Computer Physics Communications 180 (2009) 2698–2716

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

40th Anniversary Issue

# Automated generation of lattice QCD Feynman rules ☆

A. Hart [a,*], G.M. von Hippel [b], R.R. Horgan [c], E.H. Müller [a]

[a] SUPA, School of Physics and Astronomy, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, UK
[b] Theory Group, Deutsches Elektronen-Synchrotron DESY, Platanenallee 6, 15738 Zeuthen, Germany
[c] DAMTP, CMS, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK

ARTICLE INFO

*Article history:*
Received 31 March 2009
Accepted 22 April 2009
Available online 3 May 2009

*PACS:*
11.15.Ha
12.38.Gc

*MSC:*
81-04
81T13
81T15
81T18
81T25
81V05
65S05
41A58

*Keywords:*
Quantum Chromodynamics
QCD
Lattice QCD
Perturbation theory

ABSTRACT

The derivation of the Feynman rules for lattice perturbation theory from actions and operators is complicated, especially for highly improved actions such as HISQ. This task is, however, both important and particularly suitable for automation. We describe a suite of software to generate and evaluate Feynman rules for a wide range of lattice field theories with gluons and (relativistic and/or heavy) quarks. Our programs are capable of dealing with actions as complicated as (m)NRQCD and HISQ. Automated differentiation methods are used to calculate also the derivatives of Feynman diagrams.

**Program summary**

*Program title:* HiPPY, HPsrc
*Catalogue identifier:* AEDX_v1_0
*Program summary URL:* http://cpc.cs.qub.ac.uk/summaries/AEDX_v1_0.html
*Program obtainable from:* CPC Program Library, Queen's University, Belfast, N. Ireland
*Licensing provisions:* GPLv2 (see Additional comments below)
*No. of lines in distributed program, including test data, etc.:* 513 426
*No. of bytes in distributed program, including test data, etc.:* 4 893 707
*Distribution format:* tar.gz
*Programming language:* Python, Fortran95
*Computer:* HiPPy: Single-processor workstations. HPsrc: Single-processor workstations and MPI-enabled multi-processor systems
*Operating system:* HiPPy: Any for which Python v2.5.x is available. HPsrc: Any for which a standards-compliant Fortran95 compiler is available
*Has the code been vectorised or parallelised?:* Yes
*RAM:* Problem specific, typically less than 1 GB for either code
*Classification:* 4.4, 11.5
*Nature of problem:* Derivation and use of perturbative Feynman rules for complicated lattice QCD actions.
*Solution method:* An automated expansion method implemented in Python (HiPPy) and code to use expansions to generate Feynman rules in Fortran95 (HPsrc).
*Restrictions:* No general restrictions. Specific restrictions are discussed in the text.
*Additional comments:* The HiPPy and HPsrc codes are released under the second version of the GNU General Public Licence (GPL v2). Therefore anyone is free to use or modify the code for their own calculations. As part of the licensing, we ask that any publications including results from the use of this code or of modifications of it cite Refs. [1,2] as well as this paper. Finally, we also ask that details of these publications, as well as of any bugs or required or useful improvements of this core code, would be communicated to us.
*Running time:* Very problem specific, depending on the complexity of the Feynman rules and the number of integration points. Typically between a few minutes and several weeks. The installation tests provided with the program code take only a few seconds to run.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
*E-mail address:* a.hart@ed.ac.uk (A. Hart).

0010-4655/$ – see front matter 
doi:10.1016/j.cpc.2009.04.021

*References:*

[1] A. Hart, G.M. von Hippel, R.R. Horgan, L.C. Storoni, Automatically generating Feynman rules for improved lattice eld theories, J. Comput. Phys. 209 (2005) 340–353, doi:10.1016/j.jcp.2005.03.010, arXiv:hep-lat/0411026.

[2] M. Lüscher, P. Weisz, Efficient Numerical Techniques for Perturbative Lattice Gauge Theory Computations, Nucl. Phys. B 266 (1986) 309, doi:10.1016/0550-3213(86)90094-5.



## 1. Introduction

Non-abelian gauge theories are the most important ingredient in our present understanding of elementary particles and their interactions. In particular, quantum chromodynamics (QCD) is now universally believed to be the correct theory of the strong interactions. However, while perturbation theory has been used successfully in describing the scattering of particles by partons, the perturbative series does not converge at hadronic energy scales. Moreover, the phenomena of confinement and the hadronic spectrum are fundamentally beyond the reach of perturbation theory. Therefore, non-perturbative Monte Carlo simulations of lattice-regularised QCD are crucial in order to obtain a full description and understanding of QCD phenomena.

