

Tk[*n*, *list*]
takes the first *n* or last -*n* entries in *list*.

```
Tk⊣Tier
Tk[$n_⊸Natp[$n],$list_⊸Listp[$list]] :: Ar[$n,$list]
Tk[$n_⊸Natp[-$n],$list_⊸Listp[$list]] :: \
                Cat[Ar[{Len[$list]+{$n+1,0}},$list]]
```

Ins[*elem*, *list*, *i*]
inserts the entry *elem* into *list* at position *i*.

```
Ins⊣Tier
Ins[$elem,$list_⊸Contp[$list],$n_⊸Natp[$n+1]] :: \
        Cat[Ar[$n-1,$list],{$elem},Ar[{{$n,Len[$list]}},$list]]
```

Rm[*list*, *n*]
removes the *n* th entry from *list*.

```
Rm⊣Tier
Rm[$list_⊸Listp[$list],$n_⊸Natp[$n]] :: Cat[Ar[$n-1,$list], \
        Ar[{{$n+1,Len[$list]}},$list]]
Rm[$list_⊸Contp[$list],$n_⊸Natp[$n]] :: \
        Cat[Ar[Len[$list],$list,$X1~⊸$n]]
```

Lrpt[*list*, *n*]
yields a list consisting of *n* repetitions of *list*.

```
Lrpt_:Tier
Lrpt[$list_>Contp[$list],$n_>Natp[$n]] :: \
        Cat[Repl[$list,$n]]
```

```
#I[1]::  <XList0
#I[2]::  t:{a,b,c,d,e,f,g}
#O[2]:   {a,b,c,d,e,f,g}
#I[3]::  First[t]
#O[3]:   a
#I[4]::  Last[t]
#O[4]:   g
#I[5]::  Prep[1,t]
#O[5]:   {1,a,b,c,d,e,f,g}
#I[6]::  App[1,t]
#O[6]:   {a,b,c,d,e,f,g,1}
#I[7]::  Tk[4,t]
#O[7]:   {a,b,c,d}
#I[8]::  Tk[-4,t]
#O[8]:   {d,e,f,g}
#I[9]::  Ins[1,t,4]
#O[9]:   {a,b,c,1,d,e,f,g}
#I[10]:: Rm[t,4]
#O[10]:  {a,b,c,e,f,g}
#I[11]:: Lrpt[{a,b,c},4]
#O[11]:  {a,b,c,a,b,c,a,b,c,a,b,c}
```

XList1

Sublist manipulation

lists – list substitution – sublist positions – sequence positions

S.Wolfram
Jul 1981

LPos[*sub*, *list*]

find positions at which the sublist *sub* appears in *list*.

```
LPos[$sub_:Contp[$sub],$list_:Contp[$list]] :: \
        (Lcl[%l,%2]; %l:{}; Do[%1,Len[$list]-Len[$sub]+1, \
        For[%2:0, P[$sub[%2+1]=$list[%1+%2]] & %2<Len[$sub], \
        Inc[%2],]; If[%2=Len[$sub],%l:Cat[%l,{%1}]]]; %l)
LPos[$sub_:(Contp[$sub] & Len[$sub]=1),$list_:Contp[$list]] :: \
        Flat[Pos[$sub[1],$list,2]]
```

LSub[*list2*, *list1*, *list*]

substitutes *list2* for all occurrences of the sublist *list1* in *list*.

```
LSub[$list2,$list1,$list] :: (Lcl[%f]; %f_:Flat; \
S[S[Ap[%f,$list],Ap[%f,$list1]->Ap[%f,$list2],1,Inf],%f->List])
```

LS[*list*, *rep1*, *rep2*, . . .]

applies successively the replacements *repi* specified for sublists in *list*. (The *repi* may contain patterns.)

```
LS[$list,$$reps] :: (Lcl[%l]; %l:$list; \
Map[%l:LSub[$1[2],$1[1],%l],List[$$reps],1,  \
                  ($2[0]='Rep) | ($2[0]='Repd)]; %l)
```

LIn[*list1*, *list*]

yields 1 if *list1* is a sublist of *list*, and 0 otherwise.

```
LIn[$list1,$list] :: (Lcl[%f]; %f_:Flat; ~P[~Match[Ap[%f, \
                  Cat[{$$%1},$list1,{$$%2}]],Ap[%f,$list]]])
```

```
#I[1]::  <XList1

#I[2]::  t:{a,b,c,a,b,d,e,a,c}

#O[2]:   {a,b,c,a,b,d,e,a,c}

#I[3]::  LPos[{a,b},t]

#O[3]:   {1,4}

#I[4]::  LSub[{g,h,i},{a,b},t]

#O[4]:   {g,h,i,c,g,h,i,d,e,a,c}

#I[5]::  LS[t,{a,b}->{g,h,i},{e}->{r,s}]

#O[5]:   {g,h,i,c,g,h,i,d,r,s,a,c}

#I[6]::  LIn[{a,b},t]

#O[6]:   1

#I[7]::  LIn[{a,d},t]
```

#D[7]: 8

XLogic

Elementary propositional calculus

C.Feynman
Aug 1981

Note p=q represents the logical biconditional "if and only if p then q".

Absorption laws

```
$p | ($p & $$q) : $p
$p & ($p | $$q) : $p
$p || ($p | $$q) :: ~$p & Or[$$q]
$p || ($p & $$q) : $p & ~$$q
$p & ($p || $q) : $p & ~$q
$p | ($p || $q) : $p | $q
$p | (~$p & $$q) :: $p | And[$$q]
$p & (~$p | $$q) : $p & Or[$$q]
$p || (~$p | $$q) : 1
$p || (~$p & $$q) : 1
$p & (~$p || $q) : $p & $q
$p | (~$p || $q) : $p | ~$q
```

Commutative and associative laws built in.

Distributive laws

```
SLogic[1]:       $p | ($$q & $r) --> ($p | And[$$q]) & ($p | $r)
SLogic[2]:       $p & ($$q | $r) --> ($p & Or[$$q]) | ($p & $r)
```

Idempotence

```
$p & $p : $p
$p | $p : $p
$p || $p : 0
```

Identity laws built in.

Complement laws

```
~~$p : $p
$p | ~$p : 1
$p & ~$p : 0
$p || ~$p : 1
$p => ~$p : ~$p
```

DeMorgan's laws

```
~($$p | $q) : (~$$p) & (~$q)
~($$p & $q) : (~$$p) | (~$q)
~($p || $q) : ($p=$q)
```

Reflexive law

```
$p => $p : 1
```

Antisymmetric law

```
($p => $q) & ($q => $p) : $p=$q
```

Transitive law

```
($p => $q) & ($q => $r) : ($p=>$r) & ($p=>$q) & ($q=>$r)
```

Consensus

```
SLogic[3] :     ($p|$$q)&(~$p|$$r) --> ($p|$$q)&(~$p|$$r)&($$q|$$r)
SLogic[4] :     ($p&$$q)|(~$p&$$r) --> ($p&$$q)|(~$p&$$r)|($$q&$$r)
```

Complete sum and product

```
Cdisj[$r] :     S[S[$r,SLogic[1],Inf],SLogic[2],Inf]
Cconj[$r] :     S[S[$r,SLogic[2],Inf],SLogic[1],Inf]
```

Alternate expressions

```
SLogic[5] :     $p=>$q --> $q | ~$p
SLogic[6] :     $p||$q --> ($p | $q) & (~$p | ~$q)
SLogic[7] :     $p||$q --> $p & ~$q | ~$p & $q
```

Misfeatures: Loops infinitely on input.

XLogic2

Elementary logic with quantifiers

C.Feynman
Aug 1981

Quant[*q1*, *q2*, . . ., *stat*]
represents the statement *stat* subject to the quantifiers *q1*, *q2*, ...

All[*x*, (*l*: (*all*))]
represents the quantifier "for all x in the list l". The second argument is optional; if it is not supplied the quantifier "for all x" is represented. In this case, automatic simplification is not possible

Some[*x*, (*l*: (*all*))]
represents the quantifier "for some x in the list l". The second argument may be omitted; in this case the quantifier "there exists an x such that" is represented. In this case, automatic simplification is not possible.

```
Quant[$$q,Quant[$$r]] : Quant[$$q,$$r]
Quant[$s] : $s
```

DeMorgan's law

```
~Quant[All[$x],$$q,$s] : Quant[Some[$x],~Quant[$$q,$s]]
~Quant[Some[$x],$$q,$s] : Quant[All[$x],~Quant[$$q,$s]]
```

Simplification of restricted-range quantifiers

Misfeatures: THESE HAVE NOT BEEN FULLY TESTED, DUE TO A BUG IN THE SIMPLIFIER WHILE THEY WERE BEING WRITTEN. If you use them and something goes wrong, it might be my fault.

```
Quant[$$x,All[$y,$z → (Listp[$z] & ~ P[$z = {}])],$prop] :: Quant[$$x,\
            Ap[And,Map[S[$prop,$y -> $q],$z]]]
Quant[$$x,Some[$y,$z → (Listp[$z] & ~ P[$z = {}])],$prop] :: Quant[$$x,\
            Ap[Or,Map[S[$prop,$y -> $q],$z]]]
```

XLogicPr

Logical truth table generation

Boolean algebra – state table – first-order logic

S.Wolfram
Jul 1981

PrTF[*expr*]

prints a table giving the values of the logical expression *expr* for all possible truth values of symbols appearing in it.

```
PrTF[$expr] :: (Lcl[%p]; Ap[Pr,Cat[%p:Cont[$expr],{$expr}]]; \
                    Ar[Ar[Len[%p],`{{0,1}}], \
             Ap[Pr,S[Cat[%p,{$expr}],Ldist[%p->List[$$1]]]]]])
```

```
#I[1]::  <XLogicPr

#I[2]::  PrTF[(p&r)=>(q|~p)]

p        q        r        p & r => q | ~ p
0        0        0        1
0        0        1        1
0        1        0        1
0        1        1        1
1        0        0        1
1        0        1        0
1        1        0        1
1        1        1        1

#O[2]:   {[0]: {[0]: {[0]: 1, [1]: 1}, [1]: {[0]: 1, [1]: 1}},

                [1]: {[0]: {[0]: 1, [1]: 0}, [1]: {[0]: 1, [1]: 1}}}
```

XLor

Lorentz vectors

relativistic mechanics – four vectors – Minkowski space

S.Wolfram
Jul 1981

Ldot[*list1*, *list2*]
forms the contraction of two Lorentz vectors with a metric of signature ---+

```
Ldot[$list1,$list2] :: {-1,-1,-1,1}.($list1 $list2)
```

XMKS

MKS/SI units

physical quantities – dimensional analysis – physical constants
units conversion – CGS units

S.Wolfram
Jul 1981

```
<XDim
```

Units for fundamental dimensions

```
SSI[0,1] :      length -> metre
SSI[0,2] :      mass -> kilogram
SSI[0,3] :      time -> second
SSI[0,4] :      current -> ampere
SSI[0,5] :      temperature -> kelvin
SSI[0,6] :      intensity -> candela
SSI[0,7] :      amount -> mole
```

fundamental units

```
SSI[1,1] :      metre -> length
SSI[1,2] :      kilogram -> mass
SSI[1,3] :      second -> time
SSI[1,4] :      ampere -> current
SSI[1,5] :      kelvin -> temperature
SSI[1,6] :      candela -> intensity
SSI[1,7] :      mole -> amount
```

derived units

```
SSI[2,1] :      hertz -> frequency
SSI[2,2] :      newton -> force
SSI[2,3] :      pascal -> pressure
SSI[2,4] :      joule -> energy
SSI[2,5] :      watt -> power
SSI[2,6] :      coulomb -> charge
SSI[2,7] :      volt -> voltage
SSI[2,8] :      ohm -> resistance
SSI[2,9] :      siemens -> conductance
```

```
SSI[2,10] :     mho -> conductance
SSI[2,11] :     farad -> capacitance
SSI[2,12] :     weber -> flux
SSI[2,13] :     tesla -> weber/area
SSI[2,14] :     henry -> inductance
SSI[2,15] :     gray -> dose
SSI[2,16] :     becquerel -> activity
SSI[2,17] :     lumen -> intensity steradian
SSI[2,18] :     lux -> illuminance
```

multipliers

```
SSI[2,1] :      exa -> 1*^18
SSI[2,2] :      peta -> 1*^15
SSI[2,3] :      tera -> 1*^12
SSI[2,4] :      giga -> 1*^9
SSI[2,5] :      mega -> 1*^6
SSI[2,6] :      kilo -> 1*^3
SSI[2,7] :      hecto -> 1*^2
SSI[2,8] :      deca -> 10
SSI[2,9] :      deci -> 0.1
SSI[2,10] :     centi -> 1*^-2
SSI[2,11] :     milli -> 1*^-3
SSI[2,12] :     micro -> 1*^-6
SSI[2,13] :     nano -> 1*^-9
SSI[2,14] :     pico -> 1*^-12
SSI[2,15] :     femto -> 1*^-15
SSI[2,16] :     atto -> 1*^-18
```

XMOrd

Multiplicative orders

coding theory – arithmetic over finite fields
S.Wolfram
Oct 1982

<XLCM

Prerequisites: XLCM

MOrd[*k*,*n*]

yields the multiplicative order of *k* modulo *n*. (Smallest *j* such that *k*^*j* = 1 modulo *n*). *

```
MOrd ⌴: Tier
MOrd[$k,$n_⌴(Len[Nfac[$n]]=1)] :: (Lcl[i,t]; For[i:1; t:B[$k], \
        N[Mod[t,$n]]~=1, Inc[i], t:t B[$k]]; Ret[i])
MOrd[$k,$n_⌴(Gcd[$k,$n] ~= 1)] : 0
MOrd[$k,$n] :: Ap[LCM, Map[MOrd[$k,$1[1]~$1[2]], Nfac[$n]]]
MOrd[$k,1] : 0
```

GMOrd[*k*,{*n1*,*n2*,...}]

generalized multidimensional multiplicative order of *k*.

```
GMOrd ⌴: Tier
GMOrd[$k,$list_⌴(Listp[$list] & Ap[And,Ldist[Gcd[$k,$list] = 1]])] :\
    (Lcl[t,i]; For[i:1; t:B[$k], Ap[And,N[Ldist[Mod[t,$list]~=1]]], \
                  Inc[i], t:t B[$k]]; Ret[i])
```

Misfeatures: Test in GMOrd should Ldist before failing Eq test

XMSet

Automatic memo definition

dynamic programming – look-up

T.Shaw
Jul 1981

expr ::: ***val***

assigns the value *val* to the projection *expr* in such a way as to store explicitly each form of *expr* requested.

```
MSet_⊐Tier
_MSet[Smp]:0
_MSet[Pr][$expr,$val] :: Sx[" ::: ",{$expr,$v},4]
Sxset[":::",MSet,4]
$expr ::: $val :: $expr :: $expr : $val
```

```
#I[1]::  <XMSet

#I[2]::  f[0]:f[1]:1

#O[2]:   1

#I[3]::  f[$x]:::$x f[$x-2]

#O[3]:   ' f[$x] : $x f[$x - 2]

#I[4]::  f

#O[4]:   {[0]: 1, [1]: 1, [$x]:: f[$x] : $x f[$x - 2]}

#I[5]::  f[6]

#O[5]:   48

#I[6]::  f

#O[6]:   {[6]: 48, [4]: 8, [2]: 2, [0]: 1, [1]: 1,
                           [$x]:: f[$x] : $x f[$x-2]}

#I[7]::  f[11]

#O[7]:   10395

#I[8]::  f

#O[8]:   {[11]: 10395, [9]: 945, [7]: 105, [5]: 15, [3]: 3, [6]: 48, [4]: 8,
              [2]: 2, [0]: 1, [1]: 1, [$x]:: f[$x] : $x f[$x - 2]}
```

