Accurate Scaled Summation: Identifying a method that reduces Floating Point Error

Emmet McDonald

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master in Computer Science

Supervisor: David Gregg

April 2024

Accurate Scaled Summation: Identifying a method that reduces Floating Point Error

Emmet McDonald, Master in Computer Science
University of Dublin, Trinity College, 2024

Supervisor: David Gregg

In any summation problem, Floating Point Error can occur, which reduces the accuracy of the final output. This is obviously undesirable as these outputs are usually used, and their accuracy would lead to better results. This project specifically deals with Scaled Summation problems, where the inputs being summed are weighted by known weights. Such a project is important in regards to the topic of Computer Science as many neural networks can be seen as a series of interconnected Scaled Summation problems, and reducing the Floating Point Error present in them will only lead to more accurate models being produced. In this paper I discuss several summation methods and their effect on the final output's Floating Point Error, and I introduce two key ideas with an aim to reduce this Floating Point Error further. While the first idea, which relies on the Expected Mean of the outcome, generally introduces ∼12-20% more absolute error and ∼30% more relative error, the second idea, which creates a "Hyper"-Sorted permutation of weights for Pairwise Summation, generates only ∼66% of the absolute error and only ∼38% of the relative error that would be generated with an unsorted set of weights..

Acknowledgments

Thank you to my supervisor, Dr. David Gregg, for his guidance and advice throughout the year.

Thank you to my family for their support and care for the past 5 years, (not to mention the previous 19 years!).

And thank you to my fellow students and to the staff of the School of Computer Science and Statistics for providing me with this education.

Emmet McDonald

University of Dublin, Trinity College
April 2024

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Emmet McDonald

April 17, 2024

Contents

Abstract **i**

Acknowledgments **ii**

Chapter 1 Motivation **1**

1.1 Explanation of Terms 1

1.1.1 Floating Point Error 1

1.1.2 Scaled Summation 2

1.2 Reducing Floating Point Error 2

1.2.1 Maximum Rounding Error 2

1.2.2 Catastrophic Cancellation 3

1.2.3 Partial Sums 4

1.3 Other Information & Assumptions 5

1.3.1 Distributions 5

1.3.2 Scale of Inputs 6

Chapter 2 Methods of Summation **7**

2.1 Weights 7

2.2 Simple 8

2.3 Huffman 13

2.4 Pairwise 18

2.5 Interval 25

2.6 Summary 33

Chapter 3 Design & Implementation **34**

3.1 Using the Expected Mean 34

3.1.1 Design 34

3.1.2 Implementation 35

3.2 "Hyper"-Sorted Pairwise 42

3.2.1 Design 42

3.2.2 Implementation 42
3.3 Summary 46

Chapter 4 Evaluation **47**
4.1 Experiments 47
4.1.1 Batch 1 (Normal Distribution, $\mu = 0.5, \sigma = \frac{1}{6}$) 49
4.1.2 Batch 2 (Uniform Distribution) 55
4.1.3 Batches 1 & 2 Combined 61
4.2 Results 62
4.3 Summary 66

Chapter 5 Conclusions & Future Work **67**
5.1 Future Work 68

Bibliography **69**

List of Tables

4.1 Batch 1_1 Results table 49
4.2 Batch 1_2 Results table 49
4.3 Batch 1_3 Results table 50
4.4 Batch 1_4 Results table 50
4.5 Batch 1_5 Results table 51
4.6 Batch 1_6 Results table 51
4.7 Batch 1_7 Results table 52
4.8 Batch 1_8 Results table 52
4.9 Batch 1_9 Results table 53
4.10 Batch 1_{10} Results table 53
4.11 Batch 1 Accumulated Results table 54
4.12 Batch 1 Adjusted Accumulated Results table 54
4.13 Batch 2_1 Results table 55
4.14 Batch 2_2 Results table 55
4.15 Batch 2_3 Results table 56
4.16 Batch 2_4 Results table 56
4.17 Batch 2_5 Results table 57
4.18 Batch 2_6 Results table 57
4.19 Batch 2_7 Results table 58
4.20 Batch 2_8 Results table 58
4.21 Batch 2_9 Results table 59
4.22 Batch 2_{10} Results table 59
4.23 Batch 2 Accumulated Results table 60
4.24 Batches 1 & 2 Accumulated Results table 61
4.25 Batches 1 & 2 Adjusted Accumulated Results table 61
4.26 Table 4.11 Percentage Comparison 62
4.27 Table 4.12 Percentage Comparison 62
4.28 Table 4.23 Percentage Comparison 63
4.29 Table 4.24 Percentage Comparison 64

4.30 Table 4.25 Percentage Comparison 64
4.31 Expected Mean Evaluation 65
4.32 Pairwise Percentage Comparison 65

List of Figures

1.1 Scaled Summation Diagram . . . 2
1.2 Simple Summation Visualisation . . . 4
1.3 Pairwise Summation Visualisation . . . 5

2.1 Simple Summation Visualisation . . . 9
2.2 Simple Example (Negative Output) . . . 10
2.3 Simple Example (Positive Output) . . . 11
2.4 Simple Example (Output Close to Zero) . . . 12
2.5 Huffman Summation Visualisation . . . 13
2.6 Huffman Example (Negative Output) . . . 15
2.7 Huffman Example (Positive Output) . . . 16
2.8 Huffman Example (Output Close to Zero) . . . 17
2.9 Pairwise Summation Visualisation (Random Order) . . . 18
2.10 Pairwise Summation Visualisation (Worst Case Scenario) . . . 19
2.11 Pairwise Summation Visualisation (Sorted) . . . 20
2.12 Pairwise Example w/ Default Permutation (Negative Output) . . . 21
2.13 Pairwise Example w/ "Simple"-Sorted Permutation (Negative Output) . . . 22
2.14 Pairwise Example w/ Default Permutation (Positive Output) . . . 22
2.15 Pairwise Example w/ "Simple"-Sorted Permutation (Positive Output) . . . 23
2.16 Pairwise Example w/ Default Permutation (Output Close to Zero) . . . 23
2.17 Pairwise Example w/ "Simple"-Sorted Permutation (Output Close to Zero) 24
2.18 $\text{Interval}_{\text{LT}}$ Summation Visualisation . . . 25
2.19 $\text{Interval}_{\text{GT}}$ Summation Visualisation . . . 26
2.20 $\text{Interval}_{\text{LT}}$ Example (Negative Output) . . . 27
2.21 $\text{Interval}_{\text{GT}}$ Example (Negative Output) . . . 28
2.22 $\text{Interval}_{\text{LT}}$ Example (Positive Output) . . . 29
2.23 $\text{Interval}_{\text{GT}}$ Example (Positive Output) . . . 30
2.24 $\text{Interval}_{\text{LT}}$ Example (Output Close to Zero) . . . 31
2.25 $\text{Interval}_{\text{GT}}$ Example (Output Close to Zero) . . . 32

3.1 Output distributions for a Scaled Summation 35
3.2 Simple Example w/ Expected Mean 38
3.3 Huffman Example w/ Expected Mean 39
3.4 Pairwise Example w/ Expected Mean (Default Permutation) 40
3.5 Pairwise Example w/ Expected Mean ("Simple"-sorted Permutation) . . . 40
3.6 $\text{Interval}_{\text{LT}}$ Example w/ Expected Mean 41
3.7 $\text{Interval}_{\text{GT}}$ Example w/ Expected Mean 41
3.8 Pairwise Summation Visualisation ("Hyper"-Sorted) 42
3.9 Pairwise Example w/ "Hyper"-Sorted Permutation (Negative Output) . . . 44
3.10 Pairwise Example w/ "Hyper"-Sorted Permutation (Positive Output) . . . 45
3.11 Pairwise Example w/ "Hyper"-Sorted Permutation (Output Close to Zero) 45

Chapter 1

Motivation

Introduction to the material covered in the dissertation, wherein I explain the terms in my Dissertation title, and discuss the ways Floating Point Error can be reduced, as well as any assumptions I made in the pursuance of this project.

1.1 Explanation of Terms

Two key elements of this project, Floating Point Error and Scaled Summation, require a level of explanation before one can dive into understanding the project (and this dissertation) properly.

1.1.1 Floating Point Error

Consider the value $\frac{1}{3}$, which can be written in binary notation as $0.\overline{01}_2$. Limitations of physics mean that a computer cannot store the infinite bits required to accurately depict $\frac{1}{3}$, and so, attempting to store such a number would result in a round-off, or rounding error of $0.\overline{01}_2 * 2^x$, where x would depend on the finite amount of bits used for fidelity. While the loss of fidelity may seem inconsequential, as the value lost is a tiny fraction of the output, these small errors can lead to major problems, as seen in Lorenz (1963); Parker (2019). Of course, the rational nature of $\frac{1}{3}$ means that it could simply be stored as a numerator, 1, and a denominator, 3, and any operations upon it involving other rational numbers could use similar number representation. However, this standard of representing numbers falls apart when dealing with irrational numbers like π or e, and it is known that there are more irrational numbers than rational numbers (Cantor, 1879, 1891). Because of this it is standard to use the IEEE 754 Standard for Floating Point Arithmetic.(IEEE 754-2019, 2019)

According to IEEE 754-2019 (2019), single-precision floating point numbers are represented as a 32-bit value, where bit 0 refers to the sign, bits 1-8 refer to the exponent (x from the prior example), and the remaining 23 bits refer to the mantissa, where any fidelity of the number is expressed.

Floating Point Error (FPE) encompasses rounding error, but can be greater as the FPE in a value a, where $a = b + c$ and (a, b, c) are floating point numbers, is equal to the rounding error of a plus the rounding errors of b and c.

1.1.2 Scaled Summation

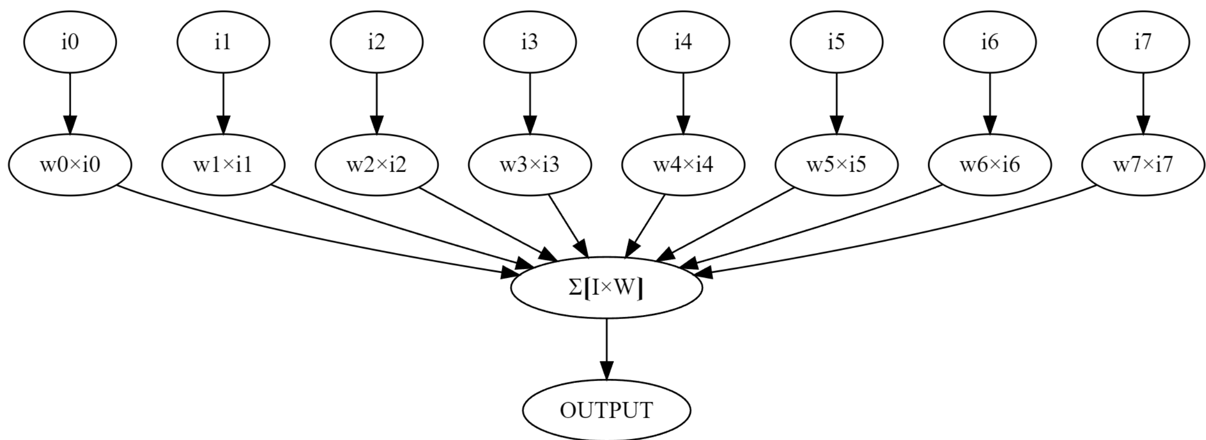

Figure 1.1: A visual aid to understand how Scaled Summation works; values *i0 - i7* represent inputs, which are multiplied by weights *w0-w7* and then summed together, to produce an output

Often referred to as a 'Weighted Sum Model', a Scaled Summation is a summation of multiple inputs, which are multiplied by associated weights before being added together to produce an output (Fishburn, 1967). This process is depicted in Figure 1.1.

Scaled Summation problems may be recognisable to the reader as the basis of most Neural Networks, where outputs of nodes are multiplied by weights and added together in order to produce either values for nodes in different layers, or the ultimate output value(s) of the network (Zell, 1994).

1.2 Reducing Floating Point Error

1.2.1 Maximum Rounding Error

There are several key pieces of information with regards to Floating Point Error that we must consider, which will help give the reader insight into why different methods of

summation can produce different amounts of Floating Point Error.

The first such piece of information is that the maximum rounding error of any given value is known. That is to say, assuming we are attempting to store a known floating point value in a limited number of bits, we know the maximum possible rounding error is $\frac{1}{2}$ the value of the Unit of Least Precision (ULP) (Goldberg, 1991).

The Unit of Least Precision refers to the 'lowest value' bit of the floating point value which, in our examples, would be the rightmost bit in the mantissa, the value of which is influenced by the exponent. Specifically, the maximum rounding error of a given floating point value is $1 * 2^{x-24}$, where x is the exponent of said value.

What this ultimately means is that, the closer a floating point value is to 0, the lower its maximum rounding error is. This immediately presents a target for any methods created to solve this problem - if we keep our partial sums as close to 0 as possible, FPE should be reduced.

It is important to note that our value of $1 * 2^{x-1}$ is only the maximum rounding error of a floating point value. When we are considering the total FPE, then the maximum FPE would be $1 * 2^{x-1} + FPE_1 + FPE_2$, where FPE_1 is the existing FPE of one value, and FPE_2 is the existing FPE of the other.

1.2.2 Catastrophic Cancellation

A key phenomenon to avoid in floating point arithmetic in order to reduce FPE is Catastrophic Cancellation (Goldberg, 1991). This cancellation occurs when two floating point numbers of similar value and opposite sign are added together. For example consider a partial sum $0.6 - 0.600000001$, the answer is -0.000000001 but, the IEEE representations of 0.6 and 0.600000001 are both 0.60000002384185791015625, meaning the value of the answer is completely lost, even though -0.000000001 can be approximated in the IEEE 754 standard. Of course, not every case of Catastrophic Cancellation completely eliminates the answer but knowing that the lower value bits of the mantissa are most affected by FPE, and assuming the higher value bits are cancelling each other out, we can see the relative error of the answer rising quickly.

It is important to note that some amount of 'Benign' Cancellation is still ideal in reducing overall FPE (Goldberg, 1991), as the summation of a positive and negative number does always generate an output with a lower absolute value than either initial number, which will reduce rounding error, as discussed in Section 1.2.1. Catastrophic Cancellation should still be avoided if possible but, in practice, a balance of 'Benign' and Catastrophic Cancellation will probably occur in most summations.

1.2.3 Partial Sums

The previous subsections make mention of 'partial sums', this term refers to the addition of two values that must be completed as part of the overall scaled summation. Different methods of summation for the same base values can lead to different partial sums being added, which can reduce the accumulation of FPEs.

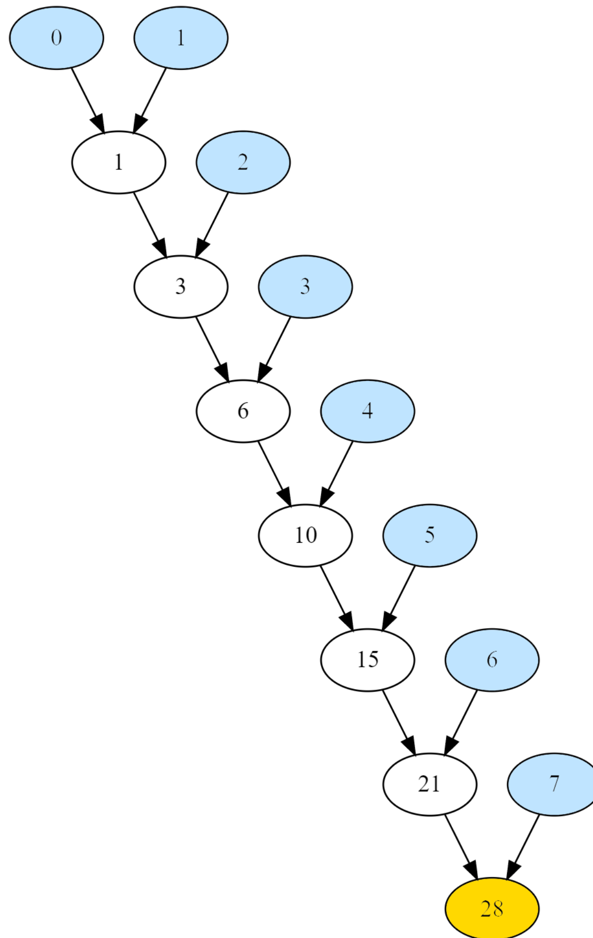

Figure 1.2: Visualisation of 'simple' summation of the numbers 0-7, inputs are highlighted in blue and the output (28) is highlighted in orange

To better understand this, we will consider two summation methods, 'simple' summation, which can be written as $(((((((x_0+x_1)+x_2)+x_3)+x_4)+x_5)+x_6)+x_7)$, and pairwise summation, which can be written as $(((x_0+x_1)+(x_2+x_3))+((x_4+x_5)+(x_6+x_7)))$. These two methods are visualised in Figures 2.1 and 1.3, respectively, where the values $x_0 - x_7 = 0 - 7$.

Based on our knowledge that the maximum FPE of a node is $\frac{ULP}{2} + FPE_1 + FPE_2$, we can see that the pairwise summation method reduces the possible values of FPE_1 and FPE_2, by not letting those values accumulate as much as in the simple summation

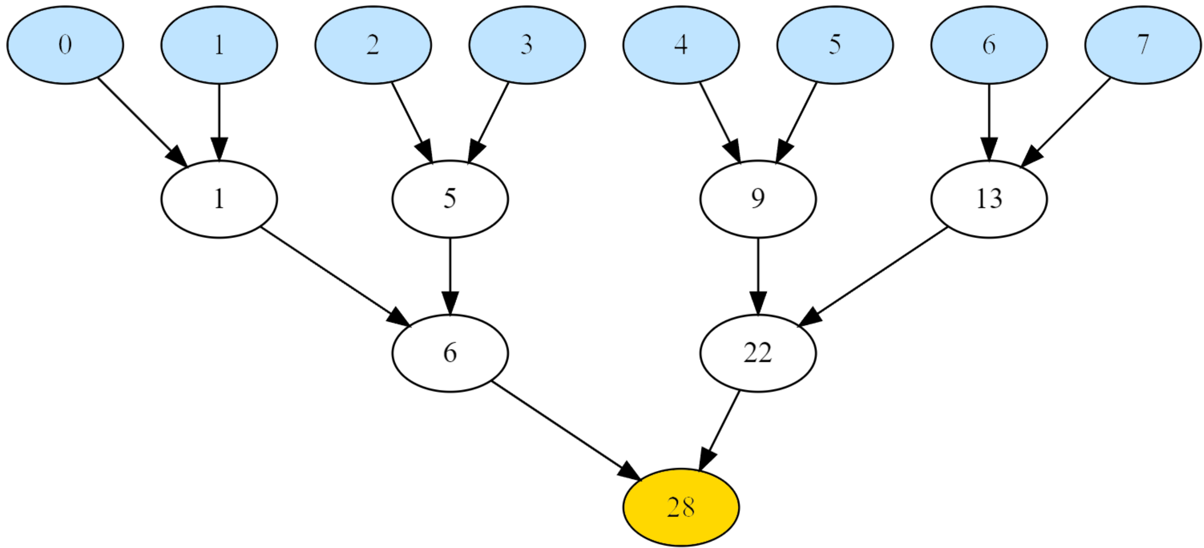

Figure 1.3: Visualisation of 'pairwise' summation of the numbers 0-7, inputs are highlighted in blue and the output (28) is highlighted in orange

method. The depth of the pairwise tree is 3 (or $log_2(n)$, where n = number of inputs), while the depth of the simple tree is 7 (or $n - 1$, where n = number of inputs).

1.3 Other Information & Assumptions

We have already established that different methods of summation can reduce the 'maximum' FPE of the output of our scaled summation, but there is other information that can be gleaned about the values to add (the inputs multiplied by their relevant weights), that can be used when attempting to derive new summation methods.

1.3.1 Distributions

When working on this project, I decided that the random inputs should be considered to be normally distributed. The reasoning for this is as follows:

Firstly, assuming that our scaled summation is occurring within a neural network, where the inputs are simply the outputs of other scaled summation problems, we know that, if that scaled summation's inputs are normally random, its output is also normal as the sum of independent normally distributed random variables is itself normally distributed (Weisstein, 2024).

Similarly, if that scaled summation's inputs were somehow uniformly distributed, the output can be treated as being normally distributed. This output actually follows the

Irwin-Hall distribution (Irwin, 1927; Hall, 1927) but, by the Central Limit Theorem, as the number of inputs n increases, this distribution approaches the normal distribution.

With this knowledge that any Scaled Summation problems 'inside' a neural network will have random inputs that are normally distributed, we will give more weight to results of tests run on different summation methods wherein the inputs are generated from a normal distribution as opposed to a uniform one. Tests where the inputs are generated from a uniform distribution will still be run, as the initial layers of a neural network would receive their inputs from the 'outside', and so it is worthwhile to see how different summation methods would work with an approximation of those such values.

1.3.2 Scale of Inputs

Other information about the inputs (after multiplication by their respective weights), is incredibly sparse but we can make some semi-confident assumptions in regards to their sizes.

Since the inputs themselves are normally distributed in the range (0,1), we know that, in a general sense, the inputs are likely to be 0.5, or relatively close to 0.5. Obviously we're working in a very fuzzy area here, but using this assumption that all the inputs (before multiplication) are roughly the same we can just look at the known weights when attempting to create a summation method that reduces FPE. After all, we can't 'see' the inputs themselves until the summation method is being tested, so we need to assume something about them.

This idea that the relationships between weights stay roughly the same before and after they are multiplied by the inputs can allow us to create summation methods that rely on ordering these weights by size, for example.

Chapter 2

Methods of Summation

I began this project by exploring different existing methods of summation, such as 'Simple', Huffman, and Pairwise Summation. I was also provided a codebase as worked on by Professor David Gregg, wherein he had explored another summation method; Interval. Throughout this chapter I will describe each individual method using a simple, integer graph, and then analyse those methods when used on real floating point weights generated when testing the methods.

2.1 Weights

It is important to note that the 'outputs' named here are only the output of a scaled summation with these weights, if all inputs are equal to 1. We will be following this assumption when examining all graphs in this chapter, as the values of the inputs for these analyses is arbitrary, as long as they are relatively similar, according to the assumption made in Section 1.3.2. In practice, the actual output of Scaled Summations roughly follow a normal distribution with $\mu = \frac{\text{Output}}{2}$ (This is discussed further in Section 3.1.1).