The lattice regularisation with a lattice spacing $a$ does, however, introduce a sharp momentum cutoff at the momentum scale $\pi/a$. Connecting lattice measurements to their continuum counterparts therefore requires renormalisation factors accounting for the excluded high-frequency modes. In particular, renormalisation is needed for QCD matrix elements, and for fixing the bare quark masses to be used in the lattice Lagrangian. Renormalisation is also necessary to determine the running of the strong coupling constant $\alpha_s$ and to relate the lattice regularisation scale $\Lambda_{\rm lat}$ to $\Lambda_{\rm QCD}$. Since the lattice regularisation also introduces discretisation errors, renormalisations are also used to "improve" the lattice action in order to reduce these discretisation errors at a given lattice spacing.

While in some cases the renormalisation constants can be determined non-perturbatively [1,2], results at finite lattice spacing can depend upon the method used (cf. e.g. [3]), and non-perturbative methods do not cope well with operators that mix under renormalisation. For these reasons, lattice perturbation theory plays an important role in determining the renormalisation constants needed to extract continuum predictions from lattice QCD.

Given the breakdown of perturbation theory in low energy QCD, one might doubt whether it could work on the lattice. An argument in favour of its use is given in [4]: since the renormalisation factors may be thought of as compensating for the ultraviolet modes excluded by the lattice regulator, and since for typical lattice spacings $a \lesssim 0.1$ fm, the excluded modes have momenta in excess of 5 GeV, the running QCD coupling $\alpha_s(\pi/a)$ is small enough that perturbation theory should rapidly converge. The wide range of results reviewed in [5,6] show that perturbation theory is useful for a large range of lattice QCD processes. The assumption that non-perturbative effects do not contribute on these short length scales, can be tested directly in some cases by comparing higher order perturbative calculations with Monte Carlo simulations performed at a range of weak couplings [7–12], or by using so-called stochastic techniques [13]. In all these cases, the non-perturbative contributions turned out to be small. Other comparisons, such as [3], cannot distinguish non-perturbative effects from higher-order perturbative corrections.

Lattice perturbation theory therefore provides a reliable, and the only systematically improvable, method for determining the full range of renormalisation constants [5].

As in the continuum, the calculation of lattice Feynman diagrams is a two-stage process. First, the lattice action and any operator insertions are Taylor-expanded in the (bare) coupling constant to give the propagators and vertices that form the Feynman rules (the "vertex expansion" stage). Secondly, these rules are then used to construct and numerically evaluate Feynman diagrams, possibly after some algebraic simplification (the "Feynman diagram evaluation" stage).

The latter task is more complicated than in the continuum due to the presence of Lorentz symmetry violating terms at finite lattice spacing, and on a finite lattice volume also by the more complicated nature of discrete momentum sums as compared to momentum integrals. Diagrams are therefore usually evaluated using numerical routines like Vegas [14,15], or proprietary mathematical packages, possibly after manipulation using other computer algebra packages like Form [16].

The greater difficulty is posed, however, by the task of vertex expansion. Deriving the Feynman results on the lattice is far more complicated than in the continuum for a number of reasons. Firstly, lattice gauge fields are elements of the gauge Lie group itself rather than of its Lie algebra. To obtain the perturbative expansion of the action, we must therefore expand exponentials of non-commuting objects. As a consequence, the Feynman rules even for the simplest lattice action are already much more complicated than their continuum counterparts.

Secondly, modern lattice theories generally contain a large number of additional (renormalisation group irrelevant) terms chosen to improve specific aspects of the Monte Carlo simulation, such as the rate of approach to the continuum or chiral limits of QCD. Since there is no unique prescription for these terms, and the best choice depends on which quantities we are most interested in simulating, a large number of different lattice actions and operators are in use. Subtle though the differences between the lattice formulations may be, each choice provides a completely separate regularisation of QCD with its own set of renormalisation constants and in particular its own lattice Feynman rules. For a long time, the complications of the perturbative expansion have led to the calculations of renormalisation factors lagging far behind new developments in the improvement of lattice theories. In many cases this has restricted the physical predictions that could be obtained from the simulations.

An automated method for deriving lattice Feynman rules for as wide a range of different theories as possible is therefore highly desirable. The vertex expansion should be fast enough not to impose undue constraints on the choice of action. To avoid human error, the user should be able to specify the action in a compact and intuitive manner. Since the evaluation of the Feynman diagrams can be computationally intensive, and will often need to be carried out in parallel on costly supercomputing facilities, parsimony dictates that the Feynman rules should be calculated in advance on a different system, and rendered as machine readable files that can be copied to the supercomputer for the Feynman diagram evaluation stage.

