

Finding Trott Constants

Michael Trott

A Trott constant is a number whose continued fraction representation (simple or not) is the same as the digits of its radix representation. For instance, in base 10, $0.1084101\dots = 0 + 1 / (1 + 1 / (0 + 1 / (8 + 1 / (4 + 1 / (1 + 1 / (0 + 1 / (1 \dots)))))))$. This Corner implements a search method for such numbers and lists its results. The best number we found has more than 400 digits in common between its nonsimple, zero-free continued fraction representation and its radix representation.

■ Introduction

A few years ago I dreamed of a number whose continued fraction digits and decimal digits were identical. Once awake, I could not remember the digits of the number. So I tried to re-create the number (if such a number even exists) with *Mathematica*. I found that such a number (and even more than one) seems to exist.

In this Corner, we will implement a search program that looks for such numbers. All of the following will be purely experimental; I have not been able to prove existence, uniqueness, or any properties of these numbers.

■ Existing Sequences

The On-Line Encyclopedia of Integer Sequences currently (January 2006) lists the following Trott constants:

www.research.att.com/projects/OEIS?Anum=A039662
0, 1, 0, 8, 4, 1, 0, 1, 5, 1, 2, 2, 3, 1, 1, 1, 3, 6, 1, 5, 1, 1, 2, 9, 0, ...

Meaning the following holds:

$$0 + 1 / (1 + 1 / (0 + 1 / (8 + 1 / (4 + 1 / (1 + 1 / (0 + 1 / (1 \dots))))))) = 0.1084101512\dots$$

www.research.att.com/projects/OEIS?Anum=A091694
0, 2, 7, 3, 9, 4, 4, 1, 9, 5, 7, 3, 9, 2, 7, 1, 6, 1, 7, 1, 7, 1, 4, 5, 9, ...

So the following holds:

$$0 + 2 / (7 + 3 / (9 + 4 / (4 + 1 / (9 + 5 / (7 + 3 / (9 + 2 / (7 \dots))))))) = 0.2739441957\dots$$

www.research.att.com/projects/OEIS?Anum=A113307

0, 4, 8, 2, 6, 7, 7, 2, 8, 1, 9, 3, 9, 1, 8, 1, 5, 9, 9, 4, 9, ...

$$0 + 4 / (8 + 2 / (6 + 7 / (7 + 2 / (8 + 1 / (9 + 3 / (9 + 1 / (8 \dots))))))) = 0.4826772819\dots$$

Section 1.2.3 of *The Mathematica GuideBook for Programming* uses a version of the following one-liner to find such a number.

```
In[1]:= cfDigits = Nest[First/@Take[#, Min[43, Length[#]]] & [Sort[
  {#, N[Abs[FromDigits[#, 1], 10] -
    Fold[#2[[2]] / (#2[[1]] + #1) &, #][[-2]] / #][[-1]],
    Partition[Reverse[Drop[#, -2]], 2]], 50]} & /@Flatten[
  Flatten[Table[Join[#, {i, j}], {i, 0, 9}, {j, 9}], 1] & /@#, 1],
  (#1[[2]] < #2[[2]]) &]] &, {0}], 71][[1]]
```

```
Out[1]:= {0, 2, 7, 3, 9, 4, 4, 1, 9, 5, 7, 3, 9, 2, 7, 1, 6, 1, 7, 1, 7, 1, 4, 5, 9,
  1, 5, 2, 7, 2, 4, 2, 8, 5, 9, 1, 9, 2, 7, 3, 7, 2, 5, 1, 8, 7, 7, 2, 9,
  8, 8, 1, 9, 8, 6, 2, 9, 1, 9, 1, 7, 3, 8, 3, 7, 5, 5, 2, 8, 1, 7, 1, 7,
  7, 4, 1, 8, 1, 9, 6, 9, 4, 6, 1, 9, 1, 7, 3, 8, 2, 8, 3, 6, 2, 5, 1, 6,
  1, 5, 4, 8, 5, 9, 3, 6, 4, 7, 1, 9, 2, 5, 8, 9, 4, 9, 8, 9, 1, 5, 1,
  7, 2, 7, 3, 9, 1, 9, 6, 7, 6, 9, 2, 8, 1, 9, 7, 9, 7, 9, 1, 8, 6, 8}
```

Here is a continued fraction made from this sequence of integers.

```
In[2]:= (heldCF = With[{l = C /@ cfDigits}, DeleteCases[
  Hold[0 + #] & @@ {Fold[#2[[2]] / (#2[[1]] + #1) &,
    l][[-2]] / l][[-1]], Partition[Reverse[Drop[l, -2]], 2]]],
  C, Infinity, Heads -> True]) // ToString[#, InputForm] &

Out[2]:= Hold[0 + 2/(7 + 3/(9 + 4/(4 + 1/(9 + 5/(7 + 3/(9 + 2/(7 + 1/(6 + 1/(7 + 1/
  (7 + 1/(4 + 5/(9 + 1/(5 + 2/(7 + 2/(4 + 2/(8 + 5/(9 + 1/(9 + 2/(7 + 3/(7 +
  2/(5 + 1/(8 + 7/(7 + 2/(9 + 8/(8 + 1/(9 + 8/(6 + 2/(9 + 1/(9 + 1/(7 + 3/
  (8 + 3/(7 + 5/(5 + 2/(8 + 1/(7 + 1/(7 + 7/(4 + 1/(8 + 1/(9 + 6/(9 + 4/
  (6 + 1/(9 + 1/(7 + 3/(8 + 2/(8 + 3/(6 + 2/(5 + 1/(6 + 1/(5 + 4/(8 + 5/
  (9 + 3/(6 + 4/(7 + 1/(9 + 2/(5 + 8/(9 + 4/(9 + 8/(9 + 1/(5 + 1/(7 + 2/
  (7 + 3/(9 + 1/(9 + 6/(7 + 6/(9 + 2/(8 + 1/(9 + 7/(9 + 7/(1/(6/8 + 8) +
  9))))))))))))))))))))))))))))))))))))))))))))))))))))))))))]]
```

Here is the corresponding fraction and its 100-digit decimal approximation.

```
In[3]:= collapsedCFFraction = ReleaseHold[heldCF]

Out[3]:= 
$$\frac{176620790070482944327792214614818452307719298881361958226}{644732733226379679234128519930326742608159320893109660883}$$


In[4]:= N[collapsedCFFraction, 100]

Out[4]:= 0.27394419573927161717145915272428591927372518772988198629191738375`
52817177418196946191738283625161549
```

The first 100 digits of the continued fraction and the decimal expansion are identical.

```
In[5]:= Take[RealDigits[collapsedCFFraction, 10, 101, 0][[1]], 100] ==
      Take[cfDigits, 100]
```

```
Out[5]= True
```

We will implement an expanded version of the program that calculated `heldCF` and use it to find various numbers (which *MathWorld's* creator Eric Weisstein calls “Trott’s Constants” [1]) whose simple or nonsimple continued fraction approximation agrees with their base b radix representation up to a few hundred digits.

