This Maple sheet belongs to the article "Construction and implementation of asymptotic expansions for Jacobi-type orthogonal polynomials" by Alfredo. Deaño , Daan Huybrechs and Peter Opsomer

Initialising

```
> restart;
> with(LinearAlgebra):
```

Comment assignment of alpha, beta and/or h(x) if you want the symbolical results.
Taking h not equal to one is OK for computing the asymptotic expansions, but computing the exact polynomials will require much time.

```
> alpha := -0.3;   beta := 1/sqrt(2);   h:= unapply(1,x);
```

$$\alpha := -0.3$$

$$\beta := \frac{1}{2}\sqrt{2}$$

$$h := x \rightarrow 1$$

Need enough Digits: if Digits := 40; then there is suddenly a relative error of about 1 for n > 50

```
> Digits := 100: interface(displayprecision=10):
> w := x -> (1-x)^alpha*(1+x)^beta*h(x):
```

maxOrder is the maximum order of error we will see, defined here because it determines the number of c_n/d_n.

```
> maxOrder := 7:
> s3 := arg -> Matrix([[arg,0],[0,arg^(-1)]]):
```

5.1 Square roots and other algebraic singularities

```
> phi := z -> z + sqrt(z-1)*sqrt(z+1):
```

To avoid the wrong branch cuts of sqrt(z^2-1) along the imaginary axis (which would give a wrong phitilde for Re(z) positive), we define phitilde as phitilde(z) = phi(-z) = -phi(z):

```
> phitilde := z -> -phi(z):
```

5.2 Computation of contour integrals

```
> rho := 1.25:
> zeta := theta -> rho/2*exp(I*theta) + 1/rho/2*exp(-I*theta):
> T := 400:

> integrandCD := unapply(ln(h(zeta) )/(sqrt(zeta+1)*sqrt(zeta-1)*
  ( zeta +(-1)^isC)^(n+1) ), isC, zeta,n):
> nrcd := ceil(maxOrder/2):
> for nn from 0 to nrcd do
    ct(nn) := evalf(1/(2*Pi*I)*2*Pi/T*sum(integrandCD(1, zeta(2*
  Pi*t/T),nn)*(D(zeta)(2*Pi*t/T) ), t = 0..(T-1) ) );
    dt(nn) := evalf(1/(2*Pi*I)*2*Pi/T*sum(integrandCD(0, zeta(2*
  Pi*t/T),nn)*(D(zeta)(2*Pi*t/T) ), t = 0..(T-1) ) );
  end do:

> integrandPsi := unapply(log(h(zeta) )/sqrt(zeta-1)/sqrt(zeta+1)
  /(zeta - x), zeta,x):
> psit := x -> evalf(1/2*(alpha*(arccos(x)-Pi) + beta*arccos(x) )
  + sqrt(1-x)*sqrt(1+x)/(2*I*T)*sum(integrandPsi(zeta(2*Pi*t/T),
  x)*(D(zeta)(2*Pi*t/T) ), t = 0..(T-1) ) ):

> integrandDinf := unapply(ln(h(zeta) ) /(sqrt(zeta-1)*sqrt
  (zeta+1) ), zeta):
> Dinft := evalf(2^(-alpha/2-beta/2)*exp(sum(integrandDinf(zeta
  (2*Pi*t/T) )*(D(zeta)(2*Pi*t/T) ), t = 0..(T-1) )/(4*Pi*I)*2*
  Pi/T) ):
```

Exact results for h = 1 and from residue calculus, filled in for h = exp(-x^2) as mentioned in Section 6.1

```
> if int(abs(h(x) - 1), x = -1.0..1.0,numeric) = 0 then
    for nn from 0 to nrcd do
       c(nn) := 0:
       d(nn) := 0:
     od:
    psi := x -> evalf(1/2*(alpha*(arccos(x)-Pi) + beta*arccos(x)
  ) ):
    Dinf := evalf(2^(-alpha/2-beta/2) ):

  elif int(abs(h(x) - exp(-x^2) ), x = -1.0..1.0,numeric) = 0
  then
     c(0) := -1:   c(1) := -1:
     d(0) := 1: d(1) := -1:
    for nn from 2 to nrcd do
       c(nn) := 0:
       d(nn) := 0:
     od:
    psi := x -> evalf(1/2*(alpha*(arccos(x)-Pi) + beta*arccos(x)
```

```
) -x*sqrt(1-x)*sqrt(1+x)/2):
   Dinf := evalf(2^(-alpha/2-beta/2)*exp(-1/4) ):

elif assigned(alpha) then
   for nn from 0 to nrcd do
     serc := series(1/t^2*integrandCD(1,1/t,nn), t=0) assuming
t::real:
      c(nn) := evalf(coeff(serc,t^(-1) ) ):
     serd := series(1/t^2*integrandCD(0,1/t,nn), t=0) assuming
t::real:
      d(nn) := evalf(coeff(serd,t^(-1) ) ):
    od:
   serp := unapply(series(1/t^2*integrandPsi(1/t,x), t=0), x)
assuming t::real:
   psi := unapply(evalf(1/2*(alpha*(arccos(x)-Pi) + beta*arccos
(x) ) + sqrt(1-x)*sqrt(1+x)/2*coeff(serp(x), t^(-1) ) ),x):
   serD := series(1/t^2*integrandDinf(1/t), t=0) assuming
t::real:
   Dinf := evalf(2^(-alpha/2-beta/2)*exp(1/2*coeff(serD, t^(-1)
) ) ):
end if:
```

```
> nn := 'nn':
```

Checking correctness of trapezoidal rules, comparing with exact results from residue calculus:

```
> if assigned(alpha) then
    c(0) -ct(0);
    d(0) -dt(0);
    psi(0.6) -psit(0.6);
    Dinf -Dinft;
   if abs((Dinf-Dinft)/Dinf)> 2^(100-T) then
     print("Trapezoidal rule and residue calculus don't agree,
check whether h(x) is analytic."):
    end if:
end if;
```

$$0.0000000000$$

$$0.0000000000$$

$$0.0000000000 + 0.0000000000\,I$$

$$0.0000000000$$

Using 'exact' result afterwards unless de-commenting to get the result from the trapezoidal rules:

```
> #c := ct: d := dt: psi := x -> psit(x): Dinf := Dinft:
```

3 Computation of higher order terms using 4 Simplifications and explicit formulas

3.2 The definition of Delta_k(z)

```
> FrightExact := unapply(exp(signum(0,argument(x-1),-1)*I*(psi
  (x)+alpha*Pi/2) ),x):
```

\theta(z) = signum(0,argument(z-1),-1): For the right disk, (1-delta,1] belongs to the upper half plane so by symmetry around the origin, [-1,delta-1) belongs to lower half plane and (-infty,-1] belongs to upper half plane, also for Fleft (because else bpiL[1] gives about same error curve on the interval as bpiL[2]).

```
> FleftExact := unapply(exp(signum(0,argument(x-1),-1)*I*(psi
  (x)-beta*Pi/2) ), x):
```

```
> brac := proc(ab,k::integer) if k = 0 then return 1; else
  return product( (4*ab^2-(2*ii-1)^2 ), ii=1..k )/(2^(2*k)*(k!
  )); end if; end proc:
> M := unapply(Matrix([[sqrt(phi(z) ), I/sqrt(phi(z) )], [-
  I/sqrt(phi(z) ), sqrt(phi(z) )]])/(sqrt(2)*(z-1)^(1/4)*(z+1)
  ^(1/4) ),z):
```

Below the exact Delta_m (without series expansions) for the expansion of the polynomial in the boundary regions:

```
> DeltaRightExact := (k::integer,z) -> brac(alpha,k-1)/2^k/(ln
  (phi(z) ) )^k*s3(Dinf).M(z).s3(FrightExact(z) ).Matrix([[ (
  (-1)^k)/k*(alpha^2+k/2-1/4),-I*(k-1/2)],[(-1)^k*I*(k-1/2),
  (alpha^2+k/2-1/4)/k]]).s3( (FrightExact(z) )^(-1) ).
  Transpose(M(z) ).s3(Dinf^(-1) ):
> DeltaLeftExact := (k::integer,z) -> brac(beta,k-1)/2^k/(ln
  (phitilde(z) ) )^k*s3(Dinf).M(z).s3(FleftExact(z) ).Matrix([
  [ ((-1)^k)/k*(beta^2+k/2-1/4),I*(k-1/2)],[(-1)^(k+1)*I*
  (k-1/2),(beta^2+k/2-1/4)/k]]).s3( (FleftExact(z) )^(-1) ).
  Transpose(M(z) ).s3(Dinf^(-1) ):
```

B Proof of Proposition 4.2

This illustrates the proof of the simplifications of Appendix B, with a change of m to m+1 and using general alpha -> aalpha.
DeltaRcanc is Delta without the matrices that will cancel in the end, for ease of notation.

```
> DeltaRcanc := (k,z) -> product( (4*aalpha^2-(2*ii-1)^2 ),
  ii=1..k-1 )/(2^(2*k-2)*( (k-1)!))/2^k/(ln(phi(z) ) )^k*
  Matrix([[ ((-1)^k)/k*(aalpha^2+k/2-1/4),-I*(k-1/2)],[(-1)^k*
```

```
l*(k-1/2),(aalpha^2+k/2-1/4)/k]]):
> extraMatrix := (i,z) -> -(4*aalpha^2+2*i-1)/ln(phi(z) )^i*
  product( (4*aalpha^2-(2*ii-1)^2 ), ii=1..i-1 )/(2^(2*i-2)*(
  (i-1)!))/2^(i+1)/i*Matrix(2,2,shape=identity):
> m := 'm': j := 'j':
```

