



© 1997–2009, Millennium Mathematics Project, University of Cambridge.

Permission is granted to print and copy this page on paper for non-commercial use. For other uses, including electronic redistribution, please contact us.

September 2000

Features



Take a break

by Emily Dixon



Everyone knows that it is easy to make mistakes when lots of numbers are involved – and I'm not talking about all those slip-ups that occur in exams. It is so easy to get a few digits of a bank account number mixed up, or for fingers to slip on a keyboard and enter the wrong numbers. Imagine the consequences of these errors: things might go your way, you might get access to Bill Gates's bank account, for example, but the chances of that are relatively slim. Alternatively, if a bar code gets printed wrongly, you might end up paying the price of a bottle of champagne for your carton of apple juice or if the number on your airline ticket is wrong, who knows where you or your luggage might end up? Luckily, there are schemes in place to detect, and in some cases even correct, such errors almost immediately.

Error Detection

An error-detecting code is a way of transmitting data – a number, say – so that most common mistakes will be detected at once, before they can cause any damage. A very simple example would be to transmit the whole number twice. This is grossly inefficient, however. It doubles the length of the number, and even then, if an error is detected, leaves us in the dark about the correct number – was the first transmission correct, or the second? The code performs error *detection*, but not error *correction*. We'll see in this article that there are far more efficient codes available.

There are many different methods of error detection. Generally, the number to be transmitted is followed by a number of *check digits* – most often one for simple error detection, but if we are to do error correction too, at least two will be needed. Then when the number is transmitted, another calculation can be done at the receiving end to check that the received number (including the check digit) is valid. We shall look at three

Take a break

schemes used for calculating check digits: *modular* schemes, *permutation* schemes and *noncommutative* schemes, and at some examples of where they are used.

Modular Arithmetic

Modular arithmetic involves working with the remainders generated by division. For example, if 36 is divided by 7, the remainder is 1. Using modular arithmetic notation, this can be written as

$$36 = 1(\text{mod}7).$$

Similarly,

$$47 = 11(\text{mod}12),$$

$$62 = 2(\text{mod}10),$$

and in general

$$p = q(\text{mod}N) \text{ if } p - q \text{ is a multiple of the integer } N.$$

For example,

$$47 = 11(\text{mod}12) = -1(\text{mod}12),$$

since

$$47 - (-1) = 48 = 12 \times 4.$$

The simplest check digit schemes use the code number itself as part of a modular arithmetic operation. For example, take a code number, c , create a check digit, d , for this number such that:

$$c = d(\text{mod}N)$$

for some fixed modulus N .

For example, suppose the number to transmit is 12345 and we have chosen the modulus $N = 10$. The check digit would be 5, and we would transmit it after the digits of the number: 123455. If the receiver received, say, 123445, they would know there had been a mistake since 12344 is not equal to 5 (mod 10). Obviously this will only catch an error in the last digit of the number (or in the check digit), so the choice of $N = 10$ as modulus was particularly poor. We see in the next section that a different modulus can catch a fairly high proportion of the most common errors, even using this simple scheme. In general, though, it will at most tell us that there has been an error; it will not help us find where the error was, and cannot be error-correcting.

There are various different errors that can occur when numbers are written, printed or transferred in any manner. Different methods of assigning check digits are better at detecting certain kinds of errors than others. The most common types of errors that occur in practice and their frequencies, according to one study, are as follows:

Take a break

Error type	Form	Relative frequency
single error	a replaced by b	79.1%
transposition of adjacent digits	ab replaced by ba	10.2%
jump transposition	abc replaced by cba	0.8%
twin error	aa replaced by bb	0.5%
phonetic error	$a0$ swapped with $1a$ $a = 2, \dots, 9$	0.5%
jump twin error	aca replaced by acb	0.3%

Another common type of error not mentioned here is accidental insertion or deletion of characters. In the cases we will consider, the number will have a fixed length, so insertions and deletions will be automatically detected.

We are now ready to see how this is all put into practice so let's bring on our first example ...

Airline tickets

Airline tickets have a ten-digit serial number followed by a check digit. This check digit is calculated using the modular scheme discussed above, with modulus $N = 7$.

For example, a ticket might have serial number 3387972544. Since this number equals $5 \pmod{7}$, the check digit is 5. The number is printed on the ticket with the check digit: 33879725445.

This is quite a primitive method. Is it really any good at detecting errors? From the list of relative frequencies of errors, we can see that far the most important errors to detect are single-digit errors and transpositions of adjacent digits. Let's see how well this modular scheme does in these cases.

