Reducing number entry errors: solving a widespread, serious problem

Harold Thimbleby[1] Paul Cairns[2]

[1]Future Interaction Technology Lab, FIT Lab, Swansea University, Swansea SA2 8PP, UK
[2]Department of Computer Science, University of York, York YO10 5DD, UK

November 20, 2009

1 METHODS SUMMARY

We programmed three independent analyses: a Monte Carlo method, an exhaustive method, and a symbolic analysis. All program source code, both of the analyses and of the demonstration user interface, are available at [for the purposes of review, see: www.harold.thimbleby.net/numbers]

The Monte Carlo method and the symbolic analysis were both written in *Mathematica* by H.T., and the exhaustive method was written in Java by P.C., making use of Microsoft Excel for data presentation. In all methods, the general approach was to consider a number intended to be keyed, simulate user slips in the keying of that number, then parse the resultant key sequence to assign it a value, as a typical device might. We are thus able to estimate the probability of out by ten errors, and the probability of out by ten errors that would be blocked by syntax checks.

The purpose of the analysis is to see the extent to which syntax checking reduces out by 10 errors when entering numbers in a typical interactive device. The main problem in conducting this analysis is that there is no easily available data on the kinds of errors made when doing number entry, the underlying rate of errors or the distribution of numbers being entered. The analysis therefore takes an exhaustive approach looking at all possible number entries within a given range and examines the impact of errors over a range of underlying error rates and what improvements syntax checking offers over the absence of syntax checking in terms of the underlying probability of an out by 10 error occurring.

To do the exhaustive analysis, three independent methods are used. All three consider the proportion of out by 10 errors that would be blocked by syntax checking. In the first method, a Monte Carlo simulation of number entry with varying error rates is conducted. In the second, a fully exhaustive method is followed where each target number in a range is considered in turn. The third and final method takes a symbolic approach where the proportion of blocked out by 10 errors is calculated as a function of the underlying error rates.

All three methods support the finding that syntax error blocking roughly halves the probability of an out by 10 error occurring, regardless of the underlying error rate. Note that the Excel, Java and *Mathematica* program code used was written independently as a cross-check by the authors.

1.1 Parameterizing error probabilities

For a given single key-press when entering numbers, we use e to denote the probability that any key other than the intended key was pressed. From other contexts, it can be inferred that e is at most 0.1 but is more likely to be between 0.01 and 0.05. However in the case of high stress such as nurses may encounter in surgery, e may be somewhat higher, and then again with trained professionals, e may be somewhat lower. The only certainty is that e is not zero as human error is inevitable.

Sometimes, the incorrect key pressed has the effect of terminating the number entry, for example, by mistakenly pressing an `Enter` or `Run` key, which makes the device accept the entered number and start functioning. In other cases, the incorrect key is simply the wrong digit or a decimal point and number entry is able to continue.

We denote the probability of miskeying but not terminating the number entry by p and the probability of incorrect termination errors by q. Clearly $e = p + q$, but in order to simplify the variations in p and q whilst ensuring $0 \leq e \leq 1$, we define $k = q/p$, that is k is the proportion of termination to non-termination errors. Given that in many devices we considered in the medical domain (and also handheld calculators) there are many keys that when pressed cause a termination of number entry, we believe that $0.5 \leq k \leq 10$ is a realistic range for k.

1.2 The Monte Carlo method

The simplest approach for analysis is to use a Monte Carlo method, as it closely models how a user would enter a number: a certain number is intended but keying errors mean that another number may be keyed in. Thus the Monte Carlo program randomly selects a number from 0.01 to 99.99 in increments of 0.01, then works out the correct key sequence to enter this number, then uses the parameters p and q as described above to randomly alter the key sequence with the appropriate probabilities. The resulting key sequence is then parsed to obtain the value a typical device would get from that sequence of keys. The value is classified as out by 10 if the parsed value is out by a ratio of at least 10 from the intended value.

An important advantage of the Monte Carlo method is that it is both conceptually simple and simple to program, and it is therefore likely to be valid. However, because there are many numbers to sample and the probabilities of error are relatively low, the Monte Carlo method has to be run for a long time to get accurate results. Of course, the longer it is run, the more accurate the results will be.

A plot of results from a typical Monte Carlo analysis is shown in Figure **??**.