In this paper we describe a pair of software packages for deriving the Feynman rules for arbitrary lattice actions[1] and for evaluating the resulting vertices in a numerical Feynman diagram calculation. Our algorithm is based through our older algorithm [17] on the seminal work of Lüscher and Weisz [18]. A different implementation of the latter has been used in [19–21], and a similar method is employed in [22].

The new feature of the algorithm presented here is that it is capable of expanding not only gluonic actions like the algorithm of [18], and fermionic actions like our algorithm from [17], but also far more complicated multiply-smeared fermionic actions with reunitarisation such as HISQ [23], and that it supports taking advantage of the factorisation inherent in some lattice actions, such as improved lattice formulations of NRQCD [24].

As in [18] and [17], the vertex expansion is performed completely independently of any boundary conditions, allowing for instance, the use of twisted periodic boundary conditions as a gauge-invariant infrared regulator [25,26] or for changing the discrete momentum spectrum in other ways [27].

We have used the software packages described for calculations of the renormalised anisotropy in gauge theories [28,29], to study the mean link in Landau gauge for tadpole improvement [11], to measure the electromagnetic decays of heavy quark systems using NRQCD [30–33], to calculate the radiative corrections to the gluonic action due to Highly Improved Staggered (HISQ) sea quarks [34,35] and the renormalisation of the self energy of heavy quarks using moving NRQCD (mNRQCD) [36].

The code is flexible and can be easily extended, as has already been done for lattice-regularised chiral perturbation theory [37], perturbative calculations in the Schrödinger functional [38,39] and anisotropy calculations [40].

The structure of this paper is as follows. In the next section, we review the basic expansion algorithm described in Ref. [17], outlining some improvements in, for instance, the handling of automatic derivatives and spin matrices. In Section 3, we present novel extensions to the algorithm that are needed to describe complicated fermion actions, including HISQ, NRQCD and mNRQCD.

Section 4 provides details of the implementation of the algorithm in the HiPPy and HPsrc codes and of their installation, testing and use. We make some concluding remarks in Section 5. Technical details are relegated to Appendices A–C.

### 1.1. Licence

The HiPPy and HPsrc codes are released under the second version of the GNU General Public Licence (GPL v2). Therefore anyone is free to use or modify the code for their own calculations.

As part of the licensing, we ask that any publications including results from the use of this code or of modifications of it cite Refs. [17,18] as well as this paper.

Finally, we also ask that details of these publications, as well as of any bugs or required or useful improvements of this core code, would be communicated to us.

## 2. Theoretical background

In this section we describe the algorithms used to give the most efficient, yet generic, implementation of the Feynman rules for lattice actions. These extend the original work of Lüscher and Weisz [18] and developments described in Ref. [17].

### 2.1. Fields on the lattice

We consider a $D$-dimensional hypercubic spacetime lattice with lattice spacing $a$ and extent $L_\mu a$ in the $\mu$-direction:

$$\Lambda = \left\{ (x_1, \ldots, x_D) \in \mathbb{R}^D \;\middle|\; \forall \mu \in \{1, \ldots, D\}\colon \frac{x_\mu}{a} \in \{0, \ldots, L_\mu - 1\} \right\}, \tag{1}$$

where lattice sites are labelled by a vector $\boldsymbol{x} \in \Lambda$. In the following, we will usually set $a = 1$ (a lattice anisotropy can be introduced by rescaling the coupling constants in the action [28]). Let $\boldsymbol{e}_\mu$ be a right-handed orthonormal basis, and $\boldsymbol{e}_{-\mu} \equiv -\boldsymbol{e}_\mu$.

A lattice path consisting of $l$ links starting at site $\boldsymbol{x}$ can be specified by an ordered set of directions given by integers, $s_i \in \{-D, \ldots, -1, 1, \ldots, D\}$:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{s}) \equiv \left\{ \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{s} = [s_0, s_1, \ldots, s_{l-1}] \right\}, \tag{2}$$

with the $j$th point on the path being

$$\boldsymbol{z}_j = \begin{cases} \boldsymbol{x}, & j = 0, \\ \boldsymbol{z}_{j-1} + a\boldsymbol{e}_{s_{j-1}}, & 0 < j \leqslant l, \end{cases} \tag{3}$$

and $\boldsymbol{y} \equiv \boldsymbol{z}_l$.

