Number Representation

Eric Hehner

It is no exaggeration to say that our usual representation of natural numbers as a sequence of digits, for example 123 (one hundred and twenty-three), is one of humankind's greatest inventions. This essay discusses three enhancements that make the representation more useful: the point (.), the rep (·), and the quote (').

Point

Points are in common use; the point is placed somewhere in the sequence of digits. For example, 12.34 . In many parts of the world, a comma is used instead of a point, but the idea is the same. If the base of the number representation is ten, the digits are called decimal digits, and the point is called a decimal point; if the base of the number representation is two, the digits are called binary digits, and the point is called a binary point; and similarly for other bases. Placing the point one digit to the left of the right end is division by 10 . Placing it two digits to the left of the right end is division by 100 . And so on. For example, $12.34 = 1234 / 100$. A point can be at the left end of the digits, or between two digits, or at the right end of the digits. When it is at the right end, it is division by 1 , which leaves the number unchanged. For example, $123. = 123 / 1 = 123$. For the sake of uniformity, we may say that there is always a point, but when it is at the right end, we may not bother to write it (it is there but invisible). Moving the point left one place is division by 10 ; moving it right one place is multiplication times 10 . We can even move the point left when it is at the left end because there are invisible 0s to the left of the left end; they become visible when the point is moved past them. For example, $.123 / 10 = .0123$. Likewise a point can be moved right even when it is at the right end because there are invisible 0s to the right of the right end. For example, $123. \times 10 = 1230.$.

A point enables us to represent those nonnegative rational numbers that are the result of dividing a natural number (0, 1, 2, and so on) by a power of the base (1, 10, 100, and so on).

Rep

There are several notations in use that are equivalent to the rep. One is to write a line over top of some of the rightmost digits. For example, $12.3\overline{45}$. This is a finite way of writing an infinite sequence of digits $12.34545454545\rightarrow$. (The $\rightarrow$ means "and so on forever". It is not part of the number representation. It is part of the explanation of the number representation, just like these words.) But a line over top of some digits is typographically inconvenient, so a rep (raised dot) is used instead: $12.3{\cdot}45$. The digits to the right of the rep are understood to be repeated forever. The rep can come before the point. For example, $1{\cdot}2.3 = 12.323232323\rightarrow$. The rep can be in the same position as the point. For example, $12{:}34 = 12.3434343434\rightarrow$. The rep can be at the left end or the right end. When a rep is at the right end, the invisible 0s to the right of the right end are repeated. For the sake of uniformity, we may say that there is always a rep, but when it is at the right end, we may not bother to write it (it is there but invisible). For example,

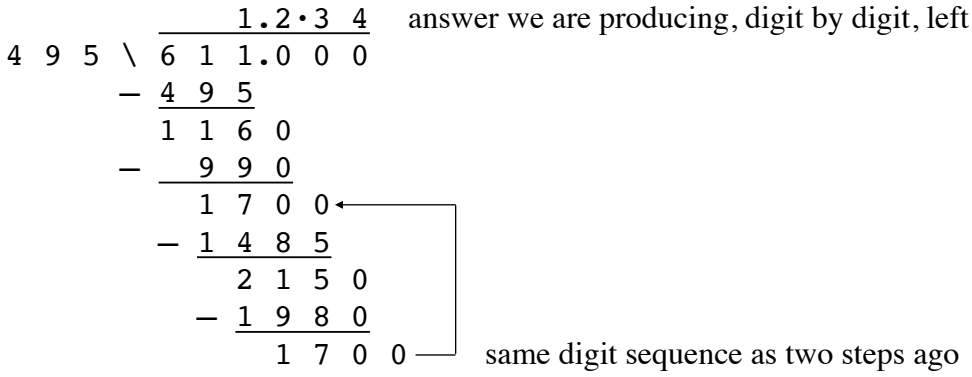
$$1.2 = 1.2{\cdot} = 1.2{\cdot}0 = 1.20000000\rightarrow$$

If there is a rep but no point, the point is invisibly at the right end of the visible digits. For example,

$$1{\cdot}2 = 1{\cdot}2. = 12.2222222222\rightarrow$$

Using a sequence of digits with a point and a rep, we can represent all and only the nonnegative rational numbers. It is therefore impossible to perform a negation operation with only a point and rep. The addition, subtraction, and multiplication operations all start at the right side of their operands, and work from right to left. But a rep indicates an infinite sequence of digits with no right end, making these operations difficult in general.