Negative: (-0.992312, -0.858598, -0.827232, -0.762807, -0.693924, -0.628675, -0.594080, -0.556399, -0.408592, -0.379897, -0.336798, -0.047263, 0.506048, 0.747933, 0.769607, 0.969195). **Output** = -0.4093795

Positive: (-0.911727, -0.778005, -0.637901, -0.253816, -0.062501, 0.105751, 0.167028, 0.193899, 0.210779, 0.244602, 0.514353, 0.584487, 0.710941, 0.804684, 0.915598, 0.987290). **Output** = 2.795462

Close to Zero: (-0.920034, -0.804572, -0.526475, -0.332712, -0.326809, -0.303425, -0.225544, 0.011541, 0.019537, 0.081116, 0.292802, 0.322470, 0.340642, 0.648018, 0.846174, 0.912863). **Output** = 0.035592

These three weight sets were curated based on their outputs, so that the reader can see how a 'standard' negative or positive output would be found, as well as an scenario where most of the weights cancel each other out, bringing the output close to zero.

While these graphs will generally follow to the logic derived and discussed when examining the visualisation figures, the greater number of initial weights (16 versus the visualisations' 8), as well as the randomly generated nature of those weights, has provided some features that are worth noting, and thus have been noted in their respective sections.

Across all of the weight set graphs, the 'positive' and 'negative' weight sets are both treated similarly, which is to be fully expected as all methods barring the initial sorting of values for the "simple"-sorted pairwise summation ignore the sign of the weights in favour of their absolute values.

2.2 Simple

This Simple Summation method is the 'default' summation method many programmers employ, as it can be implemented using very little code (As seen in Listing 2.1), and is very intuitive.

```
float output = 0
for(int i = 0; i < len(weights); i++){
    output = output + weight[i]
}
```

Listing 2.1: Pseudocode implementation of Simple Summation

However, it is also the least effective summation method with regards to FPE reduction. The depth of the Simple Summation tree is $n - 1$, where $n =$ number of inputs, and the tree is arranged in such a way that the FPE of each partial sum is that sum's rounding error plus the FPE of every previously calculated partial sum.

The simple summation graphs in all three weight sets (Figures 2.2, 2.3, and 2.4) are consistently the worst performing graphs for that weight set, based on maximum absolute partial sum value and tree depth, and their presence here is mostly as a 'worst case scenario' when comparing to the other example graphs in this chapter.

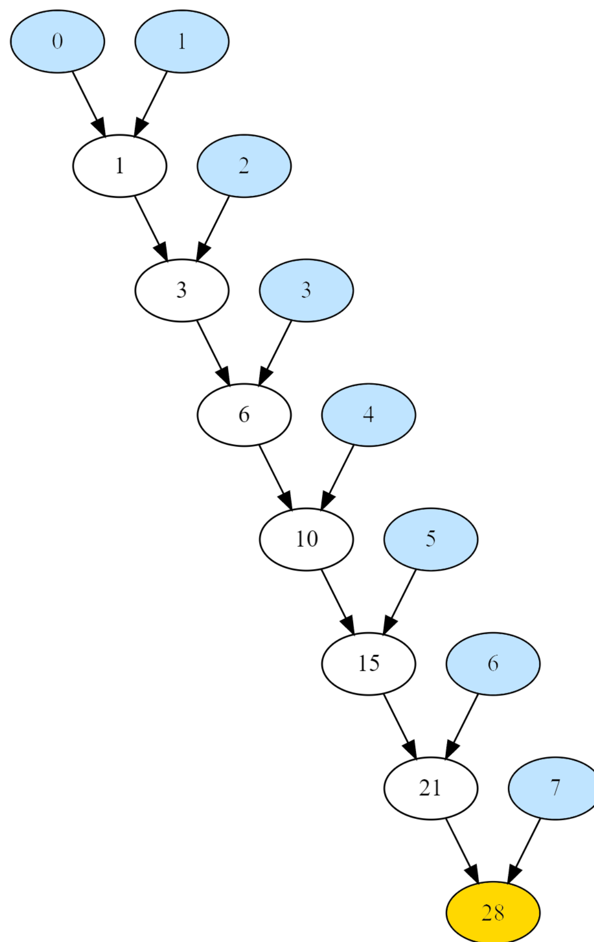

Figure 2.1: Visualisation of 'Simple' summation of the numbers 0-7, inputs are highlighted in blue and the output (28) is highlighted in orange.

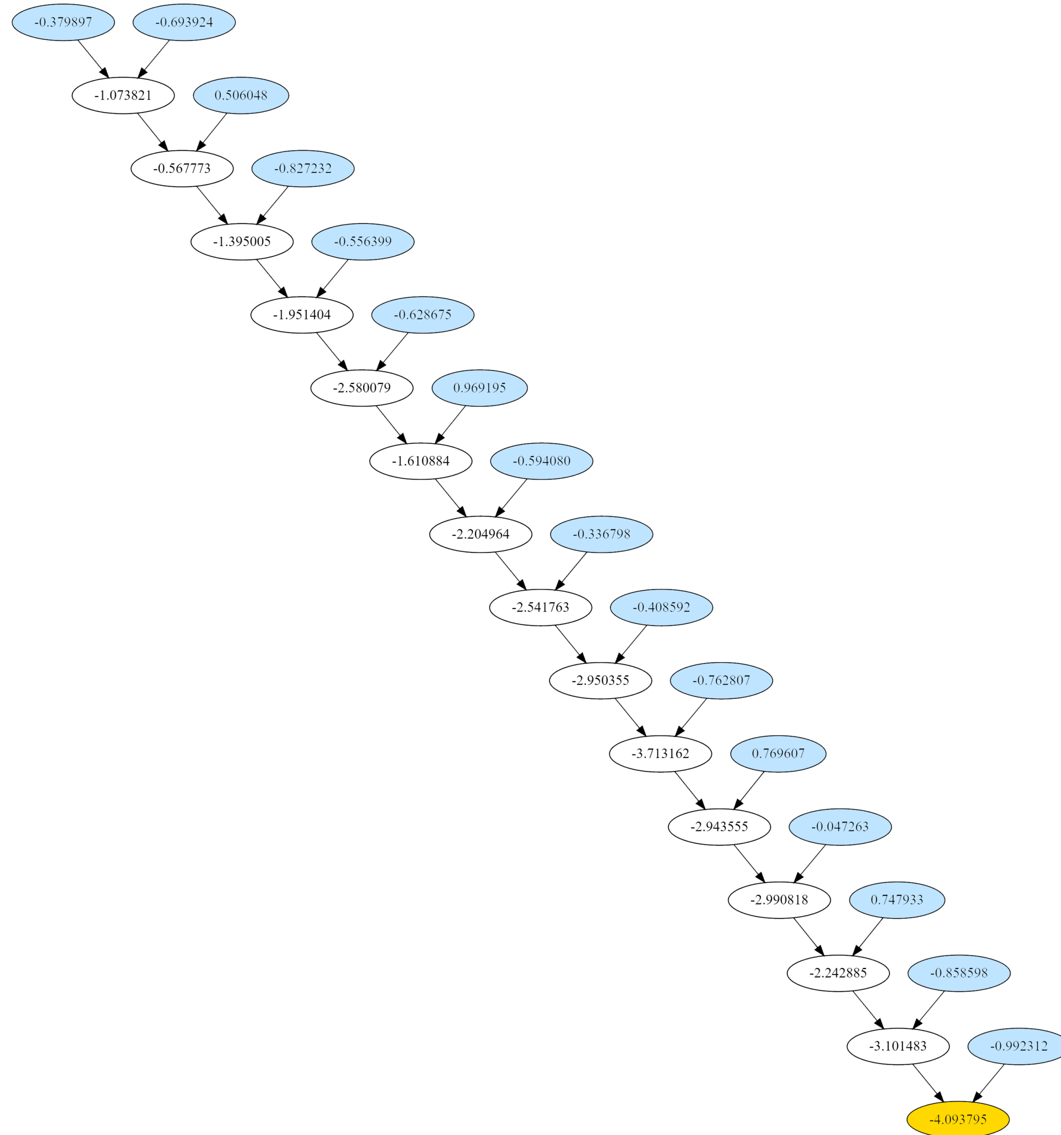

Figure 2.2: Real example of the Simple summation method, using the **'Negative'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

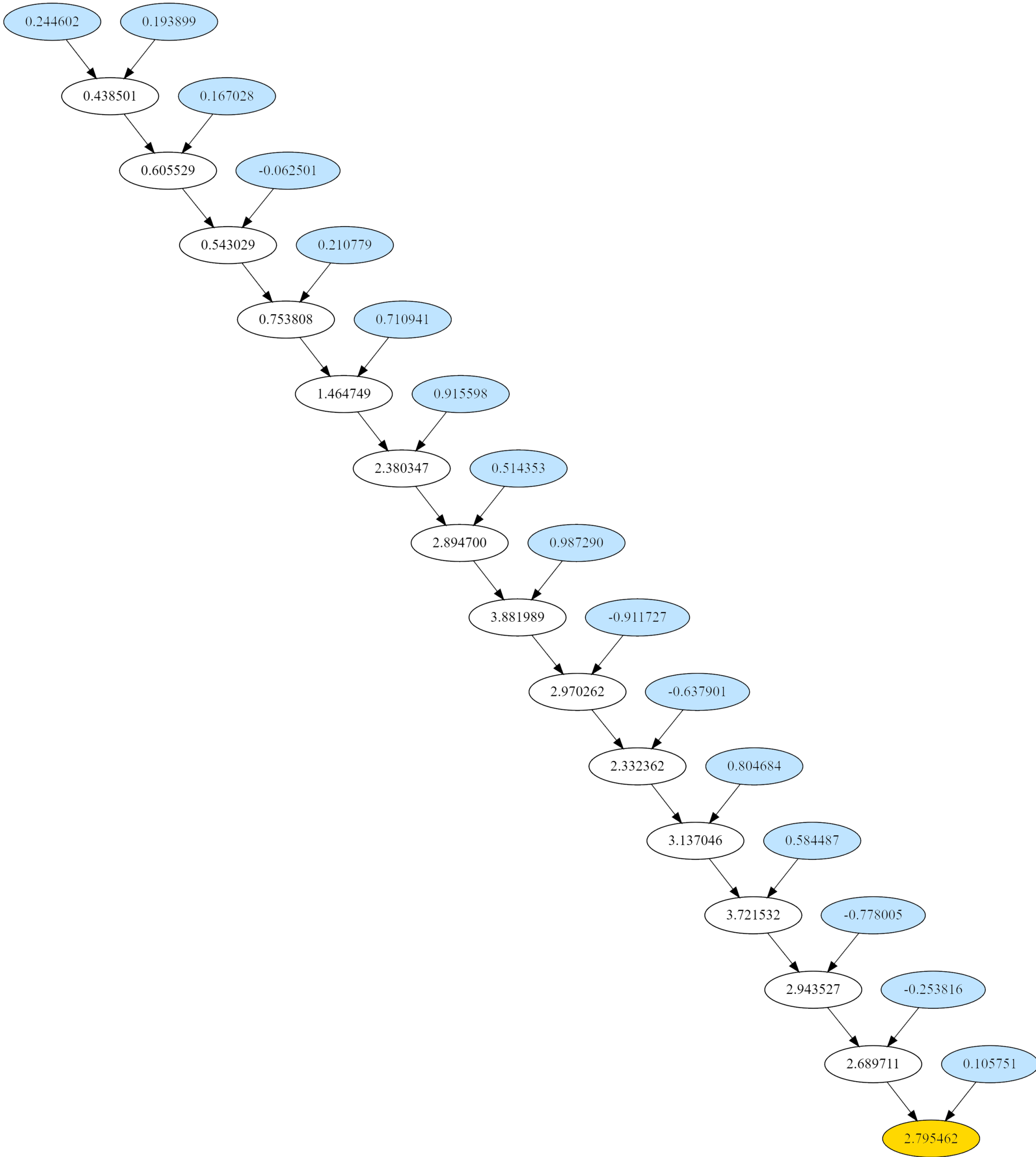

Figure 2.3: Real example of the Simple summation method, using the **'Positive'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

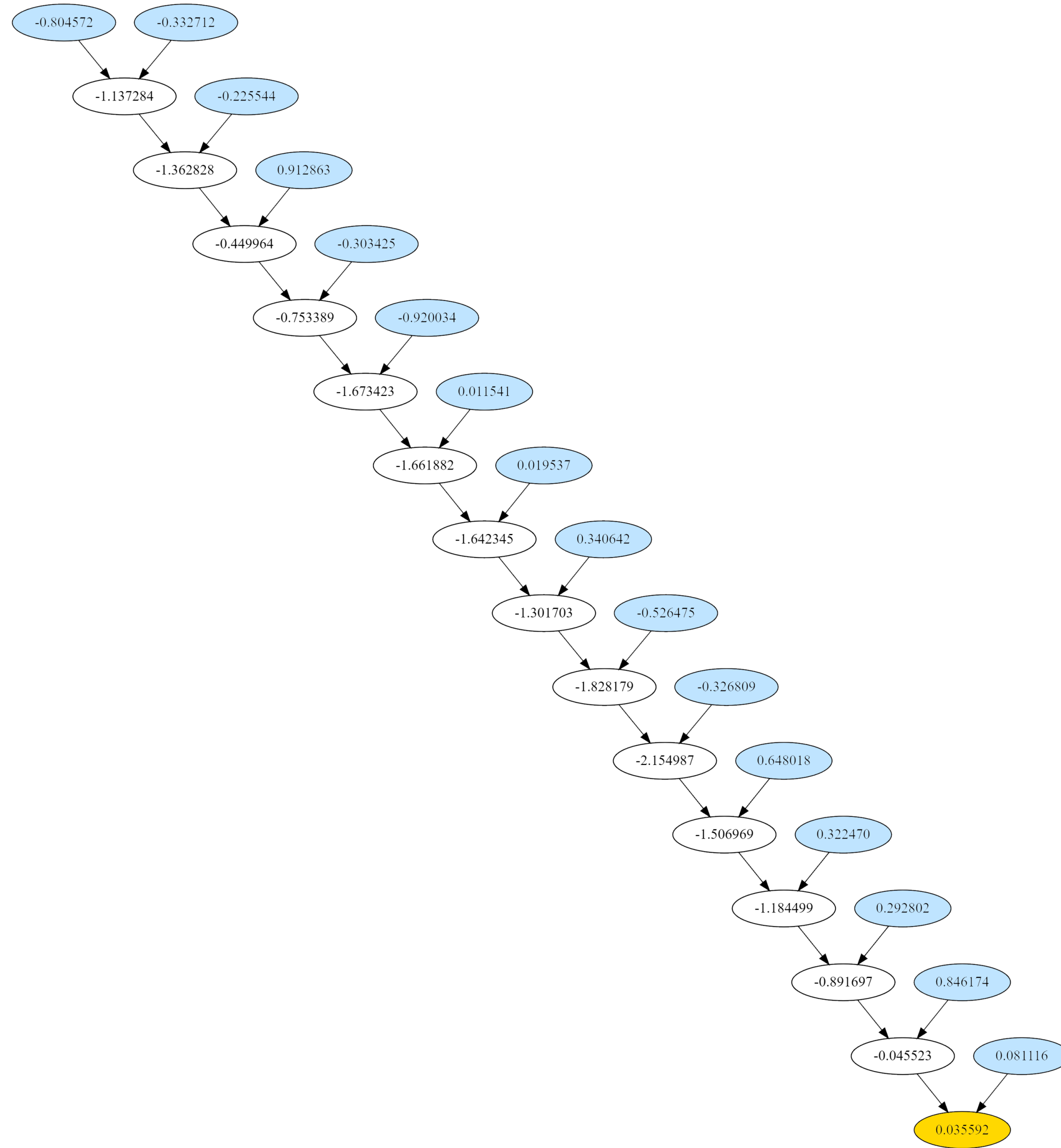

Figure 2.4: Real example of the Simple summation method, using the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

2.3 Huffman

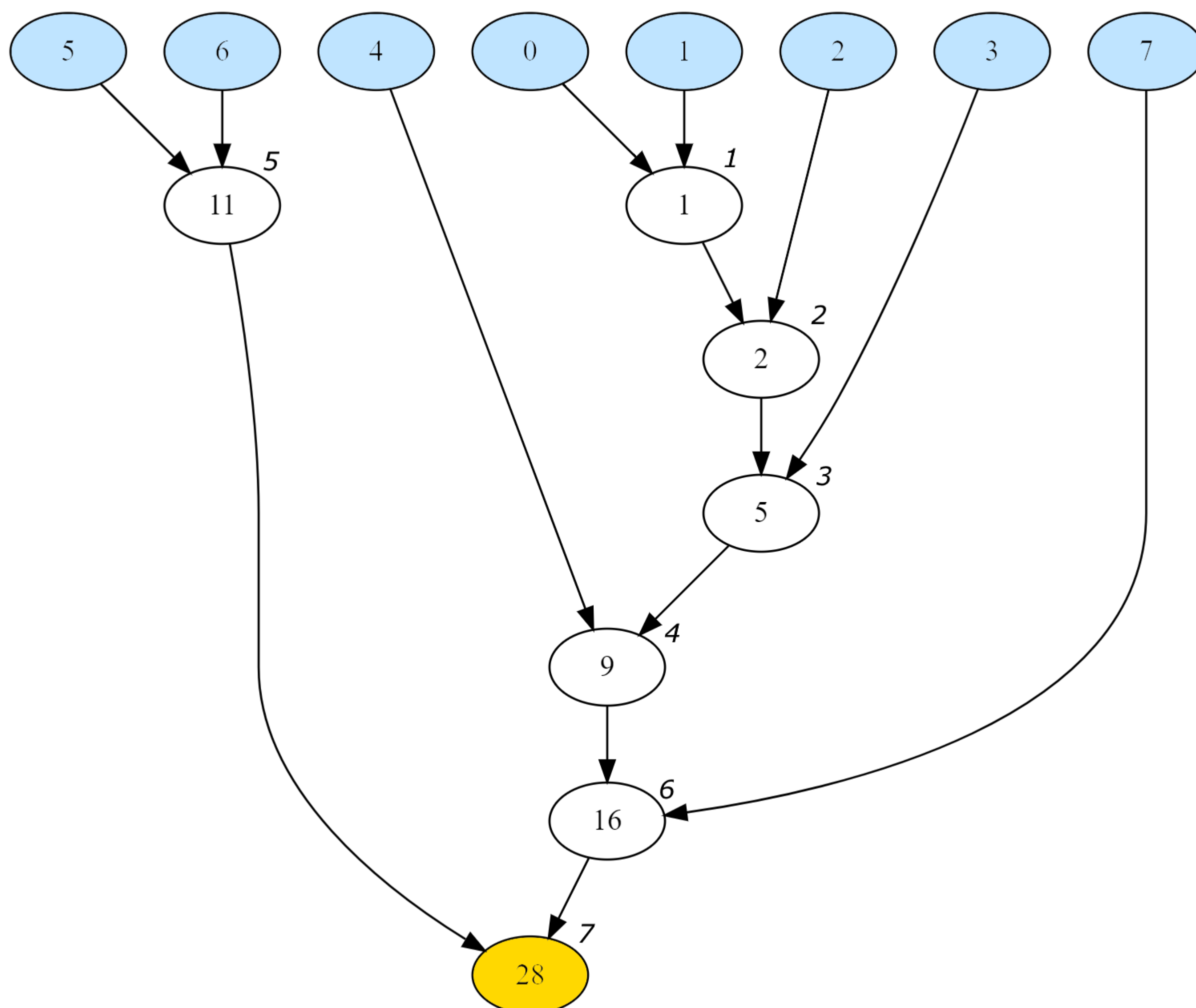

Figure 2.5: Visualisation of 'Huffman' summation of the numbers 0-7, inputs are highlighted in blue and the output (28) is highlighted in orange. Each partial sum is annotated to convey the order with which they were identified

The Huffman Summation method, is based on the idea of Huffman trees, as initially described in Huffman (1952). The idea to use Huffman trees for summation stems from Barabasz et al. (2020), a paper that looked at FPE in the Convolution of Deep Neural Networks. Specifically, this method attempts to build a summation tree using the weight values the same way that Huffman's trees use symbol frequency; by pairing the two 'lowest' values together. It is important to note that, while 'lowest' refers to the smallest value in a standard Huffman tree, the Huffman summation method used defines 'smallest' based on the weight's absolute value, as this was what was used in Barabasz et al. (2020). A pseudocode example of this method is can be seen in Listing 2.2

```
float[] summableOptions = weights
while(len(summableOptions) > 1) {
    val1 = findLowestAbsValue(summableOptions)
    val2 = findLowestAbsValue(summableOptions)
    partialSum = val1 + val2
    addToTree(val1, val2, partialSum)
    removeFromList(summableOptions, val1)
    removeFromList(summableOptions, val2)
    addToList(summableOptions, partialSum)
}
```

Listing 2.2: Pseudocode example for building a Huffman Summation Tree

A sample of the Huffman summation tree, for the weights 0-7 is shown in Figure 2.5. The partial sums are marked in the order they were identified as the 'lowest'.

Examining the Huffman Summation method's performance with the weight sets (assuming, as in all graphs in this chapter, that all inputs are 1), we can see that the depths of the generated graphs (Figures 2.6, 2.7, and 2.8), are very close to the depth of the sample tree, which is curious as the sample tree has half the initial values of the weight sets. This seems to indicate that the Huffman Method has a better performance when dealing with weights generated from a uniform distribution, which is backed up when examining how the weights have been grouped at the top of the graphs. Similar values (like -0.556399 and -0.594080 in Figure 2.6 or 0.193899 and 0.210779 in Figure 2.7) are added together earlier in the tree. It is worth noting that the approach of adding weights and nodes based on their absolute similarity does increase the risk of Catastrophic Cancellation (discussed in Section 1.2.2), this is most clearly shown in Figure 2.8, where 0.444392 and -0.40800 are summed together to produce the output 0.035592.

