I. INTRODUCTION

Since most ML models are too complex for humans to properly interpret, methods have been devised to study these models and to provide the ability to understand how models arrive at certain predictions. This explanation is valuable for model developers, who need to understand e.g. the limitations from a model on a global perspective to adjust the model architecture or the training data. It is also valuable for model users, who are interested in the factors that drive specific model predictions. Hence, those methods can improve the overall model quality as well as model acceptance. In addition, governmental initiatives like the European Union’s ‘Artificial Intelligence Act’ [1] require that models applied in high-risk areas (e.g. access to education and essential private services) provide a minimum level of transparency for users, which highlights the importance of xAI methods.

In this paper, we will study models based on so-called Parameterized Quantum Circuits (PQCs) [2], which are also known as variational quantum circuits or quantum neural networks [3], and how xAI methods can be adjusted to these type of models.

Definition 1 (Parametrized Quantum Circuit (PQC)). A PQC with $N$ Qubits and $n+1$ features consists of $n+1$ single-qubit rotation gates $R_{\omega}^{(j)}(\varphi_i)$, as well as two-qubit entanglement gates. Here, $j = 1, \ldots, N$ denotes on which qubit the gate acts on, $\cdot = x, y, z$ is a placeholder for the specific axis of rotation, and $i = 1, \ldots, n+1$ indexes the parameters. For our purposes measurements in this circuit are always performed at the end and are in the computational basis.

Remark 2. In some circuits features are re-uploaded. This means that the feature is encoded by some rotation several times at different parts of the circuit. In the following, we will continue the analysis without re-uploading. However, the re-uploading case is easily realised by just equalizing some features $x_i = x_j = \ldots$.

Theorem 3 (Output and Learning Behavior). For a quantum circuit as defined in Definition 1 the following holds [4]: The output is given as a truncated Fourier series with \( \Omega = \{-1, 0, 1\}^{n+1} \) as frequency spectrum

\[
  f(\varphi) = \sum_{\omega \in \Omega} a_{\omega} \sin(\omega, \varphi) + b_{\omega} \cos(\omega, \varphi).
\]

This means that the result of PQCs can be viewed as truncated Fourier Series. In our case the parameters $a_{\omega}, b_{\omega}$ are real numbers and $a_{\omega}$, $b_{\omega}$ are trigonometric polynomials in $\theta$ [4]. The structure of the circuit dictates the amount of control available in training the circuit.

II. LITERATURE REVIEW

With the general availability of early quantum computers in the cloud [6–8] a new field developed in finding useful problems to tackle with the current generation of noisy quantum computers [9, 10]. At the core of this effort are the aforementioned PQCs, as they have some inherent resistance against systematic errors of the devices. QAOA [11] and VQE [12] are the most famous algorithms that employ this parameterized approach. Later on, it has been repurposed for various quantum machine learning models [13, 14]. There are still a lot of open questions regarding the effectiveness of training [15, 16] and their expressivity [17] as machine learning models.

Nevertheless, there is some interesting research into the behaviour of PQCs as machine learning models [4]. We try to build upon this work by introducing explainable AI (xAI) to the field of quantum machine learning. Standard xAI methods might help elucidate the behavior of current PQC-based machine learning models.
III. REVIEW OF MODEL-AGNOSTIC EXPLAINABILITY METHODS

There are many model-agnostic explainability methods currently available. Since our main concern is with PQC based models, we are going to assume that our models are at least continuously differentiable. The methods under consideration in our paper are KernelSHAP [22] and Integrated Gradients [23].

In the following, $f$ denotes the model, $x$ the input value for which an explanation is sought and $b$ a base value.

A. Integrated Gradients (IG)

IG [23] is a fast method relying on evaluating the gradient $\nabla f$ at equidistant points. Denote by $\gamma_{i,N} = i/Nx + (1 - i/N)b; i = 0, \ldots, N$ an $N$ mesh. For the purposes of computing IG efficiently we need to approximate the partial derivative. To do this, we move $\gamma_{N;i}$ along the $e$ axis by a shift $\delta_e$. Using the trapezoid rule and a simple approximation for the partial derivative, we get:

$$\text{IG}(e) = \frac{1}{0} \frac{\partial f}{\partial x_e}(\gamma(s)) \frac{d\gamma_e}{ds}(s) ds$$

$$\approx \frac{x_e - b_e}{2N} \sum_{i=0}^{N-1} \frac{\partial f}{\partial x_e}(\gamma_{N;i+1}) + \frac{\partial f}{\partial x_e}(\gamma_{N;i})$$

$$= \frac{x_e - b_e}{N} \sum_{i=1}^{N-1} \frac{\partial f}{\partial x_e}(\gamma_{N;i})$$

$$+ \frac{x_e - b_e}{2N} \left( \frac{\partial f}{\partial x_e}(x_e) + \frac{\partial f}{\partial x_e}(b_e) \right)$$

$$\approx \frac{x_e - b_e}{2N\delta_e} \sum_{i=1}^{N-1} f(\gamma_{N;i} + \delta_e) - f(\gamma_{N;i} - \delta_e)$$

In the above derivation the end terms were neglected since for large $N$ they will vanish to zero.

IG is a very fast method for differentiable models. It follows a path in a straight line from start $b$ to end $x$. This makes IG a good choice for large, high-dimensional models, especially in computer vision.

In equation 2 we present two ways of estimating IG values. One with gradients and in the last line with a specific gradient approximation. In settings where differentiable models have good gradient approximations readily available, the first version can be used. If not, the latter can be used as well.

B. SHAP

We are restricting our discussion of SHAP to Baseline SHAP (BS) which is a sub-variant of KernelSHAP. BS uses a single base value $b \in \mathbb{R}^{n+1}$ and an input value $x \in \mathbb{R}^{n+1}$. To get KernelSHAP from BS we can use BS with several base vectors $b_1, \ldots$ and average the results over those.

