CERN-LHCEFTWG-2025-001
CERN-LPCC-2025-003
LHCHWG-2025-050
MITP-25-066

Joint LHC EFT Working Group & LHC Higgs Working Group Note:

POPxf: An Exchange Format for Polynomial Observable Predictions

Ilaria Brivio[1] *(ed.), Ken Mimasu*[2] *(ed.), Peter Stangl*[3] *(ed.),*
Anke Biekötter[4]*, Ana R. Cueto Gómez*[5]*, Charlotte Knight*[6]*, Luca Mantani*[7]*, Eleonora Rossi*[8]*,*
Alejo N. Rossia[9,10]*, Aleks Smolkovič*[11]

[1] Dipartimento di Fisica e Astronomia, Università di Bologna and INFN Bologna, Italy
[2] School of Physics and Astronomy, University of Southampton, Highfield, Southampton SO17 1BJ, United Kingdom
[3] Institute of Physics, Johannes Gutenberg University Mainz, Staudingerweg 7, 55128 Mainz, Germany
[4] Karlsruhe Institute of Technology, Institute for Theoretical Particle Physics, Wolfgang-Gaede-Straße 1, 76131 Karlsruhe, Germany
[5] Universidad Autónoma de Madrid, Campus Universitario de Cantoblanco, Madrid, Spain
[6] Imperial College, London, United Kingdom
[7] Instituto de Fisica Corpuscular (IFIC), Universidad de Valencia-CSIC, E-46980 Valencia, Spain
[8] University of Oxford, Oxford, United Kingdom
[9] Dipartimento di Fisica e Astronomia "G. Galilei", Università di Padova, Via F. Marzolo 8, I-35131, Padova, Italy
[10] Istituto Nazionale di Fisica Nucleare, Sezione di Padova, Via F. Marzolo 8, I-35131, Padova, Italy
[11] Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

Abstract
We introduce the Polynomial Observable Prediction Exchange Format, POPxf, a structured, machine-readable data format for the publication and exchange of semi-analytical theoretical predictions in high energy physics. The format is designed to encode observables that can be expressed in terms of polynomials in model parameters, with particular emphasis on Effective Field Theory applications. All relevant assumptions and metadata are recorded explicitly, and the treatment of uncertainties and correlations is flexible enough to capture parameter-dependent effects. The format aims to improve reproducibility, facilitate global fits and reinterpretations, and streamline the use of theoretical predictions across the particle physics community.

Keywords
Data format, Theoretical predictions, Effective Field Theory

Contents

1 Introduction

Theoretical predictions play a central role in particle physics phenomenology, allowing the interpretation of experimental data in terms of the underlying model parameters. In many cases, such predictions can be expressed in terms of polynomials in these parameters. A prominent example is observables in an Effective Field Theory (EFT), where they depend on a set of

Wilson coefficients. In particular, the amplitudes in an EFT are often linear in the Wilson coefficients, such that the associated observables, depending on squared amplitudes, are given in terms of quadratic polynomials in the Wilson coefficients. This structure arises generically in analyses within the Standard Model Effective Field Theory (SMEFT) and the Weak Effective Theory (WET) truncated at dimension six. Other examples beyond EFT include observables in flavour physics that depend on hadronic form factors, which enter amplitudes linearly and are themselves typically expressed using polynomial parameterisations.

Despite the ubiquity of such polynomial expressions, they are rarely published in their entirety and often remain hidden in various public and private codebases. Even when predictions based on polynomial expressions are made accessible, this is often done using varying conventions, with insufficient metadata, and/or in formats that are not machine-readable. Such practices complicate reproducibility, hinder validation and cross-comparison, and lead to significant duplication of effort across the community. A standardised, machine-readable data format for sharing such predictions would address many of these issues. It would facilitate the reuse of theoretical predictions in global fits, reinterpretations, and future measurements. It would also help document and preserve the assumptions – such as EFT basis choices, renormalisation scales, input parameter values, or normalisation conventions – that are crucial for the correct use and interpretation of the results.

This note proposes the Polynomial Observable Prediction Exchange Format, `POPxf`, a structured data format designed to encode polynomial parameterisations of observables as functions of model parameters, with a focus on EFT applications, while remaining general enough to support other use cases. The format supports arbitrary polynomial degree and is sufficiently general to describe both single-valued and multi-bin observables, entire sectors of related observables, as well as observables defined through functions of multiple polynomial components.

Returning to the example of EFT predictions, where observables depend on a set of beyond-the-standard-model (BSM) Wilson coefficients, C_i, the scattering amplitude for a given process often takes the form

$$\mathcal{A} = \mathcal{A}_{\rm SM} + \sum_i C_i \, \mathcal{A}_i, \tag{1}$$

so that the corresponding observable O, typically proportional to $|\mathcal{A}|^2$, becomes a quadratic polynomial in the C_i:

$$O = O_{\rm SM} + \sum_i C_i \, O_i^{\rm int} + \sum_{i \leq j} C_i C_j \, O_{ij}^{\rm quad}. \tag{2}$$

Here, $O_{\rm SM}$ is the Standard Model (SM) prediction, $O_i^{\rm int}$ encodes the interference between SM and BSM amplitudes, and $O_{ij}^{\rm quad}$ captures the pure BSM-squared contributions. This structure is general for dimension-six SMEFT and WET analyses, for which the expansion of the amplitude is truncated at $\mathcal{O}(1/\Lambda^2)$, where Λ is the cutoff parameter of the EFT. While quadratic polynomial structures are common, higher-order terms can arise in EFTs when including dimension-eight operators or computing beyond leading order. The proposed data format supports arbitrary polynomial degree to remain general and extensible. In general, model parameters such as Wilson coefficients can be complex. In such cases, observables may depend separately on the real and imaginary parts of the parameters. For example, a quadratic monomial $C_i C_j$ must be interpreted as a combination of terms such as $\mathrm{Re}(C_i)\mathrm{Re}(C_j)$, $\mathrm{Re}(C_i)\mathrm{Im}(C_j)$, etc. The format explicitly supports such terms via a real-imaginary decomposition.

Although we use the term 'observable' to refer to a quantity whose prediction is a single real number, in many cases the objects of interest are vector-valued quantities, i.e. sets of observables in the aforementioned sense. Examples include binned distributions and angular observables. In such cases, each bin or component is treated as a separate observable, and the predictions of the entire set of observables are encoded via vector-valued polynomial coefficients. The data format accommodates this naturally via array structures compatible with numerical data structures such as `numpy` or `Mathematica` arrays.

Some observables are not themselves polynomials in the parameters but are functions of one or more polynomial expressions. Examples include ratios of decay widths or angular observables formed from linear combinations of angular coefficients. The data format supports such derived observables by allowing them to be defined in terms of functional combinations of underlying polynomials.

Finally, the treatment of uncertainties and correlations in this format assumes that the observable is expressed as a polynomial in the model parameters. In many cases, the observable itself is already a polynomial – for example, cross sections or partial decay widths derived from squared amplitudes. In the case where the observable is defined as a function of one or more polynomials, it is expanded in the model parameters to yield a single polynomial expression, typically truncated at second order. This expansion defines the level at which parameter-dependent uncertainties and correlations are interpreted, and provides a consistent and efficient framework for uncertainty propagation [1].

The proposed data format is based on the `JSON` standard, a lightweight, text-based data format that is both human-readable and natively supported in most programming languages and computer algebra systems, particularly in `python` and `Mathematica`, which are used extensively in the particle physics community. It provides a simple and transparent way to encode central values, uncertainties and correlations as well as all relevant metadata in a structured, hierarchical format. Specific design goals include:

- general applicability to any observable that can be expressed in terms of a polynomial in model parameters, including but not limited to EFT Wilson coefficients;
- support for arbitrary polynomial degree, enabling the inclusion of higher-order terms such as those arising from dimension-eight EFT operators;
- support for sectors of related observables, such as binned distributions, angular observables, or correlated decay channels;
- support for observables defined as functions of one or multiple polynomial parameterisations, such as normalised distributions or ratios of decay widths;
- explicit encoding of all assumptions, such as parameter basis definitions, input parameter values, renormalisation scale, and operator normalisations, to ensure reproducibility;
- consistent treatment of theoretical uncertainties from various sources, including support for parameter-dependent uncertainties;
- support for correlated uncertainties between observables, including the possibility of parameter-dependent correlations.

In the following sections, we will present the proposed structure of this data format in detail and illustrate its use with several concrete examples. Our aim is to provide a general framework that can be readily adopted by both theorists and experimentalists for the publication, exchange,

and application of theoretical predictions, with a special emphasis on EFT-based parametrisations. Section 2 outlines the general formalism for data files describing predictions that can be expressed as a function of polynomials in the model parameters, as well as the associated uncertainties. Section 3 presents an overview of the `JSON` structure used to encode the polynomial data, and introduces the two intended modes of use for the data format: *single-polynomial* mode and *function-of-polynomials* mode. Section 4 provides a detailed specification of the fields in the `POPxf` format. Section 5 describes the structure of the separate `POPxf` correlation files that are used to store correlation coefficients of correlated observables. Finally, Section 6 briefly summarises and concludes.

The concrete specification of the `POPxf` data format in terms of `JSON` schemas is hosted in a public repository at https://github.com/pop-xf, which also includes a lightweight validator and associated command line tool as well as a collection of example files.

For impatient readers who are keen to get started immediately, we recommend taking a brief look at Section 3, Fig. 1, and the explicit examples given in Section A.

2 General Formalism

2.1 Data Files

We consider polynomial observable predictions defined within a set of N data files, which are labelled by an index $n \in [1, N]$. The terms *observable* and *polynomial* refer to real-valued scalar quantities. The data file with index n defines:

- A basis of $S^{(n)}$ *parameters* $C_s^{(n)}$ on which the predictions depend (in the case of EFT predictions, the parameters are the Wilson coefficients), and which are labelled by indices $s \in [1, S^{(n)}]$. Each parameter can be real or complex. Denoting the number of real and complex parameters by $S_{\mathbb{R}}^{(n)}$ and $S_{\mathbb{C}}^{(n)}$, respectively, with $S^{(n)} = S_{\mathbb{R}}^{(n)} + S_{\mathbb{C}}^{(n)}$, we denote the real-valued, real and imaginary parts of the parameters as $\hat{C}_r^{(n)}$, which are labelled by indices $r \in [1, R^{(n)}]$ with $R^{(n)} = S_{\mathbb{R}}^{(n)} + 2\, S_{\mathbb{C}}^{(n)}$. We group the $\hat{C}_r^{(n)}$ into the real-valued vector $\vec{C}^{(n)}$. Note that this vector is in general *not* a vector of the (potentially complex) parameters $C_s^{(n)}$, but a vector of their real and imaginary parts $\hat{C}_r^{(n)}$, i.e. the components of the vector are given by $[\vec{C}^{(n)}]_r = \hat{C}_r^{(n)}$.
- A set of $K^{(n)}$ *polynomials* $P_k^{(n)}$, which are labelled by indices $k \in [1, K^{(n)}]$. The polynomials are specified in terms of *polynomial coefficients* $\vec{p}_k^{(n)}$ (see below).
- A set of $M^{(n)}$ *observable predictions* $O_m^{(n)}$, which are labelled by indices $m \in [1, M^{(n)}]$. The observable predictions are defined in terms of $M^{(n)}$ functions of the polynomials $P_k^{(n)}$, which we denote as *observable expressions* $E_m^{(n)}$,

$$O_m^{(n)} = E_m^{(n)} \left(P_1^{(n)}, P_2^{(n)}, ..., P_{K^{(n)}}^{(n)} \right) . \tag{3}$$

  In many practical examples, all observables defined in a given data file are themselves polynomials, such that the number of polynomials and observables coincide, $K^{(n)} = M^{(n)}$, and each observable expression $E_m^{(n)}$ is a trivial identity function of a single polynomial labelled by $k = m$,

$$O_m^{(n)} = E_m^{(n)} \left(P_m^{(n)} \right) = P_m^{(n)} . \tag{4}$$

In this special case, no observable expressions have to be specified, and the observable predictions are directly given in terms of the polynomial coefficient $\vec{p}_m^{(n)}$, which are then denoted as *observable coefficients* $\vec{o}_m^{(n)} = \vec{p}_m^{(n)}$. But in the general case, observables are given by non-trivial functions of multiple polynomials.

