ON THE SIGN CHANGES OF $\psi(x) - x$

Heuristic computations accompanying the paper

M. Grześkowiak, J. Kaczorowski, Ł. Pańkowski, M. Radziejewski

This notebook documents how we searched for contour parameters to be used in the rigorous proof. Wherever possible, we use the notation from the paper, section “The proof of Theorem 1”. Some peculiarities of the code below result from how Mathematica is designed:

- Wherever we need a variable with a capitalized name, we precede it with lowercase “c”, for “capital”, because names starting with a capital letter are reserved for Mathematica symbols.
- Numerical constants (e.g., those used in defining the contour) are entered using Mathematica’s backtick notation, like 0.025`100. This tells Mathematica that it is ok to perform high-precision calculations on this number. An exact decimal that we meant, 0.025, would be interpreted as a machine-precision number, a notion which would propagate to all derived quantities and cause any high-precision settings to be disregarded. This is not always desirable.

This notebook is also available as a PDF file for viewing without Mathematica.

Initialization

7/13/24 07:57:37 In[1]:=

```
cN = 21; (* The main parameter, N in the paper. *)
epsilon = 3.32`100 × 10^-13; (* Obtained from Algorithm 1 for the given N. *)
precision = 100; (* The precision to be used in most of our calculations. *)
numZeros = 10 000; (* Zeros to be used in estimating R_N(y),
denoted N' in the paper. *)
$MaxExtraPrecision = 2 * precision;
Parallelize[imZero =
   Table[N[Im[ZetaZero[k]], {Infinity, precision + 10}], {k, 1, numZeros}]];
gamma[n_] := imZero[[n + 1]]; (* Conversion between our 0-
 based indexing and Mathematica's 1-based arrays. *)
rho[n_] := 1 / 2 + I gamma[n]; (* Conversion between our 0-
 based indexing and Mathematica's 1-based arrays. *)
omega[n_]  := gamma[n] - gamma[0];
cNPrime = numZeros - 2;
error = 10 ^ (5 - precision); (* For basic upward rounding. *)
```

```
aUpperBound[n_] := (Abs[rho[0]] + error) / (Abs[rho[n]] - error)
(* Upper bound for |a(n)|. *)
cT1 = (gamma[cNPrime] + gamma[cNPrime + 1]) / 2;
(* The upper bound for R_N(y). *)
cRNprec[y_] :=
  Sum[aUpperBound[n] Exp[y (error - omega[n])], {n, cN + 1, cNPrime}] +
   ((Abs[rho[0]] + error) Exp[y (gamma[0] - cT1 + error)] / (cT1 - error))
    (Log[cT1 + error] (4 + 1 / (2 Pi y)) + 2 / (y (cT1 - error)));
(* The upper bound for R_N is slightly expensive to compute,
so we are going to pre-compute it at 0.00001, 0.00002, ..., 0.15. *)
cRNprecTop = 0.15`100;
cRNprecRes = 15 000;
Parallelize[
  cRNprecArray = Table[cRNprec[cRNprecTop * j / cRNprecRes], {j, 1, cRNprecRes}]];
cRNest[y_] := Module[{t, j},
   t = cRNprecRes * y / cRNprecTop;
   j = Floor[t];
   t -= j;
   If[j ≥ cRNprecRes, cRNprecArray[[j]],
    cRNprecArray[[j]] (1 - t) + cRNprecArray[[j + 1]] t]
  ];
norm[x_] := Abs[x - Round[x]];
cFNEta[z_] :=
  1 - Sum[(Abs[rho[0]] / Abs[rho[n]]) Exp[omega[n] * z * I], {n, 1, cN}];
f[z_] :=
  -Sum[I * omega[n] * (Abs[rho[0]] / Abs[rho[n]]) Exp[omega[n] * z * I], {n, 1, cN}];

(* The function below gives a non-rigorous estimate of the final result,
meant for finding contour parameters using fewer contour points. *)
resultApproximate[(* contour position *) paramdelta_?NumericQ,
   paramcontourTop_?NumericQ, paramcontourBottom_?NumericQ,
   (* a function that goes from the value 0 at 0 to the value π at 4,
   that determines α_j`s in the paper *) argFun_,
   (* m/8, i.e. half of the number of divisions
    of the contour on each side *) contourRes_?NumericQ,
   (* l, i.e. the resolution for approximating weight functions *)
   weightRes_?IntegerQ
  ] :=
  Module[
   {
    delta, contourTop, contourBottom,
    m, t, z1, cRNArray, alpha,
    b, xPrime, xBis, y,
    indexForRN, M, u,
    betaPrime, betaBis, phiPrime, phiBis, c, symmetricPairs, phi, c0j, cn0,
    operation, wnInverse, wnMax, wnScale, localc, localphi,
    cosphi, iMax, cosValue, previous, product, volume, returnValue,
```

```
 epsmachine,
 i, j, n
},
delta = SetPrecision[paramdelta, precision];
contourTop = SetPrecision[paramcontourTop, precision];
contourBottom = SetPrecision[paramcontourBottom, precision];
m = 8 * contourRes;
t[j_] := j / m;
(*z1(t_j) is to be stored as z1[[j+1]].*)
z1 = Join[Table[(-0.5`100 + j / (2 * contourRes)) * delta + contourBottom * I,
    {j, 0, 2 * contourRes}], Table[0.5`100 * delta +
      (contourBottom + (contourTop - contourBottom) * j / (2 * contourRes)) * I,
    {j, 1, 2 * contourRes}], Table[(0.5`100 - j / (2 * contourRes)) * delta +
     contourTop * I, {j, 1, 2 * contourRes}], Table[-0.5`100 * delta +
      (contourTop - (contourTop - contourBottom) * j / (2 * contourRes)) * I,
    {j, 1, 2 * contourRes}]];
alpha = Join[Table[-Pi - argFun[1 - (j - 1) / contourRes], {j, 1, contourRes}],
   Table[-Pi + argFun[(j - 1) / contourRes], {j, 1, 4 * contourRes}],
   Table[Pi - argFun[4 - (j - 1) / contourRes], {j, 1, 3 * contourRes + 1}]];

(*Claim 1.*)
(*Subsequent alpha[[j]] should differ by less than π.*)
If[
 Max[Table[Abs[alpha[[j + 1]] - alpha[[j]]], {j, 1, m}]] > Pi - error, Return[-1.1]];
(*This value should be less than π.*)
If[Max[Table[Abs[z1[[j + 1]] - z1[[j]]], {j, 1, m}]] * omega[cN] > Pi - error,
 Return[-1.2]];

xPrime = Table[Re[z1[[j]]], {j, 1, m}];
y = Table[Im[z1[[j]]], {j, 1, m}];
cRNArray = Table[cRNest[y[[j]]], {j, 1, m}];
u = Table[Re[cFNEta[z1[[j]]] Exp[-I * alpha[[j]]]] - cRNArray[[j]], {j, 1, m}];

(*Claim 2.*)
(*This value should be positive.*)
If[Min[u] < error, Return[-2]];

betaPrime =
 Table[Table[Pi + omega[n] * xPrime[[j]] - alpha[[j]], {j, 1, m}], {n, 1, cN}];
phiPrime =
 Table[Table[2 * Pi * norm[betaPrime[[n]][[j]] / (2 * Pi)], {j, 1, m}], {n, 1, cN}];
c = Table[
   Table[(aUpperBound[n] / u[[j]]) * Exp[-omega[n] * y[[j]]], {j, 1, m}], {n, 1, cN}];
(*Because of the symmetry cFNEta[-x+y I]==
 cFNEta[x+y I] we can skip half of the intervals.This part does not
   appear in the paper as it is just a numerical optimization.We do
   not include it in the rigorous check.This allows us to shorten
```

```
  the iterative process of choosing best division points.*)
(*We leave out the repeated entries.*)
For[n = 1, n ≤ cN, n++,
 c⟦n⟧ = c⟦n⟧⟦contourRes + 1 ;; 5 * contourRes⟧;
 phiPrime⟦n⟧ = phiPrime⟦n⟧⟦contourRes + 1 ;; 5 * contourRes⟧;
];
u = u⟦contourRes + 1 ;; 5 * contourRes⟧;
y = y⟦contourRes + 1 ;; 5 * contourRes⟧;
phi = phiPrime;
wnMax = Table[Min[1, Max[Table[
     (c⟦n⟧⟦j⟧ + error) * (Cos[phi⟦n⟧⟦j⟧] + 1), {j, Length[c⟦n⟧]}]]], {n, cN}];
For[n = cN - 1, n ≥ 1, n--, wnMax⟦n⟧ += wnMax⟦n + 1⟧];
wnScale = Table[1, {n, cN}];
For[n = 2, n ≤ cN - 1, n++, wnScale⟦n⟧ =
  Max[1, Floor[(1 / 2 - error) / (wnScale⟦n - 1⟧ * wnMax⟦n⟧)]] * wnScale⟦n - 1⟧];
wnScale⟦cN⟧ = wnScale⟦cN - 1⟧;
For[n = cN, n ≥ 1, n--,
 operation = "Computing";
 epsmachine = 0.0 + epsilon;
 wnInverse = Table[0.5, {i, weightRes}];
 For[j = 1, j ≤ Length[c⟦n⟧], j++,
  localc = 0.0 + wnScale⟦n⟧ * (c⟦n⟧⟦j⟧ + error);
  (*To machine precision*)
  localphi = 0.0 + phi⟦n⟧⟦j⟧;
  cosphi = Cos[localphi];
  iMax = Floor[Min[1, localc * (cosphi + 1)] * weightRes];
  For[i = 1, i ≤ iMax, i++,
   (*Solve c(Cos[ϕ]-Cos[ϕ+2π(epsilon+x)])==i/weightRes*)
   cosValue = cosphi - i / (weightRes * localc);
   wnInverse⟦i⟧ =
    Min[wnInverse⟦i⟧, (ArcCos[cosValue] - localphi) / (2 Pi) - epsmachine]
  ];
 ];
 For[i = weightRes, i ≥ 1, i--,
  wnInverse⟦i⟧ = Floor[wnInverse⟦i⟧ * weightRes ^ 2]
 ];
 For[i = weightRes, i ≥ 2, i--,
  wnInverse⟦i⟧ -= wnInverse⟦i - 1⟧
 ];
 If[n == cN,
  previous = wnInverse
  ,
  operation = "Convoluting";
  If[wnScale⟦n⟧ < wnScale⟦n + 1⟧,
   For[i = 1, i ≤ Floor[weightRes * wnScale⟦n⟧ / wnScale⟦n + 1⟧], i++,
    previous⟦i⟧ = Sum[previous⟦j⟧, {j, 1 + (wnScale⟦n + 1⟧ / wnScale⟦n⟧) * (i - 1),
       (wnScale⟦n + 1⟧ / wnScale⟦n⟧) * i}]
```

```
      ];
      For[i = 1 + Floor[weightRes * wnScale[[n]] / wnScale[[n + 1]]], i ≤ weightRes, i ++,
       previous[[i]] = 0
      ]
     ];
     product =
      Join[{0}, ListConvolve[previous, wnInverse, 1, 0][[1 ;; weightRes - 1]]];
     previous = product
    ]
   ];
   volume = 2 ^ cN * Sum[product[[i]], {i, weightRes}] / weightRes ^ (2 cN);
   returnValue = 2 * volume / delta;
   returnValue
  ];
listplot[t_, label_ : ""] := ListPlot[t, ImageSize → Large,
   GridLines → Automatic, PlotRange → Full, PlotLabel → label];

(*Simple 1D optimization,
suitable for functions that take minutes to evaluate at a single point.*)
findMax[function_, x1_, x2_, iter_] :=
  Module[{args, vals, i, j, go},
   go[num_] := Module[{},
     vals[[num]] = function[args[[num]]]
    ];
   args = {0, 0, 0, 0, 0};
   vals = {0, 0, 0, 0, 0};
   i = 0;
   For[j = 1, j ≤ 5, j ++,
    args[[j]] = x1 + (j - 1) * (x2 - x1) / 4
   ];
   For[j = 1, j ≤ 5, j ++,
    go[j]
   ];
   For[i = 0, i < iter, ,
    j = Ordering[vals, -1][[1]];
    If [j ≥ 2 && j ≤ 4,
     i += 1;
     args = {args[[j - 1]], (args[[j - 1]] + args[[j]]) / 2,
       args[[j]], (args[[j]] + args[[j + 1]]) / 2, args[[j + 1]]};
     vals = {vals[[j - 1]], 0, vals[[j]], 0, vals[[j + 1]]};
     If[vals[[1]] ≥ vals[[5]],
      go[2];
      If[vals[[2]] ≤ vals[[3]], go[4]]
      ,
      go[4];
      If[vals[[4]] ≤ vals[[3]], go[2]]
     ]
```

```
      ,
      i -= 1;
      If[j == 1,
       args =
        {2 * args[[1]] - args[[5]], 2 * args[[1]] - args[[3]], args[[1]], args[[3]], args[[5]]};
       vals = {0, 0, vals[[1]], vals[[3]], vals[[5]]};
       go[2];
       If[vals[[2]] ≥ vals[[3]], go[1]]
       ,
       args =
        {args[[1]], args[[3]], args[[5]], 2 * args[[5]] - args[[3]], 2 * args[[5]] - args[[1]]};
       vals = {vals[[1]], vals[[3]], vals[[5]], 0, 0};
       go[4];
       If[vals[[4]] ≥ vals[[3]], go[5]]
      ]
     ];
    ];
    j = Ordering[vals, -1][[1]];
    {args[[j]], vals[[j]]}
   ];
```

