New gfun New Gfun 7400 5/94

From inria.fr!Bruno.Salvy Mon May 9 15:33:50 +0200 1994
Received: by ninet.research.att.com; Mon May 9 09:34 EDT 1994
Received: from rully (rully.inria.fr [128.93.8.64]) by concorde.inria.fr (8.6.9/8.6.9) with ESI
Received: (from salvy@localhost) by rully (8.6.8/8.6.6) id PAA20171; Mon, 9 May 1994 15:33:50 ·
Date: Mon, 9 May 1994 15:33:50 +0200
Message-Id: <199405091333.PAA20171@rully>
From: Bruno.Salvy@inria.fr (Bruno Salvy)
To: njas@research.att.com
CC: plouffe@geocub.greco-prog.fr
In-reply-to: <199405091010.MAA03076@concorde.inria.fr> (njas@research.att.com)
Subject: gfun2.01
Status: RO

Here is a slightly improved version.

The output of ratpoly is now checked before returning from listtoratpoly or seriestoratpoly. No more incorrect result should arise this way.

The output of dsolve in guessgf is now checked for undetermined coefficients, which are then solved for using the elements of the list.

The help file for listtoseries, seriestolist, listtolist and seriestoseries has been made clearer.

To install this new version, save the last part of this mail in a file called gfun2.01, edit the lines

```
macro(MAPLE5.3  = true);
macro(MAPLE5.2  = false);
```

to match the version of Maple you're using, then run

```
maple -s < gfun2.01
```

it will create a file gfun.m which you can use as usual.

Bruno.

```
-------------------------- cut here --------------------------
##    -*-Maple-*-
##    Title:     gfun package (for Generating FUNctions)
##    Created:   Wed Mar  4 15:13:42 1992
##    Authors:   Bruno Salvy <Bruno.Salvy@inria.fr>
##               Paul Zimmermann <Paul.Zimmermann@inria.fr>
##
## Description: converts implicit equations into differential equations,
##              differential equations into recurrences and vice-versa,
##              ordinary into exponential recurrences.
##              also converts lists to linear recurrences or differential
##              equations.
##
##  Some of the ideas used in this file are due to S. Plouffe and F. Bergeron.
##
## Modifications:
```

```
##   formatdiffeq, formatrec added.                                  B.S. Dec 92
##   rectoproc, diffeqtorec, rectodiffeq, rectoproc,
     borel, invborel rewritten.                                      B.S. Dec 92
##   2.0 new version                                                 B.S. May-Nov 93
##   2.01 a few extra checks, clarified help for listtolist          B.S. May 94

#######################################################################
## CONVENTION: a recurrence is an expression of the form
##      sum(p[i]*u(n+i),i=0..d)                    (E)
## or
##    { sum(p[i]*u(n+i),i=0..d), u(0)=a0,..., u(k)=ak }.
##
## The p[i]'s are polynomials in n. The sequence(s) represented by such
## a recurrence are solutions of (E) for n>=k, where k is the largest
## positive integer solution to p[d](n-d)=0, or 0 if this does not cancel.
## For n<k, the values of the sequence are given by the initial conditions,
## and it is part of the convention that u(i)=0 for i<0.
#######################################################################

macro(MAPLE5.3  = true);
macro(MAPLE5.2  = false);
macro(
algebraicsubs   =  gfun[algebraicsubs],
algeqtodiffeq   =  gfun[algeqtodiffeq],
cauchyproduct   =  gfun[cauchyproduct],
cheapgausselim  = `gfun/gausselim`,
`diffeq+diffeq` =  gfun[`diffeq+diffeq`],
`diffeq*diffeq` =  gfun[`diffeq*diffeq`],
diffeqtorec     =  gfun[diffeqtorec],
firstnonzero    = `gfun/firstnonzero`,
formatdiffeq    = `gfun/formatdiffeq`,
formatpoleq     = `gfun/formatpoleq`,
formatrec       = `gfun/formatrec`,
gensolvelin     = `gfun/gensolvelin`,
`goodinitvalues/diffeq` = `gfun/goodinitvalues/diffeq`,
`goodinitvalues/rec` = `gfun/goodinitvalues/rec`,
guesseqn        =  gfun[guesseqn],
guessgf         =  gfun[guessgf],
hadamardproduct =  gfun[hadamardproduct],
inicond         = `gfun/inicond`,
lindep          = `gfun/lindep`,
listprimpart    = `gfun/listprimpart`,
listtoalgeq     =  gfun[listtoalgeq],
listtodiffeq    =  gfun[listtodiffeq],
listtohypergeom =  gfun[listtohypergeom],
listtolist      =  gfun[listtolist],
listtoratpoly   =  gfun[listtoratpoly],
listtorec       =  gfun[listtorec],
listtoseries    =  gfun[listtoseries],
`l2r/l2r`       = `gfun/listtorec/listtorec`,
`l2h/l2h`       = `gfun/listtohypergeom/listtohypergeom`,
makediffeq      = `gfun/makediffeq`,
makediffeqdiff  = `gfun/makediffeqdiff`,
makerec         = `gfun/makerec`,
maxdegcoeff     =  gfun['maxdegcoeff'],
maxdegeqn       =  gfun['maxdegeqn'],
```

```
maxindex          = `gfun/maxindex`,
maxordereqn       =  gfun['maxordereqn'],
mindegcoeff       =  gfun['mindegcoeff'],
mindegeqn         =  gfun['mindegeqn'],
minindex          = `gfun/minindex`,
minordereqn       =  gfun['minordereqn'],
mygcdex           = `gfun/gcdex`,
myisolve          = `gfun/isolve`,
optionsgf         =  gfun['optionsgf'],
ratpolytocoeff    =  gfun[ratpolytocoeff],
`ratpolytocoeff/elmt`=`gfun/ratpolytocoeff/elmt`,
ratsolvelin       = `gfun/ratsolvelin`,
`rec+rec`         =  gfun[`rec+rec`],
`rec*rec`         =  gfun[`rec*rec`],
rectodiffeq       =  gfun[rectodiffeq],
rectoproc         =  gfun[rectoproc],
remove            = `gfun/remove`,
`s2d/s2d`         = `gfun/seriestodiffeq/seriestodiffeq`,
`s2a/s2a`         = `gfun/seriestoalgeq/seriestoalgeq`,
seriestoalgeq     =  gfun[seriestoalgeq],
seriestodiffeq    =  gfun[seriestodiffeq],
seriestohypergeom =gfun[seriestohypergeom],
seriestolist      =  gfun[seriestolist],
seriestoratpoly   =  gfun[seriestoratpoly],
seriestorec       =  gfun[seriestorec],
seriestoseries    =  gfun[seriestoseries],
systomatrix       = `gfun/systomatrix`,
typecheck         = `gfun/typecheck`
);

######################## Global Constants ##############################

`gfun/version`:=2.01:
optionsgf:=['ogf','egf']:
maxordereqn:=2: # default 2nd order
minordereqn:=1: # default 1st order
maxdegcoeff:=3: # default degree 3 coefficients
mindegcoeff:=0: # default constant coefficients
maxdegeqn:=6:   # default maximum 6th degree
mindegeqn:=2:   # default minimum 2nd degree

######################## Type Checking ##############################

`type/gfun/identity`:=proc(x) type(x,`=`) and op(1,x)=op(2,x) end:
`type/gfun/free`:=proc(x,y) not has(x,y) end:

# type(y(0),initeq(y))            -> true
# type(D(y)(0),initeq(y))         -> true
# type((D@@k)(y)(0),initeq(y))    -> true
# otherwise                       -> false
`type/gfun/initeq` := proc(expr,y)
local f;
    if not type(expr,function(0)) then RETURN(false) fi;
    f := op(0,expr);
    RETURN(f=y or f='D(y)' or (type(f,function(identical(y)))) and
            type(op(0,f),`@@`(identical(D),integer)))
```

```
end: # `type/gfun/initeq`

This is to avoid several type checks of the same expression
typecheck:=proc (n)
local i;
    if n=1 then          # 1, x, <met>
        if nargs>2 and type([args[2..3]],[list,name]) then
            if nargs=3 then RETURN('stamped',args[2..3],op(1,optionsgf))
            elif nargs=4 and type(gfun[cat(`listtoseries/`,args[4])],procedure)
                then RETURN('stamped',args[2..4])
            elif nargs>4 then ERROR(`too many arguments`)
            else ERROR(`invalid argument`,args[4])
            fi
        elif nargs<3 then ERROR(`too few parameters`)
        elif not type(args[2],list) then ERROR(`not a list`,args[2])
        elif not type(args[3],name) then ERROR(`not a name`,args[3])
        else ERROR(`invalid arguments`)
        fi
    elif n=2 then        # 1, y(x), <[met]>
        if nargs>2 and type([args[2..3]],[list,function(name)]) then
            if nargs=3 then RETURN('stamped',args[2..3],optionsgf)
            elif nargs=4 and type(args[4],list) and
                type([seq(gfun[cat(`listtoseries/`,i)],i=args[4])],
                    list(procedure)) then RETURN('stamped',args[2..4])
            elif nargs>4 then ERROR(`too many arguments`)
            else ERROR(`invalid argument`,args[4])
            fi
        elif nargs<3 then ERROR(`too few arguments`)
        elif not type(args[2],list) then ERROR(`not a list`,args[2])
        elif not type(args[3],function(name)) then
            ERROR(`invalid unknown function`,args[3])
        else ERROR(`invalid arguments`)
        fi
    elif n=3 then        # 1, x, [<met>]
        if nargs>2 and type([args[2..3]],[list,name]) then
            if nargs=3 then RETURN('stamped',args[2..3],optionsgf)
            elif nargs=4 and type(args[4],list) and
                type([seq(gfun[cat(`listtoseries/`,i)],i=args[4])],
                    list(procedure)) then RETURN('stamped',args[2..4])
            elif nargs>4 then ERROR(`too many arguments`)
            elif nargs=4 then ERROR(`invalid argument`,args[4])
            fi
        elif nargs<3 then ERROR(`too few parameters`)
        elif not type(args[2],list) then ERROR(`not a list`,args[2])
        elif not type(args[3],name) then ERROR(`not a name`,args[3])
        else ERROR(`invalid argments`)
        fi
    elif n=4 then        # s, x, y
        if nargs=4 and type([args[2..4]],[series,name,name]) then
            RETURN('stamped',args[2..4])
        elif nargs<>4 then ERROR(`wrong number of arguments`)
        elif not type(args[2],series) then ERROR(`not a series`,args[2])
        elif not type(args[3],name) then ERROR(`not a name`,args[3])
        elif not type(args[4],name) then ERROR(`not a name`,args[4])
        else ERROR(`invalid arguments`)
        fi
```

```
elif n=5 then        # l, y(x)
    if nargs=3 and type([args[2..3]],[list,function(name)]) then
        RETURN('stamped',args[2..3])
    elif nargs<>3 then ERROR(`wrong number of arguments`)
    elif not type(args[2],list) then ERROR(`not a list`,args[2])
    elif not type(args[3],function(name)) then
        ERROR(`invalid unknown function`,args[3])
    else ERROR(`invalid arguments`)
    fi
elif n=6 then        # s, y(x), <[met]>
    if nargs>2 and type([args[2..3]],[series,function(name)]) then
        if nargs=3 then RETURN('stamped',args[2..3],optionsgf)
        elif nargs=4 and type(args[4],list) and
            type([seq(gfun[cat(`listtoseries/`,i)],i=args[4])],
                list(procedure)) then RETURN('stamped',args[2..4])
        elif nargs>4 then ERROR(`too many arguments`)
        else ERROR(`invalid argument`,args[4])
        fi
    elif nargs<3 then ERROR(`too few arguments`)
    elif not type(args[2],series) then ERROR(`not a series`,args[2])
    elif not type(args[3],function(name)) then
        ERROR(`invalid unknown function`,args[3])
    else ERROR(`invalid arguments`)
    fi
elif n=7 then        # s, <[met]>
    if nargs>1 and type(args[2],series) then
        if nargs=2 then RETURN('stamped',args[2],optionsgf)
        elif nargs=3 and type(args[3],list) and
            type([seq(gfun[cat(`listtoseries/`,i)],i=args[3])],
                list(procedure)) then RETURN('stamped',args[2..3])
        elif nargs>3 then ERROR(`too many arguments`)
        else ERROR(`invalid argument`,args[3])
        fi
    elif nargs=1 then ERROR(`too few arguments`)
    elif not type(args[2],series) then ERROR(`not a series`,args[2])
    else ERROR(`invalid arguments`)
    fi
elif n=8 then        # s, <met>
    if nargs>1 and type(args[2],series) then
        if nargs=2 then RETURN('stamped',args[2],'ogf')
        elif nargs=3 and type(gfun[cat(`listtoseries/`,args[3])],procedure)
            then RETURN('stamped',args[2..3])
        elif nargs>3 then ERROR(`too many arguments`)
        else ERROR(`invalid argument`,args[3])
        fi
    elif nargs=1 then ERROR(`too few arguments`)
    elif not type(args[2],series) then ERROR(`not a series`,args[2])
    else ERROR(`invalid arguments`)
    fi
elif n=9 then        # l, <met>
    if nargs>1 and type(args[2],list) then
        if nargs=2 then RETURN('stamped',args[2],'ogf')
        elif nargs=3 and type(gfun[cat(`listtoseries/`,args[3])],procedure)
            then RETURN('stamped',args[2..3])
        elif nargs>3 then ERROR(`too many arguments`)
        else ERROR(`invalid argument`,args[3])
```

```
            fi
        elif nargs=1 then ERROR(`too few arguments`)
        elif not type(args[2],list) then ERROR(`not a list`,args[2])
        else ERROR(`invalid arguments`)
        fi
    elif n=10 then      # recurrence, function(name), function(name)
        if nargs>4 then ERROR(`too many arguments`)
        elif nargs<4 then ERROR(`too few arguments`)
        elif not type([args[3..4]],[function(name),function(name)]) then
            ERROR(`invalid arguments`)
        fi
    else ERROR(`should not happen`)
    fi;
end: # typecheck

################## Modifications of existing Maple code ############

## Adapted from solve/linear/integer
ratsolvelin := proc (equations, unknowns)
local eqn, eqns, vars, sol, sols, i, k, l, pivot, var, c, j;
    # Suppress trivial equations and
    # normalize: convert from Q to Z and divide out by the content
    eqns := {seq(l/icontent(l)*sign(l), l=  equations minus {0})};
    # Solve the remaining system using a sparse implementation
    for k while eqns <> {} do
        l:=map(length,[op(eqns)]);
        member(min(op(l)),l,'i');
        eqn := eqns[i];
        vars := indets(eqn) intersect unknowns;
        var := vars[1];
        pivot := coeff(eqn,var,1);
        for j from 2 to nops(vars) do
            # Eliminate the unknown with the smallest coefficient
            c := coeff(eqn,vars[j],1);
            if length(c) < length(pivot) then pivot := c; var := vars[j] fi
        od;
        eqns:=subs(var=-1/pivot*subs(var=0,eqn),subsop(i=NULL,eqns)) minus {0};
        sol[k]:=var,eqn,pivot
    od;
    sols := {};j:=0;
    for i from k-1 by -1 to 1 do
        eqn[i] := - subs( sol[i][1] = 0, sols, sol[i][2] );
        if eqn[i]<>0 then j:=j+1;l:=subsop(j=i,l) fi;
        sols := sols union {sol[i][1] = eqn[i] / sol[i][3]}
    od;
    # Parameterize an infinite solution space
    vars := {op(unknowns)} minus {seq(op(1,i),i=sols)};
    if vars={} then RETURN(sols)
    elif nops(vars)=1 then RETURN({op(subs(op(vars)=1,sols)),op(vars)=1})
    elif select(type,{seq(op(2,i),i=sols)},rational) minus {0} <> {} then
        RETURN({op(subs(seq(i=0,i=vars),sols)),seq(i=0,i=vars)})
    else RETURN(FAIL) # we do not want to treat this
    fi
end: # ratsolvelin

# A simple interface to solve/linear
```

```
gensolvelin:=proc ()
local s, t;
    s:=readlib(`solve/linear`)(args);
    t:=select(type,s,`gfun/identity`);
    if nops(t)<>1 then RETURN(FAIL) fi;
    RETURN(subs(op(1,op(t))=1,(1=1)=(op(1,op(t))=1),s))
end: # gensolvelin


# this procedure does not do exactly the same as gcdex, but is much faster
# it returns g' and assign s,t such that s*a+t*b=g'
# and g'=lambda gcd(a,b) with lambda a rational fraction independent of y
#################################################################
##########  ONE CAN PROBABLY IMPROVE IT MORE   ####################
#################################################################
mygcdex := proc(a,b,y,s,t)
local q,r,u,v,du,dv,alpha,beta,oldalpha,oldbeta,dr,lr;
    u:=collect(a,y); v:=collect(b,y);
    if u=0 then RETURN(b) elif v=0 then RETURN(a) fi;
    oldalpha:=1; oldbeta:=0;
    alpha:=0; beta:=1;
    # u = oldalpha*a + oldbeta*b
    # v = alpha*a + beta*b
    du:=degree(u,y);
    dv:=degree(v,y);
    do
        if du<dv then
            dv; dv:=du; du:="";
            v; v:=u; u:="";
            alpha; alpha:=oldalpha; oldalpha:="";
            beta; beta:=oldbeta; oldbeta:="";
        fi;
        userinfo(3,'gfun',`degrees of the polynomials`,du,dv);
        userinfo(5,'gfun','polynomials',u,v);
        q:=lcoeff(u,y)/lcoeff(v,y)*y^(du-dv);
        r:=u-collect(q*v,y);
        do
            dr:=degree(r,y);
            lr:=coeff(r,y,dr);
            if normal(lr)=0 then
                r:=subs(y^dr=0,r);
                if degree(r,y)=dr then break fi
            else break
            fi
        od;
        if r=0 then break fi;
        oldalpha-q*alpha; oldalpha:=alpha; alpha:="";
        oldbeta-q*beta; oldbeta:=beta; beta:="";
        u:=v;du:=dv;
        v:=r;dv:=degree(v,y);
    od;
    if dv=0 then
        if nargs=4 then s:=rem(alpha/v,b,y) fi;
        if nargs=5 then t:=rem(beta/v,a,y) fi;
        RETURN(1)
    else
```

```
        if nargs=4 then s:=rem(alpha,b,y) fi;
        if nargs=5 then t:=rem(beta,a,y) fi;
        RETURN(v)
    fi;
end: # mygcdex

# This is really needed because MapleV's isolve is terrible
# eqn is a polynomial in n
myisolve:=proc (eqn, n)
local sol, i, d, f;
    if not has(eqn,n) then RETURN({}) fi;
    # These two lines are due to M. Monagan:
    if type(eqn,polynom(rational,n)) then
        RETURN(select(type,{seq(i[1],i=roots(eqn))},integer)) fi;
    f:=collect(eqn,n);
    d:=degree(f,n);
    if d=1 then
        f:=-coeff(f,n,0)/coeff(f,n,1);
        if type(f,integer) then RETURN({f})
        elif not has(f,RootOf) and not has(f,`&RootOf`) then RETURN({})
        else f:=myevala(Normal(f));
            if type(f,integer) then RETURN({f}) else RETURN({}) fi
        fi;
    elif d=2 then
        f:=[seq(coeff(f,n,i),i=0..2)];
        d:=sqrt(op(2,f)^2-4*op(3,f)*op(1,f));
        RETURN(select(type,map(normal,
                {(d-op(2,f))/2/op(3,f),-(d+op(2,f))/2/op(3,f)}),integer))
    elif type(eqn,`*`) then RETURN(`union`(op(map(myisolve,[op(eqn)],n))))
    else
        sol:=traperror(isolve(expand(eqn),n));
        if sol=lasterror then RETURN({}) fi;
        RETURN({seq(op(2,i),
            i=select(type,map(op,[sol]),identical(n)=integer))})
    fi
end: # myisolve

######################### Various Utilities ###################

firstnonzero:=proc(pol,n) RETURN(max(-1,op(myisolve(pol,n)))+1) end:

# returns the smallest i such that u(n+i) appears in a recurrence
minindex := proc(rec,u,n)
    RETURN(min(op(map(op,indets(rec,u(linear(n))))))-n)
end: # minindex

# returns the largest i such that u(n+i) appears in a recurrence
maxindex := proc(rec,u,n)
    RETURN(max(op(map(op,indets(rec,u(linear(n))))))-n)
end: # maxindex

# the opposite of select
if not MAPLE5.3 then
remove:=subs(proc(x) if _func(args) then x else _identity fi end =
proc(x) if not _func(args) then x else _identity fi end, op(select)):
else
```

```
remove:=proc(f,a) map(subs(_f=f,
    proc(x) if not _f(args) then x else NULL fi end),
    args[2..nargs]) end fi:

# systomatrix
#  Input: a system of homogeneous linear equations and a list of variables V,
#         and the name of a vector B.
#  Output: a matrix A such that the system is equivalent to A.V=B.
# Almost like linalg[genmatrix], but more suitable for our purpose.
systomatrix:=proc (sys, V, B)
local a, i, j, eqn, zero;
    if sys={} then
        B:=array(1..1,[0]);
        RETURN(array(1..1,1..nops(V),[[0$nops(V)]])) fi;
    zero:=[seq(i=0,i=V)];
    a:=array(1..nops(sys),1..nops(V),sparse);
    B:=array(1..nops(sys));
    for i to nops(sys) do
        eqn:=op(i,sys);
        if type(eqn,`=`) then eqn:=op(1,eqn)-op(2,eqn) fi;
        for j to nops(V) do a[i,j]:=coeff(eqn,op(j,V)) od;
        B[i]:=-subs(zero,eqn) od;
    RETURN(op(a))
end: # systomatrix

#listprimpart
# Input: a list of polynomials and a variable
# Output: the list where common factors have been removed.
listprimpart:=proc (l, z)
local g, i, T, q;
    g:=content(convert([seq(op(i,l)*T^(i-1),i=1..nops(l))],`+`),T,'q');
    if has(g,z) then RETURN([seq(coeff(q,T,i),i=0..nops(l)-1)])
    else RETURN(l) fi
end: # listprimpart

# This is used to find a linear dependency.
# The result is the last element of the last line which is not 0.
# The pivoting must not be done as gausselim does it.
# This is a fraction free gausselim using the input matrix for the intermediate
# computations. Also, no type checking is performed.
cheapgausselim:=proc (A, nlin, ncol)
local c, i, j, k, def;
    def:=0;
    for c to nlin do
        for k from c+def to ncol-1 while normal(A[c,k])=0 do A[c,k]:=0 od;
        if k=ncol then RETURN(A[c,ncol]) fi;
        for k from c to nlin while normal(A[k,c+def])=0 do A[k,c+def]:=0 od;
        if k=nlin+1 then def:=def+1; c:=c-1; next fi;
        for i in [seq(i,i=c..k-1),seq(i,i=k+1..nlin)] do
            if A[i,c+def]=0 then next fi;
            for j from c+def+1 to ncol do
                A[i,j]:=A[k,c+def]*A[i,j]-A[i,c+def]*A[k,j] od;
            A[i,c+def]:=0  # this line and the next commented line add some
        od;
        userinfo(5,'rsolve',`line `,k,` eliminated`);
        # speedup for large matrices:
```

```
        if k=c then for i from c to ncol-1 do A[c,i]:=0 od fi;
        userinfo(6,'rsolve','remaining matrix ',print(op(A)));
    od;
    RETURN(FAIL)
end: # cheapgausselim

#lindep
# Input:  a vector u = [u1 , ... , uk] and a matrix A = array(1..k,1..l)
#         such that u[i] = sum(A[i,j] e[j]) for some e[j].
#         The coefficients are rational functions in x.
# Output: a linear dependency relation between the u[i] if there is one,
#         FAIL otherwise
lindep := proc(A, u, x)
local k, i, l, v, rel, unk, j;
    k := nops(u);
    if linalg[rowdim](A)<>k then ERROR(`incorrect number of rows`) fi;
    l := linalg[coldim](A);
    userinfo(2,'gfun',`looking for a linear dependency in a`,k,'x',l,'matrix');
    unk:=[seq(v[i],i=1..k)];
    rel := cheapgausselim(
        linalg[augment](A,linalg[matrix](k,1,unk)),k,l+1);
    if rel=FAIL then RETURN(FAIL) fi;
    rel:=primpart(numer(convert([seq(normal(subs([i=1,seq(j=0,j=subs(i=NULL,unk))],
        rel))*i,i=unk)],`+`)),unk);
    RETURN(convert([seq(subs([i=1,seq(j=0,j=subs(i=NULL,unk))],rel)
        *u[op(i)],i=unk)],`+`))
end: # lindep

#formatdiffeq
# Input:  a list [diffeq,y(x)] containing a linear differential equation
            with polynomial coefficients and its unknown
#         (optional) Y and X
#             names to be assigned the unknown function and its variable
#         (optional) iniconds to be assigned the initial conditions
# Output: a list of polynomials in x [u(x),p_0(x),...,p_d(x)] meaning
#                                                   (d)
#            eqn = p_0(x) y(x) + ... + p_d(x) y    (x) + u(x)
#
#   This is where the type checking is done.
#
formatdiffeq:=proc (l,Y,X,iniconds)
local r, y, x, i, difford, j, lvar, t;
    if nops(l)<>2 then ERROR(`wrong number of arguments`) fi;
    if not type(op(2,l),function(name)) then
        ERROR(`invalid unknown`,op(2,l)) fi;
    x:=op(op(2,l)); y:=op(0,op(2,l)); if nargs>1 then X:=x; Y:=y fi;
    if type(op(1,l),set)  then
        r:=select(has,op(1,l),x);
        if nops(r)>1 then ERROR(`invalid differential equation`) fi;
        if nops(r)=0 then
            ERROR(`the unknown variable does not appear in the equation`) fi;
        if nargs>3 then iniconds:=op(1,l) minus r fi;
        r:=op(r)
    else r:=op(1,l); if nargs>3 then iniconds:={} fi fi;
    if type(r,`=`) then r:=op(1,r)-op(2,r) fi;
    if select(has,indets(r,y(anything)),x) minus {y(x)} <> {} then
```

```
        ERROR(`invalid differential equation`) fi;
    r:=expand(numer(normal(convert(r,D))));
    lvar:= {seq(op(2,op(0,op(0,i))),i=select(proc(u,y)
        has(u,D) and has(u,`@@`) and has(u,y) end,indets(r,function),y))};
    if lvar<>{} then difford:=max(op(lvar))
    elif has(r,D(y)(x)) then difford:=1
    else difford:=0 fi;
    D(y):=D(y);
    r:=[subs([seq((D@@j)(y)(x)=0,j=0..difford)],r),
        seq(coeff(r,(D@@j)(y)(x),1),j=0..difford)];
    if not type(r,list(polynom(anything,x))) then
        ERROR(`non-polynomial coefficients`) fi;
    RETURN(listprimpart(r,x))
end: # formatdiffeq

# makediffeq
# Input:  a differential equation in the format returned by formatdiffeq
#         the variables y and x, meaning the unknown function is y(x)
#         (optional) a set of initial conditions
# Output: the corresponding differential equation, which is in a set if
#         there are initial conditions.
makediffeq:=proc (deq, y, x, ini)
local r, i;
    r:=convert([seq(deq[i+2]*(D@@i)(y)(x),i=0..nops(deq)-2),deq[1]],`+`);
    if nargs=3 or ini={} then RETURN(r) fi;
    RETURN({r,op(ini)})
end: # makediffeq

# makediffeqdiff
# Input:  a differential equation in the format returned by formatdiffeq
#         the variables y and x, meaning the unknown function is y(x)
#         (optional) a set of initial conditions
# Output: the corresponding differential equation, which is in a set if
#         there are initial conditions. The difference with makediffeq
#         is that here diff is used instead of D.
makediffeqdiff:=proc (deq, y, x, ini)
local r, i;
    r:=convert([deq[1],deq[2]*y(x),seq(deq[i+2]*diff(y(x),x$i),
        i=1..nops(deq)-2)],`+`);
    if nargs=3 or ini={} then RETURN(r) fi;
    RETURN({r,op(ini)})
end: # makediffeqdiff

#formatrec
# Input:  a list [rec,u(n)] containing a linear recurrence with
#            polynomial coefficients and its unknown
#         u, n names to be assigned the unknown function and variable
#         (optional) iniconds to be assigned the initial conditions
#         (optional) one: boolean saying whether one is interested in
#                         only one solution.
# Ouput:  a list of polynomials in n: [b(n),p_0(n),...,p_d(n)] meaning
#
#                  rec=p_0(n)u(n)+...+p_d(n)u(n+d)+b(n)
#
#  This is where the type checking is done.
#
```

```
formatrec:=proc (l, u, n, iniconds, one)
local r, i, U, N, mi, ma, locini;
    if nops(l)<2 or nops(l)>5 then
        ERROR(`rsolve: wrong number of arguments`) fi;
    if not type(op(2,l),function(name)) then
        ERROR(`invalid unknown`,op(2,l)) fi;
    N:=op(op(2,l)); U:=op(0,op(2,l)); if nargs>1 then n:=N; u:=U fi;
    if type(op(1,l),set) then
        r:=select(has,op(1,l),N);
        if nops(r)>1 then ERROR(`invalid recurrence`) fi;
        if nargs>3 then locini:=op(1,l) minus r; iniconds:=locini;
            if not type(locini,set(U(integer)=anything)) then
                ERROR(`invalid initial conditions`,locini)
            fi fi;
        r:=op(r)
    else r:=op(1,l); if nargs>3 then iniconds:={} fi
    fi;
    if nargs>4 then if nops(l)>4 and type(op(5,l),boolean) then one:=op(5,l)
    else one:=false fi fi;
    if type(r,`=`) then r:=op(1,r)-op(2,r) fi;
    mi:=minindex(r,U,N); ma:=maxindex(r,U,N);
    r:=collect(r,[seq(U(N+i),i=mi..ma)],normal);
    if mi<>0 then r:=subs(N=N-mi,r) fi;
    if has(map(denom,{op(r)}),N) then
        r:=collect(numer(normal(r)),[seq(U(N+i),i=mi..ma)],normal) fi;
    r:=[subs([seq(U(N+i)=0,i=0..ma-mi)],r),seq(coeff(r,U(N+i),1),i=0..ma-mi)];
    if has(r,U) or not type(r,list(polynom(anything,n))) then
        ERROR(`invalid recurrence or unknown`) fi;
    RETURN(listprimpart(r,N))
end: # formatrec

# makerec
# Input:  a recurrence in the format returned by formatrec
#         the variables u and n, meaning the unknown sequence is u(n)
#         (optional) a set of initial conditions
# Output: the corresponding recurrence, which is in a set if there are
#         initial conditions.
makerec:=proc (rec, u, n, ini)
local r, i;
    r:=convert([seq(rec[i+2]*u(n+i),i=0..nops(rec)-2),rec[1]],`+`);
    if nargs=3 or ini={} then RETURN(r) fi;
    RETURN({r,op(ini)})
end: # makerec

# formatpoleq
# Input:  a list [p, y(z)] containing a polynomial in two variables and
#          an unknown function and possibly initial conditions.
#         y, z names to be assigned the unknown function and variable
#         (optional) iniconds to be assigned the initial conditions.
# Output: the polynomial, type checked and without its inital values
formatpoleq:=proc (l, y, z, iniconds)
local Y, Z, P;
    if nops(l)<2 or nops(l)>4 then
        ERROR(`formatpoleq: wrong number of arguments`) fi;
    if not type(op(2,l),function(name)) then
        ERROR(`invalid unknown`,op(2,l)) fi;
```

```
    Z:=op(1,op(2,l));Y:=op(0,op(2,l)); if nargs>1 then y:=Y; z:=Z fi;
    if type(op(1,l),`=`) then P:=op(1,op(1,l))-op(2,op(1,l))
    else P:=op(1,l) fi;
    if not type(P,polynom(anything,[Y,Z])) then ERROR(`invalid argument`,P) fi;
    if nargs=4 then
        if nops(l)>2 then
            if type(op(3,l),set) then iniconds:=op(3,l)
            else ERROR(`invalid argument`,op(3,l)) fi;
        else iniconds:={} fi fi;
    RETURN(P)
end: # formatpoleq

# `goodinitvalues/rec`
# Input:  a recurrence in the format returned by formatrec
#         the unknown sequence and its variable
#         (optional) some initial conditions
# Output: a set of equalities u(k)=v_k, from which all the other values
#         can be deduced by solving the recurrence for its maximal index.
#         These equalities are of the type u(k)=u(k) when this value is
#         arbitrary.
#         The result is an ERROR when no initial condition can be found.
`goodinitvalues/rec`:=proc (rec, u, n, ini)
local n0, order, i, inds, minind, maxind, sys, r, sol, b, a,minind2,lmin, j, k;
    order:=nops(rec)-2;
    n0:=firstnonzero(subs(n=n-order,rec[nops(rec)]),n);
    if nargs>3 then inds:=map(op,indets(ini,u(integer))) else inds:={} fi;
    maxind:=max(order-1,op(inds),n0-1)-order;
    minind:=min(op(inds),max(0,n0-order-1));
    minind2:=max(order-1,n0-1,op(inds))-order+1;
    if minind2>minind then lmin:=[minind,minind2] else lmin:=[minind] fi;
    r:=makerec(rec,u,n);
    for j to nops(lmin) do
        sys:={op(ini),seq(subs(n=i,r),i=op(j,lmin)..maxind)};
        if sys={} then RETURN({seq(u(i)=u(i),i=0..minind-1)}) fi;
        a:=systomatrix(sys,[seq(u(i),i=minind..maxind+order)],'b');
        sol:=linalg[linsolve](a,b);
        if sol=NULL and j=nops(lmin) then
            ERROR(`no valid initial conditions`)
        elif sol<>NULL then break fi
    od;
    sol:=convert(sol,list);
    for i in indets(sol,_t[anything]) do
        if member(i,sol,'k') then sol:=subs(i=u(minind+k-1),sol) fi od;
    RETURN({seq(u(i)=u(i),i=0..minind-1),
            seq(u(i)=sol[i-minind+1],i=minind..minind+nops(sol)-1)})
end: # `goodinitvalues/rec`