2.2 Polynomials

Each polynomial $P_k^{(n)}$ can be expressed as a scalar product of two vectors: a vector of *polynomial coefficients* $\vec{p}_k^{(n)}$, and a vector of *parameter monomials* $\vec{V}^{(n)}$,

$$P_k^{(n)} = \vec{p}_k^{(n)} \cdot \vec{V}^{(n)} \,. \tag{5}$$

For polynomials of degree d, the vector $\vec{V}^{(n)}$ is given by the set of all monomials up to degree d formed from the components of $\vec{C}^{(n)}$.

In the following, we will focus on second-order polynomials[1], for which $\vec{V}^{(n)}$ takes the form

$$\vec{V}^{(n)} = \begin{pmatrix} 1 \\ \vec{C}^{(n)} \\ \mathrm{vech}(\vec{C}^{(n)} \otimes \vec{C}^{(n)}) \end{pmatrix} , \tag{6}$$

where vech denotes half-vectorization,[2] and we have split $\vec{V}^{(n)}$ into three parts corresponding to the parameter monomials of degrees zero, one, and two. We denote the components of the vector $\vec{V}^{(n)}$ as $[\vec{V}^{(n)}]_\alpha$ labelled by indices $\alpha \in [0, A^{(n)}]$, where in the case of second-degree polynomials $A^{(n)}$ is given by

$$A^{(n)} = R^{(n)} + R^{(n)}(R^{(n)}+1)/2 = R^{(n)}(R^{(n)}+3)/2 \,, \tag{7}$$

such that

$$[\vec{V}^{(n)}]_\alpha = \begin{cases} 1 & \text{if } \alpha = 0 \\ [\vec{C}^{(n)}]_\alpha & \text{if } \alpha \in [1, R^{(n)}] \\ [\mathrm{vech}(\vec{C}^{(n)} \otimes \vec{C}^{(n)})]_{\alpha - R^{(n)}} & \text{if } \alpha \in [R^{(n)}+1, A^{(n)}] \end{cases} . \tag{8}$$

A vector of polynomial coefficients can then be written as

$$\vec{p}_k^{(n)} = \begin{pmatrix} a_k^{(n)} \\ \vec{b}_k^{(n)} \\ \vec{c}_k^{(n)} \end{pmatrix} , \tag{9}$$

[1] In principle, the format can support higher degree polynomials, with a corresponding generalisation of the equations that follow. For example the vectors $\vec{V}^{(n)}$ can be extended by higher degree monomials of the $\vec{C}^{(n)}$ parameters, with a corresponding extension of the $\vec{p}^{\,(n)}$ coefficients.

[2] The half-vectorization of a symmetric $n \times n$ matrix A is defined as the column vector with $n(n+1)/2$ components obtained by stacking the lower triangular columns of A on top of each other:

$$\mathrm{vech}(A) = (A_{11}, A_{21}, \ldots, A_{n1}, A_{22}, A_{32}, \ldots, A_{n2}, \ldots, A_{nn})^T$$

Applying vech to the outer product of a vector $\vec{v}$ with itself, $\vec{v} \otimes \vec{v}$, yields a vector of all unique degree-two monomials formed from the components of $\vec{v}$,

$$\mathrm{vech}(\vec{v} \otimes \vec{v}) = (v_1^2, v_1 v_2, v_1 v_3, ..., v_1 v_n, v_2^2, v_2 v_3, ..., v_2 v_n, ..., v_{n-1}^2, v_{n-1} v_n, v_n^2)^T \,.$$

such that the polynomials can be given by

$$P_k^{(n)} = \vec{p}_k^{\,(n)} \cdot \vec{V}^{(n)} = a_k^{(n)} + \vec{b}_k^{\,(n)} \cdot \vec{C}^{(n)} + \vec{c}_k^{\,(n)} \cdot \mathrm{vech}(\vec{C}^{(n)} \otimes \vec{C}^{(n)}) \,, \tag{10}$$

i.e. $a_k^{(n)}$, $\vec{b}_k^{\,(n)}$, and $\vec{c}_k^{\,(n)}$ contain the polynomial coefficients that multiply parameter monomials of degrees zero, one, and two, respectively. In other words, $a_k^{(n)}$ is the parameter-independent constant term, while $\vec{b}_k^{\,(n)}$ and $\vec{c}_k^{\,(n)}$ contain the coefficients of the terms linear and quadratic in the parameters.

The numerical values of the polynomial coefficients $\vec{p}_k^{\,(n)}$ uniquely define the polynomials $P_k^{(n)}$ as functions of the parameters $\vec{C}^{(n)}$.

2.3 Observable Predictions

We distinguish between two modes of use for specifying observable predictions:

1. The *function-of-polynomials* (FOP) mode defines observable predictions $O_m^{(n)}$ in the general case, in which they are given by non-trivial observable expressions $E_m^{(n)}$, which are arbitrary functions of polynomials $P_k^{(n)}$ (note that both the $O_m^{(n)}$ and the $P_k^{(n)}$ are real-valued scalar quantities),

$$O_m^{(n)}\big|_{\mathrm{FOP}} = E_m^{(n)} \left(P_1^{(n)}, P_2^{(n)}, ..., P_{K^{(n)}}^{(n)} \right) \,. \tag{11}$$

2. The *single-polynomial* (SP) mode can be used if all observables in a given data file are themselves polynomials, i.e. if their observable expressions $E_m^{(n)}$ are all trivial identity functions. In this case, no observable expressions have to be specified, and the observable predictions have the same structure as the polynomials defined above. In SP mode, we express each observable prediction $O_m^{(n)}$ directly as a scalar product of *observable coefficients* $\vec{o}_m^{\,(n)}$, and the vector of parameter monomials $\vec{V}^{(n)}$ defined in Section 2.2,

$$O_m^{(n)}\big|_{\mathrm{SP}} = \vec{o}_m^{\,(n)} \cdot \vec{V}^{(n)} \,. \tag{12}$$

   Focusing again on second-order polynomials, the observable coefficients can be written as

$$\vec{o}_m^{\,(n)} = \begin{pmatrix} a_m^{(n)} \\ \vec{b}_m^{\,(n)} \\ \vec{c}_m^{\,(n)} \end{pmatrix} \,, \tag{13}$$

   such that the observable predictions are given by

$$O_m^{(n)}\big|_{\mathrm{SP}} = \vec{o}_m^{\,(n)} \cdot \vec{V}^{(n)} = a_m^{(n)} + \vec{b}_m^{\,(n)} \cdot \vec{C}^{(n)} + \vec{c}_m^{\,(n)} \cdot \mathrm{vech}(\vec{C}^{(n)} \otimes \vec{C}^{(n)}) \,, \tag{14}$$

   analogous to the polynomials in Eq. (10).

While in SP mode an observable prediction is exactly given by a single polynomial, in FOP mode it can be approximated by a single polynomial if all parameters are small, $\vec{C}^{(n)} \ll 1$ and Taylor expansion around $\vec{C}^{(n)} = \vec{0}$ is possible. Such an approximation will be needed for our treatment of parameter-dependent uncertainties discussed in Section 2.4, which relies on each

observable being expressed as a single polynomial. If all parameters are small, we express them as a product of a small quantity $\epsilon \ll 1$ and order-one parameters $\vec{\mathcal{C}}^{(n)}$, i.e.

$$\vec{C}^{(n)} = \epsilon\, \vec{\mathcal{C}}^{(n)} \qquad \text{with} \qquad \epsilon \ll 1 \qquad \text{and} \qquad \vec{\mathcal{C}}^{(n)} = \mathcal{O}(1)\,. \tag{15}$$

For second-order polynomials, the vector of parameter monomials, $\vec{V}^{(n)}$, thus contains terms up to $\mathcal{O}(\epsilon^2)$, and the observable predictions in the general case can be expanded as

$$O_m^{(n)}\big|_{\text{FOP}} = \vec{o}_m^{\,(n)} \cdot \vec{V}^{(n)} + \mathcal{O}(\epsilon^3) = a_m'^{(n)} + \epsilon \left(\vec{b}_m'^{(n)} \cdot \vec{\mathcal{C}}^{(n)}\right) + \epsilon^2 \left(\vec{c}_m'^{(n)} \cdot \text{vech}(\vec{\mathcal{C}}^{(n)} \otimes \vec{\mathcal{C}}^{(n)})\right) + \mathcal{O}(\epsilon^3)\,. \tag{16}$$

This approximated observable prediction has exactly the same form as the SP mode prediction, and we have defined the corresponding observable coefficients $\vec{o}_m^{\,(n)}$ in terms of the primed quantities $a_m'^{(n)}$, $\vec{b}_m'^{(n)}$, and $\vec{c}_m'^{(n)}$. They can be obtained from the observable expressions $E_m^{(n)}$ and the polynomial coefficients $a_k^{(n)}$, $\vec{b}_k^{(n)}$, and $\vec{c}_k^{\,(n)}$ through [1]

$$\begin{aligned}
a_m'^{(n)} &= [G_m^{(n)}]\,, \\
\vec{b}_m'^{(n)} &= \sum_{k_1} [G_m^{(n)}]_{k_1}\, \vec{b}_{k_1}^{(n)}\,, \\
\vec{c}_m'^{(n)} &= \sum_{k_1} [G_m^{(n)}]_{k_1}\, \vec{c}_{k_1}^{\,(n)} + \frac{1}{2} \sum_{k_1,k_2} [G_m^{(n)}]_{k_1 k_2}\, D^T_{R^{(n)}}\, \text{vec}(\vec{b}_{k_1}^{(n)} \otimes \vec{b}_{k_2}^{(n)})\,,
\end{aligned} \tag{17}$$

where $D^T_{R^{(n)}}$ is the transpose of the duplication matrix[3] with $R^{(n)}$ the number of parameters, and the coefficients $[G_m^{(n)}]_{k_1 k_2 \ldots k_\ell}$ of the multivariate Taylor expansion are defined as the ℓ-th derivatives of the observable expressions $E_m^{(n)}$ evaluated at $\epsilon = 0$, i.e. for $P_k^{(n)} = a_k^{(n)}$:

$$[G_m^{(n)}]_{k_1 k_2 \ldots k_\ell} = \left. \frac{\partial^\ell E_m^{(n)} \left(P_1^{(n)}, P_2^{(n)}, ..., P^{(n)}_{K^{(n)}}\right)}{\partial_{P_{k_1}^{(n)}} \partial_{P_{k_2}^{(n)}} \ldots \partial_{P_\ell^{(n)}}} \right|_{P_1^{(n)}=a_1^{(n)},\; P_2^{(n)}=a_2^{(n)},\ldots,\; P^{(n)}_{K^{(n)}}=a^{(n)}_{K^{(n)}}}\,, \tag{19}$$

such that the $\ell = 0$ coefficient is

$$[G_m^{(n)}] = E_m^{(n)} \left(a_1^{(n)}, a_2^{(n)}, ..., a^{(n)}_{K^{(n)}}\right)\,. \tag{20}$$

In the case where the $E_m^{(n)}$ are identity functions, $E_m^{(n)}\left(P_m^{(n)}\right) = P_m^{(n)}$, we recover the observable prediction in SP mode, with $[G_m^{(n)}] = a_m^{(n)}$, $[G_m^{(n)}]_{k_1} = \delta_{m,k_1}$, and $[G_m^{(n)}]_{k_1 \ldots k_\ell} = 0$ for $\ell \geq 2$.