Single digit errors

As we've just seen, an airline ticket has a ten-digit serial number, followed by a check digit. Let's call the serial number a , and the check digit a_c . Then a_c is calculated so that

$$a = a_c \pmod{7}.$$

The number a itself consists of ten digits; let's call them a_1, a_2, \dots, a_{10} . A single-digit error consists of replacing one of these digits a_i with some other digit, a'_i , say, giving a new (and wrong) serial number a' , or possibly the right serial number but the wrong check digit.

Will a single-digit substitution in the serial number show up – that is, will it change the value of $a \pmod{7}$? The answer is yes, as long as the substituted digit itself has changed $\pmod{7}$. (You might like to try to convince yourself that this is because 7 has no factors in common with 10, the base in which the numbers are expressed.)

In other words, this code will catch most single-digit substitutions. It will miss those where $a_i = a'_i \pmod{7}$,

Take a break

i.e. it will miss the errors $0 \leftrightarrow 7, 1 \leftrightarrow 8$ and $2 \leftrightarrow 9$ in the serial number. For example, suppose the example ticket number above were copied as 33879795445. A "2" has been incorrectly copied as a "9". A check will verify that $3387979544 = 5 \pmod{7}$, and so the error will be missed.

Each digit could be any of the 10 digits 0–9; a substitution could replace it with any of the other 9 digits; and there are 10 digits in all. The number of possible substitutions is thus $9 \times 10 \times 10 = 900$, and 60 of those will be missed.

Other possible single digit errors that could occur involve the check digit. The numbers 7,8 and 9 are not allowed remainders, so there are only 7 valid check digits, {0,,6}. This means there are 63 possible errors, but all of them would be detectable.

Therefore the single error detection rate is 903/963 or 93.8%.

Throughout these calculations, we have assumed that all errors have the same probability of being introduced (ie 5 being substituted for 6 is as likely as 1 being substituted for 9). In practice, this is unlikely to be true, but there is not enough data available to calculate more accurate probabilities.

Transposition of adjacent digits

Now we look at transpositions (where two adjacent digits ab are transposed to ba). Among the first 10 digits there are 9 pairs that could be transposed, and for any pair there are 100 possibilities. If the digits of a pair are the same, transposing them won't make any difference, so for each pair there are 90 possibilities that could lead to an error. The error will be undetectable if the digits transposed are equal mod 7 – namely if they are 07, 70, 18, 81, 29 or 92. So there are 6 undetectable transpositions out of each 90. The total number of possible errors is 810, of which 54 would not be detected.

For the final pair (the 10th digit and the check digit), there are 70 possibilities, since the check digit cannot be 7, 8 or 9. Of these, 63 would lead to an error if the digits were transposed, but all such errors would be detectable.

In all, of 873 possible transposition errors, only 54 would be missed, giving a detection rate of 93.8%.

These error detection rates are quite high, but could they be higher? Let's see if better results can be obtained simply by using a different value for N . The choice of modulus 9 is often used: for example, the identity numbers on US Postal Orders is 10 digits long, and consists of a 9–digit serial number and a check digit equal to the serial number mod–9 remainder. If we were to do the calculations for a number of the same length as that used in the example above (10 digits followed by a check digit), we would find the single error detection rate to be 98.0%, which is slightly higher than with a modulus of 7. However, the detection rate for the transposition of adjacent digits is 9.1%, which is much lower. (An error will only be caught if it involves the check digit.)

European Article Numbering Code

Barcodes are familiar to most as they are found on the majority of the things we buy. Look at a barcoded item, and underneath the barcode itself there is a string of digits, the last of which is a check digit. These use a slightly more complicated scheme of assigning check digits, involving a "weighted sum" of the digits of the number.

To calculate a "weighted sum" of a series of numbers, we choose a fixed sequence of numbers called

Take a break

Weights. Each number in the series is multiplied by the corresponding weight before being added to the total. For example, suppose we have chosen weights (1,2,3), and the series is (7,8,9). The weighted sum is

$$1 \times 7 + 2 \times 8 + 3 \times 9 = 7 + 16 + 27 = 50.$$

Barcodes in Europe follow the European Article Numbering Code (EAN) format. There are two versions, EAN-8 and EAN-13, which use 8 and 13 digits respectively. For both, the weights chosen are alternate 1's and 3's. We'll look at the 8-digit version (the calculations for the 13-digit version are very similar).

Since the last digit is a check digit, only 7 of the 8 digits actually encode information.



Sample of a 13-digit EAN bar code.

