Combo Project Seven

Combinatorics (01:640:454)

Project Code Base

Team 7

Samuel Minkin

Kenneth Chan

Submission Date

December 14th, 2020

Table of Contents

Table of Contents **2**

Module 0 **3**

Existing Code 3

Module 1 **22**

Helper Functions 22

Module 2 **24**

Math Functions: Modulus, Multiply, Divide (Ratio), Add/Sub 24

Module 3 **32**

Analytics Functions: Plots, Moments, Ratios 32

Module 0

1. Existing Code

Existing Code Provided by Dr. Z

```
#########################################################################################################
######
##This is ComboProject7.txt, a Maple package to generate and investigate integer sequences
#
# that are the diagonal coefficients of rational functions in two, three, and four variables
#
#It finds recurrences, growth rates, and critical exponents
####It is the Maple package created by Team 7 in Dr. Z.'s Combinatorics Class at
#
#Rutgers University, Fall 2020.
#
# Save this file as `ComboProject7.txt`, to use it
#
#Type, in a Maple session
#
#read `ComboProject2.txt`):
#
#and then to get a list of the functions type
#
#Help():
#
#For Help with any of the functions, type
#
#Help(FunctionName)
#
#Team Leader: Samuel Minkin
#
#Team members: Samuel Minkin, Kenneth Chan
#
#########################################################################################################
#######



print(`----------------------`):
print(``):
print(`Team Leader: Samuel Minkin `):
```

```
print(``):
print(`Other Team members: Kenneth Chan `):
print(`----------------------`):

print(`This is ComboProject7.txt, a Maple package that is part of Project 2 in Dr. Z.'s
Combinatorics Class at Rutgers University`):
print(`Its purpose is to create a database of integer sequences  that are in the diagonals of
the Taylor`):
print(`expansions of rational functions of the form 1/(1-a*x-b*y-c*x*y) with small
coefficients, a,b,c`):
print(`and analogously for two and three variables`):


print(`For a list of all the functions type: Help(); `):
print(`For Help with any of the functions, type  Help(FunctionName):`):
with(combinat):
Help:=proc()
if nargs=0 then
print(`The available procedures are`):
print(` AsyOpe, AsySeq, CriExp, FindRec, GrowthConst, DiagSeq2, DiagSeq3, Info2, Info3,  `):
print(` nthCoeffSeq2, nthCoeffSeq3, getGF2, getGF3, checkEquality2, checkEquality3,
DiagSeq2Mod, DiagSeq3Mod, `):
print(` multSeq2, multSeq3, multSeq2K, multSeq3K, findRatios2, findRatios3, addDiagSeq2,
addDiagSeq3, `):
print(` subDiagSeq2, subDiagSeq3, diagSeq2KComps, diagSeq3KComps, AveAndMoms, plotVals, `):


elif nargs=1 and args[1]=AsyOpe then
print(`IT DOES NOT ALWAYS GIVE THE RIGHT ANSWER. MUST HAVE BUGS. USE WITH CAUTION`):
print(`AsyOpe(ope,n,N,k):The asymptotic to order k of a dominant solution to ope(n,N). `):
print(`AsyOpe((n+1)/(n+2)-3*(2*n+3)/(n+2)*N+N^2,n,N,4); `):

elif nargs=1 and args[1]=AsySeq then
print(`AsySeq(L,n,k): tries to fit the sequence L to be of the form
C*mu^n*n^theta*(1+c1/n+...+ck/n^k). Try:`):
print(`L:=SeqW({[1,0],[0,1],[1,1]},100): AsySeq(L,n,2);`):

elif nargs=1 and args[1]=CriExp then
print(`CriExp(ope,n,N): The critical exponent of the operator ope in n and N Try`):
print(`CriExp((n+1)/(n+2)-3*(2*n+3)/(n+2)*N+N^2,n,N):`):


elif nargs=1 and args[1]=DiagSeq2 then
print(`DiagSeq2(f,x,y,L1): The first L1 terms of the sequence of coefficients of x^i*y^i in the
rational function f of variables x and y.`):
print(`Try: `):
print(`DiagSeq2(1/(1-x-y),x,y,10);`):
```

```
elif nargs=1 and args[1]=DiagSeq3 then
print(`DiagSeq3(f,x,y,z,L1): The first L1 terms of the sequence of coefficients of x^i*y^i*z^i
in the rational function f of variables x and y and z.`):
print(`Try: `):
print(`DiagSeq3(1/(1-x-y-z),x,y,z,10);`):

elif nargs=1 and args[1]=Findrec then
print(`Findrec(L,n,N,MaxC): Given a list L tries to find a linear recurrence equation with`):
print(`poly coffs. ope(n,N), where n is the discrete variable, and N is the shift operator in n
`):
print(`of maximum COMPLEXCITY, DEGREE+ORDER<=MaxC`):
print(`If none exists, it returns FAIL. . Try:`):
print(`e.g. try Findrec([1,1,2,3,5,8,13,21,34,55,89],n,N,2);`):

elif nargs=1 and args[1]=GrowthConst then
print(`GrowthConst(ope,n,N): The growth constant of sequence of integers satistying the
recurrence given by the`):
print(`shift operator, ope. Try: `):
print(`GrowthConst(N^2-N-1,n,N);`):


elif nargs=1 and args[1]=Info2 then
print(`Info2(f,x,y,n,N,L1,L2,k): Given a rational function f of the variables x and y, symbols
n and N, a positive integer L1, another positive integer L2,`):
print(`and a small positive integer k, outputs a list consisting of`):
print(`(i) itself`):
print(`(ii) The first L1 terms of the sequence of coefficients of x^i*y^i`):
print(`(iii) The guessed recurrence operator satisfied by the sequence of complexity at most L2
or FAIL`):
print(`(iv) The growth constant`):
print(` (v) The critical exponent `):
print(` (vi) The estimated asymptotics to order k `):
print(` Try: `):
print(`Info2(1/(1-x-y),x,y,n,N,100,10,2);`):


elif nargs=1 and args[1]=Info3 then