Finding a good set of parameters

For a given delta we try to find a satisfactory argument function of a very simple form, then find a good value of contour bottom (Y_0), then the argument function and Y_0 again.
We also try an alternative approach, namely to do the same in reverse order.

7/13/24 08:05:17 In[25]:=

```
parameters1[delta_] := Module[
   {aFixed = 1.8, y0Fixed = 0.0468, cRes = 1000, wRes = 4000},
   aFixed =
    findMax[Function[a, resultApproximate[delta, 0.14, y0Fixed, Interpolation[
         {{0, 0}, {1, a}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
         InterpolationOrder → 1], cRes, wRes]],
      aFixed - 0.2, aFixed + 0.2, 10][[1]];
   y0Fixed =
    findMax[Function[y0, resultApproximate[delta, 0.14, y0, Interpolation[
         {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
         InterpolationOrder → 1], cRes, wRes]],
      y0Fixed - 0.00016, y0Fixed + 0.00016, 8][[1]];
   aFixed =
    findMax[Function[a, resultApproximate[delta, 0.14, y0Fixed, Interpolation[
         {{0, 0}, {1, a}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
         InterpolationOrder → 1], cRes, wRes]],
      aFixed - 0.2, aFixed + 0.2, 10][[1]];
   y0Fixed =
    findMax[Function[y0, resultApproximate[delta, 0.14, y0, Interpolation[
```

```
          {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
          InterpolationOrder → 1], cRes, wRes]],
       y0Fixed - 0.00016, y0Fixed + 0.00016, 8][[1]];
   {y0Fixed, aFixed}
  ];
parameters2[delta_] := Module[
   {aFixed = 1.8, y0Fixed = 0.0468, cRes = 1000, wRes = 4000},
   y0Fixed =
    findMax[Function[y0, resultApproximate[delta, 0.14, y0, Interpolation[
          {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
          InterpolationOrder → 1], cRes, wRes]],
       y0Fixed - 0.00016, y0Fixed + 0.00016, 8][[1]];
   aFixed =
    findMax[Function[a, resultApproximate[delta, 0.14, y0Fixed, Interpolation[
          {{0, 0}, {1, a}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
          InterpolationOrder → 1], cRes, wRes]],
       aFixed - 0.2, aFixed + 0.2, 10][[1]];
   y0Fixed =
    findMax[Function[y0, resultApproximate[delta, 0.14, y0, Interpolation[
          {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
          InterpolationOrder → 1], cRes, wRes]],
       y0Fixed - 0.00016, y0Fixed + 0.00016, 8][[1]];
   aFixed =
    findMax[Function[a, resultApproximate[delta, 0.14, y0Fixed, Interpolation[
          {{0, 0}, {1, a}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
          InterpolationOrder → 1], cRes, wRes]],
       aFixed - 0.2, aFixed + 0.2, 10][[1]];
   {y0Fixed, aFixed}
  ];
```

The next part of the computation was performed on a remote machine and we only record the result here.

7/13/24 08:05:17 In[27]:=

```
(*
Parallelize[fit=Table[
    If[j≤8,
     Join[{0.127+0.001*j},parameters1[0.127+0.001*j]],
     Join[{0.127+0.001*(j-8)},parameters2[0.127+0.001*(j-8)]]
    ],{j,16}]];*)
fit = {{0.128, 0.046959062499999996, 1.8543945312500005},
   {0.129, 0.04695250000000001, 1.8616210937500002},
   {0.13, 0.04695250000000001, 1.8689453124999997},
   {0.131, 0.04695250000000001, 1.876171875},
   {0.132, 0.046891875, 1.88369140625}, {0.133, 0.046891875, 1.8909179687500002},
   {0.134, 0.046891875, 1.89775390625}, {0.135, 0.04689187500000001,
    1.90537109375}, {0.128, 0.046959062499999996, 1.85439453125},
   {0.129, 0.0469525, 1.86162109375}, {0.13, 0.046893437499999996, 1.869140625},
   {0.131, 0.046891875, 1.8763671875}, {0.132, 0.04689187500000001,
    1.88369140625}, {0.133, 0.04689187500000001, 1.89091796875},
   {0.134, 0.04689187500000001, 1.8977539062499997},
   {0.135, 0.046891875, 1.90517578125}};
```

We check that the differences between parameters fitted in two ways are insignificant:

7/13/24 08:05:17 In[28]:=

```
fit⟦1 ;; 8⟧ - fit⟦9 ;; 16⟧
```

7/13/24 08:05:17 Out[28]=

$\{\{0., 0., 4.440892099 \times 10^{-16}\}, \{0., 6.938893904 \times 10^{-18}, 2.220446049 \times 10^{-16}\},$
$\{0., 0.0000590625, -0.0001953125\}, \{0., 0.000060625, -0.0001953125\},$
$\{0., -1.387778781 \times 10^{-17}, 0.\}, \{0., -1.387778781 \times 10^{-17}, 2.220446049 \times 10^{-16}\},$
$\{0., -1.387778781 \times 10^{-17}, 2.220446049 \times 10^{-16}\},$
$\{0., 1.387778781 \times 10^{-17}, 0.0001953125\}\}$

7/13/24 08:05:17 In[29]:=

```
Parallelize[
  res = Table[resultApproximate[fit⟦j⟧⟦1⟧, 0.14, fit⟦j⟧⟦2⟧, Interpolation[
      {{0, 0}, {1, fit⟦j⟧⟦3⟧}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
      InterpolationOrder → 1], 1000, 4000], {j, Length[fit]}]];
```

7/13/24 08:09:18 In[30]:=

```
res⟦1 ;; 8⟧ - res⟦9 ;; 16⟧
```

7/13/24 08:09:18 Out[30]=

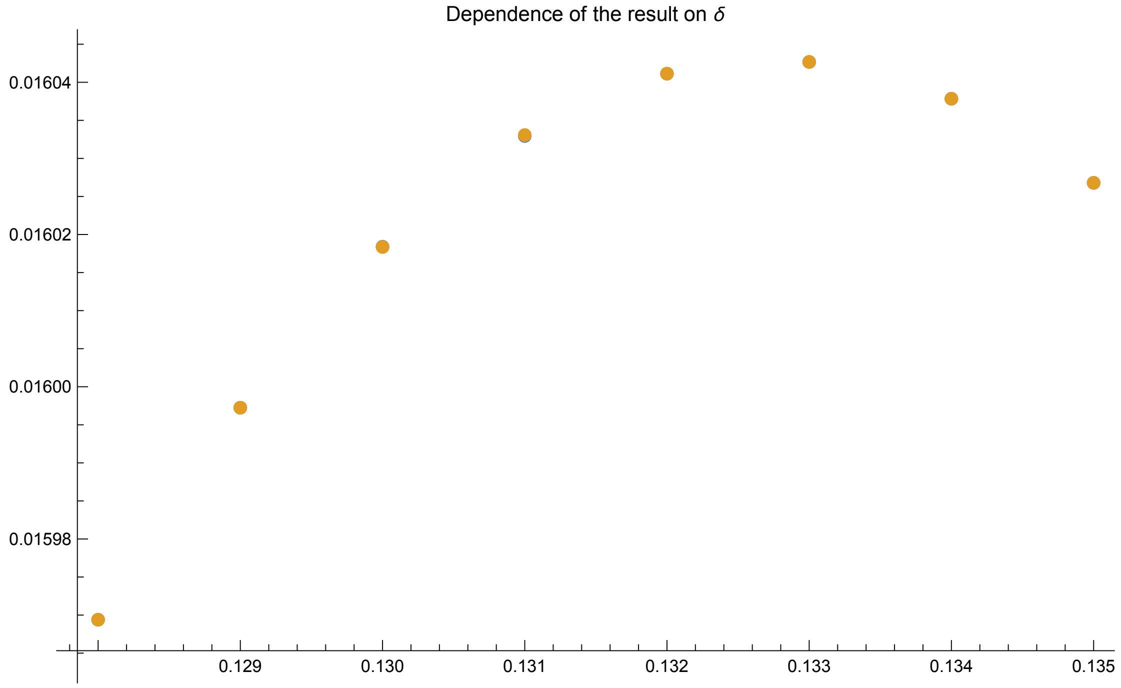
$\{0. \times 10^{-102}, 0. \times 10^{-102},$
$4.2298803231961148626337645263989429170519427547053810211772040155272739224792741338297307186 97 \times 10^{-8},$
$-1.32962617288136697808382305506911135452431945864790310614547561777088984010063299789216163276 06 \times 10^{-7}, 0. \times 10^{-102}, 0. \times 10^{-102}, 0. \times 10^{-102},$
$-1.92633040932797405738233477134132404145009861113092012382295725318019245096489685202372249109 \times 10^{-9}\}$

Again, the differences are insignificant.