2.4 Observable Uncertainties and Correlations

We consider Gaussian uncertainties in observable predictions that can be both correlated and depend on the $\vec{C}^{(n)}$ parameters. In SP mode – when the observables are themselves polynomials

[3] The duplication matrix D_n is the unique $n^2 \times n(n+1)/2$ matrix that for any symmetric $n \times n$ matrix A relates its vectorization $\text{vec}(A)$ and its half-vectorization $\text{vech}(A)$ by [2]

$$\text{vec}(A) = D_n\, \text{vech}(A)\,. \tag{18}$$

– the uncertainties and correlations of the observable predictions $O_m^{(n)}|_{\rm SP}$ can be expressed analytically in terms of the uncertainties and correlations of their observable coefficients $\vec{o}_m^{\,(n)}$ (see below). In FOP mode – when the observables are given by arbitrary functions of polynomials – for the purposes of uncertainties we only consider the case of small parameters, which allows us to expand the observable predictions $O_m^{(n)}|_{\rm FOP}$ as in Eq. (16) with approximate observable coefficients $\vec{o}_m^{\,(n)}$ given by Eq. (17). This allows us to treat uncertainties and correlations in the same way for both SP and FOP observables, defined through the uncertainties and correlations of the exact (in SP mode) or approximate (in FOP mode) observable coefficients $\vec{o}_m^{\,(n)}$.

In the following, we express the uncertainties of the observable coefficients $\vec{o}_m^{\,(n)}$ by the vector

$$\vec{\sigma}_m^{\,(n)} \tag{21}$$

and the correlations between observable coefficients $\vec{o}_m^{\,(n)}$ (of observable m in data file n) and $\vec{o}_{m'}^{\,(n')}$ (of observable m' in data file n') by the correlation matrix

$$\rho_{mm'}^{(nn')} \,. \tag{22}$$

In addition, we denote the components of the vector $\vec{\sigma}_m^{\,(n)}$ as

$$[\vec{\sigma}_m^{\,(n)}]_\alpha \tag{23}$$

and the components of the matrix $\rho_{mm'}^{(nn')}$ as

$$[\rho_{mm'}^{(nn')}]_{\alpha\alpha'} \tag{24}$$

with $\alpha \in [0, A^{(n)}]$ and $\alpha' \in [0, A^{(n')}]$, where, as in Eq. (7), for second-order polynomials $A^{(n)} = R^{(n)}(R^{(n)}+3)/2$ and $A^{n'} = R^{(n')}(R^{(n')}+3)/2$.

The correlation matrix $\rho_{mm'}^{(nn')}$ and the uncertainties $\vec{\sigma}_m^{\,(n)}$ and $\vec{\sigma}_{m'}^{\,(n')}$ can be combined into a covariance matrix for the observable coefficients $\vec{o}_m^{\,(n)}$ and $\vec{o}_{m'}^{\,(n')}$, which we denote as

$$\tilde{\Sigma}_{mm'}^{(nn')} \,. \tag{25}$$

The components of this matrix are given by

$$[\tilde{\Sigma}_{mm'}^{(nn')}]_{\alpha\alpha'} = [\vec{\sigma}_m^{\,(n)}]_\alpha \, [\rho_{mm'}^{(nn')}]_{\alpha\alpha'} \, [\vec{\sigma}_{m'}^{\,(n')}]_{\alpha'} \,, \tag{26}$$

i.e. the covariance matrix for the observable coefficients can be expressed in terms of the observable coefficients' uncertainties and correlations.

We allow different sources of uncertainties and correlations to be defined separately (e.g. from Monte Carlo statistics, remaining scale-dependence, parton distribution functions, etc.). In this case, separate uncertainties

$$[\vec{\sigma}_m^{\,(n)}]_\alpha^{(\text{source})} \tag{27}$$

and correlations

$$[\rho_{mm'}^{(nn')}]_{\alpha\alpha'}^{(\text{source})} \tag{28}$$

are defined for each source. The components of the total covariance matrix for the observable coefficients are then obtained by summing the individual covariance matrices for each source,

$$[\tilde{\Sigma}^{(nn')}_{mm'}]_{\alpha\alpha'} = \sum_{\text{(source)}} [\tilde{\Sigma}^{(nn')}_{mm'}]^{\text{(source)}}_{\alpha\alpha'} = \sum_{\text{(source)}} [\vec{\sigma}^{\,(n)}_{m}]^{\text{(source)}}_{\alpha} \, [\rho^{(nn')}_{mm'}]^{\text{(source)}}_{\alpha\alpha'} \, [\vec{\sigma}^{\,(n')}_{m'}]^{\text{(source)}}_{\alpha'} \,. \tag{29}$$

The uncertainties and correlations of the observable predictions $O^{(n)}_m$ and $O^{(n')}_{m'}$ can be expressed in terms of a covariance matrix of the observables, Σ, which can be written as a block matrix, with the block $\Sigma^{(nn')}$ corresponding to the covariances between data files n and n',

$$\Sigma = \begin{pmatrix} \Sigma^{(11)} & \Sigma^{(12)} & \dots & \Sigma^{(1N)} \\ \Sigma^{(21)} & \Sigma^{(22)} & \dots & \Sigma^{(2N)} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma^{(N1)} & \Sigma^{(N2)} & \dots & \Sigma^{(NN)} \end{pmatrix} \,. \tag{30}$$

The components of the block $\Sigma^{(nn')}$ can be labelled by indices m and m' and correspond to the covariances of observable predictions $O^{(n)}_m$ and $O^{(n')}_{m'}$ and we denote them as

$$[\Sigma^{(nn')}]_{mm'} \,. \tag{31}$$

In case of a single data file, $N = 1$, the components of the full covariance matrix of the observable predictions $O^{(1)}_m$ and $O^{(1)}_{m'}$ are simply

$$\Sigma_{mm'} = [\Sigma^{(11)}]_{mm'} \,. \tag{32}$$

2.4.1 Parameter-independent Uncertainties and Correlations

When considering uncertainties and correlations of observable predictions, it is common practice to make the assumption that their parameter dependence can be neglected, and that they are well approximated by the uncertainties and correlations of the parameter-independent constant term. This is in particular the case in EFT applications where the parameters are new physics Wilson coefficients. In this case, the constant term is nothing but the SM prediction, and considering only the SM uncertainties and correlations is often an excellent approximation.

Neglecting the parameter-dependence, the covariances of observable predictions $O^{(n)}_m$ and $O^{(n')}_{m'}$ are given by

$$[\Sigma^{(nn')}]_{mm'} = [\tilde{\Sigma}^{(nn')}_{mm'}]_{00} \tag{33}$$

where $[\tilde{\Sigma}^{(nn')}_{mm'}]_{00}$ are the covariances of the constant terms in the observable predictions $O^{(n)}_m$ and $O^{(n')}_{m'}$ (the SM covariances in new physics EFT applications), defined by Eq. (26) in case of a single source of uncertainties and Eq. (29) in case of multiple sources.

2.4.2 Parameter-dependent Uncertainties and Correlations

If the parameter-dependence of the uncertainties and correlations cannot be neglected, it is still possible to express the covariances $[\Sigma^{(nn')}]_{mm'}$ of observable predictions $O^{(n)}_m$ and $O^{(n')}_{m'}$ in terms

of the observable coefficients' covariances $[\tilde{\Sigma}^{(nn')}_{mm'}]_{\alpha\alpha'}$ and the parameter monomials $[\vec{V}^{(n)}]_\alpha$ and $[\vec{V}^{(n')}]_{\alpha'}$, as discussed in [1],

$$[\Sigma^{(nn')}]_{mm'} = \sum_{\alpha,\alpha'} [\vec{V}^{(n)}]_\alpha \, [\tilde{\Sigma}^{(nn')}_{mm'}]_{\alpha\alpha'} \, [\vec{V}^{(n')}]_{\alpha'} \,, \tag{34}$$

where $[\tilde{\Sigma}^{(nn')}_{mm'}]_{\alpha\alpha'}$ is defined by Eq. (26) in case of a single source of uncertainties and Eq. (29) in case of multiple sources.

3 Overview of the `POPxf` `JSON` Format

This section provides a technical overview of the `JSON` structure used to encode theoretical predictions of observables that can be expressed as polynomials in model parameters. It introduces the overall layout and key structural concepts, including the separation between contextual metadata and numerical prediction data, and the distinction between single-polynomial and function-of-polynomials use cases. Figure 1 shows a graphical representation of the fields in the `POPxf` data structure.

3.1 Top-Level Structure

Predictions of a set of observables are encoded as a single `JSON` object containing exactly three required top-level fields:

- **`$schema`**: Keyword that declares the `JSON` schema specification used by the file. This allows identifying a given `JSON` file as a `POPxf` file and validating its structure. The value must be `"https://json.schemastore.org/popxf-1.0.json"` for the first version of this format and the version number will be incremented if the format is extended in the future;
- **`metadata`**: contextual and structural information required to interpret and reproduce the prediction;
- **`data`**: numerical information representing observables in terms of their polynomial coefficients and uncertainties on these.

No additional top-level keys are defined or permitted in the current schema.

3.2 Two Modes of Use

As introduced in Section 2.3, the format distinguishes between two structurally distinct ways to define an observable prediction.

1. Function-of-polynomials mode.

The observables are defined as functions of one or more named polynomials. This most general case allows for non-polynomial structures such as ratios, products, or square roots of polynomial components. In this case, the following fields are used:

- **`metadata.polynomial_names`** (***required***): array of polynomial names;

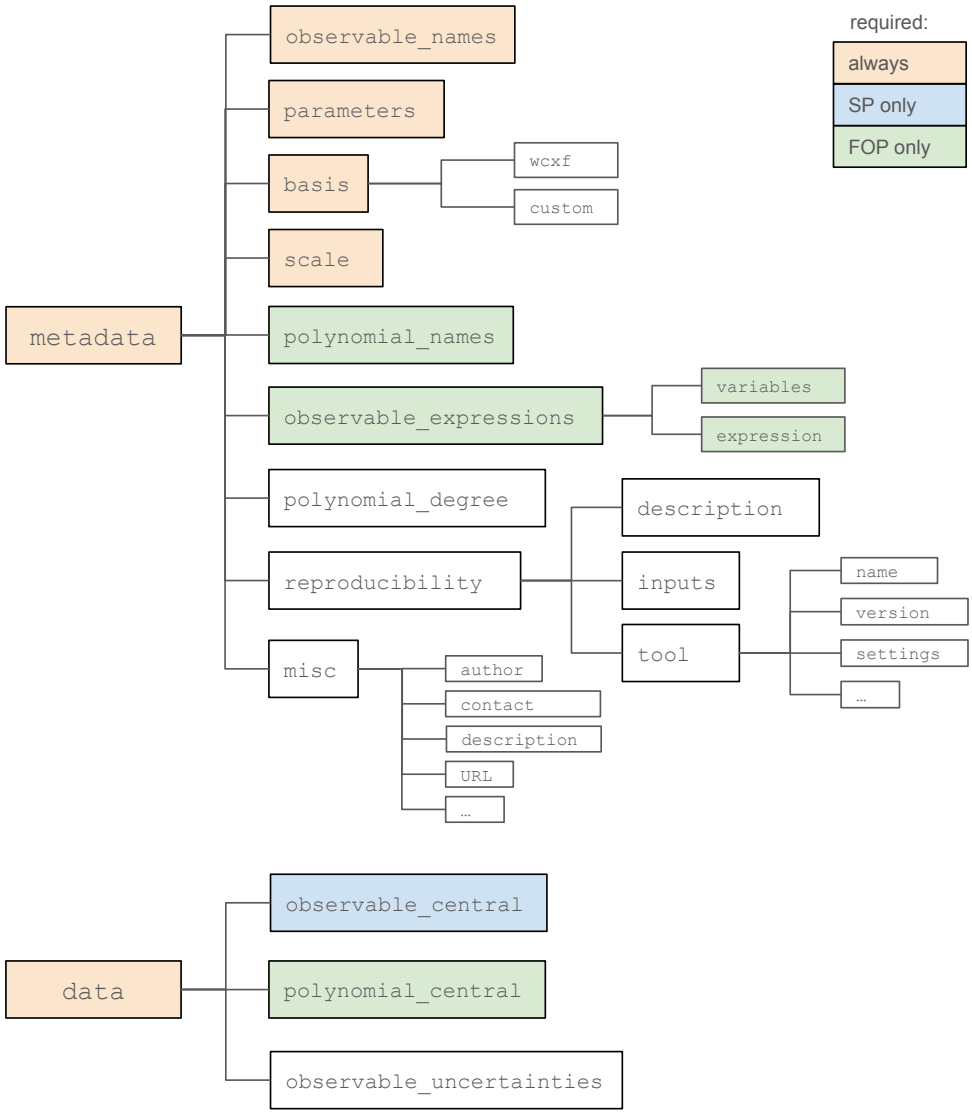

Fig. 1: Graphical representation of the `JSON` structure of the `POPxf` data format for the two top-level fields, `metadata` and `data`. Coloured boxes indicate required fields (blue and green for SP and FOP modes, respectively, orange for both).

