A Docent's Farewell to Charles Babbage's Difference Engine No. 2

Charles L. Rino
Mountain View Computer History Museum Docent
crino@comcast.net
http://chuckrino.com/wordpress/

Abstract

An exact replica of a difference engine constructed 152 years after it was designed by the 19th century mathematician Charles Babbage was operated at the Mountain View, California Computer History Museum, courtesy of Naythan Myhrvold, from April 2008 until February 2016. I had the good fortune to become a Computer History Museum docent in 2011. Shortly thereafter I joined the team of operators and presenters who demonstrated the machine for thousands of museum visitors. This paper is a summary of my DE2 docent experience and a fond farewell to DE2.

1 Introduction

A reconstruction of Charles Babbage's Difference Engine No. 2 (DE2) was on loan to the Mountain View, California Computer History Museum (CHM) from April 9, 2008 through January 2016. The Charles Babbage story and how the first DE2 came to be constructed more than 150 years after it was designed can be found in Doron Swade's book, *The Cogwheel Brain, [1]* and his paper, *The Construction of Charles Babbage's Difference Engine No. 2, [2].* Briefly, evolving industrial-age commerce created a demand for accurate tables of mathematical functions. Mathematicians knew how to perform the computations [3], but manual execution was time consuming and translating the results to print was error prone. In a popular anecdote, Charles Babbage was comparing tables of functions with his astronomer colleague and lifelong friend John Hershel. They found numerous discrepancies, which caused Babbage to exclaim, "I wish to God that these calculations had been executed by steam." Executed by steam was a metaphor for automation.

In 1823 Babbage designed a difference engine that could compute and generate stereographs for printing error-free tables of mathematical functions. He petitioned the British government and received a large amount of funding to build his first design, Difference Engine No. 1. However, after nearly ten year's

work only one third of the machine's 24,000 parts and a working fragment had been constructed. In an attempt to accelerate the effort Babbage got into a dispute with his machinist, Joseph Clement. The project came to an abrupt end when Clement dismissed his workers and quit.

Eleven years into the effort with only the *beautiful fragment* in hand, Babbage turned his full attention to the design of an analytic engine capable of general-purpose computation under punched-card control. He is most famous for his *analytic engine* design, which foreshadowed modern digital computers. The design for his second calculating engine, Difference Engine No. 2 (DE2), used refinements developed for the analytic engine. He petitioned the British government again in 1848 with and offer to fulfill his earlier commitment, but the offer was declined. Charles Babbage died in 1871 bitterly disappointed that his designs were never realized. He made the prophetic remark "Another age must be the judge."

The London Science Museum inherited Babbage's designs, papers, the beautiful fragment, and other artifacts collected by his youngest son, Henry. The collection resided in the London Museum for over a century. In 1985 Doron Swade, a new curator, started a project to construct a realization of DE2 from Babbage's plans using materials and procedures that could have been implemented in Babbage's lifetime. Alan Bromley, an an Australian engineer, submited a proposal at about the same time.

Construction of the computation section of DE2 was completed just in time for a demonstration in 1991 to celebrate the bicentennial of Babbage's birth. The unfinished DE2 came to the attention of Microsoft's former chief technical officer, Nathan Myhrvold. He agreed to fund the completion of DE2 provided that the London Science Museum build a second complete replica for his personal use. Upon its completion in 2008 Myhrvold generously lent his DE2 replica to the Mountain View Computer History Museum where thousands of museum visitors saw the machine operate and learned its fascinating story. It was not uncommon to have upwards of 100 visitors viewing the demonstrations with audible gasps as they viewed the carry operations for the first time. In early February, 2016 DE2 in perfect working order was carefully crated to complete its journey to Seattle, Washington where it now resides in the lobby of Nathan Myhrvold's company, Intellectual Ventures.

For the cadre of Babbage enthusiasts, the DE2 experience was exhilarating. I had the good fortune to become a CHM docent in 2011 when the main exhibition "Revolution, The First 2000 Years of Computing" opened to the public. Demonstrations of DE2 were already underway, but the Babbage team of presenters, operators, and the well-healed maintenance crew was being expanded. I joined the Babbage team in 2012. The DE2 team was unknowingly participating in a living historical discovery of what Charles Babbage would have encountered had DE2 been built in his lifetime. Our experience through the sad farewell party January 11, 2016 is captured in photo-documented html on Ed Thalen's website [1].

[1] http://ed-thelen.org/BabInstCHM/index.html

What follows is a sketch of a docent presentation. One of our docents attired himself in period dress and lectured from a small table as Charles Babbage might have done. Another docent wore the cap and atire of Joseph Clement while operating DE2, which might be ironic. On one occasion a large bug was found in the oil pan at the base of the machine. Although the bug didn't get into the gears and disrupt the DE2 operation, it was dutifully pasted into the log book as Grace Hopper had done with a moth (bug) that did disrupt the operation of the Mark II relay computer in the early 1940s.

Our docent presentations included a variant of the historical sketch, usually embellished with amusing Babbage lore, such as his letter to Alford Lord Tennyson:

> Sir:
>
> In your otherwise beautiful poem "The Vision of Sin" there is a verse which reads – "Every moment dies a man, Every moment one is born." It must be manifest that if this were true, the population of the world would be at a standstill. In truth, the rate of birth is slightly in excess of that of death. I would suggest that in the next edition of your poem you have it read – "Every moment dies a man, Every moment 1 1/16 is born." The actual figure is so long I cannot get it onto a line, but I believe the figure 1 1/16 will be sufficiently accurate for poetry.
>
> I am, Sir, yours, etc.,
>
> Charles Babbage

I find it hard to believe this letter was meant other than as a tongue-in-cheek remark. After the historical introduction docents explained, with varying amounts of detail, how the machine generates and prints error-free tables of mathematical functions. As already noted, the highlight of the presentation was seeing the machine in operation. A chalk-board sidebar explained how the method of differences reduces the serial evaluation of a polynomial to a sequence of additions. The name difference engine comes from the method of differences.

Figure 1 shows DE2 ready for a demonstration. The entire apparatus weighs 5 metric tons. A heavy supporting frame is hidden under the platform. The left-hand side of the machine generates page stereographs that would be filled with molten lead for type setting and subsequent printing. A crank and bevel-geared rotation shaft can be seen on the right-hand side with a hand crank that powers the engine through a 4:1 reduction gear. The central upper frame, the computation section, contains eight columns of 31 bronze figure wheels. The position of each figure wheel represents a digit from 0 to 9. Subgroups of the 31-digit numbers represented by each column can be isolated. This feature is used to advantage to maintain a cycle count.