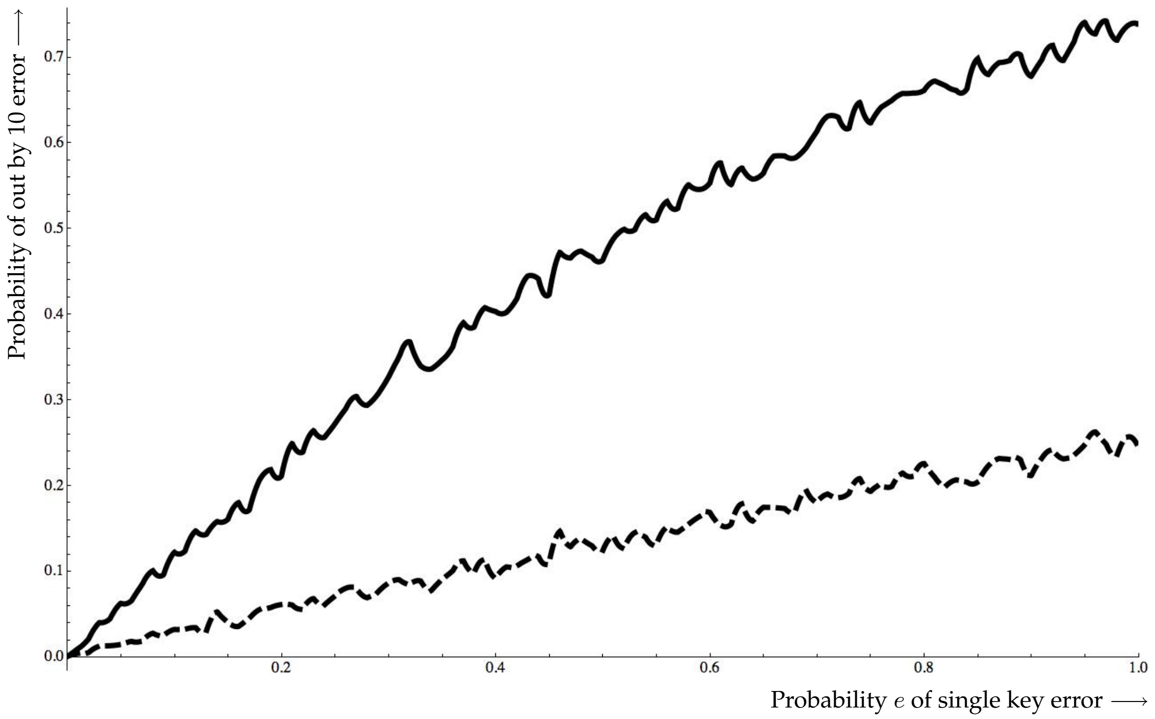

Figure 1: **A plot of the probabilities of an out by 10 error and a blocked out by 10 error as found in the Monte Carlo method as a function of e.** The solid line shows the behaviour of a simulated calculator-type device; the dashed line shows the reduction in out by 10 errors by blocking syntax errors. For all plotted points, $k = 10$. (The plot is the result of a short Monte Carlo run, hence the random variation; compare with Figure **??** which shows a smooth plot based on exact symbolic results.)

1.3 Exhaustive simulation

In this approach, every number from 0.01 to 99.99 in increments of 0.01 is treated as the target number that a person is intending to enter into a device. Each target number is already syntactically correct, for example having no leading zeroes or decimal points without a subsequent fractional part. The analysis then considers every possible miskeying of the target number and calculates the probability of producing each erroneous entry based on the parameters e and k. Additionally, the miskeyed entries are marked as to whether they would produce an out by 10 error, and whether syntax checking of the keyed entry would have blocked it. Thus, by summing the probabilities, it is possible to say what the probability of an out by 10 error is for a given target entry, and also what proportion of those errors would be blocked.

1.3.1 Java code

The simulation was written in Java v2 and full source code is provided online. The code allows the programmer to specify a set of values of e and k for the program to work through. For each pair of values, the program first generates each syntactically correct target number in the chosen range, then generates all possible alternative entries together with the relevant probabilities. The program does not just consider out by 10 errors, but also other error proportions, specifically out by r errors for $r = 1.5, 2, 5, 7, 5$ and 10.

For each target number, the program summarizes:

- The probability of correctly entering the target number
- The probability of producing a syntactically invalid miskeying
- For the different proportions r
  - The probability of an out by r error
  - The probability of an out by r error being invalid
  - The proportion of the probabilities of invalid out by r to all out by r errors

A single output file is produced containing all of these summaries for all considered numbers for each pair of values e and k. The files are in standard comma-separated variable (CSV) format. The data generated is illustrated in Table **??**.

Dosage	Correct	Invalid	PropErr	PropErrAndInvalid	PropR
:	:	:	:	:	:
0.9	0.985404884	0.005983099	0.008957196	0.005638271	10
1	0.995	0.002333333	0.002333333	0.002333333	10
1.1	0.985074875	0.00431869	0.005654681	0.002000444	10
:	:	:	:	:	:

Table 1: **Illustrative CSV output.** CSV data generated from the Java simulation (shown as a conventional table) for $e = 0.005, k = 0.5$.