■ Some Formatting Functions

We start by defining functions to format continued fractions. The function `formatNonSimpleContinuedFraction` formats a list of integers as a nonsimple continued fraction. To avoid autoevaluation of the result to a single rational number, we use `HoldForm`.

```
In[6]:= formatNonSimpleContinuedFraction[l_List] := With[{f = 1[[1]],
      l = C / @ If[EvenQ[Length[l]], Append[Rest[l], 1], Rest[l]]},
      DeleteCases[HoldForm[f + #] & @@
        {Fold[#2[[2]] / (#2[[1]] + #1) &, 1[[[-2]]] / 1[[[-1]]],
          Partition[Reverse[Drop[l, -2]], 2]}], C, Infinity,
      Heads → True] /. HoldPattern[Times[Power[1, -1], x_] :=> x];
```

Here is an example.

```
In[7]:= formatNonSimpleContinuedFraction[{0, 1, 2, 3, 4, 5, 6, 7, 8, ...}]
```

```
Out[7]= 0 +  $\frac{1}{2 + \frac{3}{4 + \frac{5}{6 + \frac{7}{8 + \dots}}}}$ 
```

We use `formatNonSimpleContinuedFraction` to define a similar function for simple continued fractions.

```
In[8]:= formatContinuedFraction[l_] := formatNonSimpleContinuedFraction@
      Flatten[{First[l], Transpose[{Table[1, {Length[l] - 1}], Rest[l]}]}]
```

Here again is an example.

```
In[9]:= formatContinuedFraction[{0, 1, 2, 3, 4, 5, 6}]
```

```
Out[9]= 0 +  $\frac{1}{1 + \frac{1}{2 + \frac{1}{3 + \frac{1}{4 + \frac{1}{5 + \frac{1}{6}}}}}}$ 
```

For a slightly less readable, but more space-conserving way to write such continued fractions, we define two more functions.

```

In[10]:= formatNonSimpleContinuedFractionShortString[l_List] := StringTake[
    ToString[formatNonSimpleContinuedFraction[l], InputForm], {10, -2}]

In[11]:= formatNonSimpleContinuedFractionShortString[Range[20]]

Out[11]= 1 + 2/(3 + 4/(5 + 6/(7 + 8/(9 + 10/(11 +
    12/(13 + 14/(15 + 16/(17 + 18/(19 + 20))))))))))

In[12]:= formatContinuedFractionShortString[l_List] := StringTake[
    ToString[formatContinuedFraction[l], InputForm], {10, -2}]

In[13]:= formatContinuedFractionShortString[
    {0, 1, 6, 1, 5, 2, 8, 3, 8, 4, 8, 8, 9}]

Out[13]= 0 + 1/(1 + 1/(6 + 1/(1 + 1/(5 + 1/(2 + 1/
    (8 + 1/(3 + 1/(8 + 1/(4 + 1/(8 + 1/(8 + 1/9))))))))))

```

■ Difference Between Continued Fractions and Radix Forms

The next two functions construct a rational number (an integer in degenerate cases) from the lists of digits.

```

In[14]:= fromContinuedFraction = FromContinuedFraction;

In[15]:= fromNonSimpleContinuedFraction[l_List] :=
    With[{l = If[EvenQ[Length[l]], Append[l, 1], 1]},
    Fold[#2[[2]] / (#2[[1]] + #1) &, l[[ -2]] / l[[ -1]],
    Partition[Reverse[Drop[l, -2]], 2]]]

```

To measure the quality of a Trott constant, we compare the value of the radix form and the continued fraction. The function `radixCFDifference` calculates the difference.

```

In[16]:= δCF[l_List, base_Integer] :=
    N[Abs[FromDigits[{1, 1}, base] - fromContinuedFraction[l]], 20];

In[17]:= δNSCF[l_List, base_Integer] := N[Abs[
    FromDigits[{1, 1}, base] - fromNonSimpleContinuedFraction[l]], 20];

In[18]:= radixCFDifference[kind: ("CF" | "NSCF"), digitList_, base_] :=
    If[kind === "CF", δCF, δNSCF][digitList, base]

```

Here again are two examples.

```

In[19]:= radixCFDifference["CF", {0, 1, 0, 8, 4, 1, 0, 1, 5, 1, 2, 2, 3, 1}, 10]

Out[19]= 1.5739703371637833060 × 10-9

In[20]:= radixCFDifference["NSCF", {0, 1, 6, 1, 5, 2, 8, 3, 8, 4, 8, 8, 9}, 10]

Out[20]= 6.0952401746724890830 × 10-10

```

The function `numberOfCoincidingDigits` calculates the number of coinciding digits in the continued fraction and base b decimal representation (including the leading zero).

```

In[21]:= numberOfCoincidingDigits[kind : ("CF" | "NSCF"), digitList_, base_] :=
Module[{fromList, λ, fraction, digitPairs, pos},
  fromList = If[kind == "CF", fromContinuedFraction,
    fromNonSimpleContinuedFraction];
  λ = Length[digitList];
  fraction = fromList[digitList];
  digitPairs =
    Transpose[{RealDigits[fraction, base, λ, 0][[1]], digitList]};
  pos = Position[Equal@@@digitPairs, False, {1}, 1];
  If[pos == {}, Length[digitList], pos[[1, 1]] - 1]

```

Here are our two examples.

```

In[22]:= numberOfCoincidingDigits["CF",
  {0, 1, 0, 8, 4, 1, 0, 1, 5, 1, 2, 2, 3, 1}, 10]

```

Out[22]= 9

```

In[23]:= numberOfCoincidingDigits["NSCF",
  {0, 1, 6, 1, 5, 2, 8, 3, 8, 4, 8, 8, 9}, 10]

```

Out[23]= 10

Here we analyze the three sequences from the integer database.

```

In[24]:= numberOfCoincidingDigits["CF",
  {0, 1, 0, 8, 4, 1, 0, 1, 5, 1, 2, 2, 3, 1, 1, 1, 3, 6, 1,
    5, 1, 1, 2, 9, 0, 8, 1, 1, 4, 0, 6, 4, 1, 5, 0, 9, 1, 1, 2,
    2, 1, 5, 8, 0, 9, 0, 9, 3, 9, 0, 9, 0, 9, 1}, 10]