Let $F = [n + 1]$ denote the set of features. Let $e \in F$ the feature for which an explanation is desired. Let $S \subseteq F \setminus \{e\}$ be a set of features. Denote by $\bar{S}$ the opposite set in $F \setminus \{e\}$ such that $F = S \cup \bar{S} \cup \{e\}$. Next we define the intermediate vectors

$$(gs)_i = \begin{cases} x_i, & i \in S; \\ b_i, & \text{otherwise}. \end{cases}$$ (3)

Then the BS value for $e$ is defined as

$$\text{Sh}_f(e) = \frac{1}{n+1} \sum_{S \subseteq F \setminus \{e\}} \frac{|S|!(n - |S|)!}{n!} (f(g_{S \cup \{e\}}) - f(gs))$$

$$= \frac{1}{(n + 1)!} \sum_{P \in \text{Perm}(F)} (f(g_{P \cup \{e\}}) - f(g_P))$$

(4)

In the last line $\text{Perm}(F)$ denotes the set of permutations of $F$ and $P$ a specific permutation of $F$. $P_e$ denotes the elements of the permutation that occurred before feature $e$. I.e. Let $P = (1, 3, 2)$ then $P_2 = \{1, 3\}$.

C. Comparison of IG and BS

Both IG and BS are black-box methods. They do not generally pose many requirements for explainability. The strongest assumption is the requirement of models to be differentiable. This might exclude tree models and neural networks using step-functions as activation function. However, this can be alleviated by using a regularization procedure (i.e. folding against an appropriate derivative of a $C^\infty$ function).
Both IG and BS are path-dependent feature-perturbation methods. IG samples points along the line spanned by \((b, x)\) while BS samples its points from the set of vertices of the axes-aligned orthotope spanned by \((x, b)\). Here we can see their similar structure.

BS and IG are model-centric explainability methods. Let \((X, F, \mathbb{P})\) denote the probability space from which data is drawn to train the model \(f\). Both IG and BS will ignore the underlying correlations of the data. BS treats features as independent. In case of linear Models this can be overcome by a correlation correction if the data follows a multivariate Gaussian distribution. IG will assume a correlation of features along the direction of a single axis. We need \(2(N-1)!\) function evaluations for computing the IG-value along a single axis. We need \(2(N-1)n\) to compute IG-values for all features.

\[
\text{IG}_e(g) = \text{IG}_e(f) + \frac{x_e - b_e}{2N\delta_e} \sum_{i=1}^{N-1} \epsilon_{\gamma_{N,i} + \delta_e} - \epsilon_{\gamma_{N,i} - \delta_e}
\]
\[
\text{IG}_e(g) = \text{IG}_e(f) + \frac{x_e - b_e}{2N\delta_e} \sum_{e \in S} \epsilon_i
\]

Using the central limit theorem (CLT), the error \(X_N\) converges in distribution to \(\mathcal{N}(0, \frac{n^2}{2(n+1)}\).

We employ the same approach in analysing the stability of BS. In equation 4 two definitions of BS were given. The summation over all permutations has duplicate function evaluations over single points, while the definition using the power set of \(F \setminus \{e\}\) has the individual evaluations scaled by a term, weighing extremal points stronger than others.

Let us first consider the (inefficient) implementation via the permutation sum. If the function is evaluated at the same point for every permutation anew, then each error for the sum is independent. Using linearity of the SHAP values we have: \(\text{Sh}_\delta(e) = \text{Sh}_f(e) + \text{Sh}_\epsilon(e)\) and further:

\[
\text{Sh}_\epsilon(e) = \frac{1}{(n+1)!} \sum_{P \in \text{Perm}(F)} \epsilon_{x} - \epsilon_{x}'.
\]

Using the CLT we get \(X_n = \text{Sh}_\epsilon(e) \sim \mathcal{N}(0, \frac{2\sigma^2}{n+1})\).

The inefficient algorithm reduces the impact of any error, while the number of function evaluations explodes significantly. When evaluating the error with the standard formulation we face a different scenario:

\[
\text{Sh}_\epsilon(e) = \frac{1}{(n+1)!} \sum_{S \subseteq F \setminus \{e\}} (S!)(n - |S|)! (\epsilon_S - \epsilon_S')
\]
\[
= \frac{1}{n+1} \sum_{s=0}^{n} \sum_{S \subseteq F \setminus \{e\}; |S| = s} \binom{n}{s}^{-1} (\epsilon_S - \epsilon_S').
\]

The inner sum, sums over \(\binom{n}{s}\) many sets. For \(s = 0\) or \(s = n\) the number of summations is 1. This exactly fits our intuition that errors at the extremal points of the spanned box have the strongest impact on the outcome. However, even in this case we can expect some convergence against a normal distribution. As the number of features increases we expect \(X_n = \text{Sh}_\epsilon(e) \sim \mathcal{N}(0, \frac{\sigma^2}{n})\).