A sequence of digits with a rep is produced by the division operation that was taught in schools for centuries prior to the widespread availability of electronic calculators. As an example, we can do the division 611 / 495 . To the right of the right end of each operand there is an invisible point, and an invisible rep, and invisible 0s, which we will need to make visible one by one as we proceed through the division. We repeatedly guess the next digit in the answer, multiply the divisor 495 times that guess, and reduce the dividend by that amount. The guess must be the largest digit such that the divisor times the guess is less than or equal to the current dividend. Division is a sequence of guesses, multiplications, and subtractions. When the current dividend is the same digit sequence as a previous dividend, we know where to insert the rep, and we are done.



Quote

A quote is just like a rep but in the opposite direction. For example, 12'34 says that the digits 12 are repeated forever to the left: 12'34 = ←1212121234 (using ← to mean "and so back"; it is not part of the number representation). The quote can come at the left end, between two digits, or at the right end. When a quote is at the left end, the invisible 0s to the left of the left end are repeated. For the sake of uniformity, we may say that there is always a quote, but when it is at the left end, we may not bother to write it (it is there but invisible). For example,

123 = '123 = 0'123 = ←000000123

So the natural numbers look familiar: 0 , 1 , 2 , and so on.

Using a quote, negative integers will look familiar if you know radix complement arithmetic (in base 2 it's 2's complement; this is 10's complement). To negate, take the complement of each digit (0 becomes 9; 1 becomes 8; 2 becomes 7; and so on) and then add 1 to the final digit. So 0'123 becomes 9'876 and then add 1 to get 9'877 . To negate 9'877 , complement each digit and get 0'122 and then add 1 to get 0'123 .

–123 = 9'877

–9'877 = 123

Addition works the same for numbers with a quote as it does for numbers without a quote. For example, we will add –123 + 123 and get 0 (the 1's in the middle row are the carries).

```
   9'8 7 7
   1 1 1
+ 0'1 2 3
= 0'0 0 0
```

For another example, we add $-123 + 456$ and get 333.

```
   9'8 7 7
   1 1 1
+  0'4 5 6
=  0'3 3 3
```

Subtraction works as follows.

```
   2 7 0 0 4
     1 1 1
−  0 3 5 2 6
=  2 3 4 7 8
```

In the rightmost column, to subtract 6 from 4, we do not borrow from the 0 to the left of the 4; instead we carry 1 to the 2 to the left of the 6. The carry makes the 4 into 14 so we subtract 6 from 14 and get 8. In the next column, the carry is added to the 2, so we are subtracting 3 from 0. And so on. With this kind of subtraction we can subtract $0'123$ from $0'$ and get $9'877$.

```
   0'0 0 0
   1 1 1
−  0'1 2 3
=  9'8 7 7
```

Multiplication with a quote is exactly the same as multiplication without a quote. For example, we multiply -2×13 and get -26.

```
     9'8
   × 1 3
     9'4
   9'8
= 9'7 4
```

As noted previously, a sequence of digits with a rep is produced by the usual division operation, which we will now call rep division; it produces digits from left to right. A sequence of digits with a quote is produced by a division operation that works in the opposite direction, from right to left, which we call quote division. For example, let us evaluate $191 / 33$ using quote division. The dividend is 191 and its rightmost digit is 1. The divisor is 33, and its rightmost digit is 3. We need to multiply the divisor times a digit d such that $d \times 3$ ends in 1. So the rightmost digit of the answer is 7. Now $7 \times 33 = 231$, so that's what we subtract from the dividend. Delete the rightmost 0 (which divides by 10), giving a new dividend of $9'6$. Repeat, until we get a dividend that we have seen before. That tells us where to put the quote, and we're done.

```
    1 2'7          answer we are producing, digit by digit, right to left
    1 9 1 / 3 3    dividend / divisor
−   2 3 1          minus divisor times 7
    9'6 ←          difference, delete rightmost 0, new dividend
−     6 6          minus divisor times 2
    9'3            difference, delete rightmost 0, new dividend
−     3 3          minus divisor times 1
  9'6 ——           same digit sequence as two steps ago
```

In quote division, each digit is determined by the rightmost digit of the current dividend and the rightmost digit of the divisor; there's no guess as there is in rep division. Unfortunately, this works only when the divisor has no factors in common with the base. In decimal, the divisor's rightmost digit must be 1 , 3 , 7 or 9 . If the divisor's rightmost digit is 0 , just delete it (thus dividing the divisor by 10), and move the point one place to the left in the answer. If the divisor's rightmost digit is 2 , 4 , 6 , or 8 , multiply both the divisor and the dividend times 5 , then delete the divisor's rightmost 0 , and adjust the point in the answer as before. If the divisor's rightmost digit is 5 , multiply both the divisor and the dividend times 2 , then delete the divisor's rightmost 0 and adjust the point in the answer as before. It is annoying to have to get rid of the factors of the base, but once that's done, quote division can proceed without guessing.