```

Out[24]= 35

```

In[25]:= numberOfCoincidingDigits["NSCF",
  {0, 2, 7, 3, 9, 4, 4, 1, 9, 5, 7, 3, 9, 2, 7, 1, 6, 1, 7, 1,
    7, 1, 4, 5, 9, 1, 5, 2, 7, 2, 4, 2, 8, 5, 9, 1, 9, 2, 7, 3, 7,
    2, 5, 1, 8, 7, 7, 2, 9, 8, 8, 1, 9, 8, 6, 2, 9, 1, 9, 1, 7, 3,
    8, 3, 7, 5, 5, 2, 8, 1, 7, 1, 7, 7, 4, 1, 8, 1, 9, 6, 9, 4, 6,
    1, 9, 1, 7, 3, 8, 2, 8, 3, 6, 2, 5, 1, 6, 1, 5, 4, 8, 5}, 10]

```

Out[25]= 71

```

In[26]:= numberOfCoincidingDigits["NSCF",
  {0, 4, 8, 2, 6, 7, 7, 2, 8, 1, 9, 3, 9, 1, 8, 1, 5, 9, 9, 4,
    9, 3, 9, 2, 9, 1, 5, 2, 7, 9, 6, 1, 7, 2, 7, 8, 9, 1, 8, 3, 5,
    6, 7, 1, 7, 5, 7, 1, 7, 1, 5, 1, 9, 2, 9, 2, 9, 1, 6, 3, 7, 3,
    6, 1, 8, 2, 6, 1, 6, 1, 8, 1, 7, 6, 9, 2, 9, 1, 5, 2, 9, 3, 9,
    5, 6, 2, 7, 2, 6, 5, 8, 1, 3, 1, 8, 3, 8, 6, 8, 3, 6, 1}, 10]

```

Out[26]= 71

We define another function for later use. Given a sequence, the function `shortenedSequence` shortens it to the minimal length that exhibits the maximal number of coinciding digits.

```
In[27]:= shortenedSequence[
  kind : {"CF" | "NSCF"}, digitList_List, base_Integer] :=
Module[{dλ = numberOfCoincidingDigits[kind, digitList, base],
  dropCounter = 1}, While[numberOfCoincidingDigits[kind,
  Drop[digitList, -dropCounter], base] == dλ, dropCounter++];
  Drop[digitList, -dropCounter + 1]]
```

Here a continuation of the sequence (which does not improve) is shortened to its optimal length.

```
In[28]:= shortenedSequence["CF",
  {0, 1, 0, 8, 4, 1, 0, 1, 5, 1, 2, 2, 3, 1, 2, 4, 5, 6, 3, 2, 4, 5, 4, 3}, 10]
Out[28]= {0, 1, 0, 8, 4, 1, 0, 1, 5, 1, 2, 2, 3, 1, 2}
In[29]:= {radixCFDifference["CF", %, 10],
  numberOfCoincidingDigits["CF", %, 10]}
Out[29]= {1.3684358256515306703 × 10-10, 10}
```

■ Investigating Small Sequences

Now let us have a preliminary look at the differences between the radix and continued fraction representations. The function `allNDigitSequences` returns a list of all $(n + 1)$ -digit base b sequences. We will always assume that the first digit is zero.

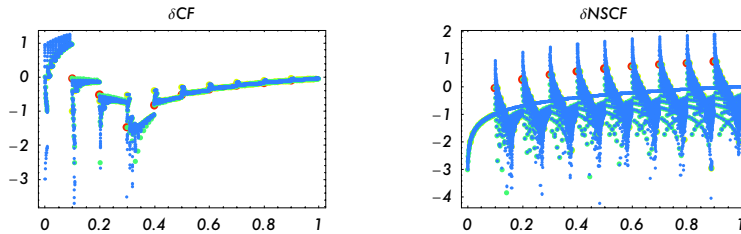
```
In[30]:= allNDigitSequences[n_, base_, pre_ : {0}] :=
  Flatten[Table[Evaluate[Join[pre, Table[j[k], {k, n}]]],
    Evaluate[Sequence @@ Table[{j[k], 0, base - 1}, {k, n}]]], n - 1]
```

The function `sequencesAndδs` adds the difference between the radix and continued fraction representations to the sequences.

```
In[31]:= sequencesAndδs[n_, δ_, base_, pre_ : {0}] :=
  Cases[#, {_?NumberQ, _?NumberQ} &@ (Internal`DeactivateMessages[
    {N[FromDigits[{#, 1}, base]], N[Log[10, Abs[δ[#, base]]]]}] & /@
    allNDigitSequences[n, base, pre])
```

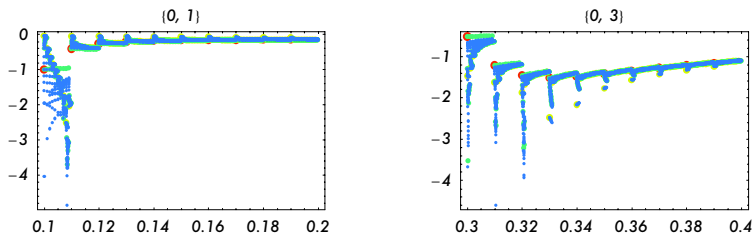
Here we show the differences for the simple and nonsimple continued fractions for sequences of length 2 to 5. We color the differences according to the original sequence length.

```
In[32]:= Show[GraphicsArray[
  Graphics[{Table[{Hue[(n - 1) / 5], PointSize[(18 - 3 n) 0.002],
    Point /@ sequencesAndδs[n, #, 10]}, {n, 1, 4}]],
  PlotLabel → #, PlotRange → All, Frame → True] & /@ {δCF, δNSCF}]]
```



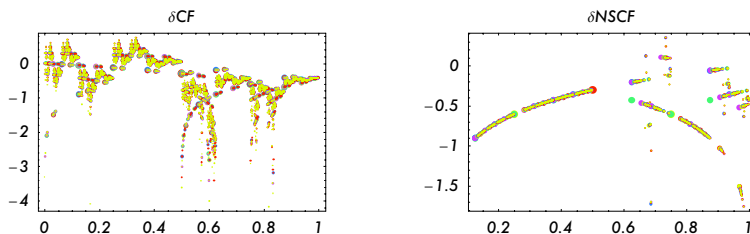
The second graphic shows the partially periodic, fractal-like behavior of the differences. For these short sequences, we can obtain a few coinciding digits for 0.1... and 0.3... for simple continued fractions. The next graphics zoom into these areas.

```
In[33]:= Show[GraphicsArray[Graphics[
  {Table[{Hue[(n - 1) / 5], PointSize[(18 - 3 n) 0.002],
    Point /@ sequencesAndδs[n, δCF, 10, #]}, {n, 1, 4}]],
  PlotLabel → #, PlotRange → All, Frame → True] & /@ {{0, 1}, {0, 3}}]]
```