7/13/24 08:09:18 In[31]:=

```
ListPlot[
 {Table[{fit[[j]][[1]], res[[j]]}, {j, 8}], Table[{fit[[j]][[1]], res[[j]]}, {j, 9, 16}]},
 ImageSize → Large, PlotLabel → "Dependence of the result on δ"]
```

7/13/24 08:09:18 Out[31]=

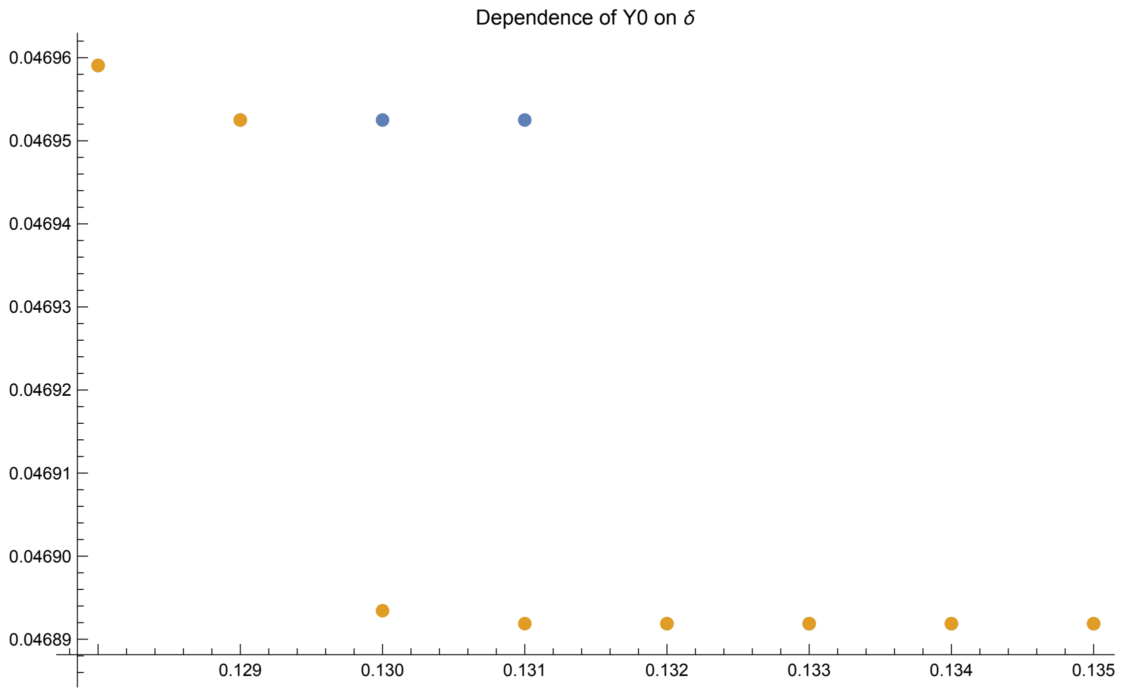


Good, so we can concentrate on $\delta \in [0.132, 0.1335]$.

7/13/24 08:09:18 In[32]:=

```
ListPlot[{Table[{fit[[j]][[1]], fit[[j]][[2]]}, {j, 8}],
   Table[{fit[[j]][[1]], fit[[j]][[2]]}, {j, 9, 16}]}, ImageSize → Large,
 PlotLabel → "Dependence of Y0 on δ", PlotRange → Full]
```

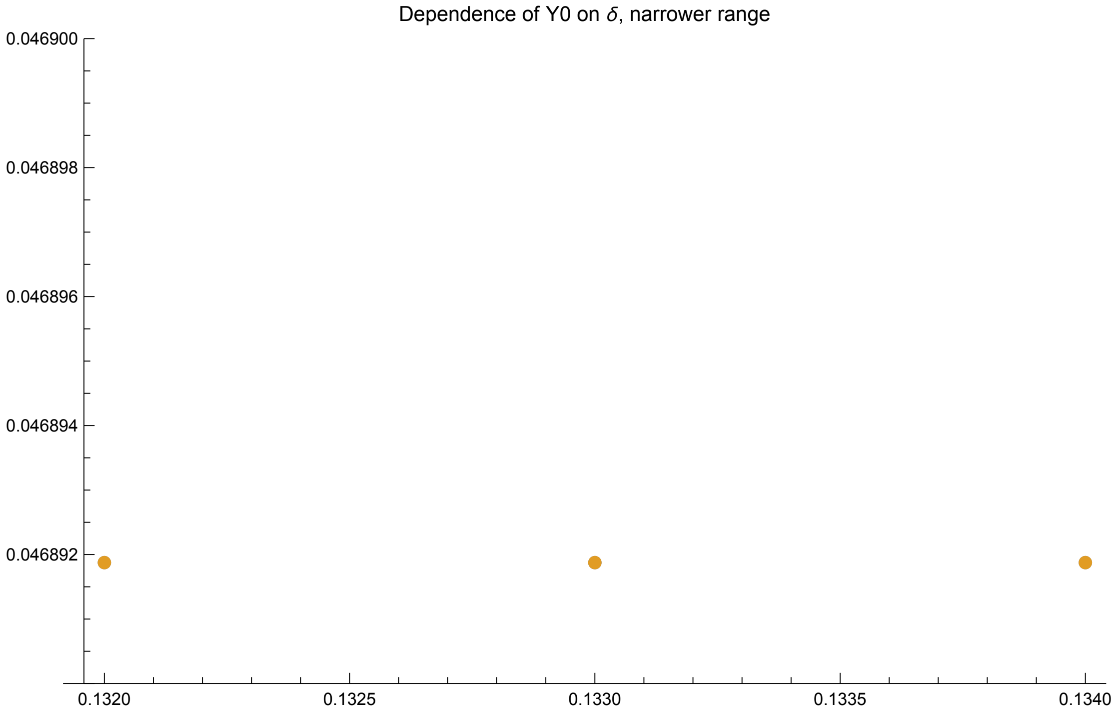
7/13/24 08:09:18 Out[32]=



7/13/24 08:09:18 In[33]:=

```
ListPlot[{Table[{fit⟦j⟧⟦1⟧, fit⟦j⟧⟦2⟧}, {j, 5, 7}],
  Table[{fit⟦j⟧⟦1⟧, fit⟦j⟧⟦2⟧}, {j, 13, 15}]},
 ImageSize → Large, PlotRange → {0.04689, 0.0469},
 PlotLabel → "Dependence of Y0 on δ, narrower range"]
```

7/13/24 08:09:18 Out[33]=

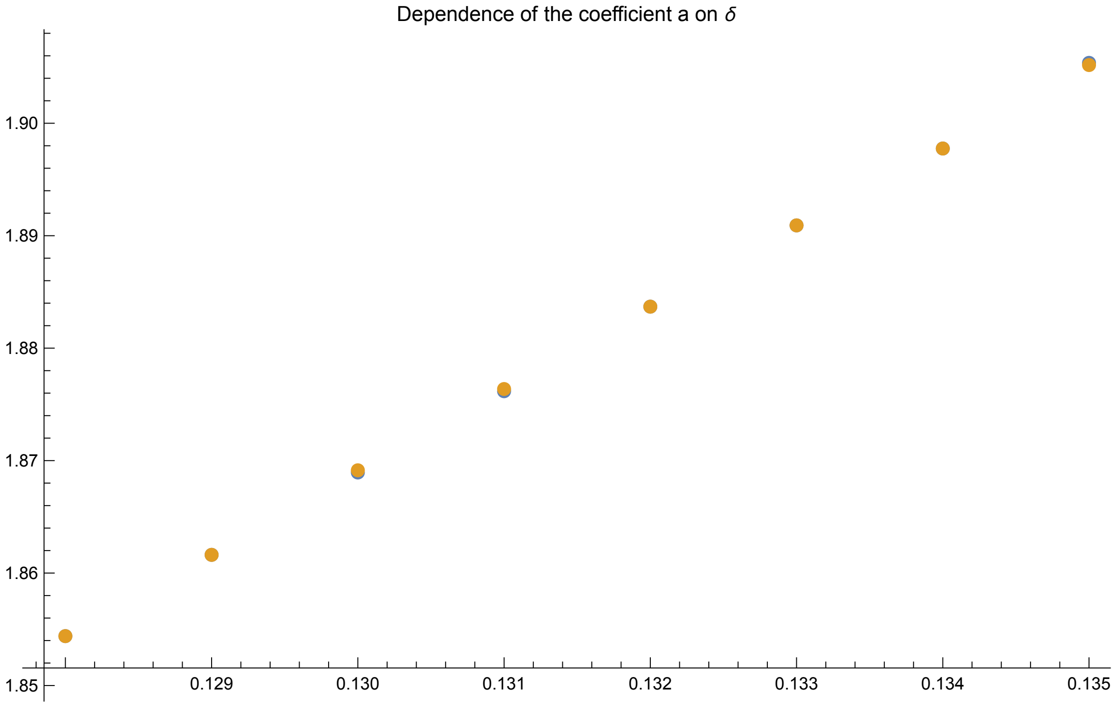


We can just take $Y_0 = 0.046892$.

7/13/24 08:09:18 In[34]:=

```
ListPlot[{Table[{fit⟦j⟧⟦1⟧, fit⟦j⟧⟦3⟧}, {j, 8}],
  Table[{fit⟦j⟧⟦1⟧, fit⟦j⟧⟦3⟧}, {j, 9, 16}]}, ImageSize → Large,
 PlotLabel → "Dependence of the coefficient a on δ"]
```

7/13/24 08:09:18 Out[34]=



7/13/24 08:09:18 In[35]:=

```
Fit[Table[{fit⟦j⟧⟦1⟧, fit⟦j⟧⟦3⟧}, {j, 5, 7}], {1, x, x^2}, x]
```

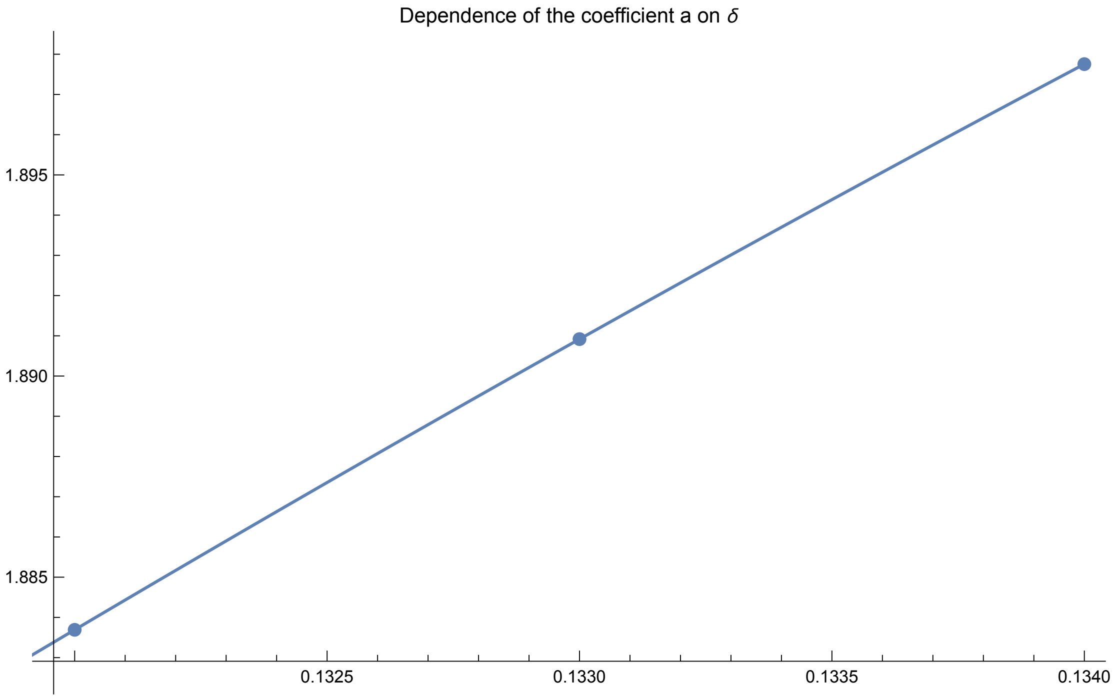
7/13/24 08:09:18 Out[35]=

$$-2.499121094 + 58.984375\, x - 195.3125\, x^2$$

7/13/24 08:09:18 In[36]:=

```
Show[{ListPlot[Table[{fit[[j]][[1]], fit[[j]][[3]]}, {j, 5, 7}],
   ImageSize → Large, PlotLabel → "Dependence of the coefficient a on δ"],
  Plot[-2.4991210937616275` + 58.984375000174424` x - 195.31250000065387` x^2,
   {x, 0.131, 0.134}]}]
```