We don't have to get rid of factors of the base if we use a prime base. Base 2 is prime, and it makes quote division particularly easy. Any 0s at the right end of the divisor are deleted, and the point in the answer will be moved that many places to the left. If the current dividend's right end is 1 , that's the next bit of the answer, and we subtract the divisor, and delete the difference's rightmost 0 . If the current dividend's right end is 0 , that's the next bit of the answer, and we don't subtract the divisor, and we delete the difference's rightmost 0 . The difference then becomes the current dividend.

The quote allows us to represent both positive and negative numbers. To determine whether a number with a quote is positive or negative, just compare the first digit of the repeating sequence to the left of the quote with the first digit after the quote. If the first digit of the repeating sequence is larger, the number is negative. If the first digit after the quote is larger, the number is positive. If those two digits are equal, either the number is 0 , or the representation can be shortened (see **Duplication** below). For example, 12'345 is positive because 1 is less than 3 . And 9' is negative because the first digit of the repeating part, which is 9 , is larger than the first digit after the quote, which is an invisible 0 . And '9 is positive because the first digit of the repeating part, which is an invisible 0 , is less than the first digit after the quote, which is 9 .

With just a quote, we can represent those rational numbers that are not the result of dividing an integer by a power of the base. As stated earlier, a point enables us to represent those nonnegative rational numbers that are the result of dividing a natural number by a power of the base. With a point and a quote, we can represent all rational numbers. So the point and quote are more expressive (all rationals) than the point and rep (all nonnegative rationals). With a point, a rep, and a quote, we can again represent all rational numbers; we do not gain expressivity over just a point and a quote.

We use a colon (:) for a point and a rep in the same position, an exclamation (!) for a quote and a point in the same position, a raised colon (⁚) for a rep and a quote in the same position, and a triplon (⁝) for all three marks in the same position.

Evaluation

The most commonly used arithmetic operators are negation ($-a$), addition ($a+b$), subtraction ($a-b$), multiplication ($a \times b$), division (a/b), and exponentiation (a^b). Evaluation of an expression that contains these operators means finding an equivalent expression as just a sequence of digits, using a point, rep, or quote as necessary, but not using any of these operators. Other commonly used operators are comparisons ($a<b$, $a>b$, $a \le b$, $a \ge b$, $a=b$, $a \ne b$). The result of a comparison is either *true* or *false* .

123+45 is an unevaluated expression; evaluating it (doing the addition) produces 168 . Likewise –5 is an unevaluated expression; evaluating it (doing the negation) produces 9'5 . Likewise 1/3 is an unevaluated expression; evaluating it (doing the division) produces either :3 or 6'7 , depending on which division algorithm is used. Evaluating –1/3 produces 3' . Evaluating 10^3 produces 1000 . Evaluating 123>45 produces *true* .

If someone asks "What is 123+45 ?", they would not be satisfied to be told "It's 123+45 .", leaving the addition unevaluated. Similarly, if someone asks "What is –5 ?", they should not be satisfied to be told "It's –5 .", leaving the negation unevaluated. But we are so used to seeing a number less than zero written with a negation operator that we may not realize it is unevaluated. Textbooks talk about the "sign-and-magnitude" representation of a number, leaving the wrong impression that the minus-sign is just part of the number representation, not an operator. As a result, beginning students of algebra and programming sometimes make the error of thinking that $-x$ is negative, or less than zero. The "–" in –5 is exactly the same as the "–" in $-x$ and in $-(x+y)$; it is not a sign indicating a negative number; it is the negation operator; its result may be positive or negative or zero.

Unevaluation

Unevaluation is the reverse of evaluation. It means removing a point, rep, or quote and replacing them with arithmetic operators. For example, 1.23 = 123 / 100 .

