Ada Lovelace's Calculation of Bernoulli's Numbers

Rainer Glaschick, Paderborn, Germany
(rainer@glaschick.de)
http://rclab.de/analyticalengine/
2016-09-29

1. Introduction

In 1843, an article with the title *Sketch of the Analytical Engine...* [ML1843] was published, that contains an extensive and on its level fairly complete description of Charles Babbage's plans for a fully automatic computing engine. Its name has been motivated by the aim that it should be able to numerically solve all computations in the field of mathematical analysis (p.688):

> ... all the Operations of analysis come within the domain of the engine.

The article consists of a translation of a report by L.F. Menabrea, originally written in French, and supplements designated as *Note A* to *Note G*. The author was Augusta Ada King, Countess of Lovelace, who signed the *Notes* not by name, only with her initials *A.A.L.* — for the reasons her biographies should be consulted, that are also the place to the never-ending discussion about her mathematical ability, while this article strongly supports her excellence.

The closing *Note G* got much attention, because the corresponding table now is regarded the first fairly complex program ever. The use of machine instructions, variables and a conditional loop is easily seen; but it is instructive to look into the very details of the program — which few people might have done, not only because several typos are not widely known (the most complete and reliable source seems to be [MCK1989], in particular see p. 159, ft), but also because the program needs a feature, that was not considered in the engine at the time the article was written. Probably this was more clearly seen by Ada Lovelace than by Babbage himself.

There are strong indications that the machine could not simply overwrite variables, never explicitely mentioned but implicitely honored by Ada Lovelace as well as Charles Babbage, and, as known so far, not obeyed by all that did build an emulator.

The reader should be very careful not to take things for granted that are common today; nearly everybody, myself definitely included, had to find out that he did so at some place. This concerns in particular the assumption that variables can be easily overwritten, and that operation cards are used to perform an operation, once the operands are loaded.

2. The *Analytical Engine*

As no *Analytical Engine* was ever built, there is no such thing as **the** *Analytical Engine*. The plans were continuously modified and improved; and because there was never an attempt to build at least a small prototype, there was no motivation to fix a design.

Nevertheless, plans 16, 25 and 28 could be seen as milestones. Plan 25 is dated from 1840 and probably closest to the concept that Babbage introduced in Turin and which was the basis for the work of Menabrea and thus Ada Lovelace. That version was also engraved and given away by Babbage; it is today the most used graphical view. Loops are not present and are only found in Plan 28 from 1845 (see Bromley in [AB2000], p.6 bottom to p.7 top), significantly later than Ada Lovelace's article.

The article is obviously on a generic level and not specifically linked to a certain version. E.g. it is not mentioned that a machine according to Plan 25 would produce a double precision result for a multiplication, thus the operations were not just the plain four given in the article.

In fact the nature of the machine is presented by examples, without presenting the model of the machine explicitly. That would not have been possible anyhow, as the concept of the *Analytical Engine* was completely new and no terminology existed for it. For today's reader, however, that is familiar with electronic computers, the hypothetical machine can be sketched coarsely. It will be called an *abstract machine*, as it is an abstraction from various plans, and, as is the principle for abstract machines, omit several technical details.

2.1. The abstract *Analytical Engine*

The abstract Engine implied by Ada Lovelace has:

- a calculating unit (the *mill*) for the four basic calculations
- a storage unit (the *store*) for holding numeric values in variables, also called columns
- a control unit, using punched cards for programming

A calculation step transfers two operands from the store to the mill, calculates the result, and sends it back to the store. Selection of the operands and the operation was done by different punched cards. It follows that normally one operation card and three variable cards were used, two for reading the operands and one for storing the result.

To understand *Note G*, a three-address-machine with four operations is sufficient:

```
Vx + Vy -> Vr
Vx - Vy -> Vr
Vx × Vy -> Vr
Vx ÷ Vy -> Vr
```

where `Vx` and `Vy` are the operand variables and `Vr` is the variable of the result.

In the example of the Bernoulli numbers in *Note G* operations are repeated in a loop, but the mechanism is not explained. The Table only contains a descriptive text and no symbolic indication.

To allow this, a conditional repetition, that goes back a predefined number of operations and variable cards if the result in the mill is not zero, is written as:

```
? ≠ 0 : => k
```

The letter `k` represents the number of the operation that shall be executed next, although a real machine would use the number of cards to rewind.

Note that Menabrea wrote (S.685):

> These new cards may follow the first, but may only come into play contingently upon one or other of the two circumstances just mentioned taking place.

and refers to zero-crossings and overflows, that allow to activate alternative program parts.

According to the current knowledge, there were no plans for loops and conditional jumps of a maturity similar to the arithmetic operations, so that the above conditional backward jump must be seen as very generic.

Note that using Jaquard cards for weaving, only a linear sequence is used; all repeats are unraveled, which explains the enormous number of cards cited in the article.

3. The program

To demonstrate the exceptional power of the *Analytical Engine*, the last example in *Note G* explains the calculation of Bernoulli's numbers. They were probably well known to every professional mathematician, as Ada Lovelace does not bother to explain them or motivate their use. For today's readers, additional information is contained in the appendix. To follow the next chapters, this information is not necessary, if the reader just accepts the equations given.

The example in Note G is preceeded by simpler ones, starting by calculating the two unknowns from the six coefficients in two equations, already in the translated part on p. 681, i.e. also in the french original written by Menabrea, which corresponds to unpublished notes by Babbage [CB1837] dated 1837. While these documents still need in-depth analysis, the first impression is that the tables used by Menabrea and Lovelace contain more information than Babbage's ones. Nevertheless, the first simple program ever was written down by Babbage and published by Menabrea; and it shows the sequential control of calculations by cards as well as the reuse of intermediate values, the latter being the primary motivation of Babbage to abandon the Difference Engine. However, the example in Note G is much closer to what we consider a non-trivial computer program today.