7/13/24 08:09:18 Out[36]=



7/13/24 08:09:18 In[37]:=

```
Fit[Table[{fit[[j]][[1]], fit[[j]][[3]]}, {j, 13, 15}], {1, x, x^2}, x]
```

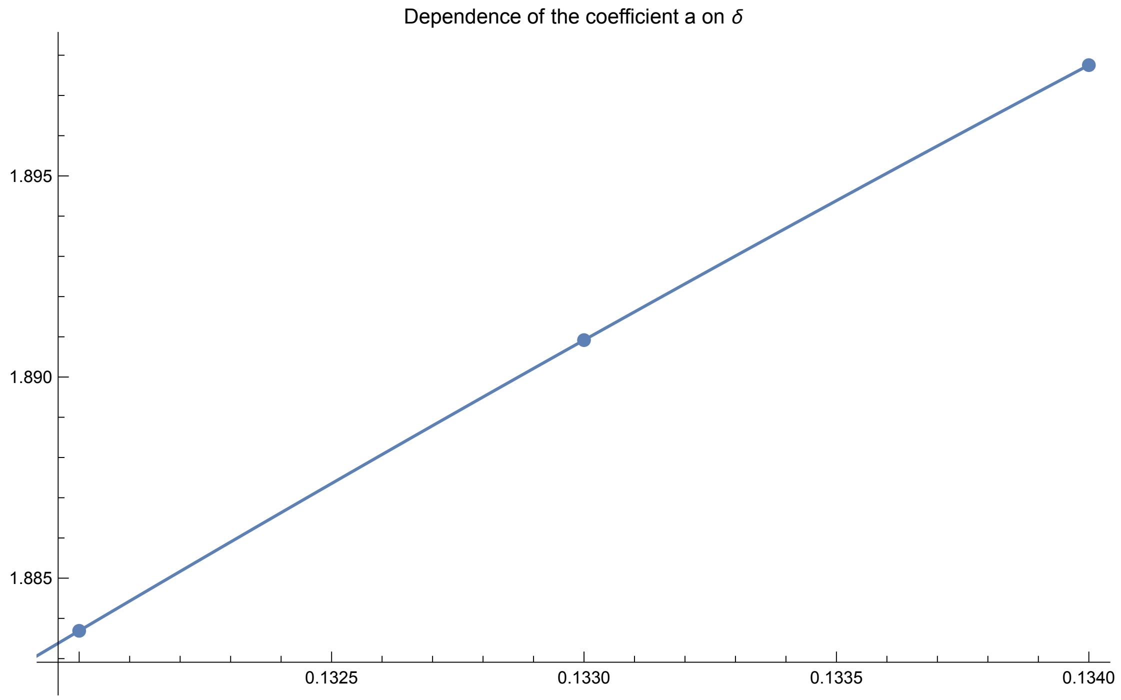
7/13/24 08:09:18 Out[37]=

$-2.499121094 + 58.984375\, x - 195.3125\, x^2$

7/13/24 08:09:18 In[38]:=

```
Show[{ListPlot[Table[{fit[[j]][[1]], fit[[j]][[3]]}, {j, 13, 15}],
   ImageSize → Large, PlotLabel → "Dependence of the coefficient a on δ"],
  Plot[-2.4991210937582196` + 58.98437500012334` x - 195.3125000004628` x^2,
   {x, 0.131, 0.134}]}]
```

7/13/24 08:09:18 Out[38]=



The second fit looks slightly better near 0.133.

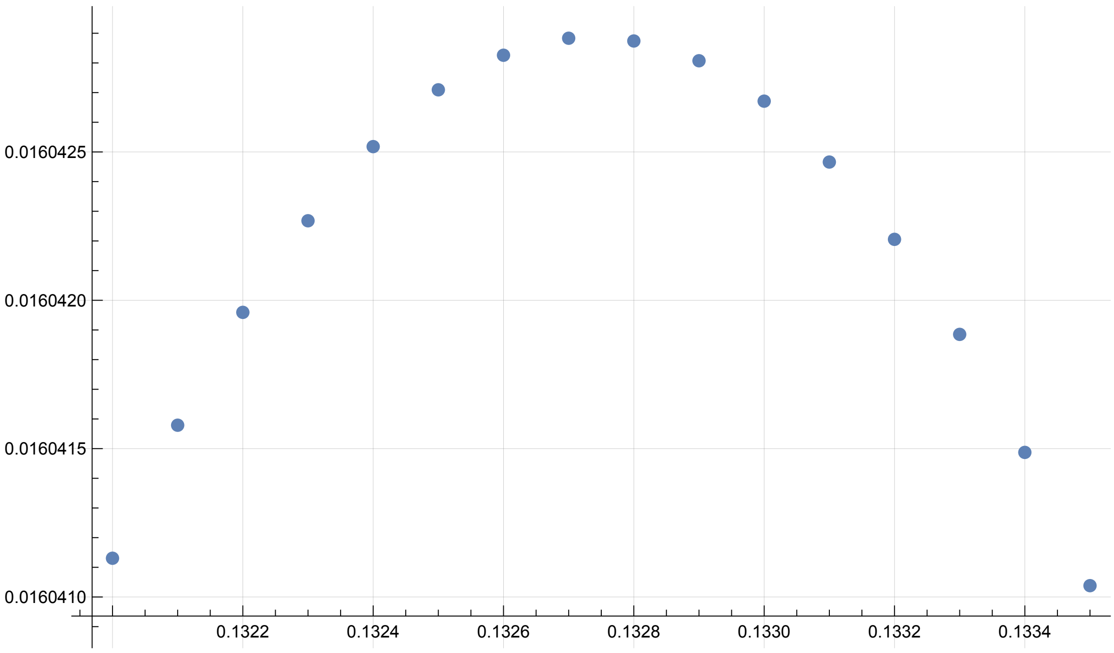
7/13/24 08:09:18 In[39]:=

```
resultFitted[delta_] := resultApproximate[delta, 0.14, 0.046892,
   Interpolation[{{0, 0}, {1, -2.4991210937582196` + 58.98437500012334` delta -
        195.3125000004628` delta^2}, {3, N[Pi, precision]},
     {4, N[Pi, precision]}}, InterpolationOrder → 1], 1000, 4000];
```

7/13/24 08:09:18 In[40]:=

```
Parallelize[
  res = Table[{delta, resultFitted[delta]}, {delta, 0.132, 0.1335001, 0.0001}]];
listplot[res]
```

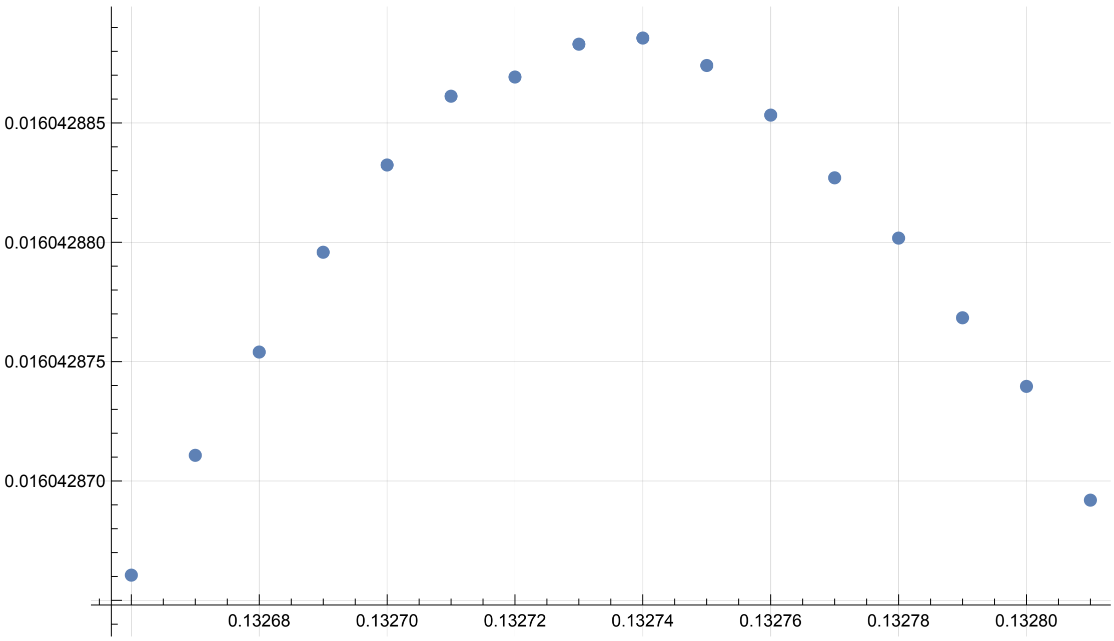
7/13/24 08:13:17 Out[41]=



7/13/24 08:13:17 In[42]:=

```
Parallelize[res =
   Table[{delta, resultFitted[delta]}, {delta, 0.13266, 0.13281001, 0.00001}]];
listplot[res]
```

7/13/24 08:17:15 Out[43]=



7/13/24 08:17:15 In[44]:=

```
delta = 0.132737;
y0Fixed = 0.046892;
aFixed =
  -2.4991210937582196` + 58.98437500012334` delta - 195.3125000004628` delta^2;
resultApproximate[delta, 0.14, y0Fixed,
 Interpolation[{{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
  InterpolationOrder → 1], 1000, 4000]
```

7/13/24 08:18:48 Out[47]=

```
0.01604288879728413441198883773201645654470961705251735108170976225437928281492
 20632582193323387585912 8
```

Re-fitting the basic parameters

Should we still change any of the parameters just a little?

7/13/24 08:18:48 In[48]:=

```
Module[
  {cRes = 1000, wRes = 4000},
  y0Fixed =
   findMax[Function[y0, resultApproximate[delta, 0.14, y0, Interpolation[
        {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
        InterpolationOrder → 1], cRes, wRes]],
     y0Fixed - 0.0000016, y0Fixed + 0.0000016, 6][[1]];
  aFixed =
   findMax[Function[a, resultApproximate[delta, 0.14, y0Fixed, Interpolation[
        {{0, 0}, {1, a}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
        InterpolationOrder → 1], cRes, wRes]],
     aFixed - 0.002, aFixed + 0.002, 6][[1]];
  y0Fixed =
   findMax[Function[y0, resultApproximate[delta, 0.14, y0, Interpolation[
        {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
        InterpolationOrder → 1], cRes, wRes]],
     y0Fixed - 0.0000016, y0Fixed + 0.0000016, 6][[1]];
  aFixed =
   findMax[Function[a, resultApproximate[delta, 0.14, y0Fixed, Interpolation[
        {{0, 0}, {1, a}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
        InterpolationOrder → 1], cRes, wRes]],
     aFixed - 0.002, aFixed + 0.002, 6][[1]];
 ];
```

7/13/24 14:04:24 In[49]:=

```
resultApproximate[delta, 0.14, y0Fixed,
 Interpolation[{{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}},
  InterpolationOrder → 1], 1000, 4000]
```

7/13/24 14:22:23 Out[49]=

```
0.01604289158889246906405381835627723436628448150333709647699165060473573821692
 77899609992394490957216 93
```

It seems we have the initial parameters. For the record, let us display their values...

7/13/24 14:22:23 In[50]:=

```
y0Fixed
```

7/13/24 14:22:23 Out[50]=

```
0.0468917625
```

7/13/24 14:22:23 In[51]:=

```
aFixed
```

7/13/24 14:22:23 Out[51]=

```
1.88889899
```

...and set them again to the same (useful if we had to close Mathematica and resume calculations later).