The CSV files are then further processed in Excel to produce a summary header of the max, min and average values of all of the target number summary values. The headers are all collected into a single Excel file. The online information includes an example Excel file produced for a single pair of values for $e = 0.05$ and $k = 10$, as well as the file of all headers.

1.3.2 Results

The main results are the proportions of probabilities of invalid (and therefore blocked) out by 10 errors to all out by 10 errors. These were produced for $e = 0.005, 0.01, 0.02, 0.05$ and 0.1 and $k = 1.5, 2, 5$ and 10 and are given in Table **??**. For particular values of k it is notable that differences in e have only a small effect on the proportion of out by 10 errors that are blocked by syntax checking. Also, even for what we believe to be very low values of k, syntax checking still blocks around a third of all out by 10 errors.

1.4 Symbolic method

Mathematica is a symbolic mathematics system that can be programmed, but it does not need to know exact values. For example, adding $p + q + p$ in *Mathematica* will give the expression $2p + q$ as its result, whereas in a conventional programming language specific numerical values would be needed for both p and q before they could be added. The result in a conventional language

		k				
		0.5	1.5	2.0	5.0	10
	0.005	0.327	0.429	0.451	0.504	0.526
	0.01	0.328	0.431	0.453	0.505	0.527
e	0.02	0.330	0.433	0.456	0.508	0.530
	0.05	0.336	0.443	0.465	0.517	0.538
	0.1	0.348	0.458	0.480	0.531	0.552

Table 2: **Summary of out by 10 probabilities.** Proportion of probabilities of out by 10 errors that are blocked by syntactic checking over the range of target numbers of 0.01 to 99.99 in steps of 0.01.

would be numerically equal to $2p + q$ (within limits of precision), but would be a number, not an expression as it is in *Mathematica*. In particular, in *Mathematica* values of p and q can be assigned later, for instance to plot a graph, whereas in a conventional programming language, assigning different values to p or q later would not change the previously calculated expression's value.

In *Mathematica* we follow exactly the same procedure as in simulating number entry in Java, except we do not need explicit values for p and q, our probabilities of keying error. The advantage is that graphs can be drawn from the results, as the results are formulas in p and q; the disadvantage of *Mathematica* is that it is much slower than Java, so we do not consider the same range of numbers as in the two previous methods but instead we restrict the range to all values from 0.1 to 99.9 in increments of 0.1.

To draw graphs, each possible number parsed is considered and its out by r value calculated. Each out by r value is put in bins (say with a resolution of 0.01, that is $0.01, 0.02, 0.03, \ldots$) and the symbolic probability of that r value accumulated. At the end of a run, the bins contain the symbolic probabilities of each out by r value, from which a graph is readily drawn by considering ranges of p and q of interest. Figures **??** and **??** are examples.