IV. EXTENDING SHAP TO PQCS

A. Mathematical Results

We are going to employ the result from Theorem 3 to find a (faster) extension of BS for PQCs. First we recall some definitions.
**Definition 4** (Multivariate Polynomials). Let $V$ be a $K$ vector-space, where $K$ is a field. We define a polynomial $p$ as an element of $\bigoplus_{i=0}^p S^i(V)$ where $S^i(V)$ denotes the space of symmetric multivariate forms. Elements of $S^i(V)$ are also called monomials of order $i$. Each monomial of order $i$ can be represented by a totally symmetric tensor $\Lambda_i$. Totally symmetric tensors allow for a rank-1 decomposition as follows

$$\Lambda_i = \sum_{j=1}^p \lambda_j v_j^{\otimes i}; \alpha_j \in K, v_j \in V. \quad (7)$$

This form is easily stored on a computer. To show the relationship between rank one compositions and Fourier Series note the simple Taylor expansion.

$$\exp(-i\langle \omega, \varphi \rangle) = \sum_{k \geq 0} \frac{(-1)^k i^k \langle \omega, \varphi \rangle^k}{k!} \approx \sum_{k=0}^K \frac{(-1)^k i^k \langle \omega, \varphi \rangle^k}{k!} \quad (8)$$

Expanding all terms in a Fourier Series shows that the wave vectors $\omega^{\otimes k}$ represent a natural choice for the rank-1 decomposition. Note that Taylor expansions are not the only choice here. Other polynomial approximation algorithms can be used as well. Chebyshev approximations will reduce the required polynomial order by about half.

**Proposition 5** (PolynomialSHAP). Let $\Lambda$ be a totally symmetric tensor of order $r$ with a rank-one decomposition as in $\text{(7)}$. Let $x, b \in \mathbb{R}^m$ be input and baseline, respectively. Then the BS values are given by:

$$\text{Sh}(e) = 2 \sum_{j+m+k=r \atop 0 \leq j, m, k \leq r \atop m \text{ odd, } k \text{ even}} \sum_{\gamma \in \mathbb{N}^n \atop |\gamma| = k} \frac{1}{l(\gamma) + 1} \binom{k}{\gamma} \sum_{i=1}^p \langle v_i, M \gamma \rangle \langle v_i, \Delta_e \rangle^m \lambda_i \prod_{h=1}^n (v_{ih} \alpha_h)^m. \quad (9)$$

Here the following definitions are made:

1. $\mathbb{N}$ denotes the natural numbers with $0$;
2. $M = (x + b)/2$; $\Delta_e = x - b$; $\Delta_{e'}$ is the vector $\Delta_e = 0$ for $i \neq e$ and $\Delta_e$ otherwise; $\alpha_h = ((x + b)/2)_h$;
3. $l(\gamma)$ denotes the number of odd indices in $\gamma$.

The proof of this statement can be found in the supplement. The formula above extends the concept of Linear SHAP to Multivariate polynomial functions and runs in $O(m^n)$, where $m$ denotes the number of features and $r$ the degree of the polynomial.

**B. Algorithmic Description and Runtime**

Algorithm 1 describes the qSHAP routine.

**Algorithm 1 Compute SHAP Values for PQCs $f$**

Require: $f$ function describing the PQC, $n$ number of features, $N_i$ number of times feature $i \in \{1, \ldots, n\}$ is encoded. $S$ set of samples in data-space, $k$ order of polynomial approximation.

1: Set of admissible wave-vectors $\Omega = \prod_{i=1}^n [-N_i, N_i]_Z$.
2: for $\omega \in \Omega$ do
3: Compute Fourier Coefficients $a_\omega, b_\omega$ with sample points $S$
4: Using taylors theorem, compute polynomial extension up to order $k$.
5: end for
6: Set Sh(e) $\leftarrow$ 0 for all features $1 \leq e \leq n$.
7: for $\omega \in \Omega$ do
8: for $(k=1)$ to $K$ do
9: Use PolynomialSHAP to compute contributions: $\text{Sh}_{\omega,k}(e)$.
10: $\text{Sh}(e) \leftarrow \text{Sh}(e) + \text{Sh}_{\omega,k}(e)$.
11: end for
12: end for

Essentially the algorithm has three steps:

1. Evaluate the PQC on random sample points.
2. Compute the Fourier decomposition. And form a multivariate polynomial approximation in rank-one form.
3. Use the PolynomialSHAP Algorithm to compute the contributions from each term.

**C. Open Issues and Further Direction**

Assuming access to quantum hardware the limiting issue for this approach is to obtain a reasonable rank-1 approximation. Currently, having access to the full set of Fourier Modes is required.

The algorithm does not scale exponentially in the number of qubits used for the PQC, but rather in the number of features present in the algorithm. At present no efficient algorithm for Fast Fourier transformation is known to the authors that can take advantage of the way PQCs activate their Fourier terms. The Fourier terms seem to be concentrated close to the border of $\Omega$.

Subsequent steps of the algorithm run at most in polynomial time.

**V. EXPERIMENTS**

Our experiments are conducted with the 2 by 2 Bars and Stripes problem. All possible images are shown in
FIG. 1. For our examples we use the bars and stripes data set. All possibilities of two by two pixel images are shown in this figure. The two images on the left are examples of stripes and the two images on the right are examples of bars. Our xAI models are applied to classifiers trained to distinguish bars from stripes.

Figure 1. The classifiers were trained to distinguish between the images of bars and images of stripes. The three classifiers each used a different parameterized quantum circuit shown in figures 5, 6 and 7. We are going to compare the KernelSHAP implementation of BS, IG, and our approximate qSHAP algorithm both with noise and without noise on these trained PQC classifiers.

For all xAI algorithms and backends we use the same model with the same parameters. That means that the training for each model was performed on classical hardware. The simulations in this case had no noise. Then we compared the results of the different xAI methods across a range of different hardware: classical simulation with no noise, classical simulation with shot-noise, classical simulation with shot-noise and a noise model and execution on a real QPU. To reach optimal model performance in each scenario it is generally recommended to perform the training on the same hardware, on which the model is used later on. However, in this case each model would have different model parameters and the results of the xAI methods would be harder to compare. For this reason we deployed the classically trained model across all hardware options.

Other techniques used for noise reduction on NISQ machines are mentioned in [24]. We did not train the models for each of the noise models further. Instead we can use the different performance of the explainers as a check of robustness against noise.