7/13/24 14:22:23 In[52]:=

```
delta = 0.132737;
y0Fixed = 0.0468917625;
aFixed = 1.888899;
```

Fine-tuning the argument function

Now we try to fit an optimal piecewise-linear function to serve as the argument function for the contour we have selected. Of course, it is entirely possible that some other contour with a different argument function would give a better result, but we have no way to check that. We keep adding breakpoints to our piecewise linear function and find optimal values at each breakpoint individually, assuming that the other values remain unchanged. Then we compare results for argument functions of the form $t \cdot f + (1 - t) \cdot g$, where f is the new function, g the old function and $t = 0, 1/8, \ldots, 15/8$. We pick the best one. We graph the argument functions labelling breakpoints with + or – if the argument function is convex, respectively concave at a given point. Gridlines correspond to divisions between horizontal and vertical parts of the right-hand side of the contour.

7/13/24 14:22:23 In[55]:=

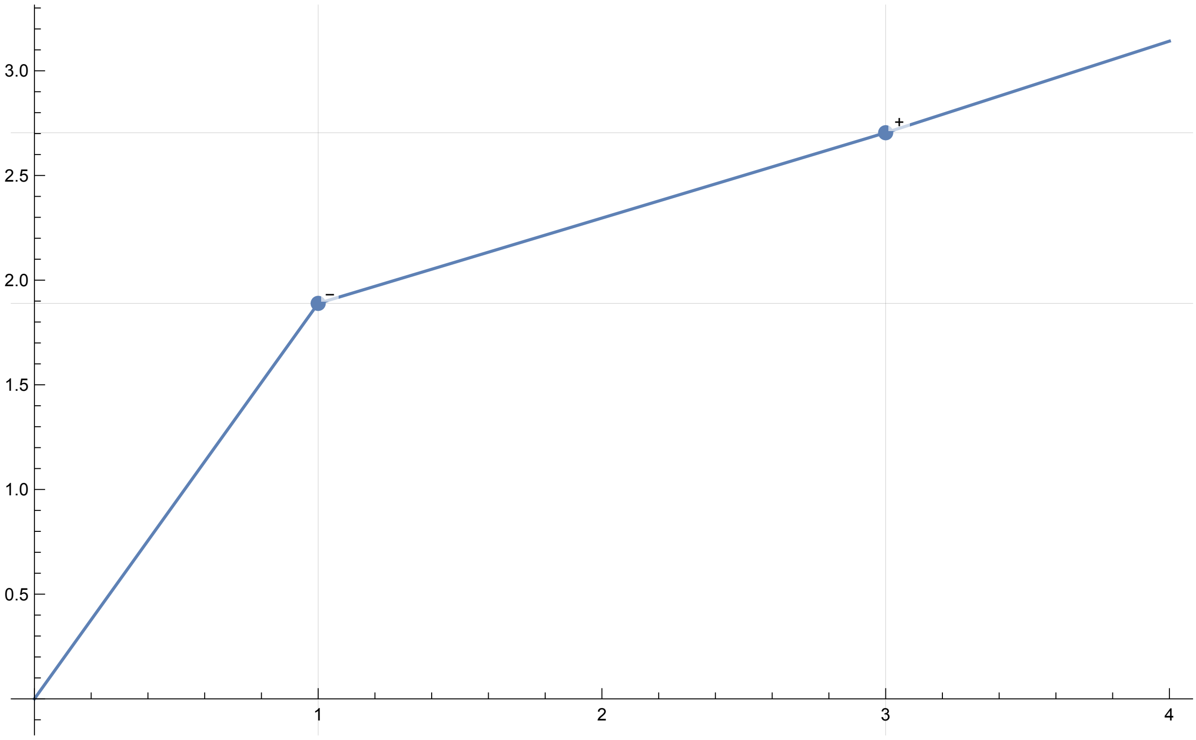
```
optimizeArguments[delta_, contourTop_, contourBottom_,
   argArr_, contourRes_, weightRes_, mobility_, iter_] := Module[
   {improvements, diffs, subst, m, res, top},
   m = Length[argArr] - 2;
   subst[x_, j_] :=
    Join[argArr[[1 ;; j]], {{argArr[[j + 1]][[1]], x}}, argArr[[j + 2 ;; m + 2]]];
   diffs = Table[argArr[[j + 1]][[2]] - argArr[[j]][[2]], {j, m + 1}];
   Parallelize[improvements = Table[
      {argArr[[j + 1]][[1]], findMax[Function[x, resultApproximate[delta, contourTop,
            contourBottom, Interpolation[subst[x, j], InterpolationOrder → 1],
            contourRes, weightRes]], argArr[[j + 1]][[2]] - mobility * diffs[[j]],
          argArr[[j + 1]][[2]] + mobility * diffs[[j + 1]], iter][[1]]}, {j, m}]];
   improvements = Join[{argArr[[1]]}, improvements, {argArr[[m + 2]]}];
   Parallelize[res = Table[resultApproximate[delta, contourTop, contourBottom,
       Interpolation[(1 - t) * argArr + t * improvements, InterpolationOrder → 1],
       contourRes, weightRes], {t, 0, 1.99, 1.0 / 8}]];
   top = Ordering[res, -1][[1]];
   Print[SetPrecision[res[[1]], 8], " -> ",
    SetPrecision[res[[top]], 8], " (moved by ", top - 1, "/8)."];
   (1 - (top - 1) / 8) * argArr + ((top - 1) / 8) * improvements
  ];
addBreakpoints[argArr_, points_] := Module[{f, narr},
   f = Interpolation[argArr, InterpolationOrder → 1];
   narr =
    Join[argArr, Table[{points[[j]], f[points[[j]]]}, {j, 1, Length[points]}]];
   Sort[narr]
  ];
double[argArr_] := addBreakpoints[argArr,
   Table[(argArr[[j]][[1]] + argArr[[j + 1]][[1]]) * 0.5, {j, Length[argArr] - 2}]];
show[argArr_] := Module[{f, slope, signed},
   f = Interpolation[argArr, InterpolationOrder → 1];
   slope[v_] := v[[2]] / v[[1]];
   signed[point_, change_] := If[Abs[change] < 10 ^ (-6), point,
     Labeled[point, If[change > 0, "+", "-"]]
    ];
   Show[{Plot[f[x], {x, 0, 4}, ImageSize → Large,
      GridLines → {{1, 3}, {f[1], f[3]}}], ListPlot[Table[signed[argArr[[j + 1]],
        slope[argArr[[j + 2]] - argArr[[j + 1]]] - slope[argArr[[j + 1]] - argArr[[j]]]],
       {j, Length[argArr] - 2}]]}, ImageSize → Full]
  ];
argArr = {{0, 0}, {1, aFixed}, {3, N[Pi, precision]}, {4, N[Pi, precision]}};
```

7/13/24 14:22:23 In[60]:=

```
argArr = optimizeArguments[delta, 0.14, y0Fixed, argArr, 1000, 4000, 0.1, 6];
show[argArr]
```

0.016042892 -> 0.016382128 (moved by 8/8).

7/13/24 15:03:57 Out[61]=



7/13/24 15:03:57 In[62]:=

```
argArr
```

7/13/24 15:03:57 Out[62]=

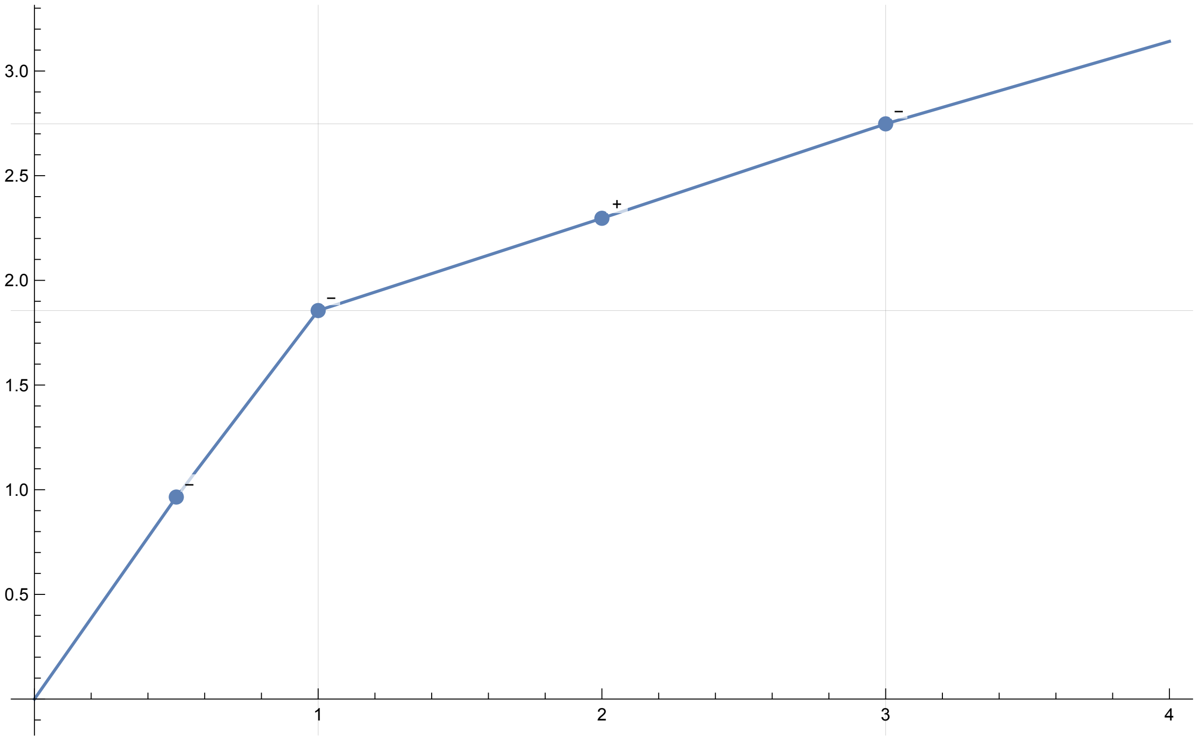
{{0, 0}, {1, 1.888995533}, {3, 2.704128542}, {4,
3.14159265358979323846264338327950288419716939937510582097494459230781640628
620899862803482534211706 8}}

7/13/24 15:03:58 In[63]:=

```
argArr = double[argArr];
argArr = optimizeArguments[delta, 0.14, y0Fixed, argArr, 1000, 4000, 0.1, 6];
show[argArr]
```

0.016382128 -> 0.016494354 (moved by 6/8).