Proof (general case: no specific j) that s_j D_{m+1-j} +s_{m+1-j} D_j = 0 for **m even** and j odd and even:

```
> map(fnormal,simplify(DeltaRcanc(j,z).DeltaRcanc(m+1-j,z)+
  (DeltaRcanc(m+1-j,z) +extraMatrix(m+1-j,z) ).DeltaRcanc(j,z)
  ) ) assuming j::odd,m::even;
```

$$\begin{bmatrix} 0.0000000000 & 0.0000000000 \\ 0.0000000000 & 0.0000000000 \end{bmatrix}$$

```
> map(fnormal,simplify( (DeltaRcanc(j,z) +extraMatrix(j,z) ).
  DeltaRcanc(m+1-j,z)+DeltaRcanc(m+1-j,z).DeltaRcanc(j,z) ) )
  assuming j::even, m::even;
```

$$\begin{bmatrix} 0.0000000000 & 0.0000000000 \\ 0.0000000000 & 0.0000000000 \end{bmatrix}$$

m odd, j odd and even:

```
> jodd := simplify( DeltaRcanc(j,z).DeltaRcanc(m+1-j,z)+
  DeltaRcanc(m+1-j,z).DeltaRcanc(j,z) ) assuming j::odd,
  m::odd:
> jeven := simplify( (DeltaRcanc(j,z)+extraMatrix(j,z) ).
  DeltaRcanc(m+1-j,z)+(DeltaRcanc(m+1-j,z) +extraMatrix(m+1-j,
  z) ).DeltaRcanc(j,z) ) assuming j::even, m::odd:
> map(fnormal,simplify(jodd+jeven) );
```

$$\begin{bmatrix} 0.0000000000 & 0.0000000000 \\ 0.0000000000 & 0.0000000000 \end{bmatrix}$$

Sum up previous results for odd m from j=1 to m/2=n taking the s_n D_n term out of the sum. This is an illustration of the proof of the simplifications with the telescopic sum, with a slight adjustment to notation outputted by Maple.

```
> res := (-1)^n*subs(j=n,(1-cos((1/2)*Pi*(2*aalpha+1))^2)*4^(-
  j)*ln(phi(z))^(-2*j)*(-2*j+2*aalpha+1)*(4*aalpha^2+4*j^2-1)*
  (2*j+2*aalpha-1)*GAMMA(j+aalpha-1/2)^2*GAMMA(j-aalpha-1/2)
  ^2/(16*Pi^2*GAMMA(j+1)^2) )   +   sum( (-1)^j*subs(m=2*n-1,
  (1/16)*ln(phi(z))^(-1-m)*GAMMA(m+1/2-j+aalpha)*GAMMA(m+1/2-
  j-aalpha)*GAMMA(j+aalpha-1/2)*GAMMA(j-aalpha-1/2)*(1-cos(
  (1/2)*Pi*(2*aalpha+1))^2)*2^(-m)*(16*aalpha^4-16*j^4+32*j^3*
  m-16*j^2*m^2+8*aalpha^2*m+32*j^3-40*j^2*m+8*j*m^2-24*j^2+16*
  j*m+8*j-2*m-1)/(Pi^2*GAMMA(j+1)*GAMMA(m+2-j)) ),j=1..n-1):
> simplify( (res-extraMatrix(2*n,z)(1))/extraMatrix(2*n,z)(1)
```

```
) assuming n::integer;
```

0

4.1 Simplifications

Extra matrices in s_k^{right/left}(z) for the expansions of the polynomial in the boundary regions later, implicitly present in the computation of the W's.

```
> for i from 1 to (maxOrder-1) do
    if i mod 2 = 0 then
      sR[i] := -(4*alpha^2+2*i-1)/ln(phi(z) )^i*brac(alpha,
  i-1)/2^(i+1)/i*Matrix(2,2,shape=identity):
      sL[i] := -(4*beta^2+2*i-1)/ln(phitilde(z) )^i*brac
  (beta,i-1)/2^(i+1)/i*Matrix(2,2,shape=identity):
    else
      sR[i] := Matrix(2,2):
      sL[i] := Matrix(2,2):
    end if:
    sR[i] := unapply(sR[i],z):
    sL[i] := unapply(sL[i],z):
  od:
```

4.2 Precomputing a series expansion for s_k^{right}(z)

Computing series expansion of (ln(phi(v+1)))^(-k)

```
> f := n -> pochhammer(1/2,n)/(-2)^n/(n!)/(1+2*n):

> g[1,0] := 1:
> for n from 1 to round( (maxOrder-2)/2) do
    g[1,n] := -add(g[1,jj]*f(n-jj),jj=0..n-1):
  od:
> for k from 2 to (maxOrder-1) do
    for n from 0 to round( (maxOrder-2)/2) do
      g[k,n] := add(g[k-1,l]*g[1,n-l],l=0..n):
    end do:
  end do:
```

Computing series expansion of entries of s_k without constants, constant functions and constant matrices

```
> for n from 0 to round( (maxOrder-2)/2) do
    UpsOdd[n] := binomial(-1/2,n)/2^(n+1/2)*(2*n+1)/(2*n-1):
    UpsEven[n] := 0:
  od:
  UpsEven[0] := 1:
```

```
> for n from 0 to round( (maxOrder-2)/2) do
    tmp := sum(binomial(1/2,jj)*(1/2)^jj*c(n-jj), jj=0..n);
    rho0[1,n] := f(n)*(alpha+beta) +tmp; # = rho_{1,n,alpha+beta}
    rhop[1,n] := f(n)*(alpha+beta+1) +tmp; # = rho_{1,n,alpha+beta+1}
    rhom[1,n] := f(n)*(alpha+beta-1) +tmp; # = rho_{1,n,alpha+beta-1}
  od:
```

```
> for k from 2 to (maxOrder-1) do
    for n from 0 to round( (maxOrder-2)/2) do
      rho0[k,n] := add(rho0[k-1,jj]*rho0[1,n-jj], jj=0..n): # = rho_{k,n,alpha+beta}
      rhop[k,n] := add(rhop[k-1,jj]*rhop[1,n-jj], jj=0..n): # = rho_{k,n,alpha+beta+1}
      rhom[k,n] := add(rhom[k-1,jj]*rhom[1,n-jj], jj=0..n): # = rho_{k,n,alpha+beta-1}
    od:
  od:
```

```
> for n from 0 to round( (maxOrder-2)/2) do
    H0Odd[n] := 2*add(2^jj*rho0[2*jj,n-jj]/( (2*jj) !), jj=1..n): # = H_{n,alpha+beta}^{odd}
    HpOdd[n] := 2*add(2^jj*rhop[2*jj,n-jj]/( (2*jj) !), jj=1..n): # = H_{n,alpha+beta+1}^{odd}
    HmOdd[n] := 2*add(2^jj*rhom[2*jj,n-jj]/( (2*jj) !), jj=1..n): # = H_{n,alpha+beta-1}^{odd}
    if (type(maxOrder,even) or not (n = round( (maxOrder-2)/2 ) ) ) then
      H0Even[n] := 2^(3/2)*add(2^jj*rho0[2*jj+1,n-jj]/( (2*jj+1) !), jj=0..n): # = H_{n,alpha+beta}^{even}
      HpEven[n] := 2^(3/2)*add(2^jj*rhop[2*jj+1,n-jj]/( (2*jj+1) !), jj=0..n): # = H_{n,alpha+beta+1}^{even}
      HmEven[n] := 2^(3/2)*add(2^jj*rhom[2*jj+1,n-jj]/( (2*jj+1) !), jj=0..n): # = H_{n,alpha+beta-1}^{even}
    end if:
  od:
  H0Odd[0] := 2:
  HpOdd[0] := 2:
  HmOdd[0] := 2:
```

```
> for n from 0 to round( (maxOrder-2)/2) do
    X0Odd[n] := add(binomial(-1/2,jj)*2^(-jj-3/2)*H0Odd[n-jj], jj=0..n): # = X_{n,alpha+beta}^{odd}
    XpOdd[n] := add(binomial(-1/2,jj)*2^(-jj-3/2)*HpOdd[n-jj], jj=0..n): # = X_{n,alpha+beta+1}^{odd}
```

```
    XmOdd[n] := add(binomial(-1/2,jj)*2^(-jj-3/2)*HmOdd[n-jj], jj=0..n): # = X_{n,alpha+beta-1}^{odd}
    if (type(maxOrder,even) or not (n = round( (maxOrder-2)/2 ) ) ) then
      X0Even[n] := add(binomial(-1/2,jj)*2^(-jj-3/2)*H0Even[n-jj], jj=0..n): # = H_{n,alpha+beta}^{even}
      XpEven[n] := add(binomial(-1/2,jj)*2^(-jj-3/2)*HpEven[n-jj], jj=0..n): # = X_{n,alpha+beta+1}^{even}
      XmEven[n] := add(binomial(-1/2,jj)*2^(-jj-3/2)*HmEven[n-jj], jj=0..n): # = X_{n,alpha+beta-1}^{even}
    end if:
od:
```

```
> for k from 1 to (maxOrder-1) do
    mo := round( (maxOrder-1-k)/2)-1+round(k/2):
    if(k::odd) then
      for n from 0 to mo do
        T[n] := Matrix([[(alpha^2+k/2-1/4)/k*UpsOdd[n] + (k-1/2)*X0Odd[n]   ,   Dinf^2*(2*I*(alpha^2+k/2-1/4)/k*binomial(-1/2,n)*2^(-n-3/2) -I*(k-1/2)*XpOdd[n]) ]   ,   [(2*I*(alpha^2+k/2-1/4)/k*binomial(-1/2,n)*2^(-n-3/2) -I*(k-1/2)*XmOdd[n])/Dinf^2      ,      -(alpha^2+k/2-1/4)/k*UpsOdd[n] - (k-1/2)*X0Odd[n] ]]): # = T_{k,n}^{odd}
        Wr[k,n-(k+1)/2] := brac(alpha,k-1)/2^(3*k/2)*add(g[k,jj]*T[n-jj], jj=0..n): # = W_{k, m = n-(k+1)/2}^{right}
      od:
    else
      for n from 0 to mo do
        T[n] := Matrix([[(alpha^2+k/2-1/4)/k*UpsEven[n] + (k-1/2)*X0Even[n]   ,   -I*Dinf^2*(k-1/2)*XpEven[n] ]   ,   [-I*(k-1/2)*XmEven[n]/Dinf^2   ,   (alpha^2+k/2-1/4)/k*UpsEven[n] - (k-1/2)*X0Even[n] ]]): # = T_{k,n}^{even}
        Wr[k,n-k/2] := brac(alpha,k-1)/2^(3*k/2)*(-(4*alpha^2+2*k-1)/2/k*g[k,n]*Matrix([[1,0],[0,1]]) + add(g[k,jj]*T[n-jj], jj=0..n) ): # = W_{k, m = n-k/2}^{right}
      od:
    end if:
od:
```