For periodic boundary conditions, the momentum vectors are

$$\boldsymbol{k} = \frac{2\pi}{a} \left( \frac{\bar{k}_1}{L_1}, \ldots, \frac{\bar{k}_D}{L_D} \right), \quad 0 \leqslant \bar{k}_\mu < L_\mu,\ \bar{k}_\mu \in \mathbb{Z}, \tag{4}$$

and the Fourier expansion of a function $\phi$ is

$$\tilde{\phi}(\boldsymbol{k}) = \sum_{\boldsymbol{x}} e^{-i\boldsymbol{k}\cdot\boldsymbol{x}} \phi(\boldsymbol{x}), \qquad \phi(\boldsymbol{x}) = \frac{1}{V} \sum_{\boldsymbol{k}} e^{i\boldsymbol{k}\cdot\boldsymbol{x}} \tilde{\phi}(\boldsymbol{k}), \tag{5}$$

where $V = \prod_\mu L_\mu$ is the lattice volume.

[1] We shall use the term "action" so as to include measurement operators here and in the following.

Twisted boundary conditions [41] provide a useful gauge-invariant infrared regulator in perturbative calculations [18]. These change the momentum spectrum, converting colour factors into "twist matrices" associated with momenta interstitial to the reciprocal lattice (with the introduction of an additional quantum number, "smell", for fermions [11,42,43]). The HPsrc code fully supports such boundary conditions but for simplicity we only discuss periodic boundary conditions in this paper.

Following Ref. [18], we denote the gauge field associated with a link as $U_{\mu>0}(\boldsymbol{x}) \in SU(N)$, and define $U_{-\mu}(\boldsymbol{x}) = U_\mu^\dagger(\boldsymbol{x} - a\boldsymbol{e}_\mu)$. The gauge potential $A_\mu \in \mathrm{alg}(SU(N))$ associated with the midpoint of the link is defined through

$$U_{\mu>0}(\boldsymbol{x}) = \exp\left(agA_\mu\left(\boldsymbol{x} + \frac{a}{2}\boldsymbol{e}_\mu\right)\right) = \sum_{r=0}^{\infty} \frac{(agA_\mu(\boldsymbol{x} + \frac{a}{2}\boldsymbol{e}_\mu))^r}{r!} \tag{6}$$

where $g$ is the bare coupling constant. In terms of the anti-Hermitian generators of $SU(N)$,

$$A_\mu = A_\mu^a T_a, \qquad [T_a, T_b] = -f_{abc}T_c, \qquad \mathrm{Tr}(T_a T_b) = -\frac{1}{2}\delta_{ab}. \tag{7}$$

Quark fields $\psi(\boldsymbol{x})$ transform according to the representation chosen for the generators $T_a$, which we take to be the fundamental representation (other choices will affect the colour factors, but not the structure of our algorithm).

### 2.2. Perturbative expansion of Wilson lines

The Wilson line $L(\boldsymbol{x}, \boldsymbol{y}, U)$ associated with the lattice path $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{s})$ is a product of links

$$L(\boldsymbol{x}, \boldsymbol{y}, U) = \prod_{i=0}^{l-1} U_{s_i}(\boldsymbol{z}_i) = \prod_{i=0}^{l-1} \exp\left[\mathrm{sgn}(s_i) agA_{|s_i|}\left(\boldsymbol{z}_i + \frac{a}{2}\boldsymbol{e}_{s_i}\right)\right]. \tag{8}$$

As all actions and operators can be written as sums of Wilson lines (possibly terminated by fermion fields), our goal is to efficiently expand $L$ in terms of the gauge potential in momentum space:

$$L(\boldsymbol{x}, \boldsymbol{y}; A) = \sum_r \frac{(ag)^r}{r!} \sum_{\boldsymbol{k}_1, \mu_1, a_1} \cdots \sum_{\boldsymbol{k}_r, \mu_r, a_r} \tilde{A}_{\mu_1}^{a_1}(\boldsymbol{k}_1) \ldots \tilde{A}_{\mu_r}^{a_r}(\boldsymbol{k}_r) V_r(\boldsymbol{k}_1, \mu_1, a_1; \ldots; \boldsymbol{k}_r, \mu_r, a_r). \tag{9}$$

The vertex functions $V_r$ factorise as

$$V_r(\boldsymbol{k}_1, \mu_1, a_1; \ldots; \boldsymbol{k}_r, \mu_r, a_r) = C_r(a_1, \ldots, a_r) Y_r^{\mathcal{L}}(\boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r) \tag{10}$$

with a momentum- and path-independent Clebsch–Gordan (colour) factor $C_r$

$$C_r(a_1, \ldots, a_r) = \prod_{i=1}^{r} T_{a_i}. \tag{11}$$

It is therefore more efficient to calculate just the expansion of the reduced vertex functions, $Y_r^{\mathcal{L}}$ (with an appropriate description of the colour trace structure where ambiguous — see Appendix B of Ref. [17] for further details). The reduced vertex function can be written as a sum of terms, each of which contains an exponential. For convenience, we will call each term a "monomial":