Let $x = 1.2\cdot34$. The repeating part is 2 digits, so we multiply times 100 in order to get a new number with the same repeating digits in the same positions: $100\times x = 12\cdot3.4$.
Now $100\times x - x = 99\times x = 12\cdot3.4 - 1.2\cdot34 = 122.2$.

```
      1 2·3.4 3 4
  –   0 0 1.2·3 4
  =   1 2 2.2·0 0
```

So $x = 122.2 / 99 = 1222/990$. Hence $1.2\cdot34 = 1222/990$.

Let $x = 1.2'34$. This repeating part is also 2 digits, so multiply times 100 getting $100\times x = 12'3.4$. Now $100\times x - x = 99\times x = 12'3.4 - 1.2'34 = 2.166$.

```
      1 2'3.4 0 0
  –   1 2 1.2'3 4
  =   0 0'2.1 6 6
```

So $x = 2.166 / 99 = 2166/99000$. Hence $1.2'34 = 2166/99000$.

Let $x = 43'2.1$. Since the first digit of the repeating part (4) is greater than the first digit after the repeating part (2), it is a negative number. We begin by negating it to get a positive number.
$-x = -43'2.1 = 56'7.8 + .1 = 56'7.9$
Now $100\times(-x) - (-x) = -99\times x = 56'790. - 56'7.9 = 222.1$.

```
      5 6'7 9 0.0
  –   5 6 5 6'7.9
  =   0 0'2 2 2.1
```

So $x = -222.1/99 = -2221/990$. Hence $43'2.1 = -2221/990$.

Duplication

Each rational number has many representations. One criterion for choosing among them is to make the representation as short as possible. In the following equations, all three marks (point, rep, quote) are shown. In each equation, the left side can be shortened to the right side.

!1·2323	=	!1·23	repetition within repeating part
1212'3:	=	12'3:	repetition within repeating part
!12·32	=	!1·23	repeating part can be rolled to the left
21'23:	=	12'3:	repeating part can be rolled to the right
000'1:00	=	'1:	leading and trailing 0s can be invisible
·0!	=	⋮	leading and trailing 0s can be invisible

When the placement of the point requires some leading 0s to be made visible, we choose to place the quote as far right as possible.

!001· = .00'1·

This does not shorten the representation, but it has the advantage that we can determine whether a number is positive or negative by comparing the first digit of the repeating sequence to the left of the quote with the first digit after the quote. When the placement of the point requires some trailing 0s to be made visible, we arbitrarily choose to place the rep as far right as possible.

'1·00. = '100:

Another duplication of representation occurs when there is an infinite trailing sequence of 9s.

12.34·9 = 12.35

Proof: $9 \times 12.34\cdot9 = 10 \times 12.34\cdot9 - 1 \times 12.34\cdot9 = 123.4\cdot9 - 12.34\cdot9 = 111.15$

```
   1 2 3.4•9
−    1 2.3 4•9
=  1 1 1.1 5
```

So 12.34·9 = 111.15 / 9 = 12.35 .

For positive rational numbers, the quote and the rep offer competing representations. Here are two representations of one-third.

6'7 = :3

Dividing 1 by 3 using quote division produces the left side; dividing 1 by 3 using rep division produces the right side. Each can be verified by multiplying it times 3 . To convert a positive number from quote representation to rep representation, unevaluate it and then re-evaluate it using rep division. To convert a positive number from rep representation to quote representation, unevaluate it and then re-evaluate it using quote division.

Here is another way to shorten a representation that needs some explanation.

·1! = ⋮

Expanding ·1! we get a doubly-infinite sequence of 1s with a point somewhere.

←1111111.1111111→

Multiplying that times 10 moves the point right one place. But there's no difference.

$10 \times \cdot1! = \cdot1!$

Therefore ·1! = 0 . (But see this warning.) A similar calculation shows that ·123! = 0 , and the same for any other digit sequence preceded by a rep and followed by a quote, with a point anywhere.

With all three marks (point, rep, quote), finding the shortest form is complicated. With just point and quote (rep is invisibly at the right end), every rational number has a unique shortest form found as follows: remove repetitions within the repeating part, roll the repeating part as far as possible to the right, and delete unnecessary leading and trailing 0s.

Understanding

An unevaluated expression, or a partially evaluated expression, can sometimes be more understandable than a fully evaluated expression.