XMask

List element masking

select – choose – extract – remove

S.Wolfram
Aug 1982

Mask[*mask*, *list*]

masks out those elements of *list* for which the corresponding elements of *mask* are not determined to be true.

```
Mask[[$mask_=Contp[$mask],$list_=Contp[$list]]] :: \
                        Cat[Ar[Len[$list],$list,$mask[$%1]]]
```

```
#I[1]::   <XMask

#I[2]::   Ar[10]

#O[2]:    {1,2,3,4,5,6,7,8,9,10}

#I[3]::   Map[Evenp,%]

#O[3]:    {0,1,0,1,0,1,0,1,0,1}

#I[4]::   Mask[@3,@2]

#O[4]:    {2,4,6,8,10}
```

XMat1

Matrix input and generation

enter matrix – interactive matrix input – diagonal matrices

S.Wolfram
Jul 1981

MRd[*n*, *m*]

reads in turn each entry of an $n * m$ matrix.

```
MRd[$n,$m] :: Ar[{$n,$m},Pr[{$1,$2},": "];Rd[]]
```

Diag[*list*]

yields a matrix with diagonal *list* and all other entries zero.

```
Diag[$list_=Contp[$list]] :: Ar[Ar[2,`Len[$list]], \
                             If[$X1=$X2,$list[$X1],0]]
```

```
#I[1]::   <XMat1

#I[2]::   MRd[2,2]

{1,1}     : a1
{1,2}     : a1
{2,1}     : b1
{2,2}     : b2

#O[2]:    {{a1,a2},{b1,b2}}

#I[3]::   Diag[{3,2,a}]

#O[3]:    {{3,0,0},{0,2,0},{0,0,a}}
```

XMat2

Structural matrix operations

diagonals – minors

S.Wolfram
Jul 1981

```
<XMat3
```

LDiag[*mat*]
yields a list of the elements on the leading diagonal of *mat*.

```
LDiag[$mat_=Matp[$mat]] :: Ar[Min[Dim[$mat][1],Dim[$mat][2]], \
                                   $mat[$%1,$%1]]
```

TDiag[*mat*]
yields a list of the elements on the trailing diagonal of *mat*.

```
TDiag[$mat_=Matp[$mat]] :: Ar[Min[Dim[$mat][1],Dim[$mat][2]], \
                    $mat[$%1,Dim[$mat][2]-$%1+1]]
```

Minor[*expr*,*i*,*j*]
forms the *ij* th minor of the matrix *expr*.

```
_Minor[Init] :: <XList8
Minor[$mat_=Matp[$mat],$i_=Natp[$i],$j_=Natp[$j]] :: \
      Map[Rm[$%1,$j],Rm[$mat,$i]]
```

```
#I[1]::  <XMat2

#I[2]::  m:Ar[{3,3},f]

#O[2]:   {{f[1,1],f[1,2],f[1,3]},{f[2,1],f[2,2],f[2,3]},{f[3,1],f[3,2],f[3,3]}}

#I[3]::  LDiag[m]

#O[3]:   {f[1,1],f[2,2],f[3,3]}

#I[4]::  TDiag[m]

#O[4]:   {f[1,3],f[2,2],f[3,1]}

#I[5]::  Minor[m,2,3]

#O[5]:   {{f[1,1],f[1,2]},{f[3,1],f[3,2]}}
```

XMat3

Matrix character tests

square matrix – symmetric matrix – antisymmetric matrix
diagonal matrix – projection matrix – matrix classes
matrix types

S.Wolfram
Jul 1981

Matp[*expr*]
yields 1 if *expr* represents a matrix (rank 2 tensor).

```
Matp[$expr] :: Fullp[$expr,2]
```

Sqmatp[*expr*]
yields 1 if *expr* represents a square matrix.

```
Sqmatp[$expr] :: Matp[$expr] & P[Dim[$expr][1] = Dim[$expr][2]]
```

Symp[*expr*]
yields 1 if *expr* represents a symmetric matrix.

```
Symp[$expr] :: Matp[$expr] & P[$expr = Trans[$expr]]
```

Asymp[*expr*]
yields 1 if *expr* represents an antisymmetric matrix.

```
Asymp[$expr] :: Matp[$expr] & P[$expr = -Trans[$expr]]
```

Diagp[*expr*]
yields 1 if *expr* represents a diagonal matrix.

```
Diagp[$expr] :: Matp[$expr] & \
                 P[$expr-Diag[LDiag[$expr]]=Ar[Dim[$expr,2],0]]
```

Projecp[*expr*]
yields 1 if *expr* represents a projection matrix.

```
Projecp[$expr] :: Matp[$expr] & Ex[$expr.$expr=$expr]
```

XMat4

Algebraic matrix operations

matrix power – repeated transformation – hermitean adjoint
matrix adjoint – characteristic polynomial – eigenvalue equation
generalized trace

S.Wolfram

Jul 1981

```
<XMat3
```

Mpow[*mat*, *n*]
yields the *n* th power of the matrix *mat* (for integer *n*).

```
Mpow[$mat_=Sqmatp[$mat],0] :: Ar[Ar[2,Len[$mat]]]
Mpow[$mat_=Sqmatp[$mat],$n_=Natp[$n]] :: Dot[Repl[$mat,$n]]
Mpow[$mat_=Sqmatp[$mat],$n_=Natp[-$n]] :: Dot[Repl[Minv[$mat],-$n]]
```

Adj[*expr*]
forms the Hermitean adjoint of *expr*.

```
Adj[$expr] :: Conj[Trans[$expr]]
```

Cof[*expr*, *i*, *j*]
forms the *ij* th cofactor of the matrix *expr*.

```
_Cof[Init] :: <XMat2
Cof[$expr_=Matp[$expr],$i_=Natp[$i],$j_=Natp[$j]] :: \
        Det[Minor[$expr,$i,$j]]
```

Charpol[*expr*, *var*]
forms the characteristic polynomial for the matrix *expr* with respect to *var*.

```
Charpol[$expr_=Matp[$expr],$var] :: Det[$expr - $var Ar[Dim[$expr]]]
```

Gentr[*expr*, *k*]
forms the *k* th order trace of the matrix *expr*.

```
Gentr[$expr_=Matp[$expr],$k_=Natp[$k]] :: (Lcl[%lam]; \
                   Coef[%lam^$k,Ex[Charpol[$expr,%lam]]])
```

```
#I[1]::  <XMat4

#I[2]::  m:{{a1,a2},{b1,b2}}

#O[2]:   {{a1,a2},{b1,b2}}

#I[3]::  Cof[m,1,1]

#O[3]:   b2

#I[4]::  Charpol[m,x]

#O[4]:   -a2 b1 + (a1 - x) (b2 - x)

#I[5]::  Ex[%]

                                       2
#O[5]:   a1 b2 - a1 x - a2 b1 - b2 x + x
```

```
#I[6]::  Gentr[m,2]
#O[6]:   1
#I[7]::  Gentr[m,1]
#O[7]:   -a1 - b2
```

• XPer

XMathPR

Special printing forms for mathematical functions

S.Wolfram
Jul 1981

```
_Exp[Pr][$x] :: Fmt[{{0,0},{1,1}},E,$x]

_Log[Pr][[$x,$y]] :: Fmt[{{0,0},{1,-1},{2,0},{3,0},{4,0}},Log,$y,"[",$x,"]"]

_Poc[Pr][[$x,$n]] :: Fmt[{{0,0},{1,0},{2,0},{3,-1}},"(",$x,")",$n]

_Comb[Pr][[$n,$m]] :: Fmt[{{0,0},{0,-1},{0,1},{0,2},{1,0},{2,0},{2,1},\
                       {3,0},{4,0},{4,-1},{4,1},{4,2}},"|","\","|","/", \
                       " ",$m,$n," ","|","/","|","\"]
```

Not optimal

```
_Expi[Pr][[$n,$z]] :: Fmt[{{0,0},{1,-1},{2,0},{3,0},{4,0}}, \
                       E,$n,"[",$z,"]"]

_Psi[Pr][[$z,$n]] :: Fmt[{{0,0},{1,1},{2,1},{3,1},{4,0},{5,0},{6,0}}, \
                       Psi,"(",$n,")","[",$z,"]"]

_Li[Pr][[$z]] :: Fmt[{{0,0},{1,-1},{2,0},{3,0},{4,0}},Li,2,"[",$z,"]"]
_Li[Pr][[$n,$z]] :: Fmt[{{0,0},{1,-1},{2,0},{3,0},{4,0}},Li,$n,"[",$z,"]"]

_Ber[Pr][[$n]] :: Fmt[{{0,0},{1,-1}},B,$n]

_Eul[Pr][[$n]] :: Fmt[{{0,0},{1,-1}},E,$n]
```

XMaxind

Find maximal index

S.Wolfram
Jul 1981

Maxind[*list*]
yields the maximal index in *list*.

```
Maxind[$list_⇒Contp[$list]] :: Len[$list]
Maxind[$list] :: Ap[Max,Ar[Len[$list],Ind[$list,$1]]]
```

```
#I[1]::  <XMaxind

#I[2]::  t:{[4]:a,[5]:b,[2]:c,[1]:d}

#O[2]:   {[4]: a, [5]: b, [2]: c, [1]: d}

#I[3]::  Maxind[t]

#O[3]:   5
```

XMorse

Morse code translator

S.Wolfram
Jul 1981

Deciphers Morse code (international version).

```
" ":=""
".-":="a"; "-... ":="b"; "-.-. ":="c"; "-.. ":="d"; ". ":="e"; "..-. ":="f";
"--. ":="g"; ".... ":="h"; ".. ":="i"; ".---":="j"; "-.-":="k"; ".-.. ":="l";
"--":="m"; "-. ":="n"; "---":="o"; ".--. ":="p"; "--.-":="q"; ".-. ":="r";
"... ":="s"; "-":="t"; "..-":="u"; "...-":="v"; ".--":="w"; "-..-":="x";
"-.--":="y"; "--.. ":="z";
```

```
#I[1]::   <XMorse

#I[2]::   ... -- .--.

#O[2]:    smp
```

XN

Number type conversion

arbitrary precision – fixed precision – arbitrary accuracy
fixed accuracy – type casting – type coercion – type conversion

S.Wolfram
Aug 1982

FN [*number*]

converts *number* from arbitrary to fixed precision form.

```
FN[F[$1,$2,$3,$$,$e,$a]] :: ($1 + ($2 + $3/10*^3)/10*^3)*^($e-4)
FN[$n_=Numbp[$n]] : $n
FN[$x] : $x
```

NF [*number*]

converts *number* to 12-digit precision form.

```
NF[$n_=Numbp[$n]] :: $n F[1]
NF[$n] : $n
```

```
#I[1]::   <XN

#I[2]::   N[Pi]

#O[2]:    3.14159

#I[3]::   NF[%]

#O[3]:*   (3.14159265359)

#I[4]::   FN[%]

#O[4]:    3.14159
```

Misfeatures: These coercions should be done automatically when they are required, making this file redundant.

XNAT

Natural units

S.Wolfram
Jul 1981

Basic constants :

Qc velocity of light in vacuo
Qhbar Planck's constant
Qk Boltzmann's constant
Qalpha Dimensionless fine structure constant
QG Newton's constant of gravitation
QN Avogadro's number.

```
<XDim
```

1. Conversion from system with Qhbar : Qc : Qk : 1

```
SNAT[1,1] :     length -> Qhbar/(Qc mass)

SNAT[1,2] :     time -> Qhbar/(Qc^2 mass)

SNAT[1,3] :     temperature -> (Qk mass Qc^2)^-1

SNAT[1,4] :     current -> mass (Qhbar^-1 Qc^-5 alpha)^(1/2)

SNAT[1,5] :     amount -> QN


#I[1]::   <XNAT

#I[2]::   current energy/length

          current energy
#O[2]:    --------------
              length

#I[3]::   Si[%,SDim]

          current length mass
#O[3]:    -------------------
                     2
                 time

#I[4]::   Si[%,SNAT[1]]

            11/2       1/2       3
          Qc       alpha      mass
#O[4]:    ----------------------
                    3/2
               Qhbar
```

XNMap

Multi-element Map

S.Wolfram
Jul 1981

NMap[*f*, *list*, *n*]
applies *f* to groups of *n* elements in *list*.

```
NMap[Smp]:{0,Inf,Inf}
_NMap[Init] :: <XUnFlat
NMap[$f,$list_=Contp[$list],$n_=Natp[$n]] :: \
        Map[Ap[$f,$%1],UnFlat[$list,$n]]
```

```
#I[1]::  <XNMap

#I[2]::  t:Ar[10]

#O[2]:   {1,2,3,4,5,6,7,8,9,10}

#I[3]::  NMap[f,t,2]

#O[3]:   {f[1,2],f[3,4],f[5,6],f[7,8],f[9,10]}

#I[4]::  NMap[f,t,3]

#O[4]:   {f[1,2,3],f[4,5,6],f[7,8,9],f[10]}
```

XNSol

Numerical solution of equations by Newton's method

S.Wolfram
Jul 1981

NSol[*eqn*, *x*, *x0*, *acc*]
attempts to find a solution for *x* in the equation *eqn* using Newton's method starting at the point *x* = *x0*, with accuracy *acc*.

```
NSol_:Tier
NSol[$a=$b,$x_=Symbp[$x],$x0_=Numbp[N[$x0]],$acc_=Numbp[$acc]] ::  \
        (Lcl[%f,%df,%x,%x0];%x:$x0;%f:$a-$b;%df:D[%f,$x]; \
        Loop[,%x:N[(%x0:%x)-S[%f/%df,$x->%x]],Abs[%x-%x0]>$acc];%x)
```

```
#I[1]::  <XNSol

#I[2]::  NSol[Sin[x]=x,x,1,1/1000]

#O[2]:   .001476886

#I[3]::  N[Sin[%]]

#O[3]:   .001476886
```

XNorm

Vector norm

S.Wolfram
Jul 1981

Norm[*list*]

yields the norm of the vector represented by *list*.

```
Norm[$list] :: Sqrt[Ap[Plus,$list^2]]
```

XOrbit

Planetary orbits

Solar system – celestial mechanics – astronomical data
orbital elements – Kepler's laws

S.Wolfram
Jul 1981

[Handbook of British Astronomical Association (1982); Astronomical Ephemeris]

Fundamental parameters:
- Epoch : time in Julian days (T)
- LEpoch : longitude of planet at epoch (L)
- LPeri : longitude of perihelion
- LNode : longitude of ascending node
- Incl : inclination to ecliptic (i)
- Ecc : eccentricity (e)
- MDist : mean distance (a)
- Per : sidereal period in days (P)