3.1. The calculations

Starting point for the example are several equations for different methods to calculate Bernoulli's numbers. The equations are marked (1.) to (4.) in the article.

The article claims that the second one is well known, but presumably all were common knowledge for mathematicians of that time (a question that might be studied separately), because the finally used equation (8.) is rewritten from the fourth. It was chosen for *the illustration of the powers of the engine*; moreover, it is the only one that does not use an infinite series and thus avoids discussions on termination and accuracy.

Formula (8.) reads:

$$0 = -\frac{1}{2}\cdot\frac{2n-1}{2n+1} + B_1\left(\frac{2n}{2}\right) + B_3\left(\frac{2n\cdot(2n-1)\cdot(2n-2)}{2\cdot3\cdot4}\right)$$
$$+B_5\left(\frac{2n\cdot(2n-1)\cdot(2n-2)\cdot(2n-3)\cdot(2n-4)}{2\cdot3\cdot4\cdot5\cdot6}\right) + \ldots + B_{2n-1}$$

Evidently it allows, as noted in the article, to calculate B_{2n-1}, once all previous B_{2i-1} have already been calculated.

For this purpose, equation (8.) is represented as:

$$0 = A_0(n) + A_1(n)\cdot B_1 + A_3(n)\cdot B_3 + \ldots + A_{2n-3}(n)\cdot B_{2n-3} + B_{2n-1}$$

The above notation differs from equation (9.) in the article, in that the remark in the article that *A_1, A_3 &c. being ... functions of n ...* is made explicit using functional notation.

Thus the coefficients are (not given in this form in the article):

$$A_0(n) = -\frac{1}{2} \cdot \frac{2n-1}{2n+1}$$
$$A_1(n) = \frac{2n}{2}$$
$$A_3(n) = A_1(n) \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4}$$
$$A_5(n) = A_3(n) \cdot \frac{2n-3}{5} \cdot \frac{2n-4}{6}$$

For a given n, the values can be calculated from already calculated ones of the same n, with the significant difference to Bernoulli's numbers, that only the last one is needed and only for the current round.

So the equation for the next of Bernoulli's numbers has the finally used form:

$$-B_{2n-1} = A_0(n) + A_1(n) \cdot B_1 + A_3(n) \cdot B_3 + ... + A_{2n-3}(n) \cdot B_{2n-3}$$

Obviously Ada Lovelace has overlooked the negative sign for B_{2n-1}, because it is — wrongly — not honored in the table; so the formulas are a bit more bulky than necessary.

An example for B_5 and B_7:

$$-B_5 = A_0(3) + A_1(3) \cdot B_1 + A_3(3) \cdot B_3$$
$$-B_7 = A_0(4) + A_1(4) \cdot B_1 + A_3(4) \cdot B_3 + A_5(4) \cdot B_5$$

To calculate the next of Bernoulli's numbers, a weighted sum of the so far calculated numbers is built, in which the weights (the $A_i(n)$) must be re-calculated in each round, and are not needed for the next round. Thus a single variable is sufficient to keep track of the value needed.

To better understand the algorithm, it is written in pseudo-code in contemporary notation as follows:

```
B[1] = 1/6
for n = 2 to 10
    n2 = 2*n
    k = n2 - 1
    a0 = - (n2 - 1) / (n2 + 1) / 2
    a1 = n2 / 2
    x = a0 + B[1] * a1
    aj = a1
    for j=3 to k-1 step 2
        aj = aj * (n2 - (j-2)) / j
        aj = aj * (n2 - (j-1)) / (j + 1)
        x = x + B[j] * aj
    B[k] = -x
```

A modern loop with a test at the entry is used instead of the test at the end; the latter was common in machine language programming from the beginning. So in the above code B_3 can already be calculated this way. For better comparison, the signs and the suboptimal calculation of $A_1 = 2\frac{n}{2}$ instead of $A_1 = n$ was kept.

3.2. The tabular program

For the systematic notation of her examples, Ada Lovelace extended a tabular representation (similar to the form used by Babbage already 1837), which also documents the processing steps. The program proper is contained in just the first columns; the remaining columns document the processing and the use of variables (for typos, see the appendix).

The variables are written in a special notation. Mathematicians, then as well as today, consider a variable to have a fixed value (in its context). The new and innovative element of variables that can be replaced with different values — rightly described at length in the article — requires a notation, where (indexed) variables in the table always have the same value. Consequently, the lower index is the numerical storage name (not address, there is no address arithmetic in the machine), and the upper prefixed index indicates the state throughout the calculation. When the upper index is 0, the contents of the variable is always 0: ${}^0V_i = 0$.

For example, in line 2 the variable V_1 is subtracted from V_4, and the result stored back to V_4, which could have been written as:

$${}^1V_4 - {}^1V_1 = {}^2V_4$$

The equals sign is present in the tables from by Menabrea, but not in the last table in Note G.

Oddly enough, it appears in the table on p.681 in the column headed *Indication of change of value on any column*. Menabrea writes V_0 sur V_0, V_4 id. V_4, etc. when the variable is restored, and V_4 id....., when it is zeroed after read. Lovelace changed this in the translation to ${}^1V_0 = {}^1V_0$ etc. if the variable is restored, and ${}^1V_4 = {}^0V_4$, if the variable is zeroed. This is rather confusing under the assumption that the upper indices were created to avoid equations like $x = x + y$, but instead (mis-) used here in the — nowadays common — sense of *is replaced by*, but in inverse order. Also, the term *any variable* obviously means *any input variable*, and in this table a variable is never operand and result in the same line.

For this table on p.681, this column thus just tells, in a confusing notation, if the operands (input variables) are retained or zeroed.