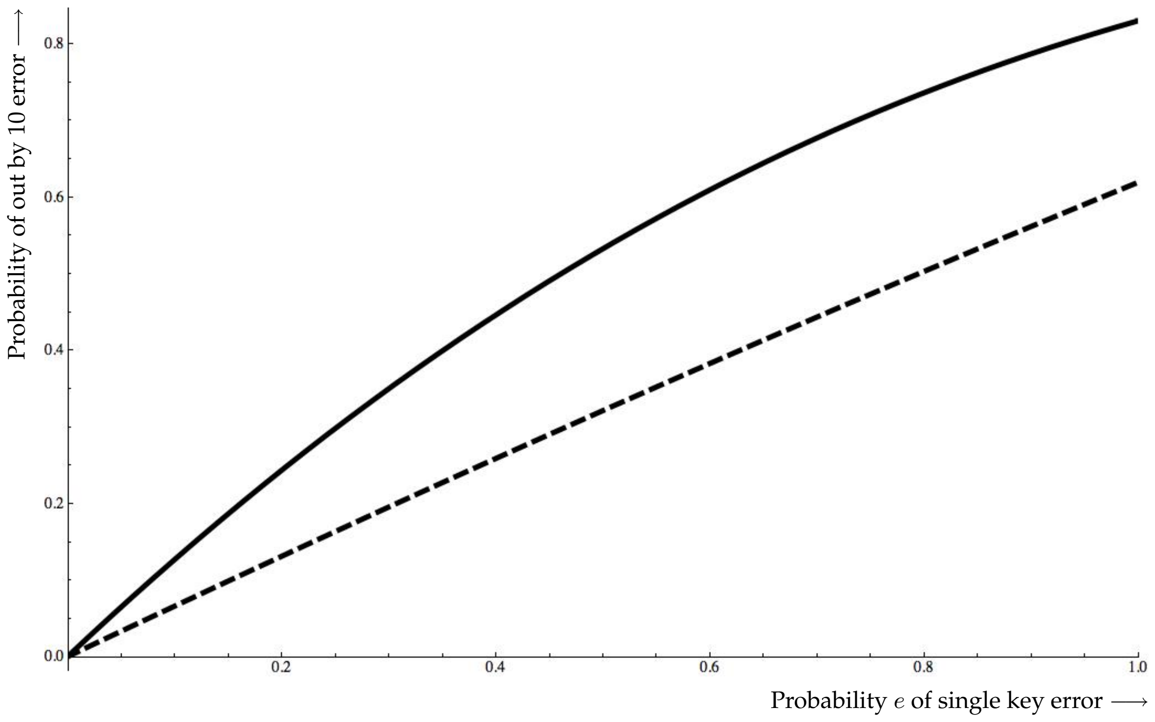

Figure 2: **New user interface blocks errors.** Plot of probability of an out by 10 error against probability of single key errors generated by *Mathematica*; the more likely a user is to make keying errors, the more likely an out by ten error. As in Figure **??**, the solid line shows the behaviour of a simulated calculator-type device; the dashed line shows the reduction in out by 10 errors by blocking syntax errors. The range of numbers covered in this plot is 0.1 to 99.9 and $k = 10$.

1.4.1 Mathematica code and results

The complete documented *Mathematica* code and results are available on the online material for this article.

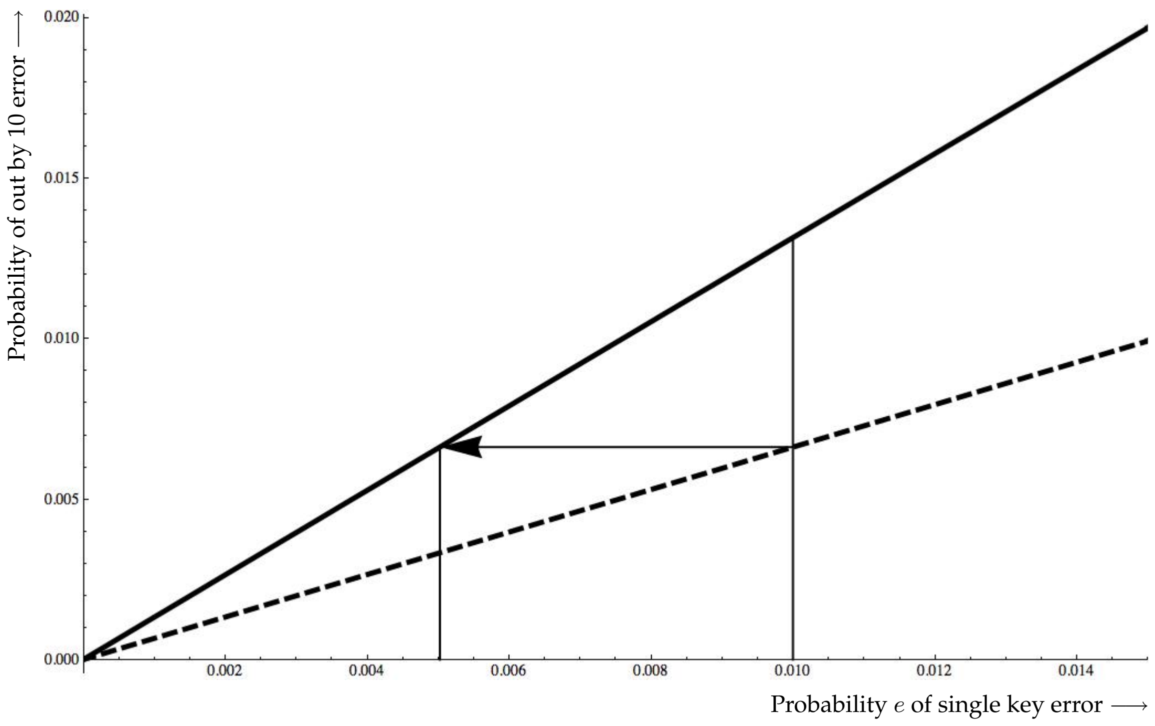

Figure 3: **The potential impact of blocking errors.** Enlargement of graph from Figure **??** to show slope at small e; as before the solid line shows the behaviour of a simulated calculator-type device; the dashed line shows the reduction in out by ten errors by blocking syntax errors. The graph illustrates well how a user with a given e (here 0.01) has their effective error rate approximately halved; since the graph is approximately linear, this is also equivalent to halving their keying error rate, e. Normally, reducing e by training or more careful checking becomes increasingly costly the smaller e, but the technique here has advantages even as $e \to 0$.