A. Single-qubit classifier

The single qubit classifier uses the same qubit to upload the top two pixels of the 2 by 2 Bars and Stripes images. The circuit for this classifier is given in Figure 5 and allows it to effectively learn an XOR operation of the two inputs. These inputs are the two top pixels of a bars and stripes image. If they are the same, the image shows stripes and if they are different, the image shows bars.

Looking at the xAI models in the noiseless case, we see that the two upper pixel have an effect towards the model prediction. The bottom pixel roughly have a predictive value of zero. This is the expected behavior as only the upper pixels are used for the model.

With the inclusion of the first noise model this effect does not seem to change with KernelSHAP and our qSHAP method. IG, on the other hand, already seems to struggle due to the noise of this model. With increasing noise this discrepancy becomes more and more pronounced. The noise model on the real hardware seems to be the most severe, causing a lot of artifacts for IG and to a lesser extent for KernelSHAP. The qSHAP method seems to be the most robust against this noise. The noise even leads to negative contributions for a pixel. This is most pronounced in the bottom left pixel for IG, yet this pixel is not used in the classifier.

B. Two-qubit classifier

The two-qubit model is slightly more complicated compared to the one-qubit model. It uses a circuit with two qubits, where each qubit is initialized with a rotation dependent on one of the upper two pixels of the bars and stripes image. Again the bottom two pixel are not used in the classifier.

The different xAI models fare similarly for this classifier. This can potentially be explained by the fact that noise does not have as large an influence as for the one-qubit classifier.
models due to e.g. noise. Nevertheless, in the noiseless case, all xAI models show that only the lower two qubits have an effect on the classification.

**D. Classification Performance vs. Explainability**

In xAI research one is often faced between achieving a high classification performance or achieving fast and "good" explanations. To a limited degree we see this issue appear here as well. The four-qubit model has the highest number of trainable parameters, meaning that in theory, the space of functions it can encode is much larger. The classification problem presented here is very simple and can be solved in various ways by only taking a subset of the features. In our four-qubit circuit, the qubits that are furthest away from the measurement qubit experience a lesser weighing than the other qubits. This behaviour is similar to classical neural networks where it is harder for the training algorithms to correctly adjust the weights. Furthermore, the quantum nature of PQCs introduces noise, whose impact increases with the size and complexity of the circuit.

**E. Runtime for qSHAP**

When discussing the runtime of this algorithm we should discuss three parts here:

1. The Sampling Step.
2. The Fourier Step.
3. The Polynomial Step.

During the Sampling Step the quantum circuit is sampled $S$ times. The amount of samples required depends on the dimension of the feature space, the circuit noise, etc. This step is linear in the number of samples used. The necessary number of samples should be drawn from convergence criteria from the stochastic integration methods used. However, in practice the size of the circuit is also an issue. There are many steps involved in compiling the logical circuit to the circuit that is evidently used by the hardware. A more complex circuit will introduce a higher overhead, resulting in less evaluations per minute.

Assuming a probabilistic rate of convergence of $O((n+1)^{-1/2})$ the number of circuit evaluations should scale like $O(e^{-1}(n+1)^2)$.

In our runs we used between 250 and 300 circuit evaluations and the below table lists the total time it took to gather the data:
| Algorithm      | Samples | Time [s]     |
|---------------|---------|--------------|
| Single Qubit  | 250     | 399.526641   |
| Two Qubit     | 250     | 497.247195   |
| Four Qubit    | 300     | 682.531422   |

TABLE I. Table comparing the runtimes recorded for our experiments of qSHAP on the Rigetti-M-11 hardware.

Starting with the Fourier Step we do not use the QPU any longer. This step is used to estimate which Fourier modes are activated. Importantly, the search space scales exponentially with the number of features and not the number of qubits (as the phase space does). Thus the limiting factor is the number of features in the model. Let $N_\omega$ denote the number of fourier modes found.

Lastly, the Polynomial Step runs in polynomial time in the number of features. Importantly, this step is also depending on $N_\omega$.

F. Runtime for IG

IG was executed with 20 nodes on the line from base value to the input value. For each point we had to evaluate the circuit eight times to approximate the partial derivatives. The runtimes are shown below.

| Algorithm      | Samples | Time [sec]   |
|---------------|---------|--------------|
| Single Qubit  | 20      | 240.778704   |
| Two Qubit     | 20      | 261.569914   |
| Four Qubit    | 20      | 343.405340   |

TABLE II. Table comparing the runtimes recorded for our experiments of qSHAP on the Rigetti-M-11 hardware.

VI. CONCLUSION

In this paper we have applied two black-box explainers to nascent quantum machine learning models. The drawback of these standard black-box xAI models is their exponential scaling and their behavior under the noise of NISQ devices. As a first step in resolving these issues we propose qSHAP, which is an xAI method specifically designed for PQCs.

Please note that this algorithm does not resolve issues related to the exponential scaling. However, the algorithm does not scale exponentially with respect to the number of qubits, but with respect to the number of features.

For qSHAP, the adverse effect of noise is considerably reduced compared to the other explainers.

While there is a large overlap between xAI for PQCs and the general simulation of circuits, we expect some interesting new approaches to arrive from the recent breakthroughs in simulating circuits used to show quantum advantage [13].

Future work can also be focused on finding efficient algorithms for approximating PQCs with multivariate polynomials in rank-one decomposition. Together with the PolynomialSHAP subroutine, this would result in an efficient algorithm for computing SHAP values for PQCs.

VII. ACKNOWLEDGEMENTS

This work was supported by the German Federal Ministry for Economic Affairs and Climate Action through project PlanQK (01MK20005D).

Special thanks goes to Till Duesberg who provided welcome vetting of many aspects of the initial drafts and Dr. Cristian Grozea who provided constructive criticism throughout. We also want to thank the d-fine internal reviewers Dr. Daniel Ohl de Mello and Dr. Ulf Menzler.