Perhaps the most important attributes of a number, at least for human understanding, are its sign (is it greater than zero, equal to zero, or less than zero?) and magnitude (how far is it from zero?). For that purpose, a positive number is best represented with a point and rep. For example, it is clear that $12:3$ is more than 12 but less than 13 units from zero. But $6'79$, which is the same number, is less obviously so. For that same purpose, a negative number is best represented as an unevaluated negation with a point and rep. For example, $-12:3$ is clearly between 12 and 13 units less than zero. But $3'21$, which is the same number, is less obviously so.

If you cut a pie into three equal parts, how much of the pie is each part? The answer is obtained by doing the division $1/3$, and we get either $:3$ or $6'7$. But the unevaluated expression $1/3$ says more clearly that it's the amount of pie that results from cutting one pie into three parts.

The magnitude of a large number like 12300000000000000 is not so apparent because we have to count the digits. Scientific notation represents this number as 1.23×10^{16}. Leaving the exponentiation and the multiplication unevaluated makes the magnitude clearer. (On the other hand, any author writing for non-scientists feels the need to explain that 10^{16} is a 1 followed by sixteen 0s; the author explains 10^{16} by evaluating it.) Similarly for small numbers like 0.000000000000000123. Unevaluated scientific notation 1.23×10^{-16} saves us from counting digits.

In extreme cases, evaluation is impractical. If we evaluate 2^{65536} we get a digit sequence with about twenty thousand digits in it, and the evaluation obscures the fact that it's a power of 2. (In fact, this example is an exponentiation tower of six 2s, which is typographically inconvenient.)

In some cases, evaluation is impossible. Finite sequences of digits with point, rep, and quote represent all and only the rational numbers. We can write the unevaluated expression $2^{1/2}$, which is a square root of two, an irrational number, but we cannot evaluate it. We can approximate it, for example, 1.41421356237, but we cannot evaluate it exactly. We can write the unevaluated expression $(-1)^{1/2}$, which is a square root of minus one, an imaginary number, but we cannot evaluate it, not even approximately.

Computation

A number might be an input or an output, coming from or going to a human, in which case it should be expressed in the form that is best for human understanding, as just discussed in the previous section. Or a number might be part of a computation, coming from a previous operation and going into a subsequent operation. In this section, we look at what form is best for that purpose.

Rational numbers represented with (a negation and) a point and a rep cannot, in general, be

added, subtracted, or multiplied in that form, due to the lack of a right end. They also cannot be divided because division requires subtraction. The easiest way to do these operations is to translate the numbers to quote notation, then perform the operations. Rep division for natural numbers was described earlier as a sequence of guesses, multiplications, and subtractions, then comparing the current dividend with all previous dividends in order to recognize the repetition and insert the rep. A computer does not "guess"; it repeatedly subtracts the divisor from the current dividend until the result is negative, then backs up one subtraction. (There are tricks to speed this up, but we won't go into them here.)

For comparing the magnitudes of two numbers, rep is better than quote. In rep notation, the usual algorithm is the following: align the points; if necessary, make enough leading 0s visible so that the leftmost digits of the two operands are aligned; then compare corresponding digits, left-to-right, until the first pair that differ.

```
    1 2.3•4 5
>   0 5:4 3
=   true
```

This comparison algorithm is not quite correct. It incorrectly says that $1.23\cdot9$ is smaller than 1.24, but they are equal.

Rational numbers represented with a point and quote can be added, subtracted, and multiplied the same way as natural numbers plus repetition detection. They can be negated. Quote division, in a nonprime base (like ten), starts by removing factors of the base from the divisor. In a prime base (like 2), this step is void. Then quote division proceeds just like multiplication, producing the answer digits from right to left, each one determined by the rightmost digit of the current difference and divisor (no "guesses"; trivial in binary). Comparison is a subtraction followed by sign determination.

Rational numbers in the unevaluated form a/b or $-a/b$, where a is a natural number and b is a positive natural, can be negated, added, subtracted, multiplied, and divided without evaluating them, producing a result in the same form. To add two unevaluated (negations and) divisions, for example $(a/b) + (-c/d)$, requires the following steps.