4.2 Precomputing a series expansion for s_k^{left}(z)

Unassign to avoid bugs due to a conflict with previous computations for right W's

```
> f := 'f': g := 'g': UpsOdd := 'UpsOdd': UpsEven := 'UpsEven': tmp := 'tmp': rho0 := 'rho0': rhop := 'rhop':
```

```
rhom := 'rhom': H0Odd := 'H0Odd': HpOdd := 'HpOdd': HmOdd :=
'HmOdd': H0Even := 'H0Even': HpEven := 'HpEven': HmEven :=
'HmEven': X0Odd := 'X0Odd': XpOdd := 'XpOdd': XmOdd :=
'XmOdd': X0Even := 'X0Even': XpEven := 'XpEven': XmEven :=
'XmEven': T := 'T':
> f := n -> pochhammer(1/2,n)/2^n/(n!)/(1+2*n):

> g[1,0] := 1:
> for n from 1 to round( (maxOrder-2)/2) do
    g[1,n] := -add(g[1,jj]*f(n-jj),jj=0..n-1):
  od:
> for k from 2 to (maxOrder-1) do
    for n from 0 to round( (maxOrder-2)/2) do
      g[k,n] := add(g[k-1,l]*g[1,n-l],l=0..n):
    end do:
  end do:

> for n from 0 to round( (maxOrder-2)/2) do
    UpsOdd[n] := sqrt(-1/2)/(-2)^n*binomial(-1/2, n)*(2*n+1)/
  (2*n-1):
    UpsEven[n] := 0:
  od:
  UpsEven[0] := 1:

> for n from 0 to round( (maxOrder-2)/2) do
    tmp := -add(binomial(1/2,jj)*(-1/2)^jj*d(n-jj), jj=0..n);
    rho0[1,n] := f(n)*(alpha+beta) +tmp; # = rho_{1,n,alpha+
  beta}
    rhop[1,n] := f(n)*(alpha+beta+1) +tmp; # = rho_{1,n,
  alpha+beta+1}
    rhom[1,n] := f(n)*(alpha+beta-1) +tmp; # = rho_{1,n,
  alpha+beta-1}
  od:

> for k from 2 to (maxOrder-1) do
    for n from 0 to round( (maxOrder-2)/2) do
      rho0[k,n] := sum(rho0[k-1,jj]*rho0[1,n-jj], jj=0..n):
  # = rho_{k,n,alpha+beta}
      rhop[k,n] := sum(rhop[k-1,jj]*rhop[1,n-jj], jj=0..n):
  # = rho_{k,n,alpha+beta+1}
      rhom[k,n] := sum(rhom[k-1,jj]*rhom[1,n-jj], jj=0..n):
  # = rho_{k,n,alpha+beta-1}
    od:
  od:

> for n from 0 to round( (maxOrder-2)/2) do
    H0Odd[n] := 2*sum((-2)^jj*rho0[2*jj,n-jj]/( (2*jj) !),
```

```
jj=1..n): # = H_{n,alpha+beta}^{odd}
   HpOdd[n] := 2*sum((-2)^jj*rhop[2*jj,n-jj]/( (2*jj) !),
jj=1..n): # = H_{n,alpha+beta+1}^{odd}
   HmOdd[n] := 2*sum((-2)^jj*rhom[2*jj,n-jj]/( (2*jj) !),
jj=1..n): # = H_{n,alpha+beta-1}^{odd}
   if (type(maxOrder,even) or not (n = round( (maxOrder-2)/2
) ) ) then
      H0Even[n] := (-2)^(3/2)*sum((-2)^jj*rho0[2*jj+1,n-jj]/
( (2*jj+1) !), jj=0..n): # = H_{n,alpha+beta}^{even}
      HpEven[n] := (-2)^(3/2)*sum((-2)^jj*rhop[2*jj+1,n-jj]/
( (2*jj+1) !), jj=0..n): # = H_{n,alpha+beta+1}^{even}
      HmEven[n] := (-2)^(3/2)*sum((-2)^jj*rhom[2*jj+1,n-jj]/
( (2*jj+1) !), jj=0..n): # = H_{n,alpha+beta-1}^{even}
   end if:
od:
H0Odd[0] := 2:
HpOdd[0] := 2:
HmOdd[0] := 2:
```

```
> for n from 0 to round( (maxOrder-2)/2) do
   X0Odd[n] := sum(binomial(-1/2,jj)*(-2)^(-jj)*H0Odd[n-jj],
jj=0..n)/sqrt(-8):
   XpOdd[n] := sum(binomial(-1/2,jj)*(-2)^(-jj)*HpOdd[n-jj],
jj=0..n)/sqrt(-8):
   XmOdd[n] := sum(binomial(-1/2,jj)*(-2)^(-jj)*HmOdd[n-jj],
jj=0..n)/sqrt(-8):
   if (type(maxOrder,even) or not (n = round( (maxOrder-2)/2
) ) ) then
      X0Even[n] := sum(binomial(-1/2,jj)*(-2)^(-jj)*H0Even
[n-jj], jj=0..n)/sqrt(-8):
      XpEven[n] := sum(binomial(-1/2,jj)*(-2)^(-jj)*HpEven
[n-jj], jj=0..n)/sqrt(-8):
      XmEven[n] := sum(binomial(-1/2,jj)*(-2)^(-jj)*HmEven
[n-jj], jj=0..n)/sqrt(-8):
   end if:
od:
```

```
> for k from 1 to (maxOrder-1) do
   mo := round( (maxOrder-1-k)/2)-1+round(k/2):
   if(k::odd) then
      for n from 0 to mo do
         T[n] := Matrix([[(beta^2+k/2-1/4)/k*UpsOdd[n] -
(k-1/2)*X0Odd[n]   ,   Dinf^2*(2*I*(beta^2+k/2-1/4)/k*
binomial(-1/2,n)*(-2)^(-n)/sqrt(-8) -I*(k-1/2)*XpOdd[n]) ]
,  [(2*I*(beta^2+k/2-1/4)/k*binomial(-1/2,n)*(-2)^(-n)/sqrt
(-8) -I*(k-1/2)*XmOdd[n])/Dinf^2   ,   -(beta^2+k/2-1/4)/k*
UpsOdd[n] + (k-1/2)*X0Odd[n] ]]): # = T_{k,n}^{odd}
         Wl[k,n-(k+1)/2] := brac(beta,k-1)/(-2)^(3*k/2)*add
```

```
(g[k,jj]*T[n-jj], jj=0..n): # = W_{k,m=n-(k+1)/2}^{left}
      od:
   else
     for n from 0 to mo do
       T[n] := Matrix([[(beta^2+k/2-1/4)/k*UpsEven[n] -
(k-1/2)*X0Even[n]  ,  -I*Dinf^2*(k-1/2)*XpEven[n] ]  ,
[-I*(k-1/2)*XmEven[n]/Dinf^2  ,  (beta^2+k/2-1/4)/k*
UpsEven[n] + (k-1/2)*X0Even[n] ]]): # = T_{k,n}^{even}
       Wl[k,n-k/2] := brac(beta,k-1)/(-2)^(3*k/2)*(-(4*
beta^2+2*k-1)/2/k*g[k,n]*Matrix([[1,0],[0,1]]) +add(g[k,jj]*
T[n-jj], jj=0..n)): # = W_{k,m=n-k/2}^{left}
      od:
   end if:
od:
```