[1] Council of European Union, Proposal - laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts (2021).
[2] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, Parameterized quantum circuits as machine learning models, Quantum Science and Technology 4, 043001 (2019).
[3] E. Farhi and H. Neven, Classification with quantum neural networks on near term processors, arXiv preprint 10.48550/arXiv.1802.06002 (2018).
[4] M. Schuld, R. Sweke, and J. J. Meyer, Effect of data encoding on the expressive power of variational quantum-machine-learning models, Phys. Rev. A 103, 032430 (2021).
[5] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, Data re-uploading for a universal quantum classifier, Quantum 4, 226 (2020).
[6] IBM, IBM quantum experience (2016).
[7] Amazon, AWS bracket (2020).
[8] Microsoft, Azure quantum (2021).
[9] S. J. Devitt, Performing quantum computing experiments in the cloud, Phys. Rev. A 94, 063209 (2016).
[10] W. Zeng, B. Johnson, R. Smith, N. Rubin, M. Reagor, C. Ryan, and C. Rigetti, First quantum computers need smart software, Nature 549, 149 (2017).
[11] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, arXiv preprint 10.48550/arXiv.1411.4028 (2014).
**Appendix A: Diagrams of the Circuits**

This section contains the circuits that we used for our experiments.

**FIG. 5.** Single-Qubit circuit that can learn the 2 by 2 Bars and Stripes by using two pixels in a row or in a column. This equates to learning XOR.

**FIG. 6.** Two-Qubit circuit that can learn the 2 by 2 Bars and Stripes by using two pixels in a row or in a column. This equates to learning XOR.
Appendix B: Proof of Proposition 5

In this section we detail the proof for Proposition 5. The following Lemma demonstrates our approach for the quadratic model and motivates our general approach.

Lemma 6. Consider the BS problem with input $x$ and baseline $b$ for a quadratic form model $f: \mathbb{R}^{n+1} \rightarrow \mathbb{R}, x \mapsto x^T C x$ where $C \in \mathbb{R}^{(n+1) \times (n+1)}$ denotes a symmetric matrix. With mean $M = \frac{1}{2} (x + b)$ and difference $\Delta = (x - b)$. The shap values can be computed as:

$$\text{Sh}(e) = 2 M^T C \Delta_e$$  \hspace{1cm} (B1)

Proof. Using $b^T C a = a^T C b$ for all $a, b \in \mathbb{R}^{n+1}$:

$$\text{Sh}(e) = \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||S^c|}{(n+1)!} \left[ (x_S + x_e + b_S)^T C (x_S + x_e + b_S) - (x_S + b_e + b_S)^T C (x_S + b_e + b_S) \right]$$

$$= \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||S^c|}{(n+1)!} 2 \left( x_S + b_S + \frac{1}{2} (x_e + b_e) \right)^T C (x_e - b_e)$$

Evaluating the left side of the form:

$$2 \left( x_S + b_S + \frac{1}{2} (x_e + b_e) \right)$$

$$= 2 \left( \frac{x_S + x_e + x_S + b_e + b_S + b_S}{2} + \frac{x_S - b_S - (x_S - b_S)}{2} \right)$$

$$= 2 \left( M + \frac{1}{2} (\Delta_S - \Delta_S) \right)$$

Thus:

$$\text{Sh}(e) = \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||S^c|}{(n+1)!} \left[ 2 M^T C \Delta_e + (\Delta_S - \Delta_S^c)^T C \Delta_e \right]$$

Since we are summing over all subsets of $[n+1] \setminus \{e\}$ we have an even number of summands. The transform $S \mapsto S^c$ is bijective and maps the summand:

$$\frac{|S||S^c|}{(n+1)!} (\Delta_S - \Delta_S^c)^T C \Delta_e \mapsto -\frac{|S||S^c|}{(n+1)!} (\Delta_S - \Delta_S^c)^T C \Delta_e.$$ 

Thus the second sum cancels out and we are left with:

$$\text{Sh}(e) = 2 M^T C \Delta_e \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||S^c|}{(n+1)!}$$
We evaluate the last sum. Instead of directly summing over all subsets \( S \) we sum over the subsets of given size \( s \) and then over all sizes. Assume we have subset of \( S \) of size \( s \). There are \( \binom{n}{s} \) ways to select \( s \) elements out of the set \([n+1] \setminus \{e\}\). Thus we can transform the sum:

\[
\sum_{S \subseteq [n+1] \setminus \{e\}} \binom{|S|}{s} \binom{|S|!}{(n+1)!} = \sum_{s=0}^{n} \sum_{S \subseteq [n+1] \setminus \{e\}} s!(n-s)! \binom{n}{(n+1)!} \\
= \sum_{s=0}^{n} \frac{s!(n-s)!}{(n+1)!} \binom{n}{s} \\
= \sum_{s=0}^{n} \frac{1}{n+1} = 1
\]

\[
\square
\]

For two vector spaces \( V, W \) and bases \( a_1, \ldots, a_n \) and \( b_1, \ldots, b_m \) we can represent any \( r \)-multilinear map \( \Lambda \) by the values of \( \Lambda(a_{i_1}, \ldots, a_{i_r}) \) in the base of \( W \). These values are denoted by \( \Lambda_{i_1,\ldots,i_r;j} \) for \( 1 \leq j \leq m; 1 \leq i_1, \ldots, i_r \leq n \). If \( W \) is one-dimensional. This representation is thus naturally in the tensor product written as:

\[
\Lambda(v_1, \ldots, v_r) = \sum_{1 \leq j_1, \ldots, j_r \leq n} \alpha_{1;j_1} \cdots \alpha_{r;j_r} \Lambda(a_{i_1} \otimes \cdots \otimes a_{i_r}) \\
= \sum_{j=1}^{m} \sum_{1 \leq j_1, \ldots, j_r \leq n} \Lambda_{i_1,\ldots,i_r;j} \alpha_{1;j_1} \cdots \alpha_{r;j_r} b_j
\]