print(`Info3(f,x,y,z,n,N,L1,L2,k): Given a rational function f of the variables x, y and z, `):
print(`and symbols n and N, a positive integer L1, another positive integer L2,`):
print(`and a small positive integer k, outputs a list consisting of`):
print(`(i) itself`):
print(`(ii) The first L1 terms of the sequence of coefficients of x^i*y^i*z^i`):
print(`(iii) The guessed recurrence operator satisfied by the sequence of complexity at most L2
or FAIL`):
print(`(iv) The growth constant`):
print(` (v) The critical exponent `):
print(` (vi) The estimated asymptotics to order k `):
```

```
print(` Try: `):
print(`Info3(1/(1-x-y-z),x,y,z,n,N,100,10,2);`):


elif nargs=1 and args[1]=SeqFromRec then
print(`SeqFromRec(ope,n,N,Ini,K): inputs a linear recurrence operator with constant
coefficients`):
print(`in the discrete variable n and the shift operator N, and a list of intetegers Ini whose
length is the order of ope`):
print(`outputs the list of length K of the sequence that satisfies the unerlying recurrence`):
print(`For example, try:`):
print(`SeqFromRec(N-n-1,n,N,[1],10);`):


elif nargs=1 and args[1]=nthCoeffSeq2 then
print(`nthCoeffSeq2(f,x,y,n): Given a generating function f in two variables and an index n,`):
print(`will return a(n,n), the coefficient of x^ny^n`):
print(`For example, try:`):
print(`nthCoeffSeq2(1/(1-x-y-x*y),100);`):

elif nargs=1 and args[1]=nthCoeffSeq3 then
print(`nthCoeffSeq3(f,x,y,z,n): Given a generating function f in three variables and an index
n,`):
print(`will return a(n,n,n), the coefficient of x^ny^nz^n`):
print(`For example, try:`):
print(`nthCoeffSeq3(1/(1-x-y-z),10);`):

elif nargs=1 and args[1]=getGF2 then
print(`getGF2(a,b,c): Given coefficients a,b,c, will return f:=1/(1-a*x-b*x-c*x*y)`):
print(`For example, try:`):
print(`getGF2(1,1,1);`):

elif nargs=1 and args[1]=getGF3 then
print(`getGF3(a,b,c,d,e,f): Given coefficients a,b,c,d,e,f will return
f:=1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z)`):
print(`For example, try:`):
print(`getGF2(1,1,1,1,1,1);`):

elif nargs=1 and args[1]=checkEquality2 then
print(`checkEquality2(f1,f2,x,y): Checks if the sequences a(m,n),b(m,n) corresponding to
{f1,f2},`):
print(`rational functions in the variables {x,y}, are equal`):
print(`For example, try:`):
print(`checkEquality2(1/(1-x-y),(1+(1-x-y)/(1-(1-x-y)^2),x,y)`):

elif nargs=1 and args[1]=checkEquality3 then
print(`checkEquality3(f1,f2,x,y,z): Checks if the sequences a(m,n,k),b(m,n,k) corresponding to
{f1,f2},`):
```

```
print(`rational functions in the variables {x,y,z}, are equal`):
print(`For example, try:`):
print(`checkEquality3(1/(1-x-y-z),(1+(x-y-z))/(1-(x-y-z)^2),x,y,z)`):

elif nargs=1 and args[1]=DiagSeq2Mod then
print(`DiagSeq2Mod(f,x,y,L1,m): Given a rational function in two variables x,y,`):
print(`it will return the first L1 elements mod m.`):
print(`For example, try:`):
print(`DiagSeq2Mod(1/(1-x-y-x*y),x,y,20,5);`):

elif nargs=1 and args[1]=DiagSeq3Mod then
print(`DiagSeq3Mod(f,x,y,z,L1,m): Given a rational function in three variables x,y,z,`):
print(`it will return the first L1 elements mod m.`):
print(`For example, try:`):
print(`DiagSeq3Mod(1/(1-x-y-z),x,y,z,20,5);`):

elif nargs=1 and args[1]=multSeq2 then
print(`multSeq2(f1,f2,x,y,n): Given two rational functions {f1,f2} in two variables {x,y},`):
print(`of the form 1/(1-a*x-b*x-c*x*y), will return the element by element product of the first
n terms`):
print(`in both sequences, i.e., [L1[1]*L2[1], L1[2]*L2[2], ..., L1[n]*L2[n]].`):
print(`For example, try:`):
print(`multSeq2(1/(1-x-y-x*y),1/(1-x-y-x*y),x,y,20);`):

elif nargs=1 and args[1]=multSeq3 then
print(`multSeq3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in three variables
{x,y,z},`):
print(`of the form 1/(1-a*x-b*x-c*x*y-d*x*z-e*y*z-f*x*y*z),`):
print(`will return the element by element product of the first n terms in both sequences,`):
print(`i.e., [L1[1]*L2[1], L1[2]*L2[2], ..., L1[n]*L2[n]].`):
print(`For example, try:`):
print(`multSeq3(1/(1-x-y-z),1/(1-x-y-z),x,y,z,20);`):

elif nargs=1 and args[1]=multSeq2K then
print(`multSeq2K(L,x,y,n): Given an arbitrarily long list L, consisting of rational`):
print(`functions in two variables {x,y} of the form 1/(1-a*x-b*x-c*x*y),`):
print(`ill return the element by element product of the first n terms in all sequences,`):
print(`i.e., [L1[1]*L2[1]*...*LK[1], L1[2]*L2[2]*...*LK[2], ..., L1[n]*L2[n]...*LK[n]].`):
print(`For example, try:`):
print(`multSeq2K([1/(1-x-y-x*y),1/(1-x-y-x*y)],x,y,20);`):

elif nargs=1 and args[1]=multSeq3K then
print(`multSeq2K(L,x,y,n): Given an arbitrarily long list L, consisting of rational`):
print(`functions in two variables {x,y} of the form 1/(1-a*x-b*x-c*x*y),`):
print(`ill return the element by element product of the first n terms in all sequences,`):
print(`i.e., [L1[1]*L2[1]*...*LK[1], L1[2]*L2[2]*...*LK[2], ..., L1[n]*L2[n]...*LK[n]].`):
print(`For example, try:`):
print(`multSeq3K([1/(1-x-y-z),1/(1-x-y-z)],x,y,z,20);`):
```

```
elif nargs=1 and args[1]=findRatios2 then
print(`findRatios2(f1,f2,x,y,n): Given two rational functions {f1,f2} in the variables