(4.8) U_{k,m}^{right/left} and 3.3 Recursive computation of R_k(z)

```
> R[0] := Matrix(2,2,shape=identity):
> Rright[0] := Matrix(2,2,shape=identity):
  Rleft[0] := Matrix(2,2,shape=identity):

> l := 'l': j := 'j': n := 'n': i := 'i':
  for k from 1 to (maxOrder-1) do
    pp := round(k/2):

    for m from 1 to pp do
      Uright[k,m] := Wr[k,-m] +add(add(Uright[k-j,l].Wr[j,l-
  m], l = max( (m-round(j/2) ),1)..round((k-j)/2) ), j=1..
  (k-1) ) +add(add(add(pochhammer(1-i-n,n)/2^(i+n)/(n!)*Uleft
  [k-j,i], i=1..round((k-j)/2) ).Wr[j,-n-m], n = 0..(round
  (j/2)-m) ), j=1..(k-1) ):

      Uleft[k,m] := Wl[k,-m] +add(add(Uleft[k-j,l].Wl[j,l-
  m], l = max( (m-round(j/2) ),1)..round((k-j)/2) ), j=1..
  (k-1) ) +add(add(add(pochhammer(1-i-n,n)/(-2)^(i+n)/(n!)*
  Uright[k-j,i], i=1..round((k-j)/2) ).Wl[j,-n-m], n = 0..
  (round(j/2)-m) ), j=1..(k-1) ):

      if assigned(alpha) then
        Uright[k,m] := evalf(Uright[k,m]):
        Uleft[k,m] := evalf(Uleft[k,m]):
      else
        Uright[k,m] := simplify(Uright[k,m]):
        Uleft[k,m] := simplify(Uleft[k,m]):
      end if:
```

```
    od:

   R[k] := add(Uright[k,mm]/(z-1)^mm + Uleft[k,mm]/(z+1)^mm,
mm = 1..pp):
   Rright[k] := R[k] - add(R[k-mm].(DeltaRightExact(mm,z) +
sR[mm](z) ), mm = 1..k ):
   Rleft[k] := R[k] - add(R[k-mm].(DeltaLeftExact(mm,z) +sL
[mm](z) ), mm = 1..k ):
od:
k := 'k':
```

You can use this to save the symbolical or numerical results for U to use them in Matlab (or CodeGeneration[Fortran] or ...)

```
> if false then
    if FileTools[Exists]("Utest.m") then
      FileTools[Remove]("Utest.m"):
    end if:
    for k from 1 to maxOrder-1 do
      for m from 1 to round(k/2) do
        CodeGeneration[Matlab](Uright[k,m], output =
"Utest.m", resultname = cat("UrightQ",k,m) ):
        CodeGeneration[Matlab](Uleft[k,m], output = "Utest.
m", resultname = cat("UleftQ",k,m) ):
      od:
    od:
end if:
```

A Expressions for the first four higher order terms

Check results with explicit results from the appendix for first three terms, mixed with other notation.

```
> A1 := (4*alpha^2-1)/16*s3(Dinf).Matrix([[-1,I],[I,1]]).s3
  (Dinf^(-1) ):
> simplify(A1-Uright[1,1]);
```

$$\begin{bmatrix} 1.0000000000\,10^{-101} & 1.0000000000\,10^{-101}\,I \\ -4.0000000000\,10^{-101}\,I & 0.0000000000 \end{bmatrix}$$

```
> B1 := (4*beta^2-1)/16*s3(Dinf).Matrix([[1,I],[I,-1]]).s3
  (Dinf^(-1) ):
> simplify(B1-Uleft[1,1]);
```

$$\begin{bmatrix} -1.0000000000\,10^{-101} & -4.0000000000\,10^{-101}\,I \\ 2.0000000000\,10^{-101}\,I & 3.0000000000\,10^{-101} \end{bmatrix}$$

```
> Ah2 := (al,be,c0) -> 8*al+8*be+8*c0-4*be^2+1:
> Bh2 := (al,be,c0) -> -8*al-8*be-8*c0+4*al^2+4*be^2-10:
> Ch2 := (al,be,c0) -> -8*al-8*be-8*c0-4*al^2-4*be^2+10:
> Dh2 := (al,be,c0) -> -8*al-8*be-8*c0-4*be^2+1:
> A2 := (4*alpha^2-1)/256*s3(Dinf).Matrix([[Ah2(alpha,beta,c
  (0) ),I*Bh2(alpha,beta,c(0) )],[I*Ch2(alpha,beta,c(0) ),Dh2
  (alpha,beta,c(0) )]]).s3(Dinf^(-1)):
> simplify(A2-Uright[2,1]);
```

$$\begin{bmatrix} 1.0000000000\,10^{-101} & -1.0000000000\,10^{-101}\,I \\ -1.0000000000\,10^{-101}\,I & -1.0000000000\,10^{-101} \end{bmatrix}$$

```
> B2 := (4*beta^2-1)/256*s3(Dinf).Matrix([[-Ah2(beta,alpha,-d
  (0) ),I*Bh2(beta,alpha,-d(0) )],[I*Ch2(beta,alpha,-d(0) ),-
  Dh2(beta,alpha,-d(0) )]]).s3(Dinf^(-1)):
> simplify(B2-Uleft[2,1]);
```

$$\begin{bmatrix} 0.0000000000 & -1.0000000000\,10^{-101}\,I \\ 4.0000000000\,10^{-101}\,I & 1.0000000000\,10^{-101} \end{bmatrix}$$

```
> Ah3 := (al,be,c0,d0) -> 16*(4*be^2-1)*(c0+d0+2*al+2*be) -2*
  (4*be^2-1)*(2*al^2+2*be^2-1)-128*(al+be)^2 -128*c0*(c0+2*
  al+2*be):
> q3p := (al,be,c0,d0) -> 128*(al+be)^2+128*c0*(c0+2*al+2*be)
  -388/3*al^2-84*be^2+64/3*al^4+16*be^4+48*al^2*be^2+176:
> r3p := (al,be,c0,d0) -> -128*(al+be)*(al^2+be^2)+320*(al+be)
  -64*be^2*(c0+d0)-128*c0*al^2+304*c0+16*d0:
> Dh3 := (a,b,f,g) -> 16*(4*b^2-1)*(f+g+2*a+2*b)+2*(4*b^2-1)*
  (2*a^2+2*b^2-1)+128*((a+b)^2+f*(f+2*a+2*b)):
> A3 := (4*alpha^2-1)/8192*s3(Dinf).Matrix([[Ah3(alpha,beta,c
  (0),-d(0) ),I*q3p(alpha,beta,c(0),-d(0) )+I*r3p(alpha,beta,c
  (0),-d(0) )],[I*q3p(alpha,beta,c(0),-d(0) ) -I*r3p(alpha,
  beta,c(0),-d(0) ),Dh3(alpha,beta,c(0),-d(0) )]]).s3(Dinf^
  (-1) ):
> simplify(A3-Uright[3,1]);
```

$$\begin{bmatrix} -6.0000000000\,10^{-102} & -5.0000000000\,10^{-102}\,I \\ 0.0000000000\,I & 2.0000000000\,10^{-102} \end{bmatrix}$$

```
> B3 := (4*beta^2-1)/8192*s3(Dinf).Matrix([[-Ah3(beta,alpha,-d
  (0),c(0) ),I*q3p(beta,alpha,-d(0),c(0) )+I*r3p(beta,alpha,-d
  (0),c(0) )],[I*q3p(beta,alpha,-d(0),c(0) ) -I*r3p(beta,
  alpha,-d(0),c(0) ),-Dh3(beta,alpha,-d(0),c(0) )]]).s3(Dinf^
  (-1) ):
> simplify(B3-Uleft[3,1]);
```

$$\begin{bmatrix} -7.0000000000\,10^{-102} & -1.0000000000\,10^{-101}\,I \\ -0.0000000000\,I & 1.0000000000\,10^{-102} \end{bmatrix}$$

```
> C3 := ((4*alpha^2-1)*(4*alpha^2-9)*(4*alpha^2-25)/12288)*s3
  (Dinf).Matrix([[-1,I],[I,1]]).s3(Dinf^(-1) ):
> simplify(C3-Uright[3,2]);
```

$$\begin{bmatrix} 0.0000000000 & 8.0000000000\,10^{-102}\,I \\ 0.0000000000 & 0.0000000000 \end{bmatrix}$$

```
> D3 := ((4*beta^2-1)*(4*beta^2-9)*(4*beta^2-25)/12288)*s3
  (Dinf).Matrix([[-1,-I],[-I,1]]).s3(Dinf^(-1) ):
> simplify(D3-Uleft[3,2]);
```

$$\begin{bmatrix} 1.0000000000\,10^{-101} & 1.4000000000\,10^{-101}\,I \\ 0.0000000000 & 0.0000000000 \end{bmatrix}$$

```
> v4 := (a,b,f0,g0,f1) -> (1-4*b^2)/6*(384*(f0^2+g0^2-f0*g0
  +3*(a+b)*(f0-g0))  +16*((a^2+b^2)^2+a^2*b^2)+1196*(a+b)^2
  -88*a*b-219):
> w4 := (a,b,f0,g0,f1) -> -4*((4*b^2-1)*(8*b^2+4*a^2 -11)*g0 +
  48*(4*a^2-9)*f1 -768*a*b*f0 -f0*(128*f0*(f0+3*(a+b)) +16*
  b^2*(b^2+2*a^2+25) + 312*a^2+139) -2*(a+b)*(b^2*(24*b^2+24*
  a^2+46)+58*a^2+128*a*b+3) ):
> x4 := (a,b,f0,g0,f1) -> 4*((4*b^2-1)*(8*b^2+12*a^2-29)*
  g0+48*(4*a^2-9)*f1 -768*a*b*f0 -f0*(128*f0*(f0+3*(a+b))+16*
  b^2*(b^2+6*a^2+16)+8*a^2*(8*a^2-7)+643)-4*(a+b)*(b^2*(12*
  b^2+36*a^2-31)+a^2*(16*a^2-65)+64*a*b+132)):
> y4 := (a,b,f0,g0,f1) -> 4/3*(48*(4*b^2-1)*g0*(g0-f0-3*(a+b))
  +b^4*(48*a^2+542)+48*f0^2*(4*b^2+12*a^2-28) + 144*(a+b)*(4*
  b^2+8*a^2-19)*f0+a^4*(56*b^2+422)+a*b*(1152*(a^2+b^2)+988*a*
  b-2880)+16*a^6-1393*b^2-951*a^2-498+8*b^6):
> A4 := ((4*alpha^2-1)/65536)*s3(Dinf).Matrix([[v4(alpha,beta,
  c(0),d(0),c(1) )+w4(alpha,beta,c(0),d(0),c(1) )  ,  I*x4
  (alpha,beta,c(0),d(0),c(1) )+I*y4(alpha,beta,c(0),d(0),c(1)
  ) ] , [ I*x4(alpha,beta,c(0),d(0),c(1) )-I*y4(alpha,beta,c
  (0),d(0),c(1) )  ,  v4(alpha,beta,c(0),d(0),c(1) )-w4(alpha,
  beta,c(0),d(0),c(1) ) ]]).s3(Dinf^(-1) ):
> B4 := ((4*beta^2-1)/65536)*s3(Dinf).Matrix([[-v4(beta,alpha,
  -d(0),-c(0),d(1) )-w4(beta,alpha,-d(0),-c(0),d(1) )  ,  I*x4
  (beta,alpha,-d(0),-c(0),d(1) )+I*y4(beta,alpha,-d(0),-c(0),d
  (1) ) ]  ,  [ I*x4(beta,alpha,-d(0),-c(0),d(1) )-I*y4(beta,
  alpha,-d(0),-c(0),d(1) )  ,  -v4(beta,alpha,-d(0),-c(0),d(1)
  )+w4(beta,alpha,-d(0),-c(0),d(1) )]]).s3(Dinf^(-1) ):
> simplify(B4-Uleft[4,1]);
```