The format uses a modulus 10 scheme, with check digits (a_c) defined by

$$a_c = -(a_1, a_2, \dots, a_6, a_7) \cdot (3, 1, 3, 1, 3, 1, 3) \pmod{10}.$$

For example, if we start with the number 1234567 in the EAN-8 scheme, then our check digit is

$$\begin{aligned} a_c &= -(1, 2, 3, 4, 5, 6, 7) \cdot (3, 1, 3, 1, 3, 1, 3) \pmod{10} \\ &= -(1 \times 3 + 2 \times 1 + 3 \times 3 + 4 \times 1 + 5 \times 3 + 6 \times 1 + 7 \times 3) \pmod{10} \\ &= -60 \pmod{10} = 0, \end{aligned}$$

which makes the full bar code number 12345670.

Again, we must worry about how effective this scheme is in detecting errors.

Single error detection rate

If a digit d whose weight is 1 is changed to c , the weighted sum will change by $d - c$. The error will go undetected only if $d - c = 0 \pmod{10}$. But this happens only when $d = c$, in which case there has not been an error after all, so all errors of this kind are caught.

What if the weight were 3? Then the error would be undetected if $3(d - c) = 0 \pmod{10}$. But again, this cannot happen unless $d = c$. Thus, this method has a 100% single error detection rate.

Take a break

Transposition of adjacent digits detection rate

Suppose two adjacent digits, cd , are transposed to dc . If $c \in \mathbb{Z}_{10}^{\text{TM}}$ weight is 3 (hence $d \in \mathbb{Z}_{10}^{\text{TM}}$ weight is 1), the weighted sum is changed by

$$(3c + d) - (c + 3d) = 2(c - d)$$

which will be detected unless $2(c - d) = 0 \pmod{10}$, which can happen only if c and d differ by 5. The same would have applied if c had been weighted by 1 and d by 3.

As a result, the transpositions that will go undetected must involve $0 \leftrightarrow 5, 1 \leftrightarrow 6, 2 \leftrightarrow 7, 3 \leftrightarrow 8$ and $4 \leftrightarrow 9$. So, 10 transpositions are undetectable.

There are 100 possibilities for each pairing, and the transposition of 90 of these would result in an error. Therefore the detection rate is $80/90 = 88.9\%$.

International Standard Book Number (ISBN)

Almost every book published has an International Standard Book Number (ISBN) printed on it. The ISBN is a nine-digit code with a tenth digit which is – you guessed it – a check digit. ISBNs use a weighted modulus-11 scheme. This has the great advantage that it detects *all* single digit errors and transpositions. The disadvantage is that the remainder can be 10, which is not a digit. A remainder of 10 is represented as X in ISBNs. The fact that the resulting "number" must occasionally include a non-digit is a little untidy, and for some applications it would be highly undesirable. For example, some automatic credit-card booking systems require you to dial your credit card number on the keypad of your phone. This would work badly if the card number was occasionally liable to include a non-digit.

However, it is important to catch mistyped card numbers. We'll see later that credit cards use a very clever scheme whose detection rate for single errors and transpositions is better than the airline ticket and EAN schemes we've looked at so far.

Choosing the optimum modulus for a weighted scheme

The effectiveness of this type of scheme obviously depends quite a lot on the weights and the modulus. What conditions are needed for errors to be detected, and is there a best choice for the modulus?

Single digit errors

If a digit c is replaced by a different digit d , the error will be undetected when $(c - d)$ multiplied by the appropriate weight, w , is a multiple of N . This will happen less often if N has no factors in common with w ; perhaps you can see now why the barcode system uses weights of 1 and 3 (rather than 1 and 2, say). Other than that, the larger N is, the better. Once N is at least 10, all such errors will be caught if N has no common factor with any of the weights.

Transpositions

On the other hand, if adjacent digits c and d are transposed, the transposition will go undetected when $(c - d)$ multiplied by the *difference* between their weights is a multiple of N . We'd like to arrange for successive weights to have no factor in common with N . Unfortunately, if N is even (say $N = 10$), and if as above the weights themselves have no common factor with N (so they're all odd), then their differences must be even.

Take a break

Generally, the more prime N is, the better life will be. This is why, if the serial number has digits 0–9, we needed a modulus of at least 11 before we could be certain of catching all single–digit errors and all transposition errors. However, as we saw, a scheme that uses $N = 11$ has to choose some method of coping with the possibility of the remainder being 10.

Permutations

A permutation (of a set of digits, say) is a rule for systematically replacing one digit with another. You have probably met them in the form of substitution ciphers, such as the one where each letter of the alphabet is shifted up one, so "IFMMP" means "HELLO". This particular permutation is just one long *cycle* – A goes to B, B goes to C, and so on all the way to Z which goes back to A. Like any permutation, it can be applied more than once; for instance if it is applied twice to A, the result is C.