Heliocentric quantities:

MAnom[*t*]
mean anomaly at time *t* from epoch

```
MAnom[$t] : 2Pi $t/Per + LEpoch - LPeri
```

EccAnom[*t*]
eccentric anomaly at time *t* from epoch

XPR

Special output forms

S.Wolfram
Jul 1981

PSig

prints as

```
---
\
/
---
```

```
_PSig[Pr] : Fmt[{{3,2},{2,2},{1,2},{1,1},{1,0},{1,-1},{2,-1},{3,-1}},\
      "_","_","_","\","/","_","_","_"]
```

PPi

prints as

```
 ___
 | |
 | |
```

```
_PPi[Pr] : Fmt[{{1,-1},{1,0},{1,1},{2,1},{3,1},{3,0},{3,-1}},\
      "|","|","_","_","_","|","|"]
```

Plam

prints as

```
\
/\
```

```
_Plam[Pr] : Fmt[{{1,-1},{1,0}},"/\","\"]
```

PLam

prints as

```
 /\
/  \
```

```
_PLam[Pr] : Fmt[{{1,-1},{2,0},{3,-1}},"/","/\","\"]
```

PDelta

prints as

```
 /\
/__\
```

```
_PDelta[Pr] : Fmt[{{1,-1},{2,-1},{3,-1},{2,0}},"/","__","\","/\"]
```

PGamma

prints as

```
 ____
 |
 |
```

```
_PGamma[Pr] : Fmt[{{1,-1},{1,0},{1,1}},"|","|","____"]
```

PXi

prints as

```
----
 --
----
```

```
_PXi[Pr] : Fmt[{{1,-1},{2,-1},{3,-1},{2,0},{1,1},{2,1},{3,1}},"-","--","-", \
 "--","-","--","-"]
```

PInt

prints as

```
/
|
/
```

```
_PInt[Pr] : Fmt[{{1,1},{1,0},{1,-1}},"/","|","/"]
```

PSqrt

prints as

```
  --
 /
\/
```

```
_PSqrt[Pr] : Fmt[{{1,-1},{2,0},{3,1}},"\/","/","--"]
```

```
#I[1]::  <XPR

#I[2]::  PPi^2

          ___2
#O[2]:*  |  |
         |  |

#I[3]::  _f[Pr]:PSig

         ---
         \
#O[3]:*  /
         ---

#I[4]::  f+1/f

         ---
         \        1
#O[4]:*  /    +  ---
         ---     ---
                 \
                 /
                 ---
```

XPad

List padding

left justify – right justify – trailing zeroes – leading zeroes

S.Wolfram
Aug 1982

LPad[*list, elem, len*]
pads out the list *list* on the left with element *elem* to length *len* if necessary.

```
LPad[[$list,$elem,$len]] :: If[Len[$list] >= $len,$list, \
                                  Cat[Ar[$len-Len[$list], `$elem],$list]]
```

RPad[*list, elem, len*]
pads out the list *list* on the right with element *elem* to length *len* if necessary.

```
RPad[[$list,$elem,$len]] :: If[Len[$list] >= $len,$list, \
                                  Cat[$list,Ar[$len-Len[$list], `$elem]]]
```

XPauli

Pauli sigma matrices

S.Wolfram
Jul 1981

Sigma[*i*]

gives a representation of the the *i* th Pauli sigma matrix.

```
Sigma[0] : {{1,0},{0,1}}
Sigma[1] : {{0,1},{1,0}}
Sigma[2] : {{0,-I},{I,0}}
Sigma[3] : {{1,0},{0,-1}}
```

XPause

Pause

S.Wolfram
Jul 1981

Pause[*n*]
pause for *n* seconds.

```
Pause[$n_=Numbp[$n]] :: (Lcl[c0];c0:Clock[][1]; \
                         Loop[Clock[][1]-c0<$n,1])
```

XPeel

Sublist peeling

S.Wolfram
Jul 1981

Peel[{*expr1*, *expr2*, . . .}]
represents the sequence *expr1*,*expr2*,... in a list. ("Peels" away lists).

```
Peel[$list_=Listp[$list]] :: Ap[Np,$list]
```

```
#I[1]::  <XPeel
#I[2]::  {Peel[{a,b}],{c,d}}
#O[2]:   {a,b,{c,d}}
```

XPer

Matrix permanents

determinants

L.Yaffe
August 1982

Per[*mat*]

yields the permanent of the matrix *mat*

[E.R.Caianiello: "Combinatorics and Renormalization in Quantum Field Theory", p.29]

```
<XMat2
Per[$mat_=Sqmatp[$mat]] :: Sum[$mat[1,i] Per[Minor[$mat,1,i]], \
                        {i,1,Dim[$mat][1]}]
Per[{{$x}}] :: $x
```

```
#I[1]::  <XPer

#I[2]::  m:Ar[{3,3},f]

#O[2]:   {{f[1,1],f[1,2],f[1,3]},{f[2,1],f[2,2],f[2,3]},{f[3,1],f[3,2],f[3,3]}}

#I[3]::  Per[m]

#O[3]:   f[1,1] (f[2,2] f[3,3] + f[2,3] f[3,2])

                  + f[1,2] (f[2,1] f[3,3] + f[2,3] f[3,1])

                  + f[1,3] (f[2,1] f[3,2] + f[2,2] f[3,1])

#I[4]::  Det[m]

#O[4]:   f[1,1] (f[2,2] f[3,3] - f[2,3] f[3,2])

                  - f[1,2] (f[2,1] f[3,3] - f[2,3] f[3,1])

                  + f[1,3] (f[2,1] f[3,2] - f[2,2] f[3,1])
```

XPerm0

Permutations

S.Wolfram
Jul 1981

A permutation is represented by a contiguous list of sequential integers in any order.

Permp[*expr*]
: yields 1 if *expr* represents a permutation.

```
Permp[$expr] :: P[Sort[$expr]=Ar[Len[$expr]]]
Permp[$expr_=~Contp[$expr]] : 0
```

Fiper[*list1*, *list2*]
: finds if possible a permutation which places the elements of *list2* in the order of *list1*.

```
Fiper[$list1_=Contp[$list1],      \
              $list2_=(Contp[$list2]&Len[$list1]=Len[$list2])] :: \
        Flat[Ar[Len[$list1],Pos[$list2[$%1],$list1]]]
```

Apper[*perm*, *list*]
: applies the permutation *perm* to *list*.

```
Apper[$perm_=Permp[$perm],$list_=Contp[$list]] :: \
        Ar[Len[$list],$list[$perm[$%1]]]
```

```
#I[1]::  <XPerm0
#I[2]::  Permp[{1,3,2,4}]
#O[2]:   1
#I[3]::  Permp[{1,3,1,4}]
#O[3]:   0
#I[4]::  t1:{a,c,d,b}
#O[4]:   {a,c,d,b}
#I[5]::  Fiper[{a,b,c,d},t1]
#O[5]:   {1,3,4,2}
#I[6]::  Apper[%,{a,b,c,d}]
#O[6]:   {a,c,d,b}
```

XPerm1

Elementary operations on permuations

S.Wolfram
Jul 1981

```
<XPerm0
```

Pcomp[*perm1, perm2*]
forms the compostion (product) of the two permutations *perm1* and *perm2*.

```
Pcomp[$p1_=Permp[$p1],$p2_=Permp[$p2]] :: Ar[Len[$p1],$p2[$p1[$%1]]]
```

Ppow[*perm, n*]
forms the *n* th power of the permutation *perm*.

```
Ppow[$p_=Permp[$p],0] :: Ar[Len[$p]]
Ppow[$p_=Permp[$p],$n_=Natp[$n]] :: \
                    S[$p,$%1-->Ar[Len[$p],$%1[$p[$%2]]],$n-1]
Ppow[$p_=Permp[$p],$n_=Natp[-$n]] :: Ppow[Pinv[$p],-$n]
```

Pinv[*perm*]
yields the inverse of the permutation *perm*.

```
Pinv[$p_=Permp[$p]] :: Ar[Len[$p],Pos[$%1,$p][1,1]]
```

```
#I[1]::  <XPerm1
#I[2]::  p1:{1,5,4,2,3}
#O[2]:   {1,5,4,2,3}
#I[3]::  p2:{5,1,3,2,4}
#O[3]:   {5,1,3,2,4}
#I[4]::  Pcomp[p1,p2]
#O[4]:   {5,4,2,1,3}
#I[5]::  Pcomp[p2,p1]
#O[5]:   {3,1,4,5,2}
#I[6]::  Ppow[p2,6]
#O[6]:   {4,5,3,1,2}
#I[7]::  Pinv[p1]
#O[7]:   {1,4,5,3,2}
#I[8]::  Pcomp[%,p1]
#O[8]:   {1,2,3,4,5}
```

XPermC

Cycle decomposition of permutations

S.Wolfram
Jul 1981

Cycles are represented by projections of the form `C[i1,i2,i3,...]`.

```
<XPerm8
```

ToC[*perm*]

yields a list of cycles whose composition is *perm* (cycle decomposition).

```
ToC[$p_=Permp[$p]] :: (Lcl[%a,%t,%n,%l]; \
        %a:{};%t:Ar[Len[$p],1];Do[%1,Len[$p],If[%t[%1], \
                        For[%n:$p[%1];%l:{}, \
        %t[%n], %n:$p[%n], %t[%n]:0;%l:Cat[%l,{%n}]]; \
                %a:Cat[%a,{Ap[C,%l]}]]];%a)
```

ToP[*cycs*]

yields the permutation represented by the list of cycles *cycs*.

```
ToP[$cycs_=Listp[$cycs]] :: (Lcl[%l]; Map[Ar[Len[$c], \
        %l[Cyc[$c,-1][$i]]:$c[$i]],$cycs,1]; %l)
```

```
#I[1]::   <XPermC

#I[2]::   ToC[{1,5,7,2,4,3,6}]

#O[2]:    {C[1],C[5,4,2],C[7,6,3]}

#I[3]::   ToP[%]

#O[3]:    {1,5,7,2,4,3,6}
```

XPhys

Fundamental physical constants

S.Wolfram
Jul 1981

1. Principal constants

speed of light in vacuo
```
  SPhys[1,1] :    Qc -> 2.997924580*^8 metre second^-1
```

Planck's constant
```
  SPhys[1,2] :    Qh -> 6.626179*^-34 joule second
```

Dirac's constant
```
  SPhys[1,3] :    Qhbar -> 1.0545887*^-34 joule second
```

charge on electron
```
  SPhys[1,4] :    Qe -> 1.6021892*^-19 coulomb
```

mass of electron
```
  SPhys[1,5] :    Qme -> 9.109534*^-31 kilogram
```

Avogadro's number
```
  SPhys[1,6] :    QN -> 6.022045*^23 mole^-1
```

constant of gravitation
```
  SPhys[1,8] :    QG -> 6.6720*^-11 newton metre^2 kilogram^-2
```

2. Atomic constants

fine structure constant
```
  SPhys[2,1] :    Qalpha -> 7.2973506*^-3
```

Rydberg constant (infinite nuclear mass)
```
  SPhys[2,2] :    QRinf -> 1.097373177*^7 metre^-1
```

Bohr radius
```
  SPhys[2,3] :    Qa0 -> 5.2917706*^-11 metre
```

electron Compton wavelength
```
  SPhys[2,4] :    QlambdaC -> 2.4263089*^-12 metre
```

classical "electron radius"
```
  SPhys[2,5] :    Qre -> 2.817938*^-15 metre
```

Thomson cross-section
```
  SPhys[2,6] :    QsigmaT -> 6.6522448*^-29 metre^2
```

Bohr magneton

```
SPhys[2,7] :     QmuB -> 9.274078e^-24 joule tesla^-1
```

nuclear magneton
```
SPhys[2,8] :     QmuN -> 5.050824e^-27 joule tesla^-1
```

gyromagnetic ratio of free proton
```
SPhys[2,9] :     Qgamma -> 2.6751987e^8 second^-1 tesla^-1
```

electron volt
```
SPhys[2,10] :    QeV -> 1.6021892e^-19 joule
```

3. Thermal constants

molar gas constant
```
SPhys[3,1] :     QR -> 8.31441 joule kelvin^-1 mole^-1
```

Loschmidt number
```
SPhys[3,2] :     QL -> 2.686754e^25 metre^-3
```

Boltzmann constant
```
SPhys[3,3] :     Qk -> 1.380662e^-23 joule kelvin^-1
```

Stefan-Boltzmann constant
```
SPhys[3,4] :     Qsigma -> 5.67032e^-8 watt metre^-2 kelvin^-4
```

XPlhist

Histogram plotting

S.Wolfram
Jul 1981

Plhist[*list*]
plot *list* as a histogram.

```
Plhist[$list] :: Plot[{Ar[Len[$list],   \
Line[{Pt[{Ind[$list,$1],0}],Pt[{Ind[$list,$1],Elem[$list,{$1}]}], \
Pt[{Ind[$list,$1+1],Elem[$list,{$1}]}],Pt[{Ind[$list,$1+1],0}]}]], \
                    Axes[{0,0}]}]
```

XPlot

Operations on plots

S.Wolfram
Jul 1981

PCat[*p1*, *p2*, . . .]

combines the plots *p1*, *p2*, ... into a single plot.

```
PCat[Plot[$p1,$$p1],Plot[$p2,$$p2]] :: Plot[Union[$p1,$p2]]
```

PAp[*trx*, *try*, *pl*]

applies the templates *trx* and *try* respectively to each point in the plot *pl*.

```
PAp[$trx,$try,$pl] :: S[$pl,Pt[{$x,$y},$$f]--> \
                         Pt[{Ap[$trx,{$x}],Ap[$try,{$y}]},$$f]]
```

Ycut[*plot*, {*ymin*, *ymax*}]

replot plot in region *ymin* < *y* < *ymax*.

```
Ycut[Plot[$1,$$2],{$ymin,$ymax}] :: Plot[$1,,{$ymin,$ymax}]

Circle[{$x,$y},$r] :: \
        Graph[{$r Sin[%1]+$x,$r Cos[%1]+$y},%1,0,2Pi][1,1]
```

Future enhancements: Add functions to include labels, arrows etc.