7/13/24 15:41:46 Out[65]=



7/13/24 15:41:47 In[66]:=

```
argArr
```

7/13/24 15:41:47 Out[66]=

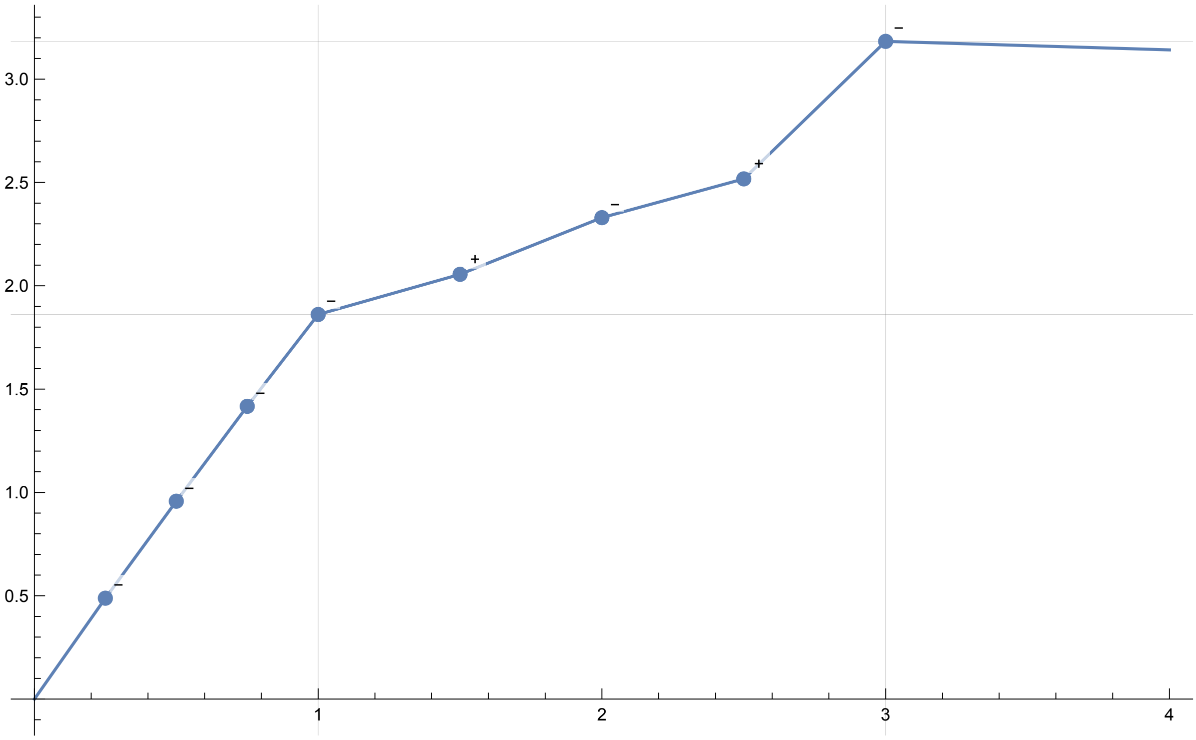
```
{{0, 0}, {0.5, 0.964974183}, {1, 1.855788896},
 {2., 2.296562037}, {3, 2.747583755}, {4,
  3.14159265358979323846264338327950288419716939937510582097494459230781640628⋱
   62089986280348253421170 68}}
```

7/13/24 15:41:47 In[67]:=

```
argArr = double[argArr];
argArr = optimizeArguments[delta, 0.14, y0Fixed, argArr, 1000, 4000, 0.1, 6];
show[argArr]
```

```
0.016494354 -> 0.016553223 (moved by 5/8).
```

7/13/24 16:42:56 Out[69]=



7/13/24 16:42:56 In[70]:=

```
argArr
```

7/13/24 16:42:56 Out[70]=

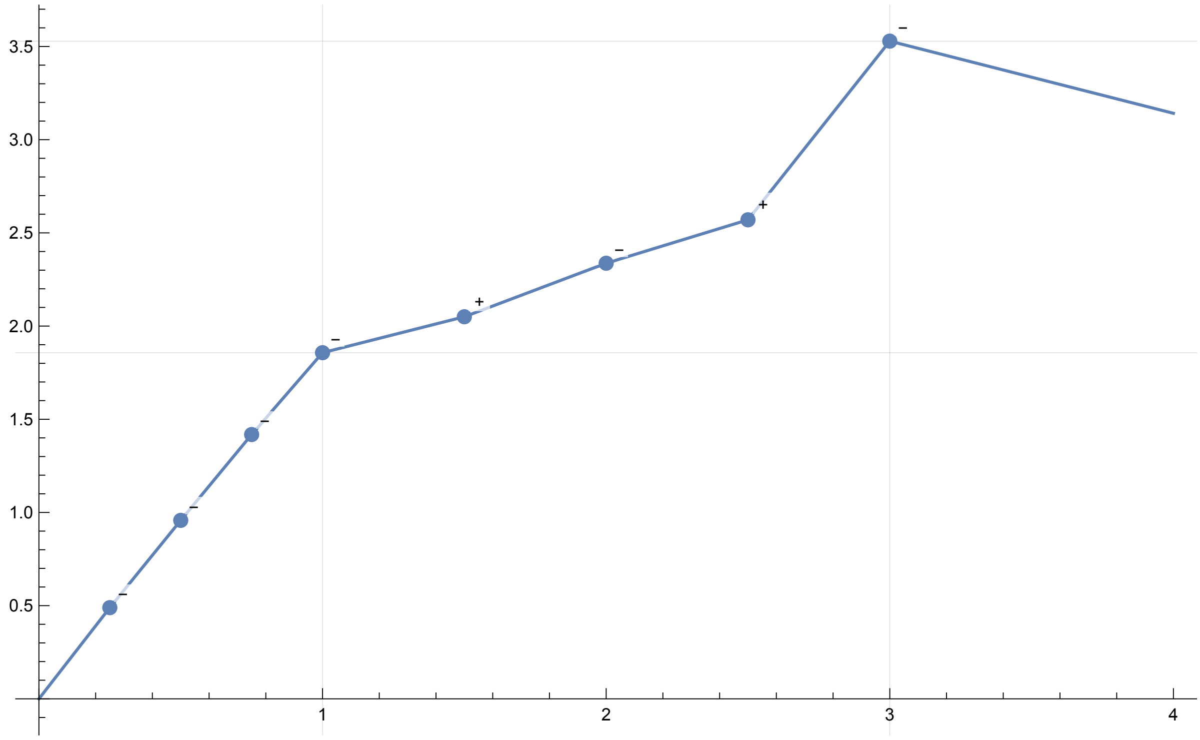
```
{{0, 0}, {0.25, 0.4881412371}, {0.5, 0.9574724128},
 {0.75, 1.416688577}, {1, 1.861110594}, {1.5, 2.055406615},
 {2., 2.32970726}, {2.5, 2.517338049}, {3, 3.182852901}, {4,
  3.14159265358979323846264338327950288419716939937510582097494459230781640628
   6208998628034825342117068}}
```

7/13/24 16:42:57 In[71]:=

```
argArr = optimizeArguments[delta, 0.14, y0Fixed, argArr, 1000, 4000, 0.05, 6];
show[argArr]
```

```
0.016553223 -> 0.016555796 (moved by 8/8).
```

7/13/24 17:44:52 Out[72]=



7/13/24 17:44:52 In[73]:=

```
argArr
```

7/13/24 17:44:52 Out[73]=

```
{{0, 0}, {0.25, 0.4893540425}, {0.5, 0.9575822513},
 {0.75, 1.41790715}, {1, 1.857102938}, {1.5, 2.049993385},
 {2., 2.337103937}, {2.5, 2.570109503}, {3, 3.52876306}, {4,
  3.14159265358979323846264338327950288419716939937510582097494459230781640628
   6208998628034825342117068}}
```

How far can we reach using a slightly higher resolution?

7/13/24 17:44:52 In[74]:=

```
resultApproximate[delta, 0.14, y0Fixed,
 Interpolation[argArr, InterpolationOrder → 1], 2000, 20 000]
```

7/13/24 17:59:42 Out[74]=

```
0.01679563382490573709488766327227215050568532146353442110993364916712981506520
 709102199391581748152511
```

Are any of the breakpoints redundant?

If we include them in the paper, we should definitely make sure that each one is significant.

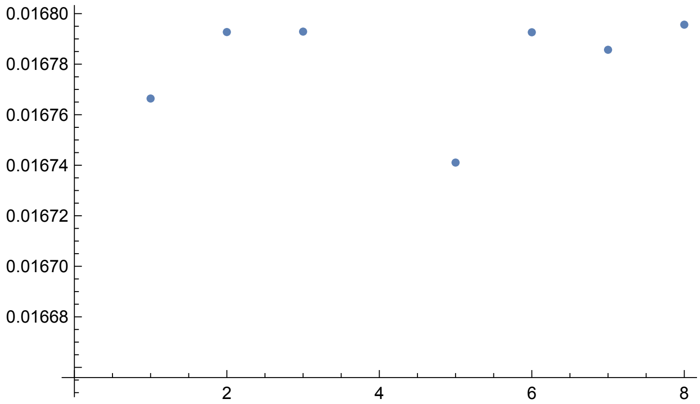
7/13/24 17:59:42 In[75]:=

```
Parallelize[redundancyCheck = Table[resultApproximate[delta, 0.14,
      y0Fixed, Interpolation[Join[argArr[[1 ;; j]], argArr[[j + 2 ;; 10]]],
       InterpolationOrder → 1], 2000, 20 000], {j, 8}]];
```

7/13/24 18:19:12 In[76]:=

```
ListPlot[redundancyCheck]
```

7/13/24 18:19:12 Out[76]=



7/13/24 18:19:12 In[77]:=

```
Grid[{redundancyCheck}]
```

7/13/24 18:19:12 Out[77]=

0.01676640427207729412271393528677230355229721362936168743565717912059103592345285836229508232220874870	0.01679271340470174595176464872657008155623445811364433114546163106434771659330427351507668617090902610	0.01679288383288203636877392462811443930025676019188017090167809491138197493093960407276569500594092738	0.01525966730161666488806523692395908083316797523737994021609503242342863374822767378433362641949613415	0.01674105572572974517399241662038935331292992250375122081511008722899144925339292744152718627677090730	0.01679262484030645488793851215836594897797589359985274756962866518958845127940478239459695851517013618	0.01678569299726448061757928428248101436681695039112929630016361789317909388872748087394621576949459193	0.01679563390648982876344509918811630063717497264179539001754720577254937109401046358039694367958591395

7/13/24 18:19:12 In[78]:=

```
resultApproximate[delta, 0.14, y0Fixed,
 Interpolation[Join[argArr⟦1 ;; 2⟧, argArr⟦5 ;; 6⟧, argArr⟦8 ;; 8⟧,
   argArr⟦10 ;; 10⟧], InterpolationOrder → 1], 2000, 20 000]
```

7/13/24 18:34:44 Out[78]=

```
0.01677126004139469194559618015201194319110500373476663157536497245859254128133679035026877854045607191
```

7/13/24 18:34:44 In[79]:=

```
Join[argArr⟦1 ;; 2⟧, argArr⟦5 ;; 6⟧, argArr⟦8 ;; 8⟧, argArr⟦10 ;; 10⟧]
```

7/13/24 18:34:44 Out[79]=

```
{{0, 0}, {0.25, 0.4893540425}, {1, 1.857102938},
 {1.5, 2.049993385}, {2.5, 2.570109503}, {4,
  3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170680}}
```

Since 3 is a natural breakpoint, we replace 2.5 with 3 and extrapolate the value so that the value at 2.5 will match.