The situation for the last table (Note G, Bernoulli's numbers) is nearly the same (ignoring the modified table header *Indication of change in the value on any Variable*), in that the modification of the operands (*Variables acted upon*) is shown. The exception is line 21. If the pattern on the other lines is followed, the second line should indicate the change in the operand, not the result, i.e. read ${}^5V_{11} = {}^3V_{11}$ instead of ${}^0V_{12} = {}^2V_{12}$, and in the previous column of the table ${}^0V_{12}$ should be replaced by ${}^2V_{12}$. Also, the zero in the column for V_{11} must be dropped, as the value is needed if the loop is repeated.

The notation with the (constant) upper indices comes to its limits (or requires variables as upper indices and additional lines to keep track of these), once sequences of operations are repeated, because then the upper indices are dependent on the run.

For the last line, Ada Lovelace found the — somewhat lame — subterfuge that the value is for the next run, and thus has the upper index of 1 instead of 2. which is also applied above for line 21. For the inner loop, she just explains it verbally in the text.

In the table header in the columns for the variables, the initial values are given in boxes (except the last column, where the result is denoted).

3.3. The columns of the table

In the first column the operations are simply numbered; these numbers are for reference only and not (necessarily) punched on cards.

In the second column the arithmetic operation is given.

In the third column, the operands are given, in infix notation with the operation, which must be the same as in column 2.

In the fourth column, the variable that receives the result is denoted, thus its upper index is incremented compared to the previous use in this column (see above for exceptions).

The fifth column tells about the modification of the operands (which is also rather strange for us today), except the first case: As explained above, both sides only differ if an operand is either used for the result, or cleared on load.

The next columns are further comments, that document the flow of the calculation, where the sixth column (*Statement of Results*) just comments the partial result obtained so far, i.e. the current result of the mill, expressed in symbolic mathematical form.

The other columns show the contents of the variables. The circle and three numbers in the header represent the sign and variable number; this is used in other tables and not significant here. The box in the header line gives the initial values; for V_1, V_2 and V_3 number cards (not yet mentioned) are used; for V_{21}, V_{22} and V_{23} these are possibly already calculated results of previous rounds.

Whenever a value changes, the equation for the new value is given in the column, or zero, if the variable is zeroed on load.

Between operation 23 and 24, the — not harmless — sentence is found, that will be explained below:

> Here follows a repetition of Operations thirteen to twenty-three

The table of *Note G* can thus be presented in symbolic form as (comments added without delimiter):

```
 1:  V2  × V3  -> V4, V5, V6     2n
 2:  V4  - V1  -> V4             2n-1
 3:  V5  + V1  -> V5             2n+1
 4:  V4  / V5  -> V11            (2n-1)/(2n+1)
 5:  V11 / V2  -> V11            (2n-1)/(2n+1)/2
 6:  V13 - V11 -> V13            A0 = - ...
 7:  V3  - V1  -> V10            n-1 = 3
 8:  V2  + V7  -> V7             2+0 = 2, i.e. V2 -> V7
 9:  V6  / V7  -> V11            A1 = 2n/2 = Aj
10: V21 * V11 -> V12             B1*A1
11: V12 + V13 -> V13             V12 -> V13: x = V12
12: V10 - V1  -> V10             j-1 = 2
13: V6  - V1  -> V6              2n-1
14: V1  + V7  -> V7              2+1=3 (j)
15: V6  / V7  -> V8              (2n-1)/3
16: V8  * V11 -> V11             Aj * (2n-1)/3
17: V6  - V1  -> V6              2n-2
18: V1  + V7  -> V7              3+1=4 (j+1)
```

```
19: V6  / V7  -> V9            (2n-2)/4
20: V9  * V11 -> V11           Aj * (2n-2)/4 -> Aj = A3
21: V22 * V11 -> V12           A3 * B3
22: V12 + V13 -> V13           x + A3*B3 -> x
23: V10 - V1  -> V10           j-1 = 1
    ? ≠ 0 : ~> 13              repeat if j>0
24: V24 - V13 -> V24           B7 = -x
25: V1  + V3  -> V3            n+1 -> n
```

Compared to the original table, the following corrections were applied:

- in line 4, the operands are exchanged
- in line 24, the negative value is formed

Also, the erasing load of variables is not used here, values are just overwritten like today, see below for details.

The initial values of the variables — there are no literals with operations — are:

```
V1 = 1                          constant 1
V2 = 2                          constant 2
V3 = 4                          n
V21 = 0.166667                  B1 = 1/6
V22 = -0.03333                  B3 = -1/30
```

3.4. The flow

The table shows the calculation of B_7, provided that the numerical values for B_1, B_3 and B_5 are already contained in the resp. variables. This can be done by number cards. Alternatively, previous runs could be used, but in this case the calculation must be terminated, as written in the article, after operation 7 or 12, and the value stored like in line 24, but in a different variable. As the value of B_7 is to be calculated, the variable V_3 must be set to the value 4.

The calculation of B_7 is a linear program flow upto and including operation 21, which should be clear from the previous explanations:

- with operation 6, the value of A_0
- with operation 11, the value of $A_0 + A_1 \cdot B_1$
- with operation 22, the value of $A_0 + A_1 \cdot B_1 + A_3 \cdot B_3$

is calculated and the latter deposited in V_{13}. Operation 23 is the same as operation 12 and prepares the calculation of A_4 from A_3 in the same manner, in which A_3 was calculated from A_1 using the operations 13 to 20.

In the line between 23 and 24 it is indicated that operations 13 to 23 are to be repeated. Thus, it is a jump back to operation 13, i.e. a program loop; consequently, no new operation numbers are used. However, there is a small, but very relevant difference: in operation 21 instead of V_{22}, which contains B_3, the variable V_{23}, namely the value of B_5, has to be used. This would be called today an indexed memory access or a linear address advance. It is commented rather tersely by Ada Lovelace and will be explained later.