XPolar

Polar graphs

S.Wolfram
Jul 1981

Polar[*expr*, *theta*, *npt*]
yields a polar plot of *expr* obtained by evaluation at *npt* points in the angle *theta*.

```
Polar[$expr,$theta,$npt] :: \
Graph[{$expr Cos[$theta],$expr Sin[$theta]},$theta,0,2Pi,,,$npt]
```

XPoly

Information on polynomials

S.Wolfram
Jul 1981

LCoef[*poly*, *var*]
yields a list of the coefficients of powers of *var* in *poly*.

```
LCoef[$poly,$var] :: Ar[Expt[$var,$poly],Coef[$var^$%1,$poly]]
```

LExpt[*poly*, *var*]
yields a list of the exponents with which *var* appears in *poly*.

```
LExpt[$poly,$var] :: Union[Expt[$var,$poly,List]]
```

```
#I[1]::   <XPoly

#I[2]::   t:Ex[(a+x)^3 (1-x)^2]

                 2        3        4       2         2  2        2  3         3      3  2
#O[2]:    3a x  - 6a x  + 3a x  + 3 a  x - 6 a  x  + 3 a  x  - 2 a  x + a  x

                  3     3      4     5
              + a  + x  - 2 x  + x

#I[3]::   LCoef[%,x]

              2      3           2    3                2
#O[3]:    {3 a  - 2 a ,3a - 6 a  + a ,1 - 6a + 3 a ,-2 + 3a,1}

#I[4]::   LExpt[t,x]

#O[4]:    {1,2,3,4,5}
```

XPolynum

Polygonal numbers

triangular numbers – pentagonal numbers – number theory functions
figurate numbers

S.Wolfram and P.Leyland

Jan 1982

Polynum[*k*, *n*]

k th *n* - gonal number (*n*:3 yields triangular numbers; *n*:4 squares)

```
Polynum[$k_=Natp[$k],$n_=Natp[$n]] : \
        $k ((Sn-2)$k+4-$n)/2
```

XPow

Simplification of powers

radicals – square root – canonical form – prime decomposition

S.Wolfram
Feb 1982

Reduce power of composite number to product of powers of primes

```
<<XPrimep
($x_=Intp[$x] & ~Primep[$x])^$y :: \
                            Ap[Mult,Map[Ap[$1^($2 $y),$0],Nfac[$x]]]
```

Warning: Redefines Pow; Pow processed more slowly.

XPrime

Potentially prime numbers

S.Wolfram and P.Leyland
Jan 1982

• Nfac; Prime; XPrimep

Fer[*n*]

n th Fermat number.

```
Fer[$n_=Natp[$n]] : B[2]^2^$n + 1
```

Mer[*n*]

n th Mersenne number.

```
_Mer[Init] :: <XPrimep
Mer[$n_=Primep[$n]] : B[2]^$n - 1
```

Inum[*n*]

n th I number.

```
Inum[$n_=Natp[$n]] : (B[10]^$n-1)/9
```

Jnum[*n*]

n th J number.

```
Jnum[$n_=Natp[$n]] : B[10]^$n + 1
```

XPrimep

Primality testing

S.Wolfram
Jul 1981

Primep[*expr*]
yields 1 if *expr* is a prime number, and 0 otherwise.

```
Primep[$expr] :: Natp[$expr] & P[Ap[Plus,Nfac[$expr]][2]=1]
```

Future enhancements: Should use a serious primality testing algorithm.

XProj

Projection manipulation

S.Wolfram
Jul 1981

PCat[*proj1*, *proj2*]

yields a projection whose filters are the concatenation of those in the projections *proj1*, *proj2* (which must have the same projector).

```
PCat[$x_=Projp[$x],$y_=(Projp[$y] & $y[0]=$x[0])] :: \
          Proj[$x[0],{S[$x,$x[0]->Np],S[$y,$y[0]->Np]}]
```

Pap[*temp*, *proj1*, *proj2*, . . .]

treats the filters of the projections *proj1*, *proj2*, ... as lists, and applies the template *temp* to them, yielding a projection with the resulting list as its filters.

```
_Pap[Smp]:{0}
Pap[$temp,$$proj_=(Ap[And,Map[Projp,{$$proj}]] & \
          Ap[Eq,Map[$%[0],{$$proj}]])] :: \
          Proj[$$proj[1][0],Ap[temp,Map[S[$%1[0]->Np,{$$proj}]]]]
```

XPrtable

Tabular output

printing — tabulation — tabular data — boxes — ruled
columnar — report generation — formatting — matrices

J.Greif
Nov 1981

vline[w]

make a vertical line of length w

```
vline[$w] :: vthing[$w,"|"]
```

vblank[w]

make a vertical blank of length w

```
vblank[$w] :: vthing[$w," "]
```

vthing[w, x]

make a column of vertical things x of height w

```
vthing[$w,$x]::Fmt[{[$xyzzy]: {0,$xyzzy}},Repl[ $x,$w]]
```

hline[w]

make a horizontal line of length w

```
hline[$w] :: hthing[$w,"-"]
```

hthing[w, x]

make a row of horizontal things x of length w

```
hthing[$w,$x]::Fmt[Null,Repl[ $x,$w]]
```

box[x]

make a box around expression x

```
box[$x] :: (Lcl[%h,%v,%f]; hv[%h,%v,$x]; \
Fmt[{{1,0},{-1,0},{Inf,0},{1,-Inf},{1,Inf},{0,Inf},{0,-Inf}},\
$x,vline[%v],vline[%v],\
hline[%h],hline[%h],hline[1],hline[1]])
```

vbar[x]

print a list x horizontally with bars separating the elements

```
vbar[$x_=Listp[$x]] :: bar[$x,vline]
bar[$x_=Listp[$x],$z] :: (Lcl[%v,%h,%s,%v1]; hv[%h,%v,$x]; \
%s:S[Ar[Len[$x],If[$i=Len[$x],[Null,$x[$i],Null],\
[Null,$x[$i],Null,$z[%v]]]],{$$u}->$$u,{0}])
```

col[x]

print list x horizontally inside box with bars separating the elts

```
col[$x_=Listp[$x]] :: box[Fmt[{[$s]:{$s,0}},vbar[$x]]]
```

topbot[x]

print object x with a horizontal bar above and below

```
topbot[$x] :: (Lcl[%f,%h]: %f:Prdsp[$x]; %h:%f[2]+%f[3]+2;\
Fmt[{}{1,0},{1,-Inf},{1,Inf},{0,Inf},{0,-Inf}},$x,\
hline[%h],hline[%h],hline[1],hline[1]])
```

hv[*h*, *v*, *x*]

find height *v* and width *h* of expression *x*

```
hv[$h,$v,$x] :: (Lcl[%f]; %f:Prdsp[$x]; $h:%f[2]+%f[3]+2; \
$v:%f[4]+%f[5]+1; )
```

colmat[*x*]

print out columnated, and boxed matrix, i.e. a table

```
colmat[$x_=Fullp[$x,2]] :: (Lcl[%h,%l,%v,%t,%q]; %t:Trans[$x];\
%l:Ar[Len[%t], Fmt[{[$s]:{0,-2$s}},S[%t[$i], {$$u}->$$u, {0}]]];\
col[%l])
```

Prmat[*x*]

print out matrix in 2-D form

```
Prmat[$x_=Fullp[$x,2]] :: (Lcl[%h,%l,%v,%t,%q]; %t:Trans[$x];\
%l:Ar[Len[%t], Fmt[{[$s]:{0,-2$s}},S[%t[$i], {$$u}->$$u, {0}]]];\
Fmt[{[$s]:{$s,0}},bar[%l,vblank]])
```

```
#I[1]::  <XPrtable

#I[2]::  Ar[{3,3},Pow]

#O[2]:   {{1,1,1},{2,4,8},{3,9,27}}

#I[3]::  Prmat[%]

         1   1   1

#O[3]:   2   4   8

         3   9   27

#I[4]::  colmat[@2]

          --------------
         |  1 | 1 | 1    |
         |    |   |      |
#O[4]:   |  2 | 4 | 8    |
         |    |   |      |
         |  3 | 9 | 27   |
          --------------
```

XPsmp

Heuristic simplification of rational forms

S.Wolfram
Jul 1981

Psmp[*expr*]
usually yields a usefully simplified form of the rational expression *expr*.

```
Psmp[$e] :: Map[Fac,Col[Ex[$e]]]
Psmp_Ldist
```

XQuadres

Quadratic residues

Number theory – modular arithmetic – rings

P.Leyland and S.Wolfram
Feb 1982

Quadres[*p*]
yields a list of all quadratic residues modulo *p*.

```
Quadres[$p_=Natp[$p-1]] :: Union[Cat[Ar[{{0,Floor[$p/2]}},Mod[$1^2,$p]]
```

XRAr

Array generation from recurrence relations

Sequence generation – congruential random number generation
recurrence relations

S.Wolfram

Aug 1982

RAr [*n*, *temp*, *start*]

generates *n* terms in a sequence starting from *start* by applying the template *temp* to each preceding term.

```
RAr[[$n_=Natp[$n],$temp,$start]] :: \
        (Lcl[t]; t:$start; Ar[$n,`(Lcl[to];to:t;t:Ap[$temp,{t}];to)])
```

```
#I[1]::  <XRAr

#I[2]::  RAr[5,f[$1],1]

#O[2]:   {1,f[1],f[f[1]],f[f[f[1]]],f[f[f[f[1]]]]}

#I[3]::  RAr[10,$1+2,1]

#O[3]:   {1,3,5,7,9,11,13,15,17,19}

#I[4]::  RAr[10,Mod[13 $1+7,11],1]

#O[4]:   {1,9,3,2,0,7,10,5,6,8}

#I[5]::  RAr[3,$1/(1+$1),a]

                 a                   a
#O[5]:   {a,-----,--------------------}
              1 + a                   a
                      (1 + a) (1 + -----)
                                   1 + a
```

XRamp

Ramp function

S.Wolfram
Jul 1981

Ramp[x]
represents the unit ramp function.

```
Ramp[$x] : (Abs[$x] + $x)/2
```

XRandC

Continuous random number generation

S.Wolfram
Jul 1981

NRand[(*x*:0),(*sd*:1)]
generates a random number from a normal (Gaussian) distribution with mean *x* and standard deviation *sd*.

```
NRand_:Tier
NRand[]::NRand[0,1]
NRand[$mean]::NRand[$mean,1]
NRand[$mean,$sd] :: \
        N[$mean + $sd Sqrt[-2 Log[Rand[]]] Cos[(!N[2Pi]) Rand[]]]
```

BRand[*rho*]
generates a pair of random numbers from a bivariate normal distribution with zero mean, unit variance and correlation coefficient *rho*.

```
BRand[$rho] :: (Lcl[%x1]; \
                {%x1:NRand[], N[$rho %x1 + Sqrt[1-$rho^2] NRand[]]})
```

ERand[*theta*]
generates a random number from the exponential distribution Exp[-x/*theta*].

```
ERand[$theta] :: N[-$theta Log[Rand[]]]
```

ARRand[*a*, *b*, *dist*, *maxdist*]
uses an acceptance-rejection method to generate a random number between *a* and *b* from the distribution defined by the template *dist* whose maximum value is not less than *maxdist*. Number of attempts necessary is proportional to (*b*-*a*) *maxdist*.

```
ARRand[$a,$b,$dist,$maxdist] :: (Lcl[%y];  \
                Loop[,%y:N[$a + ($b-$a) Rand[]], \
                Rand[] >= N[Ap[$dist,{%y}]/$maxdist]]; %y)
```

XRandD

Discrete random number generation

S.Wolfram
Jul 1981

IRand[*n*]
generates a random integer from a uniform distribution between 0 and n-1.

```
IRand[$n] :: Floor[Rand[$n]]
```

PNorm[*list*]
yields a normalized list of probabilities from a list of relative frequencies.

```
PNorm[$list] :: N[$list/Ap[Plus,$list]]
```

PCum[*list*]
yields a list of cumulative probabilities.

```
PCum[$list] :: (Lcl[%t]; %t:0; Map[%t:%t+$%1;%t,PNorm[$list]])
```

DRand[{*cp1*, *cp2*, ...}]
yields a random position in the list with distribution determined by the cumulative probabilities *cpi*.

Misfeatures: Not optimal algorithm

```
DRand[$list] :: (Lcl[%x]; %x:Rand[]; Pos[$1_=$1>%x,$list,2,1][1,2,1])
```

XRandL

Random list element selection

S.Wolfram
Jul 1981

LRand[*list*]

yields one of the elements of *list*, randomly chosen with equal probabilities.

```
⊥LRand[Init] :: <XRandD
LRand[$list_→Listp[$list]] :: Ent[$list,{1+IRand[Len[$list]]}]
```

LDRand[*list*, *prob*]

yields one of the elements of *list*, randomly chosen with relative frequencies given by *prob*.

```
⊥LDRand[Init] :: <XRandD
LDRand[$list,$prob_→(Contp[$prob] & Len[$prob]=Len[$list])] :: \
        $list[DRand[PCum[$prob]]]
```

ORand[*list*]

yields a random reordering of *list*.

```
_ORand[Init] :: <XPerm
ORand[$list] :: (Lcl[%1]; %1:Ar[Len[$list],`Rand[]]; \
        Apper[Fiper[%1,Sort[%1]],$list])
```

XRepart

Graphical part replacement

Repart[*expr*]
replaces a graphically-selected part of *expr*.

```
Repart[$e] :: Ap[Set,{Ap[$e,{L[$e]}],Rd["new part:"]}]
_Repart[Smp]:0
```

XRev

Symbolic identities for list operations

S.Wolfram

Jul 1981

```
Rev[Rev[$x]] : $x

Cyc[Cyc[$list,$n1],$n2] :: Cyc[$list,$n1+$n2]
```

XRnd

Integer rounding

truncation – type casting – entier – floor – ceiling

S.Wolfram
Sep 1982

Rnd[*n*]
rounds *n* to the nearest integer.

```
Rnd[$n_:Numbp[$n]] :: Floor[$n+0.5]
```

```
#I[1]::  <XRnd
#I[2]::  Rnd[2.2]
#O[2]:   2
#I[3]::  Rnd[2.7]
#O[3]:   3
#I[4]::  Rnd[-3.6]
#O[4]:   -4
```

U/6

XRot2

Rotations in two dimensions

S.Wolfram
Jul 1981

Vec2p[*expr*]

tests whether *expr* represents a two-dimensional vector.

```
Vec2p[$expr] :: Contp[$expr] & Len[$expr]=2
```

RotM2[*theta*]

yields a matrix representing a two-dimensional rotation through an angle of *theta* radians.

```
RotM2[$theta] : {{Cos[$theta],Sin[$theta]},{-Sin[$theta],Cos[$theta]}}
```

Rot2[*vec*, *theta*, (*pt*: {0,0})]

rotates the two-dimensional vector *vec* about the point *pt* through an angle of *theta* radians.

```
Rot2_:Tier
Rot2[$vec_=Vec2p[$vec],$theta] : RotM2[$theta].$vec
Rot2[$vec_=Vec2p[$vec],$theta,$pt_=Vec2p[$pt]] :: \
                              $pt + RotM2[$theta].($vec-$pt)
```

XRot3

Rotations in three dimensions

S.Wolfram
Jul 1981

Vec3p[*expr*]

tests whether *expr* represents a three-dimensional vector.

```
Vec3p[$expr] :: Contp[$expr] & Len[$expr]=3
```

RotM3[*phi*, *theta*, *psi*]

yields a matrix representing a three-dimensional rotation specified by the Euler angles *phi*, *theta*, *psi*.

```
RotM3[$phi,$theta,$psi] : \
          {{Cos[$psi] Cos[$phi] - Cos[$theta] Sin[$phi] Sin[$psi], \
           Cos[$psi] Sin[$phi] + Cos[$theta] Cos[$phi] Sin[$psi], \
           Sin[$psi] Sin[$theta]}, \
           {-Sin[$psi] Cos[$phi] - Cos[$theta] Sin[$phi] Cos[$psi], \
           -Sin[$psi] Sin[$phi] + Cos[$theta] Cos[$phi] Cos[$psi], \
           Cos[$psi] Sin[$theta]}, \
           {Sin[$theta] Sin[$phi], -Sin[$theta] Cos[$phi], Cos[$theta]}}
```