{x,y}.`):
print(`Returns the ratios element-wise between the diagonal sequences generated by f1,f2,`):
print(`i.e., [L1[1]/L2[1], L1[2]/L2[2], ..., L1[n]/L2[n]].`):
print(`For example, try:`):
print(`findRatios2(1/(1-x-y-x*y),1/(1-x-y-2*x*y),x,y,20);`):

elif nargs=1 and args[1]=findRatios3 then
print(`findRatios3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in the variables
{x,y,z}.`):
print(`Returns the ratios element-wise between the diagonal sequences generated by f1,f2,`):
print(`i.e., [L1[1]/L2[1], L1[2]/L2[2], ..., L1[n]/L2[n]].`):
print(`For example, try:`):
print(`findRatios3(1/(1-x-y-z),1/(1-x-y-2*z),x,y,z,20);`):

elif nargs=1 and args[1]=addDiagSeq2 then
print(`addDiagSeq2(f1,f2,x,y,n): Given two rational functions {f1,f2} in the variables
{x,y}.`):
print(`Returns the addition element-wise between the diagonal sequences generated by f1,f2,`):
print(`i.e., [L1[1]+L2[1], L1[2]+L2[2], ..., L1[n]+L2[n]].`):
print(`For example, try:`):
print(`addDiagSeq2(1/(1-x-y-x*y),1/(1-x-y-2*x*y),x,y,20);`):

elif nargs=1 and args[1]=addDiagSeq3 then
print(`addDiagSeq3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in the variables
{x,y,z}.`):
print(`Returns the addition element-wise between the diagonal sequences generated by f1,f2,`):
print(`i.e., [L1[1]+L2[1], L1[2]+L2[2], ..., L1[n]+L2[n]].`):
print(`For example, try:`):
print(`addDiagSeq3(1/(1-x-y-z),1/(1-x-y-2*z),x,y,z,20);`):

elif nargs=1 and args[1]=subDiagSeq2 then
print(`subDiagSeq2(f1,f2,x,y,n): Given two rational functions {f1,f2} in the variables
{x,y}.`):
print(`Returns the subtraction element-wise between the diagonal sequences generated by
f1,f2,`):
print(`i.e., [L1[1]-L2[1], L1[2]-L2[2], ..., L1[n]-L2[n]].`):
print(`For example, try:`):
print(`subDiagSeq2(1/(1-x-y-x*y),1/(1-x-y-2*x*y),x,y,20);`):

elif nargs=1 and args[1]=subDiagSeq3 then
print(`subDiagSeq3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in the variables
{x,y,z}.`):
print(`Returns the subtraction element-wise between the diagonal sequences generated by
f1,f2,`):
print(`i.e., [L1[1]-L2[1], L1[2]-L2[2], ..., L1[n]-L2[n]].`):
```

```
print(`For example, try:`):
print(`subDiagSeq3(1/(1-x-y-z),1/(1-x-y-2*z),x,y,z,20);`):

elif nargs=1 and args[1]=diagSeq2KComps then
print(`diagSeq2KComps(f,k,x,y,n): Given a generating function f and an integer k, returns the
first n terms`):
print(`in the diag seq of f^k/k!, which means exactly k components depending on the context.`):
print(`For example, try:`):
print(`diagSeq2Kcomps(1/(1-x-y-x*y),2,x,y,20);`):

elif nargs=1 and args[1]=diagSeq3KComps then
print(`diagSeq3KComps(f,k,x,y,z,n): Given a generating function f and an integer k, returns the
first n terms`):
print(`in the diag seq of f^k/k!, which means exactly k components depending on the context.`):
print(`For example, try:`):
print(`diagSeq3Kcomps(1/(1-x-y-z),2,x,y,z,20);`):

elif nargs=1 and args[1]=AveAndMoms then
print(`Borrowed from Dr.Z, lecture 22, Math 454`):
print(`AveAndMoms(f,x,K): given an ordinary generating function f of x, that is a
weight-enumerator of some`):
print(`numerical attribute,and a positive integer K, outputs `):
print(`(i) The average  (ii) the standar-deviation, (iii) the 3rd through the K-th SCALED
moments about the mean.`):
print(`For example, try:`):
print(`AveAndMoms((1+x)^1000,x,6);`):

elif nargs=1 and args[1]=plotVals then
print(`plotVals(L1,L2): Given {L1}, a list of x-coordinates, and {L2}, a list of
y-coordinates,`):
print(`where L1 and L2 have the same size, will return a plot of L2 vs. L1`):
print(`For example, try:`):
print(`plotVals([1,2,3],[4,5,6]);`):

else
 print(`There is no Help for`):

print(args):
fi:

end:




##Start CODE taken from Dr. Z.'s Maple package Findrec.txt
ezra:=proc()
if args=NULL then
```

```
print(` FindRec.txt: A Maple package for empirically guessing partial recurrence`):
print(`equations satisfied by Discrete Functions of ONE Variable`):
print():
print(`For help with a specific procedure, type "ezra(procedure_name);"`):
print(`Contains procedures:  `):
print(`  findrec, Findrec, FindrecF`):
print():

elif nargs=1 and args[1]=findrec then
print(`findrec(f,DEGREE,ORDER,n,N): guesses a recurrence operator annihilating`):
print(`the sequence f of degree DEGREE and order ORDER.`):
print(`For example, try: findrec([seq(i,i=1..10)],0,2,n,N);`):

elif nargs=1 and args[1]=Findrec then
print(`Findrec(f,n,N,MaxC): Given a list f tries to find a linear recurrence equation with`):
print(`poly coffs. ope(n,N), where n is the discrete variable, and N is the shift operator `):
print(`of maximum DEGREE+ORDER<=MaxC`):
print(`e.g. try Findrec([1,1,2,3,5,8,13,21,34,55,89],n,N,2);`):

elif nargs=1 and args[1]=FindrecF then
print(`FindrecF(f,n,N): Given a function f of a single variable tries to find a linear
recurrence equation with`):
print(`poly coffs. .g. try FindrecF(i->i!,n,N);`):


elif nargs=1 and args[1]=SeqFromRec then
print(`SeqFromRec(ope,n,N,Ini,K): Given the first L-1`):
print(`terms of the sequence Ini=[f(1), ..., f(L-1)]`):