After the repeat — not visible in the table — variable V_{10} is reduced to zero with operation 23, so the conditional backward jump will not jump.

In step 24, the composed result B_7 in is V_{13} and sign-inverted transferred to V_{24}.

In step 25 the variable V_3 is incremented by one, which prepares the calculation of B_9;

now the program could be restarted, to calculate the next of Bernoulli's numbers; this could even be done by another surrounding loop, as no manual intervention is required.

3.5. Memory peculiarities

At three places (p.677, p.680 and p.708) it is mentioned that reading a variable leaves a zero value in the store, which must be restored if the value is required for further operations. Thus there are two variants of *Variable Cards* that transfer a value from the store to the mill, which Lovelace called *Retaining Supply cards* and *Zero Supply cards*. Surely many readers have regarded this as insignificant optimisation and did not give it further attention.

Alan Bromley [AB1998] writes on p.31:

> This form of storage exhibits a destructive readout; following the read operation, all figure wheels will stand at zero, irrespective of the digit originally stored, and the number originally stored is lost.

Curiously enough, all relevant technologies for main memory (core memory and dynamic semiconductor) also use a destructive readout.

Babbage's solution is remarkably elaborate, as Bromley continues:

> If it is desired not to loose the number, then it must be stored, as it is read, on another set of figure wheels. For this purpose, each figure axis of the mill of the Analytical Engine is provided with two figure wheels in each cage.

Menabra mentioned on p.680, that the secured numbers are written back while the mill is working:

> ... while the mill is working ... the machine will inscribe them anew on any two columns that may be indicated to it through the cards; and ... there is not reason why they should not resume their former places.

This would make sense if the mill would create two copies of the number received, and send one back to a variable additionally indicated on the card, instead of duplicating each variable[1].

So far the erasing load still is an unimportant optimisation, and using the retaining read could slow down the machine a bit. But thinking about the mechanics, storing a number in a variable that is not zero by just turing the wheels with the number given off by the mill, will result in a digitwise addition and frequent error jam stops, as Babbage had planned to detect accidential overruns and other errors[2].

A deeper inspection of the examples reveals that a value is only stored if the variable was zero before, unless it just supplied one of the operands. At start, all working- and output-variables are considered zero; probably by the same way as the input variables are set. In the examples, variables are only overwritten on p.715 (presented without table) and in Note G. Note G is the only example where numbers are overwritten which did not supply one operand to the current operation; and in all these cases, they are set to zero (by a zero reading) before. In Babbage's programming notes [CB1837], a coarse inspection shows that there are freqently zeroes in the tables, evidently when the value of a variable is no longer needed.

Turning it the other way: The example in Note G as well as Babbage's notes do not write to a variable that is not zero before (or used as operand in the same operation). As the zero reading of a variable is not directly expressed in the table, but with a zero in the variable column for the next operation, when overwriting a variable within the same operation, there is no way to indicate that a zero readout is used.

To store into a non-zero variable, that variable would have to be zeroed first, before the digits received from the mill can be applied. This is certainly extra effort, for which several solutions are possible, including to always clear the destination while the mill is calculating. This measure, however, should have been identified by all the people that so far have analysed the drawings. As Babbage tried to make the store as simple as possible, this additional effort was probably either discarded, or not seen as necessary.

So I come to the conclusion that storing to a variable that is not zero would not be provided for in the Analytical Engine.

Lovelace explained thoroughly the concept of a variable that could set to different value, as this was a new concept at that time. In mathematical formulas variables were — and are! — not changed; if necessary, indices are used, to differentiate similar values; Lovelace uses upper indices for this purpose. So one would expect at least a sidenote that writing to variable would require that it was cleared before, but no such remark has been found.

With professional numerical calculations on paper, values written down are never erased (unless erroneous) or overwritten; new values are written to empty places (or new variables, as in the example p.681). So Ada Lovelace and Babbage might have seen it completely natural that only erased variables are be overwritten.

Ada Lovelace writes on p.707:

> Now the ordinary rule is, that the value *returns* to the Variable; unless it has been foreseen that no use for that value can recur, in which case zero is substituted.

Admittedly, no explicit rule could be found that a variable must be zero before it can be overwritten.

This rule is already obeyed in the first example on p.681, where column 7 indicates — not very straightforward — which variables are erased during readout.

Interesting is the following sentence:

> At the *end* of a calculation, therefore, every column ought as a general rule to be zero, excepting those for results.

Those who have read Alan Turing's *On computable Numbers...*, finds there the indication that at the end of a calculation — and only then — all auxiliary fields are to be erased.

So it is presumed that a 0 is missing in line 24 of the column for V_{13}, as correctly noted in the 5th column.

Using `Zn` instead of `Vn` for an erasing load, the above symbolic notation reads:

```
1:  V2  × V3  -> V4, V5, V6     2n
2:  Z4  - V1  -> V4             2n-1
3:  Z5  + V1  -> V5             2n+1
4:  Z4  / Z5  -> V11            (2n-1)/(2n+1)
5:  Z11 / V2  -> V11            (2n-1)/(2n+1)/2
6:  Z13 - Z11 -> V13            A0 = - ...
7:  V3  - V1  -> V10            n-1 = 3
8:  V2  + Z7  -> V7             2+0 = 2, i.e. V2 -> V7
9:  V6  / V7  -> V11            A1 = 2n/2 = Aj
10: V21 * V11 -> V12            B1*A1
11: Z12 + Z13 -> V13            V12 -> V13: x = V12
```

```
12: Z10 - V1  -> V10          j-1 = 2
13: Z6  - V1  -> V6           2n-1
14: V1  + Z7  -> V7           2+1=3 (j)
15: V6  / V7  -> V8           (2n-1)/3
16: Z8  * Z11 -> V11          Aj * (2n-1)/3
17: Z6  - V1  -> V6           2n-2
18: V1  + Z7  -> V7           3+1=4 (j+1)
19: V6  / V7  -> V9           (2n-2)/4
20: Z9  * Z11 -> V11          Aj * (2n-2)/4 -> Aj = A3
21: V22 * V11 -> V12          A3 * B3 !! index mod !!
22: Z12 + Z13 -> V13          x + A3*B3 -> x
23: Z10 - V1  -> V10          j-1 = 1
    ? ≠ 0 : ~> 13             repeat if j>0
24: Z24 - V13 -> V24          B7 = -x
25: V1  + Z3  -> V3           n+1 -> n
```

In general, this is noticed in the fifth colum right side with an upper index of zero, or correspondingly a zero in the column for the variable.