# `goodinitvalues/diffeq`
# Input:  a differential equation in the format returned by formatdiffeq
#         the unknown function and its variable
#         (optional) some initial conditions
# Output: a set of equalities (D@@k)(y)(0)=v_k, from which all the others
#         can be computed.
#         These equalities are of the type (D@@k)(y)(0)=(D@@k)(y)(0) when
#         this value is arbitrary.
#         The result is an ERROR when no initial condition can be found.
```

```
`goodinitvalues/diffeq`:=proc (deq, y, z, ini)
local u, init, i, sol, maxorder;
    if nargs=4 then init:=ini else init:={} fi;
    if subs(z=0,deq[nops(deq)])=0 then
        maxorder:=max(nops(deq)-3,seq(op(2,op(0,op(0,i))),
            i=indets(ini,`gfun/initeq`(y))minus{y(0),D(y)(0)}))
    else maxorder:=nops(deq)-3 fi;
    u:=series(eval(subs(y(z)=convert([seq((D@@i)(y)(0)*z^i/i!,
        i=0..maxorder),O(1)*z^(maxorder+1)],`+`),init,
        makediffeqdiff(deq,y,z))),z,infinity);
    u:={seq(coeff(u,z,i),i=0..maxorder)} minus {O(1)};
    sol:=solve(u,{seq((D@@i)(y)(0),i=0..maxorder)}
        intersect indets(u,function(0)));
    if subs(z=0,op(nops(deq),deq))<>0 then
        init:=select(proc(x,y) member(op(1,x),y) end,init,[seq((D@@i)(y)(0),
            i=0..maxorder)]) fi;
    if sol=NULL then ERROR(`no valid initial conditions`) fi;
    RETURN(remove(type,sol union init,`gfun/identity`))
end: # `goodinitvalues/diffeq`

######################### Conversion Routines ###################

seriestolist:=proc()
local s, meth, l, x, i;
    if args[1]<>'stamped' then RETURN(seriestolist(typecheck(8,args))) fi;
    s:=args[2]; meth:=args[3]; x:=op(0,s);
    if op(nops(s)-1,s)=O(1) then l:=[seq(coeff(s,x,i),i=0..op(nops(s),s)-1)]
    else l:=[seq(coeff(s,x,i),i=0..op(nops(s),s))] fi;
    if meth='ogf' then RETURN(l)
    else RETURN(seriestolist('stamped',listtoseries('stamped',l,x,meth),'ogf'))
    fi
end: # seriestolist

listtolist:=proc()
local x;
    if args[1]<>'stamped' then RETURN(listtolist(typecheck(9,args))) fi;
    if args[3]='ogf' then RETURN(args[2]) fi;
    RETURN(seriestolist('stamped',
        listtoseries('stamped',args[2],x,args[3]),'ogf'))
end: #listtolist

seriestoseries:=proc ()
    if args[1]<>'stamped' then RETURN(seriestoseries(typecheck(8,args))) fi;
    RETURN(listtoseries('stamped',seriestolist('stamped',args[2],'ogf'),
        op(0,args[2]),args[3]))
end: # seriestoseries

listtoseries:=proc ()
    if args[1]<>'stamped' then RETURN(listtoseries(typecheck(1,args))) fi;
    RETURN(gfun[cat(`listtoseries/`,args[4])](args[2],args[3]))
end: # listtoseries

gfun[`listtoseries/egf`]:=proc(l,x)
local i;
    RETURN(series(convert([seq(op(i,l)*x^(i-1)/(i-1)!,i=1..nops(l)),
        O(x^(nops(l)))],`+`),x,nops(l)))
```

```
end:

gfun[`listtoseries/laplace`]:=proc(l,x)
local i;
    RETURN(series(convert([seq(op(i,l)*x^(i-1)*(i-1)!,i=1..nops(l)),
        O(x^(nops(l)))],`+`),x,nops(l)))
end:

gfun[`listtoseries/ogf`]:=proc(l,x)
local i;
    RETURN(series(convert([seq(op(i,l)*x^(i-1),i=1..nops(l)),
        O(x^(nops(l)))],`+`),x,nops(l)))
end: # `listtoseries/ogf`

gfun[`listtoseries/revogf`]:=proc(L,x)
local l, i, t;
    l:=L;
    while op(1,l)=0 and l<>[] do l:=subsop(1=NULL,l) od;
    if l=[] then ERROR(`cannot revert 0 series`) fi;
    Order:=nops(l)+1;
    RETURN(solve(series(convert([seq(op(i,l)*t^i,i=1..nops(l)),t^Order],
        `+`),t)=x,t))
end: # `listtoseries/revogf`

gfun[`listtoseries/revegf`]:=proc(L,x)
local l, i, t;
    l:=L;
    while op(1,l)=0 and l<>[] do l:=subsop(1=NULL,l) od;
    if l=[] then ERROR(`cannot revert 0 series`) fi;
    Order:=nops(l)+1;
    RETURN(solve(series(convert([seq(op(i,l)*t^i/i!,i=1..nops(l)),t^Order]
        ,`+`),t)=x,t))
end: # `listtoseries/revegf`

gfun[`listtoseries/lgdogf`]:=proc(l,x)
local i;
    RETURN(series(convert([seq(i*op(i+1,l)*x^(i-1),i=1..nops(l)-1)],`+`)/
            convert([seq(op(i,l)*x^(i-1),i=1..nops(l))],`+`),x,nops(l)-1))
end: # `listtoseries/lgdogf`

gfun[`listtoseries/lgdegf`]:=proc(l,x)
local i;
    RETURN(series(convert([seq(op(i,l)*x^(i-2)/(i-2)!,i=2..nops(l))],`+`)/
        convert([seq(op(i,l)*x^(i-1)/(i-1)!,i=1..nops(l))],`+`),x,nops(l)-1))
end: # `listtoseries/lgdegf`

listtodiffeq:=proc()
local result, ex, methods, method, y, x, s, unkn, expr;
    if args[1]<>'stamped' then RETURN(listtodiffeq(typecheck(2,args))) fi;
    expr:=args[2];unkn:=args[3];methods:=args[4];
    y:=op(0,unkn);x:=op(unkn);ex:=expr;
    for method in methods do
        s:=listtoseries('stamped',ex,x,method);
        userinfo(3,'gfun',`Trying the `,method,s);
        result:=`s2d/s2d`(s,x,y);
        if result<>FAIL then
```

```
            userinfo(2,'gfun','The',method,``seems to satisfy``,result);
            RETURN([inicond(s,result,y,x),method])
        fi
    od;
    FAIL
end: # listtodiffeq

seriestodiffeq:=proc ()
    if args[1]<>'stamped' then RETURN(seriestodiffeq(typecheck(6,args))) fi;
    RETURN(listtodiffeq('stamped',
        seriestolist('stamped',args[2],'ogf'),args[3],args[4]))
end: # seriestodiffeq

listtoalgeq:=proc()
local result, ex, methods, method, y, x, s, unkn, expr;
    if args[1]<>'stamped' then RETURN(listtoalgeq(typecheck(2,args))) fi;
    expr:=args[2];unkn:=args[3];methods:=args[4];
    y:=op(0,unkn);x:=op(unkn);ex:=expr;
    for method in methods do
        s:=listtoseries('stamped',ex,x,method);
        userinfo(3,'gfun',`Trying the `,method,s);
        result:=`s2a/s2a`(s,x,y);
        if result<>FAIL then
            userinfo(2,'gfun','The',method,``seems to satisfy``,result);
            RETURN([result,method])
        fi
    od;
    FAIL
end: # listtoalgeq

seriestoalgeq:=proc ()
    if args[1]<>'stamped' then RETURN(seriestoalgeq(typecheck(6,args))) fi;
    RETURN(listtoalgeq('stamped',
        seriestolist('stamped',args[2],'ogf'),args[3],args[4]))
end: # seriestoalgeq

listtoratpoly:=proc()
local result, ex, methods, method, s, x;
    if args[1]='stamped' then ex:=args[2];x:=args[3];methods:=args[4]
    else RETURN(listtoratpoly(typecheck(3,args))) fi;
    Order:=nops(ex);
    for method in methods do
        s:=listtoseries('stamped',ex,x,method);
        userinfo(3,'gfun',`Trying the `,method,s);
        result:=normal(factor(convert(s,ratpoly)));
        if degree(numer(result),x)+degree(denom(result),x)<Order-1
        and length(result)<length(ex)+20
        then
            s:=series(s-result,x);
            if s=0 or op(1,s)=O(1) then
                userinfo(2,'gfun','The',method,``seems to be``,result);
                RETURN([result,method])
            fi
        fi
    od;
    FAIL
```

```
end: # listoratpoly

riestoratpoly:=proc () # yes, it's stupid to convert it to a list now.
    if args[1]<>'stamped' then RETURN(seriestoratpoly(typecheck(7,args))) fi;
    RETURN(listtoratpoly('stamped',
            seriestolist('stamped',args[2],'ogf'),op(0,args[2]),args[3]))
end: # seriestoratpoly

`s2d/s2d`:=proc (s, x, y)
local serie,nbterms, unkncoef, eqnproto, eqns, homog, i, j, solver,
degcoef, ordereqn, ldeg, zerocoeff, eqn, neqns, trueinds, dim, inds, sol;
    serie:=eval(subs(O=<0>,s));
    # +1 for the constant term:
    nbterms:=op(nops(serie),serie)+1;
    unkncoef:=array(0..maxordereqn+1,0..maxdegcoeff);
    eqnproto:=convert([seq(unkncoef[0,j]*x^j,j=0..maxdegcoeff),
        convert([seq(unkncoef[1,j]*x^j,j=0..maxdegcoeff)],`+`)*y(x),
        seq(convert([seq(unkncoef[i+1,j]*x^j,j=0..maxdegcoeff)],`+`)*
        diff(y(x),x$i),i=1..maxordereqn)],`+`);
    eqns:=series(eval(subs(y(x)=serie,eqnproto)),x,nbterms);
    if type([seq(op(2*i-1,serie),i=1..iquo(nops(serie),2))],list(rational))
    then solver:=op(ratsolvelin) else solver:=op(gensolvelin) fi;
    for homog from 0 to 1 do for degcoef from mindegcoeff+1 to maxdegcoeff+1 do
    for ordereqn from minordereqn to maxordereqn do
        if homog=0 then ldeg:=[0,degcoef$(ordereqn+1)]
        else ldeg:=[degcoef$(ordereqn+2)] fi;
        userinfo(2,'gfun',``Trying degree sequence``,map(x->x-1,ldeg));
        zerocoeff:=seq(seq(unkncoef[i,j]=0,j=op(i+1,ldeg)..maxdegcoeff),
            i=0..ordereqn+1),seq(seq(unkncoef[i,j]=0,j=0..maxdegcoeff),
            i=ordereqn+2..maxordereqn+1);
        eqn:=eval(subs(zerocoeff,eqns));
        neqns:=iquo(nops(eqn),2)-ordereqn-1;
        if neqns=0 then next fi;
        trueinds:=indets(eqn,unkncoef[anything,anything]);
        dim:=nops(trueinds);
        if neqns<dim+1 or dim=1 then next fi;
#       This "+1" avoids coincidences.
#       if neqns<dim or dim=1 then next fi;
        inds:={seq(seq(unkncoef[i,j],j=0..op(i+1,ldeg)-1),i=0..ordereqn+1)};
        sol:=solver({seq(op(2*i-1,eqn),i=1..neqns)},trueinds);
        if sol = FAIL or type(sol,set(anything=0)) then next fi;
        RETURN(eval(subs({op(sol),seq(j=0,j=inds minus trueinds),zerocoeff},
            eqnproto)))
    od od od;
    FAIL
end: # `s2d/s2d`

`s2a/s2a`:=proc (s, x, y)
local serie, nbterms, unkncoef, eqnproto, eqns, i, j, solver,
degcoef, degeqn, ldeg, zerocoeff, eqn, neqns, trueinds, dim, inds, sol;
    serie:=eval(subs(O=<0>,s));
    nbterms:=op(nops(serie),serie)-1;
    unkncoef:=array(0..maxdegeqn+1,0..maxdegcoeff);
    eqnproto:=convert([seq(convert([seq(unkncoef[i,j]*x^j,j=0..maxdegcoeff)],
        `+`)*y(x)^i,i=0..maxdegeqn)],`+`);
    eqns:=series(eval(subs(y(x)=serie,eqnproto)),x,nbterms+1);
```

```
    if type([seq(op(2*i-1,serie),i=1..iquo(nops(serie),2))],list(rational))
    then solver:=op(ratsolvelin) else solver:=op(gensolvelin) fi;
    for degcoef from mindegcoeff+1 to maxdegcoeff+1 do
    for degeqn from mindegeqn to maxdegeqn do
        ldeg:=[degcoef$(degeqn+1)];
        userinfo(4,'gfun',``trying degree sequence``,map(x->x-1,ldeg));
        zerocoeff:=seq(seq(unkncoef[i,j]=0,j=op(i+1,ldeg)..maxdegcoeff),
            i=0..degeqn),seq(seq(unkncoef[i,j]=0,j=0..maxdegcoeff),
            i=degeqn+1..maxdegeqn+1);
        eqn:=eval(subs(zerocoeff,eqns));
        neqns:=iquo(nops(eqn),2)-1;
        if neqns=0 then next fi;
        trueinds:=indets(eqn,unkncoef[anything,anything]);
        dim:=nops(trueinds);
        if neqns<dim+1 or dim=1 then next fi;
        inds:={seq(seq(unkncoef[i,j],j=0..op(i+1,ldeg)-1),i=0..degeqn)};
        sol:=solver({seq(op(2*i-1,eqn),i=1..neqns)},trueinds);
        if sol = FAIL or type(sol,set(anything=0)) then next fi;
        RETURN(eval(subs({op(sol),seq(j=0,j=inds minus trueinds),zerocoeff},
            eqnproto)));
    od od;
    FAIL
end: # `s2a/s2a`

listtorec:=proc()
local result, methods, method, u, n, s, unkn, expr;
    if args[1]<>'stamped' then RETURN(listtorec(typecheck(2,args))) fi;
    expr:=args[2];unkn:=args[3];methods:=args[4];
    u:=op(0,unkn);n:=op(unkn);
    for method in methods do
        s:=listtolist('stamped',expr,method);
        userinfo(3,'gfun',`Trying the `,method,s);
        result:=`l2r/l2r`(s,n,u);
        if result<>FAIL then
            userinfo(2,'gfun','The',method,``seems to satisfy``,result);
            RETURN([result,method])
        fi
    od;
    FAIL
end: # listtorec

seriestorec:=proc ()
    if args[1]<>'stamped' then RETURN(seriestorec(typecheck(6,args))) fi;
    RETURN(listtorec('stamped',
            seriestolist('stamped',args[2],'ogf'),args[3],args[4]))
end: # seriestorec

`l2r/l2r`:=proc (l,n,u)
local unkncoef, solver, homog, i, j, k, p, sys,
degcoef, ordereqn, ldeg, zerocoeff, eqn, trueinds, dim, inds, sol;
    if type(l,list(rational))
        then solver:=op(ratsolvelin) else solver:=op(gensolvelin) fi;
    unkncoef:=array(0..maxordereqn+1,0..maxdegcoeff);
    for i from 0 to maxordereqn+1 do
        p[i]:=convert([seq(unkncoef[i,j]*n^j,j=0..maxdegcoeff)],`+`)
    od;
```

```
    eqn:=convert([seq(p[i]*'op'(n+i,l),i=1..maxordereqn+1),p[0]],`+`);
    sys:=[seq(subs(n=k-1,eqn),k=1..nops(l))];
    for homog from 0 to 1 do for degcoef from mindegcoeff+1 to maxdegcoeff+1 do
    for ordereqn from minordereqn to maxordereqn do
        if homog=0 then ldeg:=[0,degcoef$(ordereqn+1)]
        else ldeg:=[degcoef$(ordereqn+2)] fi;
        userinfo(2,'gfun',``Trying degree sequence``,map(x->x-1,ldeg));
        zerocoeff:=seq(seq(unkncoef[i,j]=0,j=op(i+1,ldeg)..maxdegcoeff),
            i=0..ordereqn+1),seq(seq(unkncoef[i,j]=0,j=0..maxdegcoeff),
            i=ordereqn+2..maxordereqn+1);
        eqn:={op(eval(subs(zerocoeff,[op(1..nops(l)-ordereqn,sys)])))}minus{0};
        if eqn={} then next fi;
        trueinds:=indets(eqn,unkncoef[anything,anything]);
        dim:=nops(trueinds);
        if nops(eqn)<dim+1 or dim=1 then next fi;
        inds:={seq(seq(unkncoef[i,j],j=0..op(i+1,ldeg)-1),i=0..ordereqn+1)};
        sol:=solver(eqn,trueinds);
        if sol = FAIL or type(sol,set(anything=0)) then next fi;
        RETURN(eval(subs({k=n,op(sol),seq(j=0,j=inds minus trueinds),
            zerocoeff},{convert([seq(p[i]*u(n+i-1),i=1..ordereqn+1),p[0]],`+`),
            seq(u(i-1)=op(i,l),i=1..ordereqn)})))
    od od od;
    FAIL
end: # `l2r/l2r`

inicond:=proc (s, eqn, y ,x)
local order, deq, i;
    deq:=select(has,indets(eqn,'diff(anything,identical(x))'),y(x));
    for order while deq<>{y(x)} do deq:=subs(diff(y(x),x)=y(x),deq) od;
    RETURN({eqn,seq((D@@i)(y)(0)=coeff(s,x,i)*i!,i=0..order-2)})
end: # inicond

listtohypergeom:=proc()
local result, methods, method, s, unkn, expr;
    if args[1]<>'stamped' then RETURN(listtohypergeom(typecheck(3,args))) fi;
    expr:=args[2];unkn:=args[3];methods:=args[4];
    for method in methods do
        s:=listtolist('stamped',expr,method);
        userinfo(3,'gfun',`Trying the `,method,s);
        result:=`l2h/l2h`(s,unkn);
        if result<>FAIL then
            userinfo(2,'gfun','The',method,``seems to satisfy``,result);
            RETURN([result,method])
        fi
    od;
    FAIL
end: # listtohypergeom

seriestohypergeom:=proc ()
    if args[1]<>'stamped' then RETURN(seriestohypergeom(typecheck(7,args))) fi;
    RETURN(listtohypergeom('stamped',
            seriestolist('stamped',args[2],'ogf'),op(0,args[2]),args[3]))
end: # seriestohypergeom

`l2h/l2h`:=proc (l, x)
local a, a0, k, eqn, u, v, w, den, i, z, c;
```

```
    a:=1;
    for k while op(1,a)=0 do a:=subsop(1=NULL,a) od;
    a0:=op(1,a);k:=k-1;
    if nops(a)<5 then RETURN(FAIL) fi;
    a:=[seq(op(i,a)/a0,i=2..nops(a))];
    eqn:=normal((6*a[4]*a[1]**2*a[2]+9*a[2]*a[3]**2+6*a[4]*a[1]*a[3]-6*
        a[3]**2*a[1]**2+a[2]**2*a[3]*a[1]-16*a[4]*a[2]**2)*x**2+(-32*a[4]
        *a[2]**2+5*a[2]**2*a[3]*a[1]+6*a[4]*a[1]*a[3]+18*a[4]*a[1]**2*
        a[2]+27*a[2]*a[3]**2-24*a[3]**2*a[1]**2)*x+6*a[2]**2*a[3]*a[1]-
        18*a[3]**2*a[1]**2+12*a[4]*a[1]**2*a[2]);
    if eqn=0 then v:=1
    elif degree(eqn,x)=0 then RETURN(FAIL)
    else v:=op(1,[solve(eqn,x)])
    fi;
    den:=normal(4*a[2]**2*v**2-3*a[3]*v**2*a[1]-a[2]*v**2*a[1]**2-3*a[2]*v*
        a[1]**2+8*a[2]**2*v-3*a[3]*v*a[1]-2*a[2]*a[1]**2);
    if den=0 then RETURN(FAIL) fi;
    w:=normal(-2*(2*a[2]**2*v+4*a[2]**2-3*a[3]*v*a[1]-3*a[3]*a[1])*v)/den;
    if type(w,negint) or w=0 then RETURN(FAIL) fi;
    z:=normal(-3*a[3]*v*a[1]**2+a[1]*a[2]**2*v+3*a[3]*v*a[2]-3*a[3]*
        a[1]**2+2*a[2]**2*a[1]);
    if z=0 then RETURN(FAIL) fi;
    u:=-normal(2*a[2]**2*v+4*a[2]**2-3*a[3]*v*a[1]-3*a[3]*a[1])*a[1]/z;
    z:=2*z/den;
    userinfo(3,'gfun',`candidate: hypergeom(`,[u,v],[w],z*x,`)`);
    c:=u*(u+1)*(u+2)*(u+3)*v*(v+1)*(v+2)*(v+3)/w/(w+1)/(w+2)/(w+3)*z^4/24;
    for i from 5 to nops(a) do
        c:=c*(u+i-1)*(v+i-1)*z/(w+i-1)/i;
        if c<>op(i,a) then RETURN(FAIL) fi;
    od;
    userinfo(2,'gfun',`hypergeom found, parameters:`,[u,v],[w],z*x);
    RETURN(simplify(a0*x^k*hypergeom([u,v],[w],z*x),hypergeom))
end: # `l2h/l2h`