Rot3[*vec*, *phi*, *theta*, *psi*, (*pt*: {0,0,0})]

rotates the three-dimensional vector *vec* about the point *pt* through the Euler angles *phi*, *theta*, *psi*.

```
Rot3_:Tier
Rot3[$vec_=Vec3p[$vec],$theta] : RotM3[$theta].$vec
Rot3[$vec_=Vec3p[$vec],$theta,$pt_=Vec3p[$pt]] :: \
                              $pt + RotM3[$theta].($vec-$pt)
```

XRpoly

Random polynomial generation

S.Wolfram
Jul 1981

Ranup[(*n*:10)]
generates a random univariate polynomial of size *n*.

```
Ranup_:Tier
Ranup[$n_:Natp[$n]] :: \
        Ex[Rex[$n,{{x,1},{2,1},{-1,1}},{{Plus,1,-7},{Mult,1,-5}}]]
Ranup[] :: Ranup[10]
```

Ranmp[(*n*:10),(*nx*:3)]
generates a random polynomial of size *n* in an average of *nx* variables.

```
Ranmp_:Tier
Ranmp[$n_:Natp[$n],$nx_:Natp[$nx]] :: \
        Ex[Rex[$n,Cat[Ar[{{10,10+$nx-1}},Impl[{$%1}]],{{2,1},{-1,1}}],\
                {{Plus,1,-7},{Mult,1,-5}}]]
Ranmp[$n_:Natp[$n]] :: Ranmp[$n,3]
Ranmp[] :: Ranmp[10,3]
```

```
#I[1]::  <XRpoly

#I[2]::  Ranup[10]

                3        4       5
#O[2]:   168 x  + 240 x  + 78 x

#I[3]::  Ranup[10]

                     2
#O[3]:   2 + 5x + 8 x

#I[4]::  Ranmp[10,3]

                                     2
#O[4]:   12 + 3a + 2b + 4c + 8a c

#I[5]::  Ranmp[10,3]

                                    2   5          2   6         3   5          2      5
#O[5]:   12b + 10a b + 4b c + 96 a  c  + 32 a  c  + 88 a  c  + 16 a  b c

                 2
            + 2 b
```

XScan

Scan function

scan – search – mu function – find – first

S.Wolfram
Jan 1982

Scan[*temp*, *n*]

yields a list of the first *n* positive integers satisfying the criterion *temp*.

```
_Scan[Smp]:{Inf,8}
Scan_:Tier
Scan[$temp,$n_=Natp[$n]] :: (Lcl[%t,%i,%f]; For[%i:0;%t:1, \
        %i<$n, Inc[%t], If[Rel[Ap[$temp,{%t}]],%f[Inc[%i]]:%t]]; %f)
```

Future enhancements: Extend to k-tuples of integers

```
#I[1]::   <XScan

#I[2]::   Scan[Intp[N[Sqrt[$1]]],5]

#O[2]:    {1,4,9,16,25}

#I[3]::   Scan[$1^2-4$1>3,2]

#O[3]:    {5,6}
```

XSerSol

Series solution of differential equations

power series – ordinary differential equations

J.Greif

Aug 1982

PsSol[***dexpr*****,** ***serord*****,** ***bc*****,** (***dep***:y)**,** (***ind***:x)]

solves the differential equation *dexpr* = 0 for the dependent function *dep* as a function of the independent variable *ind* in power series form to order *serord*, given boundary conditions *bc* expressed as a list of expressions to be set = 0

```
PsSol_:Tier
_PsSol[Smp]:{0,Inf,0,0,Inf}
PsSol[$dex,$so,$bc]::PsSol[$dex,$so,$bc,y,x]
PsSol[$dex,$so,$bc,$dep,$ind]:: (Lcl[%y,%dep,%ex,%l,%bc,%l,%a,qdiff]; \
          %ex:S[$dex,$dep->%dep]; %bc:S[$bc,$dep->%dep];\
          %dep_:Tier; %y_:Tier; \
          %y[$x,$n]::Ps[1,$x,0,$n,{[$i]:%a[$i]}]; \
          %dep:%y[$ind,$so]; \
          only now do we know indep var so can define qdiff\
          qdiff[%dep[$y]]::D[%dep[$ind],{$ind,1,$y}]; \
          qdiff[D[%dep[$y],{$y,$n,$z}]]::D[%dep[$ind],{$ind,$n+1,$z}]; \
          qdiff[$x]::D[$x,{$ind,1,$ind}]; \
          %ex:Si[%ex,Hold[D[%dep[$x],{$x,$n,$z}]]-> \
                    D[%y[$ind,$so],{$ind,$n,$z}]]; \
          next line gets around incomplete simplification bug\
          %ex:S[%ex,DPAT-->0];\
          %ex:Si[%ex,%dep[$ind]->%y[$ind,$so]]; \
          make a properly truncated Ps if still a sum of terms\
          If[P[%ex[0]=Hold[Plus]],%ex:Ps[%ex,$ind,0,$so]];\
          %bc:Si[%bc,Hold[D[%dep[$x],{$x,$n,$z}]]-> \
                    D[%y[$ind,$so],{$ind,$n,$z}]]; \
          next line gets around incomplete simplification bug\
          %bc:S[S[%bc,DPAT-->0], Hold[Ps[$$x]]-->Ax[Ps[$$x]]];\
          %bc:Si[%bc,%dep[$x]->Lim[%y[$ind,$so],$ind,$x]]; \
          %ex is a Ps and %bc is an expr or list \
          %l:If[Listp[%bc], Ldist[%bc=0], {%bc=0}];\
          %l:Union[Cat[%l,Ldist[%ex[5]=0]]]; \
          %s: Sol[%l, Cat[Ar[{{0,$so}},%a]]]; \
          Ps[1,$ind,0,$so,Ar[{{0,$so}},S[%a[$1],%s]]])
```

Input niceties -- the differential expression can be entered in the form

```
f[x] y' + g[x] y''+ h[x] y - i[x]
```

```
Sxset["'",qdiff,2]
_qdiff[Pr,$x]::Sx["'",{$x},2]
```

Defeat encasement extension

```
_Ps[Extr,qdiff]::qdiff
_Ps[Extr,Pr]::Pr
_Ps[Extr,Eq]::Eq
_Ps[Extr,Hold]::Hold
_Ps[Extr,Rep]::Rep
_Ps[Extr,Si]::Si
```

```
#I[1]::  <XSerSol

#I[2]::  de: Cos[x] y' + Exp[-3x] y'' + Sin[x] y - Sin[14x]

#O[2]:*  y Sin[x] + (y') Cos[x] + ((y')') Exp[-3x] - Sin[14x]
```

```
#I[3]::  bc:{y[0]-2, y[0]'-1}

#O[3]:*  {-2 + y[0],-1 + (y[0]')}

#I[4]::  PsSol[de,8,bc]
```

#O[4]:* $2 + x - \frac{x^2}{2} + \frac{5 x^3}{3} + \frac{29 x^4}{12} - \frac{515 x^5}{24} - \frac{10091 x^6}{240} + \frac{59567 x^7}{840} + \frac{1448983 x^8}{6720}$

Misfeatures: Has fixed point of expansion 0. Assumes Taylor series will work. Thus solution must exist and be finite at x=0.

XSets

Elementary finite set theory

S.Wolfram

Jul 1981

Sets are represented as ordered lists with no repeated elements. A list may be placed in this canonical form by Union[list]. Once in this form, the equivalence of two sets is determined by Eq.

Union[*set1*, *set2*, . . .]
forms the union of the sets *set1*, *set2*, ...

Inter[*set1*, *set2*, . . .]
forms the intersection of the sets *set1*, *set2*, ...

Cmpl[*set*, *uset*]
yields the complement of *set* with respect to the universal set *uset*.

```
Cmpl[$list_->Listp[$list],$ulist_->Listp[$ulist]] ::  \
                    Cat[Ar[Len[$ulist],$ulist,,~In[$%1,$list]]]
```

Sub[*set*, *crit*]
yields a subset of *set* all of whose elements satisfy the condition *crit*.

```
Sub[$set,$crit] :: Flat[Ar[Len[$set],,,$crit]]
```

Subp[*sub*, *set*]
yields 1 if *sub* is a subset of *set* and 0 if it is not.

```
Subp[$sub,$set] :: Len[Union[$sub,$set]]=Len[Union[$set]]
```

Disjp[*set1*, *set2*]
yields 1 if the sets *set1* and *set2* are disjoint (have no elements in common).

```
Disjp[$set1,$set2] :: P[Inter[$set1,$set2]={}]
```

Multset[*set1*, *set2*, . . .]
forms the product set of the *seti* (set of all possible ordered n-tuples of elements from n sets).

```
Multset[$$set] :: Flat[Outer[List,$$set],Len[List[$$set]]-1]
```

Powset[*set*]
forms the set of all possible subsets of *set* (power set).

```
Powset[$l_->Contp[$l]] :: Flat[Lcl[%f,%g]; S[Dist[Ap[%g,Ar[Len[$l], \
        %f[{},{$l[$%1]}]]],{%g,%f,List,%g}],%g->Cat],Len[$l]-1]
```

```
#I[1]::   <XSets

#I[2]::   s1:Union[{a,b,a,c,d}]

#O[2]:    {a,b,c,d}
```

```
#I[3]::  s2:{c,e,f}
#O[3]:   {c,e,f}
#I[4]::  s3:Ar[5]
#O[4]:   {1,2,3,4,5}
#I[5]::  Union[s1,s2,s3]
#O[5]:   {1,2,3,4,5,a,b,c,d,e,f}
#I[6]::  Inter[s1,s2]
#O[6]:   {c}
#I[7]::  Cmpl[s2,s1]
#O[7]:   {a,b,d}
#I[8]::  Sub[s3,Evenp]
#O[8]:   {2,4}
#I[9]::  Subp[{a,c},s1]
#O[9]:   1
#I[10]::  Disjp[s1,s3]
#O[10]:   1
#I[11]::  Multset[s1,s2]
#O[11]:   {{{a,c},{a,e},{a,f}},{{b,c},{b,e},{b,f}},{{c,c},{c,e},{c,f}},
                {{d,c},{d,e},{d,f}}}
#I[12]::  Powset[s2]
#O[12]:   {{},{f},{e},{e,f},{c},{c,f},{c,e},{c,e,f}}
```

XSetsSX

Set theory notation

S.Wolfram
Jul 1981

aUbUc... or a U b U c ...
stands for Union[a,b,c,...].

```
Sxset["U",Union,3]
```

aIbIc... or a I b I c ...
stands for Inter[a,b,c,...].

```
Sxset["I",Inter,3]
```

a\b or a \ b
stands for Cmpl[b,a].

```
Sxset["\",Cmpl,5]
```

aCb or a C b
stands for Subp[a,b].

```
Sxset["C",Subp,4]
```

a^*b^*c... or a ^* b ^* c ...
stands for Multset[a,b,c...].

```
Sxset["^*",Multset,3]
```

^^a or ^^ a
stands for Powset[a].

```
Sxset["^^",Powset,1]
```

XSfctl

Subfactorial

derangements – rencontres numbers

S.Wolfram
Sep 1982

Sfctl[*n*]

subfactorial of *n* (number of derangements of *n* objects).

```
Sfctl[$n_=Natp[$n]] : $n Sfctl[$n-1] + (-1)^$n
Sfctl[1] : 1
```

[Sloane: Handbook of Integer Sequences, sect. 3.13]

XShowtime

Display of simplification times

S.Wolfram
Jul 1982

Showtime

causes simplification times for each output line to be printed.

```
Showtime :: (Pre :: (Lcl[%1];Pr[Time[%1:$1]];If[Valp[%1],%1,]))
Showtime1 :: (Post :: (Pr[N[Last[#T]]];$1))
```

XSign

Sign simplification rules

S.Wolfram
Jul 1981

```
Sign[$x $$x] :: Sign[$x] Sign[$$x]
Sign[$x^($n_Evenp[$n])] : 1
Sign[Abs[$x]] : 1
```

XSol

Inverses of elementary transcendental functions

S.Wolfram

Jul 1981

```
Sol[Exp[$x]=$y,$x]  ::  Sol[$x=Log[$y],$x]
Sol[Log[$x]=$y,$x]  ::  Sol[$x=Exp[$y],$x]
Sol[Sin[$x]=$y,$x]  ::  Sol[$x=Asin[$y],$x]
Sol[Cos[$x]=$y,$x]  ::  Sol[$x=Acos[$y],$x]
Sol[Tan[$x]=$y,$x]  ::  Sol[$x=Atan[$y],$x]
Sol[Asin[$x]=$y,$x]  ::  Sol[$x=Sin[$y],$x]
Sol[Acos[$x]=$y,$x]  ::  Sol[$x=Cos[$y],$x]
Sol[Atan[$x]=$y,$x]  ::  Sol[$x=Tan[$y],$x]
Sol[Sinh[$x]=$y,$x]  ::  Sol[$x=Asinh[$y],$x]
Sol[Cosh[$x]=$y,$x]  ::  Sol[$x=Acosh[$y],$x]
Sol[Tanh[$x]=$y,$x]  ::  Sol[$x=Atanh[$y],$x]
Sol[Asinh[$x]=$y,$x]  ::  Sol[$x=Sinh[$y],$x]
Sol[Acosh[$x]=$y,$x]  ::  Sol[$x=Cosh[$y],$x]
Sol[Atanh[$x]=$y,$x]  ::  Sol[$x=Tanh[$y],$x]
Sol[Gd[$x]=$y,$x]  ::  Sol[$x=Agd[$y],$x]
Sol[Agd[$x]=$y,$x]  ::  Sol[$x=Gd[$y],$x]
```

XSpare

Removal of almost all values

S.Wolfram
Jul 1981

Spare[*v1*, *v2*, ...]
removes all values except those of *v1*, *v2*, ...

```
_Spare[Smp]:8
Spare[$$x] :: (Lcl['Spare,$$x]; Set[])
```

```
#I[1]::  <XSpare

#I[2]::  a:b:c:1

#O[2]:   1

#I[3]::  Spare[a]

#O[3]:   Spare[' a]

#I[4]::  {a,b,c}

#O[4]:   {1,b,c}
```

XStat

Univariate statistics

S.Wolfram
Jul 1981

Mean[*list*]
gives the mean of the values in *list*.

```
Mean[$list] :: Ap[Plus,$list]/Len[$list]
```

GMean[*list*]
gives the geometric mean of the values in *list*.

```
GMean[$list] :: N[Ap[Mult,$list^(1/Len[$list])]]
```

HMean[*list*]
gives the harmonic mean of the values in *list*.

```
HMean[$list] :: N[Len[$list]/Ap[Plus,Map[1/$1,$list]]]
```

Med[*list*]
gives the median of the values in *list*.

```
Med[$list_→Oddp[Len[$list]]] :: Sort[$list][(Len[$list]+1)/2]
Med[$list_→Evenp[Len[$list]]] :: \
          (Lcl[%l]; %l:Sort[$list]; (%l[Len[$list]/2] + \
                    %l[Len[$list]/2+1])/2 )
```

MD[*list*]
gives the mean deviation of the values in *list*.

```
MD[$list] :: Mean[Abs[$list-Mean[$list]]]
```