Note that in line 21, the variable V_{11} (containing the current A_j) is not cleared on load, although this is indicated in the corresponding column. If the loop is repeated, the value is needed. Clearing this working variable has to be added to line 25.

If loops would have been implemented and the machine used, it would have been an additional effort to ensure that variables like this are cleared after the loop terminates. Lovelace claims with reference to line 25, that clearing a variable can be done by Variable cards. However, this would require to reset the variable, an extra effort for which no hint has been observed until now. On the other hand, there is also no indication that a global reset that would clear all variables was ever considered. Such a global reset might have been a real challenge to the (mechanical) power supply, so clearing the required variables by Variable Cards is more probable.

Note that if a variable at the same time supplies a value and receives the result, first example in line 2, an erasing load has to be used. No indication for this rule has been found; probably neither Ada Lovelace nor Babbage were aware of this peculiarity — which would have become apparent if ever a machine had been built.

Moreover, the simulator made by Charles Walker ([http://www.fourmilab.ch]) does silently overwrite any value, as does any simulator known so far. It can be seen from the above example, that zero variable cards have to be used quite often, but Walker's examples rarely use them. This is one case where the seamless overwriting of variables is taken for granted from our present experience.

The simulator, fortunately published as frees source by Walker, was changed to allow to reject overwriting non-zero variables, and the above modified version with frequent zero readings delivered the correct result.

Note that the ENIAC, built more than hundred years later, would have had the same problem, as it used the electronic equivalent of a rotating wheel, a shift register. They solved the problem deftly by rotaing exactly ten times independent of the value stored, and send the remaining pulses after the register has changed from 9 to 0, i.e. at the end of the cycle. However, there, as the shift register was build from ten interconnected flip-flops, clearing a register was simple, so there was no need for a zero readout.

3.6. Indexing and address incrementing

It had be mentioned that for the calculation of Bernoulli's numbers with the recursion formula chosen by Ada Lovelace, a very compact program was possible, but went beyond the machines as conceived at that time.

In particular, there is no such concept of a memory address in the machine; the numbers of the variables are just names, as Bromley in [AB2000] p.12 cites Maurice Wilkes. Nevertheless, Lovelace has seen the need for such a device, while Babbage apparently never solved this architectural challenge, as he did not find a general solution to solve sets of equations, as noted by Bromley in [AB1998], p.44.

In using today's knowlege, we note that there is a difference between address advancement, now known as auto-increment index register, and random indexing.

In the case of address advancement, a value copied from a card to an index register can just be used or incremented, depending on the card used. A special card could be used to denote the use of the index register instead of a normal address. The effort is limited; only indices that were provided by cards, are to be used.

More difficult is a random indexing, in which the contents of any variable can serve as an address or index (added to an address), as the numbers have to be decoded. Until plan 25 the holes in the variable cards were assigned each to a variable, i.e. a 1-of-n coding was used. When reading a variable, only one hole could be present. If more than one were punched, serious jam would be the result (at least if the values are different). When writing, it allows the simultaneous transfer of more than one variable, as used in line 1 of the table. Thus, a decimal number stored in a variable would have to be decoded. For this purpose, the variables could be grouped in tens, and a 1-to-10 decoder for each digit would be sufficient; a mechanically feasible task.

Relatively late Babbage has considered a binary coding of the memory locations, as he realised that to access 1024 variables, only 10 holes would be necessary. A binary column selection would have been required, which is not overly complicated, I presume. However, it would conflict seriously with any decimal indexing, as then the complete decimal number had to be converted to binary by hardware[3]. This is a real barrier and perhaps another reason to use binary instead of decimal arithmetic[4].

Note that large memory can be accessed only via a memory matrix with acceptable effort. For a matrix with 1024 memory cells, 32 rows and columns have to be driven, a significantly less effort for decoding than to decode to 1024 bars.

4. Historical context

It has already been mentioned that in line 21 in the first round of the loop the variable V_{22} (containing B_3) is used, while in the second round, the value for B_5 from V_{23} is required. Correspondingly, in the calculation of B_9, within the loop three different values are required in line 21, etc. So in the above pseudo code, `B[j]`, an indexed array, was used.

Ada Lovelace writes upon this (p.729):

> The only exception to a *perfect identity* in *all* the processes and columns used, for every repetition of Operations (13..23) is, that Operation 21 always requires one of its factors from a new column, and Operation 24 always puts its result on a new column.

This is not only not representable in the formalism used in the table, it was unsolved at that time. Ada Lovelace did see the problems, she continues:

> But as these variations follow the same law at each repetition, (Operation 21 always requiring its factor from a columns *one* in advance of that which is used the previous time, and Operation 24 always putting its result on the column *one* in advance of that which received the previous result), they are

> easily provided for in arranging the recurring group (or cycle) of Variable-cards.

She correctly notes that no random access is required, only a linear index increment. No such insight has is apparently contained in Babbage's writings. Although possible, its construction is far from simple.

As could be seen from a letter published by Fuegi and Francis ([JFJF2003], p.21) Ada Lovelace had been reluctant to be too clear:

> ... but have simply indicated that as the associations follow a regular rule, they would be easily provided for. I think I have done it admirably and diplomatically.