If furthermore \( \Lambda \) is symmetric then for any permutation \( \pi \) we have \( \Lambda_{i_1,\ldots,i_r;j} = \Lambda_{i_{\pi(1)},\ldots,i_{\pi(r)};j} \) and that:

\[
\Lambda(v_1, \ldots, v_r) = \sum_{1 \leq j_1, \ldots, j_r \leq n} \alpha_{1;j_1} \cdots \alpha_{r;j_r} \Lambda(a_{i_1} \otimes \cdots \otimes a_{i_r}) \\
= \sum_{j=1}^{m} \sum_{1 \leq j_1, \ldots, j_r \leq n} \sum_{|\beta| = r} \frac{n!}{\beta!} \Lambda_{i_1,\ldots,i_r;j} \alpha_{1;j_1} \cdots \alpha_{r;j_r} \Lambda(a_{i_1} \lor \cdots \lor a_{i_r}) \\
= \sum_{j=1}^{m} \sum_{1 \leq j_1, \ldots, j_r \leq n} \sum_{|\beta| = r} \frac{n!}{\beta!} \Lambda_{i_1,\ldots,i_r;j} \alpha_{1;j_1} \cdots \alpha_{r;j_r} \beta b_j
\]

\( r \)-multilinear maps are our generalisation for monomials. Below we introduce some non-standard notations for convenience:

1. Let \( x \in \mathbb{R}^n \), then for any \( l \geq 0 \) we let \( x^{\otimes l} \) denote the \( l \)-fold self-tensorproduct of \( x \);
2. For \( \beta \in \mathbb{N}^n \) and \( |\beta| = \beta_1 + \cdots + \beta_n = r \) we define \( \binom{r}{\beta} = \frac{r!}{\beta! \cdot (r-\beta)!} \);
3. Let \( \beta \in \mathbb{N}^n \) be an \( n \)-dimensional index vector of rank \( r = |\beta| \). For \( \alpha \in \mathbb{R}^n \) we have

\[
\alpha^{\wedge r} = \sum_{|\beta| = r} \binom{r}{\beta} \alpha_1^{\beta_1} \cdots \alpha_n^{\beta_n} e_1^{\wedge \beta_1} \lor \cdots \lor e_n^{\wedge \beta_n};
\]

4. We introduce a generalization of the scalar product we call \( \odot \): \( \Lambda \odot M^{\otimes l} = \sum_{i_{r-1}+1=1}^{n} \Lambda_{i_{r-1}+1,\ldots,i_r} M_{i_{r-1}+1,\ldots,i_r} \) collapses the last dimension of the tensor \( \Lambda \). Since \( \Lambda \) is symmetric \( r \)-multilinear the exact sequence of collapse is irrelevant.
With the above notation we can rewrite the multinomial expansion as:

\[
(x_1 + \ldots + x_n)^k = \sum_{|\beta|=k} \binom{k}{\beta} x_1^{\beta_1} \cdots x_n^{\beta_n} = \sum_{1 \leq \beta_1 \leq \beta_2 \leq \cdots \leq \beta_k \leq k} x_1^{\beta_1} \cdots x_n^{\beta_n} = \sum_{\beta \in [1,n]^k} (x^{\otimes k})_{\beta}.
\]

Furthermore for symmetric multilinear forms we have:

\[
(\Lambda \otimes M^{\otimes r}) \circ x^{\otimes s} = \Lambda \circ (M^{\otimes r} \otimes x^{\otimes s}) = \Lambda \circ (x^{\otimes s} \otimes M^{\otimes r})
\]

For convenience we drop the parenthesis.

**Proposition 7.** Let \( V \) be a \( K \) vector space and \( \Lambda : V^r \to K \) be a symmetric \( r \)-multilinear form. Let \( a_1, \ldots, a_r \in V \). A rank one decomposition of \( \Lambda \) is given by a natural number \( p \), vectors \( v_1, \ldots, v_p \in V \) and coefficients \( \lambda_1, \ldots, \lambda_p \in V \) such that:

\[
\Lambda(a_1, \ldots, a_r) = \sum_{i=1}^p \lambda_i v_i^{\otimes r}.
\]

Such a decomposition is not unique. The minimum number \( p \) for such a decomposition is called the (generalized) rank of \( \Lambda \). The evaluation of \( \Lambda \) is given by scalar products as follows:

\[
\Lambda(a_1, \ldots, a_r) = \sum_{i=1}^p \lambda_i \prod_{j=1}^r \langle v_i, a_j \rangle
\]

**Proof.** With that in mind we can start evaluating BS values for a symmetric \( r \)-form:

\[
\text{Sh}(e) = \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||\overline{S}|!}{(n+1)!} \left( \sum_{l,k} \binom{r}{l,k} (1 - (-1)^l) \Lambda \otimes M^{\otimes (r-m-k)} \otimes \left( \frac{\Delta_e}{2} \right)^{\otimes m} \otimes \left( \frac{\Delta_S - \Delta_{\overline{S}}}{2} \right)^{\otimes k} \right)
\]

The terms in the parenthesis can be rearranged as follows:

\[
A = x_S + b_S + e_S = \frac{x_S + x_e + x_S + b_S + b_e + b_S + x_S - b_S + x_e - b_e + b_S - x_S}{2}
\]

\[
A' = M - \frac{\Delta_e}{2} + \frac{\Delta_S - \Delta_{\overline{S}}}{2}
\]

Thus yielding:

\[
\text{Sh}(e) = \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||\overline{S}|!}{(n+1)!} \prod_{l,k} \left( \binom{r}{l,k} (1 - (-1)^l) \Lambda \otimes M^{\otimes (r-m-k)} \otimes \left( \frac{\Delta_e}{2} \right)^{\otimes m} \otimes \left( \frac{\Delta_S - \Delta_{\overline{S}}}{2} \right)^{\otimes k} \right)
\]

The above summands are zero if \( l \) is even or \( k \) is odd.