Here we show base 2 using longer sequences, all sequences up to length 12. Qualitatively we obtain similar graphics.

```
In[34]:= Show[GraphicsArray[
  Graphics[{Table[{Hue[(n - 1) / 5], PointSize[(15 - n) 0.002],
    Point /@ sequencesAndδs[n, #, 2]}, {n, 1, 12}]],
  PlotLabel → #, PlotRange → All, Frame → True] & /@ {δCF, δNSCF}]]
```



■ Search Function for the Constants

In this section we implement the main search function. Some of the searches we carry out run for many hours or even days. So, to see some progress, we define a function, `printProgressCell`, that prints the current state of the search.

```
In[35]:= printProgressCell[
  {bestSequence_, bag_, iterationCounter_, base_, poolSize_,
   commonDigits_, numberOfGroups_, differentDigitPosition_} :=
  CellPrint[Cell[TextData[Cell[BoxData[
    FormBox[GridBox[{{Cell["Max number of agreeing digits:"],
      ToString[commonDigits]}, {Cell["Iteration number:"],
      ToString[iterationCounter]}, {Cell["Base:"],
      ToString[base]}, {Cell["Current sequence length:"],
      ToString[Length[bestSequence]]},
    {Cell["Current pool size:"], ToString[Length[bag]]},
    {Cell["Maximal pool size:"], ToString[poolSize]},
    {Cell["Number of different groups:"],
      ToString[numberOfGroups]},
    {Cell["Differing digit position:"],
      ToString[differentDigitPosition]},
    {Cell["Time:"], ToString[Round[Take[Date[], -3]]]},
    {Cell["Current memory usage:"],
      ToString[Round[MemoryInUse[] / 10. ^6] <> " MB"}],
    {Cell["Maximal memory usage:"],
      ToString[Round[MaxMemoryUsed[] / 10. ^6] <> "MB"}]],
    TraditionalForm]], FontColor → RGBColor[0, 0, 1],
    GridBoxOptions → {RowSpacings → 0,
      ColumnAlignments → {Left}}]], "Print"]]
```

The function `SearchForTrottConstants` searches for a Trott constant in base *base* starting with the initial sequences *initialSequences*. Options for this function are: `RecordSearchHistory`, which determines if the maximal number of coinciding digits as a function of the iteration number should be recorded through the function `searchHistoryData`; `AllowZeroDigits`, which determines if 0 is allowed as a digit; `PrintSearchProgress`, which determines if cells showing the search progress should be printed; `MaxIterations`, which determines the maximal number of iterations; and `MinDigitsWanted`, which determines the minimum number of digits that agree that are wanted.

```
In[36]:= Options[SearchForTrottConstants] = {RecordSearchHistory → False,
  AllowZeroDigits → True, PrintSearchProgress → False,
  MaxIterations → Infinity, MinDigitsWanted → Infinity};
```

```
In[37]:= SearchForTrottConstants::noDigitGain =
  "Exit search because no further digits were gained.";
```

Here is the search function `SearchForTrottConstants`. The method is quite simple. Starting with the pool of given initial sequences *initialSequences*, we repeatedly add *m* base *b* digits, evaluate their quality (which means the number of coinciding digits in the two representations), and keep the best *poolSize* sequences in the pool. We repeat this process until either the best sequence no longer

shows improvements or the specified number of digits that agree has been achieved. The comments in the code explain the individual steps.

```

In[38]:= SearchForTrottConstants[kind : ("CF" | "NSCF"),
  base : (b_Integer), poolSize_Integer, m_Integer : 2,
  initialSequences_List : {{0}}, opts___Rule] :=
Module[{{δF, bag, iterationCounter, printSearchProgressQ,
  allowZeroDigitsQ, recordSearchHistoryQ, maxIterations,
  minDigitsWanted, bestSequence, commonDigits, fP, numberOfGroups,
  differentDigitPosition, mini j, newAll, newData},
Clear[searchHistoryData];
(* measuring the difference *)
δF = If[kind === "CF", δCF[#, base] &, δNSCF[#, base] &];
bag = initialSequences;
iterationCounter = 0;
Off[Power::infy]; Off[Infinity::indet];
(* process options *)
printSearchProgressQ =
  PrintSearchProgress /. {opts} /. PrintSearchProgress → False;
allowZeroDigitsQ = AllowZeroDigits /. {opts} /.
  AllowZeroDigits → True;
recordSearchHistoryQ = RecordSearchHistory /. {opts} /.
  RecordSearchHistory → True;
maxIterations = MaxIterations /. {opts} /. MaxIterations → Infinity;
minDigitsWanted =
  MinDigitsWanted /. {opts} /. MinDigitsWanted → Infinity;
(* iterate digit addition and selection of best sequences *)
While[
  iterationCounter < maxIterations && If[iterationCounter == 0, True,
    (* check best candidate for incremental improvements *)
    bestSequence = bag[[1]];
    commonDigits =
      numberOfCoincidingDigits[kind, bestSequence, base] + 1;
    fP[iterationCounter] = commonDigits;
    (* potentially record search progress *)
    numberOfGroups = With[{λ = Min[Length[bestSequence], 6]},
      Length[Union[Take[#, λ] & /@ bag]];
    differentDigitPosition = If[# == {}, 0, #[[1, 1]]] &[
      Position[Equal@@@Transpose[bag], False, {1}, 1]];
    If[recordSearchHistoryQ, searchHistoryData[iterationCounter] =
      {iterationCounter, commonDigits,
        numberOfGroups, differentDigitPosition}];
    (* potentially print search progress *)
    If[printSearchProgressQ, printProgressCell[
      {bestSequence, bag, iterationCounter, base, poolSize,
        commonDigits, numberOfGroups, differentDigitPosition}]];
    (* check termination conditions *)
    If[commonDigits > minDigitsWanted, False,
      If[iterationCounter < 20, True, If[fP[iterationCounter] ===
        fP[iterationCounter - 10], If[printSearchProgressQ, Message[
          SearchForTrottConstants::noDigitGain]]; False, True]]],

```

```

(* add new digits at the end of the existing sequences *)
iterationCounter++;
(* add digit tuples *)
minij = If[allowZeroDigitsQ, 0, 1];
Developer`ClearCache[];
(* avoid building up long lists with all existing digits *)
newAll = Flatten[Table[If[allowZeroDigitsQ, Rest, Identity]@
  Flatten[Table[{{#F[Join[bag[[k]], Table[i[j], {j, m}]]]},
    {k, Table[i[j], {j, m}]}},
    Evaluate[Sequence@@Table[{i[j], minij, base - 1}, {j, m}]]],
  m - 1], {k, Length[bag]}], 1];
(* form new pool *)
newData =
  Last /@ Take[Sort[newAll], Min[poolSize, Length[newAll]]];
bag = Join[bag[[#1]], #2] & @@@newData;
(* return best digit sequence *)First[bag]