Alan Bromley writes ([AB1998], p.44 left):

> With hindsight, we can note that in the Analytical Engine (at least until 1840), Babbage did not possess the variable-address concept; that is, there was no mechanism by which the machine could, as a result of a calculation, specify a particular variable in the store to be used of an operand for an instruction.

One could interpret Ada Lovelace' remark such as operation- and variable cards were used in a loop, but to fetch these cards from another batch, so to provide 200 cards for 100 of Bernoulli's numbers.

Naturally the number of several thousands of variables appraised in the article and announced by Babbage, were unrealistic. But here the principle counts, because Ada Lovelace writes in the footnote on p.729:

> ... that during the processes for the computation of *millions* of these Numbers, no other arbitrary modification would be requisite in the arrangements, excepting the above simple and uniform provision for causing one of the data periodically to receive the finite increment unity.

Ada Lovelace postulated thus — even if here only for the operation in line 25, that prepares the calculation of the next number — that a single program with a fixed number of punched cards could calculate arbitrary many of Bernoulli's numbers automatically, i.e. without operator intervention (provided that enough variables are present and the precision is sufficient).

According to Bromley, Babbage tried to find a general solution for set of linear equations, but was not successful (p.44 left):

> At the end, the notation "Not similar" signifies that Babbage hat not found the loop structure he sought.

This makes the much cited phrase in Babbage's autobiography [CB1864] much clearer (p.136):

> We discussed together the various illustrations that might be introduced: I suggested several, but the selection was entirely her own. So also was the algebraic working out of the different problems, except, indeed, that relating to the numbers of Bernoulli, which I had offered to do to save Lady Lovelace the trouble. This she sent back to me for an amendment, having detected a grave mistake which I had made in the process.

While it is still unknown to me what the *grave mistake* was, Ada Lovelace writes (cited from Fuegi and Francis [JFJF2003], p.21):

> I have ... touched on the only departures from *perfect* identity which *could* exist during the repetitions of (13 ... 23); and yet have not *committed* myself by saying if the departures would require to be met by the introduction of one or more new cards or not; ... I think I have done it admirably and diplomatically ..."

It is thus clear, that the example of the calculation of Bernoulli's numbers exceeded the features of the machines as planned, but that Ada Lovelace was so confident on its usefulness, that she stuck to the published text and the example, possibly in order to influence the construction of a real machine — which she definitely tried to support. At least she might have assumed that the example of Bernoulli's numbers would be impressive for the professional reader.

5. Appendix

5.1. Bernoulli's Numbers

In his 1713 posthumous published book *Ars Conjectandi*, Jakob Bernoulli (1654-1705) gave formulas for (finite) sums of powers:

$$1+2+3+4\ldots+n=\frac{1}{2}n^2+\frac{1}{2}n$$

$$1^2+2^2+3^2+4^2+5^2\ldots+n^2=\frac{1}{3}n^3+\frac{1}{2}n^2+\frac{1}{6}n$$

$$1^3+2^3+3^3+4^3+5^3\ldots+n^3=\frac{1}{4}n^4+\frac{1}{2}n^3+\frac{1}{4}n^2$$

$$1^4+2^4+3^4+4^4+5^4\ldots+n^4=\frac{1}{5}n^5+\frac{1}{2}n^4+\frac{1}{3}n^3-\frac{1}{30}n$$

$$1^5+2^5+3^5+4^5+5^5\ldots+n^5=\frac{1}{6}n^6+\frac{1}{2}n^5+\frac{5}{12}n^4-\frac{1}{12}n^2$$

The sum

$$S_p(n)=1^p+2^p+3^p\ldots n^p=\sum_{k=1}^{n}k^p$$

is apparently a polynomial of grade (p+1) in n, where the first four coefficients are rather neatly structured:

$$S_p(n)=\frac{1}{p+1}n^{p+1}+\frac{1}{2}n^p+\frac{p}{12}n^{p-1}+0\cdot p^{n-2}+\ldots$$

Bernoulli could give the coefficients as a permutation number and a part independent from p:

$$S_p(n)=\frac{1}{p+1}\sum_{k=0}^{p}B'_k\cdot\binom{p+1}{k}\cdot n^{p+1-k}$$

For this variant used today (marked by a tick to the letter B), Bernoulli's numbers have the following values (the other odd indexed numbers are 0):

n	0	1	2	4	6	8	10	12	14	16
B'_n	1	-1/2	1/6	-1/30	1/42	-1/30	5/66	-691/2730	7/6	-3617/510

The common definition today is

$$\frac{x}{e^x - 1} = \sum_{k=0}^{\infty} B'_k \frac{x^k}{k!}$$

and is proved for the sums of powers via the Taylor series for e^x.

Traditionally (and by Ada Lovelace), the following definition is used:

$$\frac{x}{e^x - 1} = 1 - \frac{x}{2} - \sum_{k=1}^{\infty} B_{2k-1} \frac{x^{2k}}{(2k)!}$$

with differently indexed values, that are 0 for even indices:

n	1	3	5	7	9
B_n	1/6	-1/30	1/42	-1/30	5/66

Bernoulli's numbers were used and detected in the 19th century in several cases and applications. Thus, different methods for their calculation were found.

They were useful for the efficient approximation of functions (and thus directly related to the Difference Engine), so were attractive examples. There was no practical relevance for their automatic calculation, as the few numbers needed could be calculated by hand and did not at all need a machine.

Only in the 20th century better and more efficient algorithms were found to calculate Bernoulli's numbers. Also, methods like the Tshebysheff approximation were not yet found when Ada Lovelace wrote her paper.

In her paper, she firstly mentioned that she did not use the equations (1.) to (3.), as these would not demonstrate the features of the machine adequately. One could add, that those equations used for each value an infinite series, and that the termination when the required precision was reached, would introduce an additional complexity, that was better avoided. Additionally, the method chosen by Ada Lovelace uses very few program steps, so that it can be displayed nearly completely.