$$\begin{bmatrix} 1.0000000000\,10^{-102} & -0.0000000000\,I \\ -0.0000000000\,I & 0.0000000000 \end{bmatrix}$$

```
> Eh4 := -4*alpha^2-8*beta^2+48*c(0)+48*alpha+48*beta+3:
> Fh4 := +8*alpha^2+8*beta^2-48*c(0)-48*alpha-48*beta-52:
> Gh4 := -8*alpha^2-8*beta^2-48*c(0)-48*alpha-48*beta+52:
> Hh4 := -4*alpha^2-8*beta^2-48*c(0)-48*alpha-48*beta+3:
> C4 := ((4*alpha^2-1)*(4*alpha^2-9)*(4*alpha^2-25)/(2^17*3) )
  *s3(Dinf).Matrix([[Eh4,I*Fh4],[I*Gh4,Hh4]]).s3(Dinf^(-1) ):
> simplify(C4-Uright[4,2]);
```

$$\begin{bmatrix} -1.0000000000\,10^{-101} & 0.0000000000\,I \\ -2.0000000000\,10^{-101}\,I & 0.0000000000 \end{bmatrix}$$

```
> Ih4 := -8*alpha^2-4*beta^2-48*d(0)+48*alpha+48*beta+3:
> Jh4 := -8*alpha^2-8*beta^2-48*d(0)+48*alpha+48*beta+52:
> Kh4 := +8*alpha^2+8*beta^2-48*d(0)+48*alpha+48*beta-52:
> Lh4 := -8*alpha^2-4*beta^2+48*d(0)-48*alpha-48*beta+3:
> D4 := ((4*beta^2-1)*(4*beta^2-9)*(4*beta^2-25)/(2^17*3) )*s3
  (Dinf).Matrix([[Ih4,I*Jh4],[I*Kh4,Lh4]]).s3(Dinf^(-1) ):
> simplify(D4-Uleft[4,2]);
```

$$\begin{bmatrix} 0.0000000000 & 0.0000000000\,I \\ -3.0000000000\,10^{-101}\,I & 0.0000000000 \end{bmatrix}$$

2 Asymptotic expansions

```
> n := 'n':
> Gam[0] := 1:
  betaln[0] := 1/(2*I*Dinf^2):
  betarn[0] := -Dinf^2/(2*I):
  Alph[0] := 0:
> for k from 1 to (maxOrder-1) do
     Gam[k] := 2*I*Dinf^2*(Uright[k,1](2,1) + Uleft[k,1](2,1) )/
  (n+1)^k; # Terms in asymptotic expansion of \gamma_n^2
     betaln[k] := (Uright[k,1](2,1) + Uleft[k,1](2,1) )/n^k; #
  Terms in left factor of asymptotic expansion of \beta_n
     betarn[k] := (Uright[k,1](1,2) + Uleft[k,1](1,2) )/n^k; #
  Terms in right factor of asymptotic expansion of \beta_n
     Alph[k] := (Uright[k,1](1,1) + Uleft[k,1](1,1) )/(n+1)^k +
  (Uright[k,1](2,2) + Uleft[k,1](2,2) )/n^k; # Terms in
```

```
asymptotic expansion of \alpha_n
od:
k := 'k':
```

ipi = interior monic polynomial, opi = outer monic polynomial (two equivalent formulations but the one without phi(z) is expected to better avoid numerical roundoff),
gamman = leading order coeff of orthonormal, betan & alphan = recurrence coefficients

```
> for i from 1 to maxOrder do
    ipi[i] := unapply( (2^(1/2-n)/(sqrt(w(z) )*(1-z)^(1/4)*(1+z)
^(1/4) )*Matrix([[1, 0]]).sum(R[k]/n^k, k = 0..(i-1) ).Matrix([
[Dinf*cos((n+1/2)*arccos(z) +psi(z) -Pi/4) ], [-I/Dinf*cos(
(n-1/2)*arccos(z) +psi(z) -Pi/4)]]) )(1),n,z):

    opi[i] := unapply( (2^(-1/2-n)/(sqrt(w(z) )*(1-z)^(1/4)*(1+
z)^(1/4) )*Matrix([[1, 0]]).sum(R[k]/n^k, k = 0..(i-1) ).Matrix
([[Dinf*exp(signum(0,argument(z-1),-1)*I*( (n+1/2)*arccos(z) +
psi(z) -Pi/4) ) ], [-I/Dinf*exp(signum(0,argument(z-1),-1)*I*(
(n-1/2)*arccos(z) +psi(z) -Pi/4) )]]) )(1),n,z):

   #opi[i] := unapply( ( (phi(z) )^n*exp(signum(0,argument
(z-1),-1)*I*psi(z) )/(2^n*sqrt(2*w(z) )*(z-1)^(1/4)*(z+1)^(1/4)
)*Matrix([[1, 0]]).sum(R[k]/n^k, k = 0..(i-1) ).Matrix([[Dinf*
sqrt(phi(z) )], [-I/(Dinf*sqrt(phi(z) ) )]]) )(1),n,z):

   gamman[i] := unapply(Re(sqrt(2^(2*n)/Pi/Dinf^2*sum(Gam[k], k
= 0..(i-1) ) ) ), n):
   betan[i] := unapply(Re(sum(betaln[k], k = 0..(i-1) )*sum
(betarn[k], k = 0..(i-1) ) ), n);
   alphan[i] := unapply(Re(-sum(Alph[k], k = 0..(i-1) ) ), n);
od:
```

The maximum order to which to compute and check the expansions in the boundary regions, <= maxOrder due to higher complexity in number of terms, take it lower to run the sheet faster.

```
> mob := min(maxOrder,7):
```

bpiR = right boundary monic, bpiL = left boundary monic
When evaluating only on the interval, one can take Re(ipi) & Re(bpR) & Re(bpL) to get rid of small imaginary parts.

```
> for i from 1 to mob do
   bpiR[i] := unapply( (sqrt(Pi*n*arccos(z))/(2^n*sqrt(w(z) )*
(1-z)^(1/4)*(1+z)^(1/4) )*Matrix([[1, 0]]).add(Rright[k]/n^k, k
= 0..(i-1) ).Matrix([[Dinf*(cos(psi(z) +alpha*Pi/2 + arccos(z)
/2)*BesselJ(alpha,n*arccos(z) ) + sin(psi(z) +alpha*Pi/2 +
arccos(z)/2)*(BesselJ(alpha-1,n*arccos(z) ) - BesselJ(alpha+1,
n*arccos(z) ) )/2 )], [-I/Dinf*(cos(psi(z) +alpha*Pi/2 -arccos
```

```
(z)/2)*BesselJ(alpha,n*arccos(z) ) + sin(psi(z) +alpha*Pi/2 -
arccos(z)/2)*(BesselJ(alpha-1,n*arccos(z) ) - BesselJ(alpha+1,
n*arccos(z) ) )/2 ) ]]) )(1),n,z):
  bpiL[i] := unapply( (sqrt(Pi*n*arccos(-z))/( (-2)^n*sqrt(w
(z) )*(1-z)^(1/4)*(1+z)^(1/4) )*Matrix([[1, 0]]).add(Rleft[k]
/n^k, k = 0..(i-1) ).Matrix([[Dinf*(sin(psi(z) -beta*Pi/2 +
arccos(z)/2)*BesselJ(beta,n*arccos(-z) ) + cos(psi(z) -beta*
Pi/2 + arccos(z)/2)*(BesselJ(beta-1,n*arccos(-z) ) - BesselJ
(beta+1,n*arccos(-z) ) )/2 )], [-I/Dinf*(sin(psi(z) -beta*Pi/2
-arccos(z)/2)*BesselJ(beta,n*arccos(-z) ) + cos(psi(z) -beta*
Pi/2 -arccos(z)/2)*(BesselJ(beta-1,n*arccos(-z) ) - BesselJ
(beta+1,n*arccos(-z) ) )/2 ) ]]) )(1),n,z):
od:
```