- sign comparison to determine if we will be adding or subtracting; in our example, the signs differ so we will be subtracting
- then 3 multiplications; in our example, $a\times d$, $b\times c$, $b\times d$
- then, if we are subtracting, a comparison of $a\times d$ to $b\times c$ to determine which is subtrahend and which is minuend, and what is the sign of the result; let's say $a\times d < b\times c$ so the sign will be $-$
- then the addition or subtraction; $b\times c - a\times d$ and we have $-(b\times c - a\times d)/(b\times d)$
- finding the greatest common divisor of the new dividend and divisor
- dividing dividend and divisor by their greatest common divisor to obtain a normalized result

Normalizing the result is not necessary for correctness, but without it, the space requirements quickly grow during a sequence of operations. Subtraction is almost identical to addition. Multiplication is sign determination, two multiplications, and normalization. For example,

$$(a/b) \times (-c/d) = -(a\times b)/(c\times d)$$

Division is similar to multiplication.

$$(a/b) / (-c/d) = -(a\times d)/(b\times c)$$

Comparison is similar to division.

$$(a/b) > (c/d) = (a\times d) > (b\times c)$$

Two numbers represented in scientific notation $x\times b^n$ and $y\times b^m$ can be multiplied producing an answer in scientific notation by doing a sign determination, a multiplication, and an addition

$x \times y \times b^{n+m}$. For example,

$(2.34\times10^{12})\times(-5.67\times10^{7}) = -(2.34\times5.67)\times10^{12+7} = -13.2678\times10^{19} = -1.32678\times10^{20}$

To add or subtract them requires adjusting the point and exponent of one number to make their exponents equal; checking their signs to determine whether to add or subtract; if subtracting, comparing the magnitudes to determine which is the subtrahend and which the minuend and to determine the sign of the result, then adding or subtracting, then possibly again adjusting the point and exponent. To compare them requires adjusting the point and exponent of one number to make their exponents equal, then comparing the other factors.

The time required to perform an operation depends on the number of steps, and also on the lengths of the operands, which we look at next.

Space

Quote notation and rep notation require essentially the same amount of space. But that amount can differ greatly from the space required for an unevaluated expression. We saw the example earlier that 2^{65536} requires about twenty thousand digits. The biggest win for unevaluated (negation and) division occurs for some prime divisors. For examples,

1/7 = :142857 = 285714'3
1/23 = :043478260869565217391 3 = 6956521739130434782608'7
(1/7 in binary is 1/111 = :001 = 011'1)
(1/23 in binary is 1/10111 = :00001011001 = 01111010011'1)

To represent 1/947 as an unevaluated division requires 4 decimal digits or 12 binary digits, but rep or quote notation requires 474 decimal digits or 947 binary digits. (Extra bits are required to delimit two variable-length numbers, but these are the same for all representations, so ignoring them does not affect the comparison.)

Here is a pattern of decimal quote notation such that the unevaluated negation and division takes about three time as as many digits, independent of scale.

.100000001' = –100000001/999999999000000000 (9 versus 27 digits)
.100000000000001' = –100000000000001/999999999999999900000000000000000
(15 versus 45 digits)

Exactly the same pattern in binary produces the same result.

.100000001' = –100000001/111111111000000000 (9 versus 27 bits) = –257/261632
.100000000000001' = –100000000000001/1111111111111110000000000000000
(15 versus 45 bits) = –16385/1073709056

Worst-case examples, some of which we have just seen, and worst-case analysis, which we have not presented, are not representative of the space requirements that one encounters in practice. The 180,000 shortest dividend-divisor pairs require 15.65 bits on average, and those same numbers in rep or quote notation require 39.48 bits on average. Taking the shortest dividend-divisor pairs, and then evaluating to rep or quote notation, results in a biased comparison in favor of dividend-divisor pairs. If we take all binary quote representations up to and including 14 bits (all quote positions and all radix point positions), then discard those that are not in shortest form, we have 1,551,567 numbers requiring 13.26 bits on average. If we unevaluate them to dividend-divisor pairs (with a negation if necessary), then normalize the result by removing common factors, they require 26.48 bits on average. This comparison is biased in favor of quote notation. The programs used to calculate these numbers are in the Appendix.

Approximation

Current computer hardware typically offers approximate floating-point arithmetic in two precisions: 32 bits and 64 bits. Numerical analysis is concerned with arranging a computation to get the most accurate answer possible using the approximate arithmetic provided by the computer hardware. Sometimes more accuracy is wanted than is possible using the hardware available. Sometimes perfect accuracy is wanted. It is possible to build hardware that provides perfect accuracy, but even without it, there are software packages that provide perfect accuracy. Perfect accuracy requires variable-length representations, which could, if suitable hardware were designed, be shorter on average than fixed-length hardware representations, but longer in the worst case. Ideally, each programmer should be able to choose the accuracy they need and are willing to pay for in space and time. Arithmetic should be perfect, but an approximation operator can be used to shorten representations whenever the loss of accuracy is not too damaging. This turns numerical analysis around: instead of analyzing the accuracy you are given, you choose the accuracy you can afford.