In more modern terminology the 31-wheel columns are registers. During a computation cycle (one full rotation of the main shaft) the contents of each register is added to the left adjacent register starting at the right-most register. The replaced contents of the first register is the new result. Babbage implemented a more efficient scheme, which would be recognized today as pipelining,

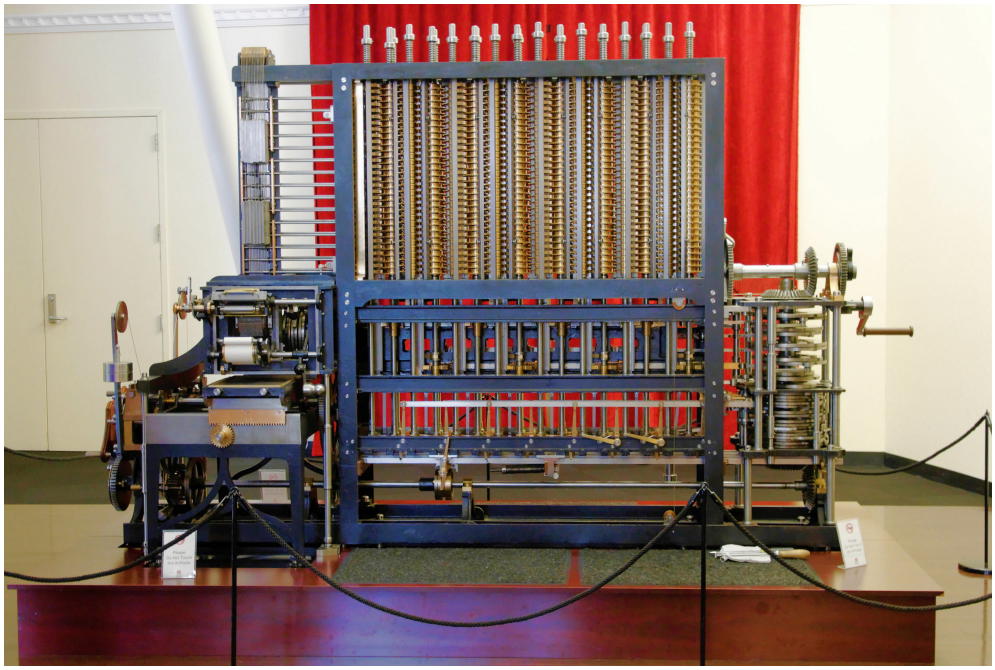

Figure 1: Babbage Difference 2 Working Replica at Mountain View, Computer History Museum.

to perform the odd-to-even additions simultaneously followed by simultaneous even-to-odd additions.

Mechanical addition was challenging to 19^{th} century mathematicians. How to manage additions that required a carry was the main problem. Babbage's solution was elegant. You first add all the number wheels simultaneously, but you keep track of the wheels that generate a carry. A second serial operation starting with the lowest wheel adds one to the wheel above each wheel with a set carry flag, but the carry operation itself can generate a carry. This was accommodated by performing the carry addition with carry detection active.

The mechanical operation of DE2 and the underlying operation principle is beautifully illustrated in 5 You Tube videos made with advanced CAD and rendering by Mike Hilton [2]. The first video demonstrates the method of finite differences, which reduces the sequential evaluation of an approximating polynomial to a series of additions. Our initial DE2 demonstrations used the polynomial

$$y(x) = 41 + 4x + 7x^2 + x^3 + 5x^4 + 9x^5 + 2x^6 + 8x^7, \tag{1}$$

which was evaluated at $x = 0, 1, 2, \cdots$. The value of x was indicated by a count

[2] https://www.youtube.com/playlist?list=PLSOxgHhh6-o8ZuhRpL9ds8wM4doxauru-

maintained with the first four digits in the results column. The 27-digit value of the polynomial represented by (1) increases monotonically. Consequently, progressively more of the DE2 figure wheels become active. As operators cranked on, small misalignments caused the machine to jam. The machine was designed to jam rather than propagate an incorrect answer, but occasionally the jams bent or broke delicate carry levers. Under Tim Robinson's leadership, numerous adjustments were made that significantly improved DE2 reliability and reduced operator stress. Operating the machine when a jam broke one or more delicate carry levers was not a pleasant experience.

Attempts to demonstrate 7-digit error-free tabulation started early in 2011, but stress on some damaged components caused a retreat to the polynomial demonstration. By August, 2013 the DE2 operation with some replaced parts was reliable enough to demonstrate 7-digit, error-free tabular computation of base-10 logarithms.

Details of why the method of differences works, how to set up the pipelining operation, and how to determine polynomial coefficients that approximate a function with a specified accuracy had to be ferreted out by the docents. As we pursued these learning challenges it was natural to engage our own PCs to try out some examples. It was easy enough to evaluate (1), generate a table of differences, and follow the prescription for adjusting the differences to support pipeline evaluation [3]. With the starting differences initialized you just add the odd differences to the even differences and then add the modified even differences to the previously used odd differences. Things start out well, but after a hundred cycles or so the PC and DE2 values begin to differ[4]. The DE2 answers are correct. The problem is accuracy. Standard 64-bit double-precision additions support only 16 digits.

In the absence of special-purpose software the only way the DE2 values can be reproduced is by executing the DE2 operations on a PC exactly as Babbage implemented the operations mechanically. An *emulator* was developed that reproduces the exact content of each of the 31-digit registers and the carry lever settings. As part of docent operating procedures, the beginning and ending results were checked against a thick tabular listing of the correct results. The emulator did get used for diagnostic mapping of carry-flag progressions that preceded recurrent jams. For a brief period the emulator was used to predict what came to be called the *correct wrong answer* when broken carry levers were bypassed. The DE2 result was not correct, but the result could be calculated. When the DE2 was operating, the clacking sound of locks that aligned the number wheels after each addition could be heard throughout the entry area. In the absence of this welcome familiar sound, a recorded sound track was added to the emulator. The sound track can be heard in the entry way to the main CHM exhibition, but it's largely unrecognized.