6 Numerical results

Initialisation

```
> with(plots): with(plottools):
```

```
> hIs1 := convert(evalf(int(abs(h(x) - 1), x = -1.0..1.0)) =
  0, truefalse):
```

The 2-logaritm of the maximum degree n to be used for the plots, has to be very low when we have to compute the exact polynomials to compare with ourselves when h is not identically one due to time requirements..

```
> if hIs1 then
     maxP2 := 7:
  else
     Digits := 20:
     maxP2 := 4:
  end if:
```

```
> if(hIs1) then
     p := (n::integer,x) -> orthopoly[P](n,alpha,beta,x)*sqrt(
  (2*n+alpha+beta+1)*factorial(n)*GAMMA(n+alpha+beta+1)/(2^
  (alpha+beta+1)*GAMMA(n+alpha+1)*GAMMA(n+beta+1) ) ):
     gammaP:= proc(k::integer) if k = 0 then return p(k,x);
  else return coeff(expand(p(k,x) ),x^k); end if; end proc:
     pi:= (n::integer,x) -> expand(p(n,x)/gammaP(n) ):
  else
     p[0] := unapply(evalf(1/sqrt(int(w(t), t=-1.0..1.0,
  numeric) ) ),x):
     v[1] := unapply(x*p[0](x) ,x):
     betaR[1] := evalf(int(w(x)*v[1](x)*p[0](x), x=-1.0..1.0,
```

```
numeric) ):
    vv[1] := unapply(v[1](x) - betaR[1]*p[0](x),x):
    gammaR[1] := evalf(sqrt(int(w(x)*vv[1](x)^2, x=-1.0..1.0,
numeric) ) ):
    p[1] := unapply(vv[1](x)/gammaR[1],x):
    print("m = 1");
    for m from 2 to 2^maxP2 do
      v[m] := unapply(x*p[m-1](x)-gammaR[m-1]*p[m-2](x),x):
      betaR[m] := evalf(int(w(x)*v[m](x)*p[m-1](x), x=-1.0.
.1.0,numeric) ):
      vv[m] := unapply(v[m](x) - betaR[m]*p[m-1](x),x):
      gammaR[m] := evalf(sqrt(int(w(x)*vv[m](x)^2, x=-1.0.
.1.0,numeric) ) ):
      p[m] := unapply(vv[m](x)/gammaR[m],x):
      print("m = ", m);
    od:
    gammaP:= proc(k::integer) if k = 0 then return p[0](x);
else return coeff(expand(p[k](x) ),x^k); end if; end proc:
    pi := (n::integer,x) -> expand(p[n](x)/gammaP(n) ):
    print("Done");
end if:
```

Sanity check:

```
> evalf(pi(14,0.4));
```

$$-0.0000339420$$

```
> evalf(ipi[4](14,0.4) );
```

$$-0.0000339420 + 0.0000000000\,I$$

Solve the three term recurrence relation in the least squares sense for different x to get the recurrence coefficients to compare with.
LinearSolve with nbr := 2 should also work but do Least squares to cancel roundoff errors.

```
> nbr := 4:
> xs := [seq(1.9*xi/nbr-1, xi=1..nbr)]:
> alphaP := proc(n::integer) return (LeastSquares(Matrix(nbr,
  2, (i,j)-> pi(n+1-j, xs[i]) ), Vector(nbr, i -> xs[i]*pi(n,
  xs[i]) -pi(n+1,xs[i]) ) ) )(1): end proc:
> betaP := proc(n::integer) return (LeastSquares(Matrix(nbr,2,
  (i,j)-> pi(n+1-j, xs[i]) ), Vector(nbr, i -> xs[i]*pi(n, xs
  [i]) -pi(n+1,xs[i]) ) ) )(2): end proc:

> weights := [seq(1+tn/maxP2, tn=0..maxP2)]:
> colors := ["Black","Green", "Blue","Orange","Cyan", "Red",
  "Gray", "Purple", "Brown", "Pink", "Gold", "Violet",
  "Khaki", seq("Turquoise", k=14..maxOrder)]:
> for i from 15 to maxOrder do
```

```
      colors[i] := colors[i-14]:
   od:
```

Interior region

Convergence is guaranteed when choosing a zInt that could be considered to be in the lens, but the asymptotic expansion for the interior region also gives good approximations and convergence elsewhere.

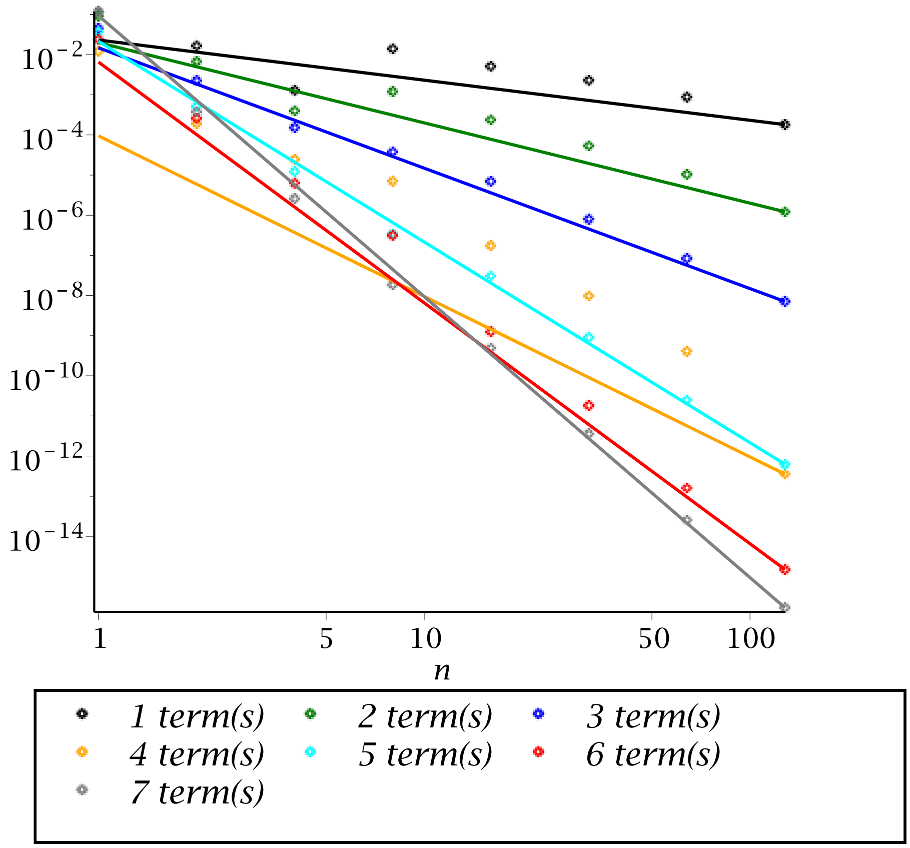
```
> zInt := 0.2:
> n := 'n': k := 'k':

> for tn from 0 to maxP2 do
      piEvi[tn] := evalf(pi(2^tn, zInt) ):
   od:

> for i from 1 to maxOrder do
      datai[i] := [seq([2^tn,evalf(abs(piEvi[tn] -ipi[i](2^tn, zInt) )/abs(piEvi[tn] ) )], tn=0..maxP2)]:
      ploti[i] := loglogplot(datai[i], style = point, color = colors[i], legend = cat(i, " term(s)") ):
      plotSi[i] := loglogplot(datai[i][maxP2+1][2]*(n/2^maxP2)^(-i), n = 1..2^maxP2, color = colors[i]):
      ldi[i] := [seq(MTM[log2]~(datai[i][tn+1]), tn=0..maxP2 )]:
      tn := 'tn':
      fcti[i] := subs(tn = MTM[log2](n),CurveFitting[LeastSquares](ldi[i],tn,weight=weights) ):
      plotLSi[i] := loglogplot(2^fcti[i],n= 1..2^maxP2, color = colors[i]):
      slopei[i] := simplify((fcti[i] - subs(n = 1,fcti[i]))/MTM[log2](n) );
   od:
> eval(slopei);
```

$table([1 = -1.0197656332, 2 = -2.0378855362, 3 = -3.0476196527, 4 = -4.6331174136, 5 = -4.9403615324, 6 = -6.2828403623, 7 = -6.8559238505])$

```
> display(seq(ploti[m], m=1..maxOrder), seq(plotSi[m], m=1..maxOrder) );
```