- **`metadata.observable_expressions`** (***required***): Python-like expressions defining how each observable is computed from the named polynomials;
- **`data.polynomial_central`** (***required***): central values of polynomial coefficients, $\vec{p}_k$, for each named polynomial;
- **`data.observable_uncertainties`** (***optional***): uncertainties, $\vec{\sigma}_m$, on the polynomial coefficients of each observable after expanding it in the model parameters (see Section 2.4);
- **`data.observable_central`** (***optional***): central values, $\vec{o}_m$, of polynomial coefficients for each observable after expanding it in the model parameters. In FOP mode, this field is optional and is not intended to be used to compute the actual central values of the observables. It may be included to allow numerical comparison between the full observable expression and its expansion in the model parameters, or to enable switching between function-of-polynomials mode and single-polynomial mode.

The presence of any of the three required fields of function-of-polynomials mode will assume the use of this mode such that the other two are also required.

2. Single-polynomial mode.

Every observable is directly defined by a single polynomial in the model parameters. In this special but commonly-used case, the following fields are used:

- **`data.observable_central`** (***required***): central values of the observable coefficients, $\vec{o}_m = \vec{p}_m$, the polynomial coefficients of the observable;
- **`data.observable_uncertainties`** (***optional***): uncertainties on these observable coefficients, $\vec{\sigma}_m$;

The `metadata` fields `polynomial_names` and `observable_expressions` are not used in single-polynomial mode.

As discussed in Section 2.4, in both modes, uncertainties and correlations are interpreted based on a polynomial expansion of each observable in the model parameters, truncated at the order defined by the `metadata` field `polynomial_degree` (see next section).

3.3 Structure of `metadata`

The `metadata` field contains all contextual and structural information required to interpret the predictions. Its subfields are listed below. A full specification of each subfield, including its structure, format, input types and allowed values, is provided in Section 4.

- **`observable_names`** (***required***): array of observable names;
- **`parameters`** (***required***): array of model parameters (e.g. names of Wilson coefficients);
- **`basis`** (***required***): definition of the parameter basis, e.g. the operator basis in an EFT;
- **`scale`** (***required***): renormalisation scale at which the parameters are defined;
- **`polynomial_names`** (***required* in FOP mode**): array of polynomial names;
- **`observable_expressions`** (***required* in FOP mode**): Python-like expressions defining how each observable is computed from the named polynomials;
- **`polynomial_degree`** (***optional***): the order of truncation in the parameter expansion (default: 2);
- **`reproducibility`** (***optional***): information needed to reproduce the predictions. Array in which each element may include the following fields:
  - **`inputs`** (***optional***): numerical values of input parameters that have been used to compute the polynomial coefficients, with optional uncertainties and correlations;
  - **`tool`** (***optional***): information about the tools/methods used to generate the predictions;
  - **`description`** (***optional***): text field with a summary of how the predictions were obtained.
- **`misc`** (***optional***): free-form documentation or provenance metadata.

3.4 Structure of `data`

The `data` field contains the numerical predictions. Its subfields, already described in Section 3.2, are listed below. A full specification of each subfield, including its structure, format, input types and allowed values, is provided in Section 4.

- **`polynomial_central` (*required* in FOP mode)**: central values of polynomial coefficients $\vec{p}_k$ for each named polynomial P_k.
- **`observable_central` (*required* in SP mode; *optional* in FOP mode)**: central values of *observable coefficients* $\vec{o}_m$, the polynomial coefficients for each observable O_m.
- **`observable_uncertainties` (*optional*)**: uncertainties $\vec{\sigma}_m$ on the observable coefficients.

Polynomial coefficients are indexed by monomials, written as stringified tuples of model parameters, and complex parameters are handled through optional tagging of real and imaginary parts (see below for more details). Values are arrays of numbers defining predictions for a set of polynomials or observables (single-element arrays for a single polynomial or observable). The length of each array must match the number of entries in the corresponding `metadata.polynomial_names` (for polynomials) and `metadata.observable_names` (for observables). Missing monomials are implicitly treated as having zero coefficients.

4 Specification of Fields in the `POPxf JSON` Format

A detailed specification of all fields in the `POPxf` data format is given below. Each subsection describes the structure, expected data type, and allowed values of the corresponding entries in the `JSON` object. The data type *object* mentioned below refers to a `JSON` object literal and corresponds to a set of key/value pairs representing named subfields. The format is divided into two main components: the `metadata` and `data` fields. An additional `$schema` field is included to specify the version of the `POPxf JSON` schema used. All quantities defined in this specification refer to a single datafile. They may be indexed by a superscript (n) with $n \in [1, N]$ to denote quantities in a collection of N datafiles. This is particularly relevant for discussing correlated predictions stored in separate files. Since this specification focuses on the format of a single datafile, we will omit the superscript (n) to keep the notation concise. As a convention, we assume that all dimensionful quantities are given in units of GeV.

4.1 `$schema` Field

The `$schema` field allows identifying a `JSON` file as conforming to the `POPxf` format and specifies the version of the `POPxf JSON` schema used. It must be set to

`"https://json.schemastore.org/popxf-1.0.json"`

for files conforming to this version of the specification. The version number will be incremented for future revisions of the `JSON` schema.

4.2 `metadata` Field

The `metadata` field contains all contextual and structural information required to interpret the numerical predictions. It is a `JSON` object with the following subfields:

`observable_names` (required, *type: array of string*)

Array of M names identifying each observable O_m. Must be an array of unique, non-empty strings, with at least one entry.

Example:

```
"observable_names": ["observable1", "observable2", "observable3"]
```

`parameters` (required, *type: array of string*)

Array of S names identifying each model parameter C_s (e.g., Wilson coefficient names). Must be an array of unique, non-empty strings, with at least one entry. In general, this includes $S_\mathbb{R}$ real-valued and $S_\mathbb{C}$ complex-valued parameters with $S = S_\mathbb{R} + S_\mathbb{C}$. The real-valued parameters and the real and imaginary parts of the complex-valued parameters are used as the $R = S_\mathbb{R} + 2\,S_\mathbb{C}$ independent variables of all polynomial terms and can be grouped together in a real-valued parameter vector $\vec{C}$ of length R.

Example:

```
"parameters": ["C1", "C2", "C3"]
```

`basis` (required, *type: object*)

Defines the parameter basis (e.g. an operator basis in an EFT). At least one of the two subfields `wcxf` and `custom` has to be present. If both subfields are present, any element of `parameters` (see above) not belonging to the `wcxf` basis is interpreted as belonging to the `custom` basis. The subfields are defined as follows:

- **`wcxf` (optional, *type: object*)**: Specifies an EFT basis defined by the Wilson Coefficient exchange format (WCxf) [3]. This object contains the following fields:
  - **`eft` (required, *type: string*)**: EFT name defined by WCxf (e.g., `"SMEFT"`)
  - **`basis` (required, *type: string*)**: Operator basis name defined by WCxf (e.g., `"Warsaw"`)
  - **`sectors` (optional, *type: array of string*)**: Array of renormalisation-group-closed sectors of Wilson coefficients containing the Wilson coefficients given in `parameters` (see above). The available sectors for each EFT are defined by WCxf.
- **`custom` (optional, *type: any*)**: Field of any type and substructure to unambiguously specify any parameter basis not defined by WCxf.

Example:

```
"basis": {
  "wcxf": {
    "eft": "SMEFT",
    "basis": "Warsaw",
    "sectors": ["dB=de=dmu=dtau=0"]
  }
}
```

`polynomial_names` (optional, *type: array of string*)

This field is required to express observables as functions of polynomials. It requires the simultaneous presence of `metadata.observable_expressions` *and* `data.polynomial_central`.

Array of K names identifying the individual polynomials P_k that enter the observable predictions through the functions defined in `metadata.observable_expressions` (see below). Must contain unique, non-empty strings.

Example:

```
"polynomial_names": ["polynomial 1", "polynomial 2"]
```

`observable_expressions` (optional, *type: array of object*)

This field is required to express observables as functions of polynomials. It requires the simultaneous presence of `metadata.polynomial_names` *and* `data.polynomial_central`.

Defines how each observable is constructed from the named polynomials. Must be an array of M objects, one per observable. The length and order of the array must match those of the `observable_names` field. Each object must contain:

- **`variables` (required, *type: object*)**: An object where each key is a string that is a Python-compatible variable name (used as variable in the `expression` field described below), and each value is a string identifying a polynomial name from `polynomial_names`. For example, `{"num": "polynomial 1", "den": "polynomial 2"}`.
- **`expression` (required, *type: string*)**: A Python-compatible mathematical expression using the variable names defined in `variables`, e.g. `"num/den"`. Standard mathematical functions like `sqrt` or `cos` that are implemented in packages like `numpy` may be used.

Example:

```
"observable_expressions": [
  {
    "variables": {
      "num": "polynomial 1",
      "den": "polynomial 2"
    },
    "expression": "num / den"
  },
  {
    "variables": {
      "num": "polynomial 2",
      "den": "polynomial 1"
    },
    "expression": "num / den"
  },
  {
    "variables": {
      "p1": "polynomial 1"
    },
    "expression": "sqrt(p1**2)"
  }
]
```

`scale` (required, *type: number, array*)

The renormalisation scale in GeV at which the parameter vector $\vec{C}$, the polynomial coefficients $\vec{p}_k \supset \vec{b}_k, \vec{c}_k, ...$, and the observable coefficients $\vec{o}_m \supset \vec{b}_m, \vec{c}_m, ...$ and their uncertainties $\vec{\sigma}_m$ are defined. The parameter vector $\vec{C}$ that enters a given polynomial P_k or observable O_m has to be given at the same scale at which the polynomial coefficients $\vec{p}_k$ or observable coefficients $\vec{o}_m$ are defined, such that the polynomial or observable itself is scale-independent up to higher-order corrections in perturbation theory.