When DE2 was set up to generate entries in a table of logarithms, docents had to adjust their presentations accordingly. Setting up the demonstration

[3] See Section 2.5.
[4] See Figure 5.

presented analytic challenges. The schemes that Babbage would have used to generate an approximating polynomial are accessible, but too intricate for demonstration purposes. So, rather than attempt to reconstruct a scheme Babbage might have used, Tim Robinson tackled the problem with the aid of the advanced programming language Mathematica. The least-squares computation formally amounts to nothing more that solving the system of linear equations represented by (29). The challenge is you have to do the calculation with 27 digit accuracy. Robinson's calculation deviates from historical discovery, but it does demonstrate how far computational resources have come since the "Age of Automation."[5]

The following 7^{th}-order polynomial will approximate $y(x) = \log 10(x)$ over the subrange $1 \leq x \leq 1.3007$ correct to 7 decimal places in steps of 0.0001:

$$\begin{aligned} y(x) \quad = \quad & 0.023813682677220936464770 2x^7 \\ & -0.223692112913365566257355x^6 \\ & +0.925289889999278691240 53x^5 \\ & -2.212611500051764058845 86x^4 \\ & +3.382634386164791815432 50x^3 \\ & +.486984232663086651211 07x^2 \\ & +2.659813209777670302364865x \\ & -1.068263322990745469188367 \end{aligned} \tag{2}$$

Even with the polynomial defined, it is a demanding exercise to complete the setup. Following the prescription, 7 consecutive values of the polynomial must be computed. Since x is a small number, hand calculation would be feasible as would transforming the differences for pipeline evaluation. The following 27-digit integers represent the initiating differences after manipulations for pipelining:

$$\begin{array}{ll} d0 & 000130318797007462720037127 \\ d1 & 000043414250565213313999556 \\ d2 & 999999956596932030387728 86 \\ d3 & 000000000000867436407448217 \\ d4 & 999999999999999740943776800 \\ d5 & 000000000000000000099725904 \\ d6 & 999999999999999999999958999 \\ d7 & 000000000000000000000000012 \end{array} \tag{3}$$

The integers starting with 9 are tens-complement negative numbers. The integers multiplied by 10^{-27} represent the true values. That is, the implicit decimal point is just ahead of the most significant digit. The computations were tweaked a bit to generate a nonzero seventh difference. Because Babbage designed the machine to evaluate seventh-order polynomials, it was felt that the seventh difference should contribute at least one or two digits, times 10^{-27}. Docents never tired of passing these tidbits on to CHM visitors.

[5] See Section 2.7.

Figure 2: DE2 docents, operators, and maintenance crew.

The remainder of this paper describes the emulator and summarizes the mathematical background for the basic operation with pipelining. We show that an n^{th}-order polynomial can be defined by a difference table comprised of differences of order up to n. Differences higher than order n do not change and thereby provide no new information. Some visitors recognized the connection between differences and the derivatives of calculus. Charles Babbage was an aggressive adherent of modernizing calculus as it was practiced in England following its development by the Cambridge University alumnus, Isaac Newton.

The DE2 story can be appreciated without the mathematical details, but as a teaching aid the historical context might be stimulating. The emulator is easily reproduced with any programming language that accommodates algebraic operation on arrays of numbers. A spread-sheet will do. The correspondence between a difference table and the generating polynomial no longer has practical utility, but it is an engaging mathematical exercise, particularly when pipelining is introduced. The pipelining "trick" was not obvious to most DE2 students including this author. An algorithm is presented that will recover the polynomial coefficients from any sequence of defining differences. Babbage was vary likely aware of such an algorithm. It would have come in handy for checking the operation.

To conclude this introduction, imagine that DE2 had been built and that approximating polynomials were being evaluated automatically with the machine

and error-free stereographs were being produced. To calculate the log 10 table entries from 1 to 10 in steps of .0001 requires at least 10,000 cycles. From our CHM experience, the machine could be operated conservatively at one full cycle every 8 sec. Allowing for maintenance, stereograph page setup, and trimming for print, at least one month would have been required to generate a 7-digit error free log 10 table, certified by Charles Babbage. Would Charles Babbage have been celebrated for that accomplishment, given the cost? There are published corrections of Babbage's tables using methods that could have been implemented in his lifetime [4]. Babbage's detractors probably would have argued that error-free tables could have been produced by conventional methods at much lower cost.

On the other hand, from a recent history of ENIAC by Haigh, Priestley, and Rope [5], the same conclusion was drawn by some during the early operation of ENIAC in 1946. We know think of ENIAC as the beginning of numerous developments that led to commercially available general-purpose digital computers. If DE2 had been build in Babbages's lifetime, the Analytic Engine would most likely have followed. At the time the laws of electricity were being discovered by Babbage's contemporaries. Babbage's famous protegee, Ada Lovelace, kindly offered to help Michael Fairday with his experiments [6]. James Clerk Maxwell began his seminal studies at the time of Babbage's death. None of the seminal discoveries of the nature of electricity depended on automated computation, but computational electromagnetics, finite elements, and such might have caught on sooner.

2 DE2 Emulation

2.1 Notation and Basic Operations

To introduce some notation, let the 8 31-digit DE2 registers be identified from left to right as d_0, d_1, d_2, d_3, d_4, d_5, d_6, and d_7. One full turn of the main shaft completes an operation cycle in two parts. During the first half cycle each odd register is simultaneously added to the adjacent even register. Formally, $d_6 \leftarrow d_7 + d_6$; $d_4 \leftarrow d_4 + d_5$; $d_2 \leftarrow d_2 + d_3$; $d_0 \leftarrow d_0 + d_1$. The left arrow means replacement of the contents of the register. The new result appears in register d_0 at the end of the first half cycle. The addition operation includes servicing any flagged carry operations and resetting registers d_7, d_5, d_3, and d_1 to their original values for use during the second half cycle. During the second half cycle the even registers are simultaneously added to the adjacent odd columns: $d_5 \leftarrow d_5 + d_6$; $d_3 \leftarrow d_3 + d_4$; $d_1 \leftarrow d_1 + d_2$. Carry flags are serviced and registers d_6, d_4, and d_2 are reset to their original values.

For emulation the 31 wheel positions are stored as integers in arrays $d_n(m)$, where m represents the figure wheel index from 1 to 31. An auxiliary array, $C(m)$, keeps track of the carry lever settings (0 => not warned, 1=> warned, 2=> warning serviced). The addition or giving off operation is implemented as an element-by-element modulo 10 addition of $d_n(m)$ to $d_{n-1}(m)$. When

additions produce a second digit the corresponding carry array entry is changed from 0 to 1 meaning a carry occurred.

The carry phase operation uses the $C(m)$ array to modify $d_{n-1}(m)$ where carries are needed. Carry polling and servicing starts at the first wheel. If a 1 is present, the carry flag is changed to 2 and 1 is added to the next wheel, $d_{n-1}(m+1)$. If that addition causes a carry, $C(m+1)$ is set equal to 1 before it is polled. A carry bypass flag inhibits the carry operation, thereby isolating the wheels above the inhibited carry.