We can write any permutation by writing down the cycles it contains. For instance, one permutation of the digits 0–9 could be written (02468)(1)(3)(5)(7)(9), showing that even numbers are cycled round (0 goes to 2, 2 goes to 4, etc), and all odd numbers are unchanged. We can call the permutation σ , so in this case, $\sigma(4) = 6$, $\sigma(3) = 3$, and so on.

Credit cards

Credit cards use an error–detecting scheme that was developed by IBM. It uses the permutation

$$\sigma = (0)(124875)(36)(9).$$

In other words $\sigma(0) = 0$, $\sigma(1) = 2$, $\sigma(2) = 4$, etc. Notice that $\sigma(x)$ is always equal to $2x \pmod{9}$.

In a 16 digit credit card number, the final digit is the check digit. Let the credit card number be $(a_1, a_2, \dots, a_{15}, a_{16})$, with a_{16} being the check digit. Then

$$a_{16} = -[\sigma(a_1) + a_2 + \sigma(a_3) + a_4 + \dots + a_{14} + \sigma(a_{15})] \pmod{10}.$$

Note that in this example the permutation was applied to a_i where i is odd (a_1, a_3 etc), because there is an odd number of digits excluding the check digit. Had this scheme been used on a number with an even number of digits excluding the check digit, the permutation would have been applied to a_i where i was even.

This scheme will catch all single–digit errors. For example if digit a_i is changed from c to d , and i is even, the remainder will change by $c - d$, which is non–zero (and is, of course, smaller than the modulus $N = 10$). If i is odd, it will change by $\sigma(c) - \sigma(d)$. This is again non–zero: $\sigma(c)$ cannot be equal to $\sigma(d)$ if σ is a permutation.

How about transpositions? If two adjacent digits c and d are transposed, one of them must have the permutation applied – say c . The remainder will be unchanged only if $\sigma(c) + d = \sigma(d) + c$. Since $\sigma(x) = 2x \pmod{9}$, this happens only when $c = d \pmod{9}$, that is, only when c and d are 0 and 9 (in either order).

Therefore, for each pair of adjacent digits, of the 90 possible transposition errors, two will be undetectable. So the detection rate for transpositions is $88/90 = 97.8\%$.

Noncommutative Schemes

The detection rates of the permutation scheme used for credit cards were pretty good, but another scheme, still using only one check digit in the range 0–9, can achieve a 100% detection rate for both single–digit errors and transpositions. On top of a permutation, it uses a so–called "noncommutative multiplication" operation on the digits of the code number.

Error Correction: two check digits

Being able to detect that an error has occurred is all well and good but it would be helpful to be able to correct it too. With more check digits, one can do just that.

Introducing a second check digit means that one can be used, as before, to detect and find the magnitude of an error, and the other can then locate and correct it.

As before, modulus 11 is an effective modulus. One good two–check–digit scheme uses modulus 11 twice but two different series of weights. Using these two check digits, all double errors can be detected and all single errors corrected.

1239552 occurred, then $1+2+3+9+5+5+2 = 27 = 5(\text{mod } 11)$. Assuming that only a single error has occurred, this tells us that one of the digits is 5 too large. $1 \times 1+2 \times 2+3 \times 3+9 \times 4+5 \times 5+5 \times 6+2 \times 7 = 119 = 9(\text{mod } 11) = 20(\text{mod } 11)$. $20 = 5 \times 4$ so this tells us that the fourth digit is where this error occurred.

The first scalar product could have told us that one of the digits was 6 too small but then no sensible answer would have been obtainable from the second scalar product. -->

We could go further and introduce even more check digits and thus be able to correct a greater number of errors. Of course, the downside is that the more check digits are used, the longer the number becomes – which is tiresome if the number is going to be typed in or copied down.

Conclusions

Error correcting schemes do not end here. They are used extensively with binary data and appear in our CD players, digital televisions and in the transmission of data from space probes. Some computer viruses even use error correcting codes to check and repair themselves if someone has modified them. A lot of these rely on polynomial arithmetic rather than check digits. One last class of error–correcting codes that do involve check digits are *Hamming codes*, which uses matrix manipulations to calculate check digits for binary data.

Error correcting also takes place in nature. It is believed that a lot of DNA which appears to be redundant is actually involved in an error correcting procedure which avoids errors in DNA replication, so it is fair to say that without efficient error detection and correction we would not be here.

About the author

Take a break



I am currently in the U6 at the Perse School for Girls, Cambridge studying for A levels in maths, further maths, physics and chemistry. I hope to go on to study maths at university in 2001 (destination unknown).

I wrote this article whilst working with the Millennium Mathematics Project, during the summer of 2000, arranged and funded by the Nuffield Science Bursary Scheme.



Plus is part of the family of activities in the Millennium Mathematics Project, which also includes the NRICH and MOTIVATE sites.