#ratpolytocoeff
# Input: a rational function of x
#        x its variable
#        n a name
# Output: the nth coefficient of the Taylor expansion at the origin of f.
ratpolytocoeff:=proc(f,x,n)
local g;
    if not type(fullparfrac,procedure) then
        g:=traperror(readlib(fullparfrac));
        if not type(g,procedure) then
        ERROR(`This function requires the procedure fullparfrac from the share library`)
        fi
    fi;
    g:=fullparfrac(f,x);
    if type(g,`+`) then g:=[op(g)] else g:=[g] fi;
    RETURN(convert(map(`ratpolytocoeff/elmt`,g,x,n),`+`))
end: # ratpolytocoeff

`ratpolytocoeff/elmt`:=proc(g,x,n)
local k, a, c, i;
    if type(g,function) then
        RETURN(op(0,g)(`ratpolytocoeff/elmt`(op(1,g),x,n),op(2..nops(g),g)))fi;
```

```
    if type(g,polynom(anything,x)) then RETURN(0) fi;
    # g must be c(a)*(x-a)^(-k)
    k:=select(has,indets(g,`^`),x);
    if nops(k)<>1 then ERROR(`report this as a bug`,g,x,n) fi;
    k:=op(k);
    a:=x-op(1,k);
    c:=g/k;
    k:=-op(2,k);
    RETURN(c/(-a)^k*a^(-n)*convert([seq(n+i,i=1..k-1)],`*`)/(k-1)!)
end: # `ratpolytocoeff/elmt`

guesseqn:=proc ()
local y, result, l, x, methods;
    if args[1]<>'stamped' then RETURN(guesseqn(typecheck(2,args))) fi;
    l:=args[2];y:=op(0,args[3]);x:=op(args[3]);methods:=args[4];
    # First try to find a rational function
    userinfo(1,'gfun',`Trying to find a rational generating function`);
    result:=listtoratpoly('stamped',l,x,methods);
    if result<>FAIL then
        RETURN([denom(op(1,result))*y(x)-numer(op(1,result)),op(2,result)]) fi;
    # Then a linear differential equation
    userinfo(1,'gfun',`Trying to find a linear differential equation`);
    RETURN(listtodiffeq('stamped',l,y(x),methods))
end: # guesseqn

guessgf:=proc ()
local interres, y, result, l, x, methods, inds, s, i;
    if args[1]<>'stamped' then RETURN(guessgf(typecheck(3,args))) fi;
    l:=args[2];x:=args[3];methods:=args[4];
    # First try to find a rational function
    userinfo(1,'gfun',`Trying to find a rational generating function`);
    result:=listtoratpoly('stamped',l,x,methods);
    if result<>FAIL then RETURN(result) fi;
    # Then trap easy hypergeometrics
    userinfo(1,'gfun',`Trying to find an hypergeometric generating function`);
    result:=listtohypergeom('stamped',l,x,methods);
    if result<>FAIL then RETURN(result) fi;
    # Then a linear differential equation
    userinfo(1,'gfun',`Trying to find a linear differential equation`);
    interres:=listtodiffeq('stamped',l,y(x),methods);
    if interres=FAIL then RETURN(FAIL) fi;
    userinfo(1,'gfun',`Trying to solve it`);
    result:=dsolve(op(1,interres),y(x));
    if result<>FAIL and result<>NULL then
        inds:=(indets(op(2,result),name) minus indets(l,name)) minus {x};
        if inds={} then RETURN([op(2,result),op(2,interres)]) fi;
        s:=series(op(2,result),x,nops(l)+1);
        s:=solve({seq(coeff(s,x,i-1)-op(i,l),i=1..nops(l))},inds);
        if s<>NULL and type(s,set) then
            RETURN(subs(s,[op(2,result),op(2,interres)])) fi
    fi;
#   if has(eqn,diff(y(x),x$2)) and not has(eqn,diff(y(x),x$3)) then
#       # try to trap hypergeometric equations there
#       result:=solvehypergeom(eqn,y,x);
#       if result<>FAIL then RETURN(result) fi;
#   fi;
```

```
    FAIL
end: # guessgf

######################## Holonomic Functions ########################

#algeqtodiffeq
# Input:  a polynomial in two variables y and z
#         the unknown function y(z)
#         (optional) a set of initial conditions
# Output: a linear differential equation verified by RootOf(P,y)
#         or FAIL if the initial conditions cannot be satisfied
algeqtodiffeq := proc()
local y, z, P, inits, g, u, d, i, Y, deq, j, r, inity, P0;
    P:=formatpoleq([args],'y','z','inits');
    g:=mygcdex(diff(P,y),P,y,'u');
    if has(g,y) then RETURN(algeqtodiffeq(normal(P/g),y(z),inits)) fi;
    d := degree(P,y); userinfo(3,'gfun',`degree is `,d);
    if d<=1 then deq:=subs(y=y(z),P)
    elif not has(P,z) then deq:=y(z)-RootOf(P,y)
    else
        Y[1]:=rem(-u/g*diff(P,z),P,y);
        for i from 2 to d-1 do # compute Y[i] = diff(y,z$i) mod P
            Y[i]:=rem(collect(diff(Y[i-1],z)+diff(Y[i-1],y)*Y[1],y),P,y) od;
        deq:=lindep(array(1..d+1,1..d,[[1,0$(d-1)],[0,1,0$(d-2)],
            seq([seq(coeff(Y[i],y,j),j=0..d-1)],i=1..d-1)]),
            [1,seq((D@@i)(y)(z),i=0..d-1)],z)
    fi;
    userinfo(1,'gfun',`differential equation is`,deq);
#    if inits={} then RETURN(deq) fi;
    P0:=subs(z=0,y=y(0),inits,P);
    if not has(P0,y) and P0<>0 then ERROR(`invalid initial condition`) fi;
    if P0<>0 then
#     This will be the correct way when Maple does not insist on the argument
#     of RootOf being irreducible.
#       inits:=inits union {y(0)=RootOf(subs(y(0)=_Z,P0))} fi;
        if traperror(evala(RootOf(subs(y(0)=_Z,P0))))<>lasterror then
            inits:=inits union {y(0)=RootOf(subs(y(0)=_Z,P0))} fi;
    fi;
    inity:=y=subs(inits,y(0));
    inits:={y(0)=op(2,inity)} union subs(y(0)=op(2,inity),inits);
    for i to d-1 do
        r:=traperror((D@@i)(y)(0)=subs([z=0,inity],Y[i]));
        if r<>lasterror then inits:=inits union {r} else break fi
    od;
    inits:=`goodinitvalues/diffeq`(formatdiffeq([deq,y(z)]),y,z,
        select(type,inits,anything=`gfun/free`(y(0))));
    if inits={} then RETURN(deq) else RETURN({deq,op(inits)}) fi
end: # algeqtodiffeq

#diffeqtorec
# Input:  eqn: differential equation (for example output of algeqtodiffeq)
#         y(z): its unknown function
#         u(n): the name of the sequence of Taylor coefficients at the origin
# Output: the linear recurrence satisfied by u(n)
#
diffeqtorec:=proc (eqn,yofz,uofk)
```

```
local iniconds, f, y, z, u, k;
    if nargs<>3 then ERROR(`wrong number of arguments`) fi;
    if not type(uofk,function(name)) then ERROR(`invalid argument`,uofk) fi;
    u:=op(0,uofk);k:=op(uofk);
    f:=formatdiffeq([eqn,yofz],'y','z','iniconds');
    RETURN(`gfun/diffeqtorec/doit`(f,y,z,u,k,iniconds))
end: # diffeqtorec

`gfun/diffeqtorec/doit` := proc(R,y,z,u,k,iniconds)
local l, ini, i, rec, j, minordrec, maxordrec, m, r, dr1;
    if has(R,[k,u]) then ERROR(`invalid arguments`) fi;
    # initial conditions
    l:={seq(op(2,op(0,op(0,i))),i=indets([iniconds,R],`gfun/initeq`(y))
        minus {y(0),D(y)(0)})};
    ini:=[y(0)=u(0),D(y)(0)=u(1),seq((D@@i)(y)(0)=u(i)*i!,i=1)];
    r:=subs(ini,R); ini:=subs(ini,iniconds);
    minordrec:=min(seq(i-degree(op(i+2,r),z),i=0..nops(r)-2));
    maxordrec:=max(seq(i-ldegree(op(i+2,r),z),i=0..nops(r)-2));
    rec:=array(sparse,minordrec..maxordrec);
    for i from 2 to nops(r) do
        for j from ldegree(op(i,r),z) to degree(op(i,r),z) do
            rec[i-2-j]:=rec[i-2-j]+coeff(op(i,r),z,j)*
                expand(convert([seq(k+m,m=1-j..i-2-j)],`*`)) od od;
    # inhomogeneous part
##    if r[1]<>0 then
#       de:=r;
#       for i from 0 to degree(r[1],z) do
#           de0:=subs(z=0,de);
#           ini:={op(ini),convert([de0[1],
                seq(de0[j]*u(j-2)*(j-2)!,j=2..nops(de0))],`+`)};
            de:=[diff(de[1],z),diff(de[2],z),
#               seq(diff(de[j],z)+de[j-1],j=3..nops(de)),de[nops(de)]] od fi;
        if r[1]=0 then dr1:=-1 else dr1:=degree(r[1],z) fi;
        ini:={op(ini),op(map(convert,[seq(subs(k=i,[coeff(r[1],z,i),seq(
            rec[j]*u(i+j),j=max(minordrec,-i)..maxordrec)]),i=0..dr1)],`+`))};
        for i from dr1+1 while i<-minordrec or
            subs(k=i,{seq(rec[maxordrec-j],j=0..(i-dr1-1))})={0} do
            ini:={op(ini),convert(subs(k=i,[seq(rec[j]*u(i+j),
                j=max(minordrec,-i)..maxordrec)]),`+`)} od;
#    fi;
    rec:=listprimpart(
        subs(k=k-minordrec,[0,seq(rec[i],i=minordrec..maxordrec)]),k);
    while nops(rec)>2 and rec[nops(rec)]=0 do
        rec:=subsop(nops(rec)=NULL,rec) od;
    ini:=select(type,`goodinitvalues/rec`(rec,u,k,ini),
        u(integer)=`gfun/free`(u));
    RETURN(makerec(map(collect,rec,k),u,k,ini))
end: # `gfun/diffeqtorec/doit`

#rectoproc
# Input:  a recurrence and its unknown function.
#         (optional) 'remember'
# Output: a procedure computing values of the sequence.
#
#   If 'remember' is given, then the procedure uses option remember to
# compute values of the sequence in linear time and linear space.
```

```
# Otherwise, if the recurrence has constant coefficients, then binary
# powering is used to make the procedure logarithmic time/logarithmic space.
# If the coefficients are not constants, then the procedure is a simple loop
# that will compute the values in linear time/constant space.
#
# The variables that are reserved for the produced code are lowercase
# all the other variables are uppercase.
rectoproc := proc(expr,yofn)
local T,Y,N,R,INITS,ARGLIST,RECSTAT,A,U,I,FIRSTSTAT,REMBR,ORDER,N0,n,a,l,res,i, INIVECT,INIMAT
    R:=formatrec([args[1..2]],'Y','N','INITS');
    REMBR:=(nargs>2) and (args[3]='remember');
    ORDER:=nops(R)-2;
    INITS:=`goodinitvalues/rec`(R,Y,N,INITS);
    if INITS={} then INITS:={seq(Y(I)=Y(I),I=0..ORDER-1)} fi;
    N0:=nops(INITS);
    R:=subs(N=N-ORDER,R);
    if MAPLE5.2 then
        ARGLIST:=[n..nonnegint]; FIRSTSTAT:=NULL
    else
        ARGLIST:=[n];
        FIRSTSTAT:=`&if`(not 'type'(n,nonnegint),
            `&ERROR`(`invalid argument`,n)) fi;
    RECSTAT:=-convert(convert([R[1],seq(R[I+2]*U[I],I=0..ORDER-1)],`+`),
        horner,N)/convert(R[nops(R)],horner,N);
    if REMBR then        ######## Easy case: linear time/linear space###
        T:=subsop(4=NULL,readlib(procmake)(`&proc`(ARGLIST,[],['remember'],
            `&statseq`(FIRSTSTAT,`&RETURN`(subs([N=n,
                seq(U[I]=`&args`[-2](n-ORDER+I),I=0..ORDER-1)],RECSTAT))))))
    elif has(R,N) or R[1]<>0 then##linear time/constant space###########
        T:=readlib(procmake)(`&proc`(ARGLIST,[i,seq(evaln(u.I),I=0..ORDER)],
            [],`&statseq`(FIRSTSTAT,op(subs(INITS,
                [seq(`&:=`(evaln(u.I),Y(N0-ORDER+I)),I=0..ORDER-1)])),
                `&for`(i,N0,1,n-1,true,`&statseq`(
                    `&:=`(evaln(u.ORDER),subs([N=i,seq(U[I]=evaln(u.I),
                        I=0..ORDER-1)],RECSTAT)),
                    seq(`&:=`(evaln(u.I),evaln(u.(I+1))),I=0..ORDER-1))),
                `&RETURN`(subs([N=n,
                    seq(U[I]=evaln(u.I),I=0..ORDER-1)],RECSTAT)))))
    else                 ######## logarithmic time/constant space#######
        INIMAT:=[[seq(-R[nops(R)-I]/R[nops(R)],I=1..ORDER)],
                seq([0$(I-1),1,0$(ORDER-I)],I=1..ORDER-1)];
        INIVECT:=subs(INITS,[seq(Y(N0-I),I=1..ORDER)]);
        A:=array(1..ORDER,1..ORDER);RES:=array(1..ORDER);
        T:=readlib(procmake)(`&proc`(ARGLIST,[a,res,i,l],[],
          `&statseq`(FIRSTSTAT,
            `&:=`(a,'array'(1..ORDER,1..ORDER,INIMAT)),
            `&:=`(res,'array'(1..ORDER,INIVECT)),
            `&:=`(l,'convert'(n-N0+1,base,2)),
            `&if`(l[1]=1,`&:=`(res,'array'(1..ORDER,op(3,
                linalg['multiply'](array(INIMAT),array(1..ORDER,INIVECT))))),
            `&if`(l=[1],`&RETURN`(res[1])),
           `&for`(i,'subsop'(l=NULL,'nops'(l)=NULL,l),true,`&statseq`(
                `&:=`(a,'array'(1..ORDER,1..ORDER,subs(A=a,
                    op(3,linalg['multiply'](A,A))))),
                `&if`(i=1,`&:=`(res,'array'(1..ORDER,op(3,subs(A=a,RES=res,
                    linalg['multiply'](A,RES))))))),
```

```
            `&RETURN`(subs(A=a,RES=res,linalg['multiply'](
                linalg['multiply'](A,A),RES)[1])))))
    fi;
    # put the initial conditions in the remember table
    for I in INITS do T(op(op(1,I))):=op(2,I) od;
    RETURN(op(T))
end: # rectoproc