To approximate a positive number in quote notation, for example 12'34.567 , first convert it to rep notation 22.445·78 , and then approximate by truncation or rounding: to 22.4458 or 22.446 or 22.45 , and so on.

Conclusion

A sequence of digits is the standard way to represent a natural number. It is also standard to use a point (or equivalent notation), extending the representation to some nonnegative rational numbers. We are taught in school, though it is not standard practice, to use a rep (or equivalent notation), extending the representation to all nonnegative rational numbers. With a point and quote, we can represent all rational numbers. With all three (point, rep, quote) we have all rational numbers with multiple representations. The rep has the advantages that magnitude is easily seen, and comparisons and approximations are easily made, but the disadvantages that negation cannot be performed, and further additions, subtractions, multiplications, and divisions are difficult. The quote has the advantage that all further arithmetic can be performed.

For some purposes (understanding or space), it is sometimes best to leave some arithmetic operations unevaluated.

Another Paper on the same topic:
E.C.R.Hehner, R.N.S.Horspool: a New Representation of the Rational Numbers for Fast, Easy Arithmetic, *SIAM Journal on Computation* v.8 n.2 p.124-134, 1979 May

a Video on the same topic: www.cs.utoronto.ca/~hehner/NRvideo.mp4

a PhD Thesis that studied quote numbers is "P-adic Number Systems for Error-free Computation" by Ruth Ann Lewis, The University of Tennessee, 1979 August (supervised by Robert Todd Gregory)

Appendix: Programs for Calculating Lengths

First we list the shortest numbers as unevaluated (negation and) division, evaluate them using binary quote and point, and compare their lengths. In both the unevaluated and evaluated forms, there are two sequences of digits with variable length, so further bits are required to delimit these sequences. These are the same for both forms, so their omission does not affect the comparison. The programming language is ProTem.

```
` Procedure  norm  normalizes a numerator and denominator by removing their greatest common
` divisor.

new gcd = ⟨a: (nat+1) → ⟨b: (nat+1) → ` greatest common divisor of  a  and  b
        a=b ⊨ a ⫤ a<b ⊨ gcd a (b–a) ⫤ gcd (a–b) b⟩⟩.

new norm  [[plan num: nat+1 [[denom: nat+1 ` normalize num/denom
        [[new g:= gcd num denom.  num:= num/g.  denom:= denom/g]]]].

` Procedure  rspace  calculates the space required for the unevaluated (negation and)
` division representation of a rational number.

new rspace = ⟨num: (nat+1) → ⟨denom: (nat+1) →
        result s: nat:= 1 ` 1 bit for the sign
        [[new n: nat:= num.  new d: nat+1:= denom.
          loop [[n:= div n 2.  s:= s+1.  if n>0 [[loop]]].
          loop [[d:= div d 2.  s:= s+1.  if d>0 [[loop]]]]⟩⟩.

` Procedure  qspace  calculates the space required for the roll-normalized quote notation
` representation of a rational number.  (It is not repeat-normalized.)  No space is given to
` the point and quote.

new qspace = ⟨num: (nat+1) → ⟨denom: (nat+1) →
        result s: nat:= 0
        [[new n: nat:= num.  new d: nat+1:= denom.
          loop [[if even n [[n:= n/2.  loop]]].
          loop [[if even d [[d:= d/2.  loop]]].
          new remainders: *int:= n.

          ` Function  rem  says whether its argument is in the  remainders  string.
          new rem = ⟨n: int → result rem: bin:= ⊥
                        [[new i: nat:= 0.
                          loop [[if i≠↔remainders
                                [[if n=remainders_i [[rem:= ⊤]]
                                  else [[i:= i+1.  loop]]]]]]⟩.

          loop [[s:= s+1.
                if even n [[n:= n/2]] else [[n:= (n–d)/2]].
                if – rem n [[remainders:= remainders; n.  loop]]]]⟩⟩.
```

new *count*: *nat*:= 0. ` number of examples
new *qsum*: *nat*:= 0. ` total length of evaluated numbers
new *rsum*: *nat*:= 0. ` total length of unevaluated numbers

for *num*:= 1;..301
⟦**for** *denom*:= 1;..301
 ⟦**new** *n*: *nat*:= *num*. **new** *d*: *nat*:= *denom*. **new** *r*: *nat*+1:= 1. **new** *q*: *nat*+1:= 1.
  norm n d. *r*:= *rspace n d*. *q*:= *qspace n d*.
  count:= *count*+2. ` because negatives and positives take the same space
  rsum:= *rsum*+*r*. *qsum*:= *qsum*+*q*⟧⟧.