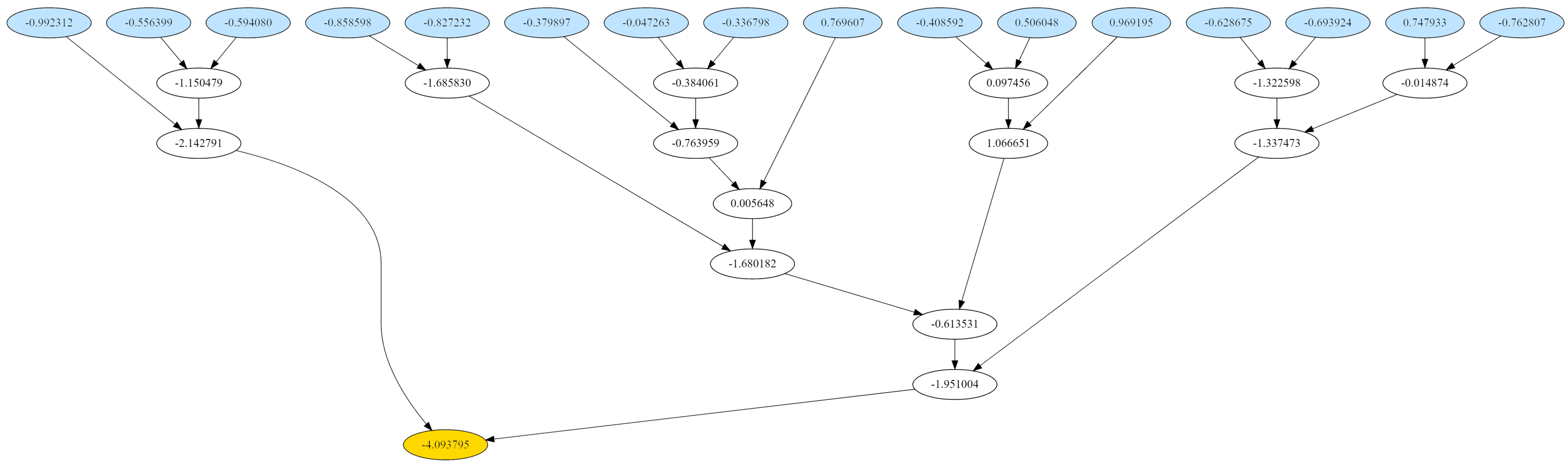

Figure 2.6: Real example of the Huffman summation method, using the **'Negative'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

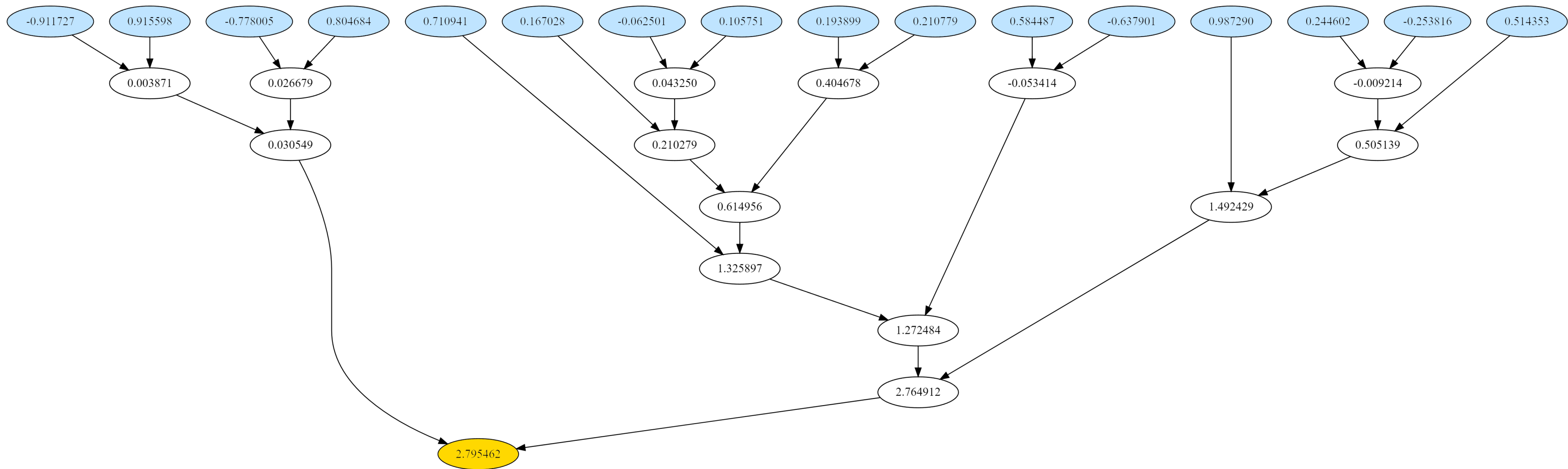

Figure 2.7: Real example of the Huffman summation method, using the **'Positive'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

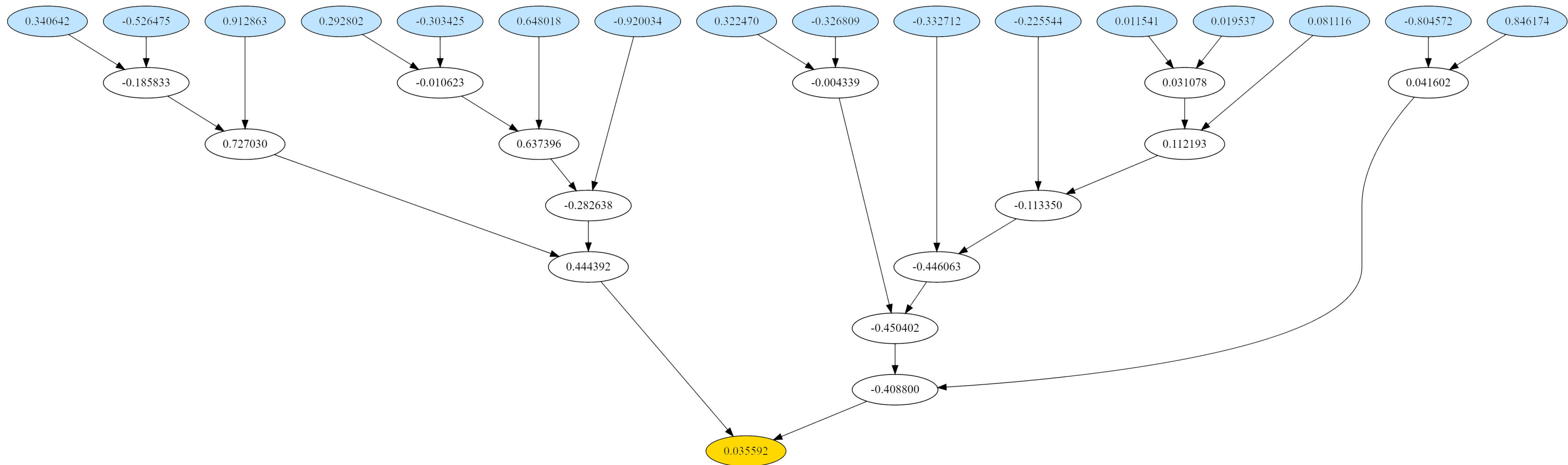

Figure 2.8: Real example of the Huffman summation method, using the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

2.4 Pairwise

A basic example of the pairwise summation method was shown in Section 1.2.3 with Figure 1.3, alongside a brief explanation of why its lower depth makes it a preferable method for reducing FPE. However a different set of weights $[-13, -8, -4, 0, 1, 9, 15, 16]$ will be used for the figures in this section.

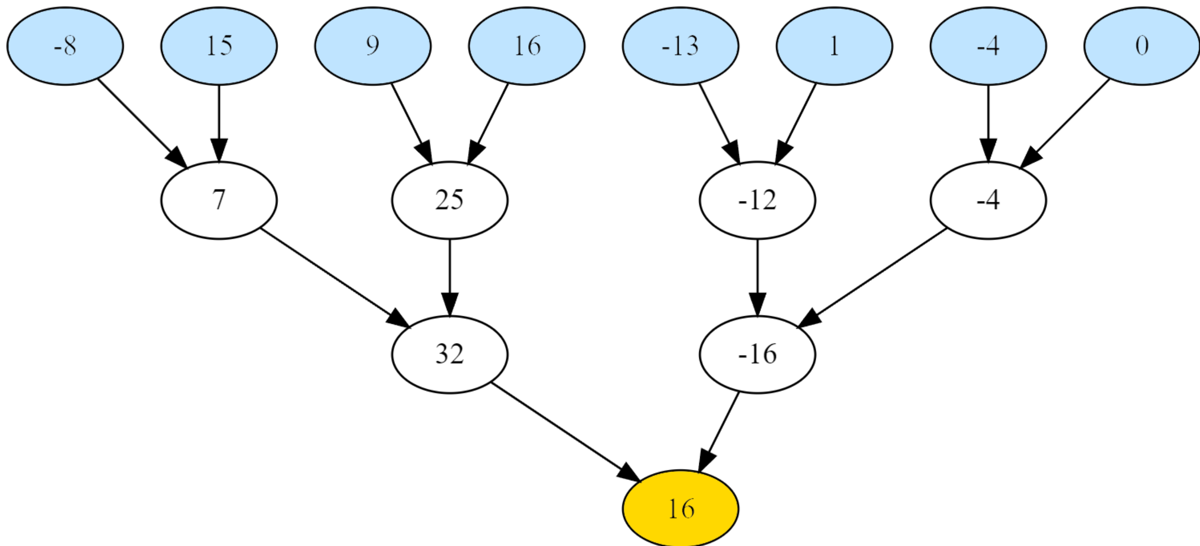

Figure 2.9: Visualisation of 'pairwise' summation of the weights $[-13, -8, -4, 0, 1, 9, 15, 16]$ in a random order, inputs are highlighted in blue and the output (16) is highlighted in orange.

Pairwise summation (McCracken and Dorn, 1964) works by building a 'pyramid' of partial sums, where each layer is made up of pairs from the previous layer. This is clear from Figures 2.9, 2.10, and 2.11. Something else clear from the aforementioned figures is that, although the sum of partial sums remains the same (32) no matter the initial order of weights, that initial order still matters somewhat.

Figure 2.9 shows an expected permutation of the weights, one that is random. This permutation leads to a mix of absolutely large and small values in the partial sums, ranging from 32 to -4. The larger absolute values here are, as previously discussed, more likely to have large amount of FPE and should ideally be avoided if possible.

Figure 2.10 shows a different permutation of weights, where they are ordered from lowest value (-13) to highest value (16). This permutation is actually the worst case scenario in terms of absolute partial sum value, as it pairs the largest absolute values together in such a way that a positive number doesn't 'cancel out' a negative number until the final summation of the tree, where the absolutely large values 41 and -25 are summed together. While the likelihood of a set of random 'real' weights presenting in order like this is very low, it is worthwhile to realize that the default, random permutation

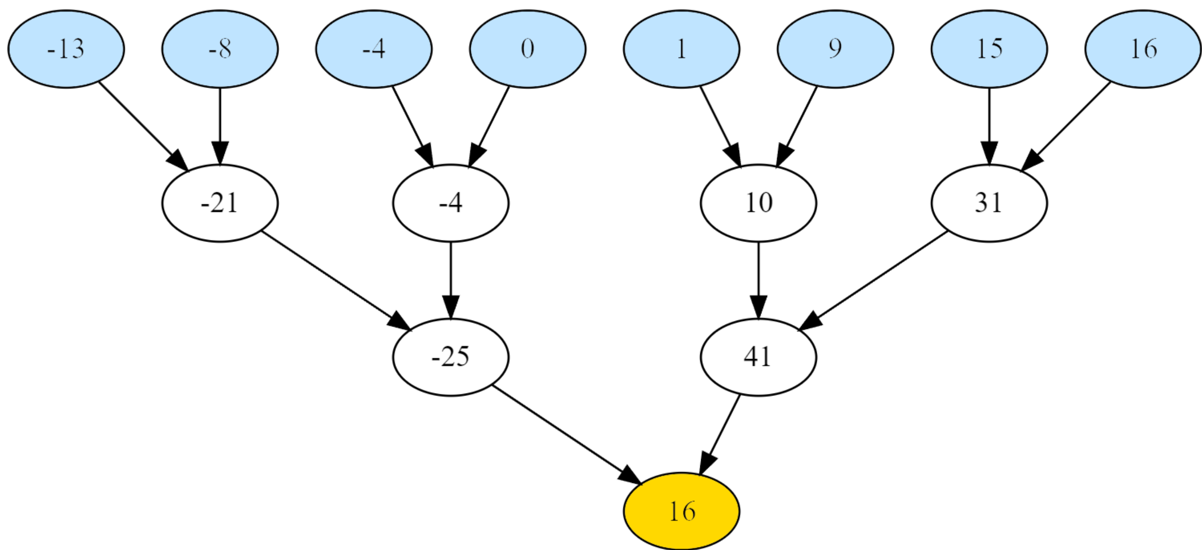

Figure 2.10: Visualisation of 'pairwise' summation of the weights $[-13, -8, -4, 0, 1, 9, 15, 16]$ ordered by the weights' values, inputs are highlighted in blue and the output (16) is highlighted in orange.

of weights could present this way, and so we must consider that a pairwise summation tree based on a random set of weights could be this bad.

Figure 2.11 shows a third permutation of weights, which I will define as "simple"-sorted, where they are sorted in an attempt to reduce these absolute partial values. Here, the initial permutation of weights is such that the greatest weight is paired with the least weight, the second-greatest weight is paired with the second-least weight, and so on. This produces relatively low absolute values for the partial sums, meaning that any FPE present would likely be lower than in Figures 2.9 or 2.10. It is also notable that all partial sums are of the same sign (in this case, positive), throughout the summation tree.

In the graphs on the following pages, you can see the performance of randomly permuted weights (Figures 2.12, 2.14, and 2.16) and the "simple"-sorted permutation of weights (Figures 3.9, 3.10, and 3.11) against the weight sets (in all 6 figures we are assuming that all inputs are 1). Across all three, the pairwise graphs have significantly less depth than the Huffman summation trees, although the maximum absolute values of most of the pairwise graphs are higher than in their Huffman counterparts. The lone exception is the "simple"-sorted pairwise summation tree for the Close to Zero weight set (Figure 2.17), which has a maximum partial sum value of 0.163904, versus the unsorted pairwise's maximum partial sum of 1.677937 and Huffman's maximum partial sum of 0.727030. It is also worth noting that, with the 'close to zero' weight set, the values in the second 'layer' of the "simple"-sorted graph (Figure 2.17 do not all hold the same sign, as the weights in this set don't all cancel each other out as cleanly as in the other sets.

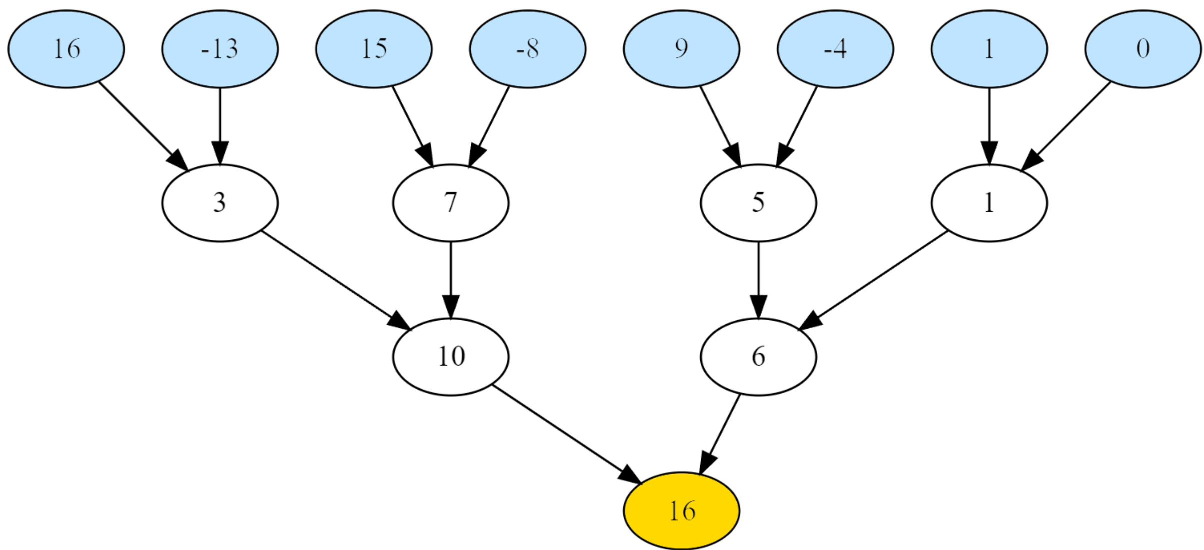

Figure 2.11: Visualisation of 'pairwise' summation of the weights $[-13, -8, -4, 0, 1, 9, 15, 16]$ sorted such that the high and low values are paired together, inputs are highlighted in blue and the output (16) is highlighted in orange.

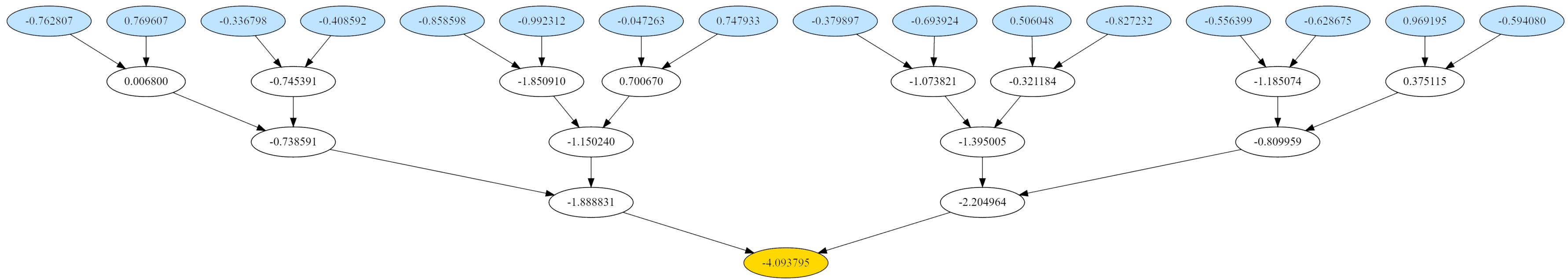

Figure 2.12: Real example of the Pairwise summation method, using the default permutation of the **'Negative'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

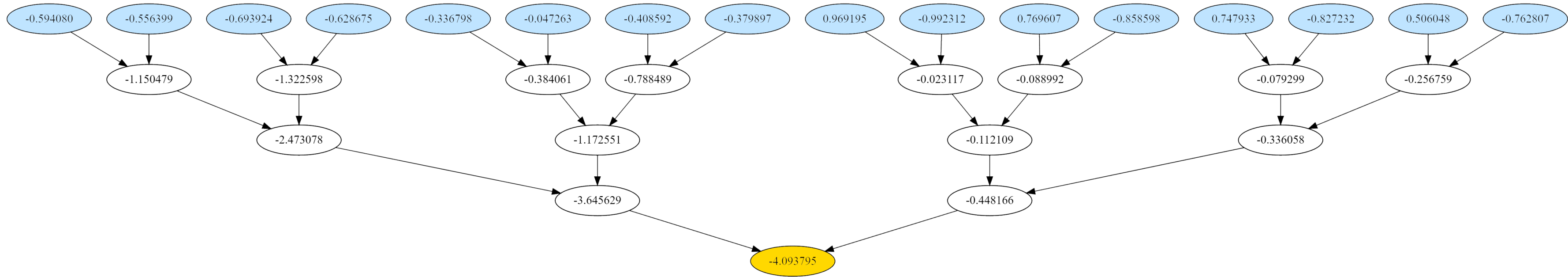

Figure 2.13: Real example of the Pairwise summation method, using the ”simple”-sorted permutation of the **’Negative’** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

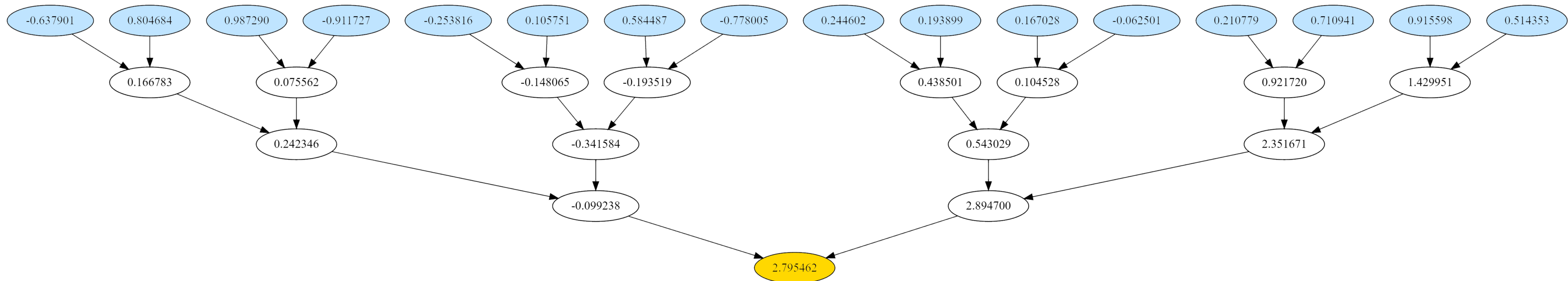

Figure 2.14: Real example of the Pairwise summation method, using the default permutation of the **’Positive’** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

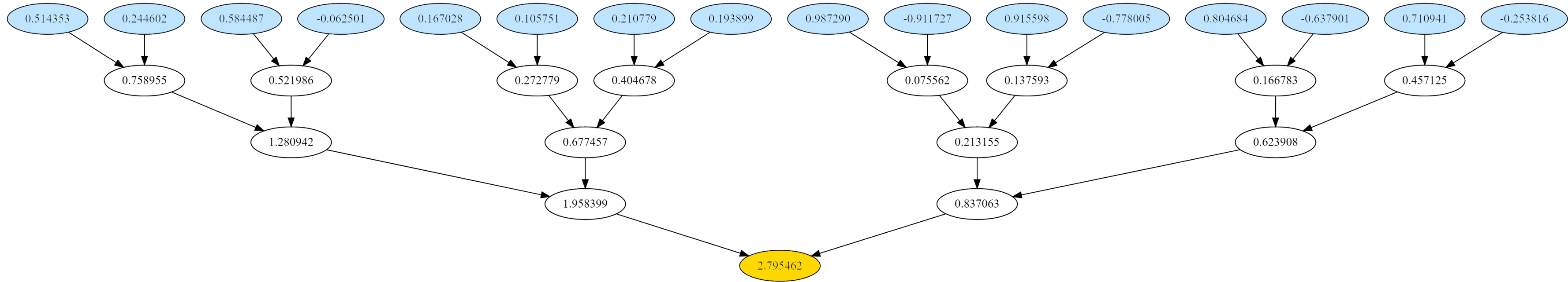

Figure 2.15: Real example of the Pairwise summation method, using the "simple"-sorted permutation of the **'Positive'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