This field can take one of two forms:

- **single number**: A common scale μ at which all polynomial coefficients $\vec{p}_k$ or observable coefficients $\vec{o}_m$ are defined.
  - If the observables O_m are expressed in terms of polynomials P_k, the polynomials are functions of the parameters evolved to the common scale μ:

$$P_k = a_k + \vec{C}(\mu) \cdot \vec{b}_k(\mu) + \dots$$

  - If the observables O_m are themselves polynomials, they are themselves functions of the parameters evolved to the common scale μ:

$$O_m = a_m + \vec{C}(\mu) \cdot \vec{b}_m(\mu) + \dots$$

- **array of numbers**: An array defining separate scales μ_k of polynomial coefficients $\vec{p}_k$ if `metadata.polynomial_names` is present, or separate scales μ_m of observable coefficients $\vec{o}_m$ if `metadata.polynomial_names` is absent.
  - If `metadata.polynomial_names` is present, the observables O_m are expressed in terms of polynomials P_k and each polynomial is a function of the parameters evolved to its corresponding scale μ_k:

    $$P_k = a_k + \vec{C}(\mu_k) \cdot \vec{b}_k(\mu_k) + \dots$$

    The length and order of the array defining the scales μ_k must match those of the field `metadata.polynomial_names`. To avoid ambiguities, the following restrictions apply to this case:
    - `data.observable_central` must be absent;
    - `data.observable_uncertainties` must be absent or only define uncertainties for the parameter-independent terms (i.e. only the SM uncertainties in EFT applications).
  - If `metadata.polynomial_names` is absent, the observables O_m are themselves polynomials and each observable is a function of the parameters evolved to its corresponding scale μ_m:

    $$O_m = a_m + \vec{C}(\mu_m) \cdot \vec{b}_m(\mu_m) + \dots$$

    The length and order of the array defining the scales μ_m must match those of the field `metadata.observable_names`.

Examples:

```
"scale": 91.1876
```

```
"scale": [100.0, 200.0, 300.0, 400.0, 500.0]
```

`polynomial_degree` (optional, *type: integer*)

Specifies the maximum degree of polynomial terms included in the expansion. If omitted, the default value is 2 (i.e., quadratic polynomial). Values higher than 2 may be used to represent observables involving higher-order terms in the model parameters. The current implementation of the `JSON` schema defining the data format supports values up to 5. Higher degrees are not prohibited in principle but are currently unsupported to avoid excessively large data structures.

Example:

```
"polynomial_degree": 2
```

`reproducibility` (optional, *type: array of object*)

Collects relevant data that may be required by a third party to reproduce the prediction. Each element of the array should be an object that corresponds to a step in the workflow and has three predefined fields: `description`, `tool` and `inputs`, specified below. In addition, any additional fields containing data deemed useful in this context can be included.

Schematic example:

```
"reproducibility": [
  {
    "description": "Description of the first step",
    "tool": { ... },
    "inputs": { ... }
  },
  {
    "description": "Description of the second step",
    "tool": { ... },
    "inputs": { ... }
  },
  ...
]
```

The predefined fields are as follows:

- **`description` (optional, *type: string*)**: Free-form text description of the method and tool used in this step of obtaining the predictions.
- **`inputs` (optional, *type: object*)**: Specifies the numerical values of input parameters used by the tool in producing the numerical values of the polynomial coefficients. Each entry maps an input name (a string) or a group of names (a stringified tuple such as `"('m1','m2')"`) to one of the following:
  - A single number: interpreted as the central value of a single, uncorrelated input parameter without uncertainty;
  - An object representing a uni- or multi-variate normal distribution describing one or more possibly correlated input parameters with uncertainties. This object can contain the subfields `mean`, `std`, and `corr`. If the key of the object is a stringified tuple of N input names (e.g., `"('m1','m2')"` with $N = 2$), describing a group of N possibly correlated input parameters, then `mean` and (if present) `std` must be arrays of length N, and (if present) `corr` must be an $N \times N$ matrix, expressed as an array of N arrays of N numbers. The subfields are defined as follows:
    - **`mean` (required, *type: number, array*)**: central value / mean; a single number for a single input name, or an array of numbers for a group of input names;
    - **`std` (optional, *type: number, array*)**: uncertainty / standard deviation; a single number for a single input name, or an array of numbers for a group of input names;

    - **`corr` (optional, *type: array of array*)**: correlation matrix; must only be used if a group of input names is given and requires the presence of `std`.
  - An object representing an arbitrary user-defined uni- or multi-variate probability distribution describing one or more input parameters. This object contains the following subfields:
    - **`distribution_type` (required, *type: string*)**: a user-defined name identifying the probability distribution (e.g. `"uniform"`);
    - **`distribution_parameters` (required, *type: object*)**: an object where each key is a user-defined name of a parameter of the probability distribution, and each value is a single number in the univariate case, or an array of numbers or arrays in the multivariate case (e.g. `{"a":0, "b":1}` for a uniform distribution with boundaries a and b).
    - **`distribution_description` (required, *type: string*)**: Description of the custom distribution implemented, defining the fields in `distribution_parameters`.

  Example:
  In the example below, `"m1"` is an input parameter with no associated uncertainty, `"m2"` and `"m3"` are a pair of input parameters with correlated, Gaussian uncertainties, and `"m4"` is a parameter that is uniformly distributed between 0 and 1.

```
"inputs": {
  "m1": 1.0,
  "('m2','m3')": {
    "mean": [1.0, 2.0],
    "std": [0.1, 0.1],
    "corr": [
      [1.0, 0.3],
      [0.3, 1.0]
    ]
  },
  "m4": {
    "distribution_type": "uniform",
    "distribution_parameters": {
      "a": 0,
      "b": 1
    },
    "distribution_description": "Uniform distribution with
    ↪  boundaries $a$ and $b$."
  }
}
```

- **`tool` (optional, *type: object*)**: Provides free-form information about the tool, software or technique used in a particular step of the workflow. The predefined subfields are `name`, `version`, and `settings`. Any number of additional fields may be included to record or link to supplementary metadata, such as model information/configuration, perturbative order, scale choice, PDF sets, simulation settings, input parameter cards, etc. The predefined subfields are as follows:

- **`name` (required, *type: string*)**: name of tool, e.g. `"MadGraph5_aMC@NLO"`, `"POWHEG"`, `"SHERPA"`, `"WHIZARD"`, `"flavio"`, `"FeynCalc"`, `"analytical calculation"`, ...
- **`version` (optional, *type: string*)**: version of the tool, e.g. `"1.2"`
- **`settings` (optional, *type: object*)**: object containing information about the tool settings with free-form substructure. For example:
  * `perturbative_order` (e.g. `"LO"`, `"NLO"`, `"NLOQCD"`, ...)
  * `PDF`: name, version, and set of the PDF used.
  * `UFO`: name and version of UFO model used, as well as any other relevant information such as flavor schemes or webpage link.
  * `cuts`: Information about kinematical cuts specifying the phase space region over which the observable is computed (e.g. acceptance effects, signal region definition, ...).
  * `scale_choice`: Nominal scale choice employed when computing the predictions. This could be an array of fixed scales or a string describing a dynamical scale choice like `"dynamical:HT/2"`. This field is particularly relevant when RGE effects are folded into the prediction, see the description of `metadata.scale` above.
  * `renormalization_scheme`: details of the renormalization scheme used in the computation.
  * `covariant_derivative_sign`: sign convention used for the covariant derivative (`"+"` or `"-"`).
  * `gamma5_scheme`: scheme used for γ_5 in dimensional regularization (`"BMHV"`, `"KKS"`, ...).
  * `evanescent`: details of the treatment of evanescent operators, e.g. a reference to the scheme used.
  * `approximations`: Any relevant approximations used, such as the use of the first leading-logarithmic approximation for RG evolution.
  * any other relevant settings specific to the tool or calculation.

Examples:

```
"tool": {
  "name": "EFTTool",
  "version": "1.0.0"
}
```

```
"tool": {
  "name": "MadGraph5_aMC@NLO",
  "version": "3.6.2",
  "settings": {
    "UFO": {
      "name": "SMEFTUFO",
      "version": "1.0.0",
      "webpage": "https://smeftufo.io"
    },
```

```
      "PDF": {
        "name": "LHAPDF",
        "version": "6.5.5",
        "set": "331700"
      },
      "perturbative_order": "NLOQCD",
      "scale_choice": [91.1876, 125.0]
    }
  }
```

```
  "tool": {
    "name": "AnalysisTool",
    "version": "1.0.0",
    "settings": {
      "cuts": {
        "pT_min": 20.0,
        "eta_max": 2.5
      },
      "code": "https://coderepository.com/analysis/example"
    }
  }
```

```
  "tool": {
    "name": "analytical calculation",
    "settings": {
      "gamma5_scheme": "KKS",
      "covariant_derivative_sign": "-",
      "renormalization_scheme": "MSbar (WCs), On-shell (mass, aS,
      ↪  aEW)",
     "evanescent": "https://doi.org/10.1016/0550-3213(90)90223-Z"
    }
  }
```

```
  "tool": {
    "name": "RGEtool",
    "version": "1.0.0",
    "settings": {
      "perturbative_order": "one-loop",
      "method": "evolution matrix formalism"
    }
  }
```

`misc` (optional, *type: object*)

Optional free-form metadata for documentation purposes. May include fields such as authorship, contact information, date, description of the observable, information identifying the associated correlation file (e.g. hash value or filename), or external references. The format is unrestricted, allowing any `JSON`-encodable content.

Example:

```
"misc": {
  "author": "John Doe",
  "contact": "john.doe@example.com",
  "description": "Example dataset",
  "URL": "johndoe.com/exampledata",
  "correlation_file": "correlations.json",
  "correlation_file_hash": "AB47BG3F11DA7DCAA5726008BAAFE176"
}
```

4.3 `data` Field

The `data` field contains the numerical representation of all polynomial terms, which define the polynomials P_k and observables O_m. This information is provided in terms of central values of polynomial coefficients $\vec{p}_k$ and observable coefficients $\vec{o}_m$, and uncertainties of observable coefficients $\vec{\sigma}_m$.

Each component of $\vec{o}_m$, $\vec{p}_k$, and $\vec{\sigma}_m$ is labelled by a *monomial key*, written as a stringified tuple of model parameters (e.g., Wilson coefficients) defined in `metadata.parameters`. For example, the key `"('C1', 'C2')"` corresponds to the monomial $C_1 C_2$. While the model parameters can be complex numbers, the monomials are defined for the real and imaginary parts of the model parameters (see below) and are therefore strictly real. The format and conventions for monomial keys are as follows:

- Each key is a string representation of a Python-style tuple: a comma-separated array of strings enclosed in parentheses.
- The length of the tuple is determined by the polynomial degree d, as defined by the `metadata` field `polynomial_degree` (default value: $d = 2$, i.e. quadratic polynomial, if `polynomial_degree` is omitted). The tuple length equals d, unless a real/imaginary tag is included (see below), in which case the length is $d + 1$.
- The first d entries in the tuple are model parameter names, as defined in the `metadata` field `parameters`. These names must be sorted alphabetically to ensure unique monomial keys (assuming the same sorting rules as Python's `sort()` method which sorts alphabetically according to ASCII or UNICODE-value, where all upper-case letters come before all lower-case letters, and shorter strings take precedence). Empty strings `''` are used to represent constant terms (equivalent to 1) and to pad monomials of lower degree. For example, for a quadratic polynomial in real parameters (see below for how complex parameters are handled):
  - A constant 1 is written as `"('','')"`,

  - A linear term C_1 is written as `"('', 'C1')"`,
  - A quadratic term C_1C_2 is written as `"('C1', 'C2')"`.
- To handle complex parameters, the tuple may optionally include a real/imaginary tag as its final element. This tag consists of `R` (real) and `I` (imaginary) characters, and its length must match the polynomial degree d. It indicates whether each parameter refers to its real or imaginary part. For example:
  - `"('', 'C1', 'RI')"` corresponds to $\mathrm{Im}(C_1)$;
  - `"('C1', 'C2', 'IR')"` corresponds to $\mathrm{Im}(C_1)\mathrm{Re}(C_2)$.
- If the real/imaginary tag is omitted, the parameters are assumed to be real. For example:
  - `"('', 'C1')"` corresponds to $\mathrm{Re}(C_1)$;
  - `"('C1', 'C2')"` corresponds to $\mathrm{Re}(C_1)\mathrm{Re}(C_2)$.

These conventions ensure a canonical and unambiguous representation of polynomial terms while offering flexibility in the naming of model parameters. Missing monomials are implicitly treated as having zero coefficients.

The `data` field is a `JSON` object with the following subfields:

`polynomial_central` (optional, *type: object*)

This field is required to express observables as functions of polynomials. It requires the simultaneous presence of `metadata.polynomial_names` *and* `metadata.observable_expressions`.