#rectodiffeq
# Input:  expr: a linear recurrence (with or without initial conditions)
#         a(n): its unknown function
#         f(t): the function sum(a(n)*t^n,n=0..infinity)
# Output: the linear differential equation satisfied by f(t).
#
rectodiffeq := proc(expr,aofn,foft)
local r, a, n, f, t, iniconds;
    if nargs<>3 then ERROR(`wrong number of arguments`) fi;
    if not type(foft,function(name)) then ERROR(`invalid argument`,foft) fi;
    f:=op(0,foft);t:=op(foft);
    r:=formatrec([expr,aofn],'a','n','iniconds');
    iniconds:=`goodinitvalues/rec`(r,a,n,iniconds);
    RETURN(`gfun/rectodiffeq/doit`(r,a,n,f,t,iniconds))
end: # rectodiffeq

`gfun/rectodiffeq/doit`:=proc (r,u,n,f,z,iniconds)
local order, diffeq, P, k, p, a, rr, ini, i, k0, inds, res, t;
    if has(r,[f,z]) then ERROR(`invalid arguments`) fi;
    order:=max(op(map(degree,r,n)));
    diffeq:=array(sparse,-1..order);
    k0:=nops(r)-2;
    D(f):=D(f);
    # To keep coefficients polynomial, we multiply them by z^(nops(r)-2)
    for k from 0 to k0 do
        P:=op(k+2,r);
        for p from 0 to degree(P,n) do
            a:=subs(n=p-k,P);
            P:=quo(P-a,n+k-p,n);
            diffeq[p]:=diffeq[p]+a*z^(k0+p-k);
            if p>0 then
                diffeq[-1]:=diffeq[-1]-collect(a*z^(k0+p-k)*diff(
                    convert([seq((D@@i)(f)(0)*z^i,i=p..k-1)],`+`),z$p),z)
            else
                diffeq[-1]:=diffeq[-1]-collect(a*z^(k0+p-k)*convert(
                [seq((D@@i)(f)(0)*z^i,i=p..k-1)],`+`),z) fi od od;
    P:=op(1,r);
    for p from 0 to degree(P,n) do
        rr[p]:=subs(n=-p-1,P);
        P:=quo(P-rr[p],(n+p+1)/(p+1),n) od;
    diffeq:=[diffeq[-1]*(1-z)^p+z^k0*convert([seq(rr[k]*(1-z)^(p-k-1),
        k=0..p-1)],`+`),seq((1-z)^p*diffeq[k],k=0..order)];
    # initial conditions
    ini:=remove(type,iniconds,`gfun/identity`);
    inds:=map(op,indets(ini,u(anything)));
    ini:=solve(subs([seq(u(i)=(D@@i)(f)(0)/i!,i=inds)],ini),
        {seq((D@@i)(f)(0),i=inds)});
    diffeq:=subs(ini,diffeq);
```

```
    # some initial conditions may correspond to polynomial inhomogeneities
    # at least when the equation is not singular at the origin.
     if nops(ini)>order then
    inds:=max(op(inds));
    if subs(z=0,diffeq[nops(diffeq)])<>0 and inds<>-infinity and inds>order
        then
        diffeq:=subsop(1=collect(diffeq[1],z)-convert(series(eval(subs(f(z)=
            convert([seq((D@@i)(f)(0)*z^i/i!,i=0..inds),O(1)*z^(inds+1)],`+`)
            ,ini,makediffeqdiff(diffeq,f,z))),z,infinity),polynom),diffeq)
    fi;
    if has(diffeq[1],f(0)) and subs(f(0)=0,diffeq[1])=0 then
        # make it homogeneous
        res:=-diff(diffeq[1],z);
        if nops(diffeq)>2 then
            diffeq:=map(collect,[0,res*diffeq[2]+diffeq[1]*diff(diffeq[2],z),
            seq(res*diffeq[i]+diffeq[1]*diff(diffeq[i],z)+diffeq[1]*diffeq[i-1]
            ,i=3..nops(diffeq)),diffeq[1]*diffeq[nops(diffeq)]],z)
        else diffeq:=map(collect,[0,diff(diffeq[2],z)*diffeq[1]+res*diffeq[2],
            diffeq[2]*diffeq[1]],z) fi fi;
    diffeq:=listprimpart(diffeq,z);
    ini:=`goodinitvalues/diffeq`(diffeq,f,z,ini);
    res:=makediffeq(subs(ini,diffeq),f,z);
    if ini<>{} then RETURN({res,op(ini)}) else RETURN(res) fi
end: # `gfun/rectodiffeq/doit`

# borel, invborel
# Input:  a linear recurrence or differential equation
#         u(n) or y(x) the variable
#         (optional) a flag 'diffeq' saying that it's a differential equation
#             by default it is a recurrence
# Output: the linear recurrence or differential equation in u(n) (or y(x))
#         satisfied by the sequence u(n)/n! in the borel case, u(n)*n! in the
#         invborel case. For differential equations, the equation is the
#         equation satisfied by the generating function of the borel/invborel
#         transform of the sequence of Taylor coefficients.
gfun[invborel]:=proc() RETURN(`gfun/borelinvborel`(false,args)) end:
gfun[borel]:=   proc() RETURN(`gfun/borelinvborel`(true, args)) end:

`gfun/borelinvborel` := proc(borel,expr,aofn)
local a, n;
    if nargs<3 then ERROR(`Not enough arguments`) # not necessary in V.2
    elif nargs=3 then
        if not type(aofn,function(name)) then ERROR(`Invalid argument`,aofn)fi;
        a:=op(0,aofn); n:=op(aofn);
        if borel then RETURN(`rec*rec`(expr,{n*a(n)=a(n-1),a(0)=1},a(n)))
        else RETURN(`rec*rec`(expr,{a(n)=n*a(n-1),a(0)=1},a(n))) fi
    elif args[4]<>'diffeq' then ERROR(`invalid argument`,args[4])
    else
        RETURN(rectodiffeq(procname(borel,
            diffeqtorec(expr,aofn,a(n)),a(n)),a(n),aofn))
    fi
end: # `gfun/borelinvborel`

# diffeq+diffeq
#Input: two differential equations Eq1 and Eq2 in the variable y(z)
#Output: a differential equation satisfied by the sum of a solution of Eq1 and
```

```
#        a solution of Eq2.
`diffeq+diffeq` := proc(Eq1,Eq2,yofz)
 cal y,z,A,d1,d2,d,i,j,f,g,eq1,eq2,C, ini1, ini2, eq, c, C2, ini;
    eq1:=formatdiffeq([Eq1,yofz],'y','z','ini1'); d1:=nops(eq1)-2;
    ini1:=remove(type,`goodinitvalues/diffeq`(eq1,y,z,ini1),`gfun/identity`);
    eq2:=formatdiffeq([Eq2,yofz],'y','z','ini2'); d2:=nops(eq2)-2;
    ini2:=remove(type,`goodinitvalues/diffeq`(eq2,y,z,ini2),`gfun/identity`);
    userinfo(2,gfun,`computing the sum of two holonomic functions of order`,
        d1,`and`,d2);
    d:=d1+d2; # maximal order of the sum
    if d=0 then
        eq:=normal(op(1,eq1)/op(2,eq1)+op(1,eq2)/op(2,eq2));
        eq:=[denom(eq),numer(eq)]
    else
        A := array(1..d+1,1..d,sparse);
        C := array(1..d+1,sparse); # constant (rational) terms
        eq1:=[seq(-eq1[i]/eq1[d1+2],i=1..d1+1)];
        # column 1..d1: f .. (D@@(d1-1))(f)
        for i to d1 do A[i,i]:=1 od;
        if d1>0 then
            for i from d1+1 to d+1 do
                c:=A[i-1,d1];
                C[i]:=diff(C[i-1],z)+c*eq1[1];
                A[i,1]:=diff(A[i-1,1],z)+c*eq1[2];
                for j from 2 to d1 do
                    A[i,j]:=diff(A[i-1,j],z)+A[i-1,j-1]+c*eq1[j+1] od od
        else C[1]:=eq1[1];
            for i from 2 to d+1 do C[i]:=diff(C[i-1],z) od fi;
        eq2:=[seq(-eq2[i]/eq2[d2+2],i=1..d2+1)];
        # column d1+1..d1+d2: g .. (D@@(d2-1))(g)
        for i to d2 do A[i,d1+i]:=1 od;
        if d2>0 then
            C2:=0;
            for i from d2+1 to d+1 do
                c:=A[i-1,d];
                C2:=diff(C2,z)+c*eq2[1];
                C[i]:=C[i]+C2;
                A[i,d1+1]:=diff(A[i-1,d1+1],z)+c*eq2[2];
                for j from 2 to d2 do
                    A[i,d1+j]:=diff(A[i-1,d1+j],z)+A[i-1,d1+j-1]+c*eq2[j+1] od
            od
        else C2:=op(eq2); C[1]:=C[1]+C2;
            for i from 2 to d+1 do C2:=diff(C2,z); C[i]:=C[i]+C2 od fi;
        eq:=lindep(A,[seq((D@@i)(y)(z)-C[i+1],i=0..d)],z);
    fi;
    # initial conditions
    ini:=subs(subs(y=f,ini1) union subs(y=g,ini2),
        {seq(i=subs(y=f,i)+subs(y=g,i),
            i={seq(op(1,i),i=ini1)} intersect {seq(op(1,i),i=ini2)})});
    if ini={} then RETURN(eq) else RETURN({eq,op(ini)}) fi
end: # `diffeq+diffeq`

# diffeq*diffeq
#Input: two differential equations Eq1 and Eq2 in the variable y(z)
#Output: a differential equation satisfied by the product of a solution of Eq1
#        and a solution of Eq2.
```

```
`diffeq*diffeq` := proc(Eq1,Eq2,yofz)
local y, z, A,d1,d2,d,i,j,f,g,eq1,eq2, ini1, ini2, e1, eq, ini,k,Y;
    eq1:=formatdiffeq([Eq1,yofz],'y','z','ini1'); d1:=nops(eq1)-2;
    ini1:=remove(type,`goodinitvalues/diffeq`(eq1,y,z,ini1),`gfun/identity`);
    eq2:=formatdiffeq([Eq2,yofz],'y','z','ini2'); d2:=nops(eq2)-2;
    ini2:=remove(type,`goodinitvalues/diffeq`(eq2,y,z,ini2),`gfun/identity`);
    userinfo(2,gfun,`computing the product of two holonomic functions of order`
        ,d1,`and`,d2);
    if d1=0 then eq:=makediffeq(formatdiffeq([numer(eval(subs(yofz=
        -Y(z)*op(2,eq1)/op(1,eq1),makediffeqdiff(eq2,y,z)))),Y(z)]),y,z)
    elif d2=0 then eq:=makediffeq(formatdiffeq([numer(eval(subs(yofz=
        -Y(z)*op(2,eq2)/op(1,eq2),makediffeqdiff(eq1,y,z)))),Y(z)]),y,z)
    else
        if op(1,eq1)<>0 then d2 else 0 fi;
        if op(1,eq2)<>0 then d1 else 0 fi;
        d:=d1*d2+"+""; # maximal order of the product
        A := array(1..d+1,1..d,sparse);
        # (D@@i)(f)*(D@@j)(g) -> column i*d2+j+1, 0<=i<d1, 0<=j<d2
        eq1:=[seq(-eq1[i]/eq1[d1+2],i=1..d1+1)];
        eq2:=[seq(-eq2[i]/eq2[d2+2],i=1..d2+1)];
        A[1,1]:=1;
        for k from 2 to d+1 do
            # fg          fg          f g^{(d2-1)}                f^{(d1-1)} g
            A[k,1]:=diff(A[k-1,1],z)+A[k-1,d2]*eq2[2]+A[k-1,(d1-1)*d2+1]*eq1[2];
            for j to d2-1 do                        # f g^{(j)}
                A[k,j+1]:=diff(A[k-1,j+1],z)                # f g^{(j)}
                    +A[k-1,j]                               # f g^{(j-1)}
                    +A[k-1,d2]*eq2[j+2]                     # f g^{(d2-1)}
                    +A[k-1,(d1-1)*d2+j+1]*eq1[2]            # f^{(d1-1)}g^{(j)}
            od;
            for i to d1-1 do                        # f^{(i)} g
                A[k,i*d2+1]:=diff(A[k-1,i*d2+1],z)          # f^{(i)} g
                    +A[k-1,(i-1)*d2+1]                      # f^{(i-1)} g
                    +A[k-1,(d1-1)*d2+1]*eq1[i+2]            # f^{(d1-1)} g
                    +A[k-1,(i+1)*d2]*eq2[2];                # f^{(i)} g^{(d2-1)}
                for j to d2-1 do                            # f^{(i)} g^{(j)}
                    A[k,i*d2+j+1]:=                 # f^{(i)} g^{(j)}
                        diff(A[k-1,i*d2+j+1],z)             # f^{(i)} g^{(j)}
                        +A[k-1,(i-1)*d2+j+1]                # f^{(i-1)} g^{(j)}
                        +A[k-1,i*d2+(j-1)+1]                # f^{(i)} g^{(j-1)}
                        +A[k-1,(i+1)*d2]*eq2[j+2]           # f^{(i)} g^{(d2-1)}
                        +A[k-1,(d1-1)*d2+j+1]*eq1[i+2]; # f^{(d1-1)} g^{(j)}
                od od od;
        if op(1,eq2)<>0 then # (D@@i)(f) -> column d1*d2+i+1, 0<=i<d1
            e1:=d1;
            for k from d2 to d+1 do                 # f
                A[k,d1*d2+1]:=
                    diff(A[k-1,d1*d2+1],z)                  # f
                    +A[k-1,d2]*op(1,eq2)                    # f g^{(d2-1)}
                    +A[k-1,(d1+1)*d2]*eq1[2];               # f^{(d1-1)}
                for i to d1-1 do                            # f^{(i)}
                    A[k,d1*d2+i+1]:=
                        diff(A[k-1,d1*d2+i+1],z)            # f^{(i)}
                        +A[k-1,d1*d2+i]                     # f^{(i-1)}
                        +A[k-1,(i+1)*d2]*eq2[1]             # f^{(i)} g^{(d2-1)}
                        +A[k-1,(d1+1)*d2]*eq1[i+2]          # f^{(d1-1)}
```

```
                        od od
        else e1:=0 fi;
        if op(1,eq1)<>0 then # (D@@i)(g) -> column d1*d2+e1+i+1, 0<=i<d2
            for k from d1 to d+1 do
                A[k,d1*d2+e1+1]:=
                    diff(A[k-1,d1*d2+e1+1],z)           # g
                    +A[k-1,(d1-1)*d2+1]*eq1[1]          # f^{(d1-1)} g
                    +A[k-1,(d1+1)*d2+e1]*eq2[2];        # g^{(d2-1)}
                for i to d2-1 do
                    A[k,d1*d2+e1+i+1]:=
                        diff(A[k-1,d1*d2+e1+i+1],z)     # g^{(i)}
                        +A[k-1,d1*d2+e1+i]              # g^{(i-1)}
                        +A[k-1,(d1-1)*d2+i+1]*eq1[1]    # f^{(d1-1)} g^{(i)}
                        +A[k-1,(d1+1)*d2+e1]            # g^{(d2-1)}
                        od od fi;
        eq:=lindep(A,[seq((D@@i)(y)(z),i=0..d)],z);
    fi;
    ini:=`goodinitvalues/diffeq`(formatdiffeq([eq,y(z)]),y,z,
        subs(subs(y=f,ini1) union subs(y=g,ini2),
        {seq(i=subs(y=f,i)*subs(y=g,i),
        i={seq(op(1,i),i=ini1)} intersect {seq(op(1,i),i=ini2)})})));
    if ini={} then RETURN(eq) else RETURN({eq,op(ini)}) fi
end: # `diffeq*diffeq`

# rec+rec
#Input: two linear recurrences rec1 and rec2 in the variable uofn (u(n))
#Output: a linear recurrence satisfied by the sum of a solution of rec1 and
#        a solution of rec2.
`rec+rec` := proc(rec1,rec2,uofn)
local y, z;
    RETURN(diffeqtorec(`diffeq+diffeq`(
        rectodiffeq(rec1,uofn,y(z)),rectodiffeq(rec2,uofn,y(z)),y(z)),
        y(z),uofn))
end: # `rec+rec`

# cauchyproduct
#Input: two linear recurrences rec1 and rec2 in the variable uofn (u(n))
#Output: a linear recurrence satisfied by \sum_{k=0}^n{u_kv_{n-k}}, where
#                u is a solution of rec1 and v is a solution of rec2.
#  I do not understand why this does not work:
# cauchyproduct := subs(`diffeq+diffeq`=`diffeq*diffeq`,`rec+rec`):
cauchyproduct := proc(rec1,rec2,uofn)
local y, z;
    RETURN(diffeqtorec(`diffeq*diffeq`(
        rectodiffeq(rec1,uofn,y(z)),rectodiffeq(rec2,uofn,y(z)),y(z)),
        y(z),uofn))
end: # cauchyproduct

# rec*rec
#Input: two linear recurrences rec1 and rec2 in the variable uofn (u(n))