7/13/24 18:34:45 In[80]:=

```
Interpolation[Join[argArr[[1 ;; 2]], argArr[[5 ;; 6]],
   {{3, 2.8301675617329556}}, argArr[[10 ;; 10]]], InterpolationOrder → 1][2.5]
```

7/13/24 18:34:45 Out[80]=

```
2.570109503
```

7/13/24 18:34:45 In[81]:=

```
resultApproximate[delta, 0.14, y0Fixed,
 Interpolation[Join[argArr[[1 ;; 2]], argArr[[5 ;; 6]], {{3, 2.8301675617329556}},
   argArr[[10 ;; 10]]], InterpolationOrder → 1], 2000, 20 000]
```

7/13/24 18:50:05 Out[81]=

```
0.0167712600413946919455961801520119431911050037347666315753649724585925412813
 36790350268778540456071 91
```

7/13/24 18:50:05 In[82]:=

```
argArr =
 Join[argArr[[1 ;; 2]], argArr[[5 ;; 6]], {{3, 2.8301675617329556}}, argArr[[10 ;; 10]]]
```

7/13/24 18:50:05 Out[82]=

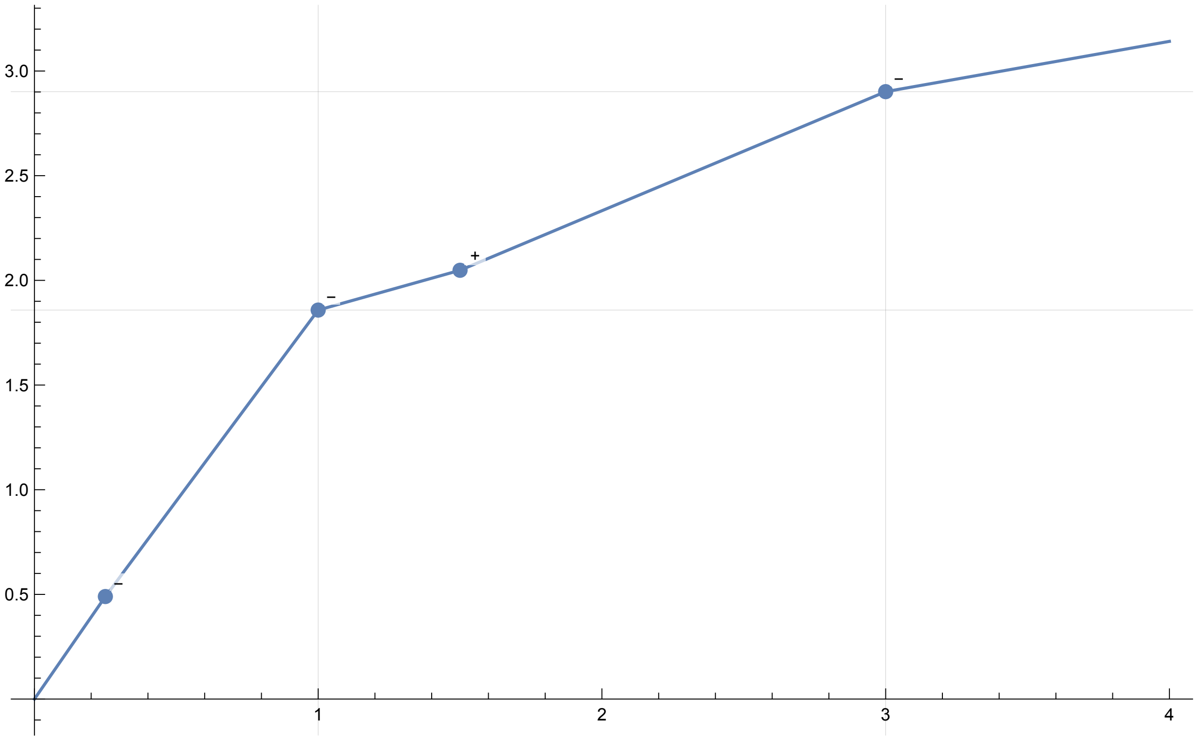
```
{{0, 0}, {0.25, 0.4893540425}, {1, 1.857102938},
 {1.5, 2.049993385}, {3, 2.830167562}, {4,
  3.1415926535897932384626433832795028841971693993751058209749445923078164062
   862089986280348253421170 68}}
```

7/13/24 18:50:06 In[83]:=

```
argArr = optimizeArguments[delta, 0.14, y0Fixed, argArr, 1000, 4000, 0.1, 6];
show[argArr]
argArr
```

0.016531535 -> 0.016534666 (moved by 8/8).

7/13/24 19:22:48 Out[84]=



7/13/24 19:22:49 Out[85]=

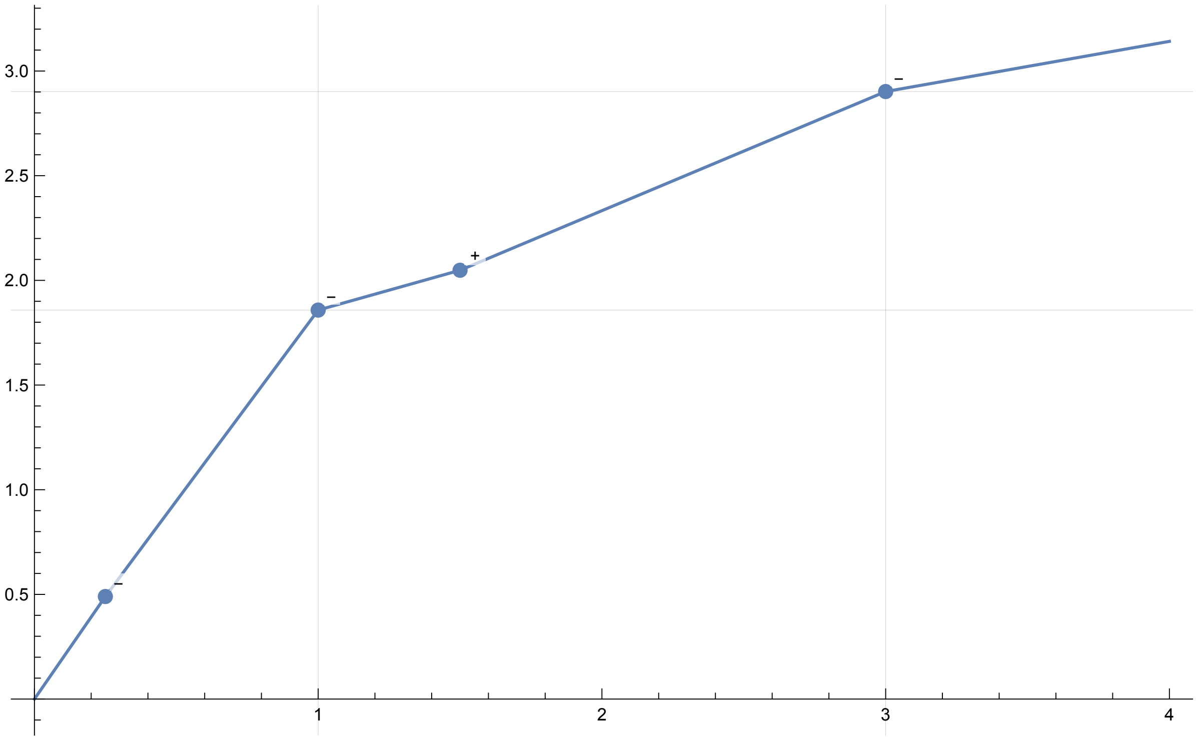
```
{{0, 0}, {0.25, 0.489747935}, {1, 1.85810324},
 {1.5, 2.048189095}, {3, 2.901392232}, {4,
  3.14159265358979323846264338327950288419716939937510582097494459230781640628·.
   62089986280348253421170 68}}
```

7/13/24 19:22:49 In[86]:=

```
argArr = optimizeArguments[delta, 0.14, y0Fixed, argArr, 1000, 4000, 0.05, 6];
show[argArr]
argArr
```

```
0.016534666 -> 0.016534671 (moved by 3/8).
```

7/13/24 19:58:27 Out[87]=



7/13/24 19:58:27 Out[88]=

```
{{0, 0}, {0.25, 0.4898193864}, {1, 1.858014753},
 {1.5, 2.048292798}, {3, 2.901891826}, {4,
  3.14159265358979323846264338327950288419716939937510582097494459230781640628
   62089986280348253421170 68}}
```

7/13/24 19:58:28 In[89]:=

```
resultApproximate[delta, 0.14, y0Fixed,
 Interpolation[argArr, InterpolationOrder → 1], 2000, 20 000]
```

7/13/24 20:13:37 Out[89]=

```
0.01677441371106705930111151922660709059040228404179804150851620685114102749 27
 7699402331861851776791578
```

7/13/24 20:13:37 In[90]:=

```
{delta, 0.14, y0Fixed, argArr}
```

7/13/24 20:13:37 Out[90]=

```
{0.132737, 0.14, 0.0468917625, {{0, 0}, {0.25, 0.4898193864},
  {1, 1.858014753}, {1.5, 2.048292798}, {3, 2.901891826}, {4,
   3.1415926535897932384626433832795028841971693993751058209749445923078164062
    862089986280348253421170 68}}}
```

A semi-rigorous estimate of the final result

Now we compute the final result, taking into account not only the values of cFNEta[] at selected contour points, but also its changes in-between. This is basically the final value mentioned in the paper, but it cannot be considered quite rigorous, as we do not use interval arithmetic here.