Var[*list*]
gives the variance of the values in *list*.

```
Var[$list] :: Mean[($list-Mean[$list])^2]
```

SD[*list*]
gives the standard deviation of the values in *list*.

```
SD[$list] :: N[Sqrt[Var[$list] Len[$list]/(Len[$list]-1)]]
```

Range[*list*]
gives the range of the values in *list*.

```
Range[$list] :: Ap[Max,$list]-Ap[Min,$list]
```

RMS[*list*]
gives the root mean square of the values in *list*.

```
RMS[$list] :: N[Sqrt[Mean[$list^2]]]
```

Fract[*list*, *r*]
yields the *r* fractile of the values in *list*.

```
_Fract[Init] :: <XLItp
Fract[$list,$r] :: LItp3[Sort[$list],$r Len[$list]]
```

Q1 [*list*]
yields the first quartile of the values in *list*.

```
Q1[$list] :: Fract[$list,1/4]
```

Q3 [*list*]
yields the third quartile of the values in *list*.

```
Q3[$list] :: Fract[$list,3/4]
```

QD [*list*]
yields the quartile deviation of the values in *list*.

```
QD[$list] :: (Q3[$list] - Q1[$list])/2
```

Mom [*list*, *n*]
yields the *nth* moment of the values in *list*.

```
Mom[$list,$n] :: Mean[$list^$n]
```

Cmom [*list*, *n*]
yields the *nth* central moment of the values in *list*.

```
Cmom[$list,$n] :: Mean[($list-Mean[$list])^$n]
```

Cum [*list*, *n*]
yields the *nth* cumulant of the values in *list*.

```
Cum[$list,1] :: Mean[$list]
Cum[$list,2] :: Var[$list]
Cum[$list,3] :: Cmom[$list,3]
Cum[$list,4] :: Cmom[$list,4] - 3 Cmom[$list,2]^2
Cum[$list,5] :: Cmom[$list,5] - 10 Cmom[$list,3] Cmom[$list,2]
```

Skew [*list*]
gives the coefficient of skewness of the values in *list*.

```
Skew[$list] :: Cmom[$list,3]/SD[$list]^3
```

Kurt [*list*]
gives the coefficient of kurtosis of the values in *list*.

```
Kurt[$list] :: Cmom[$list,4]/SD[$list]^4
```

Excess [*list*]
gives the coefficient of excess of the values in *list*.

```
Excess[$list] :: Cmom[$list,4]/SD[$list]^4 - 3
```

Qskew [*list*]
gives the quartile coefficient of skewness of the values in *list*.

```
Qskew[$list] :: (Q3[$list] - 2 Med[$list] + Q1[$list]) /  \
                         (Q3[$list] - Q1[$list])
```

Expc [*op*, *list*]
yields the expectation value of the operator represented by the template *op* on the values in *list*.

```
_Expc[Smp]:{0,Inf}
Expc[$op,$list] :: Rel[Ap[Plus,Map[$op,$list]]]/Len[$list]
```

Char[*list*, *x*]

yields the characteristic function of the values in *list*.

```
Char[$list,$x] :: Expc[Exp[I $1 x],$list]
```

```
#I[1]::  <XStat

#I[2]::  t:Ar[20,'Rand[]]

#O[2]:   {.7512801,.9799641,.3078826,.1427415,.8638017,.1446294,.4414654,
              .2309524,.664948,.1285033,.1148291,.243175,.5170353,.4259514,
              .6094183,.9010086,.02981599,.4496743,.06934847,.4439974}

#I[3]::  Post:N

#O[3]:   N

#I[4]::  Mean[t]

#O[4]:   .4230211

#I[5]::  GMean[t]

#O[5]:   .3051757

#I[6]::  HMean[t]

#O[6]:   .1837451

#I[7]::  Med[t]

#O[7]:   .4337084

#I[8]::  SD[t]

#O[8]:   .2941784

#I[9]::  Range[t]

#O[9]:   .9501481

#I[10]::  RMS[t]

#O[10]:   .5110389

#I[11]::  Q1[t]

#O[11]:   .1427415

#I[12]::  Q3[t]

#O[12]:   .6094183

#I[13]::  Skew[t]

#O[13]:   .4057557

#I[14]::  Kurt[t]

#O[14]:   1.824871

#I[15]::  Expc[Exp,t]

#O[15]:   1.592872
```

XStatus

Status information

S.Wolfram
Jul 1981

Status

prints status information.

```
Status :: \
        (Pr["Memory used so far:",State[][3],"blocks"]; \
        Pr["Total CPU time so far:",N[Clock[][2]],"seconds"]; \
        Pr["Real time so far:",Clock[][1],"seconds"]; \
        Pr["Number of input lines:",Len[#I]]; \
        Pr["Number of user symbols:",Len[Cont[]]]; \
        Pr["Total number of symbols:",Len[Cont[,1]]])
```

XStr0

Basic character string manipulation

S.Wolfram
Jul 1981

Char[*str*, *i*]
yields the *i* th character in the string *str*.

```
Char[$str,$i_⇒(0<$i⩽CLen[$str])] :: Impl[Expl[$str][$i]]
```

CLen[*str*]
gives the number of characters in the string *str*.

```
CLen[$str] :: Len[Expl[$str]]
```

CRev[*str*]
reverses the string *str*.

```
CRev[$str] :: Impl[Rev[Expl[$str]]]
```

CJoin[*str1*, *str2*, . . .]
concatenates the strings *str1*, *str2*, ...

```
CJoin[$$s] :: Impl[Ap[Cat,Map[Expl,List[$$s]]]]
```

```
#I[1]::  <XStr0
#I[2]::  t:"A character string"
#O[2]:   "A character string"
#I[3]::  Char[t,4]
#O[3]:   h
#I[4]::  Char[t,2]
#O[4]:   " "
#I[5]::  CLen[t]
#O[5]:   18
#I[6]::  CRev[t]
#O[6]:   "gnirts retcarahc A"
#I[7]::  CJoin[%,t,t]
#O[7]:   "gnirts retcarahc AA character stringA character string"
```

XStr1

Further character string manipulation

S.Wolfram
Jul 1981

```
<<List1
```

CRep [*str1*, *str*, *i*]
replaces the character at position *i* in *str* by the string *str1*.

```
CRep[$str1,$str,$i] :: (Lcl[%I]; %I:Expl[$str]; %I[$i]:Expl[$str1];   \
                        Impl[Flat[%I]])
```

CIns [*str1*, *str*, *i*]
inserts the character string *str1* at position *i* in *str*.

```
_CIns[Init] :: <<List8
CIns[$str1,$str,$i] :: Impl[Ins[Expl[$str1],Expl[$str],$i]]
```

CPos [*form*, *str*]
yields a list of the positions of the substring *form* in the string *str*.

```
CPos[$form,$str] :: LPos[Expl[$form],Expl[$str]]
```

CS [*str*, *rep1*, *rep2*, . . .]
applies successively the replacements *repi* for substrings of the string *str*. Each replacement is used until it is no longer applicable. The special "generic character" represents any single character.

```
CS[$str,$$reps] :: Impl[Ap[LS,Cat[{Expl[$str]}, \
          S[Map[Expl[$1[1]]->Expl[$1[2]],List[$$reps]], \
                ((Expl["$"][1])->$%1]]]]
```

```
#I[1]::  <<XStr1

#I[2]::  s:"the cat in the hat"

#O[2]:   "the cat in the hat"

#I[3]::  CRep[RRRR,s,5]

#O[3]:   "the RRRRat in the hat"

#I[4]::  CIns[" not",s,8]

#O[4]:   "the cat not in the hat"

#I[5]::  CPos[the,s]

#O[5]:   {1,12}

#I[6]::  CS[s,the->a,$at->XXXXX]

#O[6]:   "a XXXXX in a XXXXX"
```

XSubl

Sublists

form sublists – unflatten – group – sequences

S.Wolfram
Jan 1982

Subl [*list*, *n*]
generates a list of successive groups of *n* entries in *list*.

```
Subl[$list_=Listp[$list],$n_=Natp[$n]] :: \
     Ar[Len[$list]-$n+1,Ar[$n,$list[$1+$2-1]]]
```

• UnFlat; Outer; Trl

```
#I[1]::  <XSubl
#I[2]::  t:Ar[8]
#O[2]:   {1,2,3,4,5,6,7,8}
#I[3]::  Subl[t,2]
#O[3]:   {{1,2},{2,3},{3,4},{4,5},{5,6},{6,7},{7,8}}
#I[4]::  Subl[t,7]
#O[4]:   {{1,2,3,4,5,6,7},{2,3,4,5,6,7,8}}
```

XSum

Series summation

S.Wolfram
Jul 1981

Canonical forms for sums

Canonical form for limits

```
SSum[1] : Sum[$e,{$i,$1,$2}] --> Sum[S[$e,$i->$i-$1+1,-1,1],{$i,1,$2-$1+1}]
```

Sums of rational functions

```
SSum[2] : Sum[$e,{$i,$1,$2}] --> Sum[Pf[$e,$i],{$i,$1,$2}]
```

Simplification of sums

```
_Sum[Smp]:Inf

Sum[$x+$$x,{$i,$1,$2}] :: Sum[$x,{$i,$1,$2}] + Sum[$$x,{$i,$1,$2}]

Sum[$x,{$i_=(~In[$i,$x]),$1,$2}] :: ($2-$1+1) $x

Sum[$$n $x,{$i_=~In[$i,$$n],$1,$2}] :: $$n Sum[$x,{$i,$1,$2}]

Sum[$x/($$n $y),{$i_=~In[$i,$$n],$1,$2}] :: Sum[$x/$y,{$i,$1,$2}]/$$n
```

Sums of positive powers

```
Sum[$i,{$i,1,$n}] : $n($n+1)/2

Sum[$i^($m_=Natp[$m]),{$i,1,$n}] : (Ber[$m+1,$n+1]-Ber[$m+1])/($m+1)

Sum[(-$a)^$i $i^($m_=Natp[$m]),{$i_=~In[$i,$a],1,$n}] : \
        ((-1)^$n Eul[$m,$n+1] - Eul[$m,0])/2
```

Geometric progression

```
Sum[$a^$i,{$i_=~In[$i,$a],1,$n}] : ($a^$n - 1)/($a - 1)
```

Warning: Sum redefined to simplify its arguments

XSumPR

Special output form for Sum

S.Wolfram
Jul 1981

```
<XPR
_Sum[Pr][[$expr,{$var,$start,$end}]] :: \
Fmt[{{1,0},{3,0},{1,-1},{1,1}},PSig,$expr,$var=$start,$end]
```

```
#I[1]::  <XSumPR

#I[2]::  1+Sum[1/i^a,{i,1,Inf}]

               Inf
               ---
               \       -a
#O[2]:=  1 +   /      i
               ---
               i = 1
```

XSympol

Symmetric polynomial generation

Sympol[*n*, *x*]
generates a list of all symmetric polynomials in *n* variables *x*[i].

```
Sympol[$n_=Natp[$n],$x] :: (Lcl[%x]; \
                Ps[Ex[Prod[(1+$x[%i] %x),{%i,1,$n}]],%x,0,$n][5])
```

XTEST

General testing routines

S.Wolfram
Jul 1981

Basic test function

```
_Test[Smp]:8
Test[$f,$arg,$n] :: \
        Rpt[tin:Rel[$arg];Pr[];Pr[Rel[$f],tin];Pr[tout:Rel[Ap[$f,tin]]],$n]
```

Seed random numbers

```
Seed[$n] :: Rand[1,$n]
```

Random argument generators

numb[*x*]
: positive or negative number of maximum modulus x.

```
numb[$x] :: -$x + Rand[2$x]
```

pnumb[*x*]
: positive number of maximum size x.

```
pnumb[$x] :: Rand[$x]
```

int[*x*]
: positive or negative integer of maximum modulus x.

```
int[$x] :: Gint[numb[$x]]
```

pint[*x*]
: positive integer of maximum modulus x.

```
pint[$x] :: Gint[1+Rand[$x]]
```

nnint[*x*]
: non-negative integer of maximum modulus x.

```
pint[$x] :: Gint[Rand[$x]]
```

expr[n]
: expression of size n.

```
expr[$n] :: Rex[$n]
```

temp[*n*]
: template of size n.

```
temp[$n] :: S[Rex[$n],x->$1,z->$2]
```

```
<XRpoly
```

poly[*n*]

univariate polynomial of size *n*.

```
poly[$n] :: Ranup[$n]
```

mpoly[*n*, *m*]

multivariate polynomial in about *m* variables of size *n*.

```
mpoly[$n,$m] :: Ranmp[$n,$m]
```

list[*form*, *n*]

length <= *n* list of *forms*.

```
list[$form,$n] :: Ar[Gint[Rand[$n]+0.3],`Rel[$form]]
list_:Nosmp
```

rpt[*form*, *n*]

repetition of <= *n* occurrences of *form*

```
rpt[$form,$n] :: Ap[Np,Ap[list,{$form,$n}]]
rpt_:Nosmp
```

When Cons works for Rand, Cons all numerical routines above, and allow numbers to be found from exponential distribution, as obtained in a Cons routine (exponential distribution is more realistic than uniform in most cases).