#Output: a linear recurrence satisfied by the product of a solution of
#               rec1 by a solution of rec2.
`rec*rec` := proc(Eq1,Eq2,uofn)
local u, n, A,d1,d2,d,i,j,f,g,eq1,eq2,ini1, ini2,e1,eq1hom,eq2hom,maxinds1,maxinds2, eq,ini,k,
    eq1:=formatrec([Eq1,uofn],'u','n','ini1'); d1:=nops(eq1)-2;
    ini1:=remove(type,`goodinitvalues/rec`(eq1,u,n,ini1),`gfun/identity`);
```

```
eq2:=formatrec([Eq2,uofn],'u','n','ini2'); d2:=nops(eq2)-2;
ini2:=remove(type,`goodinitvalues/rec`(eq2,u,n,ini2),`gfun/identity`);
userinfo(2,gfun,`computing the product of two holonomic sequences of order`
    ,d1,`and`,d2);
eq1hom:=evalb(op(1,eq1)=0); eq2hom:=evalb(op(1,eq2)=0);
if eq1hom then 0 else d2 fi; if eq2hom then 0 else d1 fi;
d:=d1*d2+"+""; # maximal order of the product
if d=0 then
    eq:=op(2,eq1)*op(2,eq2)*u(n)-op(1,eq1)*op(1,eq2);
    eq1:=subs(n=n-1,[seq(-eq1[i]/eq1[d1+2],i=1..d1+1)]);
    eq2:=subs(n=n-1,[seq(-eq2[i]/eq2[d2+2],i=1..d2+1)]);
elif d1=0 then
    w:=-op(2,eq1)/op(1,eq1);
    eq:=numer(normal(op(1,eq2)+convert([seq(subs(n=n+i-1,w)*op(i+1,eq2)*
        u(n+i-1),i=1..d2+1)],`+`)));
    eq1:=subs(n=n-1,[seq(-eq1[i]/eq1[d1+2],i=1..d1+1)]);
    eq2:=subs(n=n-1,[seq(-eq2[i]/eq2[d2+2],i=1..d2+1)]);
elif d2=0 then
    w:=-op(2,eq2)/op(1,eq2);
    eq:=numer(normal(op(1,eq1)+convert([seq(subs(n=n+i-1,w)*op(i+1,eq1)*
        u(n+i-1),i=1..d1+1)],`+`)));
    eq1:=subs(n=n-1,[seq(-eq1[i]/eq1[d1+2],i=1..d1+1)]);
    eq2:=subs(n=n-1,[seq(-eq2[i]/eq2[d2+2],i=1..d2+1)]);
else
A := array(1..d+1,1..d,sparse);
eq1:=subs(n=n-1,[seq(-eq1[i]/eq1[d1+2],i=1..d1+1)]);
eq2:=subs(n=n-1,[seq(-eq2[i]/eq2[d2+2],i=1..d2+1)]);
# u(n+i)*v(n+j) -> column i*d2+j+1, 0<=i<d1, 0<=j<d2
if not eq2hom then e1:=d1 else e1:=0 fi;
A[1,1]:=1;
for k from 2 to d+1 do
    #u_n v_n  u_{n+d1-1} v_{n+d2-1}
    A[k,1]:=A[k-1,d1*d2]*eq1[2]*eq2[2];
    for j to d2-1 do # u_n v_{n+j}
        A[k,j+1]:=A[k-1,(d1-1)*d2+j]*eq1[2] # u_{n+d1-1}v_{n+j-1}
            +A[k-1,d1*d2]*eq1[2]*eq2[j+2]   # u_{n+d1-1}v_{n+d2-1}
    od;
    for i to d1-1 do # u_{n+i} v_n
        A[k,i*d2+1]:=A[k-1,i*d2]*eq2[2]     # u_{n+i-1} v_{n+d2-1}
            +A[k-1,d1*d2]*eq1[i+2]*eq2[2];  # u_{n+d1-1}v_{n+d2-1}
        for j to d2-1 do # u_{n+i} v_{n+j}
            A[k,i*d2+j+1]:=A[k-1,(i-1)*d2+j]# u_{n+i-1} v_{n+j-1}
                +A[k-1,i*d2]*eq2[j+2]       # u_{n+i-1} v_{n+d2-1}
                +A[k-1,(d1-1)*d2+j]*eq1[i+2]# u_{n+d1-1}v_{n+j-1}
                +A[k-1,d1*d2]*eq1[i+2]*eq2[j+2]# u_{n+d1-1}v_{n+d2-1}
        od
    od;
    for i to d1*d2 do A[k,i]:=collect(subs(n=n+1,A[k,i]),n) od;
od;
if not eq2hom then # u_{n+i} -> column d1*d2+i+1, 0<=i<d1
    for k from d2 to d+1 do # u_n
        A[k,d1*d2+1]:=
            A[k-1,d1*d2+1]*eq1[2]*eq2[1]    # u_{n+d1-1} v_{n+d2-1}
            +A[k-1,(d1+1)*d2]*eq1[2];       # u_{n+d1-1}
        for i to d1-1 do # u_{n+i}
            A[k,d1*d2+i+1]:=A[k-1,d1*d2+i]  # u_{n+i-1}
```

```
                    +A[k-1,(d1+1)*d2]*eq1[i+2]  # u_{n+d1-1}
                    +A[k-1,i*d2]*eq2[1]         # u_{n+i-1} v_{n+d2-1}
                    +A[k-1,d1*d2]*eq1[i+2]*eq2[1]# u_{n+d1-1}v_{n+d2-1}
            od;
            for i from d1*d2+1 to d1*d2+d1 do
                A[k,i]:=collect(subs(n=n+1,A[k,i]),n) od;
        od;
    fi;
    if not eq1hom then # v_{n+i} -> column d1*d2+e1+i+1, 0<=i<d2
        for k from d1 to d+1 do # v_n
            A[k,d1*d2+e1+1]:=
                A[k-1,(d1+1)*d2+e1]*eq2[2]      # v_{n+d2-1}
                +A[k-1,d1*d2]*eq1[1]*eq2[2];    # u_{n+d1-1} v_{n+d2-1}
            for i to d2-1 do # v_{n+i}
                A[k,d1*d2+e1+i+1]:=
                    A[k-1,d1*d2+e1+i]           # v_{n+i-1}
                    +A[k-1,(d1-1)*d2+i]*eq1[1]  # u_{n+d1-1} v_{n+i-1}
                    +A[k-1,(d1+1)*d2+e1]*eq2[i+2]# v_{n+d2-1}
                    +A[k-1,d1*d2]*eq1[1]*eq2[i+2]# u_{n+d1-1} v_{n+d2-1}
            od;
            for i from d1*d2+e1+1 to d1*d2+e1+d2 do
                A[k,i]:=collect(subs(n=n+1,A[k,i]),n) od;
        od
    fi;
    eq:=lindep(A,[seq(u(n+i),i=0..d)],n);
    fi;
    maxinds1:=max(op(map(op,indets(ini1,u(anything)))));
    maxinds2:=max(op(map(op,indets(ini2,u(anything)))));
    if maxinds1<>-infinity and maxinds1>=d1-1 then
        for i from maxinds1+1 to max(d-1,maxinds2) do
            ini1:=ini1 union {u(i)=subs([n=i-d1+1,op(ini1)],
                convert([eq1[1],seq(eq1[j+1]*u(i-j),j=1..d1)],`+`))} od fi;
    if maxinds2<>-infinity and maxinds2>=d2-1 then
        for i from maxinds2+1 to max(d-1,maxinds1) do
            ini2:=ini2 union {u(i)=subs([n=i-d2+1,op(ini2)],
                convert([eq2[1],seq(eq2[j+1]*u(i-j),j=1..d2)],`+`))} od fi;
    ini:=subs(subs(u=f,ini1) union subs(u=g,ini2),
        {seq(i=subs(u=f,i)*subs(u=g,i),
        i={seq(op(1,i),i=ini1)} intersect {seq(op(1,i),i=ini2)})});
    if ini={} then RETURN(eq) else RETURN({eq,op(ini)}) fi
end: # `rec*rec`

# hadamardproduct
#Input: two linear differential equations eq1 and eq2 in the variable yofz
#Output: a linear differential equation satisfied by the Hadamard product
# of any solution of eq1 with any solution of eq2.
hadamardproduct := proc(eq1,eq2,yofz)
local u, n;
    rectodiffeq(`rec*rec`(diffeqtorec(eq1,yofz,u(n)),
        diffeqtorec(eq2,yofz,u(n)),u(n)),u(n),yofz)
end: # hadamardproduct

# algebraicsubs
#Input: a linear differential equation Deq in the variable yofz (which is y(z))
#       a polynomial eq in y and z
#Output: a linear differential equation satisfied by f(y(z)) for any solution
```

```
# f of Deq and y of eq.
algebraicsubs := proc(Deq,eq,yofz)
local y, z, deq, P, u, d, i, d1, k, A, C, Dg, g, j, ord_eqn, c, F, f, inhomog,
invtoto, toto, eqn;
    P:=formatpoleq([eq,yofz],'y','z'); d:=degree(P,y);
    deq:=subs(z=y,formatdiffeq([Deq,yofz])); d1:=nops(deq)-2;
    g:=mygcdex(diff(P,y),P,y,'u');
    if has(g,y) then RETURN(algebraicsubs(Deq,normal(P/g),yofz)) fi;
    Dg:=rem(-u/g*diff(P,z),P,y);
    g:=mygcdex(deq[d1+2],P,y,'u');
    if has(g,y) then RETURN(algebraicsubs(Deq,normal(P/g),yofz)) fi;
    deq:=map(rem,[seq(-deq[i]*u/g,i=1..d1+1)],P,y);
    inhomog:=evalb(op(1,deq)<>0);
    if inhomog then ord_eqn:=d*d1 else ord_eqn:=d*(d1+1) fi;
    deq:=convert([op(1,deq),seq(deq[i]*F^(i-1),i=2..nops(deq))],`+`);
    eqn[0]:=f@y;
    toto:=seq((D@@i)(f)@y=F^(i+1),i=0..d1-1),D(z)=1;
    invtoto:=seq(F^(d1-i)=(D@@(d1-i-1))(f)@y,i=0..d1-1);
    for k to ord_eqn do
        eqn[k]:=subs(invtoto,
            rem(subs([(D@@d1)(f)@y=deq,D(y)=Dg,toto],D(eqn[k-1])),P,y)) od;
    for k from 0 to ord_eqn do eqn[k]:=collect(subs(toto,eqn[k]),[F,y]) od;
    A:=array(1..ord_eqn+1,1..ord_eqn,sparse);
    C:=array(1..ord_eqn+1,sparse);
    # f^{(i)}(g).g^j (i.e. F^i y^j) -> column j*d1+i+1, 0<=i<d1, 0<=j<d
    A[1,1]:=1;
    for k to ord_eqn+1 do for i from 0 to d1-1 do
        c:=coeff(eqn[k-1],F,i+1);
        for j from 0 to d-1 do A[k,j*d1+i+1]:=coeff(c,y,j) od od
    od;
    if inhomog then # g^j -> column d*d1+j, 0<j<d; g^0 -> C
        for k to ord_eqn+1 do
            c:=coeff(eqn[k-1],F,0);
            for j to d-1 do A[k,d*d1+j]:=coeff(c,y,j) od;
            C[k]:=coeff(c,y,0)
        od
    fi;
    RETURN(lindep(A,[seq((D@@i)(y)(z)-C[i+1],i=0..ord_eqn)],z))
end: # algebraicsubs

########################## Help ####################################

`help/text/gfun`:= TEXT(
`    `,
`HELP FOR: The generating function package`,
`    `,
`CALLING SEQUENCE:`,
`    <function>(args)`,
`    gfun[<function>](args)`,
`    `,
`SYNOPSIS:`,
`- The gfun package contains functions that help handle generating and`,
`holonomic functions.`,
`    `,
`- The functions available are:`,
`    `,
```

```
`  algebraicsubs       algeqtodiffeq      borel              cauchyproduct`,
`  diffeq+diffeq       diffeq*diffeq      diffeqtorec        guesseqn`,
   guessgf             hadamardproduct    invborel           listtoalgeq`,
   listtodiffeq        listtohypergeom    listtolist         listtoratpoly`,
`  listtorec           listtoseries       ratpolytocoeff     rec+rec`,
`  rec*rec             rectodiffeq        rectoproc          seriestoalgeq`,
`  seriestodiffeq      seriestohypergeom  seriestolist       seriestoratpoly`,
`  seriestorec         seriestoseries`,
`    `,
`- For help with a particular function do: ?gfun[<function>], where`,
`<function> is taken from the above list.`,
`    `,
`- An example of using a function from the gfun package is the following.`,
`To compute the generating function of the Fibonacci numbers one`,
`would use the command with(gfun,guessgf) followed by`,
`guessgf([1,1,2,3,5,8,13],x).`,
`    `,
`- Information about the computations that are being done can be obtained by`,
`setting infolevel['gfun'] to anything between 1 and 5.`,
`    `,
`- More information can be found in`,
`    "Gfun: a Maple package for the manipulation of generating and holonomic`,
`     functions in one variable."`,
`    B. Salvy and P. Zimmermann, INRIA Technical Report 143, November 1992.`,
`    `,
`- For any comment or information about new versions, if you have access to`,
`e-mail, use the address gfun@inria.fr.`,
`  `,
`SEE ALSO: with, gfun[parameters].`):

help/gfun/text/parameters`:=TEXT(
    `,
`HELP FOR: gfun parameters`,
`    `,
`CALLING SEQUENCE:`,
`    gfun['maxordereqn']                gfun['minordereqn']`,
`    gfun['maxdegcoeff']                gfun['mindegcoeff']`,
`    gfun['maxdegeqn']                  gfun['mindegeqn']`,
`    gfun['optionsgf']`,
`    `,
`SYNOPSIS:`,
`- These "global" variables control the actions of various functions of the`,
`gfun package.`,
`    `,
`- For instance, when listtodiffeq is used, only those linear differential`,
`equations whose order lies between gfun['minordereqn'] and gfun['maxordereqn']`,
`and whose coefficients have a degree lying between gfun['mindegeqn'] and`,
`gfun['maxdegeqn'] are tried.`,
`    `,
`- optionsgf determines what types of generating functions are to be tried`,
`by the guessing part of gfun, and in which order. Its default is`,
`['ogf','egf'].`,
`    `,
`- These parameters can be changed like any Maple variable. The quotes are only`,
`necessary if you have loaded the package with the function with, or if one `,
`of your personal variables is called maxordereqn, maxdegcoeff,...`
```

```
):

`help/gfun/text/maxordereqn`:=`help/gfun/text/parameters`:
`help/gfun/text/minordereqn`:=`help/gfun/text/parameters`:
`help/gfun/text/maxdegeqn`:=`help/gfun/text/parameters`:
`help/gfun/text/mindegeqn`:=`help/gfun/text/parameters`:
`help/gfun/text/maxdegcoeff`:=`help/gfun/text/parameters`:
`help/gfun/text/mindegcoeff`:=`help/gfun/text/parameters`:

`help/gfun/text/listtoalgeq`:=TEXT(
`    `,
`FUNCTIONS: gfun[listtoalgeq]    - find an algebraic equation for`,
`           gfun[seriestoalgeq]    generating function`,
`    `,
`CALLING SEQUENCES:`,
`      listtoalgeq(l, y(x), <[typelist]>)`,
`    seriestoalgeq(s, y(x), <[typelist]>)`,
`    `,
`PARAMETERS:`,
`    l - a list,`,
`    s - a series,`,
` y(x) - the unknown function and its variable`,
` [typelist] - (optional) a list of generating function types`,
`    `,
`SYNOPSIS:`,
`- The procedures listtoalgeq and seriestoalgeq compute a polynomial`,
`equation in y and x satisfied by the generating function y(x) of the`,
`expressions in l or s, this generating function being of one of the types`,
`specified by typelist. These types must correspond to existing procedures`,
`listtoseries/typename (see listtoseries).`,
`    `,
`- If typelist contains more than one element, these types are tried in`,
`order.`,
`    `,
`- If typelist is not provided, a default optionsgf=['ogf','egf'] is used.`,
`    `,
`- The output is a list whose first element is the polynomial in y(x) and x`,
`that was found, and whose second element is the type to which it corresponds.`,
`    `,
`- In the current implementation, the maximal degree in y is 6 and the `,
`maximum degree of the coefficients is 3. This can be changed by modifying`,
`the variables gfun['maxdegeqn'] and gfun['maxdegcoeff'].`,
`    `,
`- One should give as many terms as possible in the list l or the series s.`,
`    `,
`- If sufficiently many terms were given, and no solution was found, it`,
`means that the generating function does not satisfy any algebraic equation`,
`of degree less or equal to gfun['maxdegeqn'] with coefficients of degree`,
`less or equal to gfun['maxdegcoeff'].`,
`    `,
`EXAMPLES:`,
`> with(gfun):`,
`> l:=[1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786]:`,
`> listtoalgeq(l,y(x));`,
`                                          2`,
`                          [1 - y(x) + x y(x) , ogf]`,
```

```
`    `,
`> s:=series((1-sqrt(1-4*x)),x,9);     `,
                    2       3       4       5       6       7       8       9`,
`    s := 2 x + 2 x  + 4 x  + 10 x  + 28 x  + 84 x  + 264 x  + 858 x  + O(x )`,
`    `,
`> seriestoalgeq(s,y(x));            `,
`                                                  2`,
`                        [x - 1/2 y(x) + 1/4 y(x) , ogf]`,
`    `,
`SEE ALSO: gfun.`):

`help/gfun/text/seriestoalgeq`:=`help/gfun/text/listtoalgeq`:

`help/gfun/text/listtorec`:=TEXT(
`    `,
`FUNCTIONS: gfun[listtorec]   - find a recurrence for the elements`,
`           gfun[seriestorec]`,
`    `,
`CALLING SEQUENCES:`,
`    listtorec(l, u(n), <[typelist]>)`,
`  seriestorec(s, u(n), <[typelist]>)`,
`    `,
`PARAMETERS:`,
`       l  - a list,`,
`       s  - a series,`,
`     u(n) - the unknown function and its variable`,
` typelist - (optional) a list of generating function types`,
`    `,
`SYNOPSIS:`,
`- The procedures listtorec and seriestorec compute a linear recurrence with`,
`polynomial coefficients satisfied by the expressions in l or s, with`,
`a normalization specified by typelist. These types must correspond to`,
`existing procedures listtoseries/typename (see listtoseries).`,
`    `,
`- If typelist contains more than one element, these types are tried in`,
`order.`,
`    `,
`- If typelist is not provided, a default optionsgf=['ogf','egf'] is used.`,
`    `,
`- The output is a list whose first element is a set containing the`,
`recurrence and its initial conditions, and whose second element is the`,
`type to which it corresponds.`,
`    `,
`- One should give as many terms as possible in the list l or the series s.`,
`    `,
`EXAMPLE:`,
`> with(gfun):`,
`> l:=[1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786]:`,
`> listtorec(l,u(n));`,
`    `,
`              [{(- 2 - 4 n) u(n) + (n + 2) u(n + 1), u(0) = 1}, ogf]`,
`    `,
`> rsolve(op(1,"),u(n));`,
`    `,
`                                  n`,
`                                 4  GAMMA(n + 1/2)`,
```

```
`                                  ------------------`,
`                                                 1/2`,
                                   GAMMA(n + 2) Pi`,
    `,
`SEE ALSO: gfun.`):

`help/gfun/text/seriestorec`:=`help/gfun/text/listtorec`:

`help/gfun/text/listtoratpoly`:=TEXT(
`    `,
`FUNCTIONS: gfun[listtoratpoly]  - find a rational generating function`,
`           gfun[seriestoratpoly]`,
`    `,
`CALLING SEQUENCE:`,
`    listtoratpoly(l, x, <[typelist]>)`,
`  seriestoratpoly(s, <[typelist]>)`,
`    `,
`PARAMETERS:`,
`    l - a list,`,
`    s - a series,`,