print(`satisfied by the recurrence ope(n,N)f(n)=0`):
print(`extends it to the first K values`):
print(`For example, try:`):
print(`SeqFromRec(N-n-1,n,N,[1],10);`):


fi:

end:


###Findrec
#findrec(f,DEGREE,ORDER,n,N): guesses a recurrence operator annihilating
#the sequence f of degree DEGREE and order ORDER
#For example, try: findrec([seq(i,i=1..10)],0,2,n,N);
findrecVerbose:=proc(f,DEGREE,ORDER,n,N)
local ope,var,eq,i,j,n0,kv,var1,eq1,mu,a:
if (1+DEGREE)*(1+ORDER)+3+ORDER>nops(f) then
ERROR(`Insufficient data for a recurrence of order`,ORDER, `degree`,DEGREE):
```

```
fi:
ope:=0:
var:={}:

for i from 0 to ORDER do
 for j from 0 to DEGREE do
  ope:=ope+a[i,j]*n^j*N^i:
  var:=var union {a[i,j]}:
 od:
od:

eq:={}:

for n0 from 1 to (1+DEGREE)*(1+ORDER)+2 do
  eq1:=0:

  for i from 0 to ORDER do
     eq1:=eq1+subs(n=n0,coeff(ope,N,i))*op(n0+i,f):
  od:

   eq:= eq union {eq1}:
od:

var1:=solve(eq,var):

kv:={}:

for i from 1 to nops(var1) do
 mu:=op(i,var1):

if op(1,mu)=op(2,mu) then
   kv:= kv union {op(1,mu)}:
 fi:

od:

ope:=subs(var1,ope):

if ope=0 then
  RETURN(FAIL):
fi:

ope:={seq(coeff(expand(ope),kv[i],1),i=1..nops(kv))} minus {0}:

if nops(ope)>1 then
print(`There is some slack, there are `, nops(ope)):
print(ope):
RETURN(Yafe(ope[1],N)[2]):
```

```
elif nops(ope)=1 then
RETURN(Yafe(ope[1],N)[2]):
else
 RETURN(FAIL):
fi:

end:

#findrec(f,DEGREE,ORDER,n,N): guesses a recurrence operator annihilating
#the sequence f of degree DEGREE and order ORDER
#For example, try: findrec([seq(i,i=1..10)],0,2,n,N);
findrec:=proc(f,DEGREE,ORDER,n,N)
local ope,var,eq,i,j,n0,kv,var1,eq1,mu,a:
option remember:


if not findrecEx(f,DEGREE,ORDER,ithprime(20)) then
 RETURN(FAIL):
fi:

if not findrecEx(f,DEGREE,ORDER,ithprime(40)) then
 RETURN(FAIL):
fi:

if not findrecEx(f,DEGREE,ORDER,ithprime(80)) then
 RETURN(FAIL):
fi:


if (1+DEGREE)*(1+ORDER)+5+ORDER>nops(f) then
ERROR(`Insufficient data for a recurrence of order`,ORDER, `degree`,DEGREE):
fi:
ope:=0:
var:={}:

for i from 0 to ORDER do
 for j from 0 to DEGREE do
  ope:=ope+a[i,j]*n^j*N^i:
  var:=var union {a[i,j]}:
 od:
od:

eq:={}:

for n0 from 1 to (1+DEGREE)*(1+ORDER)+4 do
  eq1:=0:

  for i from 0 to ORDER do
```

```
     eq1:=eq1+subs(n=n0,coeff(ope,N,i))*op(n0+i,f):
  od:

   eq:= eq union {eq1}:
od:

var1:=solve(eq,var):

kv:={}:

for i from 1 to nops(var1) do
 mu:=op(i,var1):

if op(1,mu)=op(2,mu) then
   kv:= kv union {op(1,mu)}:
 fi:

od:

ope:=subs(var1,ope):

if ope=0 then
  RETURN(FAIL):
fi:

ope:={seq(coeff(expand(ope),kv[i],1),i=1..nops(kv))} minus {0}:

if nops(ope)>1 then
RETURN(Yafe(ope[1],N)[2]):
elif nops(ope)=1 then
RETURN(Yafe(ope[1],N)[2]):
else
 RETURN(FAIL):
fi:

end:



Yafe:=proc(ope,N) local i,ope1,coe1,L:
if ope=0 then
 RETURN(1,0):
fi:
ope1:=expand(ope):
L:=degree(ope1,N):
coe1:=coeff(ope1,N,L):
ope1:=normal(ope1/coe1):
ope1:=normal(ope1):
```

```
ope1:=
convert(
[seq(factor(coeff(ope1,N,i))*N^i,i=ldegree(ope1,N)..degree(ope1,N))],`+`):
factor(coe1),ope1:
end:


#Findrec(f,n,N,MaxC): Given a list f tries to find a linear recurrence equation with
#poly coffs.
#of maximum DEGREE+ORDER<=MaxC
#e.g. try Findrec([1,1,2,3,5,8,13,21,34,55,89],n,N,2);
Findrec:=proc(f,n,N,MaxC)
local DEGREE, ORDER,ope,L:

for L from 0 to MaxC do
for ORDER from 0 to L do
 DEGREE:=L-ORDER:
if (2+DEGREE)*(1+ORDER)+4>=nops(f) then
 print(`Insufficient data for degree`, DEGREE, `and order `,ORDER):
 RETURN(FAIL):
fi:
 ope:=findrec([op(1..(2+DEGREE)*(1+ORDER)+4,f)],DEGREE,ORDER,n,N):
     if ope<>FAIL then
       RETURN(ope):
      fi:
 od:
od:
FAIL:

end:







#SeqFromRec(ope,n,N,Ini,K): Given the first L-1
#terms of the sequence Ini=[f(1), ..., f(L-1)]
#satisfied by the recurrence ope(n,N)f(n)=0
#extends it to the first K values
SeqFromRec:=proc(ope,n,N,Ini,K)
local ope1,gu,L,n1,j1:
ope1:=Yafe(ope,N)[2]:
L:=degree(ope1,N):
if nops(Ini)<>L then
 ERROR(`Ini should be of length`, L):
fi:
```

```
ope1:=expand(subs(n=n-L,ope1)/N^L):

gu:=Ini:

for n1 from nops(Ini)+1 to K do
gu:=[op(gu), -add(gu[nops(gu)+1-j1]*subs(n=n1,coeff(ope1,N,-j1)),
j1=1..L)]:
od:

gu:

end:

#end Findrec


with(linalg):

#findrecEx(f,DEGREE,ORDER,m1): Explores whether thre
#is a good chance that there is a recurrence of degree DEGREE
#and order ORDER, using the prime m1
#For example, try: findrecEx([seq(i,i=1..10)],0,2,n,N,1003);
findrecEx:=proc(f,DEGREE,ORDER,m1)
local ope,var,eq,i,j,n0,eq1,a,A1,
D1,E1,Eq,Var,f1,n,N:
option remember:
f1:=f mod m1:
if (1+DEGREE)*(1+ORDER)+5+ORDER>nops(f) then
ERROR(`Insufficient data for a recurrence of order`,ORDER, `degree`,DEGREE):
fi:
ope:=0:
var:={}:

for i from 0 to ORDER do
 for j from 0 to DEGREE do
  ope:=ope+a[i,j]*n^j*N^i:
  var:=var union {a[i,j]}:
 od:
od:

eq:={}:

for n0 from 1 to (1+DEGREE)*(1+ORDER)+4 do
  eq1:=0:

  for i from 0 to ORDER do
     eq1:=eq1+subs(n=n0,coeff(ope,N,i))*op(n0+i,f1) mod m1:
  od:
```

```
   eq:= eq union {eq1}:
od:


Eq:= convert(eq,list):
Var:= convert(var,list):


D1:=nops(Var):
E1:=nops(Eq):
if E1<D1 then
  RETURN(true):
fi:

A1:=matrix(D1,D1):

for i from 1 to D1-1 do
for j from 1 to D1 do