The 30 DE2 polling arms are spaced 22.5 degrees apart with an extra 22.5 degrees included between the 15th and 16th wheels to avoid a physical conflict between the 15th and 16th wheel. With this arrangement the actual polling arms advance in two identical groups of 15, which means that the upper and lower halves of the column carry warnings are serviced simultaneously. If a carry occurs at wheel 15 in the first rotation, it will not be serviced during the first rotation. A second rotation of the polling arms captures and, if necessary, propagates any missed carry operations. Since the carry warnings from the first 15 wheels have already been polled and serviced, the second rotation only affects wheels 16 through 31.

The parallel carry polling could be emulated by applying the carry operation to digits 16 to 31 and then to digits 1 to 15, which are actually performed simultaneously. By stopping at 15, no carries propagate. By performing the polling in reverse order, a carry flag at digit 15 will not be serviced until the second cycle is performed. A repeat of the carry phase operations for digits 16 to 31 picks up an unserviced carry at wheel 15. The only reason for using this level of emulation would be to reproduce register settings within uncompleted cycles. The register values at the end of each full cycle can be reproduced simply by implementing the carry-phase operations from 1 to 31.

2.2 Sequential Polynomial Evaluation

To set up a polynomial for DE2 evaluation a difference table is generated. Using (1), as an example, the following difference table is easily constructed:

x	d0	d1	d2	d3	d4	d5	d6	d7
0	41	36	1492	17064	72600	139080	122400	40320
1	77	1528	18556	89664	211680	261480	162720	
2	1605	20084	108220	301344	473160	424200		
3	21689	128304	409564	774504	897360			
4	149993	537868	1184068	1671864				
5	687861	1721936	2855932					
6	2409797	4577868						
7	6987665							

(4)

The first column is the value of x. The second column is the computed value of the polynomial at x. Each difference column entry is the difference between next value and the current value in the adjacent column to the left. An astute

observer will notice that any number in the table can be reproduced by adding the number above it to the number above it in the next column. So, if you have only the differences at $x = 0$, you can extend the table indefinitely with nothing but additions. Start by repeating the largest difference, which does not change.

Pipelining will be discussed in more detail later. Looking ahead, pipelining is set up by manipulating the initial register settings as follows from left to right:

$$\begin{array}{lll} \mathbf{d}^6 \leftarrow d_n^6 - d_n^7 & \mathbf{d}^6 \leftarrow \mathbf{d}^6 - d_n^7 & \mathbf{d}^6 \leftarrow \mathbf{d}^6 - d_n^7 \\ \mathbf{d}^5 \leftarrow d_n^5 - \mathbf{d}^6 & \mathbf{d}^5 \leftarrow \mathbf{d}^5 - \mathbf{d}^6 & \\ \mathbf{d}^4 \leftarrow d_n^4 - \mathbf{d}^5 & \mathbf{d}^4 \leftarrow \mathbf{d}^4 - \mathbf{d}^5 & \\ \mathbf{d}^3 \leftarrow d_n^3 - \mathbf{d}^4 & & \\ \mathbf{d}^2 \leftarrow d_n^2 - \mathbf{d}^3 & & \end{array} \tag{5}$$

The bold face components are being modified or have been modified. Applying these operations to the difference table provides the initial values for DE2 evaluation of (1):

x	d0	d1	d2	d3	d4	d5	d6	d7
0	41	36	28	1464	360	15240	1440	40320

(6)

The extension of the table is performed in place by first performing the odd-to-even additions, followed by the even-to-odd additions.

To maintain a count, the first four weels are set to zero in every register except d1. The first wheel in d1 is set equal to 1. At the completion of each cycle one is added to d0 maintaining a count equal equal to x. With carry lever 4 in column 1 bypassed, the count in the first four digits will not affect the 27 digit polynomial result. The figure below shows the setup of the machine with the $x = 0$ value 41 in the d0 register.

CYCLE 0								
Wheel	d0	d1	d2	d3	d4	d5	d6	d7
31	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	4
8	0	0	0	1	0	5	1	0
7	0	0	0	4	3	2	4	3
6	4	3	2	6	6	4	4	2
5	1	6	8	4	0	0	0	0
4	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0

Figure 3: DE2 setup for demonstration polynomial.

DE2 emulation is implemented with two subprograms:

```
[da,carry]=GivingOff(d1,d2)
```
[dc,carry]=CarryPhase(da,carry,nwstart,nwend,DisableFlags)

The `GivingOff` routine replaces da with the modulo 10 addition of $d1 + d2$. A carry flag array is generated in the carry array with zeros indicating no carry or 1 indicating a carry warning. The CarryPhase subroutine implements the carry operations. The carry flags are serviced from wheel `nwstart` to `nwend`. If the carry flag corresponding to `nwend` is set, $da(nwend + 1)$ in incremented. To service 31 digits, $nwend = 30$. After executing `GivingOff` the carry entries are 0 or 1. After executing `CarryPhase` all the active carry entry 1 values are replaced by 2, indicating that the carries have been serviced.