7/13/24 20:13:38 In[91]:=

```
result[(* contour position *) paramdelta_?NumericQ,
  paramcontourTop_?NumericQ, paramcontourBottom_?NumericQ,
  (* a function that goes from the value 0 at 0 to the value π at 4,
  that determines α_j`s in the paper *) argFun_,
  (* half of the number of divisions of the contour on each side *)
  contourRes_?NumericQ,
  (* the resolution for approximating weight functions *)
  weightRes_?IntegerQ
 ] :=
 Module[
  {
   delta, contourTop, contourBottom,
   m, t, z1, cRNArray, alpha,
   b, xPrime, xBis, y,
   indexForRN, M, u,
   betaPrime, betaBis, phiPrime, phiBis, c, symmetricPairs, phi, c0j, cn0,
   operation, wnInverse, wnMax, localc, localphi,
   cosphi, iMax, cosValue, previous, product, volume, returnValue,
   epsmachine,
   i, j, n
  },
  delta = SetPrecision[paramdelta, precision];
  contourTop = SetPrecision[paramcontourTop, precision];
  contourBottom = SetPrecision[paramcontourBottom, precision];
  m = 8 * contourRes;
  t[j_] := j / m;
  (* z1(t_j) is to be stored as z1⟦j+1⟧. *)
  z1 = Join[
    Table[(-0.5`100 + j / (2 * contourRes)) * delta + contourBottom * I,
     {j, 0, 2 * contourRes}],
    Table[0.5`100 * delta +
      (contourBottom + (contourTop - contourBottom) * j / (2 * contourRes)) * I, {j,
      1, 2 * contourRes}],
    Table[(0.5`100 - j / (2 * contourRes)) * delta + contourTop * I,
     {j, 1, 2 * contourRes}],
    Table[-0.5`100 * delta +
      (contourTop - (contourTop - contourBottom) * j / (2 * contourRes)) * I, {j,
      1, 2 * contourRes}]
   ];
  (*Monitor[*)cRNArray = Table[
    cRNest[(contourBottom + (contourTop - contourBottom) * j / (2 * contourRes))],
    {j, 0, 2 * contourRes}]
    (*,StringJoin["Computing the estimate for R_N(s) along the contour: ",
    ToString[j+1]," of ",ToString[2*contourRes+1]," points."]]*);
```

```
(* We define and compute further quantities in the order in which they
 appear in the section "The proof of Theorem 1" of the paper. *)

 (* α_1,...,α_{m+1} *)
alpha =
 Join[Table[-Pi - argFun[1 - (j - 0.5`100) / contourRes], {j, 1, contourRes}],
  Table[-Pi + argFun[(j - 0.5`100) / contourRes], {j, 1, 4 * contourRes}],
  Table[Pi - argFun[4 - (j - 0.5`100) / contourRes], {j, 1, 3 * contourRes + 1}]];

(*Claim 1.*)
(*Subsequent alpha[[j]] should differ by less than π.*)
If[Max[Table[Abs[alpha[[j + 1]] - alpha[[j]]], {j, 1, m}]] > Pi - error,
 Return[-1.1]];
(*This value should be less than π.*)
If[Max[Table[Abs[z1[[j + 1]] - z1[[j]]], {j, 1, m}]] * omega[cN] > Pi - error,
 Return[-1.2]];

b = Table[Abs[z1[[j]] - z1[[j + 1]]] / 2, {j, 1, m}];
xPrime = Table[Min[Re[z1[[j]]], Re[z1[[j + 1]]]], {j, 1, m}];
xBis = Table[Max[Re[z1[[j]]], Re[z1[[j + 1]]]], {j, 1, m}];
y = Table[Min[Im[z1[[j]]], Im[z1[[j + 1]]]], {j, 1, m}];
indexForRN = Join[
  Table[1, {j, 1, 2 * contourRes}],
  Table[j, {j, 1, 2 * contourRes}],
  Table[2 * contourRes + 1, {j, 1, 2 * contourRes}],
  Table[2 * contourRes + 1 - j, {j, 1, 2 * contourRes}]
 ];
(* We have cRN[y[[j]]]==cRNArray[[indexForRN[[j]]]] *)
(* Print["Discrepancy for RN: ",
  Max[Table[Abs[cRNprec[y[[j]]]-cRNArray[[indexForRN[[j]]]]],{j,1,m}]]];*)
M = Table[error + Sum[aUpperBound[n] *
     omega[n] ^ 2 * Exp[-omega[n] * y[[j]]], {n, 1, cN}], {j, 1, m}];
u = Table[
  Min[
    Re[cFNEta[z1[[j]]] Exp[-I * alpha[[j]]]] +
     Min[0, Re[b[[j]] * f[z1[[j]]] Exp[-I * alpha[[j]]]]],
    Re[cFNEta[z1[[j + 1]]] Exp[-I * alpha[[j]]]] +
     Min[0, -Re[b[[j]] * f[z1[[j + 1]]] Exp[-I * alpha[[j]]]]]
   ] - b[[j]] * b[[j]] * M[[j]] / 2 - cRNArray[[indexForRN[[j]]]],
  {j, 1, m}
 ];

(*Claim 2.*)
(*This value should be positive.*)
If[Min[u] < error,
 Return[-2]];
```

```
betaPrime = Table[Table[
   Pi + omega[n] * xPrime[[j]] - alpha[[j]],
   {j, 1, m}], {n, 1, cN}];
betaBis = Table[Table[
   Pi + omega[n] * xBis[[j]] - alpha[[j]],
   {j, 1, m}], {n, 1, cN}];

(*Claim 3.*)
(*The differences betaBis[[n]][[j]]-betaPrime[[n]][[j]] should be less than π.*)
If[Max[Table[Table[
     betaBis[[n]][[j]] - betaPrime[[n]][[j]],
     {j, 1, m}], {n, 1, cN}]] > Pi - error,
 Return[-3]];

phiPrime = Table[Table[
   If[Ceiling[betaPrime[[n]][[j]] / (2 * Pi)] ≤ Floor[betaBis[[n]][[j]] / (2 * Pi)],
    0,
    2 * Pi * Min[norm[betaPrime[[n]][[j]] / (2 * Pi)], norm[betaBis[[n]][[j]] / (2 * Pi)]]
   ],
   {j, 1, m}], {n, 1, cN}];
phiBis = Table[Table[
   If[Ceiling[(betaPrime[[n]][[j]] + Pi) / (2 * Pi)] ≤
     Floor[(betaBis[[n]][[j]] + Pi) / (2 * Pi)],
    Pi,
    2 * Pi * Max[norm[betaPrime[[n]][[j]] / (2 * Pi)], norm[betaBis[[n]][[j]] / (2 * Pi)]]
   ],
   {j, 1, m}], {n, 1, cN}];
c = Table[Table[
   (aUpperBound[n] / u[[j]]) * Exp[-omega[n] * y[[j]]] *
    Which[
     phiBis[[n]][[j]] ≤ Pi / 2, 1 / Cos[(phiBis[[n]][[j]] - phiPrime[[n]][[j]]) / 2],
     phiPrime[[n]][[j]] ≥ Pi / 2, 1,
     True, 1 / Cos[(Pi / 2 - phiPrime[[n]][[j]]) / 2]
    ],
   {j, 1, m}], {n, 1, cN}];
(*
Because of the symmetry cFNEta[-x+y I]==
 cFNEta[x+y I] we can skip half of the intervals.
   This part does not appear
  in the paper as it is just a numerical optimization.
   We do not include it in the rigorous check.
   This allows us to shorten the
  iterative process of choosing best division points.
*)
symmetricPairs =
 Join[Range[contourRes, 1, -1], Range[m, 5 * contourRes + 1, -1]];
If[Max[Max[Table[Table[Abs[c[[n]][[j]] - c[[n]][[symmetricPairs[[j - contourRes]]]]],
```

```
      {j, contourRes + 1, 5 * contourRes}], {n, 1, cN}]],
   Max[Table[
     Table[Abs[phiPrime[[n]][[j]] - phiPrime[[n]][[symmetricPairs[[j - contourRes]]]]],
      {j, contourRes + 1, 5 * contourRes}], {n, 1, cN}]],
   Max[
    Table[Table[Abs[phiBis[[n]][[j]] - phiBis[[n]][[symmetricPairs[[j - contourRes]]]]],
      {j, contourRes + 1, 5 * contourRes}], {n, 1, cN}]]] > error,
 Return[-0.5]
];

(* We leave out the repeated entries. *)
For[n = 1, n ≤ cN, n++,
 c[[n]] = c[[n]][[contourRes + 1 ;; 5 * contourRes]];
 phiPrime[[n]] = phiPrime[[n]][[contourRes + 1 ;; 5 * contourRes]];
 phiBis[[n]] = phiBis[[n]][[contourRes + 1 ;; 5 * contourRes]];
];
u = u[[contourRes + 1 ;; 5 * contourRes]];
y = y[[contourRes + 1 ;; 5 * contourRes]];

(* We will be evaluating  max_{j=0,…,m} c_{n,j} max(v(φ'_{n,j}, 2πx), v(φ''_{n,j}, 2πx)),
so we merge phiPrime and phiBis to make it simpler. *)
phi = Table[Join[phiPrime[[n]], phiBis[[n]]], {n, 1, cN}];
For[n = 1, n ≤ cN, n++,
 c0j = Select[Range[m / 2],
   Function[j, phiBis[[n]][[j]] ≥ Pi / 2 && phiPrime[[n]][[j]] ≤ Pi / 2]];
 (* The set of j for max_{φ'_{n,j} ≤ π/2, φ''_{n,j} ≥ π/2} *)
 If[Length[c0j] == 0,
  (* The restricted max was empty, no need to add c_{n,0}. *)
  c[[n]] = Join[c[[n]], c[[n]]]
  ,
  (* We do need to add c_{n,0} and φ'_{n,0}=φ''_{n,0}=π/2. *)
  cn0 = aUpperBound[n] * Max[Table[Exp[-omega[n] * y[[c0j[[i]]]]] / (u[[c0j[[i]]]] *
        Cos[(Pi / 2 - phiPrime[[n]][[c0j[[i]]]]) / 2]), {i, 1, Length[c0j]}]];
  c[[n]] = Join[c[[n]], c[[n]], {cn0}];
  phi[[n]] = Join[phi[[n]], {Pi / 2}]
 ]
];
wnMax = Table[Min[1, Max[Table[
     (c[[n]][[j]] + error) * (Cos[phi[[n]][[j]]] + 1), {j, Length[c[[n]]]}]]], {n, cN}];
For[n = cN - 1, n ≥ 1, n--, wnMax[[n]] += wnMax[[n + 1]]];
For[n = cN, n ≥ 1, n--,
 operation = "Computing";
```

```
 epsmachine = 0.0 + epsilon;
 wnInverse = Table[0.5, {i, weightRes}];
 For[j = 1, j ≤ Length[c⟦n⟧], j++,
  localc = 0.0 + c⟦n⟧⟦j⟧ + error; (*To machine precision*)
  localphi = 0.0 + phi⟦n⟧⟦j⟧;
  cosphi = Cos[localphi];
  iMax = Floor[Min[1, localc * (cosphi + 1)] * weightRes];
  For[i = 1, i ≤ iMax, i++,
   (*Solve c(Cos[ϕ]-Cos[ϕ+2π(epsilon+x)])==i/weightRes*)
   cosValue = cosphi - i / (weightRes * localc);
   wnInverse⟦i⟧ =
    Min[wnInverse⟦i⟧, (ArcCos[cosValue] - localphi) / (2 Pi) - epsmachine]
  ];
 ];
 For[i = weightRes, i ≥ 1, i--,
  wnInverse⟦i⟧ = Floor[wnInverse⟦i⟧ * weightRes ^ 2]
 ];
 For[i = weightRes, i ≥ 2, i--,
  wnInverse⟦i⟧ -= wnInverse⟦i - 1⟧
 ];
 If[n == cN,
  previous = wnInverse
  ,
  operation = "Convoluting";
  product =
   Join[{0}, ListConvolve[previous, wnInverse, 1, 0]⟦1 ;; weightRes - 1⟧];
  previous = product
 ]
];
volume = 2 ^ cN * Sum[product⟦i⟧, {i, weightRes}] / weightRes ^ (2 cN);
returnValue = 2 * volume / delta;
returnValue
];
```

7/13/24 20:13:38 In[92]:=

```
result[0.132737, 0.14, 0.0468917625,
 Interpolation[{{0, 0}, {0.25`100, 0.48981938638676636`100},
   {1, 1.858014753262597`100}, {1.5`100, 2.048292798102907`100},
   {3, 2.9018918260604507`100}, {4, Pi}}, InterpolationOrder → 1], 1000, 4000]
```

7/13/24 20:17:02 Out[92]=

```
0.016353859605893459306920052785705055944080647108315378279566660677497975058 6
 37528337282834300965303 25
```

Maybe we do not need that many digits.

7/13/24 20:17:02 In[93]:=

```
result[0.132737, 0.14, 0.0468918, Interpolation[
  {{0, 0}, {0.25`100, 0.489819`100}, {1, 1.85802`100}, {1.5`100, 2.04829`100},
   {3, 2.90189`100}, {4, Pi}}, InterpolationOrder → 1], 1000, 4000]
```

7/13/24 20:20:27 Out[93]=

```
0.01635387324557644992522443690447618120055556263774533553796306188674195100362
 82823445668902212412599710
```

What resolution do we actually need?

7/13/24 20:20:28 In[94]:=

```
result[0.132737, 0.14, 0.0468918, Interpolation[
  {{0, 0}, {0.25`100, 0.489819`100}, {1, 1.85802`100}, {1.5`100, 2.04829`100},
   {3, 2.90189`100}, {4, Pi}}, InterpolationOrder → 1], 2500, 10 000]
```

7/13/24 20:43:33 Out[94]=

```
0.016640837848428682179400849486176046857429280157127897060412653734023739525·
 29863618658370507910936570
```

7/13/24 20:43:33 In[95]:=

```
result[0.132737, 0.14, 0.0468918, Interpolation[
  {{0, 0}, {0.25`100, 0.489819`100}, {1, 1.85802`100}, {1.5`100, 2.04829`100},
   {3, 2.90189`100}, {4, Pi}}, InterpolationOrder → 1], 1500, 16 000]
```

7/13/24 21:01:00 Out[95]=

```
0.016675741496460891286336505640320353672332549943453364374764415104846649525·
 8425154938494122092817562
```

It seems that the resolutions 1500/16000 will be perfectly sufficient to obtain 1/60 as the final result, and we could not get a significantly greater result by increasing the resolution.

Finally, we note that in the paper the entire contour is defined on [0,1], whereas the argument function above was on [0,4] for just one half of the contour. Therefore the argument function $\alpha(t)$ in the paper has appropriately rescaled breakpoints.