The preceding discussion motivates us to evaluate the term

\[
\sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||\overline{S}|!}{(n+1)!} \left( \frac{\Delta_S - \Delta_{\overline{S}}}{2} \right)^{\otimes k}
\]

\[
= \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S||\overline{S}|!}{(n+1)!} \sum_{\beta \in [1,n]^k} \epsilon(S)(\alpha^{\otimes k})_{\beta} e^{\otimes \beta}.
\]
Where $\alpha_i = \frac{1}{2}(\Delta_S - \Delta_{\overline{S}})$, if $i \in S$ and $\alpha_i = -\frac{1}{2}(\Delta_S - \Delta_{\overline{S}})$, if $i \in \overline{S}$. Let $\epsilon_i$ denote a sign that is $+1$ if $i \in S$ and $-1$ otherwise. Thus

$$
\sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S|!|\overline{S}|!}{(n+1)!} \sum_{\beta \in [1,n]^k} \epsilon(S)(\alpha^{\otimes k})_\beta e^{\otimes \beta}
= \sum_{\beta \in [1,n]^k} (\alpha^{\otimes k})_\beta e^{\otimes \beta} \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S|!|\overline{S}|!}{(n+1)!} \prod_{\beta_i \in \overline{S}} (-1).
$$

In the following we denote by $L$, $2l = |L|$ the set indices with odd exponents for a given monomial, whose number must be even. Let $o = 2l - \overline{S} = 2l - |S \cap L|$:

$$
\sum_{\beta \in [1,n]^k} (\alpha^{\otimes k})_\beta e^{\otimes \beta} \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S|!|\overline{S}|!}{(n+1)!} \prod_{\beta_i \in S} (-1)
= \sum_{\beta \in [1,n]^k} (\alpha^{\otimes k})_\beta e^{\otimes \beta} \sum_{o=0}^{2l} \sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S|!|\overline{S}|!}{(n+1)!} \prod_{\beta_i \in S} (-1)
$$

The indices of odd exponents distribute over $S$ and $\overline{S}$, where $o$ denotes the number of hits in $S$ and $\overline{S}$ in $\overline{S}$, respectively. The signature $(o, \overline{S})$, $o + \overline{S} = 2l$ completely determines the sign of the $\epsilon$ product. If $o$ is even, then it is positive, otherwise negative. Thus the sum becomes:

$$
\prod_{\beta_i \in \overline{S}} (-1) = (-1)^{2l-o}
$$

Since only those terms of the monomial matter, that have odd exponents, we get a combinatorial problem as follows:

Given a signature of $(o, \overline{S})$, how many ways are there to arrange $S$ and $\overline{S}$ to hit that signature?

There need to be at least $o$ elements in $S$, but also $\overline{S} = 2l - o$ in $\overline{S}$. The rest of the elements can be chosen freely.

$$
\binom{2l}{o} \binom{n-2l}{s-o}.
$$

This can be used to evaluate the inner sum now:

$$
\sum_{S \subseteq [n+1] \setminus \{e\}} \frac{|S|!|\overline{S}|!}{|S \cap L|=o} \frac{(n-2l-o)}{(n+1)!} (-1)^{2l-o}
= \sum_{s=o}^{n-2l-o} \frac{s!(n-s)!}{(n+1)!} \left( \frac{2l}{o} \right) \left( \frac{n-2l}{s-o} \right) (-1)^{2l-o}
= \left( \frac{2l}{o} \right) \left( \frac{n-2l}{n+1!} \right) (-1)^{2l-o} \sum_{s=o}^{n-2l-o} \frac{s!(n-s)!}{(s-o)! (n-s-(2l-o))!}
= \left( \frac{2l}{o} \right) \left( \frac{n-2l}{n+1!} \right) (-1)^{2l-o} \sum_{s=o}^{n-2l-o} s \cdots (s-o+1) \cdots (n-s) \cdots (n-(2l-o)+1) + 1
$$

We notice the polynomial in the sum would also be zero in the case of $0 \leq s \leq o-1$ and $n - (2l - o) + 1 \leq s \leq n$ and so we can extend the range of the sum:

$$
\left( \frac{2l}{o} \right) \left( \frac{n-2l}{n+1!} \right) (-1)^{2l-o} \sum_{s=0}^{n} s \cdots (s-o+1) \cdots (n-s) \cdots (n-(2l-o)+1) + 1.
$$

We claim now that the sum over the polynomial can always be solved and will yield a similar result depending on the parameters $o, 2l, s, k$. 
Definition 8 (Derived Sequences). Let \( v : \mathbb{Z} \to \mathbb{R} \) be a sequence. Then the following are derived sequences:

- The (first) difference sequence: \( \Delta v : \mathbb{Z} \to \mathbb{R}, s \mapsto (\Delta v)(s) := v(s) - v(s - 1) \);
- The \( n \)-th difference sequence is recursively defined: \( (\Delta^{n+1} v)(s) := (\Delta(\Delta^n v))(s) \);

With

\[
(\Delta u)(s) = s \cdots (s - o + 1) \\
v(s) = (n - s) \cdots (n - (2l - o) + 1 - s)
\]

We have:

1. \( (\Delta u)(s) \) has roots at \( 0, \ldots, o - 1 \),
2. \( v(s) \) has roots at \( n - (2l - o) + 1, \ldots, n \),
3. \( u \) could have the form \( u(s) = \frac{1}{o!} (s + 1) \cdots (s - o + 1) \) and has roots at \(-1, \ldots, o - 1 \),
4. \( \Delta v(s) = -(2l - o) \cdot (n - 1 - s) \cdots (n - (2l - o) + 1 - s) \) and has roots at \( n - (2l - o) + 1, \ldots, n - 1 \).