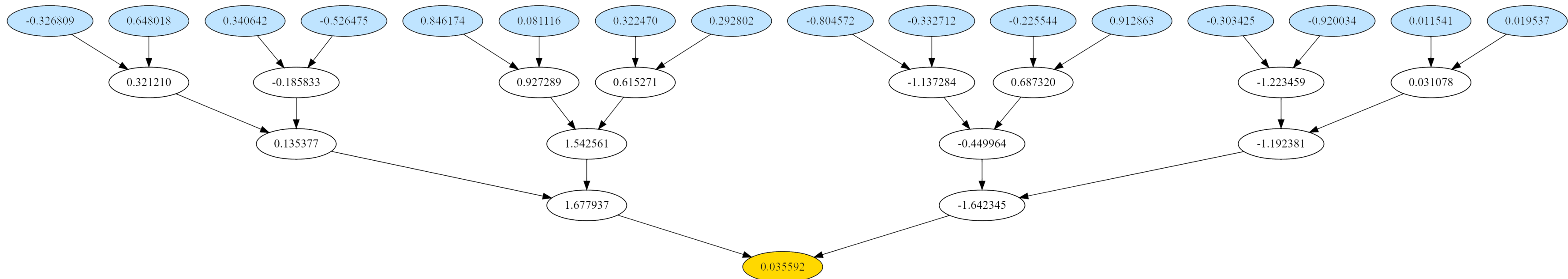

Figure 2.16: Real example of the Pairwise summation method, using the default permutation of the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

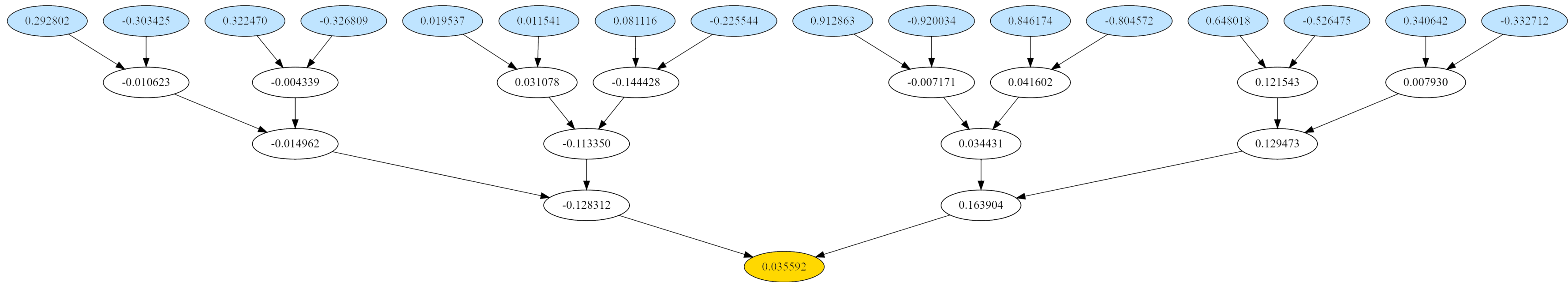

Figure 2.17: Real example of the Pairwise summation method, using the "simple"-sorted permutation of the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

2.5 Interval

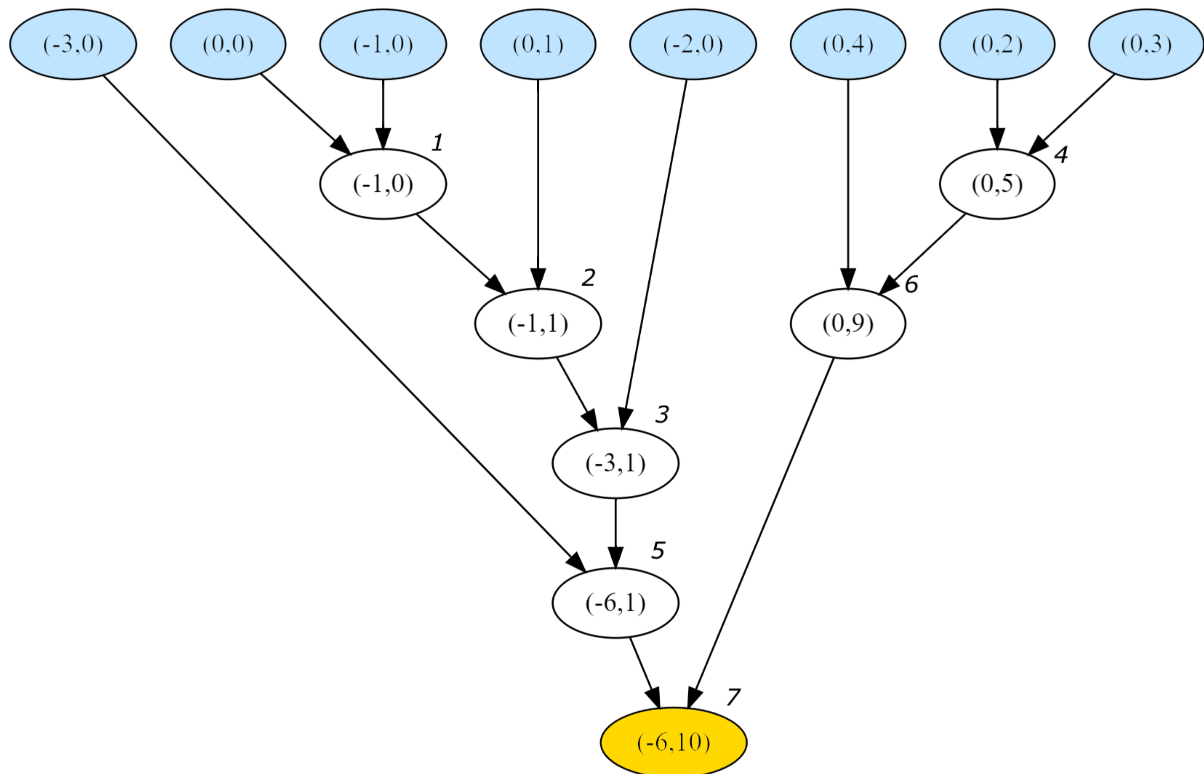

Figure 2.18: Visualisation of 'Interval$_{LT}$' summation of the numbers -3-4, inputs are highlighted in blue and the output (-6,10) is highlighted in orange. Each partial sum is annotated to convey the order with which they were identified

The 'interval' methods of summation aim to avoid the assumption made in Section 1.3.2, and treats the initial weights and partial sums as intervals instead of values. This method then applies similar logic to the Huffman method, attempting to create partial intervals with minimal range as each node in the tree is added together. Notably, there are several situations in which more than one interval could produce 'minimal range' and so, in the interest of experimentation, two 'interval' methods were coded and tested for this project; Interval$_{LT}$ and Interval$_{GT}$, which favour the interval with lesser and greater bounds respectively.

While the creation of two separate summation methods like this seems arbitrary, looking at the differences between the summation trees does show that there can be a significant difference between the partial sums created. This will be observed in more detail later in the report, with 'real' summation examples but, even in the basic examples shown, The Interval$_{GT}$ method can be seen to pair up the two greatest (3 and 4) and smallest weights (-3 and -2) together, while the Interval$_{LT}$ method only uses the maximum and

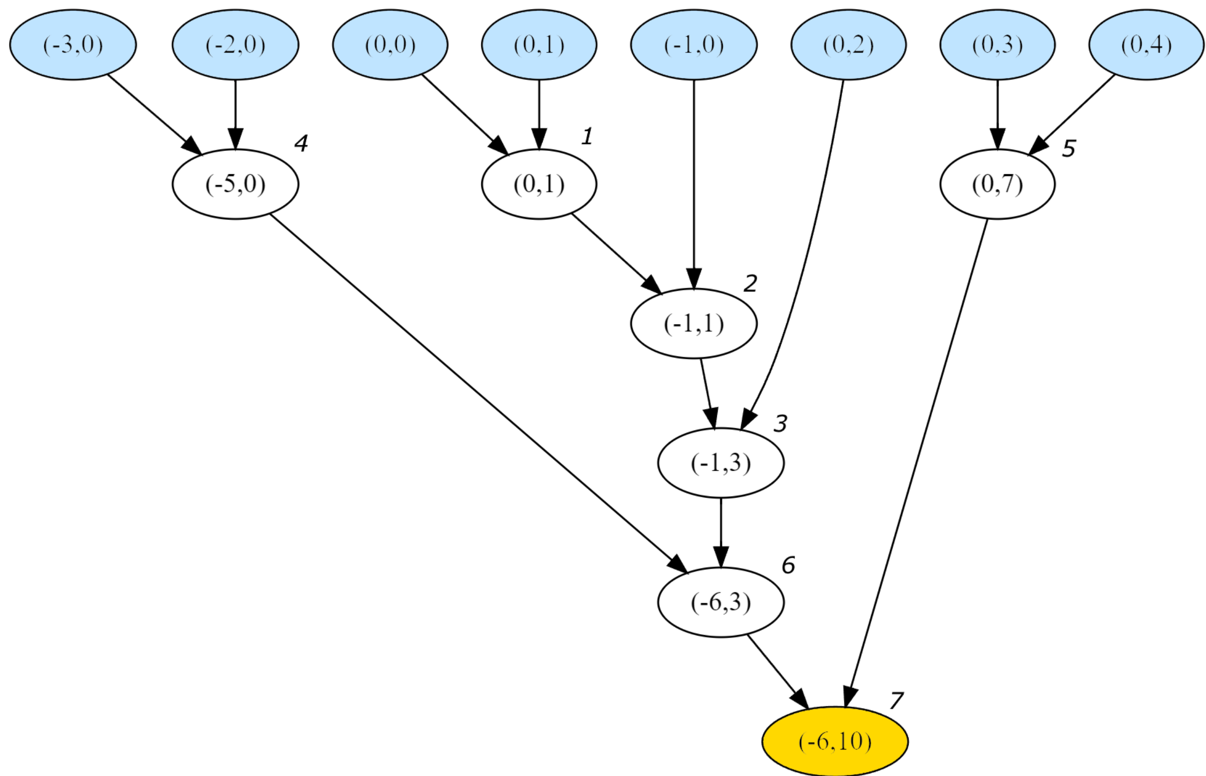

Figure 2.19: Visualisation of 'Interval$_{\mathrm{GT}}$' summation of the numbers -3-4, colorisation and labelling is the same as in Figure 2.18

minimum weights near the end of building the summation tree.

Comparing the Interval$_{\mathrm{LT}}$ and Interval$_{\mathrm{GT}}$ graphs with both the negative (Figures 2.20 and 2.21) and positive (Figures 2.22 and 2.23) weight sets (assuming all inputs are 1) shows us that Interval$_{\mathrm{LT}}$ has less depth and smaller maximum partial sums than Interval$_{\mathrm{GT}}$. The Huffman graphs for these weight sets (Figures 2.6 and 2.7) seems to be less reliable, as they share Interval$_{\mathrm{GT}}$'s high depth, but the maximum partial sums are lower than either Interval method in the negative weight set, and greater than both Interval method in the positive weight set.

Using the 'close to zero' weight set, the depths of Huffman (Figure 2.8), Interval$_{\mathrm{LT}}$ (Figure 2.24), and Interval$_{\mathrm{GT}}$ (Figure 2.25) are equal, although Huffman's maximum partial sums are notably much higher than either Interval method's. Most notably with this weight set, however, is the fact that both Interval methods produce the exact same summation tree. Since the weight set that led to this is a single, random data point, there is likely not much worthwhile information to be garnered here, but the identical graphs deserved pointing out as a possibility across both Interval methods.

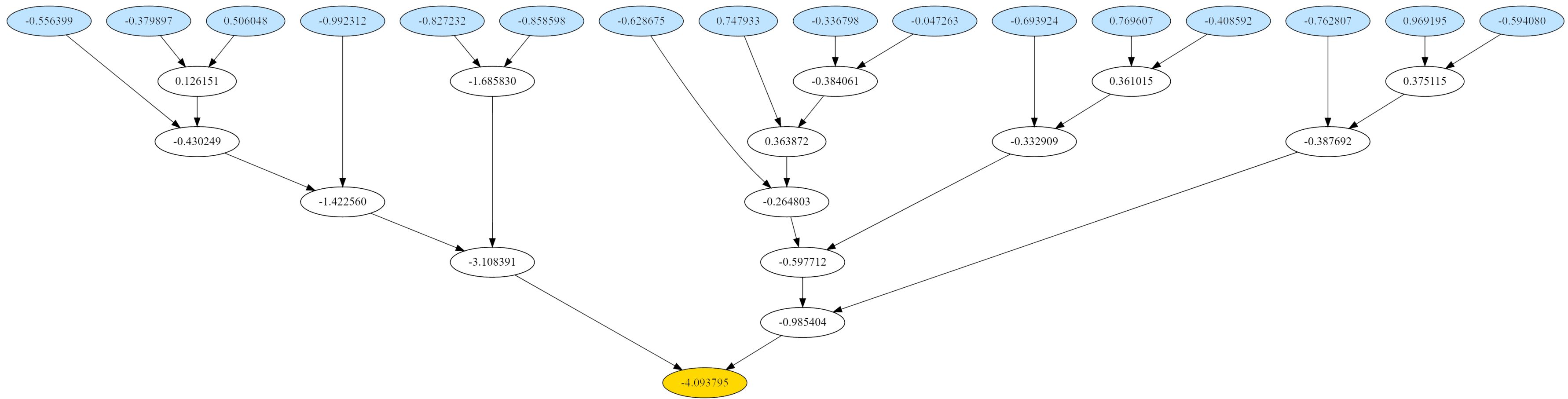

Figure 2.20: Real example of the $\text{Interval}_{\text{LT}}$ summation method, using the **'Negative'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

Draft of 6:04 pm, Wednesday, April 17, 2024

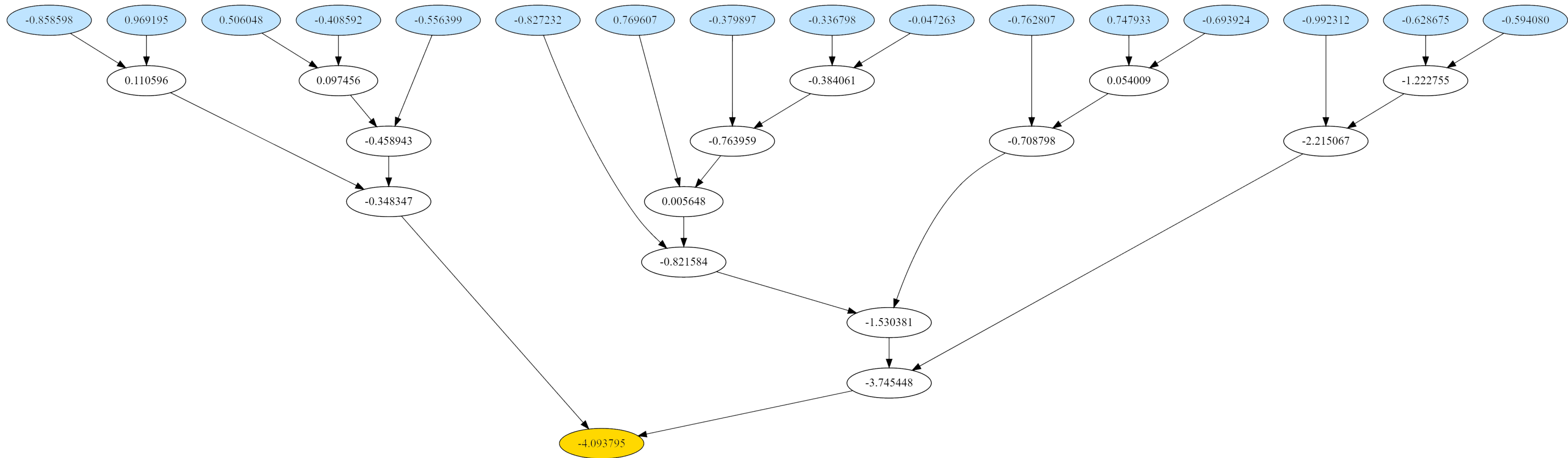

Figure 2.21: Real example of the Interval$_{\mathrm{GT}}$ summation method, using the **'Negative'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

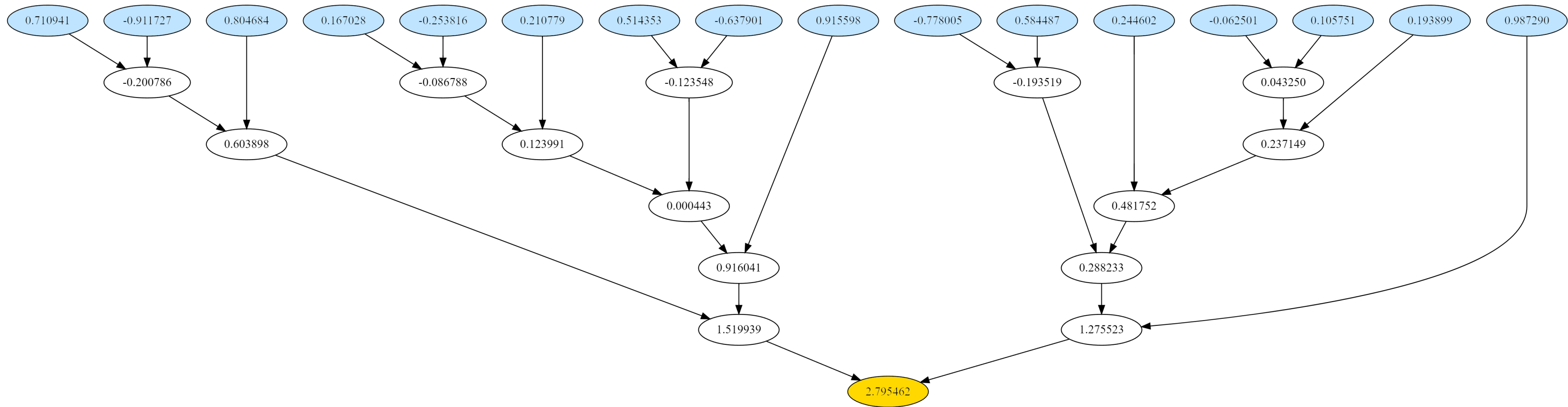

Figure 2.22: Real example of the Interval$_{\mathrm{LT}}$ summation method, using the **'Positive'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

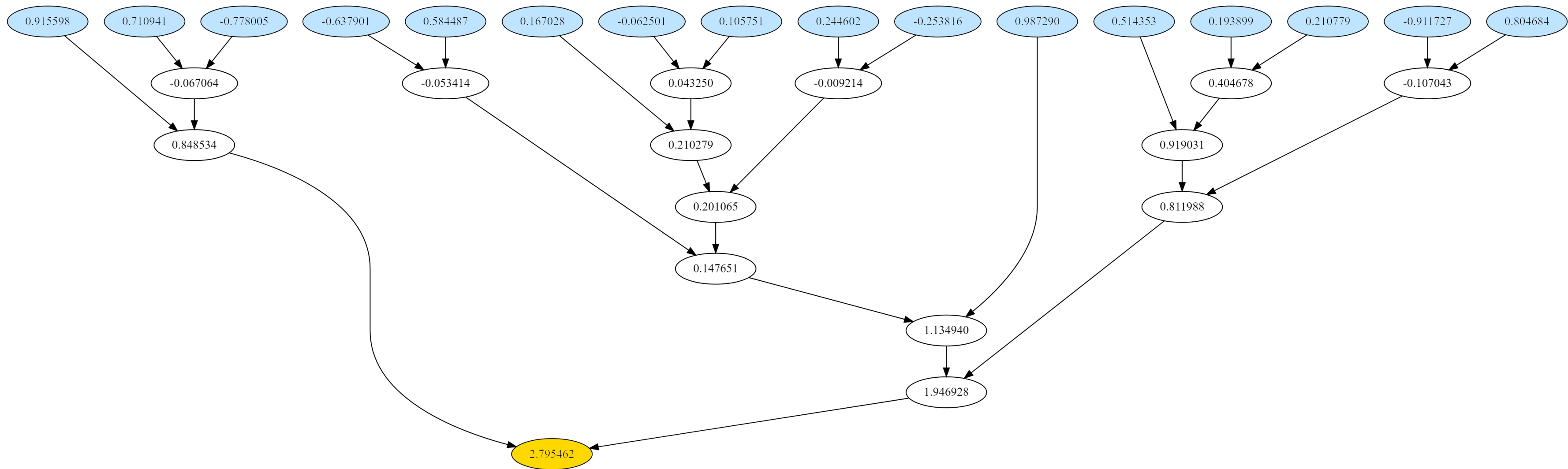

Figure 2.23: Real example of the Interval$_{\mathrm{GT}}$ summation method, using the **'Positive'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

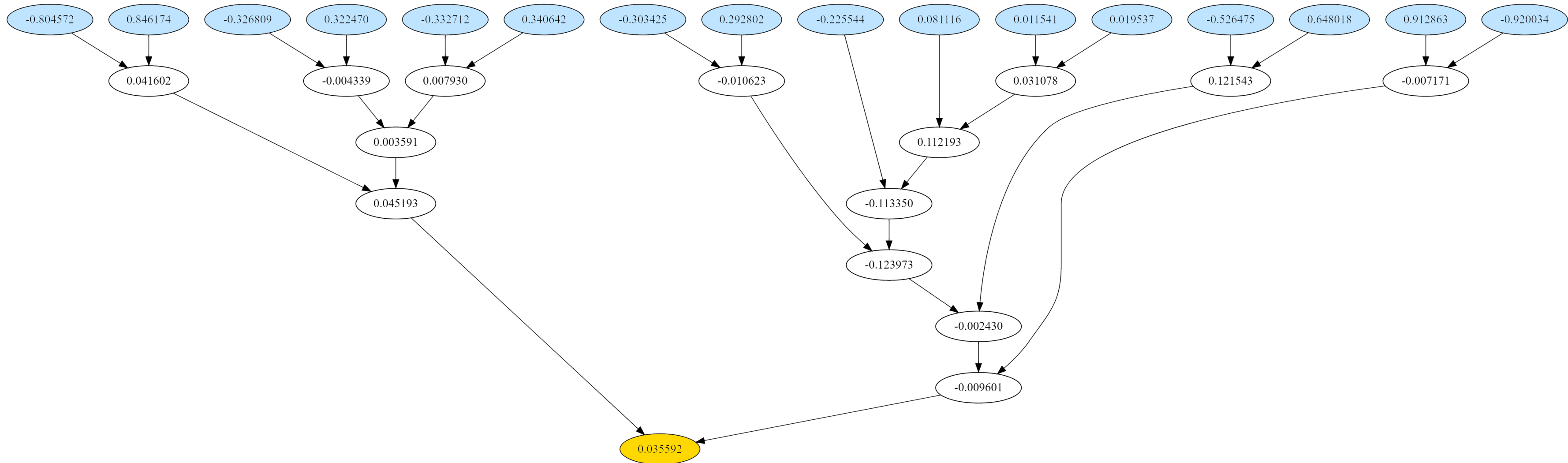

Figure 2.24: Real example of the Interval$_{\text{LT}}$ summation method, using the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

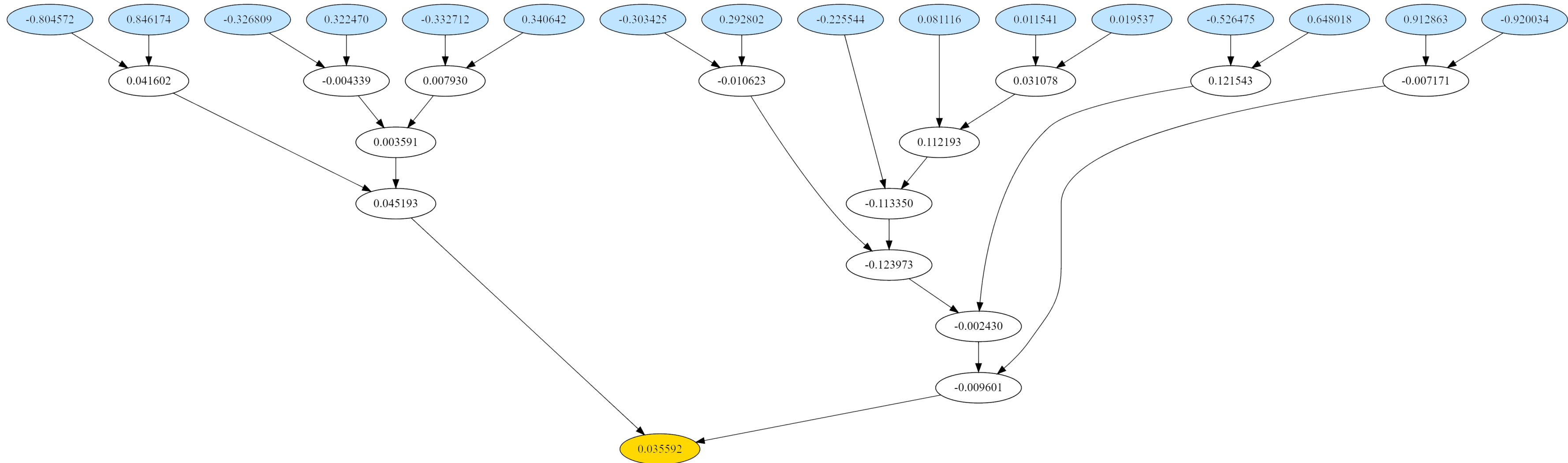

Figure 2.25: Real example of the Interval$_{\mathrm{GT}}$ summation method, using the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

2.6 Summary

I have established several well-established and newer approaches to summation problems, namely Simple Summation, Huffman Summation, Pairwise Summation (including an analysis of how different permutations of the initial weights may affect the output's FPE), and Interval Summation, where the nodes in a summation tree aren't treated as the results of partial sums, but as ranges of potential values.