`    x - the unknown variable,`,
`    [typelist] - (optional) a list of generating function types`,
`    `,
`SYNOPSIS:`,
`- The procedures listtoratpoly and seriestoratpoly compute a rational`,
`function in x for the generating function of the expressions in l or s,`,
`this generating function being of one of the types specified by typelist.`,
`These types must correspond to procedures listtoseries/typename`,
`(see listtoseries).`,
`    `,
`- These functions are frontends to convert[ratpoly] which performs the`,
`actual computation.`,
`    `,
`- If typelist contains more than one element, these types are tried in`,
`order.`,
`    `,
`- If typelist is not provided, a default optionsgf=['ogf','egf'] is used.`,
`    `,
`- The output is a list whose second element is the type for which a`,
`solution was found, and whose first element is the rational function.`,
`    `,
`- One should give as many terms as possible in the list l or the series s.`,
`    `,
`EXAMPLE:`,
`> with(gfun):`,
`> l:=[1,1,2,3,5,8,13];`,
`    `,
`                          l := [1, 1, 2, 3, 5, 8, 13]`,
`    `,
`> listtoratpoly(l,x);`,
`    `,
`                                     1`,
`                              [- ------------, ogf]`,
`                                            2`,
`                                 - 1 + x + x`,
`    `,
```

```
`SEE ALSO: gfun, convert[ratpoly].`):

`help/gfun/text/seriestoratpoly`:=`help/gfun/text/listtoratpoly`:

`help/gfun/text/listtohypergeom`:=TEXT(
`    `,
`FUNCTIONS: gfun[listtohypergeom] - find an hypergeometric generating function`,
`         gfun[seriestohypergeom]`,
`    `,
`CALLING SEQUENCE:`,
`    listtohypergeom(l, x, <[typelist]>)`,
`  seriestohypergeom(s, <[typelist]>)`,
`    `,
`PARAMETERS:`,
`     l - a list,`,
`     s - a series,`,
`     x - the unknown variable,`,
`    [typelist] - (optional) a list of generating function types`,
`    `,
`SYNOPSIS:`,
`- The procedures listtohypergeom and seriestohypergeom compute a 2F1 in x`,
`for the generating function of the expressions in l or s, this generating`,
`function being of one of the types specified by typelist. These types must`,
`correspond to procedures listtoseries/typename (see listtoseries).`,
`    `,
`- The 2F1 that are found have singular points at 0 and infinity but not`,
`necessarily at 1.`,
`    `,
`- If typelist contains more than one element, these types are tried in`,
`order.`,
`    `,
`- If typelist is not provided, a default optionsgf=['ogf','egf'] is used.`,
`    `,
`- The output is a list whose second element is the type for which an`,
`equation was found, and whose first element is the hypergeometric function.`,
`    `,
`- One should give at least 6 terms in the list l or the series s.`,
`    `,
`EXAMPLE:`,
`> with(gfun):`,
`> l:=[2,5,14,42,132,429,1430];`,
`    `,
`                          l := [2, 5, 14, 42, 132, 429, 1430]`,
`    `,
`> listtohypergeom(l,x);`,
`    `,
`                      x                          3                 3`,
` [- 16 ------------------------------ - -------------- + ---------------`,
`                 1/2                 1/2 3            1/2               1/2  2`,
`       (1 - 4 x)    (1 + (1 - 4 x)    )    (1 - 4 x)    x   (1 - 4 x)    x`,
`     `,
`                 1                1 - 4 x`,
`       - ----------------- + 1/2 -------,                                    ogf]`,
`                    1/2  3           3`,
`         2 (1 - 4 x)    x           x`,
`    `,
```

```
`SEE ALSO: gfun, gfun[guessgf].`):

`help/gfun/text/seriestohypergeom`:=`help/gfun/text/listtohypergeom`:

`help/gfun/text/listtodiffeq`:=TEXT(
`    `,
`FUNCTIONS: gfun[listtodiffeq]     - find a linear differential equation`,
`           gfun[seriestodiffeq]    for the generating function`,
`    `,
`CALLING SEQUENCES:`,
`    listtodiffeq(l, y(x),<[typelist]>)`,
`  seriestodiffeq(s, y(x),<[typelist]>)`,
`    `,
`PARAMETERS:`,
`    l   - a list,`,
`    s   - a series,`,
`   y(x) - the unknown function and its variable`,
`   [typelist] - (optional) a list of generating function types`,
`    `,
`SYNOPSIS:`,
`- The procedures listtodiffeq and seriestodiffeq compute a linear`,
`differential equation in y(x) with polynomial coefficients in x satisfied`,
`by the generating function y(x) of the expressions in l or s, this`,
`generating function being of one of the types specified by typelist. These`,
`types must correspond to procedures listtoseries/typename (see listtoseries).`,
`    `,
`- If typelist contains more than one element, these types are tried in`,
`order.`,
`    `,
`- If typelist is not provided, a default optionsgf=['ogf','egf'] is used.`,
`    `,
`- The output is a list whose second element is the type for which an`,
`equation was found, and whose first element is the differential equation`,
`satisfied by the generating function.`,
`    `,
`- In the current implementation, the maximal order is 2 and the `,
`maximum degree of the coefficients is 3. This can be changed by modifying`,
`the variables gfun['maxordereqn'] and gfun['maxdegcoeff'].`,
`    `,
`- One should give as many terms as possible in the list l or the series s.`,
`    `,
`- If sufficiently many terms were given, and no solution was found, it`,
`means that the generating function does not satisfy any linear differential`,
`equation of order less or equal to gfun['maxordereqn'] with coefficients`,
`of degree less or equal to gfun['maxdegcoeff'].`,
`    `,
`EXAMPLE:`,
`> with(gfun):`,
`> l:=[1,2,6,22,91,408,1938,9614,49335,260130,1402440,7702632,42975796,243035536`,
`> ,1390594458,8038677054,46892282815,275750636070,1633292229030,9737153323590]:`,
`> listtodiffeq(l,y(x));`,
`                                                          2  /  d      \\`,
`[{D(y)(0) = 2, y(0) = 1, 1 + (- 1 + 5 x) y(x) + (- 3/2 x + 9 x ) |---- y(x)|`,
`                                                                \\ dx     /`,
`    `,
`                                                    /   2      \\`,
```

```
`                                        2         3  |  d        |`,
`                              + (- 1/3 x  + 9/4 x ) |----- y(x)|              },`,
                                                     |   2       |`,
                                                     \\ dx       /`,
`    `,
`    ogf]`,
`    `,
`> s:=series(exp(x)/sqrt(1-x),x,7);`,
`    `,
`                                2    53   3   115  4   2947  5   31411  6      7`,
`    s := 1 + 3/2 x + 11/8 x  + ---- x  + --- x  + ---- x  + ----- x  + O(x )`,
`                                 48       128      3840      46080`,
`> seriestodiffeq(s,y(x));            `,
`    `,
`                                                    /  d      \\`,
`           [{y(0) = 1, (3/2 - x) y(x) + (- 1 + x) |---- y(x)|}, ogf]`,
`                                                    \\ dx      /`,
`    `,
`SEE ALSO: gfun, gfun[parameters].`):

`help/gfun/text/seriestodiffeq`:=`help/gfun/text/listtodiffeq`:

`help/gfun/text/listtoseries`:=TEXT(
`    `,
`FUNCTIONS: gfun[listtoseries]   - convert a list into a series`,
`           gfun[seriestolist]   - convert a series into a list`,
`           gfun[listtolist]     - convert a list into a list`,
`           gfun[seriestoseries] - convert a series into a series`,
`    `,
`CALLING SEQUENCES:`,
`    listtoseries(l, x, gf)`,
`      listtolist(l, gf)`,
`    seriestolist(s, gf)`,
`  seriestoseries(s, gf)`,
`    `,
`PARAMETERS:`,
`   l     - a list`,
`   s     - a series`,
`   x     - a name`,
`   gf    - (optional)`,
`    `,
`SYNOPSIS:`,
`- These procedures take as input a list or a series and yield a list or`,
`a series, according to their name.`,
`    `,
`- Lists are viewed as lists of coefficients of power series and reciprocally`,
`series are viewed are generating series of lists of coefficients.`,
`    `,
`- By default, listtoseries creates a power series whose coefficients are the`,
`elements of the list given as argument. Similarly, seriestolist returns a`,
`list of the coefficients of the series given as argument. The other two`,
`procedures, listtolist and seriestoseries, default to identity.`,
`    `,
`- When a third argument is given, it is treated as a type of generating`,
`function. The coefficients of the output are those of the corresponding`,
`generating function of the coefficients of the input.`,
```

```
`    `,
`- The following types of generating functions are known:`,
    `,
            ogf revogf  lgdogf  laplace`,
`           egf revegf  lgdegf.`,
`    `,
`- If type is 'ogf' (ordinary generating function), then the coefficients are`,
`the elements of l.`,
`    `,
`- If type is 'egf' (exponential generating function), then the ith coefficient`,
`is op(i,l)/i!.`,
`    `,
`- If type is 'revogf', then the series is the reciprocal of the ordinary `,
`generating function.`,
`    `,
`- If type is 'revegf', then the series is the reciprocal of the exponential`,
`generating function.`,
`    `,
`- If type is 'lgdogf', then the series is the logarithmic derivative of the`,
`ordinary generating function.`,
`    `,
`- If type is 'lgdegf', then the series is the logarithmic derivative of the`,
`exponential generating function.`,
`    `,
`- If type is 'laplace', then then the ith coefficient is op(i,l)*i!.`,
`    `,
`- The user can define his own type by creating a procedure`,
`gfun[``listtoseries/mytypeofgf``], which takes a list and a variable as input,`,
`and yields a series in this variable.`,
`    `,
`XAMPLE:`,
`> with(gfun):`,
`> l:=[1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786]:`,
`> listtoseries(l,x,'egf');`,
`                2         3          4          5    11   6    143  7    143  8`,
`     1 + x + x  + 5/6 x  + 7/12 x  + 7/20 x  + ---- x  + ---- x  + ---- x`,
`                                                 60       1680      4032`,
`    `,
`              2431   9    4199   10     4199   11      12`,
`          + ------ x  + ------ x   + ------- x   + O(x  )`,
`            181440      907200       2851200`,
`    `,
`SEE ALSO: series, convert/series, solve, gfun.`):

`help/gfun/text/listtolist`:=`help/gfun/text/listtoseries`:
`help/gfun/text/seriestolist`:=`help/gfun/text/listtoseries`:
`help/gfun/text/seriestoseries`:=`help/gfun/text/listtoseries`:

`help/gfun/text/guessgf`:=TEXT(
`    `,
`FUNCTION: gfun[guessgf] -  find a generating function from a list`,
`          gfun[guesseqn] - find a differential equation satisfied by it`,
`    `,
`CALLING SEQUENCE:`,
`    guessgf(L, x, <[typelist]>);`,
`    guesseqn(L, y(x), <[typelist]>);`,
```

```
`    `,
`PARAMETERS:`,
`    L    - a list,`,
`    x    - a name,`,
`    y    - a name,`,
`    [typelist] - (optional) a list of generating function types`,
`    `,
`SYNOPSIS:`,
`- The procedure guessgf attempts to find a closed form for the generating`,
`function specified by (L, typelist) (see gfun[listtoseries] for explanations`,
`about typelist), in the variable x.`,
`    `,
`- If typelist contains more than one element, these types are tried in`,
`order.`,
`    `,
`- If typelist is not provided, a default optionsgf=['ogf','egf'] is used.`,
`    `,
`- This function first tries to find a rational function by listtoratpoly,`,
`then calls listtohypergeom to find some hypergeometric functions,`,
`and then listtodiffeq to find a linear differential equation with polynomial`,
`coefficients which is then passed to dsolve.`,
`    `,
`- The function guesseqn only tries to find an equation satisfied by the`,
`generating function. It might succeed where guessgf fails because it does`,
`not attempt to solve this equation in closed-form.`,
`    `,
`- One should give as many terms as possible in the list L.`,
`    `,
`EXAMPLE:`,
`> with(gfun):`,
`> guessgf([1,2,4,7,11,16,22],x);`,
`                                          2`,
`                                 1 - x + x`,
`                              [- ----------, ogf]`,
`                                         3`,
`                                  (x - 1)`,
``,

`> guessgf([1,1,3,10,41,196],l,x,['lgdegf']);`,
``,
`                           [exp(x) + exp(x) x, lgdegf]`,
`> l:=[1, 4, 36, 400, 4900, 63504, 853776, 11778624, 165636900, 2363904400,`,
`34134779536, 497634306624, 7312459672336]:`,
`> guesseqn(l,y(z));`,
``,
`    [{y(0) = 1, D(y)(0) = 4,`,
``,
`                                                          /   2       \\`,
`                                /  d       \\            2  |  d        |`,
`         - 4 y(z) + (1 - 32 z) |---- y(z)| + (z - 16 z ) |----- y(z)|}, ogf]`,
`                                \\ dz      /                |   2       |`,
`                                                          \\ dz        /`,
`SEE ALSO: gfun.`):
`help/gfun/text/guesseqn`:=`help/gfun/text/guessgf`:

`help/gfun/text/ratpolytocoeff`:=TEXT(
```

```
`    `,
`FUNCTION: gfun[ratpolytocoeff] - computes the nth coefficient of a`,
                                  rational function`,
    `,
`CALLING SEQUENCE:`,
`    ratpolytocoeff(f, x, n);`,
`    `,
`PARAMETERS:`,
`    f    - a rational function in x,`,
`    x,n  - names.`,
`    `,
`SYNOPSIS:`,
`- The procedure ratpolytocoeff computes the expression for the nth`,
`coefficient of the Taylor expansion about the origin of f as a function`,
`of x.`,
`    `,
`- This procedure requires the procedure fullparfrac from the share library.`,
``,
`EXAMPLE:`,
`> with(gfun):`,
`> ratpolytocoeff(1/(1-x-x^2),x,n);`,
`    `,
`                     -----                                  (- n)`,
`                      \\         (- 1/5 - 2/5 alpha) alpha`,
`                       )      - ------------------------------`,
`                      /                       alpha`,
`                     -----`,
`                  alpha = %1`,
`    `,
`                                                  2`,
`%1 :=                      RootOf(- 1 + _Z + _Z )`,
`    `,
`SEE ALSO: gfun.`):

`help/gfun/text/algeqtodiffeq` :=
TEXT(`FUNCTION: gfun[algeqtodiffeq] - compute a differential equation for`,
`                                  an algebraic function`,
`   `,
`CALLING SEQUENCE: algeqtodiffeq(p,y(z),inits)`,
`   `,
`PARAMETERS:`,`   p - a polynom in y and z (or a polynomial equation)`,
`   y - the name of the algebraic function`,
`   z - the generic variable`,`   `,
`   inits - (optional) a set of initial conditions`,
`    `,
`SYNOPSIS:   `,
`- The polynom p defines an algebraic function, RootOf(p,y) in Maple terms.`,
`  This procedure computes a linear differential equation with polynomial`,
`  coefficients verified by the function y(z). This equation is of order at`,
`  most degree(p,y)-1.`,
``,
`- The output contains initial conditions in zero (y(0), D(y)(0), and so on),`,
`  and can thus be given directly to dsolve. In general, y(0) is a RootOf a`,
`  polynom, D(y)(0) a rational expression in y(0), (D@@2)(y)(0) a rational`,
`  expression in y(0),D(y)(0), and so on.`,
``,
```

```
`- When the algebraic function defined by the polynom is not defined in z=0,`,
`  the procedure returns NULL.`,
   `,
`EXAMPLES:`,
`> with(gfun):`,
`> algeqtodiffeq(y=1+z*y^2,y(z));`,
`    `,
`                                                  2`,
`            {1 + (- 1 + 2 z) y(z) + (- z + 4 z ) D(y)(z), y(0) = 1}`,
`    `,
`> algeqtodiffeq(56*a^3+7*a^3*y^3-14*y*z,y(z),{y(0)=-2});`,