With the above convention we can formulate:

\[
\sum_{s=0}^{n} (\Delta u)(s)v(s) = \sum_{s=0}^{n} u(s)v(s) - \sum_{s=0}^{n} u(s-1)v(s) \\
= \sum_{s=0}^{n} u(s)v(s) - \sum_{s=0}^{n} u(s-1)(v(s) - v(s - 1)) - \sum_{s=0}^{n} u(s-1)v(s - 1) \\
= \sum_{s=0}^{n} u(s)v(s) - \sum_{s=0}^{n} u(s-1)v(s) - \sum_{s=0}^{n} u(s-1)\Delta v(s) \\
= u(n)v(n) - u(-1)v(-1) - \sum_{s=-1}^{n-1} u(s)\Delta v(s) \\
= - \sum_{s=-1}^{n-1} u(s)\Delta v(s + 1).
\]

More generally we can formulate for \( 1 \leq j \leq 2l - o \):

\[
\sum_{s=0}^{n} (\Delta^j u)(s)v(s) = - \sum_{s=-1}^{n-j} (\Delta^{j-1} u)(s)\Delta v(s) \\
= (-1)^j \sum_{s=-j}^{n-j} u(s)(\Delta^j v)(s + j).
\]

So to evaluate the sum:

\[
\sum_{s=0}^{n} s \cdots (s - o + 1) \cdot (n - s) \cdots (n - (2l - o) + 1 - s)
\]

we use the slightly different settings:

\[
(\Delta^{2l-o} u)(s) = s \cdots (s - o + 1) \\
v(s) = (n - s) \cdots (n - (2l - o) + 1 - s).
\]

The \( i \)-th integral of \( \Delta^{2l-o} u \) has the form:

\[
(\Delta^{2l-o+i} u)(s) = \frac{1}{(o+1) \cdots (o+i)} (s + i) \cdots (s - o + 1)
\]
having roots at \(-i, \ldots, o-1\). The \(i-th\) derivate of \(v\) has the form:

\[
(\Delta^i v)(s) = (-1)^i (2l - o) \cdots (2l - o - i + 1)(n - i - s) \cdots (n - (2l - o) + 1 - s)
\]

Using this we can show that:

\[
\sum_{s=0}^{n} (\Delta^2 l_o u)(s) v(s + 2l - o)
\]

\[
= (-1)^{2l-o} \sum_{s=0}^{n-2l-o} u(s) (\Delta^{2l-o} v)(s)
\]

\[
= (-1)^{2l-o} \sum_{s=-(2l-o)}^{n-(2l-o)} \frac{1}{(2l)!} (s + (2l - o)) \cdots (s - o + 1)(-1)^{(2l-o)(2l-o)}!
\]

\[
= \left(\frac{2l}{o}\right)^{-1} \sum_{s=-(2l-o)}^{n-(2l-o)} (s + (2l - o)) \cdots (s - o + 1)
\]

\[
= \left(\frac{2l}{o}\right)^{-1} \left[ \frac{1}{2l+1} (s + (2l - o) + 1) \cdots (s - o + 1) \right]^{n-(2l-o)}_{-(2l-o)-1}
\]

\[
= \frac{1}{2l+1} \left(\frac{2l}{o}\right)^{-1} (n + 1) \cdots (n - 2l + 1)
\]

Which means that

\[
\left(\frac{2l}{o}\right) \frac{(n - 2l)!}{(n + 1)!} \left(-1\right)^{2l-o} \sum_{s=0}^{n} s \cdots (s - o + 1) \cdot (n - s) \cdots (n - (2l - o) + 1 - s)
\]

\[
= \left(\frac{2l}{o}\right) \frac{(n - 2l)!}{(n + 1)!} \left(-1\right)^{2l-o} \frac{1}{2l+1} \left(\frac{2l}{o}\right)^{-1} (n + 1) \cdots (n - 2l + 1)
\]

\[
= \frac{(n - 2l)!}{(2l+1)(n + 1)!} (n + 1) \cdots (n - 2l + 1)(-1)^{2l-o}
\]

\[
= \frac{(-1)^{2l-o}}{2l+1}
\]

To evaluate the rest of the sums:

\[
\sum_{\beta \in [1, n]^r} (\alpha \otimes k)_{\beta} e \otimes \beta \sum_{s=0}^{2l+1} \frac{(-1)^{2l-o}}{2l+1} = \sum_{\beta \in [1, n]^r} \frac{1}{2l+1} (\alpha \otimes k)_{\beta} e \otimes \beta
\]

\[
(B4)
\]

For an efficient evaluation of the remaining sum we have to know how \(l\) depends on \(\beta\).

It is easy to see, that given such a decomposition, storage and evaluation of such multilinear forms is much faster. Combining all of our results leads to the following expression:

\[
\text{Sh}(e)
\]

\[
= 2 \sum_{j+m+k=r \atop 0 \leq j, l, k \leq r \atop l \text{ odd}, k \text{ even}} \binom{r}{j, m, k} \sum_{\gamma \in \mathbb{N}^n} \frac{1}{2l+1} \binom{k}{\gamma} \sum_{i=1}^{n} \langle v_i, M \rangle^j v_i, \frac{\Delta_i}{2} \rangle \sum_{h=1}^{n} (v_i, \alpha_h) \gamma_h
\]