XTEx

Tensor expansion

indicial tensor calculus – tensor canonicalization
tensor symmetries

S.Wolfram
Jul 1981

TEx[*f*[*x1*, *x2*, ...], *reor*]
constructs a sum of tensor obtained by reordering the *xi* and assigning the factors specified by the permutation symmetries *reor*.

```
<XPerm8
<XArperm
TEx_:Tier
TEx[$f[$$x],$reor] :: (Lcl[%l,%o,%t,%c,%f]; _%f[Reor]:$reor; \
%o:Ap[%f,List[$$x]]; %t:0; Ap[Plus,Map[%l:Apper[$1,List[$$x]]; \
        If[In[%f,%c:Ap[%f,%l]/%o],0,Inc[%t,%c];%c Ap[$f,%l]], \
              Arperm[Len[List[$$x]]]]]/%t)
```

```
#I[1]::  <XTEx

#I[2]::  TEx[f[a,b],Sym]

         f[a,b] + f[b,a]
#O[2]:   ---------------
                2

#I[3]::  TEx[f[a,b,c],Sym]

         f[a,b,c] + f[a,c,b] + f[b,a,c] + f[b,c,a] + f[c,a,b] + f[c,b,a]
#O[3]:   ----------------------------------------------------------------
                                        6

#I[4]::  TEx[f[a,b,c],Cyclic]

         f[a,b,c] + f[b,c,a] + f[c,a,b]
#O[4]:   ------------------------------
                       3
```

XTensor

Tensor manipulations

S.Wolfram
Jul 1981

Tenp[*f*]

determines whether *f* is of type Tensor.

```
Tenp[$f] :: P[_$f[Type]=Tensor]
```

CO

denotes the basis coordinates.

```
CO : {r,theta,phi,t}
```

DIM

denotes the dimensionality of spacetime (default 4).

```
DIM:Len[CO]
```

KD[*mu, nu*]

denotes the Kronecker delta symbol.

```
_KD[Reor]:Sym
KD[$mu,$mu] : DIM
KD[$mu,$nu_/($mu~=$nu)] : 0
```

Gm[*mu, nu*]

represents the metric tensor, assumed symmetric.

```
Gm_Tensor
_Gm[Reor]:Sym
Gm[$mu_/Symbp[$mu],`$nu] : KD[$mu,$nu]

Gm[$mu_/Symbp[$mu],$nu_/Symbp[$nu]] $f[$$i1,`$mu,$$i2] :  \
                                $f[$$i1,$nu,$$i2]
Gm[$mu_/Symbp[$mu],$nu_/Symbp[$nu]] $f[`$mu,$$i2] : \
                                $f[$nu,$$i2]
Gm[$mu_/Symbp[$mu],$nu_/Symbp[$nu]] $f[$$i1,`$mu] :  \
                                $f[$$i1,$nu]
Gm[$mu_/Symbp[$mu],$nu_/Symbp[$nu]] $f[`$mu] : $f[$nu]
Gm[`$mu,`$nu] $f[$$i1,$mu,$$i2] : $f[$$i1,`$nu,$$i2]
KD[$mu_/Symbp[$mu],$nu_/Symbp[$nu]] $f[$$i1,$mu,$$i2] : \
                                $f[$$i1,$nu,$$i2]
```

Component Manipulations:

```
R4_Tensor; R2_Tensor; R_Tensor
_ChrL[Reor]:Sym[2,3]
_GU[Reor]:Sym; _GL[Reor]:Sym
_R2[Reor]:Sym
_R4[Reor]:{Asym[1,2],Asym[3,4]}
```

Find

evaluates the elements of the tensor *x*.

```
Find[$x,$y]::Ap[Ar,{Ap[Ar[$y,$i],{DIM}],$x[0]}]
```

GL

denotes the metric tensor with both indices covariant; (default Minkowskian).

```
GL : Ar[{4,4},0]; GL[4,4]:B[r]; GL[1,1]:-A[r]; GL[2,2]:-r^2
        GL[3,3]:-(r Sin[theta])^2
```

GU

denotes the metric tensor with both indices contravariant.

```
GU :: Minv[GL]
```

```
Pr["1"]
Proc[]
```

ChrL

denotes the Christoffel symbols with all indices covariant.

```
ChrL[$i_=Natp[$i],$j_=Natp[$j],$k_=Natp[$k]] :: ChrL[$i,$j,$k] :\
        1/2 (Dt[GL[$i,$j],CO[$k]] + Dt[GL[$i,$k],CO[$j]]\
        - Dt[GL[$j,$k],CO[$i]])
```

```
Pr["2"]
Proc[]
```

```
Find['ChrL,3]
```

```
Pr["3"]
Proc[]
```

R4

denotes the Reimann curvature tensor.

```
R4[$a_=Natp[$a],$b_=Natp[$b],$c_=Natp[$c],$d_=Natp[$d]]::\
R4[$a,$b,$c,$d]:(Lcl[%z,%mu,%nu,%eta];\
        %z : GL[$a,%mu] Dt[GU[%mu,%nu] ChrL[%nu,$b,$c],CO[$d]]\
        - GL[$a,%mu] Dt[GU[%mu,%nu] ChrL[%nu,$b,$d],CO[$c]]\
        + ChrL[$a,$d,%eta] ChrL[`%eta,$b,$c]\
        - ChrL[$a,$c,%eta] ChrL[`%eta,$b,$d];\
        %z:Comp[%z])
```

```
Pr["4"]
Proc[]
```

```
Find['R4,4]
```

```
Pr["5"]
Proc[]
```

R2

denotes the Ricci tensor.

```
R2[$a_=Natp[$a],$b_=Natp[$b]]::R2[$a,$b]: (Lcl[%z,%mu,%nu];\
        %z : GU[%mu,%nu] R4[%mu,$a,%nu,$b]; Comp[%z])
```

```
Find[R2[,]]
```

R

denotes the contraction of the Ricci tensor.

```
R::R:(Lcl[%z,%mu,%nu]; %z : GU[%mu,%nu] R2[%mu,%nu]; Comp[%z])
R
```

Comp

performs component manipulations.

```
        Comp[$x]::S[$x,{%r1,%r2,%r3,%r4,%r5,%r6,%r7,%r8},Inf]

%r1:    Gm[$a_=Natp[$a],$b_=Natp[$b]]-->GL[$a,$b]

%r2:    Gm[`$a_=Natp[$a],`$b_=Natp[$b]]-->GU[$a,$b]

%r3:    Gm[$$i,``$mu]-->0

%r4:    $f[$$i1,$mu,$$i2] $g[$$j1,`$mu,$$j2] --> \
                (Lcl[%1,%2];\
                If[~P[$mu[0]='Mark] & ~P['$f='GU],Dist[Sum[Sum[ \
                        GU[%1,%2] $f[$$i1,%1,$$i2] $g[$$j1,%2,$$j2]\
                        ,{%2,1,DIM}],{%1,1,DIM}],{Mult,Plus,Plus}]])

%r5:    $f[$$i1,`$mu_=Natp[$mu],$$i2] -->         \
                (Lcl[%1];\
                If[~P['$f='GU],Dist[Sum[\
                        GU[$mu,%1] $f[$$i1,%1,$$i2]\
                        ,{%1,1,DIM}],{Mult,Plus,Plus}]])

%r6:    $f[$$i,```$mu_=Natp[$mu]] --> (Lcl[%1];If[~P['$f='GU],Sum[\
                GU[$mu,%1] $f[$$i,``%1],{%1,1,DIM}])

%r7:    $f[$i,``$mu_=Natp[$mu]] -->      \
                (Lcl[%1,%2];\
                If[~P['$f='GU],Dist[\
                        Dt[$f[$i],CO[$mu]] - Sum[Sum[\
                                GU[%1,%2] ChrL[%1,$i,$mu] $f[%2]\
                        ,{%2,1,DIM}],{%1,1,DIM}],{Mult,Plus,Plus}]])

%r8:    $f[$$i,``$mu_=Natp[$mu]] -->     \
                (Lcl[%1,%2,%l,%z];\
                If[~P['$f='GU],Dist[\
                        Dt[$f[$i],CO[$mu]] - (%l:{$i};Sum[%z:%l;          \
                                %z[0]:'$f;Sum[%z[%1]:`%2;\
                                As[%z] ChrL[%2,%1,$mu]   \
                        ,{%2,1,DIM}],{%1,1,Len[%l]}]),{Mult,Plus,Plus}]])
```

XTerm

Terminal as standard output

T.Shaw
Sep 1981

Open[{,,{,19}}]

XTern

Balanced ternary representation

S.Wolfram and P.Leyland
Jan 1982

Tern[*n*]

generates a balanced ternary representation of the integer n.

```
Tern[$n_=Intp[$n]] :: (Lcl[%tot,%res,%i]; \
        %res:$n + (3^Ceil[N[Log[Abs[$n],3]+2]]-1)/2; \
        For[%i:1, %res~=0, Inc[%i], %tot[%i]:Mod[%res,3]-1; \
        %res:Ceil[%res/3]]; Rev[%tot])
```

[Knuth, vol 2 (2nd ed)]

```
#I[1]::   <XTern

#I[2]::   Tern[12345]

#O[2]:    {1,-1,0,-1,0,-1,1,1,-1,1.12777e^-10}
```

Misfeatures: example shows numerical error

XToTop

Tensor index rotation

S.Wolfram

```
Totop[$t] :: {[$n]:: Ar[Cyc[Ar[$n,Dim[$t]],-1],$t[$$1,$1]]{
```

Rotate tensor index to top

XToTop.new

Tensor index rotation

S.Wolfram

```
Totop[$t] :: {[$n]:: Ar[Cyc[Ar[$n,Dim[$t]],-1],$t[$$1,$1]]}
```

Rotate tensor index to top

XTri

Triangular list generation

group – sequences – cumulative lists

S.Wolfram
Jan 1982

Tri[*list*]
form a triangular *list* from sequences of successive entries of *list*.

```
Tri[$list_=Listp[$list]] :: Ar[Len[$list],Ar[$1,$list]]
```

• UnFlat; XSubl

```
#I[1]::  <XTri
#I[2]::  Ar[5]
#O[2]:   {1,2,3,4,5}
#I[3]::  Tri[%]
#O[3]:   {{1},{1,2},{1,2,3},{1,2,3,4},{1,2,3,4,5}}
```

XTrig

Trigonometric functions

S.Wolfram
Feb 1982

Misfeatures: Sin[17Pi/12] does not simplify completely, perhaps through multiple get confusions

Periodicity relation

```
Sin[($n_=Numbp[$n] & $n>2) Pi] : Sin[Mod[$n,2] Pi]
```

Special values

```
Sin[($n_=Ratp[$n,12] & 1<$n<2) Pi] : -Sin[$n Pi-Pi]

Sin[($n_=Ratp[$n,12] & 1/2 < $n < 1) Pi] : Cos[$n Pi-Pi/2]

Sin[Pi/12] : (Sqrt[3]-1)/(2Sqrt[2])

Sin[Pi/6] : 1/2

Sin[Pi/4] : 1/Sqrt[2]

Sin[Pi/3] : Sqrt[3]/2

Sin[5Pi/12] : (Sqrt[3]+1)/(2Sqrt[2])
```

[CRC p 227]

XTrigR

Further trigonometric functions

versine — haversine

S.Wolfram
Feb 1982

```
Vers[$a] : 1 - Cos[$a]

Hav[$a] : Vers[$a]/2

Exsec[$a] : Sec[$a] - 1

Covers[$a] : 1 - Sin[$a]

Cis[$a] : Cos[$a] + I Sin[$a]
```

[CRC p. 226]

XTup

n-tuples

S.Wolfram
Jul 1981

Tupo[*tot*, *n*]
yields a list of all possible ordered *n*-tuples of *tot* elements.

```
Tupo[$tot_=Natp[$tot],$n_=Natp[$n]] :: Tupo[$tot,$n] : \
Cat[Tupo[$tot-1,$n],Map[Cat[$1,{$tot}],Tupo[$tot-1,$n-1]]]
Tupo[$tot,0] : {{}}
Tupo[$tot,$n_=($n>$tot)] : {}
```

Tup[*tot*, *n*]
yields a list of all possible unordered *n*-tuples of *tot elements*.

```
_Tup[Init] :: <XList0
Tup[$tot_=Natp[$tot],$n_=Natp[$n]] :: Tup[$tot,$n] : \
        Cat[Tup[$tot-1,$n],Flat[Map[Ar[$n,Ins[$tot,$1,$2]], \
                                    Tup[$tot-1,$n-1]],1]]
Tup[$tot,0] : {{}}
Tup[$tot,$n_=($n>$tot)] : {}
```

Tupa[*tot*, *n*]
yields a list of unordered *n*-tuples of *tot* elements, allowing repetitions of an element.

```
Tupa[$tot,$n] :: Flat[Ar[Ar[$n,$tot],List],$n-1]
```

Pairs[*tot*]
yields a list of all possible partitionings of *tot* elements into pairs.

```
_Pairs[Init] :: <XSets
Pairs[$n_=Natp[$n/2]] :: \
        Union[Map[Sort,Trans[{Tupo[$n,$n/2], \
        Map[Cmpl[$1,Ar[$n]],Tupo[$n,$n/2]]},$n/2],2]]
```

```
#I[1]::  <XTup
#I[2]::  Tupo[3,2]
#O[2]:   {{1,2},{1,3},{2,3}}
#I[3]::  Tup[3,2]
#O[3]:   {{2,1},{1,2},{3,1},{1,3},{3,2},{2,3}}
#I[4]::  Tupa[3,2]
#O[4]:   {{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3}}
#I[5]::  Pairs[4]
#O[5]:   {{{1,2},{3,4}},{{1,3},{2,4}},{{1,4},{2,3}}}
```

XTuring

Turing machine simulation

S.Wolfram
Jul 1981

Tape[*i*]
is the *ith* symbol on the data tape. (Tape[$i]:1 defines a tape consisting solely of 1's.)

Spec[*st1*, *symb1*]
is of the form {*symb2*,*st2*} and specifies that when the machine is in state *st1* and reads the symbol *symb1* from the tape, it writes *symb2* in place of *symb1* on the tape, and makes a transition to state *st2*. The machine halts when no applicable Spec exists.

Left
is a special symbol which when read causes the tape to advance under the reading head one symbol to the left.

Right
is a special symbol causing the tape to advance to the right.

The symbol Null represents a blank position on the tape.

Start[*pos*, *st0*]
starts the Turing machine in state *st0* and at position *pos* on the tape; if the operation of the machine terminates, it yields the final position and state.

```
Start[$pos,$st0] :: (Lcl[%pos,%st]; %pos:$pos; %st:$st0; \
        Rpt[Next[Spec[%st,Tape[%pos]]],Inf]; {%pos,%st})
Next[$x_=~Valp[$x]] :: Ret[]
Next[{Left,$st}] :: (%st:$st; Dec[%pos])
Next[{Right,$st}] :: (%st:$st; Inc[%pos])
Next[{$symb,$st}] :: (Tape[%pos]:$symb; %st:$st)
```

```
#I[1]::   <XTuring

#I[2]::   Tape[$i_=$i<5]:1

#O[2]:    1

#I[3]::   Spec[1,0]:{Right,1}

#O[3]:    {Right,1}

#I[4]::   Spec[1,1]:{0,2}

#O[4]:    {0,2}

#I[5]::   Spec[2,0]:{Right,2}

#O[5]:    {Right,2}

#I[6]::   Spec[2,1]:{Right,1}

#O[6]:    {Right,1}
```

```
#I[7]::  Start[1,1]
#O[7]:   {5,1}
#I[8]::  Tape
#O[8]:   {[3]: 0, [1]: 0, [$i_ /; (5 > $i)]: 1}
```

XUnFlat

List unflattening

S.Wolfram
Jul 1981

UnFlat[*list*, *n*]
collects successive sets of *n* entries in *list* into sublists.

```
UnFlat[$list_/Contp[$list],$n_/Natp[$n]] :: \
        Cat[Ar[{{1,Len[$list],$n}},Cat[Ar[$n,$list[$1+$2-1], \
                $1+$2<=Len[$list]+1]]]]
UnFlat[$list_/Contp[$list],$n_/Natp[Len[$list]/$n]] :: \
        Cat[Ar[{{1,Len[$list],$n}},Ar[$n,$list[$1+$2-1]]]]
```

```
#I[1]::  <XUnFlat

#I[2]::  t:Ar[10]

#O[2]:   {1,2,3,4,5,6,7,8,9,10}

#I[3]::  UnFlat[t,2]

#O[3]:   {{1,2},{3,4},{5,6},{7,8},{9,10}}

#I[4]::  UnFlat[t,3]

#O[4]:   {{1,2,3},{4,5,6},{7,8,9},{10}}
```

XUnmark

Mark removal

S.Wolfram
Jul 1981

Unmark[*expr*]
removes all marks in *expr*.

```
Unmark[$expr] :: S[$expr,`$->$]
```

XVecan

Three-dimensional vector analysis

transformation of coordinate systems — orthogonal coordinates
curvilinear coordinates — separation of variables — partial differential equations

S.Wolfram

Feb 1982

CO: list of orthonormal coordinates
SF: list of scale factors associated with each coordinate
(diagonal components of metric).