`    `,
`                                2            9       3    (2)`,
`      {- y(z) z + 3 D(y)(z) z  + (- 108 a  + 2 z ) D    (y)(z), y(0) = -2,`,
`    `,
`                          1`,
`          D(y)(0) = - ----}`,
`                           3`,
`                        3 a`,
`    `,
`> algeqtodiffeq(y^5*(1-x)=1,y(x),{y(0)=1});`,
`    `,
`                      {y(x) + (- 5 + 5 x) D(y)(x), y(0) = 1}`,
`    `,
`> dsolve(",y(x));`,
``,
`                                          1`,
`                              y(x) = - ------------`,
`                                            1/5`,
`                                     (- 1 + x)`,
`    `,
`SEE ALSO: dsolve, gfun[diffeqtorec]`):

`help/gfun/text/diffeqtorec` := TEXT(
`FUNCTION: gfun[diffeqtorec] - convert a linear differential equation into`,
`                                 a recurrence`,
``,
`CALLING SEQUENCE: diffeqtorec(deqns,y(z),u(n))`,
``,
`PARAMETERS:`,
`   deqns - a linear differential equation with polynomial coefficients`,
`           or a set of one differential equation and initial conditions`,
`   y,z   - the name and the variable of the function`,
`   u,n   - the name and the index of the recurrence`,
``,
`SYNOPSIS:   `,
`- Let f be a power series solution of the differential equation. If u(n) is`,
`  the n-th Taylor coefficient of f around zero, the procedure outputs a`,
`  linear recurrence for the numbers u(n), with rational coefficients in n.`,
``,
`- The syntax is the same as that of dsolve.`,
``,
`- Combined with gfun[algeqtodiffeq], this procedure produces a linear`,
`  recurrence for the Taylor coefficients of an algebraic function.`,
``,
`EXAMPLES:`,
```

```
`> with(gfun):`,
`> diffeqtorec(y(z)=a*diff(y(z),z),y(z),v(n));`,
    `,
`                          v(n) + (- a n - a) v(n + 1)`,
`    `,
`> algeqtodiffeq(y=1+z*(y^2+y^3),y(z),{}):`,
`> diffeqtorec(",y(z),u(m));`,
`    `,
`  {u(2) = 10,`,
`    `,
`                 2                              2`,
`      (- m - 2 m ) u(m) + (- 9 - 30 m - 18 m ) u(m + 1)`,
`    `,
`                                 2                                  2`,
`           + (279 + 227 m + 46 m ) u(m + 2) + (- 26 m - 42 - 4 m ) u(m + 3),`,
`    `,
`      u(0) = 1, u(1) = 2}`,
`    `,
`SEE ALSO: gfun[algeqtodiffeq], gfun[rectodiffeq]`):

`help/gfun/text/rectoproc` := TEXT(
`FUNCTION: gfun[rectoproc] - convert a recurrence into a function`,
``,
`CALLING SEQUENCE: rectoproc(eqns, u(n),<remember>)`,
``,
`PARAMETERS:`,
`   eqns - a single equation or a set of equations`,
`   u,n  - the name and index of the recurrence`,
`   remember - (optional)`,
``,
`SYNOPSIS:`,
`- The procedure outputs a Maple procedure that, given a non negative integer`,
`  n as input, gives the n-th term u(n) of the linear recurrence.`,
``,
`- If the third argument remember is supplied, the procedure returned will`,
`  use option remember.`,
``,
`- When the coefficients of the recurrence are non constant polynomials`,
`  the returned procedure will run in linear number of arithmetic operations`,
`  and without wasting extra space if the remember option is not specified.`,
``,
`- When the coefficients are constant, the procedure will run in logarithmic`,
`  number of arithmetic operations and without wasting extra space if the`,
`  remember option is not specified.`,
``,
`- The third argument should be given each time one needs a large number of`,
`  values of the sequence.`,
``,
`- When the first terms of the recurrence are not explicitely supplied, they`,
`  are represented symbolically.`,
``,
`EXAMPLES:`,
`> with(gfun):`,
`> fib := rectoproc({f(i)=f(i-1)+f(i-2),f(0)=0,f(1)=1},f(i));`,
`fib := proc(n:nonnegint)`,
`local a,res,i,l;`,
```

```
`    a := array(1 .. 2,1 .. 2,[[1,1],[1,0]]);`,
`    res := array(1 .. 2,[1,0]);`,
     l := convert(n-1,base,2);`,
     if l[1] = 1 then res := array(1 .. 2,[1 = 1,2 = 1]) fi;`,
`    if l = [1] then RETURN(res[1]) fi;`,
`    for i in subsop(1 = (),nops(l) = (),l) do`,
`        a := array(1 .. 2,1 .. 2,[(2,2) = a[1,2]*a[2,1]+a[2,2]^2,`,
`            (1,2) = a[1,1]*a[1,2]+a[1,2]*a[2,2],(1,1) = a[1,1]^2+a[1,2]*a[2,1],`,
`            (2,1) = a[2,1]*a[1,1]+a[2,2]*a[2,1]]);`,
`        if i = 1 then`,
`            res := array(1 .. 2,`,
`              [1 = a[1,1]*res[1]+a[1,2]*res[2],2 = a[2,1]*res[1]+a[2,2]*res[2]]`,
`              )`,
`        fi`,
`    od;`,
`    RETURN(res[1]*a[1,1]^2+res[1]*a[1,2]*a[2,1]+res[2]*a[1,1]*a[1,2]+`,
`        res[2]*a[1,2]*a[2,2])`,
`end`,
`> fib(100);`,
``,
`                              354224848179261915075`,
``,
`> fib2 := rectoproc(f(i)=f(i-1)+f(i-2),f(i),remember);`,
`fib2 :=`,
`    proc(n:nonnegint) options remember; RETURN(procname(n-2)+procname(n-1)) end`,
`> fib2(100);`,
``,
`            354224848179261915075 f(1) + 218922995834555169026 f(0)`,
``,
`SEE ALSO: gfun[rectodiffeq]`):


`help/gfun/text/rectodiffeq` := TEXT(
`FUNCTION: gfun[rectodiffeq] - convert a linear recurrence into`,
`                                a differential equation`,
``,
`CALLING SEQUENCE: rectodiffeq(eqns, u(n), f(z))`,
``,
`PARAMETERS:`,
`   eqns - a single equation or a set of equations`,
`   u,n  - the name and index of the recurrence`,
`   f,z  - the name and variable of the function`,
``,
`SYNOPSIS:`,
`- Let f be the generating function associated to the sequence (u(n)):`,
`  f(z)=sum(u(n)*z^n,n=0..infinity). The procedure outputs a linear`,
`  differential equation with polynomial coefficients verified by f.`,
``,
`- The input syntax is the same as for rsolve: the first argument should be a`,
`  single recurrence relation or a set containing one recurrence relation and`,
`  boundary conditions. The recurrence relation should be linear in the`,
`  variable u, with polynomial coefficients in n. The terms of the sequence`,
`  appearing in the relation should be of the form u(n+k), with k integer.`,
``,
`- The output is either a single differential equation, or a set containing`,
`  a differential equation and initial conditions.`,
```

```
``,
`EXAMPLES:`,
`> with(gfun):`,
`> rectodiffeq({(5*n+10)*u(n)+k*u(n+1)-u(n+2),u(0)=0,u(1)=0},u(n),f(t));`,
``,
`                                        2                          3`,
`          {f(0) = 0, D(f)(0) = 0, (10 t  + k t - 1) f(t) + 5 t  D(f)(t)}`,
``,
`> diffeqtorec(",f(t),u(n));`,
``,
`          {(5 n + 10) u(n) + k u(n + 1) - u(n + 2), u(0) = 0, u(1) = 0}`,
``,
`> rectodiffeq((n-10)*u(n+1)-u(n),u(n),y(z));`,
``,
`  (10)                          (9)                            (2)`,
`{D    (y)(0) = 0, y(0) = 0, D    (y)(0) = 0, D(y)(0) = 0, D    (y)(0) = 0,`,
``,
`     (3)                (4)                (5)                (6)`,
`    D    (y)(0) = 0, D    (y)(0) = 0, D    (y)(0) = 0, D    (y)(0) = 0,`,
``,
`     (7)                (8)                                                (2)`,
`    D    (y)(0) = 0, D    (y)(0) = 0, - y(z) + (- z - 10) D(y)(z) + z D    (y)(z)}`,
``,
`> dsolve(",y(z));`,
``,
`                                                  11`,
`                                y(z) = exp(z) z    _C1`,
``,
`SEE ALSO: gfun[diffeqtorec]`):

help/gfun/text/borel` := TEXT(
`FUNCTION: gfun[borel] - compute the Borel transform of a generating function`,
``,
`CALLING SEQUENCE: borel(expr, a(n), t)`,
``,
`PARAMETERS:`,
`   expr - a linear recurrence with polynomial coefficients`,
`   a,n  - the name and index of the recurrence`,
`   t    - (optional) 'diffeq'`,
``,
`SYNOPSIS:`,
`- If (a(n),n=0..infinity) is the sequence of numbers defined by the recurrence`,
`  expr, the procedure computes the recurrence for the numbers a(n)/n!.`,
``,
`- If an optional argument 'diffeq' is given, expr is considered as a linear`,
`  differential equation with polynomial coefficients for the function a(n);`,
`  the procedure outputs a linear differential equation verified by the`,
`  Borel transform of a(n).`,
``,
`EXAMPLES:`,
`> with(gfun):`,
`> borel(a(n)=a(n-1)+a(n-2),a(n));`,
``,
`                                                  2`,
`              - a(n) + (- n - 1) a(n + 1) + (n  + 3 n + 2) a(n + 2)`,
``,
```

```
`> invborel(",a(n));`,
``,
`                                  - a(n) - a(n + 1) + a(n + 2)`,
`,
`> borel({a(n)=a(n-1)+a(n-2)+n,a(0)=0,a(1)=1},a(n));`,
``,
`  {a(2) = 3/2, a(0) = 0, a(1) = 1,`,
``,
`                                                    3        2`,
`      (n + 3) a(n) + (n + 1) a(n + 1) + (- 2 n  - 11 n  - 19 n - 10) a(n + 2)`,
``,
`                4       3        2`,
`           + (n  + 8 n  + 23 n  + 28 n + 12) a(n + 3)                        }`,
``,
`SEE ALSO: gfun[invborel]`):

`help/gfun/text/invborel` := TEXT(
`FUNCTION: gfun[invborel] - convert an exponential into ordinary recurrence`,
``,
`CALLING SEQUENCE: invborel(expr, a(n), t)`,
``,
`PARAMETERS:`,
`   expr - a linear recurrence with polynomial coefficients`,
`   a,n  - the name and index of the recurrence`,
`   t    - (optional) 'diffeq'`,
``,
`SYNOPSIS:`,
`- If (a(n),n=0..infinity) is the sequence of numbers defined by the recurrence`,
`  expr, the procedure computes the recurrence for the numbers n!*a(n).`,
``,
`  If the optional fourth argument 'diffeq' is given, expr is taken as a`,
`  linear differential equation with polynomial coefficients, and the`,
`  procedure outputs a linear differential equation satisfied by its inverse`,
`  Borel transform.`,
``,
`EXAMPLES:`,
`> with(gfun):`,
`> rec:={a(n)=a(n-1)+a(n-2),a(0)=0,a(1)=1}:`,
`> invborel(rec,a(n));`,
``,
`                                2`,
`  {a(0) = 0, a(1) = 1, (- n  - 3 n - 2) a(n) + (- n - 2) a(n + 1) + a(n + 2)}`,
``,
`> rectodiffeq(",a(n),f(t));`,
``,
`       {D(f)(0) = 1, f(0) = 0,`,
``,
`                2                        3    2                4  (2)`,
`           (2 t  + t - 1) f(t) + (4 t  + t ) D(f)(t) + t  D   (f)(t) + t}`,
``,
`> borel(",f(t),'diffeq');`,
``,
`                                      2`,
`                              t + (t  + t - 1) f(t)`,
``,
`> diffeqtorec(",f(t),a(n));`,
```

```
``,
`                {a(1) = 1, a(0) = 0, a(n + 1) - a(n + 2) + a(n)}`,
``,
`SEE ALSO: gfun[borel]`):

`help/gfun/text/diffeq+diffeq` := TEXT(
`FUNCTION: gfun[diffeq+diffeq]   - sum of two holonomic functions`,
`          gfun[diffeq*diffeq]   - Cauchy product of two holonomic functions`,
`          gfun[hadamardproduct] - Hadamard product of two holonomic functions`,
``,
`CALLING SEQUENCE: diffeq+diffeq(eq1,eq2,y(z))`,
`                  diffeq*diffeq(eq1,eq2,y(z))`,
`                  hadamardproduct(eq1,eq2,y(z))`,
``,
`PARAMETERS:`,
`   eq1,eq2 - two linear differential equations with polynomial coefficients`,
`   y,z     - the name of the holonomic function and the generic variable`,
``,
`SYNOPSIS:`,
`- If f (resp. g) is an holonomic function solution of eq1 (resp. eq2),`,
`  gfun[diffeq+diffeq] outputs a linear differential equation verified by f+g,`,
`  gfun[diffeq*diffeq] outputs a linear differential equation verified by f*g,`,
`  and gfun[hadamardproduct] outputs a linear differential equation verified`,
`  by the Hadamard product of f and g (the function whose coefficient of z^n`,
`  in the Taylor expansion around 0 is the product of the corresponding`,
`  coefficients of f and g).`,
``,
`- The differential order of the output equation is at most the sum of the`,
`  input equations differential orders for gfun[diffeq+diffeq], and their product for`,
`  gfun[diffeq*diffeq].`,
``,
`EXAMPLES:`,
`> with(gfun):`,
`> eq1 := D(y)(x)-y(x):`,
`> eq2 := (1+x)*(D@@2)(y)(x)+D(y)(x):`,
`> ``diffeq+diffeq``(eq1,eq2,y(x));`,
``,
`                                    2   (2)                      2   (3)`,
`   (- 3 - x) D(y)(x) + (1 - 2 x - x ) D   (y)(x) + (2 + 3 x + x ) D   (y)(x)`,
``,
`> ``diffeq*diffeq``(eq1,eq2,y(x));`,
``,
`                                                          (2)`,
`               y(x) x + (- 2 x - 1) D(y)(x) + (1 + x) D   (y)(x)`,
``,
`> hadamardproduct(eq1,eq2,y(x));`,
``,
`                                          (2)`,
`                {(- x - 1) D(y)(x) - x D   (y)(x), D(y)(0) = 0}`,
``,
`SEE ALSO: gfun[rec+rec], gfun[rec*rec], gfun[cauchyproduct]`):
`help/gfun/text/diffeq*diffeq` := `help/gfun/text/diffeq+diffeq`:
`help/gfun/text/hadamardproduct` := `help/gfun/text/diffeq+diffeq`:

`help/gfun/text/rec+rec` := TEXT(
`FUNCTION: gfun[rec+rec]       - sum of two holonomic recurrences`,
```

```
`          gfun[rec*rec]       - termwise product of two holonomic recurrences`,
`          gfun[cauchyproduct] - Cauchy product of two holonomic recurrences`,
`,
`CALLING SEQUENCE: rec+rec(rec1, rec2, u(n))`,
`                  rec*rec(rec1, rec2, u(n))`,
`                  cauchyproduct(rec1, rec2, u(n))`,
``,
`PARAMETERS:`,
`   rec1,rec2 - two linear recurrences with polynomial coefficients`,
`   u,n       - the variable and index of the recurrences`,
``,
`SYNOPSIS:`,
`- If a(n) and b(n) are the sequences defined respectively by rec1 and rec2,`,
`  gfun[rec+rec] outputs a recurrence for a(n)+b(n), gfun[rec*rec] outputs a`,
`  recurrence for a(n)*b(n), and gfun[cauchyproduct] outputs a recurrence for`,
`  their Cauchy product or convolution c(n) = sum(a(i)*b(n-i),i=0..n).`,
``,
`EXAMPLES:`,
`> with(gfun):`,
`> rec1:=u(n+1)=(n+1)*u(n):`,
`> rec2 := u(n+1)=2*u(n):`,
`> ``rec+rec``(rec1,rec2,u(n));`,
``,
`                  2                                2`,
`            (12 n  + 24 n + 12) u(n) + (- 8 n  - 36 n - 28) u(n + 1)`,
``,
`                      2`,
`                  + (n  + 14 n + 17) u(n + 2) + (- n - 3) u(n + 3)`,
``,
`> ``rec*rec``(rec1,rec2,u(n));`,
`,
`                           (- 2 n - 2) u(n) + u(n + 1)`,
``,
`> cauchyproduct(rec1,rec2,u(n));`,
``,
`                  (2 n + 4) u(n) + (- n - 4) u(n + 1) + u(n + 2)`,
``,
`SEE ALSO: gfun[diffeq+diffeq], gfun[diffeq*diffeq], gfun[hadamardproduct]`):
`help/gfun/text/rec*rec`:=`help/gfun/text/rec+rec`:
`help/gfun/text/cauchyproduct`:=`help/gfun/text/rec+rec`:

`help/gfun/text/algebraicsubs` := TEXT(
`FUNCTION: gfun[algebraicsubs] - substitute an algebraic function into an holonomic one`,
``,
`CALLING SEQUENCE: algebraicsubs(deq,eq,y(z))`,
``,
`PARAMETERS:`,
`   deq - linear differential equation in y(z) with polynomial coefficients`,
`   eq  - algebraic equation in y(z)`,
`   y,z - the name of the holonomic function and the generic variable`,
``,
`SYNOPSIS:`,
`- Let f be the holonomic function defined by the equation deq, and g be the`,
`  algebraic equation defined by eq, then gfun[algebraicsubs] outputs a`,
`  differential equation satisfied by the composition f@g, which is holonomic`,
`  by closure properties of holonomic functions.`,
```

```
'',
'- Let d1 be the differential order of deq, and d2 be the degree of eq. If',
   the equation deq is homogeneous, then the order of f@g is at most d1*d2',
   otherwise it is at most (d1+1)*d2.',
'',
'EXAMPLES:',
'> with(gfun):',
'# compute an holonomic equation for cos(sqrt(1-4*t)):',
'> deq := (D@@2)(f)(t)+f(t):    # cos(t) is solution',
'> eq := f^2-1+4*t:          # sqrt(1-4*t) is solution',
'> algebraicsubs(deq,eq,f(t));',
'',
'                                                        (2)',
'                 - 4 f(t) + 2 D(f)(t) + (- 1 + 4 t) D    (f)(t)',
'',
'SEE ALSO: gfun[diffeq+diffeq], gfun[rec+rec]'):

save('gfun.m');
quit
```