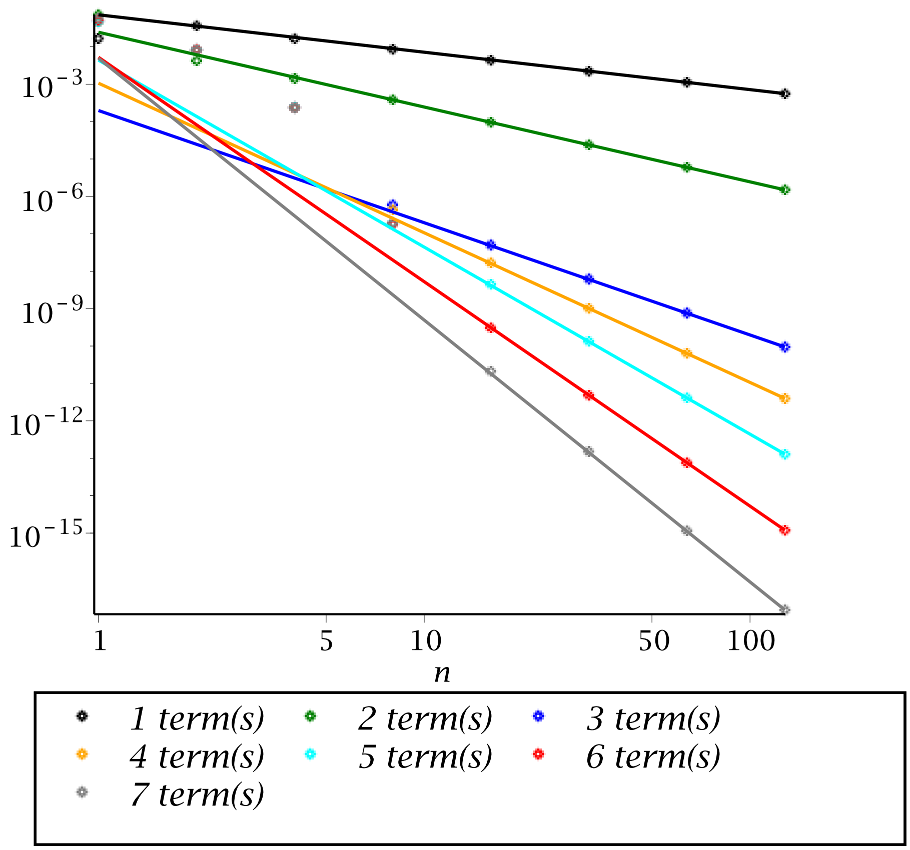


Outer region

Compare this with interior region with zOut close to interval -> this one is much less accurate and sometimes we don't even see right order of convergence (which we would see if n would be larger)

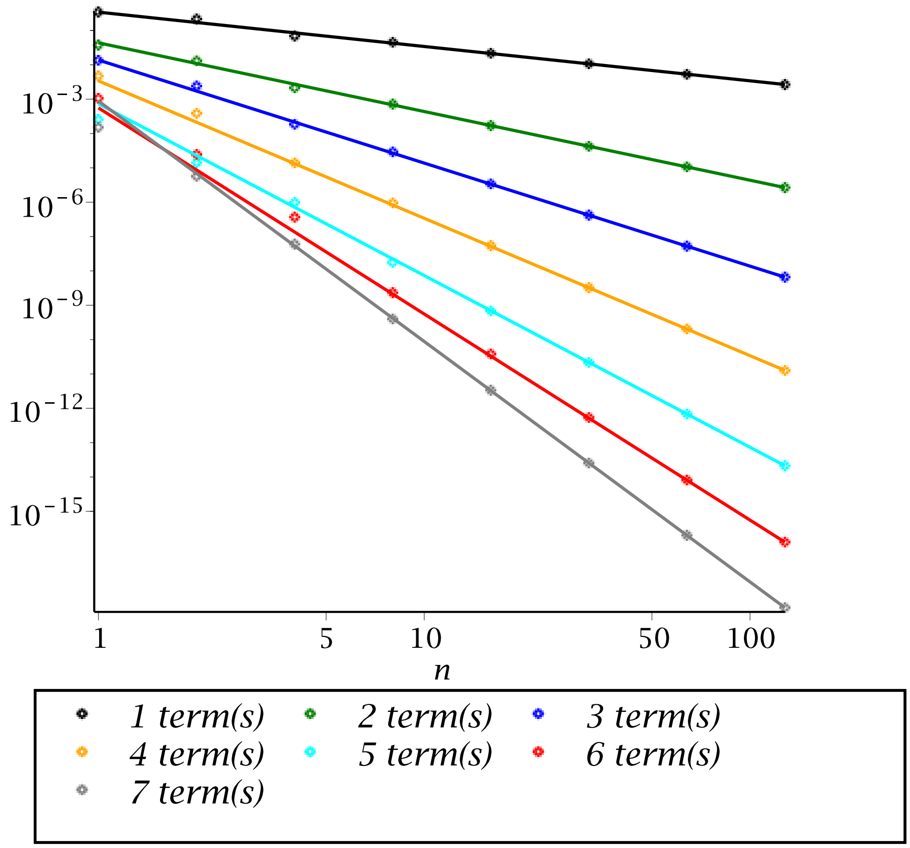
```
> zOut := 0.2 -1*I:
> for tn from 0 to maxP2 do
    piEvo[tn] := evalf(pi(2^tn, zOut) ):
  od:
> for i from 1 to maxOrder do
    datao[i] := [seq([2^tn,evalf(abs(piEvo[tn] -opi[i](2^tn, zOut) )/abs(piEvo[tn] ) )], tn=0..maxP2)]:
    ploto[i] := loglogplot(datao[i], style = point, color = colors[i], legend = cat(i, " term(s)") ):
   plotSo[i] := loglogplot(datao[i][maxP2+1][2]*(n/2^maxP2)^(-i), n = 1..2^maxP2, color = colors[i]):
  od:
```

```
> display(seq(ploto[m], m=1..maxOrder), seq(plotSo[m], m=1..
  maxOrder) );
```



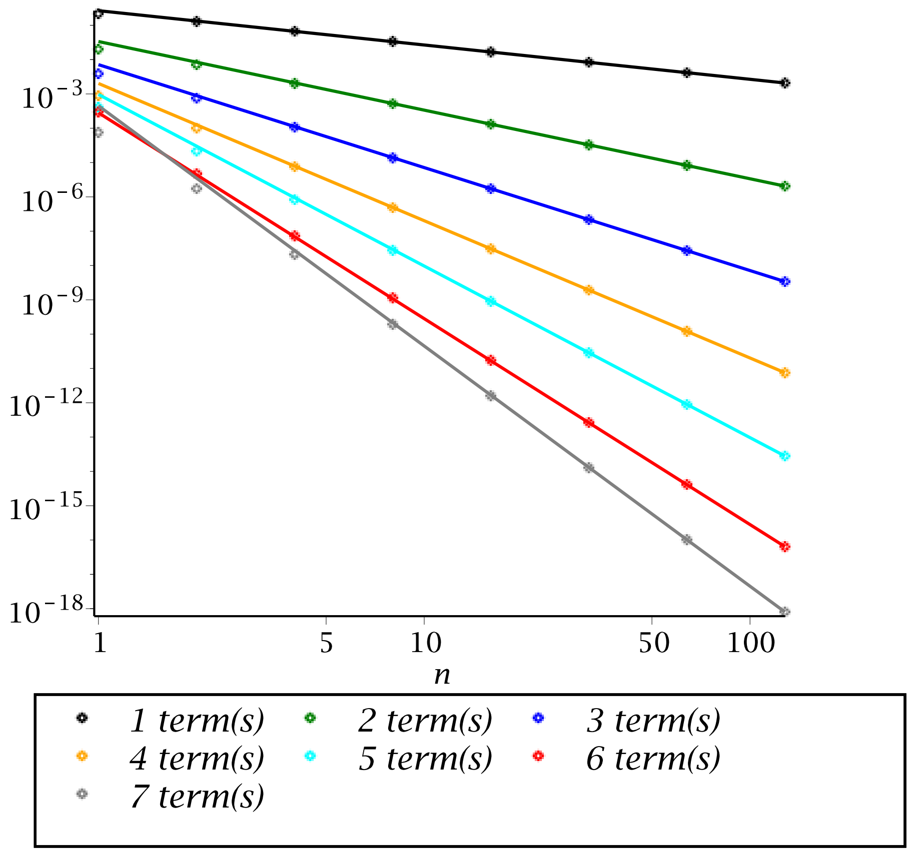
Right Boundary region

```
> zR := 0.9-0.05*I:
> for tn from 0 to maxP2 do
    piEvbR[tn] := evalf(pi(2^tn, zR) ):
  od:
> for i from 1 to mob do
    dataR[i] := [seq([2^tn,evalf(abs(piEvbR[tn] -bpiR[i]
  (2^tn,zR) )/abs(piEvbR[tn] ) )], tn=0..maxP2)]:
    plotR[i] := loglogplot(dataR[i], style = point, color =
  colors[i], legend = cat(i, " term(s)") ):
    plotSR[i] := loglogplot(dataR[i][maxP2+1][2]*(n/2^maxP2)^
  (-i), n = 1..2^maxP2, color = colors[i]):
  od:
> display(seq(plotR[m], m=1..mob), seq(plotSR[m], m=1..mob) );
```