```

This example shows SearchForTrottConstants during the search.

```

In[39]:= SearchForTrottConstants["NSCF", 10, 50,
  PrintSearchProgress -> True, MinDigitsWanted -> 10]

```

Max number of agreeing digits:	4
Iteration number:	1
Base:	10
Current sequence length:	3
Current pool size:	50
Maximal pool size:	50
Number of different groups:	50
Differing digit position:	2
Time:	{16, 41, 16}
Current memory usage:	9 MB
Maximal memory usage:	10 MB

Max number of agreeing digits:	5
Iteration number:	2
Base:	10
Current sequence length:	5
Current pool size:	50
Maximal pool size:	50
Number of different groups:	50
Differing digit position:	2
Time:	{16, 41, 19}
Current memory usage:	10 MB
Maximal memory usage:	10 MB

Max number of agreeing digits:	8
Iteration number:	3
Base:	10
Current sequence length:	7
Current pool size:	50
Maximal pool size:	50
Number of different groups:	29
Differing digit position:	2
Time:	{16, 41, 22}
Current memory usage:	10 MB
Maximal memory usage:	11 MB

Max number of agreeing digits:	8
Iteration number:	4
Base:	10
Current sequence length:	9
Current pool size:	50
Maximal pool size:	50
Number of different groups:	13
Differing digit position:	2
Time:	{16, 41, 26}
Current memory usage:	10 MB
Maximal memory usage:	11 MB

Max number of agreeing digits:	10
Iteration number:	5
Base:	10
Current sequence length:	11
Current pool size:	50
Maximal pool size:	50
Number of different groups:	7
Differing digit position:	2
Time:	{16, 41, 29}
Current memory usage:	10 MB
Maximal memory usage:	11 MB

Max number of agreeing digits:	12
Iteration number:	6
Base:	10
Current sequence length:	13
Current pool size:	50
Maximal pool size:	50
Number of different groups:	6
Differing digit position:	2
Time:	{16, 41, 33}
Current memory usage:	10 MB
Maximal memory usage:	11 MB

```
Out[39]= {0, 1, 6, 1, 5, 2, 8, 2, 5, 3, 5, 3, 8}
```

```
In[40]:= radixCFDifference["NSCF", %, 10]
```

```
Out[40]= 1.8769434928864611301 × 10-11
```

```
In[41]:= numberOfCoincidingDigits["NSCF", %, 10]
```

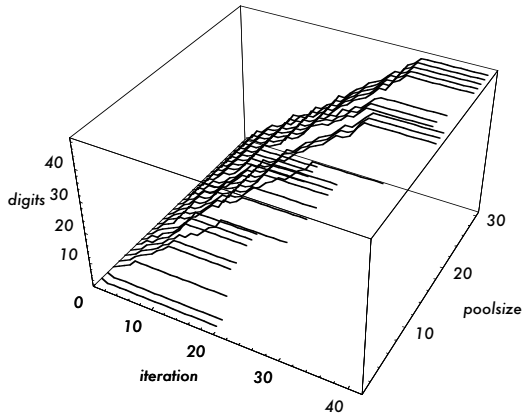
```
Out[41]= 11
```

■ Carrying Out and Monitoring Searches

A practical, relevant question for carrying out searches concerns the dependence on the pool size. This simple example shows a trend that is also visible in much larger pool sizes—a high sensitivity to the actual pool size. This sensitivity occurs because a sequence that at iteration i shows a relatively poor performance might show a much better performance at iteration $i + k$. When the pool size is not large enough, this sequence might be discarded too early. In general, the larger the pool size the better the results. But the larger the pool size, the longer the search will run and with pool sizes too large, we might run out of memory.

```
In[42]:= dataNSCF10 = Table[
  SearchForTrottConstants["NSCF",
    10, poolSize, {{0}}, PrintSearchProgress → False,
    RecordSearchHistory → True, MaxIterations → 50];
  Line[{{#1, poolSize, #2} & @@@
    (Last /@ DownValues[searchHistoryData])], {poolSize, 1, 30}];
```

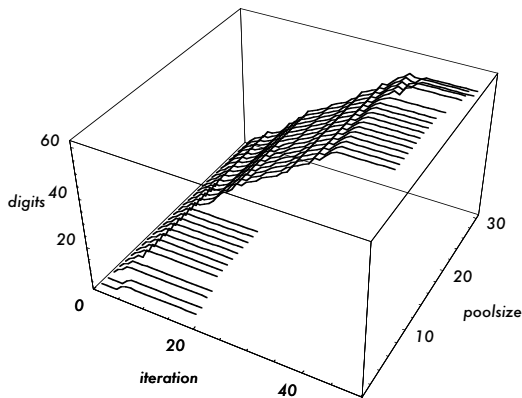
```
In[43]:= Show[Graphics3D[dataNSCF10],
  PlotRange → All, Axes → True, BoxRatios → {1, 1, 0.6},
  AxesLabel → {"iteration", "poolsize", "digits"}]
```



For nonsimple continued fractions we see a similar, non-monotonic dependence on the pool size.

```
In[44]:= dataCF10 = Table[
  SearchForTrottConstants["CF",
    10, poolSize, {{0}}, PrintSearchProgress → False,
    RecordSearchHistory → True, MaxIterations → 50];
  Line[{{#1, poolSize, #2} &@@@
    (Last /@ DownValues[searchHistoryData])], {poolSize, 1, 30}];
```

```
In[45]:= Show[Graphics3D[dataCF10], PlotRange → All,
  Axes → True, BoxRatios → {1, 1, 0.6},
  AxesLabel → {"iteration", "poolsize", "digits"}]
```



Here a larger search is carried out. We do not restrict the possible initial sequences and use a pool size of 1000.