An object representing the central values of the polynomial coefficients $\vec{p}_k$ for each named polynomial P_k. Each key must be a monomial key as defined above. Each value must be an array of K numbers whose order matches `metadata.polynomial_names`.

Example:

Specifying two polynomials, P_k, given in terms of two complex parameters C_1 and C_2 as

$$P_1 = 1.0 + 1.2\ \mathrm{Im}(C_1) + 0.8\ \mathrm{Re}(C_1)\mathrm{Re}(C_2) + 0.5\ \mathrm{Re}(C_1)\mathrm{Im}(C_2) + 0.2\ \mathrm{Im}(C_1)\mathrm{Im}(C_2)\ ,$$
$$P_2 = 1.1 + 1.3\ \mathrm{Im}(C_1) + 0.85\ \mathrm{Re}(C_1)\mathrm{Re}(C_2) + 0.55\ \mathrm{Re}(C_1)\mathrm{Im}(C_2) + 0.25\ \mathrm{Im}(C_1)\mathrm{Im}(C_2)\ .$$

```
"polynomial_central": {
  "('', '', 'RR')": [1.0, 1.1],
  "('', 'C1', 'RI')": [1.2, 1.3],
  "('C1', 'C2', 'RR')": [0.8, 0.85],
  "('C1', 'C2', 'RI')": [0.5, 0.55],
  "('C1', 'C2', 'II')": [0.2, 0.25]
}
```

`observable_central` (optional, *type: object*)

An object representing the central values of the observable coefficients $\vec{o}_m$ for each observable O_m. In case the observables are not themselves polynomials, the observable coefficients correspond to the polynomial approximation of the observables obtained from a Taylor expansion of the observable expressions defined in `metadata.observable_expressions`. Each key must be a monomial key as defined above. Each value must be an array of M numbers whose order matches `metadata.observable_names`.

Example:

Specifying three observable predictions, O_m, given in terms of the three real parameters C_1, C_2, and C_3 as

$$\begin{aligned} O_1 &= 1.0 + 1.2\, C_1 + 1.4\, C_1 C_2 + 1.6\, C_1 C_3\,, \\ O_2 &= 1.1 + 1.3\, C_1 + 1.5\, C_1 C_2 + 1.7\, C_1 C_3\,, \\ O_3 &= 2.3 + 0.3\, C_1 + 0.7\, C_1 C_2 + 0.5\, C_1 C_3\,. \end{aligned}$$

```
"observable_central": {
  "('', '')": [1.0, 1.1, 2.3],
  "('', 'C1')": [1.2, 1.3, 0.3],
  "('C1', 'C2')": [1.4, 1.5, 0.7],
  "('C1', 'C3')": [1.6, 1.7, 0.5]
}
```

`observable_uncertainties` (optional, *type: object*)

An object representing the uncertainties on the observable coefficients $\vec{\sigma}_m$ for each observable O_m. In case the observables are not themselves polynomials, the observable coefficients correspond to the polynomial approximation of the observables obtained from a Taylor expansion of the observable expressions defined in `metadata.observable_expressions`. The fields specify the nature of quoted uncertainty. In many cases there may only be a single top-level field, `"total"`, but multiple fields can be used to specify a breakdown into several sources of uncertainty (e.g., statistical, scale, PDF, ...). To avoid mistakes, the names of the top-level fields must not have the format of a monomial key (i.e., stringified tuples as defined above). The value of each top-level field can either be an object or an array of floats. Objects must have the same structure as `observable_central`, arrays must have length M. If instead of an object, an array of floats is specified, it is assumed to correspond to the parameter independent uncertainty only (e.g. the uncertainty on the SM prediction). This would be equivalent to specifying an object containing a single element with the monomial key of the constant term (e.g. `"('','')"` for a quadratic polynomial).

Examples:

```
"observable_uncertainties": {
  "total": {
    "('', '')": [0.05, 0.06, 0.01],
    "('', 'C1')": [0.1, 0.12, 0.01],
    "('C1', 'C2')": [0.02, 0.03, 0.02],
    "('C1', 'C3')": [0.05, 0.06, 0.01]
  }
}
```

Specifying only the SM uncertainties:

```
"observable_uncertainties": {
  "total": [0.05, 0.06, 0.01]
}
```

Specifying an uncertainty breakdown:

```
"observable_uncertainties": {
  "MC_stats": {
    "('', '')": [0.002, 0.0012, 0.001],
    "('', 'C1')": [0.001, 0.0015, 0.0001]
  },
  "scale": {
    "('', '')": [0.04, 0.05, 0.06],
    "('', 'C1')": [0.1, 0.12, 0.01]
  },
  "PDF": {
    "('', '')": [0.03, 0.04, 0.05],
    "('', 'C1')": [0.02, 0.08, 0.01]
  }
}
```

Specifying a breakdown for SM uncertainties only:

```
"observable_uncertainties": {
  "MC_stats": [0.002, 0.0012, 0.001],
  "scale": [0.04, 0.05, 0.06],
  "PDF": [0.03, 0.04, 0.05]
}
```

5 Data structure for correlations of observables

The JSON structure defined in Sections 3 and 4 contains the main information on observable predictions including their central values and uncertainties. Since observables in one POPxf data file (labelled by index n) can be correlated with observables in another POPxf data file (labelled by index n'), we define correlations in a separate data structure that is implemented in a separate POPxf correlation file.

5.1 File formats

We support two different file formats for POPxf correlation data:

- JSON files are primarily used for parameter-independent correlations or correlations between a relatively small number of observables. In these cases, correlation matrices typically contain $\mathcal{O}(10)$-$\mathcal{O}(1000)$ rows and columns – a size for which the JSON data format is still suitable.
- HDF5 files are primarily used for parameter-dependent correlations or correlations between a large number of observables. In these cases, correlation matrices can have $\mathcal{O}(10^4)$-$\mathcal{O}(10^6)$ rows and columns – a size for which the Hierarchical Data Format version 5 (HDF5) is particularly well suited, as it is specifically designed for storing large numerical arrays.

For both file formats, we encode the correlation matrices in the same hierarchical data structure, with differences only in syntax: in JSON, the data is represented by JSON *objects* containing *key-value pairs with nested arrays as values*, whereas in HDF5, the same structure is realized through HDF5 *groups* containing HDF5 *datasets*. In order to use a common language for both JSON and HDF5 data structures, we will use the HDF5 terms for the correpsonding JSON structures: we denote a *key-value pair where the value is a* JSON *object* as *group* and a *key-value pair where the value is a (nested) array* as *dataset*. Instead of *key* and *value*, we will use *name* and *content* (of a group or dataset). The mapping between JSON and HDF5 data structures and the common language used in the following are shown in Table 2.

JSON		HDF5		common language
key-value pair with JSON object as value	$\leftrightarrow$	HDF5 group	$\leftrightarrow$	group
key-value pair with (nested) array as value	$\leftrightarrow$	HDF5 dataset	$\leftrightarrow$	dataset
key	$\leftrightarrow$	name of HDF5 group or dataset	$\leftrightarrow$	name
value	$\leftrightarrow$	content of HDF5 group or dataset	$\leftrightarrow$	content

Table 2: Mapping between JSON and HDF5 data structures and the common language used for describing POPxf correlation data.

JSON file

```
{
  "$schema": "https://...",
  "<name of first entry>": {
    "row_names": [...],
    "col_names": [...],
    "correlations": {
      "<source 1>":[[...],...],
      "<source 2>":[[...],...],
      ...
    }
  },
  "<name of second entry>": {
    "row_names": [...],
    "col_names": [...],
    "correlations": {
      "<source 1>":[[...],...],
      "<source 2>":[[...],...],
      ...
    }
  },
  ...
}
```

HDF5 file

- attribute: `$schema` (*type: string*)
- group: `<name of first entry>`
  - dataset: `row_names` (*type: string*)
  - dataset: `col_names` (*type: string*)
  - group: `correlations`
    - dataset: `<source 1>` (*type: numeric*)
      - attribute: `scale_factor` (*optional*)
    - dataset: `<source 2>` (*type: numeric*)
      - attribute: `scale_factor` (*optional*)
    - ...
- group: `<name of second entry>`
  - dataset: `row_names` (*type: string*)
  - dataset: `col_names` (*type: string*)
  - group: `correlations`
    - dataset: `<source 1>` (*type: numeric*)
      - attribute: `scale_factor` (*optional*)
    - dataset: `<source 2>` (*type: numeric*)
      - attribute: `scale_factor` (*optional*)
    - ...
- ...

Fig. 2: Structure of POPxf correlation file in JSON (left) and HDF5 (right).

5.2 File structure

To encode correlation matrices $[\rho^{(nn')}_{mm'}]_{\alpha\alpha'}$ (as defined in Section 2.4) in a POPxf correlation file, we define one top-level entry for each pair of correlated POPxf data files, i.e. for each combination of (nn') indices with $n \geq n'$. The correlations for $n < n'$ are related to those for $n > n'$ by $[\rho^{(nn')}_{mm'}]_{\alpha\alpha'} = [\rho^{(n'n)}_{m'm}]_{\alpha'\alpha}$. Each top-level entry in the POPxf correlation file is a *group* with unique name. The names are arbitrary, but for efficient lookup we recommend using specific hash values as names, which we describe in Section 5.2.2.

POPxf correlation files in JSON format contain an additional top-level field `$schema`, which declares the JSON schema specification used by the file. This allows identifying a given JSON file as a POPxf correlation file and validating its structure. The value must be

`https://json.schemastore.org/popxf-corr-1.0.json`

for the first version of this correlation format and the version number will be incremented if the format is extended in the future. POPxf correlation files in HDF5 format contain the same value as an HDF5 top-level *attribute* called `$schema`. This allows converting POPxf correlation files from HDF5 to JSON and vice versa.

5.2.1 Structure of top-level entries

Each top-level entry contains three elements:

- **`row_names`** (***dataset, type: string***): array of observable names equal to the field `metadata.observable_names` in data file n. These observable names correspond to the row index m of $[\rho_{mm'}^{(nn')}]_{\alpha\alpha'}$.
- **`col_names`** (***dataset, type: string***): array of observable names equal to the field `metadata.observable_names` in data file n'. These observable names correspond to the column index m'.
- **`correlations`** (***group***): this group contains one numeric *dataset* for each source of correlated uncertainty (e.g., MC statistics, scale, PDF, etc.). The names of the datasets have to match the keys of the correlated `data.observable_uncertainties` fields in data files n and n'. If uncertainties are not broken down into several sources, only a single dataset named `total` may be present. Keys in the `data.observable_uncertainties` fields of data files n and n' for which no dataset is included, and keys that are only present in one of these files are assumed to correspond to uncorrelated uncertainties. The content of each included dataset is a multidimensional numerical array (nested array in `JSON`) containing the correlation coefficients. The shape of this array depends on whether the uncertainties and correlations are parameter-independent or parameter-dependent:
  - For parameter-independent correlations, the array has two dimensions:
    1. axis corresponding to the observable names in `row_names`, i.e. those in `metadata.observable_names` in data file n. The number and order of the entries has to match those of the observable names in `row_names`.
    2. axis corresponding to the observable names in `col_names`, i.e. those in `metadata.observable_names` in data file n'. The number and order of the entries has to match those of the observable names in `col_names`.
  - For parameter-dependent correlations, the array has four dimensions:
    1. axis corresponding to the observable names in `row_names`, i.e. those in `metadata.observable_names` in data file n. The number and order of the entries has to match those of the observable names in `row_names`.
    2. axis corresponding to the observable names in `col_names`, i.e. those in `metadata.observable_names` in data file n'. The number and order of the entries has to match those of the observable names in `col_names`.
    3. axis corresponding to the parameter monomials used as keys in the field `data.observable_central` in data file n. The number and order of the entries has to match those of the alphabetically sorted monomial keys.
    4. axis corresponding to the parameter monomials used as keys in the field `data.observable_central` in data file n'. The number and order of the entries has to match those of the alphabetically sorted monomial keys.