No.	ODD-2-EVEN OPERATIONS GIVING OFF	No.	EVEN-2-ODD OPERATIONS GIVING OFF
1	$[\texttt{d6},\texttt{c6}] = \texttt{GivingOff}(\texttt{d6},\texttt{d7})$	17	$[\texttt{d5},\texttt{c5}] = \texttt{GivingOff}(\texttt{d5},\texttt{d6})$
2	$[\texttt{d4},\texttt{c4}] = \texttt{GivingOff}(\texttt{d4},\texttt{d5})$	18	$[\texttt{d3},\texttt{c3}] = \texttt{GivingOff}(\texttt{d3},\texttt{d4})$
3	$[\texttt{d2},\texttt{c2}] = \texttt{GivingOff}(\texttt{d2},\texttt{d3})$	19	$[\texttt{d1},\texttt{c1}] = \texttt{GivingOff}(\texttt{d1},\texttt{d2})$
4	$[\texttt{d0},\texttt{c0}] = \texttt{GivingOff}(\texttt{d0},\texttt{d1})$		
	CARRY ROTATION 1		CARRY ROTATION 1
5	$[\texttt{d6},\texttt{c6}] = \texttt{CarryPhase}(\texttt{d6},\texttt{c6},16,31)$	20	$[\texttt{d5},\texttt{c5}] = \texttt{CarryPhase}(\texttt{d5},\texttt{c5},16,31)$
6	$[\texttt{d6},\texttt{c6}] = \texttt{CarryPhase}(\texttt{d6},\texttt{c6},1,15)$	21	$[\texttt{d5},\texttt{c5}] = \texttt{CarryPhase}(\texttt{d5},\texttt{c5},1,15)$
7	$[\texttt{d4},\texttt{c4}] = \texttt{CarryPhase}(\texttt{d4},\texttt{c4},16,31)$	22	$[\texttt{d3},\texttt{c3}] = \texttt{CarryPhase}(\texttt{d3},\texttt{c3},16,31)$
8	$[\texttt{d4},\texttt{c4}] = \texttt{CarryPhase}(\texttt{d4},\texttt{c4},1,15)$	23	$[\texttt{d3},\texttt{c3}] = \texttt{CarryPhase}(\texttt{d3},\texttt{c3},1,15)$
9	$[\texttt{d2},\texttt{c2}] = \texttt{CarryPhase}(\texttt{d2},\texttt{c2},16,31)$	24	$[\texttt{d1},\texttt{c1}] = \texttt{CarryPhase}(\texttt{d1},\texttt{c1},16,31)$
10	$[\texttt{d2},\texttt{c2}] = \texttt{CarryPhase}(\texttt{d2},\texttt{c2},1,15)$	25	$[\texttt{d1},\texttt{c1}] = \texttt{CarryPhase}(\texttt{d1},\texttt{c1},1,15)$
11	$[\texttt{d0},\texttt{c0}] = \texttt{CarryPhase}(\texttt{d0},\texttt{c0},16,31)$		
12	$[\texttt{d0},\texttt{c0}] = \texttt{CarryPhase}(\texttt{d0},\texttt{c0},1,15)$		
	CARRY ROTATION 2		CARRY ROTATION 2
13	$[\texttt{d6},\texttt{c6}] = \texttt{CarryPhase}(\texttt{d6},\texttt{c6},16,31)$	26	$[\texttt{d5},\texttt{c5}] = \texttt{CarryPhase}(\texttt{d5},\texttt{c5},16,31)$
14	$[\texttt{d4},\texttt{c4}] = \texttt{CarryPhase}(\texttt{d4},\texttt{c4},16,31)$	27	$[\texttt{d3},\texttt{c3}] = \texttt{CarryPhase}(\texttt{d3},\texttt{c3},16,31)$
15	$[\texttt{d2},\texttt{c2}] = \texttt{CarryPhase}(\texttt{d2},\texttt{c2},16,31)$	28	$[\texttt{d1},\texttt{c1}] = \texttt{CarryPhase}(\texttt{d1},\texttt{c1},16,31)$
16	$[\texttt{d0},\texttt{c0}] = \texttt{CarryPhase}(\texttt{d0},\texttt{c0},16,31)$		

2.3 The Method of Differences

It is interesting that the following development is difficult to find in a text book. One colleague, an astronomer, said it was so obvious nobody bothered to write it down. For those who don't find it obvious, let $f(x)$ represent a continuous function to be approximated by a polynomial over a prescribed range of the independent variable x . Let $P(x)$ represent the approximating polynomial of order p. An order p polynomial is defined by $p + 1$ coefficients, which are denoted $\widetilde{a}_k$ with $k = 0, 1, \cdots, p$. The approximating polynomial would normally be written as

$$\widetilde{P}(x) = \sum_{k=0}^{p} \widetilde{a}_k x^k. \tag{7}$$

The independent variable of a sampled function takes the values $x_n = n\Delta x + x_0$, whereby

$$f_n = f(n\Delta x + x_0), \text{ for } n = 0, \cdots, N-1. \tag{8}$$

However, there is no loss of generality in taking

$$P_n = \sum_{k=0}^{p} a_k n^k, \text{where } a_k = \widetilde{a}_k \left(\Delta x\right)^k. \tag{9}$$

The sample interval is absorbed in the definition of a_k. Although one set of coefficients define a polynomial, the starting point matters. Consider the M-sample shift

$$f(n\Delta x + x_0 + M\Delta x) = \sum_{k=0}^{p} a_k \left(n+M\right)^k \tag{10}$$

$$= \sum_{k=0}^{p} a'_k n^k \tag{11}$$

The two sets of coefficients a_k and a'_k are related, but large values of M might stress the number of digits required. This is of concern only for polynomial approximation.

Ordered differences are differences between two consecutive values of a function or the differences between previously computed finite differences. From a set of uniformly spaced values of any function, a finite difference table can be constructed as follows:

$$d_n^0 = f_n \tag{12}$$

$$d_n^m = d_{n+1}^{m-1} - d_n^{m-1} \tag{13}$$

The m index represents the order of the finite difference. The n index identifies the first contributing sample. A finite difference table to order 7 would be populated column by column as follows:

f_n	d_n^1	d_n^2	d_n^3	d_n^4	d_n^5	d_n^6	d_n^7
f_{n+1}	d_{n+1}^1	d_{n+1}^2	d_{n+1}^3	d_{n+1}^4	d_{n+1}^5	d_{n+1}^6	
f_{n+2}	d_{n+2}^1	d_{n+2}^2	d_{n+2}^3	d_{n+2}^4	d_{n+2}^5		
f_{n+3}	d_{n+3}^1	d_{n+3}^2	d_{n+3}^3	d_{n+3}^4			
f_{n+4}	d_{n+4}^1	d_{n+4}^2	d_{n+4}^3				
f_{n+5}	d_{n+5}^1	d_{n+5}^2					
f_{n+6}	d_{n+6}^1						
f_{n+7}							

. (14)

The first column is the function evaluated at n: $d_n^0 = fn$.

If d_n^p is constant at some order p, all higher order differences are zero. This occurs if and only if f_n is a polynomial of order p. That is, if a difference table

has precisely constant differences at order p, the function is a polynomial of order p. The proposition is proved by demonstrating that unique polynomial coefficients can be recovered from the finite differences to the constant value. An algorithm that will recover the polynomial coefficients from a difference table can be found in Appendix A. One could also solve a system of $p+1$ equations in $p+1$ unknowns. The recursion is more elegant and easier to implement.