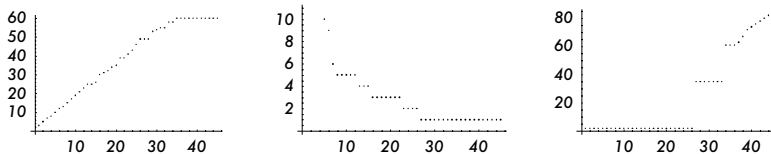
```
In[46]:= SearchForTrottConstants["CF", 10, 1000, {0}],
          PrintSearchProgress → False, RecordSearchHistory → True] // Timing
Out[46]= {871.14 Second, {0, 1, 0, 8, 4, 2, 5, 0, 6, 0, 5, 3, 2, 7, 4, 2, 4, 7, 1, 3,
          9, 0, 5, 1, 3, 1, 3, 4, 3, 3, 0, 9, 3, 1, 7, 0, 5, 6, 4, 1, 9, 1, 2, 2,
          7, 9, 0, 9, 1, 0, 7, 3, 3, 5, 3, 1, 7, 8, 1, 4, 9, 0, 9, 0, 2, 2, 9, 0,
          1, 9, 1, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0}}

In[47]:= shortenedSequence["CF", %[[2]], 10]
Out[47]= {0, 1, 0, 8, 4, 2, 5, 0, 6, 0, 5, 3, 2, 7, 4, 2, 4, 7, 1, 3, 9, 0, 5, 1,
          3, 1, 3, 4, 3, 3, 0, 9, 3, 1, 7, 0, 5, 6, 4, 1, 9, 1, 2, 2, 7, 9, 0,
          9, 1, 0, 7, 3, 3, 5, 3, 1, 7, 8, 1, 4, 9, 0, 9, 0, 2, 2, 9, 0, 1, 9}

In[48]:= {radixCFDifference["CF", %, 10],
          numberOfCoincidingDigits["CF", %, 10]}
Out[48]= {3.3325299977178384053 × 10-59, 59}
```

We achieved about 60 coinciding digits in about 15 minutes. The next three graphics show the number of coinciding digits of the optimal sequence, the number of different groups of sequences with different initial six digits, and the position of the first different digit across the pool of sequences as a function of the iteration number. We see that on average we gain a constant number of digits per iteration as long as we gain digits. We also see how the numbers in the pool converge towards a single number.

```
In[49]:= Show[Block[{data = Last /@ DownValues[searchHistoryData],
          $DisplayFunction = Identity}, GraphicsArray[
          Function[k, ListPlot[{{#1, Slot[k]} & @@@ data}] /@ {2, 3, 4}]]]
```



And here is the equivalent of the last calculation for nonsimple continued fractions. Again we use a pool size of 1000. This time, we obtain more than 100 identical digits. And again we see on average a linear increase in the digits gained as well as convergence on one final number (after about 40 iterations).


```

1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0}

```

Out[54]= {158, 1.75218×10^{-48} }

```

In[55]:= {numberOfCoincidingDigits["CF", #, 10],
  radixCFDifference["CF", #, 10] // N} & @
{0, 1, 0, 1, 0, 7, 1, 8, 2, 3, 7, 0, 8, 6, 6, 6, 1, 4, 4, 4, 1, 1,
  8, 1, 6, 1, 2, 5, 3, 1, 4, 0, 9, 6, 1, 3, 4, 0, 1, 6, 4, 1, 8,
  1, 8, 3, 4, 0, 9, 3, 5, 3, 3, 8, 6, 9, 1, 6, 0, 9, 9, 2, 2, 1, 9,
  1, 3, 1, 4, 1, 1, 3, 6, 1, 2, 4, 1, 1, 4, 1, 5, 0, 3, 1, 4, 6, 2,
  6, 1, 6, 5, 2, 2, 0, 6, 4, 2, 1, 4, 1, 9, 1, 2, 9, 4, 6, 1, 3, 8,
  3, 0, 5, 0, 6, 2, 4, 1, 1, 5, 5, 9, 5, 2, 0, 1, 9, 1, 1, 9, 3, 6,
  1, 7, 2, 5, 9, 5, 4, 0, 3, 4, 9, 0, 1, 7, 4, 2, 2, 2, 2, 8, 1, 6,
  1, 4, 3, 3, 7, 0, 7, 6, 2, 6, 4, 3, 4, 0, 8, 2, 5, 1, 5, 8, 1, 3,
  9, 0, 2, 3, 8, 7, 2, 1, 1, 1, 5, 1, 2, 3, 0, 9, 1, 6, 7, 0, 5, 2,
  6, 1, 9, 0, 6, 8, 1, 1, 5, 2, 2, 8, 0, 6, 5, 9, 7, 4, 4, 6, 2, 9,
  0, 1, 1, 1, 1, 2, 2, 1, 1, 9, 0, 2, 6, 1, 1, 1, 8, 4, 9, 0, 4, 1,
  9, 1, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0, 9, 0}

```

Out[55]= {217, 1.11976×10^{-217} }

```

In[56]:= {numberOfCoincidingDigits["NSCF", #, 10],
  radixCFDifference["NSCF", #, 10] // N} & @
{0, 4, 5, 9, 2, 4, 9, 2, 5, 1, 8, 2, 9, 3, 7, 3, 8, 1, 7, 2, 5, 4, 7,
  3, 4, 1, 9, 2, 9, 1, 8, 2, 3, 1, 7, 2, 4, 4, 5, 1, 8, 3, 8, 1, 8, 2, 8,
  1, 7, 9, 8, 4, 7, 2, 9, 1, 5, 2, 5, 2, 6, 9, 9, 1, 9, 1, 9, 2, 8, 1, 7,
  7, 8, 1, 7, 3, 9, 1, 6, 4, 4, 1, 8, 2, 6, 2, 8, 1, 7, 1, 7, 8, 8, 2, 8,
  1, 9, 7, 8, 5, 6, 2, 7, 4, 8, 1, 8, 1, 5, 5, 6, 1, 7, 6, 7, 1, 9, 6, 9,
  1, 9, 5, 6, 1, 9, 2, 8, 5, 8, 3, 9, 6, 9, 1, 7, 2, 9, 5, 9, 6, 9, 9, 9,
  1, 8, 5, 9, 1, 8, 1, 9, 3, 7, 3, 8, 2, 9, 2, 8, 2, 6, 7, 7, 2, 6, 1, 6,
  2, 7, 9, 8, 5, 8, 3, 5, 1, 7, 3, 8, 1, 6, 3, 9, 1, 9, 9, 8, 1, 9, 9, 2,
  7, 7, 2, 4, 2, 8, 1, 6, 1, 7, 6, 7, 1, 9, 1, 6, 1, 5, 1, 3, 8, 9, 1, 7,
  5, 5, 1, 8, 5, 7, 7, 7, 1, 9, 1, 7, 3, 9, 1, 9, 8, 4, 1, 7, 4, 7, 3, 7,
  1, 9, 2, 8, 2, 8, 5, 5, 3, 9, 2, 8, 6, 7, 1, 9, 3, 6, 2, 7, 4, 5, 1, 7,
  2, 5, 1, 7, 1, 7, 2, 7, 7, 6, 3, 6, 2, 9, 1, 6, 1, 5, 4, 9, 8, 9, 1,
  7, 9, 9, 2, 8, 2, 5, 1, 9, 3, 9, 4, 6, 9, 8, 1, 9, 3, 8, 2, 9, 5, 8,
  2, 7, 2, 7, 8, 8, 7, 8, 4, 8, 1, 8, 6, 7, 1, 9, 5, 9, 5, 7, 1, 9, 4,
  5, 3, 9, 5, 9, 2, 8, 5, 8, 6, 7, 1, 7, 1, 4, 5, 8, 1, 9, 1, 6, 5, 5, 1,
  8, 1, 9, 1, 6, 4, 7, 1, 9, 1, 7, 5, 5, 2, 9, 3, 9, 2, 9, 1, 6, 4, 9, 2,
  7, 5, 8, 5, 8, 1, 9, 5, 6, 5, 8, 1, 7, 2, 9, 7, 3, 1, 9, 4, 9, 4, 8,
  1, 4, 1, 6, 1, 9, 2, 9, 2, 8, 3, 8, 2, 8, 5, 7, 7, 8, 3, 9, 1, 9, 7, 8,
  1, 9, 3, 6, 2, 9, 2, 9, 5, 9, 2, 9, 6, 5, 1, 6, 4, 7, 4, 9, 2, 9, 9,
  8, 1, 9, 4, 9, 5, 7, 4, 7, 2, 7, 8, 9, 1, 9, 2, 6, 3, 9, 1, 9, 2, 9, 7,
  9, 7, 5, 1, 5, 4, 9, 1, 7, 7, 8, 1, 3, 1, 7, 1, 5, 4, 5, 3, 8, 1, 5,
  1, 9, 4, 9, 1, 7, 8, 4, 1, 9, 1, 9, 7, 8, 5, 9, 2, 7, 5, 9, 3, 7, 1, 8,
  1, 9, 8, 9, 8, 8, 1, 9, 1, 9, 1, 6, 8, 6, 1, 8, 1, 8, 9, 4, 1, 5, 2,
  6, 1, 7, 2, 9, 8, 9, 2, 6, 1, 9, 2, 7, 3, 7, 1, 6, 8, 7, 3, 8, 1, 7, 1,