  A1[i,j]:=coeff(Eq[i],Var[j]):
od:
od:

for j from 1 to nops(Var) do
  A1[D1,j]:=coeff(Eq[D1],Var[j]):
od:

if det(A1) mod m1 <>0  then
 RETURN(false):
fi:

if E1-D1>=1 then
 for j from 1 to nops(Var) do
  A1[D1,j]:=coeff(Eq[D1+1],Var[j]):
 od:

if det(A1) mod m1 <>0  then
 RETURN(false):
fi:
fi:

if E1-D1>=2 then
 for j from 1 to nops(Var) do
  A1[D1,j]:=coeff(Eq[D1+2],Var[j]):
 od:

if det(A1) mod m1 <>0  then
 RETURN(false):
```

```
fi:
fi:

true:

end:

#GrowthConst(ope,n,N): The growth constant of the operator ope in n and the shift operator N.
Try
#GrowthConst(N^2-N-1,n,N);
GrowthConst:=proc(ope,n,N) local ope1,champ,i,gu,rec:
ope1:=lcoeff(numer(ope),n):
gu:=[solve(ope1,N)]:
champ:=gu[1]:
rec:=evalf(abs(gu[1])):

for i from 2 to nops(gu) do
 if evalf(abs(gu[i]))>rec then
  champ:=gu[i]:
 rec:=evalf(abs(gu[i])):
 fi:
od:
champ:
end:


###END CODE taken from Dr. Z.'s Maple package Findrec.txt

#CriExp(ope,n,N): The critical exponent of the operator ope in n and N Try
#CriExp((n+1)/(n+2)-3*(2*n+3)/(n+2)*N+N^2,n,N):
CriExp:=proc(ope,n,N) local A,ope1,x,c,t,t0,i,eq:

ope1:=numer(ope):

A:=numer(normal(add(subs(n=1/x,coeff(ope1,N,i))*(1+i*x)^t*c^i,i=0..degree(ope1,N)))):

A:=taylor(A,x=0,4):
eq:={coeff(A,x,0),coeff(A,x,1)}:
t0:=solve(eq,{t,c}):
if t0=NULL then
RETURN(FAIL):
fi:
subs(t0,t):
end:



#AsyOpe(ope,n,N,k):The asymptotic to order k of a dominant solution to ope(n,N). Try:
```

```
##AsyOpe((n+1)/(n+2)-3*(2*n+3)/(n+2)*N+N^2,n,N,4):
AsyOpe:=proc(ope,n,N,k) local mu,t,a,i1,i,lu,ope1,A,c,x,eq,var:
print(`WARNING: IT MUST HAVE BUGS `):

mu:=GrowthConst(ope,n,N):
t:=CriExp(ope,n,N):
if t=FAIL then
 RETURN(FAIL):
fi:
ope1:=numer(ope):
lu:=1+add(a[i1]/n^i1,i1=1..k):

A:=numer(add(subs(n=1/x,coeff(ope1,N,i))*c^i*(1+i*x)^t*subs(n=(1+i*x),lu),i=0..degree(ope1,N)))
:
A:=taylor(A,x=0,k+1):
eq:={seq(coeff(A,x,i),i=0..k)}:
var:={seq(a[i],i=1..k),c}:

var:=solve(eq,var):
lu:=subs(var,lu):
mu^n*n^t*subs(var,lu):
end:


Digits:=40:

#AsySeq(L,n,k): tries to fit the sequence L to be of the form
C*mu^n*n^theta*(1+c1/n+...+ck/n^k). Try:
#L:=Gseq([{1},{1}],50); MyAsy(L,n,3);

AsySeq:=proc(L,n,k) local gu,logC,logmu,theta,c,i,eq,var,C,mu,x,ku:

if nops(L)<50 then
 print(`List must be at least of length 50`):
 RETURN(FAIL):
fi:

if 10+k>nops(L) then
 print(`k must be at most`, nops(L)-10 ):
fi:

gu:=logC+logmu*n+theta*log(n)+add(c[i]/n^i,i=1..k):

var:={logC,logmu,theta,seq(c[i],i=1..k)}:

eq:={seq(log(L[i])-subs(n=i,gu),i=nops(L)-k-2..nops(L))}:
```

```
var:=evalf(solve(eq,var)):


C:=exp(subs(var,logC)):
mu:=exp(subs(var,logmu)):
theta:=subs(var,theta):

C:=evalf(C,10):
mu:=evalf(mu,10):
theta:=evalf(theta,10):

ku:= exp(subs(var,add(c[i]*x^i,i=1..k))):
ku:=taylor(ku,x=0,k+1):
ku:=add(evalf(coeff(ku,x,i),10)/n^i,i=0..k):
evalf(C*mu^n*n^theta*ku,10):
end:


#DiagSeq2(f,x,y,L1): The first L1 terms of the sequence of coefficients of x^i*y^i in the
rational function f
#Try:
#DiagSeq2(1/(1-x-y),x,y,10);
DiagSeq2:=proc(f,x,y,L1) local M,i,f1,f2:

f1:=taylor(f,x=0,L1+1):

M:=[]:

for  i from 1 to L1 do
  f2:=taylor(coeff(f1,x,i),y=0,i+1):
 M:=[op(M), coeff(f2,y,i)]:
od:

M:
end:


#Info2(f,x,y,n,N,L1,L2,k): Given a rational function f of the variables x and y, symbols n and
N, a positive integer L1, another positive integer L2,
#and a small positive integer k, outputs a list consisting of
#(i) itself
#(ii) The first L1 terms of the sequence of coefficients of x^i*y^i
#(iii) The guessed recurrence operator satisfied by the sequence of complexity at most L2 or
FAIL
#(iv) The growth constant
#(v) The critical exponent
#(vi) The estimated asymptotics to order k
#Try:
```

```
#Info2(1/(1-x-y),x,y,n,N,100,10,2);
Info2:=proc(f,x,y,n,N,L1,L2,k) local LI,ope:
LI:=DiagSeq2(f,x,y,L1):
ope:=Findrec(LI,n,N,L2):
if ope=FAIL then
RETURN([f,LI, FAIL,FAIL, FAIL]):
fi:
[f,LI,ope,evalf(GrowthConst(ope,n,N)), CriExp(ope,n,N), AsySeq(LI,n,k)]:
end:



#DiagSeq3(f,x,y,z,L1): The first L1 terms of the sequence of coefficients of x^i*y^i*z^i in the
rational function f of variables x,y,z
#Try:
#DiagSeq3(1/(1-x-y-z),x,y,10);
DiagSeq3:=proc(f,x,y,z,L1) local M,i,f1,f2,f3:

f1:=taylor(f,x=0,L1+1):

M:=[]:

for  i from 1 to L1 do
  f2:=taylor(coeff(f1,x,i),y=0,i+1):
  f3:=taylor(coeff(f2,y,i),z=0,i+1):
   M:=[op(M), coeff(f3,z,i)]:
od:

M:
end:



#Info3(f,x,y,z,n,N,L1,L2,k): Given a rational function f of the variables x, y and z, symbols n
and N, a positive integer L1, another positive integer L2,
#and a small positive integer k, outputs a list consisting of
#(i) itself
#(ii) The first L1 terms of the sequence of coefficients of x^i*y^i*z^i
#(iii) The guessed recurrence operator satisfied by the sequence of complexity at most L2 or
FAIL
#(iv) The growth constant
#(v) The critical exponent
#(vi) The estimated asymptotics to order k
#Try:
#Info3(1/(1-x-y-z),x,y,z,n,N,100,10,2);
Info3:=proc(f,x,y,z,n,N,L1,L2,k) local LI,ope:
LI:=DiagSeq3(f,x,y,z,L1):
ope:=Findrec(LI,n,N,L2):
```

```
if ope=FAIL then
RETURN([f,LI, FAIL,FAIL, FAIL]):
fi:
[f,LI,ope,evalf(GrowthConst(ope,n,N)), CriExp(ope,n,N), AsySeq(LI,n,k)]:
end:
```