Vecp[*v*]

yields 1 if *v* is a contiguous list of three elements, representing a 3-dimensional vector, and 0 otherwise.

```
Vecp[$v] :: Contp[$v] & Len[$v]=3
```

VAbs[*vec*]

yields the norm (modulus) of the vector *vec*.

```
VAbs[$list_Contp[$list]] :: Sqrt[Ap[Plus,$list^2]]
```

Default Cartesian coordinate system:

```
CO : {x,y,z}
SF : {1,1,1}
```

Assignments for other coordinate systems:

Cartesian coordinates

```
CAR :: {CO:{x,y,z},SF:{1,1,1}}
```

Circular cylindrical coordinates

```
CYL :: {CO:{r,phi,z},SF:{1,r,1}}
```

Spherical coordinates

```
SPH :: {CO:{r,th,phi},SF:{1,r,r Sin[th]}}
```

Parabolic coordinates

```
PAR :: {CO:{xi,eta,phi},SF:{Sqrt[xi^2+eta^2],Sqrt[xi^2+eta^2],xi eta}}
```

Parabolic cylinder coordinates

```
PARCYL :: {CO:{xi,eta,z},SF:{Sqrt[xi^2+eta^2],Sqrt[xi^2+eta^2],1}}
```

Elliptic cylinder coordinates

```
ELLCYL :: {CO:{xi,eta,z},SF[1]:SF[2]:c Sqrt[Sinh[xi]^2+Sin[eta]^2]; \
                          SF[3]:1; SF}
```

Prolate ellipsoidal coordinates

```
ELLPRO :: {CO:{xi,eta,phi},SF:{c Sqrt[xi^2-eta^2]/Sqrt[xi^2-1], \
         c Sqrt[xi^2-eta^2]/Sqrt[1-eta^2], c Sqrt[1-eta^2] Sqrt[xi^2-1]}}
```

Oblate ellipsoidal coordinates

```
ELLOBL :: {CO:{xi,eta,phi},SF:{c Sqrt[xi^2+eta^2]/Sqrt[1+xi^2], \
         c Sqrt[xi^2+eta^2]/Sqrt[1-eta^2], c Sqrt[1+xi^2] Sqrt[1-eta^2]}}
```

Toroidal coordinates

```
TOR :: {CO:{xi,eta,phi},SF[1]:SF[2]:c/(Cosh[xi]-Cos[eta]); \
       SF[3]:c Sinh[xi]/(Cosh[xi]-Cos[eta]); SF}
```

Elliptic coordinates

```
ELL :: {CO:{lam,mu,nu}, ellf[$lam]:(a^2+$lam)(b^2+$lam)(c^2+$lam); \
       SF:{Sqrt[(lam-mu)(lam-nu)]/(2Sqrt[ellf[lam]]), \
           Sqrt[(mu-lam)(mu-nu)]/(2Sqrt[ellf[mu]]), \
           Sqrt[(nu-lam)(nu-mu)]/(2Sqrt[ellf[nu]])}}
```

Bipolar coordinates

```
BIP :: {CO:{xi,eta,z}, SF[1]:SF[2]:c/(Cosh[xi]-Cos[eta]); SF[3]:1; SF}
```

Arc

arc length ds^2 with respect to coordinates CO.

```
Arc :: Sum[SF[%1]^2 Dt[CO[%1]]^2,{%1,1,3}]
```

Vol

volume element dV in coordinates CO.

```
Vol :: Prod[SF[%1] Dt[CO[%1]],{%1,1,3}]
```

Grad[*f*]

gradient of scalar expression ***f*** with respect to coordinates CO.

```
Grad[$f] :: Ar[3,D[$f,CO[$1]]/SF[$1]]
```

Dvg[*f*]

divergence of vector (list) ***f*** with respect to CO.

```
Dvg[$f_=Vecp[$f]] :: Sum[D[$f[%1] (SF[1] SF[2] SF[3])/SF[%1]^2, \
                      CO[%1]], {%1, 1, 3}]/(SF[1] SF[2] SF[3])
```

Curl[*f*]

curl of vector expression ***f*** with respect to CO.

```
Curl[$f_=Vecp[$f]] :: \
        {(D[SF[3] $f[3], CO[2]] - D[SF[2] $f[2], CO[3]])/(SF[2]SF[3]),\
         (D[SF[1] $f[1], CO[3]] - D[SF[3] $f[3], CO[1]])/(SF[3]SF[1]),\
         (D[SF[2] $f[2], CO[1]] - D[SF[1] $f[1], CO[2]])/(SF[1]SF[2])}
```

Lap[*f*]

Laplacian of scalar expression ***f*** with respect to CO.

```
        Lap[$f] :: Sum[D[(SF['] SF[2] SF[3])/SF[%1]^2 D[$f,CO[%1]], \
                      CO[%1]], {%1, 1, 3}]/(SF[1] SF[2] SF[3])

#I[1]::   <XVecan

#I[2]::   t:{x^2+a y^2,x y^3,x y z}

               2     2     3
#O[2]:    {a y  + x  ,x y  ,x y z}

#I[3]::   Dvg[%]

                          2
#O[3]:    2x + x y + 3x y

#I[4]::   Grad[%]

                         2
#O[4]:    {2 + y + 3 y  ,x + 6x y,0}

#I[5]::   Vol

#O[5]:    Dt[x] Dt[y] Dt[z]

#I[6]::   SPH

#O[6]:    {{r,th,phi},{1,r,r Sin[th]}}

#I[7]::   Vol

           2
#O[7]:    r  Dt[phi] Dt[r] Dt[th] Sin[th]

#I[8]::   r^2 Sin[th]^3 Cos[phi]

           2                   3
#O[8]:    r  Cos[phi] Sin[th]

#I[9]::   Lap[%]

             2                   2      2                   4
          - r  Cos[phi] Sin[th]  + 3 r  Cos[phi] Sin[th]

                              2                  2          2
                       + 9 r  Cos[phi] Cos[th]  Sin[th]
#O[9]:    ------------------------------------------------
                                 2
                                r  Sin[th]

#I[10]::   Ex[%]

                                3                                                    2
#O[10]:    3Cos[phi] Sin[th]  - Cos[phi] Sin[th] + 9Cos[phi] Cos[th]  Sin[th]
```

Scf[*cart*]

computes the scale factors SF for the coordinates CO from the list *cart* of values for the Cartesian coordinates x,y,z in terms of the curvilinear coordinates CO[1],CO[2],CO[3].

```
        Scf[$cart] :: (SF:Ar[3,VAbs[D[$cart,CO[$1]]]])
```

[e.g. MOS chap. 12; Morse & Feshbach]

XWarn

Warning messages

S.Wolfram

Jul 1981

```
$x/0_=Prh[$x/0,"generated"] :: $x/0
0^$x_=(If[$x<0,Prh[0^$x,"generated"]]):: 0^$x
Log[0_=Prh["Log[0] generated"]] :: Log[0]
Gamma[$n_=If[Natp[-$n],Prh[Gamma[$n],"generated"]]] :: Gamma[$n]

<$f_=Prh[$f,"file unavailable"] :: <$f

($a :: ($b_=Prh[$a::$b,"impossible assignment"])) :: ($a :: $b)
($a : ($b_=Prh[$a:$b,"impossible assignment"])) :: ($a : $b)

Minv[$m_=If[Listp[$m],Prh[$m,"inversion impossible"]]] :: Minv[$m]


#I[1]::   <XWarn

#I[2]::   2/0

2
-         generated
0

          2
#O[2]:    -
          0
```

XWatch

External file input tracing

S.Wolfram
Jul 1981

Watch[*file*]

inputs the specified external *file*, printing each assignment in the file before it is made.

```
Watch[$file] :: (Set_:Setd_:Trace; Lcl[%r]; %r:<$file; Close[]; \
                    _Set[Trace]:_Setd[Trace]:; Open[]; %r)
```

XWhi

Whittaker function

S.Wolfram
Jul 1981

```
SWhi _: Ldist

SWhi[1]:   WhiM[$l,$m,$z] -> Exp[-$z/2]$z^(1/2+$m)WhiM[1/2+$m-$l,1+2$m,$z]

SWhi[2]:   WhiW[$l,$m,$z] -> Gamma[-2$m]/Gamma[1/2-$m-$l]WhiM[$z]\
                     +Gamma[2$m]/Gamma[1/2+$m-$l]WhiM[$l,-$m,$z]

SWhi[3]:   WhiM[$k,$m,$z] -> Exp[-$z/2]$z^($m+1/2)Chg[1/2+$m-$k,1+2$m,$z]

SWhi[4]:   WhiW[$l,$m,$z] -> Exp[-$z/2]$z^(1/2+$m)KumU[1/2+$m-$l,1+2$m,$z]
```

for special cases, see XChg and XKumU.

XWron

Wronskian and Jacobian

S.Wolfram
Jul 1981

Wron[{*y1*, *y2*, ...}, *x*]
forms the Wronskian of {*y1*,*y2*,...} with respect to *x*, which must vanish if the *yi* are linearly dependent.

```
Wron[$list,$x] :: Det[Ar[Len[$list],D[$list,{$x,$1-1}]]]
```

Jac[{*x1*, *x2*, ...}, {*u1*, *u2*, ...}]
forms the Jacobian of the transformation from {*x1*,*x2*,...} to {*u1*,*u2*,...}.

```
Jac[$x,$u_=Len[$u]=Len[$x]] :: Det[Ar[Len[$x],D[$x,$u[$1]]]]
```

```
#I[1]::   <XWron

#I[2]::   Wron[{x^2,Exp[x]},x]

                         2
#O[2]:    -2x Exp[x] + x  Exp[x]

#I[3]::   Jac[{x^2+y^2,x(y-1)},{x,y}]

                             2
#O[3]:    -2y (-1 + y) + 2 x
```

XYear

Day numbers

S.Wolfram
Jul 1981

Year[*month, day*]
gives the number of each day in a non-leap year

```
Month : {[Jan]:31, [Feb]:28, [Mar]:31, [Apr]:30, [May]:31, [Jun]:30, \
                [Jul]:31, [Aug]:31, [Sep]:30, [Oct]:31, [Nov]:30, [Dec]:31}
Year : (Lcl[%t];%t:0;Ar[{{{Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec}}},\
                %t+Ar[Inc[%t,Month[$%1]]]])

#I[1]::  Year[Aug,29]

#O[1]:   241
```

XYoung

Young graphs

S.Wolfram
Jul 1981

{i1,i2,...} represents a Young graph with i1 boxes in the top row, i2 in the second, ...

PDim[*list*]
: yields the dimensionality of the representation of the symmetric group corresponding to the Young graph specified by *list*.

```
PDim[$list] :: Ap[Plus,$list]!*Prod[Prod[$list[i]-$list[j]+j-i, \
                 {j,i+1,Len[$list]}],{i,1,Len[$list]}] /  \
                 Prod[($list[i]+Len[$list]-i)!,{i,1,Len[$list]}]
```

YGMult[*yg1*, *yg2*]
: yields a list of Young graphs occuring in the product of the representations of the symmetric group corresponding to *yg1* and *yg2*.

```
YGMult[$l1,$l2] :: (Lcl[%l1,%l2]; \
%l1:{Ar[Len[$l1],Ar[$l1[$1],0]]}; \
%l2:Flat[Ar[Len[$l2],Ar[$l2[$1],$1]]]; \
Do[%i,Len[%l2],%l1:Flat[Map[YGadd[$%1,%l2[%i]],%l1],1]; \
%l1:Cat[Ar[Len[%l1],%l1[$1],,YGt0[$1]&YGt1[$1,%l2[%i]]& \
                 YGt2[$1,%l2[%i]]]]]]; Map[Len,%l1,{2}])

YGadd[$list,$n] :: \
Cat[Ar[Len[$list],YGadd1[$list,$1,$n]],{Cat[$list,{{$n}}]}]
YGadd1[$list,$pos,$n] :: \
Cat[Ar[$pos + -1,$list],{Cat[$list[$pos],{$n}]},\
  Ar[{{$pos + 1,Len[$list]}},$list]]
```

Test for legal graphs

```
YGt0[$list] :: Ap[Ge,Map[Len,$list]]
YGt1[$list,$n]::Ap[Uneq,Map[$1[2],Pos[$n,$list]]]
YGt2[$list,$n] :: YGt2p[Del[0,Flat[Map[Rev,$list]]],$n]
YGt2p[$list,$n] :: \
(Lcl[%t,%r]; %r:Ar[$n,0]; For[%t:1;%c:1,%c<=Len[$list]&Ap[Ge,%r], \
Inc[%c],%l[$list[%c]]:%l[$list[%c]]+1]; If[%c>Len[$list],1,0])
YGt3[$list,$sun] :: $sun >= Len[$list]
```

```
#I[1]::  <XYoung

#I[2]::  YGMult[{1,1},{1,1}]

#O[2]:   {{2,2},{2,1,1},{2,1,1},{1,1,1,1}}

#I[3]::  Map[PDim,%]

#O[3]:   {3,2,3,3,1}

#I[4]::  PDim[{5,3,2,2,1}]

#O[4]:   21450
```

XZeta2S

Generalized Riemann zeta function

reflection formula – Rademacher's formula

S.Wolfram
Feb 1982

```
SZeta[2,1] : Zeta[$z,$r_=Ratp[$r]] --> \
        2 Gamma[1-$z] (2Pi Den[$r])^($z-1) (Sin[Pi $z/2] \
        Sum[Cos[2 Pi $r %n] Zeta[1-$z,%n/Den[$r]],{%n,1,Den[$r]}] + \
        Cos[Pi $z/2] Sum[Sin[2 Pi $r %n] Zeta[1-$z,%n/Den[$r]],{%n,1,Den[$r]}])
```

[MOS sect. 1.4]

XZeta2V

Generalized Riemann zeta function

S.Wolfram
Feb 1982

Definition of special values

```
Zeta[$s,1] : Zeta[$s]

Zeta[$s,1/2] : Zeta[$s]/(2^$s-1)

Zeta[0,$a] : 1/2-$a

Zeta[$m_=Natp[$m+1],$a] : -Ber[-$m+1,$a]/(-$m+1)
```

Derivatives

```
D[Zeta[$s,$a],{$s,1,0}] : Log[Gamma[$a]] - Log[2Pi]/2

D[Zeta[$s,$a],{$a,1,$b}] : -$s Zeta[$s+1,$b]
```

[MOS sect. 1.4]

[redacted]

XZetaR

Functions related to Riemann zeta function

sums of reciprocal powers

S.Wolfram
Feb 1982

• Catb; Li

```
Eta[$n] : (1-2^(1-$n)) Zeta[$n]

Eta[2] : Pi^2/12

Eta[4] : 7Pi^4/728
```

[AS sect. 23.2]

```
Lambda[$n] : (1-2^-$n) Zeta[$n]

Lambda[2] : Pi^2/8

Lambda[4] : Pi^4/96
```

[GR sect. 9.56]

```
Xi[$s] : $s($s-1) Gamma[$s/2]/(2 Pi^($s/2)) Zeta[$s]
```

[redacted]

INDEX

SMP LIBRARY / INDEX

SMP LIBRARY / INDEX

SMP LIBRARY / INDEX

INDEX