2.4 Forward Recursion

Having demonstrated the equivalence between a polynomial defined by a difference table and a polynomial defined by its coefficients, we consider how finite differences can be exploited. From (13) it follows that $d_{n+1}^{m-1} = d_n^{m-1} + d_n^m$. Since d_n^p is constant for all n, the following recursion extends (14) indefinitely:

$$d_{n+(l+1)}^{p-(l+1)} = d_{n+l}^{p-l} + d_{n+l}^{p-(l+1)} \text{ for } l = 0, \cdots, p-1. \tag{15}$$

This property shows formally that the sum of any two entries in a difference table reproduces the difference below the lower order difference. Starting with the first complete row in (14), recursive evaluation of f_n with only addition of differences could proceed as follows:

$$d_{n+1}^p = d_n^p \tag{16}$$

$$d_{n+1}^{p-(l+1)} = d_n^{p-l} + d_n^{p-(l+1)} \text{ for } l = 0, \cdots, p-1. \tag{17}$$

A device with $p+1$ storage registers could execute the recursion serially. With the implicit notation d_n with $n = 7, 6, \cdots, 0$ used to designate the elements in the n^{th} row of (14), the finite-difference algorithm is defined as follows:

$$\begin{aligned}
d_7' &\leftarrow d_7 \text{ (implicit)} \\
d_6' &\leftarrow d_7' + d_6 \\
d_5' &\leftarrow d_6' + d_5 \\
d_4' &\leftarrow d_5' + d_4 \\
d_3' &\leftarrow d_4' + d_3 \\
d_2' &\leftarrow d_3' + d_2 \\
d_1' &\leftarrow d_2' + d_1 \\
d_0' &\leftarrow d_1' + d_0
\end{aligned} \tag{18}$$

The primes are introduced to separate the values before (unprimed) and after the replacement operations (primed).

2.5 Pipelining

A more efficient scheme can be constructed by looking in more detail at the operations that execute the forward recursion just described. Consider reconstructing the differences that define the last completed computation cycle n. This is achieved by the following operations:

f_{n-3}						$\mathbf{d}^6_{n-3}$	$\mathbf{d}^7_n$
f_{n-2}				$\mathbf{d}^4_{n-2}$	$\mathbf{d}^5_{n-2}$	d^6_{n-2}	d^7_n
f_{n-1}		$\mathbf{d}^2_{n-1}$	$\mathbf{d}^3_{n-1}$	d^4_{n-1}	d^5_{n-1}	d^6_{n-1}	d^7_n
$\mathbf{f}_n$	$\mathbf{d}^1_n$	d^2_n	d^3_n	d^4_n	d^5_n	d^6_n	d^7_n

. (19)

The reconstruction is performed row by row with repeated use of (13). It can be seen from (19) that placing the embolden values in the 8 DE2 registers allows parallel computation. The odd-to-even additions $7+6 \rightarrow 6$, $5+4 \rightarrow 4$, $3+2 \rightarrow 2$, and $1+0 \rightarrow 0$ can be performed in parallel. Performing the even-to-odd additions $6+5 \rightarrow 5$, $4+3 \rightarrow 3$, $2+1 \rightarrow 1$ updates the odd registers for the next computation cycle. The computed paired differences effectively reproduce a stair-case difference operation. The setup for pipeline operation can be computed in place. Writing out the backward recursion explicitly defines the cycle by cycle operations:

n	$n-1$	$n-2$	$n-3$
d^6_n	$d^6_{n-1} = d^6_n - d^7_n$	$d^6_{n-2} = \left(d^6_{n-1}\right) - d^7_n$	$d^6_{n-3} = \left(d^6_{n-2}\right) - d^7_n$
d^5_n	$d^5_{n-1} = d^5_n - \left(d^6_{n-1}\right)$	$d^5_{n-2} = \left(d^5_{n-1}\right) - \left(d^6_{n-2}\right)$	
d^4_n	$d^4_{n-1} = d^4_n - \left(d^5_{n-1}\right)$	$d^4_{n-2} = \left(d^4_{n-1}\right) - \left(d^5_{n-2}\right)$	
d^3_n	$d^3_{n-1} = d^3_n - \left(d^4_{n-1}\right)$		
d^2_n	$d^2_{n-1} = d^2_n - \left(d^3_{n-1}\right)$		

The values in parentheses have-been modified by previous computations, which are executed column wise in the table. Written more compactly, the column wise extra operations required to initiate the pipeline operation, which have already been introduced follow:

$\mathbf{d}^6 \leftarrow d^6_n - d^7_n$	$\mathbf{d}^6 \leftarrow \mathbf{d}^6 - d^7_n$	$\mathbf{d}^6 \leftarrow \mathbf{d}^6 - d^7_n$
$\mathbf{d}^5 \leftarrow d^5_n - \mathbf{d}^6$	$\mathbf{d}^5 \leftarrow \mathbf{d}^5 - \mathbf{d}^6$	
$\mathbf{d}^4 \leftarrow d^4_n - \mathbf{d}^5$	$\mathbf{d}^4 \leftarrow \mathbf{d}^4 - \mathbf{d}^5$	
$\mathbf{d}^3 \leftarrow d^3_n - \mathbf{d}^4$		
$\mathbf{d}^2 \leftarrow d^2_n - \mathbf{d}^3$		

2.6 Negative Differences

Neither negative differences nor negative functional values can be excluded from the finite-difference scheme as described above. However, negative numbers can be incorporated by adding complements. Write the n-digit negative number as

$$N = -a_n 10^n - a_{n-1} 10^{n-1} - \cdots - a_1 10 - a_0. \tag{20}$$

The coefficients a_n represent positive integers. The tens complement is defined as

$$\begin{aligned} NC &= (9 - a_n)\, 10^n + (9 - a_{n-1})\, 10^{n-1} + \cdots + (9 - a_1 10) + (9 - a_0) \\ &= -|N| + (10^n - 1)\,. \end{aligned} \tag{21}$$

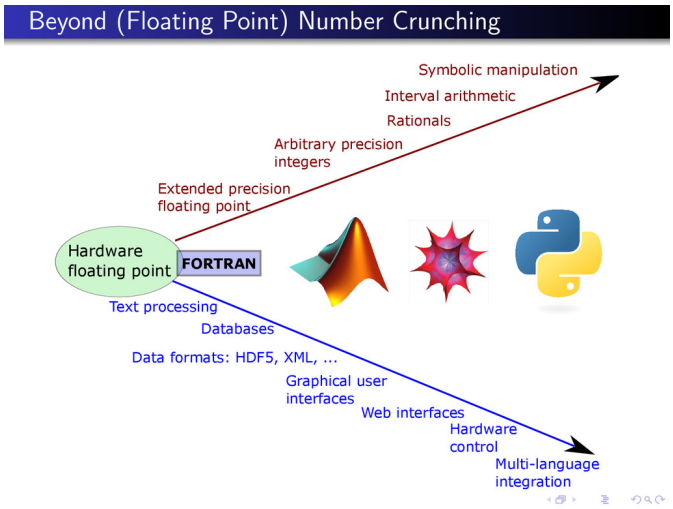

Figure 4: Evolution of computer languages for analysis.