5.2. Hints and errors

When reading the *Sketch of the Analytical Engine..* ([ML1843]), note that instead of nested brackets, overbars are used, e.g. in equation 4 on p.716.

There are also a few minor errors that are in the domain of typographical errors:

- The well-discussed case of the cos on p.637, where the phrase *when the cos of n=∝ has been foreseen* should clearly read *when the case of ...*
- In the first equations in *Note E* (p. 712), many reprints correctly substitute the apostrophe by an upper digit 1. Also, the equations refer to p.679, not to p.684
- Also in *Note E* on p.714, in equation 3 is missing a `cos`, it must read:
  $$\cos n\theta \cdot \cos\theta = \frac{1}{2}\cos[(n+1)\theta] + \frac{1}{2}\cos[(n-1)\theta]$$
- In *Note G* in the table to calculate Bernoulli's numbers, in line 4 the operands must be exchanged: V_4 must be divided by V_5.
- In the same table in line 21, in the result column it must read ${}^2V_{12}$ instead of ${}^0V_{12}$.
- In the same line 21, the zero in column V_{11} must be dropped. If the loop is repeated, the value is needed in the next round.

- In the same line 21, in the 5th column (*Indication...*) the second line ${}^0V_{12} = 2V_{12}$ is exceptional, as elsewhere in this column only *Variables acted upon* from the 3rd column are used; systematically, it would be ${}^5V_{11} = {}^3V_{11}$.
- In line 24 the negative value of V_{13} must be transferred to V_{24}, i.e. $V_{24} - V_{13}$ must be calculated. Also, in the 6th column it must read $= -B_7$
- In the same line 24, in the column for V_{13} a zero must be entered; this is correctly done in the 5th column, because not further used values should be cleared.
- In line 25 in the result column and the next one right of the equals sign, one might assume it should read ${}^2V_3$ instead of ${}^1V_3$; however, Ada Lovelace argues that the values are for the next round of calculation and thus would have the upper index 1, because they are start values.
- The last signature (p.731) is not A.A.L., but A.L.L., which is most probably just a typeset error.

No publication is known so far that shows or corrects all these issues; the most complete one seems to be [MCK1989]. Some discuss the first one at length, having not observerd any of the — more relevant — others. Maybe this is an indication that only a few really followed her description to the detail. As these are glitches of a kind that is found in many articles, they do not disqualify Ada Lovelace; on the contrary, they are so small that they show that she has fully mastered the subject, and nobody who has not dealt to this detail may not judge her skills.

5.3. Numerical values of a calculation

$$A_0(1) = -\frac{1}{2} \cdot \frac{1}{3} = -\frac{1}{6}$$

$$-B_1 = A_0(1)$$

$$B_1 = \frac{1}{6}$$

$$A_0(2) = -\frac{3}{10}$$

$$A_1(2) = 2$$

$$-B_3 = A_0(2) + A_1(2) \cdot B_1 = -\frac{3}{10} + \frac{2}{6} = \frac{-9+10}{30} = \frac{1}{30}$$

$$B_3 = -\frac{1}{30}$$

$$A_0(3) = -\frac{5}{14}$$

$$A_1(3) = 3$$

$$A_3(3) = 3 \cdot \frac{5}{3} \cdot \frac{4}{4} = 5$$

$$-B_5 = -\frac{5}{14} + \frac{3}{6} - \frac{5}{30} = \frac{-5+7}{14} - \frac{1}{6} = \frac{1}{7} - \frac{1}{6} = -\frac{1}{42}$$

$$B_5 = \frac{1}{42}$$

$$A_0(4) = -\frac{7}{18}$$

$$A_1(4) = 4$$

$$A_3(4) = 4 \cdot \frac{7}{3} \cdot \frac{6}{4} = 14$$

$$A_5(4) = 14 \cdot \frac{5}{5} \cdot \frac{4}{6} = \frac{28}{3}$$

$$-B_7 = -\frac{7}{18} + \frac{4}{6} - \frac{14}{30} + \frac{28}{3 \cdot 42} = \frac{-7+12}{18} - \frac{7}{15} + \frac{2}{9} = \frac{5}{18} + \frac{-21+10}{45} = \frac{25-22}{90} = \frac{3}{90}$$

$$B_7 = -\frac{1}{30}$$

5.4. Use of variables

The variables are used as follows:

```
V1  Constant 1
V2  Constant 2
V3  n
V4  2n-1
V5  2n+1
V6  2n, 2n-1, 2n+1
V7  j
V8  2n-1/3
V9  2n-2/4
V10 n-3
V11 Accumulator
V12 Bj*Aj
V13 Accumulator
V21 B1
V22 B3
V23 B5
V24 B7
```

5.5. Testing Programs

The pseudocode was tested with the following program:

```
\(  Print Bernoulli's numbers
    This is the algorithm used in Note G of Ada Lovelace paper.
    The progamming language used is unpublished.
\)

main (parms):
    n =: 10
    ? parms[1] ~= ()
        n =: integer from string parms[1]
    print n bernoullis

print (max) bernoullis:
    b =: []
    b[1] =: 1/6
    print 'B[1]=' _ b[1]
    ?# n =: from 2 upto max
      k =: 2*n - 1
      n2 =: 2*n
      a0 =: - (n2 - 1) / (n2 + 1) / 2
      a1 =: n2 / 2
      x =: a0 + (b[1] * a1)     \ V13
      aj =: a1
      ?# j =: from 3 upto k-1 step 2
          aj =* (n2 - (j-2)) / j
          aj =* (n2 - (j-1)) / (j + 1)
          x =+ (b[j] * aj)
      b[k] =: -x
      print "B[" _ k _ "]=" _ b[k]
```

John Walker (http://www.fourmilab.ch/babbage/cards.html) provides a simulator in Java and uses a symbolic Notation for the program- and variable cards, which had be used to check the example.