$$Y_r^{\mathcal{L}}(\boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r) = \sum_{n=1}^{n_r(\{\mu\})} f_n^{(r,\{\mu\})} \exp\left(\frac{i}{2}\sum_{j=1}^{r} \boldsymbol{k}_j \cdot \boldsymbol{v}_{n,j}^{(r,\{\mu\})}\right), \tag{12}$$

where for each combination of $r$ Lorentz indices we have $n_r$ terms, each with an amplitude $f$ and locations $\boldsymbol{v}$ of the $r$ factors of the gauge potential, which are drawn from the locations of the midpoints of the links in the path $\mathcal{L}$. To avoid floating point ambiguities, we express the components of all position $D$-vectors as integer multiples of $\frac{a}{2}$ (accounting for the factor of $\frac{1}{2}$ in the exponent).

In the HPsrc code, we use the convention that all momenta flow into the vertex, so $\sum_{i=1}^{r} \boldsymbol{k}_i = 0$.

Eq. (12) makes clear that the number of monomials depends not just on the number of gluons $r$, but also on the choice of Lorentz indices $\{\mu\}$, and that each monomial has a different amplitude and set of $r$ positions. For clarity of presentation, we will, however, suppress these additional arguments in later expressions (notably Eqs. (17), (21), (26), (44), (45)).

#### 2.2.1. Implementation notes

The generation of the Feynman rules for generic momenta thus reduces to a calculation of the amplitudes $f$ and locations $\boldsymbol{v}$ of the monomials that build up the various reduced vertices $Y_r$.

This is all carried out in the HiPPy code, a description of which can be found in Section 4 of Ref. [17]. The amplitudes and locations defining each monomial are encoded as an instance of the class `Entity`, and the collection of monomials that make up the reduced vertices is encoded as an instance of class `Field`. The data structures have been chosen to ensure that equivalent monomials are combined to minimise the size of the reduced vertex description.

Once expanded, the monomials required for the reduced vertices at each order are written to disk as a text file.

The HPsrc code reads these (previously generated) vertex files at runtime. For given momenta $\{\boldsymbol{k}\}$, Lorentz indices $\{\mu\}$ and colour indices, the $Y_r$ are constructed as given in Eq. (12), which is then multiplied by the appropriate Clebsch–Gordan colour factor(s) to form the (Euclidean) Feynman rule, $-V_r$.

### 2.3. Realistic actions: the fermion sector

Realistic lattice fermion and gauge actions require some refinements to this generic description. We begin with the fermion sector. The most general gauge- and translation-invariant action can be written as

$$S_F(\psi, U) = \sum_{\boldsymbol{x}} \sum_{\mathcal{W}} h_{\mathcal{W}} \bar{\psi}(\boldsymbol{x}) \Gamma_{\mathcal{W}} W(\boldsymbol{x}, \boldsymbol{y}, U) \psi(\boldsymbol{y}) \tag{13}$$

and consists of Wilson lines $W$ defined by open paths $\mathcal{W}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{s})$, each carrying an associated coupling constant $h_{\mathcal{W}}$ and a spin matrix $\Gamma_{\mathcal{W}}$ (possibly the identity).

Using the convention that all momenta flow into the vertex, the perturbative expansion is

$$S_F(\psi, A) = \sum_r \frac{g^r}{r!} \sum_{\boldsymbol{k}_1, \mu_1, a_1} \cdots \sum_{\boldsymbol{k}_r, \mu_r, a_r} \tilde{A}^{a_1}_{\mu_1}(\boldsymbol{k}_1) \ldots \tilde{A}^{a_r}_{\mu_r}(\boldsymbol{k}_r) \sum_{\boldsymbol{p}, \boldsymbol{q}, b, c} \tilde{\bar{\psi}}^{b}(\boldsymbol{p}) V_{F,r}(\boldsymbol{p}, b; \boldsymbol{q}, c; \boldsymbol{k}_1, \mu_1, a_1; \ldots; \boldsymbol{k}_r, \mu_r, a_r) \tilde{\psi}^{c}(\boldsymbol{q}). \tag{14}$$

The Euclidean Feynman rule for the $r$-point gluon–fermion–anti-fermion vertex is $-g^r V_{F,r}$, where the symmetrised vertex is:

$$V_{F,r}(\boldsymbol{p}, b; \boldsymbol{q}, c; \boldsymbol{k}_1, \mu_1, a_1; \ldots; \boldsymbol{k}_r, \mu_r, a_r) = \frac{1}{r!} \sum_{\sigma \in \mathcal{S}_r} \sigma \cdot C_{F,r}(b, c; a_1, \ldots, a_r) \sigma \cdot Y_{F,r}(\boldsymbol{p}, \boldsymbol{q}; \boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r), \tag{15}$$

where $\mathcal{S}_r$ is the permutation group of $r$ objects and $\sigma \in \mathcal{S}_r$ is applied to the gluonic variables, $\{\boldsymbol{k}\}$, $\{\mu\}$ and $\{a\}$. The normalisation factor of $r!$ for this is additional to the $r!$ factor arising from the Taylor expansion of the exponential in Eq. (14). The reduced vertex $Y_{F,r} = \sum_{\mathcal{W}} h_{\mathcal{W}} Y^{\mathcal{W}}_{F,r}$ is the sum of contributions from paths $\mathcal{W}$.