When presenting these methods, I analysed how their tree depths, the largest absolute values of their partial sums, and areas where the tree's layout put the summation at risk of Catastrophic Cancellation. Graphs of summation trees were provided throughout, including 'real examples', where randomly generated floating point values were used as initial weights. The Pairwise method was shown to have the lowest depth in its trees, while the Interval methods appeared to have some of the lowest maximum absolute partial sum values.

Chapter 3

Design & Implementation

Having analysed the summation methods discussed in Chapter 2, I came up with two key ideas surrounding ways to reduce FPE based on the factors that influence said FPE. These two ideas focused on reducing maximum absolute partial sum value, and improved sorting permutations of weights, respectively.

3.1 Using the Expected Mean

3.1.1 Design

As discussed in Section 1.3.1, we know that the output of any Scaled Summation problem, regardless of if the inputs' distributions are Normal or Uniform, is Normal. More specifically, the distribution is Normal with a mean of $\frac{\sum w}{2}$, where w is the set of all weights (Lemons, 2002; Irwin, 1927; Hall, 1927). meaning, based on the initial weights and the knowledge that the inputs are within the range (0,1), we can derive the distribution of the output.

For example, using the values of the **Positive** weight set from Section 2.1, we can identify the mean as $\frac{2.795462}{2} = 1.397731$ we can find the minimum (assume all negative weights have input 1, assume all positive weights have input 0) and maximum (assume all positive weights have input 1, assume all negative weights have input 0) bounds as
$(-0.911727(1) - 0.778005(1) - 0.637901(1) - 0.253816(1) - 0.062501(1) + 0.105751(0) + 0.167028(0) + 0.193899(0) + 0.210779(0) + 0.244602(0) + 0.514353(0) + 0.584487(0) + 0.710941(0) + 0.804684(0) + 0.915598(0) + 0.987290(0)) = -2.64395$
and
$(-0.911727(0) - 0.778005(0) - 0.637901(0) - 0.253816(0) - 0.062501(0) + 0.105751(1) + 0.167028(1) + 0.193899(1) + 0.210779(1) + 0.244602(1) + 0.514353(1) + 0.584487(1) + 0.710941(1) + 0.804684(1) + 0.915598(1) + 0.987290(1)) = 5.439412$

Which allows us to create distributions in Figure 3.1. As the standard deviation of the potentially normally distributed inputs is unknown, we can only know that the output is a Normal distribution with mean $\frac{\sum w}{2}$, and an unknown standard deviation.

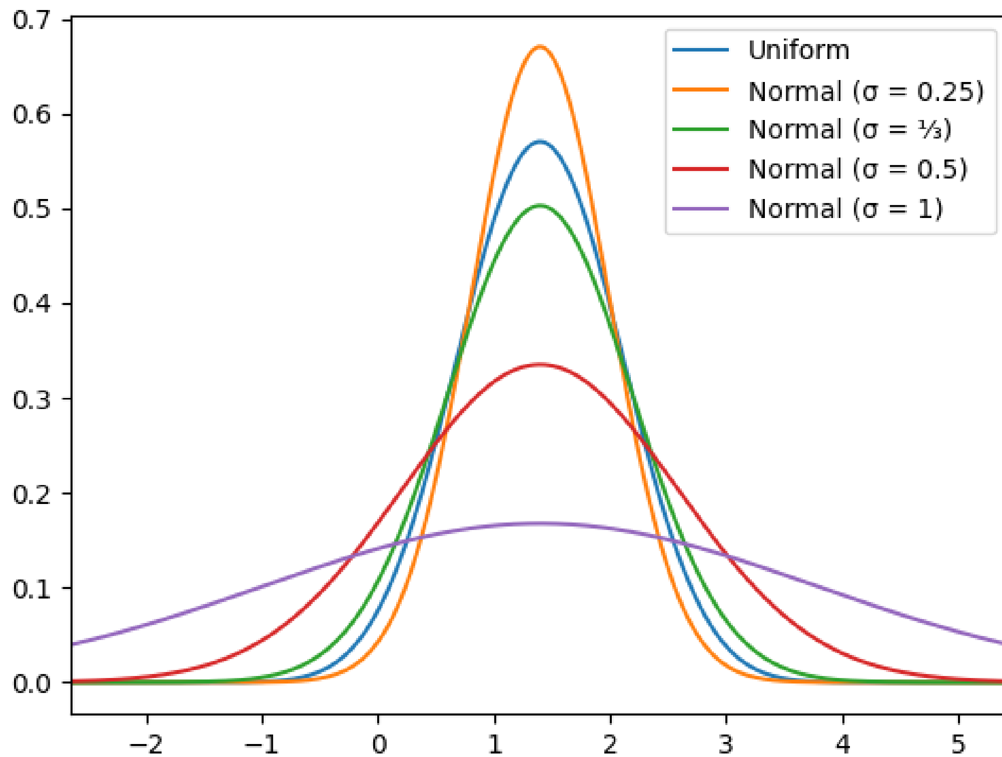

Figure 3.1: Distributions of outputs for a Scaled Summation using the **'Positive'** weights from Section 2.1. Different distributions of the initial inputs are used (as marked in the legend)

Using this known fact about the final distribution, and the knowledge that keeping the values of the partial sums in a scaled summation problem close to zero can help reduce FPE by reducing potential rounding error, I began working on a potential method adjustment that surrounded adjusting the mean of the output distribution.

3.1.2 Implementation

The basic idea of how to adjust the mean was as follows: if I can subtract the Scaled Summation's most likely output (the mean value of the output's distribution) from the summation before it begins, then every partial sum should be much closer to 0 as the mean of this 'new' Scaled Summation is 0. The original mean could then be added to the

output of the 'new' Scaled Summation in order to get the summation's 'true' output.

In practice, I now had to figure out how to 'subtract' the mean in the first place. There were two possible approaches to use here; I could subtract the weighted inputs' means from them individually, or I could treat the additive inverse mean as its own mean in an adjusted summation.

While subtracting inputs' means may seem preferable, as it is a more even-handed approach to reduce the output by the value of the expected mean, one does need to remember that the weights can be positive and negative. This means that the absolute value of everything being subtracted this way will be greater than the difference in value between the 'reduced' and 'true' outputs.

Using the **Positive** weight set as a worked example again, the absolute value of all subtracted means will be
$(0.911727(0.5)+0.778005(0.5)+0.637901(0.5)+0.253816(0.5)+0.062501(0.5)+0.105751(0.5)+ 0.167028(0.5)+0.193899(0.5)+0.210779(0.5)+0.244602(0.5)+0.514353(0.5)+0.584487(0.5)+ 0.710941(0.5) + 0.804684(0.5) + 0.915598(0.5) + 0.987290(0.5)) = 4.041681$
which is significantly larger than the actual mean of 1.397731. This disparity between subtracted value and the expected mean of the output implies that a not insignificant amount of fidelity will be lost in the 'reduced' Scaled Summation, and that this 'reduced' summation method will introduce a large amount of FPE that outweighs any potential benefit gained by reducing the partial sums.

Now it seems apparent that reducing the output by the expected mean via the introduction of the expected mean's additive inverse is the better way to go about implementing this idea. Figures 3.2 through 3.7 are examples of Summation trees for the methods covered in Chapter 2 with 17 (as opposed to 16) weights, where the 17th weight is the additive inverse of the summation's expected mean.

Analysing these Figures, the first thing a reader might note is that the outputs are not 0, but the expected mean. This is because, in implementing this mean system, I either had to build the test graphs with double the mean (to reflect the fact that every weight is left as is, not multiplied by 0.5), which would make it the absolutely largest value, leading to it not being incorporated properly into the summation trees, or I could leave the expected mean as is and build a tree that 'folds' it into the tree fluidly, allowing more partial sums to be brought closer to 0 in practice.

Direct comparisons of these Figures to the Figures that use the positive weight set without adjusting for the expected mean show generally negative changes, with only $\text{Interval}_{\text{LT}}$ (Figure 2.22 vs. Figure 3.6), $\text{Interval}_{\text{GT}}$ (Figure 2.23 vs. Figure 3.7), and "simple"-sorted pairwise (Figure 2.15 vs. Figure 3.5) showing lower maximum absolute partial sum values. Simple (Figure 2.3 vs. Figure 3.2), Huffman (Figure 2.7 vs. Figure 3.3)

and the unsorted Pairwise (Figure 2.14 vs 3.4) trees show greater maximum absolute partial sum values, and all trees have higher depths, as a result of the new weight.

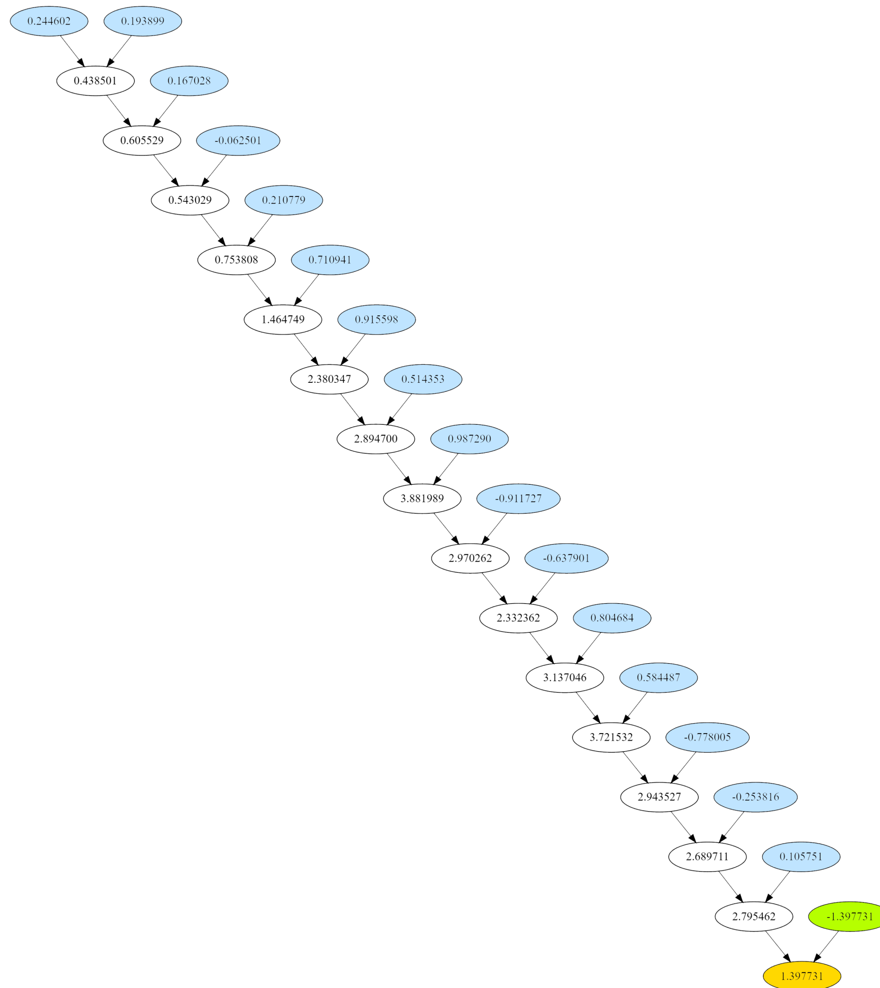

Figure 3.2: Real example of the Simple summation method, using the **'Positive'** weights from Section 2.1, including the expected mean of the output as a weight, inputs are highlighted in blue (with the expected mean weight highlighted in green) and the output is highlighted in orange.

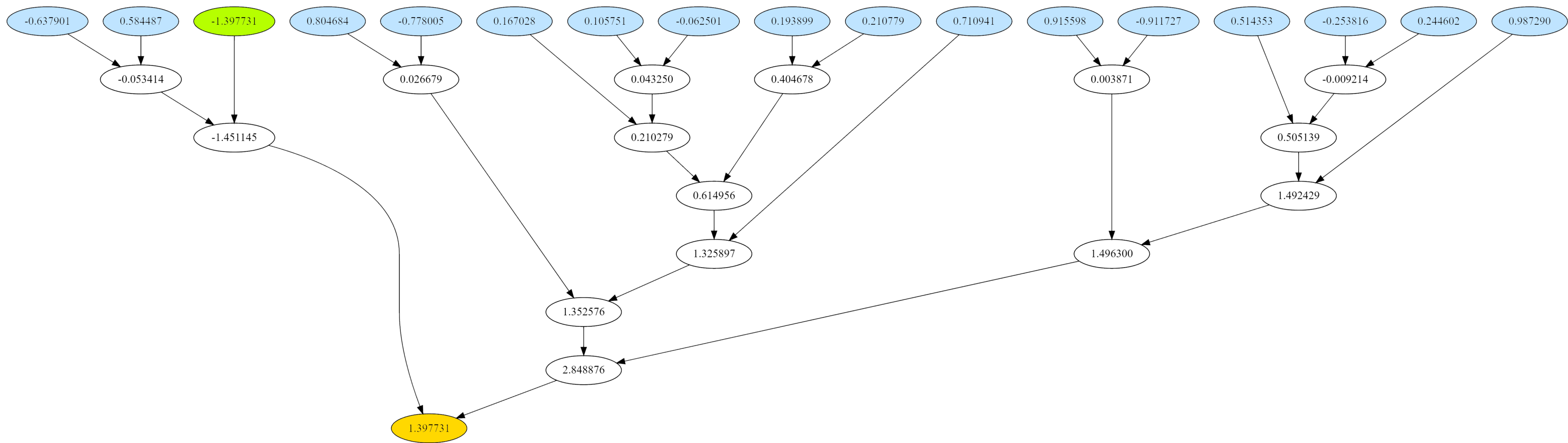

Figure 3.3: Real example of the Huffman summation method, using the **'Positive'** weights from Section 2.1, including the expected mean of the output as a weight, inputs are highlighted in blue (with the expected mean weight highlighted in green) and the output is highlighted in orange.

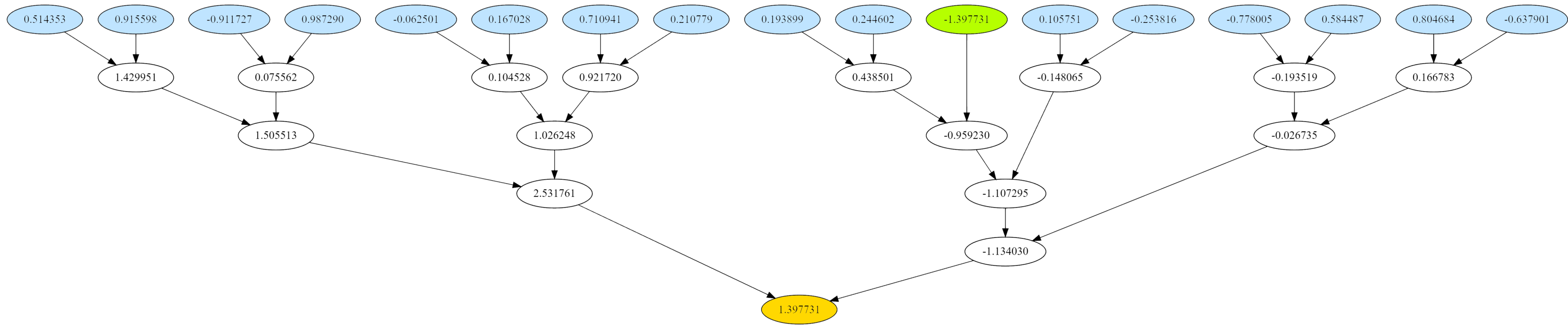

Figure 3.4: Real example of the Pairwise summation method, using the default permutation of the **'Positive'** weights from Section 2.1, including the expected mean of the output as a weight, inputs are highlighted in blue (with the expected mean weight highlighted in green) and the output is highlighted in orange.

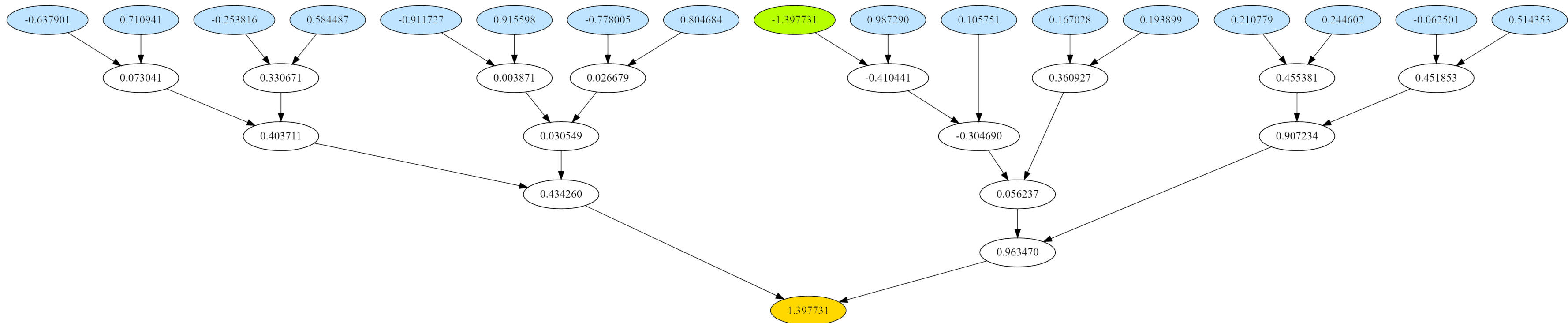

Figure 3.5: Real example of the Pairwise summation method, using the "simple"-sorted permutation of the **'Positive'** weights from Section 2.1, including the expected mean of the output as a weight, inputs are highlighted in blue (with the expected mean weight highlighted in green) and the output is highlighted in orange.

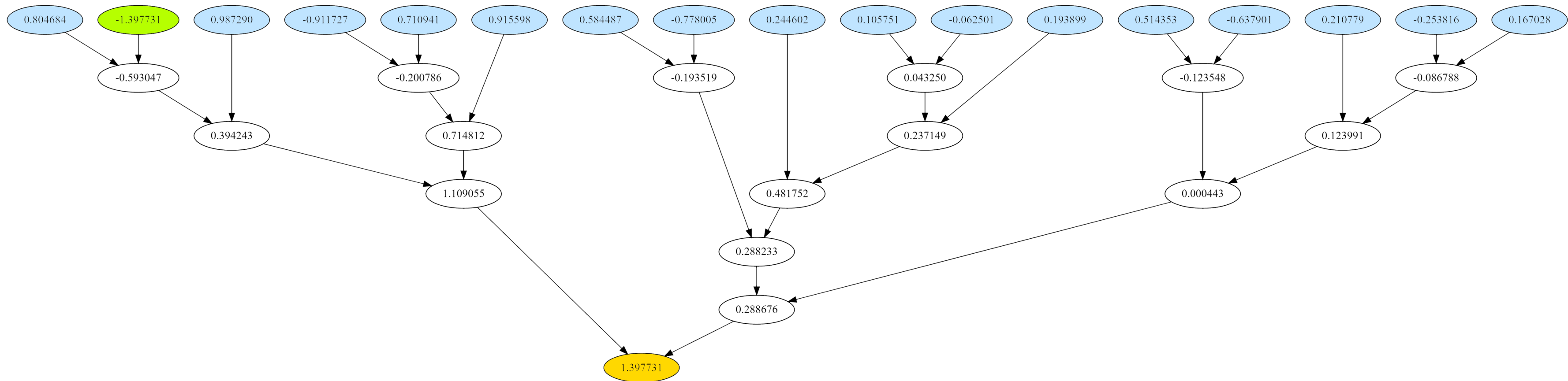

Figure 3.6: Real example of the $\text{Interval}_{\text{LT}}$ summation method, using the **'Positive'** weights from Section 2.1, including the expected mean of the output as a weight, inputs are highlighted in blue (with the expected mean weight highlighted in green) and the output is highlighted in orange.