! "count="; *count*; *nl*;
 "average unevaluated length="; *rsum*/*count*; *nl*;
 "average evaluated length="; *qsum*/*count*.

old *count*. **old** *rsum*. **old** *qsum*

Here is the output:

```
count=180000
average unevaluated length=15.653334
average evaluated length=39.479668
```

Next we list the shortest numbers evaluated using binary quote and point, unevaluate them into (negation and) division form, and compare their lengths.

new *shl* = ⟨*n*: *nat* → ⟨*m*: *nat* → ` shift *n* left *m* places; *n*×2^m
  result *r*: *nat*:= *n* ⟦**for** *i*:= 0;..*m* ⟦*r*:= *r*×2⟧⟧⟩⟩.

new *shr* = ⟨*n*: *nat* → ⟨*m*: *nat* → ` shift *n* right *m* places; *floor* (*n*×2^{-m}) or *div n* (2^m)
  result *r*: *nat*:= *n* ⟦**for** *i*:= 0;..*m* ⟦*r*:= *div r* 2⟧⟧⟩⟩.

new *count*: *nat*:= 0. ` number of examples
new *qlen*: *nat*:= 0. ` total length of quote representations
new *rlen*: *nat*:= 0. ` total length of numerator/denominator representations

for *length*:= 1;..15
⟦**for** *string*:= 0;..(*shl* 1 *length*) ` each *string* of that *length*
 ⟦**for** *quote*:= 0;..*length* ` each quote position (at least one bit to left of quote)
  ⟦**if** *even* (*shr string* (*length*–1)) ≠ *even* (*shr string* (*quote*–1)) ` roll-normalized
   ⟦**if** ` repeat-normalized
     result *repeatnorm*: *bin*:= ⊤
     ⟦**new** *len*: *nat*:= *div* (*length*–*quote*) 2. ` the length of the possibly repeating part
      trythislen ⟦**if** *len*>0 ` 1 ≤ *len* ≤ (*length*–*quote*)/2
        ⟦**new** *extract* = ⟨*i*: *nat* → ⟨*l*: *nat* → ` index *i* length *l*
          shr string i – *shl* (*shr string* (*i*+*l*)) *l* ⟩⟩.
         new *ex*:= *extract quote len*.
         if ` the negative part is a repetition (twice or more) of *ex*
           result *r*: *bin*:= ⊤
           ⟦**new** *i*: *nat*:= *quote*+*len*. ` *i*+*len* ≤ *length*

```
                    iloop ⟦new ey:= extract i len.
                        if ex=ey ⟦i:= i+len. ` i≤length
                            if i+len ≤ length ⟦iloop⟧
                            else ⟦r:= ⊥⟧⟧
                        else ⟦r:= ⊥⟧⟧⟧
                ⟦repeatnorm:= ⊥⟧ else ⟦len:= len–1. trythislen⟧⟧⟧⟧
    ⟦for point:= 0;..length+1 ` each point position (right end, interior, left end)
      ⟦if ` the rightmost bit is 1 or it's to the left of quote or point
          odd string ∨ (quote=0) ∨ (point=0)
        ⟦ ` convert to numerator/denominator
          new num: nat:= shl string (length–quote) – string – shl (shr string quote) length.
          if num<0 ⟦num:= –num⟧.
          new denom: nat:= shl (shl 1 (length–quote) – 1) point.
          norm num denom.
          ` update statistics
          count:= count+1. qlen:= qlen+length.
          rlen:= rlen+1. ` for the sign
          loop ⟦num:= div num 2. rlen:= rlen+1.
              if num>0 ⟦loop⟧⟧.
          loop ⟦denom:= div denom 2. rlen:= rlen+1.
              if denom>0 ⟦loop⟧⟧⟧⟧⟧⟧⟧⟧⟧.

! "In "; count; " examples, quote average length = ";
   qlen/count; ", num/denom average length = "; rlen/count.

old shl. old shr. old gcd. old norm. old count. old qlen. old rlen
```

Here is the output:

```
In 1551567 examples, quote average length = 13.26, num/denom average length =
26.48
```