In most cases the Clebsch–Gordan colour factor is the matrix element:

$$C_{F,r}(b, c; a_1, \ldots, a_r) = (T_{a_1} \ldots T_{a_r})_{bc}, \tag{16}$$

and the reduced vertex function has the structure:

$$Y_{F,r}(\boldsymbol{p}, \boldsymbol{q}; \boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r) = \sum_{n=1}^{n_r(\{\mu\})} \Gamma_n f_n \exp\left(\frac{i}{2}\left(\boldsymbol{p} \cdot \boldsymbol{x} + \boldsymbol{q} \cdot \boldsymbol{y} + \sum_{j=1}^{r} \boldsymbol{k}_j \cdot \boldsymbol{v}_{n,j}\right)\right), \tag{17}$$

where we understand $\Gamma_n \equiv \Gamma_{r,\{\mu\},n}$. Cases with more complicated colour structures do arise, for example the use of traceless field strengths in QCD. Such structures are accommodated in the codes; monomials with different colour structures are distinguished using "pattern lists" (discussed in Appendix B of Ref. [17]) and appropriate colour factors are applied to each when the Feynman rule is constructed.

As there are no permutation symmetries in $C_{F,r}$, there is no advantage to carrying out any symmetrisation in the HiPPy expansion code. In the HPsrc code, symmetrisation of the Feynman rule shown in Eq. (15) carries a potentially significant computational overhead: the reduced vertices must be calculated afresh for each permutation. Not all such permutations may be needed because symmetries of a Feynman diagram can reduce the number of distinct contributions to its value from the terms in Eq. (15). For this reason, symmetrisation is not carried automatically in the HPsrc code and the user must therefore explicitly construct all permutations requiring a different calculation from Eq. (15), applying the appropriate normalisation factor.

### 2.4. Realistic actions: the gluon sector

A typical gluonic action is

$$S(\psi, U) = \sum_{\boldsymbol{x}} \sum_{\mathcal{P}} c_{\mathcal{P}} \operatorname{Re} \operatorname{Tr}\big[P(\boldsymbol{x}, \boldsymbol{x}, U)\big], \tag{18}$$

built of Wilson loops $P$ defined by closed paths $\mathcal{P}(\boldsymbol{x}, \boldsymbol{x}; \boldsymbol{s})$, each with coupling constant $c_{\mathcal{P}}$. The perturbative action is

$$S_G(A) = \sum_r \frac{g^r}{r!} \sum_{\boldsymbol{k}_1, \mu_1, a_1} \cdots \sum_{\boldsymbol{k}_r, \mu_r, a_r} \tilde{A}^{a_1}_{\mu_1}(\boldsymbol{k}_1) \ldots \tilde{A}^{a_r}_{\mu_r}(\boldsymbol{k}_r) V_{G,r}(\boldsymbol{k}_1, \mu_1, a_1; \ldots; \boldsymbol{k}_r, \mu_r, a_r). \tag{19}$$

The Euclidean Feynman rule for the $r$-point gluon vertex function is $(-g^r V_{G,r})$, and the vertex $V_{G,r}$ is [18]

$$V_{G,r}(\boldsymbol{k}_1, \mu_1, a_1; \ldots; \boldsymbol{k}_r, \mu_r, a_r) = \frac{1}{r!} \sum_{\sigma \in \mathcal{S}_r} \sigma \cdot C_{G,r}(a_1, \ldots, a_r) \sigma \cdot Y_{G,r}(\boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r). \tag{20}$$

The reduced vertex $Y_{G,r} = \sum_{\mathcal{P}} c_{\mathcal{P}} Y^{\mathcal{P}}_{G,r}$ is the sum of contributions from paths $\mathcal{P}$. As before, the $(r!)$ factor normalises the symmetrisation. $Y^{\mathcal{P}}_{G,r}$ can be expanded as

$$Y^{\mathcal{P}}_{G,r}(\boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r) = \sum_{n=1}^{n_r} f_n \exp\left(\frac{i}{2} \sum_i \boldsymbol{k}_i \cdot \boldsymbol{v}_{n,i}\right). \tag{21}$$