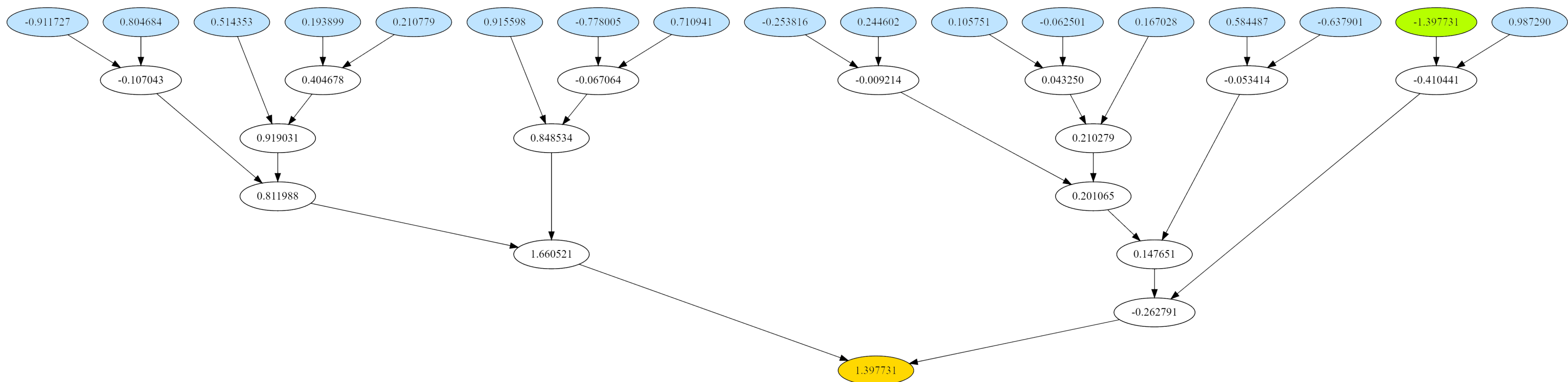

Figure 3.7: Real example of the $\text{Interval}_{\text{GT}}$ summation method, using the **'Positive'** weights from Section 2.1, including the expected mean of the output as a weight, inputs are highlighted in blue (with the expected mean weight highlighted in green) and the output is highlighted in orange.

3.2 "Hyper"-Sorted Pairwise

3.2.1 Design

While examining the pairwise summation method (As described in Section 2.4), I tried to consider any ways to potentially improve it, possibly by employing a more involved sorting method. I was reminded of the concept Gaussian sums, whereby the sum of a series of n equidistant numbers $(x_1, x_2, ...x_n)$ is equal to $\frac{(x_1+x_n)n}{2}$ $(+x_{\frac{n}{2}}$ if n is odd) (Bernt et al., 1998).

Of course, the nature of this problem does not lead itself towards equidistant values, but I was determined to examine how a similar pairing system $((x_1+x_n), (x_2+x_{n-1}), (x_3+x_{n-2})$, etc.) could effect our Scaled Summation problem if rigorously applied to every 'layer' of a pairwise summation tree. Throughout the rest of this report, I will refer to this method as "Hyper"-Sorting.

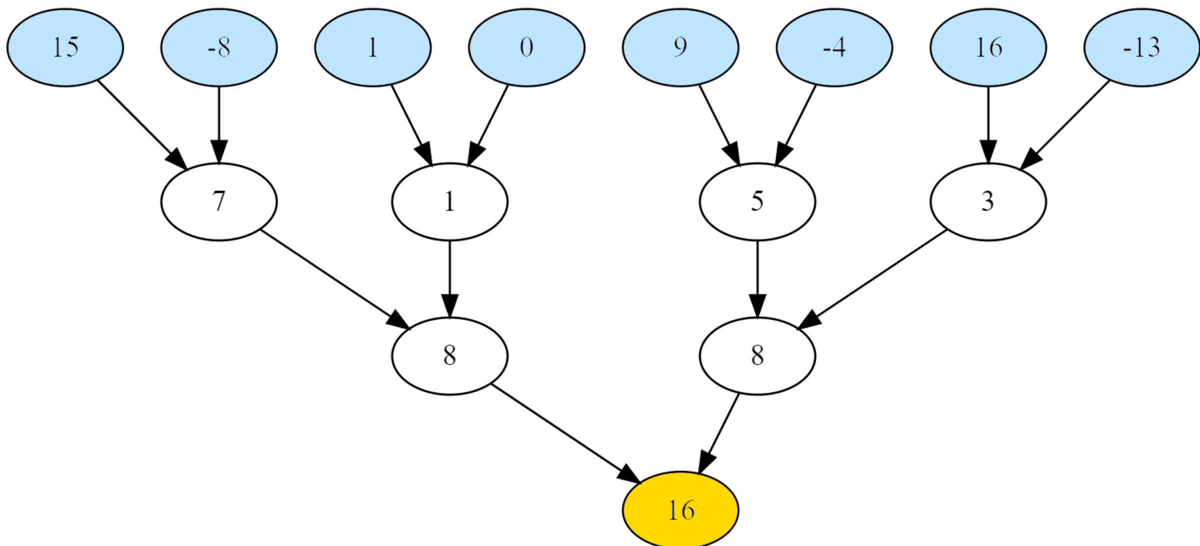

Figure 3.8: Visualisation of 'pairwise' summation of the weights $[-13, -8, -4, 0, 1, 9, 15, 16]$ sorted such that the high and low values in every layer are paired together, inputs are highlighted in blue and the output (16) is highlighted in orange.

3.2.2 Implementation

In an effort to examine how this system could affect Scaled Summation accuracy, I began by manually creating summation trees similar to 3.8 and noticed that, in each layer, all the partial sum values approach $\frac{r}{n}$, where $r =$ the overall result, and $n =$ the number of partial sums in that layer. In order to understand why this quirk is very beneficial for

reducing FPE, we need to consider the smallest possible absolute maximum partial sum in a summation tree.

Since we know the output o is the absolute largest value that any summation tree must sill have, the values of the partial sums that create o, p_1 and p_2, must be x and $o-x$. In order to minimise x without allowing $((o-x) > x)$, x must equal $o-x$, meaning $x = \frac{o}{2}$. Similarly, if we want to minimise the absolute values of p_{1_1} and p_{1_2}, they must both equal $\frac{p_1}{2}$, and so on for every p_x.

The "Hyper"-Sorted Pairwise tree achieves this partial sum minimisation as well as it can with the initial weights provided, marking it as seemingly the ideal summation method in terms of partial sum reduction, assuming that the assumptions made in this report that the inputs are all normally distributed about a mean of 0.5, and that the relative sizes of the initial weights $(w_1...w_n)$ and the weighted inputs $(w_1i_1...w_ni_n)$ are roughly the same. In terms of summation tree depth, this is a pairwise summation method, and so it ties for lowest possible summation tree depth with the other pairwise summation methods discussed in Section 2.4.

Figures 3.9, 3.10, and 3.11 on the following pages show this "hyper"-sorted pairwise summation method applied to the 'Positive', 'Negative', and 'Close to Zero' weight sets from Section 2.1. It is important to note that there is no use of the expected mean with this "hyper"-sorted pairwise method, as the functions written for the method only work for sets of 2^x weights, where $x \in N^+$. Analysing these Figures, one can see that the "hyper"-sorted pairwise summation method consistently has the lowest maximum partial sums. It is worth noting that, with the 'close to zero' dataset, the values in the second 'layer' of the "hyper"-sorted graph do not all hold the same sign and are still relatively different to each other in the deeper layers of the tree, this is due to the large relative differences between the initial weight values and it is important to note that the values still approach $\frac{r}{n}$, where $r =$ the overall result, and $n =$ the number of partial sums in that layer.

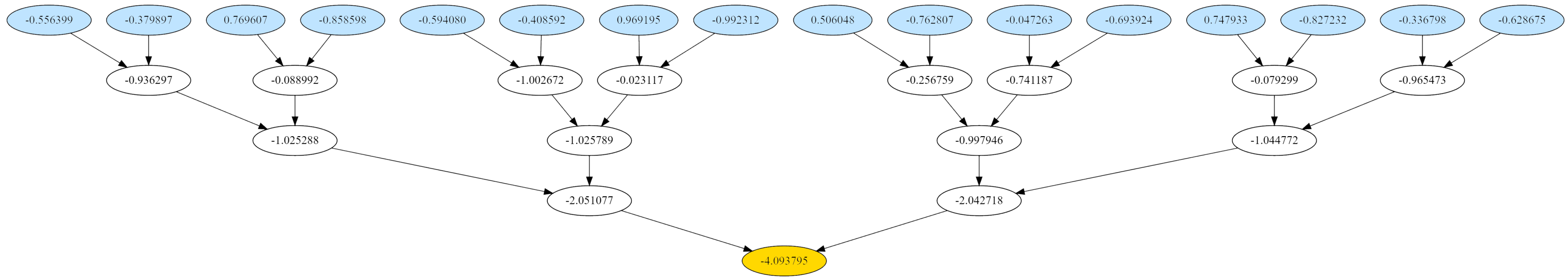

Figure 3.9: Real example of the Pairwise summation method, using the ”hyper”-sorted permutation of the **’Negative’** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

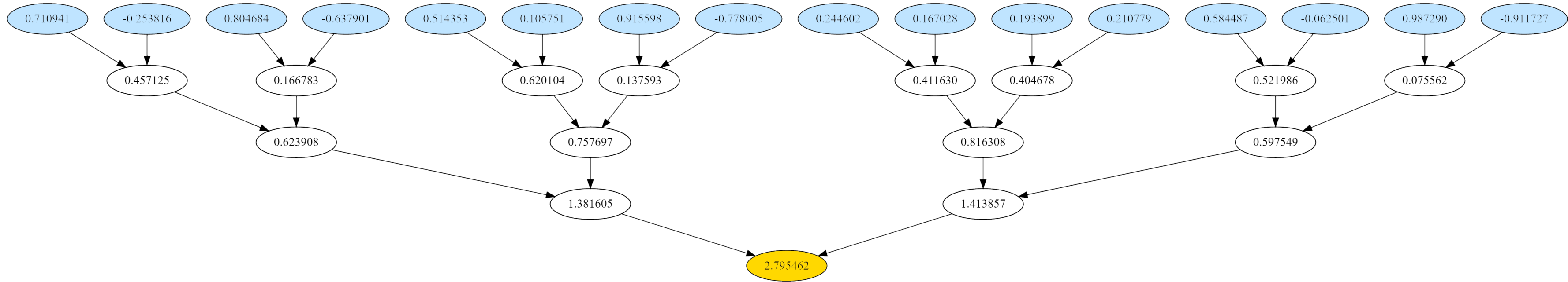

Figure 3.10: Real example of the Pairwise summation method, using the "hyper"-sorted permutation of the **'Positive'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

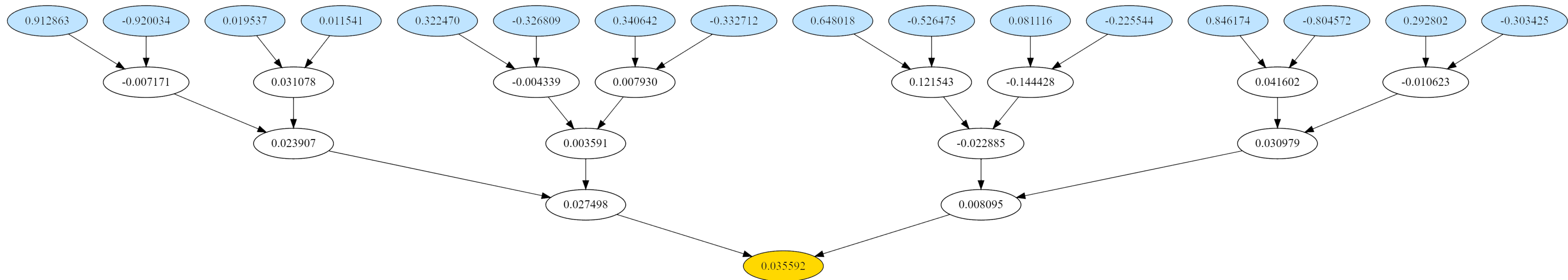

Figure 3.11: Real example of the Pairwise summation method, using the "hyper"-sorted permutation of the **'Close to Zero'** weights from Section 2.1, inputs are highlighted in blue and the output is highlighted in orange.

3.3 Summary

In this chapter, I discussed two major areas I explored in order to reduce FPE in Scaled Summation problems, Expected Mean and "Hyper"-Sorted Pairwise.

Regarding Expected Mean, I discussed how using the predictable expected mean of the summation could be used to reduce the values of all partial sums in the summation, and I then analysed how this adjustment performed when applied to the previously discussed Summation Methods.

Regarding "Hyper"-Sorted Pairwise, I described my design of the concept, where I found that sorting the weights such that each 'layer' of the pairwise summation tree paired the highest value with the lowest value, the second-highest value with the second-lowest value, and so on, may be the best method in terms of reducing absolute maximum partial sum value. 'Real example' graphs were also generated for this method, as was done in the previous chapter.

Chapter 4

Evaluation

The way to evaluate the performance of all methods discussed, as well as the new tactics discussed in Chapter 3, is quite simple in theory, although the randomness involved does make general evaluation difficult.

4.1 Experiments

The testing worked as follows:

m different 'weight sets' of size 16 would be generated from a uniform distribution with range (-1,1).

For each of these 'weight sets', n of 'input sets' of size 16 would be generated from a normal distribution with $\mu = 0.5$ and $\sigma = \frac{1}{6}$.

For each weight set, summation trees would be created according to each method, and tested against the 'input sets', with results collated.

After iterating through each 'input set' for each 'weight set', all results would then be collated.

The C++ *std::normal_distribution* and *std::uniform_real_distribution* functions were used for distribution generation. (C++Reference, 2023, 2024)

The μ was chosen to be 0.5, and the σ was chosen to be $\frac{1}{6}$ as 99.73% of generated values lie within $\pm 3\sigma$ of μ (OEIS Sequence A270712, 2024), meaning those values will be in the range (0,1). Any values generated outside that range are 'reflected' back to maintain the range (i.e. if a value x is generated such that $x > 1$, the 'produced' value will be $1 - x$).

Due to the limitations of my own personal machine, upon which these tests were run, $m = 500$, and $n = 10,000$, although the test was run 10 times using a normal distribution for the input sets (Tables 4.1 through 4.10; Tables 4.11 and 4.12 show accumulated results), and an additional 10 times using a uniform distribution for the input sets (Tables 4.13 through 4.22; Table 4.23 shows accumulated results). The results will be presented for each 'batch' where $m = 500$, with a additional 'accumulated batch' results for $m = 5,000$. Accumulated results for a combination of Batch 1 and Batch 2 are seen in Tables 4.24 and 4.25.

Results are measured in three metrics; Absolute Error (E_{Abs}), Relative (Output) Error (E_{Rel_O}), and Relative (Absolute) Error (E_{Rel_A}). These metrics are calculated as follows:

Ans_{True} (Output of a 'Simple' summation of the sum, using *Long Double* representations of the weighted inputs instead of *Float* representations).

Ans_{Found} (Output of the tested summation tree).

Ans_{Abs} (Output of a 'Simple' summation of the absolute values of *Long Double* representations of the weighted inputs).

$$E_{Abs} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} (Ans_{True} - Ans_{Found})_{weight_x;input_y}}{nm}$$

$$E_{Rel_O} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} \frac{(Ans_{True} - Ans_{Found})_{weight_x;input_y}}{Ans_{True}}}{nm}$$

$$E_{Rel_A} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} \frac{(Ans_{True} - Ans_{Found})_{weight_x;input_y}}{Ans_{Abs}}}{nm}$$

Each Table also notates the minimum possible value for E_{Abs}, which is calculated as the sum of the differences between the *Long Double* and *Float* representations of Ans_{True}, divided by nm, providing the value of E_{Abs} if the only FPE was Rounding Error at the output.

4.1.1 Batch 1 (Normal Distribution, $\mu = 0.5, \sigma = \frac{1}{6}$)

2.249345e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.409261e-8	3.295536e-7	1.365433e-8
Huffman (w/ Mean)	5.905140e-8	2.108810e-7	1.528294e-8
Interval$_{\mathbf{LT}}$	3.520181e-8	6.347914e-8	8.759012e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.751121e-8	8.927169e-8	9.673598e-9
Interval$_{\mathbf{GT}}$	3.549606e-8	7.609370e-8	8.913731e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.707240e-8	1.099127e-7	9.635634e-9
Pairwise (Unsorted)	4.997511e-8	2.000069e-7	1.239349e-8
Pairwise (Unsorted; Mean)	6.105339e-8	3.377505e-7	1.544651e-8
Pairwise (”Simple”-Sorted)	4.055107e-8	7.521054e-8	1.031649e-8
Pairwise (”Simple” Sorted; Mean)	3.989174e-8	8.005150e-8	1.012383e-8
Pairwise (”Hyper”-Sorted)	3.736997e-8	6.081339e-8	9.509740e-9
Simple	6.772147e-8	2.656558e-7	1.712278e-8
Simple (w/ Mean)	6.919548e-8	2.859874e-7	1.758247e-8

Table 4.1: Table showing the results table for Batch 1_1, the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\text{LT}}$. Best Relative (Output) Error: Pairwise (”Hyper”-Sorted). Best Relative (Absolute) Error: Interval$_{\text{LT}}$

2.204402e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.692221e-8	1.203497e-7	1.198135e-8
Huffman (w/ Mean)	5.355429e-8	2.464934e-7	1.386145e-8
Interval$_{\mathbf{LT}}$	3.419582e-8	7.086863e-8	8.742886e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.042118e-8	9.534927e-8	1.046794e-8
Interval$_{\mathbf{GT}}$	3.225722e-8	7.075500e-8	8.170248e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.707949e-8	7.207672e-8	9.549196e-9
Pairwise (Unsorted)	5.110779e-8	1.945716e-7	1.308351e-8
Pairwise (Unsorted; Mean)	5.535999e-8	2.079956e-7	1.408271e-8
Pairwise (”Simple”-Sorted)	3.586866e-8	7.855952e-8	9.198639e-9
Pairwise (”Simple” Sorted; Mean)	4.553561e-8	7.874653e-8	1.163705e-8
Pairwise (”Hyper”-Sorted)	3.620932e-8	6.523812e-8	9.318236e-9
Simple	6.739224e-8	1.558018e-7	1.729348e-8
Simple (w/ Mean)	6.915114e-8	1.654431e-7	1.776907e-8

Table 4.2: Table showing the results table for Batch 1_2 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\text{GT}}$. Best Relative (Output) Error: Pairwise (”Hyper”-Sorted). Best Relative (Absolute) Error: Interval$_{\text{GT}}$.

2.307638e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.918035e-8	1.086093e-7	1.248398e-8
Huffman (w/ Mean)	4.805286e-8	9.613027e-8	1.214340e-8
Interval$_{\mathbf{LT}}$	3.511208e-8	4.381183e-8	8.876155e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.982502e-8	5.706683e-8	1.007316e-8
Interval$_{\mathbf{GT}}$	3.339759e-8	4.208800e-8	8.463623e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.849582e-8	5.118478e-8	9.752421e-9
Pairwise (Unsorted)	4.717506e-8	9.324152e-8	1.209029e-8
Pairwise (Unsorted; Mean)	5.900274e-8	1.062411e-7	1.495346e-8
Pairwise (”Simple”-Sorted)	4.016616e-8	5.177071e-8	1.032063e-8
Pairwise (”Simple” Sorted; Mean)	4.137474e-8	5.855352e-8	1.053798e-8
Pairwise (”Hyper”-Sorted)	3.572809e-8	4.117078e-8	9.040280e-9
Simple	7.237737e-8	1.501058e-7	1.861875e-8
Simple (w/ Mean)	7.366731e-8	1.526109e-7	1.887841e-8

Table 4.3: Table showing the results table for Batch 1_3 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\text{GT}}$. Best Relative (Output) Error: Pairwise (”Hyper”-Sorted). Best Relative (Absolute) Error: Interval$_{\text{GT}}$.

1.825732e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.669606e-8	1.013604e-7	1.204043e-8
Huffman (w/ Mean)	5.400047e-8	1.297195e-7	1.395192e-8
Interval$_{\mathbf{LT}}$	3.049922e-8	4.888818e-8	7.890898e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.671443e-8	6.519742e-8	9.668664e-9
Interval$_{\mathbf{GT}}$	3.144746e-8	5.189259e-8	7.914329e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.646536e-8	7.834338e-8	9.420208e-9
Pairwise (Unsorted)	4.757960e-8	1.156512e-7	1.221907e-8
Pairwise (Unsorted; Mean)	5.181571e-8	1.143203e-7	1.357638e-8
Pairwise (”Simple”-Sorted)	2.966724e-8	5.910765e-8	7.634214e-9
Pairwise (”Simple” Sorted; Mean)	3.950283e-8	7.254150e-8	1.017523e-8
Pairwise (”Hyper”-Sorted)	3.150007e-8	5.245807e-8	8.120627e-9
Simple	5.985596e-8	1.323142e-7	1.526789e-8
Simple (w/ Mean)	6.046378e-8	1.390130e-7	1.531055e-8

Table 4.4: Table showing the results table for Batch 1_4 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise (”Simple”-Sorted). Best Relative (Output) Error: Interval$_{\text{LT}}$. Best Relative (Absolute) Error: Pairwise (”Simple”-Sorted).