```

```

9, 2, 9, 6, 5, 2, 9, 2, 5, 6, 7, 1, 8, 4, 8, 3, 8, 1, 4, 1, 7, 7, 8,
1, 8, 3, 8, 1, 6, 2, 9, 1, 5, 1, 5, 8, 9, 6, 8, 4, 9, 1, 9, 5, 4, 1,
4, 1, 4, 0, 6, 2, 5, 0, 1, 0, 1, 0, 1, 0, 1, 7, 0, 5, 6, 2, 5, 6, 0}

```

```
Out[56]= {433, 4.186720568476718 × 10-433}
```

```

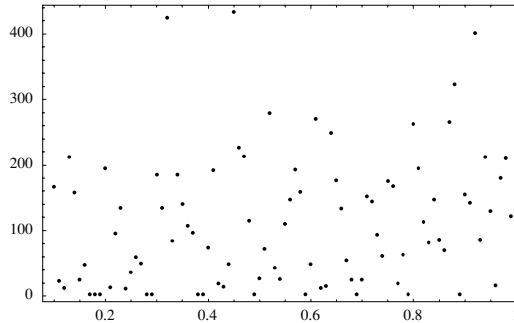
In[57]:= {numberOfCoincidingDigits["NSCF", #, 2],
          radixCFDifference["NSCF", #, 2] // N} & @
{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1}

```

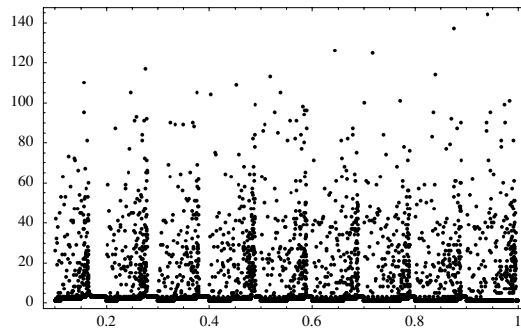
```
Out[57]= {26, 1.92473 × 10-8}
```

Digit Gains with Seeded Sequences

Here are the results for the number of achieved coinciding digits from running `SearchForTrottConstants["NSCF", 10, 103, {{0, i, j}}` for all 90 possible combinations of *i* and *j*. No obvious pattern for optimal starting digits is visible.



These are the results for the number of achieved coinciding digits from `SearchForTrottConstants["NSCF", 10, 102, {{0, i, j, k, l}}` for all 9000 possible combinations of *i*, *j*, *k*, and *l*. The pattern suggests that a larger number of agreeing digits is achieved periodically.



■ Why Searches Are Difficult

Lochs' theorem [2] states that to represent a decimal number with n digits, we need about $0.97n$ simple continued fraction terms. This means that when adding digits to our lists, the continued fraction digits and the radix digits get slowly out of sync. To ensure that the digits agree with each other, we need a pool size that becomes progressively larger.