In most cases we expect the lattice action to be real. Thus, for every monomial $(f_n; \{\boldsymbol{v}_{n,i}\})$ in Eq. (21), there must be a corresponding term $((-1)^r f_n^*; \{-\boldsymbol{v}_{n,i}\})$. We can therefore speed up the evaluation of the Feynman rules by removing the latter term, and replacing the exponentiation in Eq. (21) with "cos" for $r$ even, and with "$i$ sin" for $r$ odd. This can either be done by recognising conjugate contours in the action (e.g., $S = \frac{1}{2}\operatorname{Tr}[P + P^\dagger]$) and expanding only one, or by attaching a flag to each monomial to signal whether its complex conjugate has already been removed.

If in addition the action has the form (18) with a single trace in the fundamental representation, the colour factors are

$$C_{G,r}(a_1,\ldots,a_r)=\frac{1}{2}\left[\mathrm{Tr}(T_{a_1}\ldots T_{a_r})+(-1)^r\,\mathrm{Tr}(T_{a_r}\ldots T_{a_1})\right] \tag{22}$$

which has a number of permutation symmetries:

$$\sigma\cdot C_{G,r}=\chi_r(\sigma)C_{G,r},\quad \text{where } \chi_r(\sigma)=\begin{cases}1 & \text{for } \sigma \text{ a cyclic permutation,}\\ (-1)^r & \text{for } \sigma \text{ the inversion.}\end{cases} \tag{23}$$

There is thus a great advantage in carrying out some of the symmetrisation in Eq. (20) at the expansion stage in the HiPPy code. Many of the extra monomials generated by symmetrising over subgroup $\mathcal{Z}_r$ (generated by cyclic permutations and inversion) are equivalent and can be combined in the HiPPy code, significantly reducing the number of exponentiation operations required to construct the partially-symmetrised $Y'_{G,r}$:

$$V_{G,r}(\boldsymbol{k}_1,\mu_1,a_1;\ldots;\boldsymbol{k}_r,\mu_r,a_r)=\sum_{\sigma\in\mathcal{S}_r/\mathcal{Z}_r}\sigma\cdot C_{G,r}(a_1,\ldots,a_r)\sigma\cdot Y'_{G,r}(\boldsymbol{k}_1,\mu_1;\ldots;\boldsymbol{k}_r,\mu_r), \tag{24}$$

$$Y'_{G,r}=\sum_{\substack{\mathcal{P}\\ \sigma\in\mathcal{Z}_r}}c_{\mathcal{P}}\chi_r(\sigma)\sigma\cdot Y^{\mathcal{P}}_{G,r}. \tag{25}$$

The $\chi_r(\sigma)$ factors go into the amplitudes of the new monomials coming from the partial symmetrisation.

The number of symmetrisation steps remaining to be carried out in the HPsrc code is the number of cosets in $\mathcal{S}_r/\mathcal{Z}_r$ (one for $r\leqslant 3$, three for $r=4$, twelve for $r=5$, etc.). These symmetrisation steps (and the normalisation) are carried out automatically in the HPsrc gluon vertex modules.

### 2.5. Diagram differentiation

In many cases, such as when computing wavefunction renormalisation constants, one needs to calculate the derivative of a Feynman diagram with respect to one or more momenta. Whilst derivatives can be computed numerically using an appropriate local difference operator, such differencing schemes are frequently numerically unstable and require computing the Feynman diagram multiple times. Automatic differentiation methods [44] are a stable and cost-saving alternative.

We can easily construct the differentiated Feynman vertex using Eq. (12). If we want to differentiate with respect to momentum component $q_\nu$, we first construct a rank $r$ object $\boldsymbol{\tau}=[\tau_1,\ldots,\tau_r]$ which represents the proportion of momentum $\boldsymbol{q}$ in each leg of the Feynman diagram. Momentum conservation dictates $\sum_i\tau_i=0$. For instance, for a gluon 3-point function with incoming momenta $(\boldsymbol{p},-\boldsymbol{p}+2\boldsymbol{q},-2\boldsymbol{q})$, we would have $\boldsymbol{\tau}=[0,2,-2]$. The differentiated vertex is

$$\frac{d}{dq_\nu}Y^{\mathcal{L}}_r(\boldsymbol{k}_1,\mu_1;\ldots;\boldsymbol{k}_r,\mu_r)=\sum_{n=1}^{n_r}\frac{if_n}{2}\left(\sum_{j=1}^{r}\tau_j v_{n,j;\nu}\right)\exp\left(\frac{i}{2}\sum_{j=1}^{r}\boldsymbol{k}_j\cdot\boldsymbol{v}_{n,j}\right) \tag{26}$$

and so on for higher derivatives. We may therefore simultaneously calculate as many differentials as we need for the cost of just one exponentiation. If this momentum expansion is placed into an appropriate data structure for which appropriately overloaded operations have been defined, it is straightforward to create the Taylor series for a Feynman diagram by simply multiplying the vertex factors together. We use a slightly modified version of the TaylUR package [45,46] to do this, which encodes the Taylor series expansion in the HPsrc Fortran code as a derived type `taylor`, for which all arithmetic operations and elementary functions have been overloaded so as to respect Leibniz's and Faà di Bruno's rules for higher derivatives of products and functions.