Module 1

1. Helper Functions

Helper Functions: Used for Convenience

```
#nthCoeffSeq2(f,x,y,n): Given a generating function f in two variables and an index n, will
return a(n,n), the coefficient of x^ny^n
#Try
#nthCoeffSeq2(1/(1-x-y-x*y),100)
nthCoeffSeq2:=proc(f,n) local i:
if abs(subs({x=0,y=0},denom(f))) <> 1 or abs(numer(f)) <> 1 then
      print(`The function f should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:
coeff(taylor(coeff(taylor(f,x=0,n+1),x,n+1),y=0,n+1),y,n+1):
end:

#nthCoeffSeq3(f,x,y,z,n): Given a generating function f in three variables and an index n,
will return a(n,n,n), the coefficient of x^ny^nz^n
#Try
#nthCoeffSeq3(1/(1-x-y-z),10)
nthCoeffSeq3:=proc(f,n) local i:
if abs(subs({x=0,y=0,z=0},denom(f))) <> 1 or abs(numer(f)) <> 1 then
      print(`The function f should be of the form 1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:
coeff(taylor(coeff(taylor(coeff(taylor(f,x=0,n+1),x,n),y=0,n+1),y,n),z=0,n+1),z,n):
end:

#getGF2(a,b,c): Given coefficients a,b,c, will return f:=1/(1-a*x-b*x-c*x*y)
#Try
#getGF2(1,1,1)
getGF2:=proc(a,b,c):
(1/(1-a*x-b*y-c*x*y)):
end:

#getGF3(a,b,c,d,e,f): Given coefficients a,b,c,d,e,f will return
f:=1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z)
#Try
```

```
#getGF3(1,1,1,1,1,1)
getGF3:=proc(a,b,c,d,e,f):
(1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z)):
end:

#checkEquality2(f1,f2,x,y): Checks if the sequences a(m,n),b(m,n) corresponding to {f1,f2},
rational functions in the variables {x,y}, are equal
#Try
#checkEquality2(1/(1-x-y),(1+(1-x-y)/(1-(1-x-y)^2),x,y)
checkEquality2:=proc(f1,f2,x,y) local L1,L2:

L1:=DiagSeq2(f1,x,y,100):
L2:=DiagSeq2(f1,x,y,100):

RETURN(evalb(L1 = L2)):

end:

#checkEquality3(f1,f2,x,y,z): Checks if the sequences a(m,n,k),b(m,n,k) corresponding to
{f1,f2}, rational functions in the variables {x,y,z}, are equal
#Try
#checkEquality3(1/(1-x-y-z),(1+(x-y-z))/(1-(x-y-z)^2),x,y,z)
checkEquality3:=proc(f1,f2,x,y,z) local L1,L2:

L1:=DiagSeq3(f1,x,y,z,100):
L2:=DiagSeq3(f1,x,y,z,100):

RETURN(evalb(L1 = L2)):

end:
```