From (21) it follows that adding the complement to any positive or negative complement number in range will produce the correct answer with negative number remaining in complement form. Thus, negative numbers are written in tens complement form. If the polynomial produces a negative result, the difference engine would produce the correct answers in tens complement form. A non-zero value of digit n would indicate a negative number. Manual conversion to a signed number would be necessary.

2.7 Emulator Examples

Writing computer code requires a computer language. Computer languages are specific to user applications. FORTRAN, which was developed in the 1950s by IBM, was the most widely used computational language for analysis, and it remains in use today. However, as illustrated in Figure 4, computer languages have evolved into interactive code development environments. The three symbols following FORTRAN represent MatLab, Mathematica, and Python. MatLab and Mathematica are commercial products. Python is open-source with a large user support base.

The operation of the emulator is unaffected by word size because additions are performed digit-by-digit. The precision of direct polynomial evaluation is limited by the 64-bit double precision standard. The native precision of digital computers is determined by word size. In the 1970's, the heyday of batch processing, machines with 32-bit words were widely available. Software allowed combination of two 32-bit words to achieve *double precision* computation. The IEEE standard for 64-bit double precision is now supported directly by 64-bit

words[6]. In effect, single-precision operations now fulfill the earlier double-precision standard.

A 64-bit word can represent integers in the range

$$I = \pm 2^{63} = \pm 9223372036854775807,$$

exactly. Floating-point numbers use the first bit to indicate the sign followed by a set of bits used to represent a exponent. The remaining bits represent the fractional part of the floating-point number scaled by the exponent. The 64-bit double precision standard assigns 11 bits to the exponent and 52 bits to the fractional part. Thus, double-precision 64-bit floating point numbers cover the range $\pm 10^{-308}$ to $\pm 10^{308}$ with 15 to 17 decimal digit precision.

The emulator described in Section 2 was programed in MatLab. Since the coefficients and the range of x values are within the range of exact representation, direct evaluation of the polynomial was used to generate a difference table. The register values were then manipulated for pipelining and loaded digit-by-digit into the DE2 register columns, mimicing the manual setup of DE2. The setup is shown in Figure 3. The count is maintained in the first 4 digits of the results column. Digits 5 through 31 store the result. The first digit of column 2 is set to 1. The first 4 digits of the remaining columns are set to zero.

The upper frame of Figure 5 compares the emulator results converted to a 64 bit integers (red circles) to the result obtained by evaluating (1) with default floating-point arithmetic (blue line). The green line marks the largest floating point number that can be represented exactly. The lower frame shows the absolute error scaled to 10^{-16}. The errors would not be noticed in a standard display, nor would they be troublesome for most routine applications.

At this point we have covered all the material relevant to CHM demonstrations of DE2 up to August 2013, when the machine was reconfigured to evaluate base 10 logarithms to 7 decimal digit accuracy. To introduce logarithms, consider the value of x raised to the y^{th} power of 10, which is written as

$$x = 10^y.$$

The exponent, y, is the base-10 logarithm of x. That is,

$$y = \log 10(x).$$

A table of logarithms tells us what value of y will generate $x = 10^y$. Note that if x is divided by the n^{th} power of 10, the base-ten logarithm is offset by n:

$$y - n = \log 10(x10^{-n}).$$

Consequently, it is only necessary to evaluate $\log 10$ over one order of magnitude. One can easily keep track of the largest value of n that makes $x10^{-n}$

[6] IEEE Standard 754.

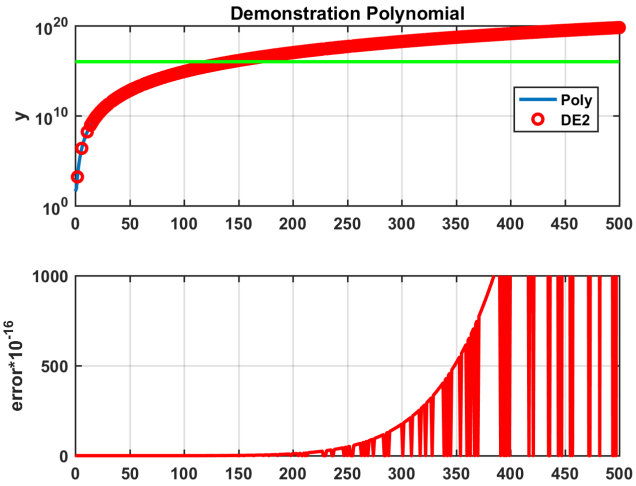

Figure 5: Polynomial PC evaluation errors.

less than say 10. The tabulation is performed over the decade 1 to 10 [7]. Published tables of base-10 logarithms list the values y_n for $x_n = 1 + n\Delta x$ for $n = 0, 1, 2, \cdots, N$. The table maker must decide the spacing Δx and precision as indicated by the number of decimal digits, $x10^{-n} = 1.ddddddddddd\cdots$.

3 Polynomial Approximation

The efficient evaluation of functions remains an active area of development to this day. For example, the routine computation of the position of artificial satellites uses simplified equations with orbital elements that are accurate for a prescribed period of time.

3.0.1 Taylor Series

The Taylor series was developed by the English mathematician Brook Taylor in 1715. It would have been known in Babbage's time. Taylor's theorem states that any well-behaved function can be represented by an infinite power series of the form

$$f(x) = \sum_{k=0}^{\infty} \frac{1}{k!} \left. f^{(k)}(x) \right|_{x=x_0} (x - x_0)^k . \tag{22}$$

where $f^{(k)}(x)$ denotes the k^{th} derivative of $f(x)$ with respect to the independent variable. The derivatives are evaluated at the starting point of the series. If

[7] $\log 10(0) = \infty$ and $\log 10(1) = 1$.

the Taylor series is truncated at p, the contribution of the neglected remainder can be bounded by the relation

$$R_p = (x - x_0)^{p+1} f^{(p+1)}(\xi) \text{ where } x_0 \leq \xi \leq x. \tag{23}$$

Since derivatives of the functions most commonly tabulated are generally known, Taylor series approximations can be implemented directly. However, because the DE2 effectively increments the index without the offset, the polynomial series must be evaluated in its Maclaurin form

$$f(x + x_0) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(x + x_0)\Big|_{x=0} x^k. \tag{24}$$

For completeness,

$$\log 10(1 + n\Delta x) = \sum_{n=0}^{p} \left[(-\Delta x)^n / (n \log(10))\right] n^n. \tag{25}$$

3.0.2 Least Squares

Because Taylor series error increases with increasing x, it is poorly suited for uniform approximation. The method of least squares allows minimization of the error over a specified interval, which is well suited to polynomial approximation. The least squares method was developed by Carl Fredric Gauss near the end of the 18^{th} century, and it would have been well know in Babbage's time.