For that purpose, the program was condensed to:

```
        A set decimal places to 10
        A write numbers with decimal point
        N1 1.0
        N2 2.0
        N3 4.0
        N21 0.16666666666666666666667
        N22 -0.03333333333333333333333
        N23 0.0238095238
    1:  V2  × V3  -> V4, V5, V6     2n
    2:  Z4  - V1  -> V4             2n-1
    3:  Z5  + V1  -> V5             2n+1
    4:  Z4  / Z5  -> V11            (2n-1)/(2n+1)
    5:  Z11 / V2  -> V11            (2n-1)/(2n+1)/2
    6:  Z13 - Z11 -> V13            A0 = - ...
    7:  V3  - V1  -> V10            n-1 = 3
    8:  V2  + Z7  -> V7             2+0 = 2, i.e. V2 -> V7
    9:  V6  / V7  -> V11            A1 = 2n/2 = Aj
    10: V21 * V11 -> V12            B1*A1
    11: Z12 + Z13 -> V13            V12 -> V13: x = V12
    12: Z10 - V1  -> V10            j-1 = 2
    13: Z6  - V1  -> V6             2n-1
    14: V1  + Z7  -> V7             2+1=3 (j)
    15: V6  / V7  -> V8             (2n-1)/3
    16: Z8  * Z11 -> V11            Aj * (2n-1)/3
    17: Z6  - V1  -> V6             2n-2
    18: V1  + Z7  -> V7             3+1=4 (j+1)
    19: V6  / V7  -> V9             (2n-2)/4
    20: Z9  * Z11 -> V11            Aj * (2n-2)/4 -> Aj = A3
    21: V22 * V11 -> V12            A3 * B3
    22: Z12 + Z13 -> V13            x + A3*B3 -> x
    23: Z10 - V1  -> V10            j-1 = 1
    13: Z6  - V1  -> V6             2n-1
    14: V1  + Z7  -> V7             2+1=3 (j)
    15: V6  / V7  -> V8             (2n-1)/3
    16: Z8  * Z11 -> V11            Aj * (2n-1)/3
    17: Z6  - V1  -> V6             2n-2
    18: V1  + Z7  -> V7             3+1=4 (j+1)
    19: V6  / V7  -> V9             (2n-2)/4
    20: Z9  * Z11 -> V11            Aj * (2n-2)/4 -> Aj = A5
    21: V23 * V11 -> V12            A5 * B5
    22: Z12 + Z13 -> V13            x + A5*B5 -> x
    23: Z10 - V1  -> V10            j-1 = 1
    24: Z24 - V13 -> V24            B7 = -x
        P
    25: V1  + Z3  -> V3             n+1 -> n
```

A small script translates the lines with the arrow bigraphs to the input required by the simulator, resulting in 170 cards to calculate B_7.

Futhremore, the simulator was changed to suppress (and flag as an error) all cases where it was tried to overwerite a nonzero variable.

5.6. References

ML1843:

L.F. Menabrea: *Sketch of the Analytical Engine invented by Charles Babbage*, with Notes from A.A. Lovelace; Scientific Memoirs, Vol. III Part XII, London 1843, pp.666-731.
Facsimile online: https://archive.org/stream/scientificmemoir03memo#page/666

PEM1991:
Philip and Emily Morrison: *Charles Babbage and his CALCULATING ENGINES*. Dover Publications 1961.

RS2004:
Helmut Richter, Bernhard Schiekel: *Potenzsummen, Bernoulli-Zahlen und die Eulersche Summenformel*. Univ. Ulm 2004. Online: http://vts.uni-ulm.de/docs/2009/7137/vts_7137_10026.pdf

AB1998:
Allan G. Bromley: *Charles Babbage´s Analytical Engine, 1838*. IEEE Annals of the History of Computing, Vol. 20, No.4, pp.29-45 (1998).

AB2000:
Allan G. Bromley: *Babbage´s Analytical Engine Plans 28 and 28a - The Programmer's Interface*. IEEE Annals of the History of Computing, Vol. 22, No.4, pp.5-19 (1998).

JFJF2003:
John Fuegi, Jo Francis: *Lovelace & Babbage and the Creation of the 1843 ´Notes´*. IEEE Annals of the History of Computing, Vol. 25, No.4, October-December, pp. 16-23 (2003).

CB1864:
Charles Babbage: *Passages From The Life Of A Philosopher*. London 1843.
Facsimile online: https://archive.org/details/bub_gb_Fa1JAAAAMAAJ

CB1837:
Charles Babbage: *Notes Series L*. Science Museum Archives, Wroughton. Not published.

PSXG2002:
Pascal Sebah, Xavier Gourdon: *Introduction on Bernoulli's numbers*. Online at http://numbers.computation.free.fr/Constants/Miscellaneous/bernoulli.ps, dated June12, 2002, retrieved on 2015-10-08.

MCK1989:
Martin Campbell-Kelly (ed.): *The Works of Charles Babbage.* 11 vols. London: William Pickering, 1989, Vol. 3

---

[1]If there were two sets of wheels, I could imagine that they could be used alternatively: while one is read out, the value — as transmitted to the mill — is stored in the other one. Next read request reduces the other one and retains the number in the first one.
[2]So Babbage may well be considered of the first computer engineer using assertions!
[3]The Siemens 2002 computer, build before 1964, is a decimal computer with 12 binary coded decimal (BCD) digits and 2000 memory cells, that are also adressed by a scheme using BCD coded numbers; thus, indexing is possible without conversion to binary.
[4]According to Tim Robinson, Babbage did explore other, in particular smaller, bases and rejected them as the number of parts exploded compared to base 10. But clearly at that stage he did not think about indexing.