Module 2

1. Math Functions: Modulus, Multiply, Divide (Ratio), Add/Sub

Math Functions

```
#DiagSeq2Mod(f,x,y,L1,m): Given a rational function in two variables x,y, it will return the
first L1 elements mod m.
#Try
#DiagSeq2Mod(1/(1-x-y-x*y),x,y,20,5);
DiagSeq2Mod:=proc(f,x,y,L1,m):

if abs(subs({x=0,y=0},denom(f))) <> 1 or abs(numer(f)) <> 1 then
      print(`The function f should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:

[seq(coeff(taylor(coeff(taylor(f,x=0,L1+1),x,i),y=0,L1+1),y,i) mod m,i=0..L1)]:

end:

#DiagSeq3Mod(f,x,y,z,L1,m): Given a rational function in three variables x,y,z it will return
the first L1 elements mod m.
#Try
#DiagSeq3Mod(1/(1-x-y-z),x,y,z,20,5);
DiagSeq3Mod:=proc(f,x,y,z,L1,m):

if abs(subs({x=0,y=0,z=0},denom(f))) <> 1 or abs(numer(f)) <> 1 then
      print(`The function f should be of the form 1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:

[seq(coeff(taylor(coeff(taylor(coeff(taylor(f,x=0,L1+1),x,i),y=0,L1+1),y,i)),z=0,L1+1),z,i)
mod m, i=0..L1)]:

end:
```

```
#multSeq2(f1,f2,x,y,n): Given two rational functions {f1,f2} in two variables {x,y} of the
form 1/(1-a*x-b*x-c*x*y), will return the element by element product of the first n terms in
both sequences, i.e., [L1[1]*L2[1], L1[2]*L2[2], ..., L1[n]*L2[n]].
#Try
#multSeq2(1/(1-x-y-x*y),1/(1-x-y-x*y),x,y,20);
multSeq2:=proc(f1,f2,x,y,n) local L1,L2:

if abs(subs({x=0,y=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:

L1:=DiagSeq2(f1,x,y,n):
L2:=DiagSeq2(f2,x,y,n):

[seq(L1[i]*L2[i],i=1..n)]:

end:

#multSeq3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in three variables {x,y,z} of
the form 1/(1-a*x-b*x-c*x*y-d*x*z-e*y*z-f*x*y*z), will return the element by element product
of the first n terms in both sequences, i.e., [L1[1]*L2[1], L1[2]*L2[2], ..., L1[n]*L2[n]].
#Try
#multSeq3(1/(1-x-y-z),1/(1-x-y-z),x,y,z,20);
multSeq3:=proc(f1,f2,x,y,z,n) local L1,L2:

if abs(subs({x=0,y=0,z=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0,z=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:

L1:=DiagSeq3(f1,x,y,z,n):
```

```
L2:=DiagSeq3(f2,x,y,z,n):

[seq(L1[i]*L2[i],i=1..n)]:

end:

#multSeq2K(L,x,y,n): Given an arbitrarily long list L, consisting of rational functions in
two variables {x,y} of the form 1/(1-a*x-b*x-c*x*y), will return the element by element
product of the first n terms in all sequences, i.e., [L1[1]*L2[1]*...*LK[1],
L1[2]*L2[2]*...*LK[2], ..., L1[n]*L2[n]...*LK[n]].
#Try
#multSeq2K([1/(1-x-y-x*y),1/(1-x-y-x*y)],x,y,20);
multSeq2K:=proc(Lf,x,y,n) local L,i,t,R:

for i from 1 to nops(Lf) do
      if abs(subs({x=0,y=0},denom(Lf[i]))) <> 1 or abs(numer(Lf[i])) <> 1 then
            print(`The function`, Lf[i], `should be of the form 1/(1-a*x-b*y-c*x*y`):
            RETURN(FAIL):
      fi:
od:

t:=nops(Lf):
L:=[]:
R:=[]:

for i from 1 to t do
      L:=[op(L), DiagSeq2(Lf[i],x,y,n)]:
od:

R:=L[1]:

for i from 2 to t do
      R:=R*~L[i]:
od:

R:
end:

#multSeq3K(L,x,y,z,n): Given an arbitrarily long list L consisting of rational functions in
three variables {x,y,z} of the form 1/(1-a*x-b*x-c*x*y-d*x*z-e*y*z-f*x*y*z), will return the
element by element product of the first n terms in all sequences, i.e.,
[L1[1]*L2[1]*...*LK[1], L1[2]*L2[2]*...*LK[2], ..., L1[n]*L2[n]...*LK[n]].
```

```
#Try
#multSeq3K([1/(1-x-y-z),1/(1-x-y-z)],x,y,z,20);
multSeq3K:=proc(Lf,x,y,z,n) local L,i,t,R:

for i from 1 to nops(Lf) do
      if abs(subs({x=0,y=0,z=0},denom(Lf[i]))) <> 1 or abs(numer(Lf[i])) <> 1 then
            print(`The function`, Lf[i], `should be of the form
1/(1-a*x-b*x-c*x*y-d*x*z-e*y*z-f*x*y*z)`):
            RETURN(FAIL):
      fi:
od:

t:=nops(Lf):
L:=[]:
R:=[]:

for i from 1 to t do
      L:=[op(L), DiagSeq3(Lf[i],x,y,z,n)]:
od:

R:=L[1]:

for i from 2 to t do
      R:=R*~L[i]:
od:

R:
end:

#findRatios2(f1,f2,x,y,n): Given two rational functions {f1,f2} in the variables {x,y}.
Returns the ratios element-wise between the diagonal sequences generated by f1,f2, i.e.,
[L1[1]/L2[1], L1[2]/L2[2], ..., L1[n]/L2[n]]
#Try
#findRatios2(1/(1-x-y-x*y),1/(1-x-y-2*x*y),x,y,20);
findRatios2:=proc(f1,f2,x,y,n) local L1,L2:

if abs(subs({x=0,y=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form 1/(1-a*x-b*y-c*x*y`):
```

```
      RETURN(FAIL):
fi:

L1:=DiagSeq2(f1,x,y,n):
L2:=DiagSeq2(f2,x,y,n):

L1/~L2:

end:

#findRatios3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in the variables {x,y,z}.
Returns the ratios element-wise between the diagonal sequences generated by f1,f2, i.e.,
[L1[1]/L2[1], L1[2]/L2[2], ..., L1[n]/L2[n]]
#Try
#findRatios3(1/(1-x-y-z),1/(1-x-y-2*z),x,y,z,20);
findRatios3:=proc(f1,f2,x,y,z,n) local L1,L2:

if abs(subs({x=0,y=0,z=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0,z=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:

L1:=DiagSeq3(f1,x,y,z,n):
L2:=DiagSeq3(f2,x,y,z,n):

L1/~L2:

end:

#addDiagSeq2(f1,f2,x,y,n): Given two rational functions {f1,f2} in the variables {x,y}.
Returns the addition element-wise between the diagonal sequences generated by f1,f2, i.e.,
[L1[1]+L2[1], L1[2]+L2[2], ..., L1[n]+L2[n]]
#Try
#addDiagSeq2(1/(1-x-y-x*y),1/(1-x-y-2*x*y),x,y,20);
addDiagSeq2:=proc(f1,f2,x,y,n) local L1,L2:
```

```
if abs(subs({x=0,y=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:

L1:=DiagSeq2(f1,x,y,n):
L2:=DiagSeq2(f2,x,y,n):

L1+L2:

end:

#addDiagSeq3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in the variables {x,y,z}.
Returns the addition element-wise between the diagonal sequences generated by f1,f2, i.e.,
[L1[1]+L2[1], L1[2]+L2[2], ..., L1[n]+L2[n]]
#Try
#addDiagSeq3(1/(1-x-y-z),1/(1-x-y-2*z),x,y,z,20);
addDiagSeq3:=proc(f1,f2,x,y,z,n) local L1,L2:

if abs(subs({x=0,y=0,z=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0,z=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:

L1:=DiagSeq3(f1,x,y,z,n):
L2:=DiagSeq3(f2,x,y,z,n):

L1+L2:

end:
```

```
#subDiagSeq2(f1,f2,x,y,n): Given two rational functions {f1,f2} in the variables {x,y}.
Returns the subtraction element-wise between the diagonal sequences generated by f1,f2, i.e.,
[L1[1]-L2[1], L1[2]-L2[2], ..., L1[n]-L2[n]]
#Try
#subDiagSeq2(1/(1-x-y-x*y),1/(1-x-y-2*x*y),x,y,20);
subDiagSeq2:=proc(f1,f2,x,y,n) local L1,L2:

if abs(subs({x=0,y=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:

L1:=DiagSeq2(f1,x,y,n):
L2:=DiagSeq2(f2,x,y,n):

L1-L2:

end:

#subDiagSeq3(f1,f2,x,y,z,n): Given two rational functions {f1,f2} in the variables {x,y,z}.
Returns the subtraction element-wise between the diagonal sequences generated by f1,f2, i.e.,
[L1[1]-L2[1], L1[2]-L2[2], ..., L1[n]-L2[n]]
#Try
#subDiagSeq3(1/(1-x-y-z),1/(1-x-y-2*z),x,y,z,20);
subDiagSeq3:=proc(f1,f2,x,y,z,n) local L1,L2:

if abs(subs({x=0,y=0,z=0},denom(f1))) <> 1 or abs(numer(f1)) <> 1 then
      print(`The function f1`, f1, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:
if abs(subs({x=0,y=0,z=0},denom(f2))) <> 1 or abs(numer(f2)) <> 1 then
      print(`The function f2`, f2, `should be of the form
1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:

L1:=DiagSeq3(f1,x,y,z,n):
```

```
L2:=DiagSeq3(f2,x,y,z,n):

L1-L2:

end:

#diagSeq2KComps(f,k,x,y,n): Given a generating function f and an integer k, returns the first
n terms in the diag seq of f^k/k!, which means exactly k components depending on the context.
#Try
#diagSeq2Kcomps(1/(1-x-y-x*y),2,x,y,20);
diagSeq2Kcomps:=proc(f,k,x,y,n) local f1:
if abs(subs({x=0,y=0},denom(f))) <> 1 or abs(numer(f)) <> 1 then
      print(`The function f should be of the form 1/(1-a*x-b*y-c*x*y`):
      RETURN(FAIL):
fi:

f1:=f^k/k!:
DiagSeq2(f1,x,y,n):

end:

#diagSeq3KComps(f,k,x,y,z,n): Given a generating function f and an integer k, returns the
first n terms in the diag seq of f^k/k!, which means exactly k components depending on the
context.
#Try
#diagSeq3Kcomps(1/(1-x-y-z),2,x,y,z,20);
diagSeq3Kcomps:=proc(f,k,x,y,z,n) local f1:
if abs(subs({x=0,y=0,z=0},denom(f))) <> 1 or abs(numer(f)) <> 1 then
      print(`The function f should be of the form 1/(1-a*x-b*y-c*x*y-d*x*z-e*y*z-f*x*y*z`):
      RETURN(FAIL):
fi:

f1:=f^k/k!:
DiagSeq3(f1,x,y,z,n):

end:
```