1.962339e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.009451e-8	1.970144e-7	1.244933e-8
Huffman (w/ Mean)	5.202579e-8	1.988334e-7	1.319346e-8
Interval$_{\mathbf{LT}}$	3.016938e-8	1.065644e-7	7.472399e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.536815e-8	1.436729e-7	8.976427e-9
Interval$_{\mathbf{GT}}$	3.069416e-8	9.223564e-8	7.780822e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.639092e-8	1.477947e-7	9.271558e-9
Pairwise (Unsorted)	5.219597e-8	4.131743e-7	1.339673e-8
Pairwise (Unsorted; Mean)	4.910780e-8	3.075329e-7	1.246150e-8
Pairwise ("Simple"-Sorted)	3.328850e-8	9.458991e-8	8.494787e-9
Pairwise ("Simple" Sorted; Mean)	3.801160e-8	1.298526e-7	9.723266e-9
Pairwise ("Hyper"-Sorted)	2.870339e-8	7.731410e-8	7.138636e-9
Simple	5.643341e-8	2.275333e-7	1.444949e-8
Simple (w/ Mean)	5.763484e-8	2.302732e-7	1.476136e-8

Table 4.5: Table showing the results table for Batch 1_5 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise ("Hyper"-Sorted). Best Relative (Output) Error: Pairwise ("Hyper"-Sorted). Best Relative (Absolute) Error: Pairwise ("Hyper"-Sorted).

1.925276e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.334840e-8	9.497444e-8	1.126118e-8
Huffman (w/ Mean)	4.952355e-8	2.642950e-7	1.259113e-8
Interval$_{\mathbf{LT}}$	3.113697e-8	5.153977e-8	8.170761e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.672628e-8	8.468611e-8	9.670448e-9
Interval$_{\mathbf{GT}}$	2.906558e-8	5.070416e-8	7.641851e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.657611e-8	6.880395e-8	9.594240e-9
Pairwise (Unsorted)	4.644328e-8	2.189701e-7	1.186397e-8
Pairwise (Unsorted; Mean)	5.378048e-8	1.385613e-7	1.412684e-8
Pairwise ("Simple"-Sorted)	3.048941e-8	5.460829e-8	7.925432e-9
Pairwise ("Simple" Sorted; Mean)	4.058294e-8	9.660827e-8	1.069856e-8
Pairwise ("Hyper"-Sorted)	2.973561e-8	5.683752e-8	7.797939e-9
Simple	6.156445e-8	2.022647e-7	1.583173e-8
Simple (w/ Mean)	6.480105e-8	2.043716e-7	1.664740e-8

Table 4.6: Table showing the results table for Batch 1_6 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{GT}}$. Best Relative (Output) Error: Interval$_{\mathrm{GT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{GT}}$.

2.189654e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.730817e-8	2.868465e-6	1.202645e-8
Huffman (w/ Mean)	5.812302e-8	1.513822e-6	1.506597e-8
Interval$_{\mathbf{LT}}$	3.297371e-8	2.834622e-6	8.512770e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.085917e-8	1.327260e-6	1.058816e-8
Interval$_{\mathbf{GT}}$	3.560853e-8	2.837534e-6	9.073737e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.938882e-8	1.335583e-6	1.025537e-8
Pairwise (Unsorted)	4.626072e-8	2.860015e-6	1.154062e-8
Pairwise (Unsorted; Mean)	5.660058e-8	5.344703e-6	1.460692e-8
Pairwise ("Simple"-Sorted)	3.573854e-8	1.262033e-6	9.084437e-9
Pairwise ("Simple" Sorted; Mean)	4.215944e-8	2.035169e-6	1.096337e-8
Pairwise ("Hyper"-Sorted)	3.299626e-8	1.381354e-6	8.486502e-9
Simple	7.360036e-8	6.594341e-6	1.892073e-8
Simple (w/ Mean)	7.495017e-8	6.651956e-6	1.932335e-8

Table 4.7: Table showing the results table for Batch 1_7 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Pairwise ("Simple"-Sorted). Best Relative (Absolute) Error: Pairwise ("Hyper"-Sorted).

1.977565	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.756835e-8	1.476566e-7	1.242648e-8
Huffman (w/ Mean)	5.252529e-8	1.420260e-7	1.336123e-8
Interval$_{\mathbf{LT}}$	2.725958e-8	5.033720e-8	6.933760e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.757630e-8	7.595508e-8	9.626054e-9
Interval$_{\mathbf{GT}}$	2.917087e-8	5.673110e-8	7.326854e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.691950e-8	7.735656e-8	9.351581e-9
Pairwise (Unsorted)	4.847831e-8	1.271641e-7	1.240603e-8
Pairwise (Unsorted; Mean)	4.806350e-8	1.275622e-7	1.222107e-8
Pairwise ("Simple"-Sorted)	3.216117e-8	5.095291e-8	8.301404e-9
Pairwise ("Simple" Sorted; Mean)	4.771760e-8	7.678476e-8	1.211450e-8
Pairwise ("Hyper"-Sorted)	3.118773e-8	5.512532e-8	7.993578e-9
Simple	7.231582e-8	1.653432e-7	1.829186e-8
Simple (w/ Mean)	7.409693e-8	1.633328e-7	1.881856e-8

Table 4.8: Table showing the results table for Batch 1_8 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Interval$_{\mathrm{LT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

1.904174e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.491766e-8	1.869697e-7	1.158478e-8
Huffman (w/ Mean)	5.105752e-8	6.583497e-7	1.335508e-8
$\text{Interval}_{\text{LT}}$	3.169437e-8	1.088651e-7	8.256049e-9
$\text{Interval}_{\text{LT}}$ (w/ Mean)	3.695368e-8	8.344419e-8	1.955777e-8
$\text{Interval}_{\text{GT}}$	3.322718e-8	1.101896e-7	8.652884e-9
$\text{Interval}_{\text{GT}}$ (w/ Mean)	3.393069e-8	8.185780e-8	8.865112e-9
Pairwise (Unsorted)	5.155568e-8	4.545142e-7	1.308052e-8
Pairwise (Unsorted; Mean)	6.018110e-8	1.769006e-7	1.601469e-8
Pairwise ("Simple"-Sorted)	3.196488e-8	1.263966e-7	8.330024e-9
Pairwise ("Simple" Sorted; Mean)	3.703338e-8	8.271749e-8	9.764800e-9
Pairwise ("Hyper"-Sorted)	3.148704e-8	1.909493e-7	8.155085e-9
Simple	6.698558e-8	1.967685e-7	1.725405e-8
Simple (w/ Mean)	6.904174e-8	2.024269e-7	1.787794e-8

Table 4.9: Table showing the results table for Batch 1_9 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise ("Hyper"-Sorted). Best Relative (Output) Error: $\text{Interval}_{\text{LT}}$. Best Relative (Absolute) Error: Pairwise ("Hyper"-Sorted).

1.914768e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.009203e-8	2.218862e-7	1.021340e-8
Huffman (w/ Mean)	5.260051e-8	2.179721e-7	1.326955e-8
$\text{Interval}_{\text{LT}}$	3.005516e-8	9.686224e-8	7.576803e-9
$\text{Interval}_{\text{LT}}$ (w/ Mean)	3.841408e-8	1.357043e-7	9.643548e-9
$\text{Interval}_{\text{GT}}$	3.131326e-8	7.178765e-8	7.991699e-9
$\text{Interval}_{\text{GT}}$ (w/ Mean)	3.936418e-8	8.844185e-8	1.005341e-8
Pairwise (Unsorted)	4.694815e-8	1.932262e-7	1.205037e-8
Pairwise (Unsorted; Mean)	5.504286e-8	1.844746e-7	1.398064e-8
Pairwise ("Simple"-Sorted)	3.216328e-8	8.606718e-8	8.089677e-9
Pairwise ("Simple" Sorted; Mean)	4.178372e-8	8.636799e-8	1.065593e-8
Pairwise ("Hyper"-Sorted)	3.000279e-8	7.970068e-8	7.589770e-9
Simple	6.781868e-8	1.990101e-7	1.690477e-8
Simple (w/ Mean)	6.915006e-8	2.116480e-7	1.722443e-8

Table 4.10: Table showing the results table for Batch 1_{10} the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise ("Hyper"-Sorted). Best Relative (Output) Error: $\text{Interval}_{\text{GT}}$. Best Relative (Absolute) Error: $\text{Interval}_{\text{LT}}$.

2.046089e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.702204e-8	4.376839e-7	1.201217e-8
Huffman (w/ Mean)	5.299199e-8	3.678522e-7	1.360760e-8
Interval$_{\mathbf{LT}}$	3.182981e-8	3.475838e-7	8.119149e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.803695e-8	2.157608e-7	1.079458e-8
Interval$_{\mathbf{GT}}$	3.216779e-8	3.460011e-7	8.192978e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.716833e-8	2.111355e-7	9.574873e-9
Pairwise (Unsorted)	4.877200e-8	4.863473e-7	1.241246e-8
Pairwise (Unsorted; Mean)	5.500082e-8	7.046042e-7	1.414707e-8
Pairwise (”Simple”-Sorted)	3.421058e-8	1.938133e-7	8.769573e-9
Pairwise (”Simple” Sorted; Mean)	4.135937e-8	2.797393e-7	1.063945e-8
Pairwise (”Hyper”-Sorted)	3.249203e-8	2.060962e-7	8.315039e-9
Simple	6.659660e-8	8.259138e-7	1.699555e-8
Simple (w/ Mean)	6.821525e-8	8.407063e-7	1.741935e-8

Table 4.11: Table showing the accumulated results table for Batch 1, the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\text{LT}}$. Best Relative (Output) Error: Pairwise (”Hyper”-Sorted). Best Relative (Absolute) Error: Interval$_{\text{LT}}$.

2.030138e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.699024e-8	1.675971e-7	1.201058e-8
Huffman (w/ Mean)	5.242188e-8	2.405223e-7	1.344556e-8
Interval$_{\mathbf{LT}}$	3.170271e-8	7.124628e-8	8.075414e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.772337e-8	9.226087e-8	1.081751e-8
Interval$_{\mathbf{GT}}$	3.178549e-8	6.916416e-8	8.095116e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.692161e-8	8.619694e-8	9.499262e-9
Pairwise (Unsorted)	4.905103e-8	2.226064e-7	1.250933e-8
Pairwise (Unsorted; Mean)	5.482306e-8	1.890377e-7	1.409598e-8
Pairwise (”Simple”-Sorted)	3.403560e-8	7.531148e-8	8.734589e-9
Pairwise (”Simple” Sorted; Mean)	4.127047e-8	8.469157e-8	1.060346e-8
Pairwise (”Hyper”-Sorted)	3.243600e-8	7.551203e-8	8.295988e-9
Simple	6.581841e-8	1.883108e-7	1.678164e-8
Simple (w/ Mean)	6.746693e-8	1.950119e-7	1.720779e-8

Table 4.12: Table showing the accumulated results table for Batch 1, without the contents of Batch 1_7, which had abnormally high values for Relative Error, the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\text{LT}}$. Best Relative (Output) Error: Interval$_{\text{GT}}$. Best Relative (Absolute) Error: Interval$_{\text{LT}}$.

4.1.2 Batch 2 (Uniform Distribution)

2.011001e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.715032e-8	9.812719e-8	1.242568e-8
Huffman (w/ Mean)	5.292509e-8	1.144586-7	1.405700e-8
Interval$_{\mathbf{LT}}$	3.491029e-8	5.600884e-8	8.975786e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.915155e-8	8.440368e-7	1.045315e-8
Interval$_{\mathbf{GT}}$	3.725583e-8	5.905715e-8	9.589161e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.002959e-8	7.454126e-8	1.039810e-8
Pairwise (Unsorted)	5.829681e-8	1.661159e-7	1.590148e-8
Pairwise (Unsorted; Mean)	5.951292e-8	1.661159e-7	1.590148e-8
Pairwise (”Simple”-Sorted)	3.718513e-8	5.473788e-8	9.609681e-9
Pairwise (”Simple” Sorted; Mean)	4.225435e-8	8.094460e-8	1.144051e-8
Pairwise (”Hyper”-Sorted)	4.059149e-8	7.491670e-8	1.040038e-8
Simple	6.318056e-8	1.372386e-7	1.646303e-8
Simple (w/ Mean)	6.640000e-8	1.417189e-7	1.742660e-8

Table 4.13: Table showing the results table for Batch 2_1 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Pairwise (”Simple”-Sorted). Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

1.838973e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.121812e-8	3.155639e-7	1.296047e-8
Huffman (w/ Mean)	5.696976e-8	3.003036-7	1.430626e-8
Interval$_{\mathbf{LT}}$	3.461391e-8	1.083331e-7	8.929786e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	3.766985-8	2.549320e-7	9.974837e-9
Interval$_{\mathbf{GT}}$	3.517095e-8	1.178548e-7	9.154255e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.021100e-8	2.595222e-7	1.049081e-8
Pairwise (Unsorted)	5.504365e-8	6.461107e-7	1.419061e-8
Pairwise (Unsorted; Mean)	6.252546e-8	3.212052e-7	1.577843e-8
Pairwise (”Simple”-Sorted)	3.818059e-8	2.217686e-7	9.732455e-9
Pairwise (”Simple” Sorted; Mean)	4.002859e-8	6.755604e-7	1.069265e-8
Pairwise (”Hyper”-Sorted)	3.900113e-8	1.283902e-7	9.970050e-9
Simple	6.923834e-8	5.463165e-7	1.798882e-8
Simple (w/ Mean)	7.333934e-8	3.090572e-7	1.893815e-8

Table 4.14: Table showing the results table for Batch 2_2 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Interval$_{\mathrm{LT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

2.202268e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.731128e-8	1.230821e-7	1.246990e-8
Huffman (w/ Mean)	6.170804e-8	1.298777e-7	1.595343e-8
Interval$_{\mathbf{LT}}$	3.530796e-8	6.423777e-8	9.268753e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.368264e-8	7.742022e-8	1.147758e-8
Interval$_{\mathbf{GT}}$	3.647972e-8	5.261279e-8	9.646541e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.307973e-8	7.132836e-8	1.143645e-8
Pairwise (Unsorted)	5.433637e-8	1.196686e-7	1.424595e-8
Pairwise (Unsorted; Mean)	6.085960e-8	1.379707e-7	1.641728e-8
Pairwise ("Simple"-Sorted)	4.171411e-8	7.066261e-8	1.110792e-8
Pairwise ("Simple" Sorted; Mean)	4.596172e-8	1.161101e-7	1.212218e-8
Pairwise ("Hyper"-Sorted)	4.029475e-8	6.560656e-8	1.058722e-8
Simple	6.814198e-8	1.238927e-7	1.819500e-8
Simple (w/ Mean)	7.087300e-8	1.315608e-7	1.905550e-8

Table 4.15: Table showing the results table for Batch 2_3 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\text{LT}}$. Best Relative (Output) Error: Interval$_{\text{GT}}$. Best Relative (Absolute) Error: Interval$_{\text{LT}}$.

1.957327e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.926273e-8	1.436896e-7	1.301848e-8
Huffman (w/ Mean)	5.655163e-8	5.119512e-7	1.542553e-8
Interval$_{\mathbf{LT}}$	3.833894e-8	1.049358e-7	1.049191e-8
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.272031e-8	1.794884e-7	1.193685e-8
Interval$_{\mathbf{GT}}$	3.887634e-8	1.164309e-7	1.037049e-8
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.788721e-8	1.510773e-7	1.311043e-8
Pairwise (Unsorted)	5.722334e-8	1.589659e-7	1.511767e-8
Pairwise (Unsorted; Mean)	5.669353e-8	3.099868e-7	1.493866e-8
Pairwise ("Simple"-Sorted)	3.870129e-8	1.066505e-7	1.037959e-8
Pairwise ("Simple" Sorted; Mean)	4.820356e-8	2.770519e-7	1.325296e-8
Pairwise ("Hyper"-Sorted)	3.567467e-8	1.075106e-7	9.313592e-9
Simple	6.754413e-8	3.622421e-7	1.809204e-8
Simple (w/ Mean)	7.219733e-8	3.665802e-7	1.951078e-8

Table 4.16: Table showing the results table for Batch 2_4 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise ("Hyper"-Sorted). Best Relative (Output) Error: Interval$_{\text{LT}}$. Best Relative (Absolute) Error: Pairwise ("Hyper"-Sorted).

2.153483e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.713184e-8	2.630115e-7	1.174995e-8
Huffman (w/ Mean)	5.386704e-8	5.850820e-7	1.339766e-8
Interval$_{\mathbf{LT}}$	4.028308e-8	1.341529e-7	1.003653e-8
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.490494e-8	2.305152e-7	1.157453e-8
Interval$_{\mathbf{GT}}$	3.635415e-8	1.304742e-7	9.211187e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	3.799476e-8	2.120647e-7	9.663951e-9
Pairwise (Unsorted)	5.603298e-8	4.704042e-7	1.420936e-8
Pairwise (Unsorted; Mean)	5.726252e-8	2.386007e-7	1.440505e-8
Pairwise (”Simple”-Sorted)	3.788221e-8	2.229834e-7	9.537449e-9
Pairwise (”Simple” Sorted; Mean)	4.203579e-8	1.306120e-7	1.066147e-8
Pairwise (”Hyper”-Sorted)	3.736412e-8	1.091534e-7	9.498991e-9
Simple	7.458798e-8	2.996539e-7	1.872627e-8
Simple (w/ Mean)	7.566135e-8	2.739839e-7	1.894580e-8

Table 4.17: Table showing the results table for Batch 2_5 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{GT}}$. Best Relative (Output) Error: Pairwise (”Hyper”-Sorted). Best Relative (Absolute) Error: Interval$_{\mathrm{GT}}$.

2.501862e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.093950e-8	1.320459e-7	1.253154e-8
Huffman (w/ Mean)	5.648534e-8	2.449166e-7	1.373064e-8
Interval$_{\mathbf{LT}}$	3.933726e-8	1.981635e-7	9.628812e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.600235e-8	2.456755e-7	1.108097e-8
Interval$_{\mathbf{GT}}$	4.230894e-8	3.216946e-7	1.038244e-8
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.251964e-8	3.510784e-7	1.035900e-8
Pairwise (Unsorted)	5.667158e-8	2.666116e-7	1.384702e-8
Pairwise (Unsorted; Mean)	6.097858e-8	3.731555e-7	1.503437e-8
Pairwise (”Simple”-Sorted)	4.730253e-8	1.202007e-7	1.136863e-8
Pairwise (”Simple” Sorted; Mean)	4.811910e-8	1.125042e-7	1.187956e-8
Pairwise (”Hyper”-Sorted)	4.124872e-8	1.277012e-7	1.004950e-8
Simple	7.918880e-8	2.829803e-7	1.898653e-8
Simple (w/ Mean)	7.919735e-8	3.052116e-7	1.894719e-8

Table 4.18: Table showing the results table for Batch 2_6 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Pairwise (”Simple”-Sorted; Mean). Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

2.290734e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.320172e-8	2.918606e-7	1.288029e-8
Huffman (w/ Mean)	5.457027e-8	2.319552e-7	1.370737e-8
Interval$_{\mathbf{LT}}$	4.278168e-8	9.167611e-8	1.037054e-8
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.693428e-8	1.103890e-7	1.152091e-8
Interval$_{\mathbf{GT}}$	3.764073e-8	1.021557e-7	9.163300e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.587085e-8	1.912947e-7	1.122340e-8
Pairwise (Unsorted)	5.309301e-8	4.864846e-7	1.285185e-8
Pairwise (Unsorted; Mean)	6.623300e-8	3.001636e-7	1.664800e-8
Pairwise (”Simple”-Sorted)	4.207821e-8	1.113955e-7	1.022097e-8
Pairwise (”Simple” Sorted; Mean)	7.898803e-8	1.731172e-7	1.215792e-8
Pairwise (”Hyper”-Sorted)	4.120575e-8	1.230607e-7	1.025962e-8
Simple	7.608570e-8	3.923291e-7	1.850527e-8
Simple (w/ Mean)	7.645104e-8	4.130112e-7	1.871051e-8

Table 4.19: Table showing the results table for Batch 2_7 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{GT}}$. Best Relative (Output) Error: Interval$_{\mathrm{LT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{GT}}$.

2.520818e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.177657e-8	1.063045e-7	1.261949e-8
Huffman (w/ Mean)	5.996210e-8	1.392924e-7	1.474323e-8
Interval$_{\mathbf{LT}}$	4.352335e-8	7.098574e-8	1.096010e-8
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.508241e-8	8.935776e-8	1.135426e-8
Interval$_{\mathbf{GT}}$	4.119476e-8	6.926378e-8	1.026904e-8
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.886028e-8	9.634435e-8	1.203221e-8
Pairwise (Unsorted)	6.056656e-8	1.487168e-7	1.489688e-8
Pairwise (Unsorted; Mean)	6.513386e-8	1.500698e-7	1.599374e-8
Pairwise (”Simple”-Sorted)	4.502098e-8	7.401243e-8	1.104753e-8
Pairwise (”Simple” Sorted; Mean)	4.765271e-8	9.626535e-8	1.182766e-8
Pairwise (”Hyper”-Sorted)	4.181050e-8	7.554463e-8	1.033975e-8
Simple	7.818812e-8	1.266864e-7	1.920192e-8
Simple (w/ Mean)	8.143603e-8	1.347995e-7	1.995567e-8

Table 4.20: Table showing the results table for Batch 2_8 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{GT}}$. Best Relative (Output) Error: Interval$_{\mathrm{GT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{GT}}$.

2.439102e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.393042e-8	2.207738e-7	1.305158e-8
Huffman (w/ Mean)	5.785348e-8	3.016069e-7	1.386462e-8
Interval$_{\mathbf{LT}}$	3.951483e-8	1.156768e-7	9.442704e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.829463e-8	1.759338e-7	1.175700e-8
Interval$_{\mathbf{GT}}$	4.784798e-8	1.203872e-7	1.153812e-8
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.582515e-8	1.511356e-7	1.112792e-8
Pairwise (Unsorted)	5.333454e-8	2.406627e-7	1.281596e-8
Pairwise (Unsorted; Mean)	6.115954e-8	2.119581e-7	1.430669e-8
Pairwise ("Simple"-Sorted)	4.740522e-8	1.234070e-7	1.107463e-8
Pairwise ("Simple" Sorted; Mean)	5.092851e-8	1.081987e-7	1.244773e-8
Pairwise ("Hyper"-Sorted)	3.906803e-8	1.370294e-7	9.408711e-9
Simple	7.724305e-8	2.530843e-7	1.832907e-8
Simple (w/ Mean)	7.989170e-8	2.677738e-7	1.903345e-8

Table 4.21: Table showing the results table for Batch 2_9 the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise ("Hyper"-Sorted). Best Relative (Output) Error: Pairwise ("Simple"-Sorted; Mean). Best Relative (Absolute) Error: Pairwise ("Hyper"-Sorted).