Lochs' theorem holds in its classic form only for simple continued fractions. But experience shows that nonsimple continued fractions behave similarly. Let us analyze the best Trott constant found with more than 400 coinciding digits.

```
In[58]:= winnerListNSCF10 =
  {0, 4, 5, 9, 2, 4, 9, 2, 5, 1, 8, 2, 9, 3, 7, 3, 8, 1, 7, 2, 5, 4, 7,
    3, 4, 1, 9, 2, 9, 1, 8, 2, 3, 1, 7, 2, 4, 4, 5, 1, 8, 3, 8, 1, 8, 2,
    8, 1, 7, 9, 8, 4, 7, 2, 9, 1, 5, 2, 5, 2, 6, 9, 9, 1, 9, 1, 9, 2, 8,
    1, 7, 7, 8, 1, 7, 3, 9, 1, 6, 4, 4, 1, 8, 2, 6, 2, 8, 1, 7, 1, 7, 8,
    8, 2, 8, 1, 9, 7, 8, 5, 6, 2, 7, 4, 8, 1, 8, 1, 5, 5, 6, 1, 7, 6, 7,
    1, 9, 6, 9, 1, 9, 5, 6, 1, 9, 2, 8, 5, 8, 3, 9, 6, 9, 1, 7, 2, 9, 5,
    9, 6, 9, 9, 9, 1, 8, 5, 9, 1, 8, 1, 9, 3, 7, 3, 8, 2, 9, 2, 8, 2, 6,
    7, 7, 2, 6, 1, 6, 2, 7, 9, 8, 5, 8, 3, 5, 1, 7, 3, 8, 1, 6, 3, 9, 1,
    9, 9, 8, 1, 9, 9, 2, 7, 7, 2, 4, 2, 8, 1, 6, 1, 7, 6, 7, 1, 9, 1, 6,
    1, 5, 1, 3, 8, 9, 1, 7, 5, 5, 1, 8, 5, 7, 7, 7, 1, 9, 1, 7, 3, 9, 1,
    9, 8, 4, 1, 7, 4, 7, 3, 7, 1, 9, 2, 8, 2, 8, 5, 5, 3, 9, 2, 8, 6, 7,
    1, 9, 3, 6, 2, 7, 4, 5, 1, 7, 2, 5, 1, 7, 1, 7, 2, 7, 7, 6, 3, 6, 2,
    9, 1, 6, 1, 5, 4, 9, 8, 9, 1, 7, 9, 9, 2, 8, 2, 5, 1, 9, 3, 9, 4, 6,
    9, 8, 1, 9, 3, 8, 2, 9, 5, 8, 2, 7, 2, 7, 8, 8, 7, 8, 4, 8, 1, 8, 6,
    7, 1, 9, 5, 9, 5, 7, 1, 9, 4, 5, 3, 9, 5, 9, 2, 8, 5, 8, 6, 7, 1, 7,
    1, 4, 5, 8, 1, 9, 1, 6, 5, 5, 1, 8, 1, 9, 1, 6, 4, 7, 1, 9, 1, 7, 5,
    5, 2, 9, 3, 9, 2, 9, 1, 6, 4, 9, 2, 7, 5, 8, 5, 8, 1, 9, 5, 6, 5, 8,
    1, 7, 2, 9, 7, 3, 1, 9, 4, 9, 4, 8, 1, 4, 1, 6, 1, 9, 2, 9, 2, 8, 3,
    8, 2, 8, 5, 7, 7, 8, 3, 9, 1, 9, 7, 8, 1, 9, 3, 6, 2, 9, 2, 9, 5, 9,
    2, 9, 6, 5, 1, 6, 4, 7, 4, 9, 2, 9, 9, 8, 1, 9, 4, 9, 5, 7, 4, 7, 2,
    7, 8, 9, 1, 9, 2, 6, 3, 9, 1, 9, 2, 9, 7, 9, 7, 5, 1, 5, 4, 9, 1, 7, 8, 4,
    7, 8, 1, 3, 1, 7, 1, 5, 4, 5, 3, 8, 1, 5, 1, 9, 4, 9, 1, 7, 8, 4,
    1, 9, 1, 9, 7, 8, 5, 9, 2, 7, 5, 9, 3, 7, 1, 8, 1, 9, 8, 9, 8, 8,
    1, 9, 1, 9, 1, 6, 8, 6, 1, 8, 1, 8, 9, 4, 1, 5, 2, 6, 1, 7, 2, 9,
    8, 9, 2, 6, 1, 9, 2, 7, 3, 7, 1, 6, 8, 7, 3, 8, 1, 7, 1, 9, 2, 9,
    6, 5, 2, 9, 2, 5, 6, 7, 1, 8, 4, 8, 3, 8, 1, 4, 1, 7, 7, 8, 1, 8,
    3, 8, 1, 6, 2, 9, 1, 5, 1, 5, 8, 9, 6, 8, 4, 9, 1, 9, 5, 4, 1, 4,
    1, 4, 0, 6, 2, 5, 0, 1, 0, 1, 0, 1, 0, 1, 7, 0, 5, 6, 2, 5, 6, 0};
```

On average we obtain about 0.7 decimal digits per continued fraction digit.

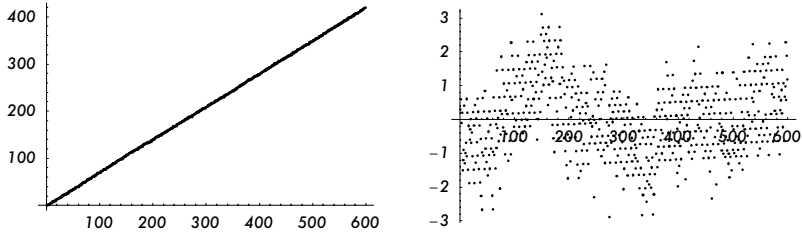
```
In[59]:= {numberOfCoincidingDigits["NSCF", #, 10],
  radixCFDifference["NSCF", #, 10]} & @
  winnerListNSCF10
Out[59]:= {433, 4.1867205684767181422 × 10-433}
In[60]:= gainDataD = Table[{k, numberOfCoincidingDigits[
  "NSCF", Take[winnerListNSCF10, k], 10]}, {k, 2, 600}];
```

```
In[61]:= fitWD[x_] = Fit[ gainDataD, {1, x}, x]
```

```
Out[61]= -1.6906 + 0.699198 x
```

The following graphics show the digit gain and the difference of the digit gain and its linear part.

```
In[62]:= Show[GraphicsArray[{ListPlot[ gainDataD ],
    ListPlot[{#1, #2 - fitWD[#1]} & @@@ gainDataD]}]]
```



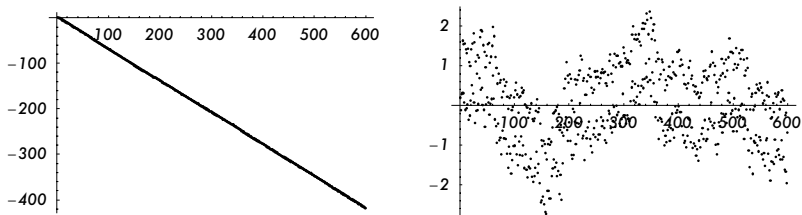
The absolute difference shows similar behavior, though different in detail.

```
In[63]:= gainDataP = Table[{k, Log[10, Abs[radixCFDifference[
    "NSCF", Take[winnerListNSCF10, k], 10]]]}, {k, 2, 600}];
```

```
In[64]:= fitWP[x_] = Fit[ gainDataP, {1, x}, x]
```

```
Out[64]= 1.7635240683987384 - 0.6991036388280274931 x
```

```
In[65]:= Show[GraphicsArray[{ListPlot[ gainDataP ],
    ListPlot[{#1, #2 - fitWP[#1]} & @@@ gainDataP]}]]
```



■ Summary

In this Corner, we presented a *Mathematica* function to search for Trott constants. The numerical results indicate that such constants exist. But no proof of existence is available. Do such constants exist in any integer base? If such constants exist, how many are there? A finite number or an infinite number? The preliminary numerical results presented here do not allow conjectures. We hope extended numerical searches will shed some light on this subject.

■ References

- [1] Eric W. Weisstein, "Trott's Constant" from *MathWorld*—A Wolfram Web Resource. mathworld.wolfram.com/TrottsConstant.html.
- [2] Eric W. Weisstein, "Lochs' Theorem" from *MathWorld*—A Wolfram Web Resource. mathworld.wolfram.com/LochsTheorem.html.

■ Additional Material

AdditionalConstants.nb

Available at www.mathematica-journal.com/issue/v10i2/download.

Michael Trott

Senior Member, Technical Staff
Wolfram Research, Inc.
mtrott@wolfram.com