  In `HDF5` correlation files, each dataset within a given `correlations` element has an optional *attribute* `scale_factor`, which contains a single floating point number. If present, all numerical values in the dataset have to be multiplied by `scale_factor` to obtain the correlation coefficients. The main use-case for the `scale_factor` attribute is to reduce the file size by implementing the correlation coefficients in a numerical format other than floating point numbers between -1.0 and $+1.0$.

For example, by setting `scale_factor` to $1/32767$, 16-bit signed integers between -32767 and $+32767$ can be used to represent correlation coefficients. If absent, `scale_factor` is assumed to be 1.0. `scale_factor` is not supported in `JSON` files. If file size matters, `HDF5` should be used instead of `JSON`.

For each pair of correlated `POPxf` data files, the corresponding correlation data can be identified by matching the `observable_names` metadata fields of these files to the `row_names` and `col_names` elements in a `POPxf` correlation file. For a more efficient lookup, we recommend creating hash values from the `row_names` and `col_names` elements and using them as the names of the top-level entries, as described in the following section. The file structure in both file formats, `JSON` and `HDF5`, is illustrated in Fig. 2.

5.2.2 Hash values for efficient lookup of top-level entries

For each pair of correlated `POPxf` data files, the corresponding correlation data can, in principle, be located by loading all top-level entries in a `POPxf` correlation file and comparing the `row_names` and `col_names` elements to the `observable_names` metadata fields of the given pair of `POPxf` data files. However, this procedure requires reading the content of all `row_names` and `col_names` elements and is therefore inefficient. To enable faster lookup, we recommend computing hash values from `row_names` and `col_names` and using them as the names of the corresponding top-level entries in the `POPxf` correlation file. Then, to retrieve the correlation data for a given pair of correlated `POPxf` data files, it is sufficient to compute the same hash value from the `observable_names` metadata fields of these files, which can then be used to directly access the matching entry in the `POPxf` correlation file.

To standardise the computation of hash values, we adopt the following procedure for a given pair of `row_names` and `col_names`:

- Join all observable names in the arrays `row_names` and `col_names` into a single string each, using the vertical bar character "`|`" as the separator. Before joining, escape all existing vertical bar characters within observable names using a backslash "\", and escape any existing backslashes by doubling them.
- Concatenate the two resulting strings (from `row_names` and `col_names`) using two vertical bars "`||`" as the separator to form a single combined string.
- Compute the MD5 message digest of the UTF-8 encoded string and represent it as a 128-bit hexadecimal value. The MD5 algorithm is chosen for its simplicity and wide availability.

The resulting hash value uniquely identifies a pair of `row_names` and `col_names`.

For a given pair of correlated `POPxf` data files, it is not known *a priori* which of them corresponds to the rows or to the columns. By computing both possible hash values and checking which one is present in the correlation file, one can identify the corresponding correlation data and, at the same time, determine which file corresponds to the rows and which to the columns. Using these hash values as top-level entry names ensures unambiguous identification and enables efficient access to the correlation data without requiring a full scan of the file contents.

6 Conclusions and Outlook

In this note, we have proposed the Polynomial Observable Prediction Exchange Format, `POPxf`, a standardised, machine-readable data format for sharing theoretical predictions that can be expressed as (functions of) polynomials in the model parameters. While the focus of this format is in EFT applications, it remains general enough to apply to other related cases. `POPxf` allows for the encoding of observable predictions via their monomial coefficients, defining observables as functions of polynomials, and for the specification of theoretical uncertainties and their possibly parameter-dependent correlations. The format allows for the inclusion of ample metadata, to maximise the reproducibility of published predictions. Adopting a common standard will reduce the duplication of efforts, facilitate validation, and maximise the accessibility and impact of such theoretical predictions.

The concrete specification of the `POPxf` data format in terms of `JSON` schemas is hosted in a public repository at https://github.com/pop-xf, which also includes a lightweight validator and associated command line tool as well as a collection of example files.

Acknowledgments. This work was done on behalf of the LHC Higgs and EFT Working Groups and we would like to thank all members for stimulating discussions that led to this document. We would also like to thank the CERN Theory division for their hospitality while the format was being developed. K.M. is supported by an Ernest Rutherford Fellowship from the STFC, Grant No. ST/X004155/1. A.S. is supported by the Slovenian Research and Innovation Agency (Grant No. J1-50219 and research core funding No. P1-0035).

A Examples of complete `JSON` structures

We give a few simplified examples of complete `JSON` files in the `POPxf` format below. The full examples including additional parameter dependencies and reproducibility information can be found at https://github.com/pop-xf/examples.

A.1 Single polynomial mode

The SP mode example below implements a SMEFT prediction for the partial W-boson width to a muon and muon-antineutrino, $\Gamma(W^- \to \mu^- \bar{\nu}_\mu)$, in the (m_W, G_F, m_Z) input scheme. The observable depends on 3 coefficients in the Warsaw basis as implemented in the `SMEFTatNLO` UFO model [4] which was used to compute the predictions with `MadGraph5_aMC@NLO` [5], as indicated by the `"custom"` field of `metadata.basis` and the `"tool"` field of the entry in `metadata.reproducibility`.

```
{
  "$schema": "https://json.schemastore.org/popxf-1.0.json",
  "metadata": {
    "basis": {
      "custom": {
        "eft": "SMEFT",
        "basis": "SMEFTatNLO",
        "definition": "https://feynrules.irmp.ucl.ac.be/wiki/SMEFTatNLO"
      }
    },
    "scale": 80.387,
    "parameters": [ "c3pl1", "c3pl2", "cll" ],
    "observable_names": ["Gamma(W -> mu nu_m)"],
    "reproducibility": [{
      "description": "Fixed-order Monte Carlo computation.",
      "tool": {
        "name": "MadGraph5_aMC@NLO",
        "version": "3.4.1",
        "settings":{ "UFO":"SMEFTatNLO 1.0.2",  "perturbative_order":"LO" }
      },
      "inputs":{ "m_W": 80.387, "G_F": 1.1663787e-5, "m_Z": 91.1876, "Lambda":1000 }
    }],
    "misc":{
      "description":"W-boson partial width to mu nu_m [GeV], (m_W,G_F,m_Z) scheme",
      "author": ["E. Celada", "L. Mantani", "K. Mimasu"],
      "contact": "ken.mimasu@soton.ac.uk"
    }
  },
  "data": {
    "observable_central": {
       "('', '')": [0.22729],
       "('', 'c3pl1')": [-0.0137796],
       "('', 'c3pl2')": [0.0137786],
       "('', 'cll')": [0.0137796],
       "('c3pl1', 'c3pl1')": [0.000208845],
       "('c3pl2', 'c3pl2')": [0.00020885],
       "('cll', 'cll')": [0.00020884],
       "('c3pl1', 'c3pl2')": [-0.00041769],
       "('c3pl1', 'cll')": [-0.00041768],
       "('c3pl2', 'cll')": [0.00041768]
    }
  }
}
```


A.2 Function-of-polynomials mode

The FOP mode example below implements a related SMEFT prediction for the ratios of partial W-boson widths to different lepton generations, $R_{\ell_i\ell_j} = \Gamma(W^- \to \ell_i^- \bar{\nu}_{\ell_i})/\Gamma(W^- \to \ell_j^- \bar{\nu}_{\ell_j})$, this time in the (α_{EW}, G_F, m_Z) input scheme, computed using `flavio` [6]. The input polynomials encode the dependence on the partial widths and the observable expressions encode their ratios. For brevity, the dependence is encoded for a subset of three Wilson coefficients in the Warsaw basis as implemented in the corresponding `WCxf` definition, and information about input parameters in the `reproducibility` field is omitted. The complete example file con-

taining the full dependence on all the contributing Wilson coefficients and including the entire `reproducibility` field can be found at https://github.com/pop-xf/examples under the name `Wlnu.json`.


```
{
  "$schema": "https://json.schemastore.org/popxf-1.0.json",
  "metadata": {
    "observable_names": ["Rmue(W->lnu)", "Rtaue(W->lnu)", "Rtaumu(W->lnu)"],
    "parameters": ["phil3_11", "phil3_22", "phil3_33"],
    "basis": {
      "wcxf": {
        "eft": "SMEFT",
        "basis": "Warsaw",
        "sectors": ["dB=de=dmu=dtau=0"]
      }
    },
    "polynomial_names": ["Gamma(W->enu)", "Gamma(W->munu)", "Gamma(W->taunu)"],
    "observable_expressions": [
      {
        "expression": "num / den",
        "variables": {"num": "Gamma(W->munu)", "den": "Gamma(W->enu)"}
      },
      {
        "expression": "num / den",
        "variables": {"num": "Gamma(W->taunu)", "den": "Gamma(W->enu)"}
      },
      {
        "expression": "num / den",
        "variables": {"num": "Gamma(W->taunu)", "den": "Gamma(W->munu)"}
      }
    ],
    "scale": 80.387,
    "reproducibility": [{"tool": {"name": "flavio", "version": "2.6.2"}}],
    "misc": {
      "description": "Using the (alpha, G_F, m_Z) input scheme.",
      "author": ["A. Smolkovic", "P. Stangl"]
    }
  },
  "data": {
    "polynomial_central": {
      "('', '')": [0.227, 0.227, 0.227],
      "('', 'phil3_11')": [7737.419, -19812.903, -19812.903],
      "('', 'phil3_22')": [-19812.903, 7737.419, -19812.903],
      "('', 'phil3_33')": [0, 0, 27550.322],
      "('phil3_11', 'phil3_11')": [929672417.073, 1295705157.301, 1295705157.301],
      "('phil3_11', 'phil3_22')": [1390270692.689, 1390270692.689, 2591410314.602],
      "('phil3_11', 'phil3_33')": [0, 0, -1201139621.914],
      "('phil3_22', 'phil3_22')": [1295705157.301, 929672417.073, 1295705157.301],
      "('phil3_22', 'phil3_33')": [0, 0, -1201139621.914],
      "('phil3_33', 'phil3_33')": [0, 0, 835106881.686]
    }
  }
}
```

A.3 Correlated uncertainties

The two SP mode examples below implement two observables, the branching ratios for $B_s \to \mu^+\mu^-$ and $B^0 \to \mu^+\mu^-$, with correlated theoretical uncertainties, computed using `flavio`. For brevity, we show only the dependence of the observables on the real and imaginary components of a subset of four WET Wilson coefficients defined by the `"flavio"` `WCxf` basis, and information about input parameters in the `reproducibility` field is omitted. The first example considers the simplified case where only the parameter-independent SM uncertainties are retained, while the second includes parameter-dependent uncertainties.