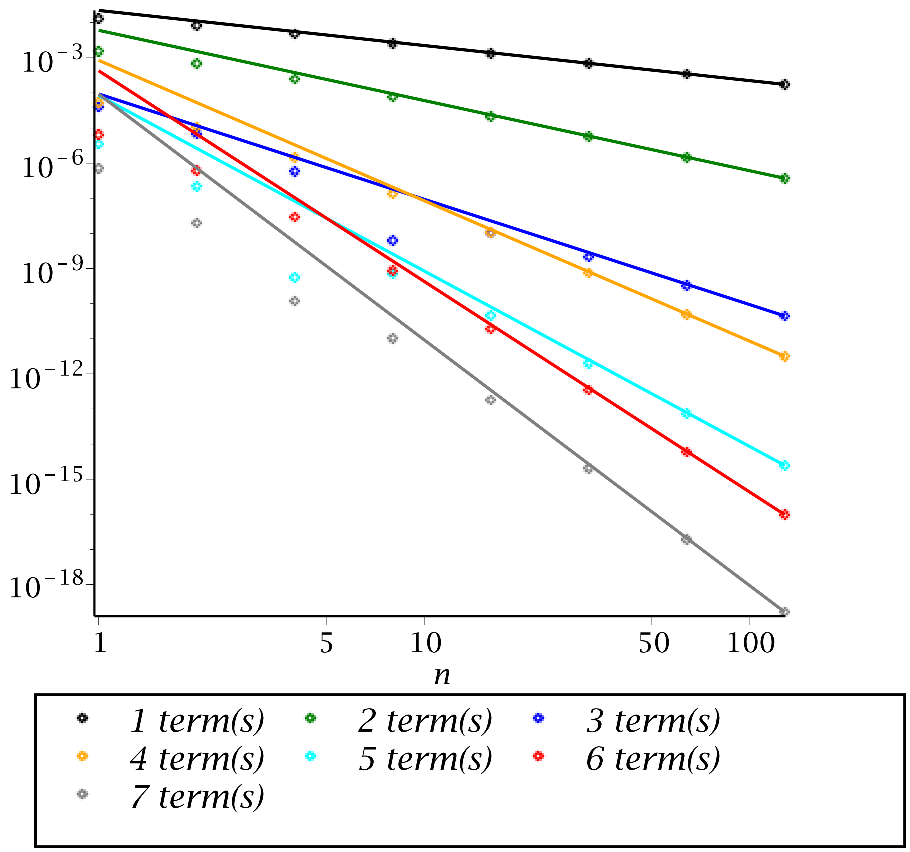
Left Boundary region

```
> zL := -1.1+0.2*I:
> for tn from 0 to maxP2 do
     piEvbL[tn] := evalf(pi(2^tn, zL) ):
  od:
> for i from 1 to mob do
     dataL[i] := [seq([2^tn,evalf(abs(piEvbL[tn] -bpiL[i]
  (2^tn,zL) )/abs(piEvbL[tn] ) )], tn=0..maxP2)]:
     plotL[i] := loglogplot(dataL[i], style = point, color =
  colors[i], legend = cat(i, " term(s)") ):
    plotSL[i] := loglogplot(dataL[i][maxP2+1][2]*(n/2^maxP2)^
  (-i), n = 1..2^maxP2, color = colors[i]):
  od:
> display(seq(plotL[m], m=1..mob), seq(plotSL[m], m=1..mob) );
```



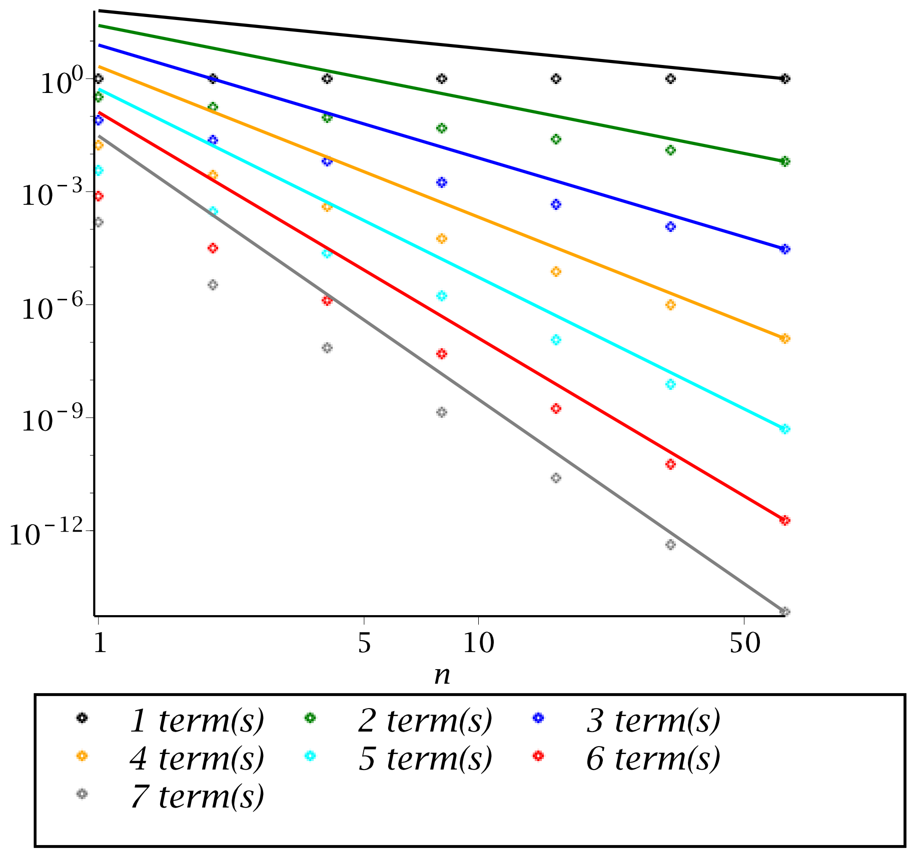
Leading order coefficients

```
> for tn from 0 to maxP2 do
    gammaReal[tn] := evalf(gammaP(2^tn) ):
  od:
> for i from 1 to maxOrder do
    dataG[i] := [seq([2^tn,evalf(abs(gammaReal[tn] -gamman[i]
  (2^tn) )/abs(gammaReal[tn] ) )], tn=0..maxP2)]:
    plotG[i] := loglogplot(dataG[i], style = point, color =
  colors[i], legend = cat(i, " term(s)") ):
    plotSG[i] := loglogplot(dataG[i][maxP2+1][2]*(n/2^maxP2)^
  (-i), n = 1..2^maxP2, color = colors[i]):
  od:
> display(seq(plotG[m], m=1..maxOrder), seq(plotSG[m], m=1..
  maxOrder) );
```



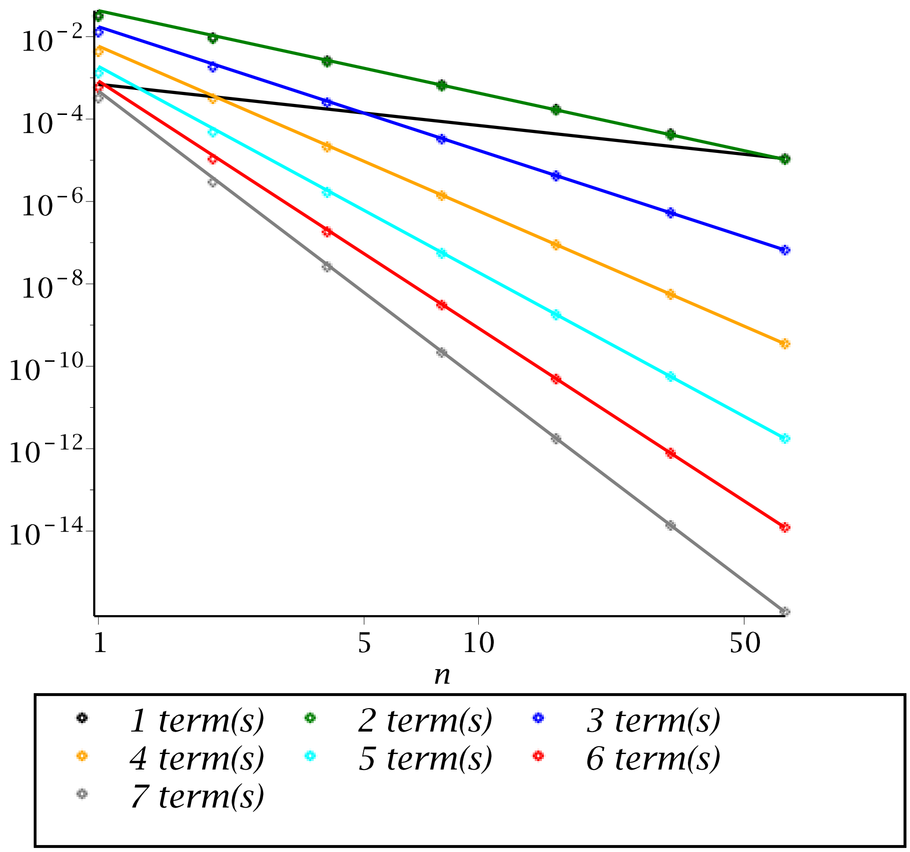
Recurrence coefficients alpha

```
> for tn from 0 to maxP2-1 do
    alphaReal[tn] := evalf(alphaP(2^tn) ):
  od:
> for i from 1 to maxOrder do
    dataa[i] := [seq([2^tn,evalf(abs(alphaReal[tn] -alphan[i]
  (2^tn) )/abs(alphaReal[tn] ) )], tn=0..maxP2-1)]:
    plota[i] := loglogplot(dataa[i], style = point, color =
  colors[i], legend = cat(i, " term(s)") ):
    plotSa[i] := loglogplot(dataa[i][maxP2][2]*(n/2^(maxP2-1)
  )^(-i), n = 1..2^(maxP2-1), color = colors[i]):
  od:
> display(seq(plota[m], m=1..maxOrder), seq(plotSa[m], m=1..
  maxOrder) );
```



Recurrence coefficients beta

```
> for tn from 0 to maxP2-1 do
    betaReal[tn] := evalf(betaP(2^tn) ):
  od:
> for i from 1 to maxOrder do
    datab[i] := [seq([2^tn,evalf(abs(betaReal[tn] -betan[i]
  (2^tn) )/abs(betaReal[tn] ) )], tn=0..maxP2-1)]:
    plotb[i] := loglogplot(datab[i], style = point, color =
  colors[i], legend = cat(i, " term(s)") ):
    plotSb[i] := loglogplot(datab[i][maxP2][2]*(n/2^(maxP2-1)
  )^(-i), n = 1..2^(maxP2-1), color = colors[i]):
  od:
> display(seq(plotb[m], m=1..maxOrder), seq(plotSb[m], m=1..
  maxOrder) );
```


10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
1
5
10
50
n
1 term(s)
2 term(s)
3 term(s)
4 term(s)
5 term(s)
6 term(s)
7 term(s)