An approximating polynomial has $p + 1$ unknowns. The least-squares solution to the polynomial approximation problem minimizes the quadratic error measure

$$\epsilon^2 = \sum_{n=0}^{N-1} \left(\sum_{k=0}^{p} a_k n^k - f_n \right)^2. \tag{26}$$

The solution, obtained by differentiating ϵ^2 with respect to a_k then setting each differentiation to equal zero. The system of linear equations that defines the least-squares solution can be written in matrix-vector form as

$$\left[V_N^p V_N^{pT} \right] \overrightarrow{a} = V_N^p \overrightarrow{f}, \tag{27}$$

where $\overrightarrow{a}$ the column vector arrangement of the unknown polynomial coefficients, and $\overrightarrow{f}$ a column vector arrangement of the N values of the function. The Vandermonde matrix, V_N^p, is defined as

$$V_N^p = \begin{bmatrix} 1 & 1 & 1 & \cdots & \cdots & 1 \\ 0 & 1 & 2 & & & (N-1)^1 \\ 0 & 1 & 2^2 & & & (N-1)^2 \\ & & & \vdots & & \\ 0 & 1 & 2^p & & & (N-1)^p \end{bmatrix}. \tag{28}$$

The formal solution for $N \geq p$ is

$$\overrightarrow{a} = \left[V_N^p V_N^{pT}\right]^{-1} V_N^p \overrightarrow{f}, \tag{29}$$

The matrix $\left[V_N^p V_N^{pT}\right]$, is $p \times p$; however, if $N = p$, the solution reduces to the simpler form

$$\overrightarrow{a} = [V_N^p]^{-1} \overrightarrow{f}. \tag{30}$$

The Vandermonde form, (29) can be solved more efficiently than the general solution of a system of linear equations. However, with small numbers of function samples, the computation time is dominated by the computation of $\left[V_N^p V_N^{pT}\right]$, and the evaluation of the right-hand side, $V_N^p \overrightarrow{f}$. Each entry of the product vector is a moment of the form

$$M_k = \sum_{n=0}^{N-1} n^k f_n \text{ for } k = 1. \tag{31}$$

Note also that dropping the first column in V_N^p is equivalent to constraining the solution to be zero at $n = 0$.

A Polynomial coefficient recovery

Finite differences of order m can be written in terms of $m + 1$ values of f_n explicitly:[8]

$$d_n^m = \sum_{k=0}^{m} (-1)^k \binom{m}{k} f_{n+m-k}. \tag{32}$$

where

$$\binom{m}{k} = \frac{m!}{(m-k)!k!}. \tag{33}$$

If f_n is a polynomial, f_{n+m-k} can be written as

$$f_{n+m-k} = \sum_{l=0}^{p} a_l (n+m-k)^l. \tag{34}$$

To determine the defining polynomial coefficients, consider the set of finite differences at $n = 0$:

$$f_{m-k} = \sum_{l=0}^{p} a_l (m-k)^l \tag{35}$$

$$= \sum_{l=0}^{p} a_l \sum_{l'=0}^{l} (-1)^{l'} \binom{l}{l'} k^{l'} m^{l-l'} \tag{36}$$

[8] Abrmowitz and Stegun 25.1[7]

Substituting f_{m-k} into the expression for d_n^m it follows that

$$d_0^{p-j} = \sum_{k=0}^{p-j} (-1)^k \binom{p-j}{k} f_{p-j-k} \tag{37}$$

$$= \sum_{k=0}^{p-j} (-1)^k \binom{p-j}{k} \sum_{l=0}^{p} a_l \sum_{l'=0}^{l} (-1)^{l'} \binom{l}{l'} (p-j)^{l-l'} k^{l'} \tag{38}$$

$$= \sum_{l=0}^{p} a_l \sum_{l'=0}^{l} (-1)^{l'} \binom{l}{l'} (p-j)^{l-l'} S_{l'}^{p-j} \tag{39}$$

where

$$S_{l'}^m = \sum_{k=0}^{m} (-1)^k \binom{m}{k} k^{l'}. \tag{40}$$

One can verify that $S_{l'}^m = 0$ for $l' < m$. It follows that

$$d_0^{p-j} = \sum_{l=p-j}^{p} a_l \sum_{l'=p-j}^{l} (-1)^{l'} \binom{l}{l'} (p-j)^{l-l'} S_{l'}^{p-j} \tag{41}$$

$$= a_{p-j} (-1)^{p-j} S_{p-j}^{p-j} \tag{42}$$

$$+ \sum_{l=p-j+1}^{p} a_l \sum_{l'=p-j}^{l} (-1)^{l'} \binom{l}{l'} (p-j)^{l-l'} S_{l'}^{p-j} \tag{43}$$

The contributions to d_0^{p-j} are determined by the coefficients from $p-j$ to p. This leads to the following recursion:

$$a_p = \frac{d_0^p}{(-1)^p S_p^p} \tag{44}$$

$$a_{p-j} = \left\{ \frac{d_0^{p-j} - \sum_{l=p-j+1}^{p} \widetilde{a}_l \sum_{l'=p-j}^{l} (-1)^{l'} \binom{l}{l'} (p-j)^{l-l'} S_{l'}^{p-j}}{(-1)^{p-j} S_{p-j}^{p-j}} \right. \text{ for } j = 1, \cdots, p \tag{45}$$

References

[1] Doron Swade. *The Cogwheel Brain*. Little, Brown, and Company, 2000.

[2] Doron D. Swade. The construction of charles babbage's difference engine no. 2. *Annals Hist. Comput.*, 5(1058-6180):70–87, 2005.

[3] I. Grattan-Guinness. Work for the hairdressers: The production of de prony's logarithmic and trigonometric tables. *Annals Hist. Comput.*, 12(3):177–185, 1990.

[4] Roegel. *A reconstruction of Charles Babbage'stable of logarithms (1827)*. LOCOMAT project:http://locomat.loria.fr, 2012.

[5] Thomas Haigh, Mark Priestley, and Crispin Rope. *ENIAC in Action*. MIT Press, 2016.

[6] Betty Alaxandra Toolel. *ADA The Enchantress f Numbers*. Critical Connection, 2010.

[7] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, Inc., 1965.