Parameter-independent uncertainties


```
{
  "$schema": "https://json.schemastore.org/popxf-1.0.json",
  "metadata": {
    "observable_names": ["BR(Bs->mumu)", "BR(B0->mumu)"],
    "parameters": ["C10_bdmumu", "C10_bsmumu", "C10p_bdmumu", "C10p_bsmumu"],
    "basis": {
      "wcxf": {
        "eft": "WET",
        "basis": "flavio",
        "sectors": ["db", "sb"]
        }
    },
    "scale": 4.8,
    "reproducibility": [{"tool": {"name": "flavio", "version": "2.6.2"}}],
    "misc": {
      "author": ["A. Smolkovic", "P. Stangl"]
    }
  },
  "data": {
    "observable_central": {
      "('', '', 'RR')": [3.629e-09, 1.014e-10],
      "('', 'C10_bdmumu', 'RR')": [0, -4.865e-11],
      "('', 'C10_bsmumu', 'RR')": [-1.742e-09, 0],
      "('', 'C10p_bdmumu', 'RR')": [0, 4.865e-11],
      "('', 'C10p_bsmumu', 'RR')": [1.742e-09, 0],
      "('C10_bdmumu', 'C10_bdmumu', 'II')": [0, 5.838e-12],
      "('C10_bdmumu', 'C10_bdmumu', 'RR')": [0, 5.838e-12],
      "('C10_bdmumu', 'C10p_bdmumu', 'II')": [0, -1.168e-11],
      "('C10_bdmumu', 'C10p_bdmumu', 'RR')": [0, -1.168e-11],
      "('C10_bsmumu', 'C10_bsmumu', 'II')": [1.837e-10, 0],
      "('C10_bsmumu', 'C10_bsmumu', 'RR')": [2.09e-10, 0],
      "('C10_bsmumu', 'C10p_bsmumu', 'II')": [-3.674e-10, 0],
      "('C10_bsmumu', 'C10p_bsmumu', 'RR')": [-4.181e-10, 0],
      "('C10p_bdmumu', 'C10p_bdmumu', 'II')": [0, 5.838e-12],
      "('C10p_bdmumu', 'C10p_bdmumu', 'RR')": [0, 5.838e-12],
      "('C10p_bsmumu', 'C10p_bsmumu', 'II')": [1.837e-10, 0],
      "('C10p_bsmumu', 'C10p_bsmumu', 'RR')": [2.09e-10, 0]
    },
    "observable_uncertainties": {
      "total": [1.046e-10, 5.945e-12]
    }
  }
}
```

The parameter-independent `POPxf` correlation `JSON` file corresponding to the above observable predictions is given below:


```
{
  "$schema": "https://json.schemastore.org/popxf-corr-1.0.json",
  "593771630098eb5325684131f80b4224": {
    "row_names": ["BR(Bs->mumu)", "BR(B0->mumu)"],
    "col_names": ["BR(Bs->mumu)", "BR(B0->mumu)"],
    "correlations": {
      "total":[
        [1.0, 0.407],
        [0.407, 1.0]
      ]
    }
  }
}
```


Parameter-dependent uncertainties

Since $B_s \to \mu^+\mu^-$ and $B^0 \to \mu^+\mu^-$ depend on different Wilson coefficients, it is convenient to define them in separate `POPxf` files. This also simplifies and reduces the size of the corresponding `POPxf` correlation file. The complete example file containing the full dependence on all the contributing Wilson coefficients and including the entire `reproducibility` field can be found at https://github.com/pop-xf/examples under the names `Bsmumu.json` and `B0mumu.json`.


```
{
  "$schema": "https://json.schemastore.org/popxf-1.0.json",
  "metadata": {
    "observable_names": ["BR(Bs->mumu)"],
    "parameters": ["C10_bsmumu", "C10p_bsmumu"],
    "basis": {
      "wcxf": {
        "eft": "WET",
        "basis": "flavio",
        "sectors": ["sb"]
      }
    },
    "scale": 4.8,
    "reproducibility": [{"tool": {"name": "flavio", "version": "2.6.2"}}],
    "misc": {
      "author": ["A. Smolkovic", "P. Stangl"]
    }
  },
  "data": {
    "observable_central": {
      "('', '', 'RR')": [3.629e-09],
      "('', 'C10_bsmumu', 'RR')": [-1.742e-09],
      "('', 'C10p_bsmumu', 'RR')": [1.742e-09],
      "('C10_bsmumu', 'C10_bsmumu', 'II')": [1.837e-10],
      "('C10_bsmumu', 'C10_bsmumu', 'RR')": [2.09e-10],
      "('C10_bsmumu', 'C10p_bsmumu', 'II')": [-3.674e-10],
      "('C10_bsmumu', 'C10p_bsmumu', 'RR')": [-4.181e-10],
      "('C10p_bsmumu', 'C10p_bsmumu', 'II')": [1.837e-10],
```

```
      "('C10p_bsmumu', 'C10p_bsmumu', 'RR')": [2.09e-10]
    },
    "observable_uncertainties": {
      "total": {
        "('', '', 'RR')": [1.046e-10],
        "('', 'C10_bsmumu', 'RR')": [4.653e-11],
        "('', 'C10p_bsmumu', 'RR')": [4.653e-11],
        "('C10_bsmumu', 'C10_bsmumu', 'II')": [4.758e-12],
        "('C10_bsmumu', 'C10_bsmumu', 'RR')": [5.427e-12],
        "('C10_bsmumu', 'C10p_bsmumu', 'II')": [9.516e-12],
        "('C10_bsmumu', 'C10p_bsmumu', 'RR')": [1.085e-11],
        "('C10p_bsmumu', 'C10p_bsmumu', 'II')": [4.758e-12],
        "('C10p_bsmumu', 'C10p_bsmumu', 'RR')": [5.427e-12]
      }
    }
  }
}
```

```
{
  "$schema": "https://json.schemastore.org/popxf-1.0.json",
  "metadata": {
    "observable_names": ["BR(B0->mumu)"],
    "parameters": ["C10_bdmumu", "C10p_bdmumu"],
    "basis": {
      "wcxf": {
        "eft": "WET",
        "basis": "flavio",
        "sectors": ["db"]
      }
    },
    "scale": 4.8,
    "reproducibility": [{"tool": {"name": "flavio", "version": "2.6.2"}}],
    "misc": {
      "author": ["A. Smolkovic", "P. Stangl"]
    }
  },
  "data": {
    "observable_central": {
      "('', '', 'RR')": [1.014e-10],
      "('', 'C10_bdmumu', 'RR')": [-4.865e-11],
      "('', 'C10p_bdmumu', 'RR')": [4.865e-11],
      "('C10_bdmumu', 'C10_bdmumu', 'II')": [5.838e-12],
      "('C10_bdmumu', 'C10_bdmumu', 'RR')": [5.838e-12],
      "('C10_bdmumu', 'C10p_bdmumu', 'II')": [-1.168e-11],
      "('C10_bdmumu', 'C10p_bdmumu', 'RR')": [-1.168e-11],
      "('C10p_bdmumu', 'C10p_bdmumu', 'II')": [5.838e-12],
      "('C10p_bdmumu', 'C10p_bdmumu', 'RR')": [5.838e-12]
    },
    "observable_uncertainties": {
      "total": {
        "('', '', 'RR')": [5.945e-12],
        "('', 'C10_bdmumu', 'RR')": [2.805e-12],
        "('', 'C10p_bdmumu', 'RR')": [2.805e-12],
        "('C10_bdmumu', 'C10_bdmumu', 'II')": [3.345e-13],
        "('C10_bdmumu', 'C10_bdmumu', 'RR')": [3.345e-13],
        "('C10_bdmumu', 'C10p_bdmumu', 'II')": [6.691e-13],
```

```
        "('C10_bdmumu', 'C10p_bdmumu', 'RR')": [6.691e-13],
        "('C10p_bdmumu', 'C10p_bdmumu', 'II')": [3.345e-13],
        "('C10p_bdmumu', 'C10p_bdmumu', 'RR')": [3.345e-13]
      }
    }
  }
}
```


The corresponding `POPxf` correlation `JSON` file is given below. The $(0, 0, 0, 0)$ entry of each numerical dataset corresponds to the parameter-independent piece documented in the example above. The remaining terms account for the parameter dependence. Since each data file describes one observable and has nine entries in `observable_central`, each numerical dataset is represented by an array of shape $(1, 1, 9, 9)$. The complete correlation file containing the full dependence on all the contributing Wilson coefficients can be found at https://github.com/pop-xf/examples under the name `corr.json`.


```
{
  "$schema": "https://json.schemastore.org/popxf-corr-1.0.json",
  "974bcd243772ce08f33a16c7fda240de": {
    "row_names": ["BR(B0->mumu)"],
    "col_names": ["BR(B0->mumu)"],
    "correlations": {
      "total": [[[
            [1.0, -0.994, 0.994, 0.977, 0.977, -0.977, -0.977, 0.977, 0.977],
            [-0.994, 1.0, -1.0, -0.994, -0.994, 0.994, 0.994, -0.994, -0.994],
            [0.994, -1.0, 1.0, 0.994, 0.994, -0.994, -0.994, 0.994, 0.994],
            [0.977, -0.994, 0.994, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0],
            [0.977, -0.994, 0.994, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0],
            [-0.977, 0.994, -0.994, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0],
            [-0.977, 0.994, -0.994, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0],
            [0.977, -0.994, 0.994, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0],
            [0.977, -0.994, 0.994, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0]
          ]]]
    }
  },
  "a262ca783a3dd055c77ec5c6c75c6ffe": {
    "row_names": ["BR(B0->mumu)"],
    "col_names": ["BR(Bs->mumu)"],
    "correlations": {
      "total": [[[
            [0.407, -0.389, 0.389, 0.35, 0.349, -0.35, -0.349, 0.35, 0.349],
            [-0.367, 0.371, -0.371, -0.356, -0.355, 0.356, 0.355, -0.356, -0.355],
            [0.367, -0.371, 0.371, 0.356, 0.355, -0.356, -0.355, 0.356, 0.355],
            [0.322, -0.347, 0.347, 0.358, 0.357, -0.358, -0.357, 0.358, 0.357],
            [0.322, -0.347, 0.347, 0.358, 0.357, -0.358, -0.357, 0.358, 0.357],
            [-0.322, 0.347, -0.347, -0.358, -0.357, 0.358, 0.357, -0.358, -0.357],
            [-0.322, 0.347, -0.347, -0.358, -0.357, 0.358, 0.357, -0.358, -0.357],
            [0.322, -0.347, 0.347, 0.358, 0.357, -0.358, -0.357, 0.358, 0.357],
            [0.322, -0.347, 0.347, 0.358, 0.357, -0.358, -0.357, 0.358, 0.357]
          ]]]
    }
  },
  "1af389d015582d6903a33587d94d45ea": {
```

```
    "row_names": ["BR(Bs->mumu)"],
    "col_names": ["BR(Bs->mumu)"],
    "correlations": {
      "total": [[[
            [1.0, -0.978, 0.978, 0.848, 0.902, -0.848, -0.902, 0.848, 0.902],
            [-0.978, 1.0, -1.0, -0.914, -0.972, 0.914, 0.972, -0.914, -0.972],
            [0.978, -1.0, 1.0, 0.914, 0.972, -0.914, -0.972, 0.914, 0.972],
            [0.848, -0.914, 0.914, 1.0, 0.939, -1.0, -0.939, 1.0, 0.939],
            [0.902, -0.972, 0.972, 0.939, 1.0, -0.939, -1.0, 0.939, 1.0],
            [-0.848, 0.914, -0.914, -1.0, -0.939, 1.0, 0.939, -1.0, -0.939],
            [-0.902, 0.972, -0.972, -0.939, -1.0, 0.939, 1.0, -0.939, -1.0],
            [0.848, -0.914, 0.914, 1.0, 0.939, -1.0, -0.939, 1.0, 0.939],
            [0.902, -0.972, 0.972, 0.939, 1.0, -0.939, -1.0, 0.939, 1.0]
          ]]]
    }
  }
}
```


References

[1] W. Altmannshofer and P. Stangl, *New physics in rare B decays after Moriond 2021*, *Eur. Phys. J. C* **81** (2021) 952, [2103.13370].

[2] J. R. Magnus and H. Neudecker, *The elimination matrix: Some lemmas and applications*, *SIAM Journal on Algebraic Discrete Methods* **1** (1980) 422–449.

[3] J. Aebischer et al., *WCxf: an exchange format for Wilson coefficients beyond the Standard Model*, *Comput. Phys. Commun.* **232** (2018) 71–83, [1712.05298].

[4] C. Degrande, G. Durieux, F. Maltoni, K. Mimasu, E. Vryonidou and C. Zhang, *Automated one-loop computations in the standard model effective field theory*, *Phys. Rev. D* **103** (2021) 096024, [2008.11743].

[5] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer et al., *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, *JHEP* **07** (2014) 079, [1405.0301].

[6] D. M. Straub, *flavio: a Python package for flavour and precision phenomenology in the Standard Model and beyond*, 1810.08132.