Module 3

1. Analytics Functions: Plots, Moments, Ratios

Analytics Functions

```
#Borrowed from Dr.Z, lecture 22, Math 454
#AveAndMoms(f,x,K): given an ordinary generating function f of x, that is a weight-enumerator
of some
#numerical attribute,and a positive integer K,
#outputs (i) The average  (ii) the standar-deviation, (iii) the 3rd through the K-th SCALED
moments about the mean. Try
#AveAndMoms((1+x)^1000,x,6);
AveAndMoms:=proc(f,x,K) local f1,av,i,L,sd:
if subs(x=1,f)=0 then
 RETURN(FAIL):
fi:

f1:=f/subs(x=1,f):

av:=subs(x=1,diff(f1,x)):

f1:=f1/x^av:

f1:=x*diff(x*diff(f1,x),x):

sd:=sqrt(subs(x=1,f1)):

L:=[av,sd]:

for i from 3 to K do
f1:=x*diff(f1,x):
L:=[op(L),subs(x=1,f1)/sd^i]:
od:

evalf(L):

end:

#plotVals(L1,L2): Given {L1}, a list of x-coordinates, and {L2}, a list of y-coordinates,
where L1 and L2 have the same size, will return a plot of L2 vs. L1
#Try
#plotVals([1,2,3],[4,5,6])
plotVals:=proc(L1,L2):
if nops(L1) <> nops(L2) then
      print(`L1 and L2 should have the same size`):
```

```
      RETURN(FAIL):
fi:

plot(Vector(L1),Vector(L2),style=point):

end:
```