2.017318e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.890735e-8	1.086322e-7	1.189779e-8
Huffman (w/ Mean)	5.330262e-8	2.085921e-7	1.315278e-8
Interval$_{\mathbf{LT}}$	4.111782e-8	8.444039e-8	1.061900e-8
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.018828e-8	1.277069e-7	1.008075e-8
Interval$_{\mathbf{GT}}$	4.231459e-8	1.407350e-7	1.040799e-8
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.613727e-8	8.988846e-8	1.163809e-8
Pairwise (Unsorted)	5.533991e-8	1.286670e-7	1.345283e-8
Pairwise (Unsorted; Mean)	6.522069e-8	1.763179e-7	1.628922e-8
Pairwise ("Simple"-Sorted)	4.074724e-8	1.257651e-7	1.013360e-8
Pairwise ("Simple" Sorted; Mean)	4.231549e-8	1.738880e-7	1.089201e-8
Pairwise ("Hyper"-Sorted)	3.732448e-8	1.319565e-7	9.407388e-9
Simple	6.783079e-8	1.300976e-7	1.713253e-8
Simple (w/ Mean)	7.145665e-8	1.378049e-7	1.822276e-8

Table 4.22: Table showing the results table for Batch 2_{10} the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Pairwise ("Hyper"-Sorted). Best Relative (Output) Error: Interval$_{\mathrm{LT}}$. Best Relative (Absolute) Error: Pairwise ("Hyper"-Sorted).

2.193289e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.008299e-8	1.803091e-7	1.256052e-8
Huffman (w/ Mean)	5.641954e-8	2.768036e-7	1.423385e-8
Interval$_{\mathbf{LT}}$	3.897291e-8	1.028610e-7	9.881077e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.347312e-8	2.335456e-7	1.112135e-8
Interval$_{\mathbf{GT}}$	3.954440e-8	1.240181e-7	9.973252e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.484155e-8	1.295283e-7	1.114804e-8
Pairwise (Unsorted)	5.599496e-8	2.842808e-7	1.415295e-8
Pairwise (Unsorted; Mean)	6.155797e-8	2.393167e-7	1.557131e-8
Pairwise (”Simple”-Sorted)	4.162175e-8	1.231584e-7	1.042125e-8
Pairwise (”Simple” Sorted; Mean)	4.964879e-8	1.944252e-7	1.173747e-8
Pairwise (”Hyper”-Sorted)	3.935836e-8	1.080870e-7	9.923520e-9
Simple	7.212295e-8	2.654522e-7	1.806205e-8
Simple (w/ Mean)	7.469038e-8	2.481502e-7	1.887464e-8

Table 4.23: Table showing the accumulated results table for Batch 2, the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Interval$_{\mathrm{LT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

4.1.3 Batches 1 & 2 Combined

2.119689e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	4.855251e-8	3.089965e-7	1.228634e-8
Huffman (w/ Mean)	5.470577e-8	3.223279e-7	1.392073e-8
Interval$_{\mathbf{LT}}$	3.540136e-8	2.252224e-7	9.000113e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.075504e-8	2.246532e-7	1.095797e-8
Interval$_{\mathbf{GT}}$	3.585610e-8	2.350096e-7	9.083115e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.100494e-8	1.703319e-7	1.036146e-8
Pairwise (Unsorted)	5.238348e-8	3.853140e-7	1.328271e-8
Pairwise (Unsorted; Mean)	5.827939e-8	4.719605e-7	1.485919e-8
Pairwise ("Simple"-Sorted)	3.791616e-8	1.584859e-7	9.595409e-9
Pairwise ("Simple" Sorted; Mean)	4.550408e-8	2.370823e-7	1.118846e-8
Pairwise ("Hyper"-Sorted)	3.592520e-8	1.570916e-7	9.119280e-9
Simple	6.935977e-8	5.456830e-7	1.752880e-8
Simple (w/ Mean)	7.145281e-8	5.444282e-7	1.814699e-8

Table 4.24: Table showing the combined results for Table 4.11 and Table 4.23, the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Pairwise ("Hyper"-Sorted). Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

2.222856e-8	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Huffman	5.109117e-8	1.831086e-7	1.293216e-8
Huffman (w/ Mean)	5.728496e-8	2.722768e-7	1.456811e-8
Interval$_{\mathbf{LT}}$	3.719770e-8	9.163541e-8	9.450785e-9
Interval$_{\mathbf{LT}}$ (w/ Mean)	4.273500e-8	1.714771e-7	1.154677e-8
Interval$_{\mathbf{GT}}$	3.754205e-8	1.016749e-7	9.509667e-9
Interval$_{\mathbf{GT}}$ (w/ Mean)	4.303324e-8	1.135396e-7	1.086700e-8
Pairwise (Unsorted)	5.528736e-8	2.667827e-7	1.403278e-8
Pairwise (Unsorted; Mean)	6.125318e-8	2.254497e-7	1.561436e-8
Pairwise ("Simple"-Sorted)	3.981966e-8	1.044578e-7	1.008202e-8
Pairwise ("Simple" Sorted; Mean)	4.785224e-8	1.469036e-7	1.175839e-8
Pairwise ("Hyper"-Sorted)	3.778651e-8	9.663106e-8	9.589215e-9
Simple	7.260071e-8	2.388226e-7	1.833879e-8
Simple (w/ Mean)	7.481963e-8	2.332432e-7	1.899075e-8

Table 4.25: Table showing the combined results for Table 4.12 and Table 4.23, the value in the top-left corner shows the minimum possible value for Absolute Error. Best Absolute Error: Interval$_{\mathrm{LT}}$. Best Relative (Output) Error: Interval$_{\mathrm{LT}}$. Best Relative (Absolute) Error: Interval$_{\mathrm{LT}}$.

4.2 Results

As a brief discussion of which error metrics are more 'important', I have to note that this decision is very dependent on whatever use case this problem presents in but, as a general rule; E_{Rel_O} will be greater when the true answer is lower, as the 'minimum' amount of Absolute Error has shown to be relatively consistent across tables, while E_{Abs} (and E_{Rel_A}, which has proven itself to remain close to E_{Abs} across all tables. The method which generates the lowest E_{Abs} also generates the lowest E_{Rel_A} in the majority of cases) will be greater when the true answer is higher, as any bits which contain error would be affected by the output's higher exponent.

In analysing the tables, I am going to focus on the Accumulated Tables, as they represent a greater number of weight sets tested and, in theory, would cancel out the values of any outliers.

Table 4.11 Percentage Comparison	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Interval$_{\mathbf{LT}}$	47.795%	47.795%	47.772%
Interval$_{\mathbf{GT}}$	48.302%	41.893%	48.207%
Pairwise ("Hyper"-Sorted)	48.789%	24.954%	48.925%

Table 4.26: Table showing the percentage FPE of the top three methods in Table 4.11 (disregarding methods adjusted by Expected Mean), where the FPE in all three metrics generated by the 'Simple' Summation Method are considered 100%

Table 4.12 Percentage Comparison	E_{Abs}	E_{Rel_O}	E_{Rel_A}
Interval$_{\mathbf{LT}}$	48.167%	37.834%	48.121%
Interval$_{\mathbf{GT}}$	48.293%	36.729%	48.238%
Pairwise ("Hyper"-Sorted)	49.281%	40.100%	48.211%

Table 4.27: Table showing the percentage FPE of the top three methods in Table 4.12 (disregarding methods adjusted by Expected Mean), where the FPE in all three metrics generated by the 'Simple' Summation Method are considered 100%

First, let us examine Batch 1, where inputs are generated from a normal distribution with $\mu = 0.5$ and $\sigma = \frac{1}{6}$.

There is a very notable outlier seen in Table 4.7, where the Relative (Output) Errors are all roughly 10^{-6}, as opposed to roughly 10^{-7} in all other Tables. Due to this outlier, two Accumulated tables were created for Batch 1: Table 4.11, which includes Table 4.7's

results, and Table 4.12, which does not. Percentage Comparison Tables for both Tables 4.11 and 4.12 can be seen in Tables 4.26 and 4.27, since the 'Simple' Summation Method is present in these tables as a 'worst case scenario', we will treat it as "100% FPE".

In both Tables, $\text{Interval}_{\text{LT}}$ has the lowest values for E_{Abs} and E_{Rel_A}, with $\text{Interval}_{\text{GT}}$ following closely behind. Disregarding the methods adjusted by the Expected Mean as, in the vast majority of cases throughout every table generated, such adjustment increases all metrics, Pairwise ("Hyper"-Sorted) shows the third-lowest values for E_{Abs} and E_{Rel_A}.

Regarding E_{Rel_O}, Pairwise ("Hyper"-Sorted) has the lowest values in the plurality of Tables 4.1 through 4.10, but in the adjusted Table 4.12, $\text{Interval}_{\text{GT}}$ has the lowest value, reflecting a consistency across the constituent tables. Surprisingly, in Tables 4.11 and 4.12, Pairwise ("Hyper"-Sorted) only has the fourth-lowest value for E_{Rel_O}, ranking below both Interval methods and Pairwise ("Simple"-Sorted).

Based on these results, one would have to accept that the Interval Summation methods are best for this problem, when the inputs are Normally distributed, although deciding whether $\text{Interval}_{\text{LT}}$ or $\text{Interval}_{\text{GT}}$ is more promising is still difficult as one would need to decide which metric is most important. It is worth noting that the computationally simpler Pairwise ("Hyper"-Sorted) summation method scores very closely to these other values, and this may position it as a viable summation candidate.

Table 4.23 Percentage Comparison	E_{Abs}	E_{Rel_O}	E_{Rel_A}
$\text{Interval}_{\text{LT}}$	54.037%	38.749%	54.706%
$\text{Interval}_{\text{GT}}$	54.829%	46.720%	55.217%
Pairwise ("Hyper"-Sorted)	54.571%	40.718%	54.941%

Table 4.28: Table showing the percentage FPE of the top three methods in Table 4.23 (disregarding methods adjusted by Expected Mean), where the FPE in all three metrics generated by the 'Simple' Summation Method are considered 100%

Let us now examine Batch 2, where inputs were generated from a uniform distribution. A Percentage Comparison Table is presented in Table 4.28.

This batch featured no major outliers that I identified, and so we only have the one accumulated results Table, Table 4.23. This table's results show that $\text{Interval}_{\text{LT}}$ has the lowest values for all 3 metrics, with Pairwise ("Hyper"-Sorted) and $\text{Interval}_{\text{GT}}$ following closely behind, in that order.

Based on these results, $\text{Interval}_{\text{LT}}$, is the best summation method, but the closeness of Pairwise ("Hyper"-Sorted) to its scores implies that it too could be a valid method, as it is a simpler summation method computationally and still scores better than $\text{Interval}_{\text{LT}}$'s

counterpart, $\text{Interval}_{\text{GT}}$.

Table 4.24 Percentage Comparison	E_{Abs}	E_{Rel_O}	E_{Rel_A}
$\text{Interval}_{\text{LT}}$	51.040%	41.273%	51.345%
$\text{Interval}_{\text{GT}}$	51.696%	43.067%	51.818%
Pairwise ("Hyper"-Sorted)	51.795%	28.788%	52.025%

Table 4.29: Table showing the percentage FPE of the top three methods in Table 4.24 (disregarding methods adjusted by Expected Mean), where the FPE in all three metrics generated by the 'Simple' Summation Method are considered 100%

Table 4.25 Percentage Comparison	E_{Abs}	E_{Rel_O}	E_{Rel_A}
$\text{Interval}_{\text{LT}}$	51.236%	38.370%	51.534%
$\text{Interval}_{\text{GT}}$	51.710%	42.573%	51.855%
Pairwise ("Hyper"-Sorted)	52.047%	40.461%	52.289%

Table 4.30: Table showing the percentage FPE of the top three methods in Table 4.25 (disregarding methods adjusted by Expected Mean), where the FPE in all three metrics generated by the 'Simple' Summation Method are considered 100%

Since all the data was already generated, Accumulated results were gathered for both batches, creating a combined batch with 10,000 weight sets and 10,000 input sets for every weight set. Since half of the inputs here are Normally distributed, while the other half were Uniformly distributed, these accumulated results may be based on more realistic scenarios for this problem. As before, I have created two combined batch accumulated tables: Table 4.24, which includes the outlier Table 4.7's results, and Table 4.25, which does not. Once again, Percentage Comparison tables are presented in Tables 4.29 and 4.30.

In these tables, $\text{Interval}_{\text{LT}}$ has the lowest results in every metric, barring the E_{Rel_O} metric in Table 4.24, in which Pairwise ("Hyper"-Sorted) has the lowest value. $\text{Interval}_{\text{GT}}$ is a close second place to $\text{Interval}_{\text{LT}}$ in every metric where it is higher, barring the E_{Rel_O} metric in Table 4.25, where Pairwise ("Hyper"-Sorted) takes second place. In all other metrics, Pairwise ("Hyper"-Sorted) takes third place.

Based on these results, one would likely still concede that $\text{Interval}_{\text{Lt}}$ is still the best Summation Method, although Pairwise ("Hyper"-Sorted)'s lower complexity in implementation may still make it preferable in some cases.

Expected Mean E_{Abs}	**Table 4.11**	**Table 4.12**	**Table 4.23**	**Table 4.24**	**Table 4.25**
Huffman	112.696%	111.559%	112.652%	112.673%	112.123%
Interval$_{\mathbf{LT}}$	119.501%	118.991%	111.547%	115.123%	114.886%
Interval$_{\mathbf{GT}}$	115.545%	116.159%	113.395%	114.360%	114.627%
Pairwise (Unsorted)	112.771%	111.767%	109.935%	111.255%	110.791%
Pairwise ("Simple"-Sorted)	120.896%	121.257%	119.286%	120.012%	120.172%
Simple	102.431%	102.505%	103.560%	103.018%	103.056%

Table 4.31: Table showing the percentage of E_{Abs} when the Expected Mean is used on different Summation Methods. All 'Accumulated' Batch Tables were used to gather these results

Pairwise Percentage Comparison	**Table 4.11**	**Table 4.12**	**Table 4.23**	**Table 4.24**	**Table 4.25**
"Hyper"-Sorted (E_{Abs})	66.620%	66.127%	70.289%	68.581%	68.346%
"Hyper"-Sorted (E_{Rel_O})	42.376%	33.922%	38.021%	40.770%	36.221%
"Hyper"-Sorted (E_{Rel_A})	66.989%	66.318%	70.116%	68.655%	68.334%
"Simple"-Sorted (E_{Abs})	70.144%	69.388%	74.331%	72.382%	72.023%
"Simple"-Sorted (E_{Rel_O})	39.851%	33.832%	43.323%	41.132%	39.155%
"Simple"-Sorted (E_{Rel_A})	70.651%	69.825%	73.633%	72.240%	71.846%

Table 4.32: Table showing the percentage difference in all three FPE metrics between the different permutations of Pairwise Summation. In each cell of the table 100% is the amount of FPE in a given metric in a given table as generated by Unsorted Pairwise Summation.

Table 4.31 shows the relative FPE (Using the E_{Abs} metric) of the summation methods adjusted by the Expected Mean. In each cell of the table 100% represents the E_{Abs} FPE of that methods performance in a given table, without Expected Mean Adjustment. As can be seen, adjusting twitch the Expected Mean increases FPE by roughly 12-20%, barring the outlier case with Simple summation, where the Expected Mean Adjustment only adds approximately 3% more FPE.

Table 4.32 shows the relative FPE (In all three metrics) of the different Pairwise Summation methods used in this report. In each cell of the table 100% represents the FPE of Unsorted Pairwise's performance in the given metric in the given table. As can be seen, the new "Hyper"-Sorted Pairwise generally only creates roughly 66% of Unsorted Pairwise's E_{Abs} or E_{Rel_A}, and roughly 38% of Unsorted Pairwise's E_{Rel_O}. These are close to the improvements brought about by "Simple"-Sorted Pairwise, but "Hyper"-Sported Pairwise is consistently more efficient with regards to FPE.

4.3 Summary

Ultimately, based on all tables generated, the Interval$_{\mathrm{LT}}$ Summation Method produces the best results in the majority of error metrics across the three input distributions seen (Normal, Uniform, and a combination of the two). Interval$_{\mathrm{GT}}$ often trails close behind, with Pairwise ("Hyper"-Sorted) consistently producing the third best results. In an ideal world, this means that the Interval$_{\mathrm{LT}}$ method would be used in the majority of cases, but it is more computationally complex than Pairwise ("Hyper"-Sorted), meaning that this permuted pairwise summation method may be preferable in some situations.

Chapter 5

Conclusions & Future Work

Before diving into any conclusions, we shall first recount the contents of the Dissertation as a whole.

We began in Chapter 1, discussing the key terms of the dissertation title, namely 'Scaled Summation', and 'Floating Point Error', before describing several ways to reduce Floating Error, by trying to reduce Rounding Error, trying to avoid Catastrophic Cancellation, and trying to minimise the depth and values of any partial sums in the Summation. It was in this chapter that I also discussed the assumptions I was making regarding both the distributions of the inputs and the relative scale of the weighted inputs.

In the next chapter, Chapter 2, I discussed several existing Summation Methods with these factors that influence FPE in mind. These discussions included simple visualisations of summation trees, some basic pseudo-code samples, and 'real' examples, which used randomly generated floating point weights in order to allow the reader to see how the methods worked in practice.

With all of the information thus far, I used Chapter 3 to present to the reader two key ideas that I had designed and implemented for testing, one less-successful idea that adjusted the sum using the known Expected Mean of the summation (which proved to add more FPE than it cancelled out, adding between ~12% and ~20% more absolute error), and one much more promising idea that created a "Hyper"-Sorted permutation of weights for pairwise summation (which was found to generate only ~66% of the absolute error generated by Unsorted Pairwise, and only ~38% of Unsorted Pairwise's relative error).

Finally, as seen in Chapter 4, numerous tables were created, showing the results of many tests of all aforementioned methods, using three error metrics, E_{Abs}, E_{Rel_O}, and E_{Rel_A}, and three different types of input distribution, Normal, Uniform, and a Combination. The complex $\text{Interval}_{\text{LT}}$ Summation Method provided what were generally the lowest values for the three error metrics, (~51.2%, ~38.4%, and ~51.5% of the error

generated by Simple Summation for the metrics E_{Abs}, E_{Rel_O}, and E_{Rel_A}) with its counterpart, $\text{Interval}_{\text{GT}}$ (generating ∼51.7%, ∼42.6%, and ∼51.9% of the error generated by Simple Summation), and the much simpler Pairwise ("Hyper"-Sorted) (generating ∼52.0%, ∼40.5%, and ∼52.3% of the error generated by Simple Summation) often taking up second and third place, respectively.

Understanding the 'best' method based on this information, requires an understanding of the complexities of the methods. The two Interval methods $\text{Interval}_{\text{LT}}$ and $\text{Interval}_{\text{GT}}$ do provide the best results, but the "Hyper"-Sorted Pairwise method is markedly less complex to implement and has the further benefit that Unsorted Pairwise is already known to be a very efficient summation method when dealing with non-scaled summation (Higham, 1993). This means that in the short term, even though $\text{Interval}_{\text{LT}}$ appears to be the method that most reduces FPE, any programmers updating their code to be more efficient may just add a "Hyper"-Sorted permutation to existing pairwise code.

Of course, it may be the case that neither Interval nor "Hyper"-Sorted Pairwise are the most efficient generally. The tests undertaken in this dissertation use randomly generated numbers and it is always possible that not enough tests were run to reflect the 'truth' of the mathematics involved.

Regardless of these doubts, I maintain that this paper is, in the least, a good start towards investigating this problem in greater depth. It may be found that there is a pure mathematical way to derive the optimal scaled summation method based on the weights alone, but the work done here should have provided enough information to point a future researcher in the correct direction.

5.1 Future Work

There are many potential areas of Future Work that one could explore after this paper. Several examples are listed below.

'Adaptive' Summation Methods: Where methods are chosen on a 'case-by-case' basis based on initial weights.

Mathematical Derivation: Where the ideal method is derived using pure mathematics, as opposed to using thousands of randomly generated variables.

Further Tests: Computational and temporal limitations effected the number of tests I could run.

Other Methods: Analysis of other existing summation methods, or the development of novel summation methods.

Bibliography

Barabasz, B., Anderson, A., Soodhalter, K. M., and Gregg, D. (2020). Error analysis and improving the accuracy of winograd convolution for deep neural networks. *ACM Transactions on Mathematical Software*, 46(4):1–33.

Bernt, B. C., Evans, R. J., and Williams, K. S. (1998). *Gauss and Jacobi Sums*. Wiley.

Cantor, G. (1879). Ueber unendliche, lineare punktmannichfaltigkeiten. 1. [on infinite, linear point manifolds. 1.]. *Mathematische Annalen*, 15:1–7.

Cantor, G. (1891). Ueber eine elementare frage der mannigfaltigkeitslehre [on an elementary question of the theory of diversity]. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 1:75–78.

C++Reference (2023). std:normal_distribution. From cppreference.com. https://en.cppreference.com/w/cpp/numeric/random/normal$_{d}istribution$.

C++Reference (2024). std:uniform_real_distribution. From cppreference.com. https://en.cppreference.com/w/cpp/numeric/random/uniform$_{r}eal_{d}istribution$.

Fishburn, P. C. (1967). Letter to the editor—additive utilities with incomplete product sets: Application to priorities and assignments. *Operations Research*, 15(3):537–542.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48.

Hall, P. (1927). The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable. *Biometrika*, 19(3/4):240–245.

Higham, N. J. (1993). The accuracy of floating point summation. *SIAM Journal on Scientific Computing*, 14(4):783–799.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

IEEE 754-2019 (2019). Standard for floating-point arithmetic. Standard, Institute of Electrical and Electronics Engineers, New York City, U.S.A.

Irwin, J. (1927). On the frequency distribution of the means of samples from a population having any law of frequency with finite moments, with special reference to pearson's type ii. *Biometrika*, 19(3/4):225–239.

Lemons, D. S. (2002). *An Introduction to Stochastic Processes in Physics.* The Johns Hopkins University Press.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):140–141.

McCracken, D. D. and Dorn, W. S. (1964). *Numerical Methods and Fortran Programming: With Applications in Science and Engineering.* Wiley.

OEIS Sequence A270712 (2024). From The On-Line Encyclopedia of Integer Sequences. https://oeis.org/A270712.

Parker, M. (2019). *Humble Pi: A Comedy of Maths Errors.* Allen Lane.

Weisstein, E. W. (2024). Normal sum distribution. From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/NormalSumDistribution.html.

Zell, A. (1994). *Simulation Neuronaler Netze [Simulation of Neural Networks].* Addison-Wesley.