In calculations that require only certain higher-order derivatives, the Taylor multiplication can be significantly sped up by defining a mask that only propagates certain terms in the Taylor expansion. This has not, however, been implemented in the distributed version of the code.

### 2.6. Spin algebra

The Feynman rules for fermions contain spin matrices (which may be Pauli matrices, e.g., for NRQCD, or Dirac matrices for relativistic fermions). We can expand a generic spin matrix $W$ using a basis $\{\Gamma_i\}$:

$$W=\sum_{i\in I(W)}w_i\Gamma_i. \tag{27}$$

The product of any two spin basis matrices $\Gamma_i$ and $\Gamma_j$ is another basis matrix (with label $n_{ij}$) times a phase:

$$\Gamma_i\Gamma_j=\phi_{ij}\Gamma_{n_{ij}}. \tag{28}$$

We choose the basis so that these phases $\phi_{ij}$ are real. Where another convention is desired, the amplitudes $w_i$ need to be adjusted appropriately.

For Pauli matrices, we use a basis $\{\Gamma_i\}=\{1,i\sigma_k\}$ and $I(W)\subseteq\{0,\ldots,3\}$, giving:

$$i\sigma_j.i\sigma_k=\begin{cases}1, & j=k,\\ \epsilon_{jkm}i\sigma_m, & j\neq k.\end{cases} \tag{29}$$

For the (Euclidean) Dirac matrices, we choose $\{\Gamma_i\} = \{1, \gamma_\mu, \sigma_{\mu\nu}, \gamma_5\gamma_\mu, \gamma_5\}$ and $I(W) \subseteq \{0, \ldots, 15\}$. We define $\gamma_5 \equiv \gamma_1\gamma_2\gamma_3\gamma_4$ and note the definition here $\sigma_{\mu\nu} \equiv \frac{1}{2}[\gamma_\mu, \gamma_\nu]$. The multiplication for the basic $\gamma$ matrices is thus

$$\gamma_\mu\gamma_\nu = \begin{cases} 1, & \mu = \nu, \\ \sigma_{\mu\nu}, & \mu \neq \nu. \end{cases} \tag{30}$$

We can thus write the product of two general spin matrices as

$$WW' = \left(\sum_{j\in I(W)} w_j\Gamma_j\right)\left(\sum_{k\in I(W')} w'_k\Gamma_k\right) = \sum_{i\in I(WW')}\left(\sum_{\substack{j\in I(W),k\in I(W') \\ n_{jk}=i}} w_j w'_k\phi_{jk}\right)\Gamma_i, \tag{31}$$

where $I(WW') = \{n_{ij} \mid i \in I(W), j \in I(W')\}$.

We use this implicit representation of the spin matrices. Matrix operations on explicit Dirac matrices take $\mathcal{O}(4^3)$ operations and introduce additional rounding errors. Eq. (31), by contrast, needs $|I(W)| \times |I(W')|$ multiplications, depending on how many basis matrices are needed to describe $W$ and $W'$. If $|I(W)|, |I(W')| < 8$, the latter method is more efficient and this is almost always the case.

There are greater gains when inverting spin matrices of the form

$$S^{-1} = a_0 1 + \sum_{\mu=1}^{4} a_\mu\gamma_\mu, \tag{32}$$

as we might do when obtaining the propagator of a relativistic quark (for NRQCD, the propagator is spin diagonal in the Pauli matrices and trivial to invert). The inverse is

$$S = b_0 1 - \sum_{\mu=1}^{4} b_\mu\gamma_\mu, \qquad b_i = a_i\left(a_0^2 - \sum_{\mu=1}^{4} a_\mu^2\right)^{-1}, \tag{33}$$

which is far more efficient than inverting a $4 \times 4$ matrix. Inversion of a general spin matrix (not of the form (32)) is less efficient with an implicit representation, but this is irrelevant in most perturbative calculations.

Since all basis matrices except the identity are traceless, taking the trace of a spin matrix is a free operation in the implicit representation.

#### 2.6.1. Implementation notes

In the HiPPy code, spin basis matrices are associated with monomials using an appropriate integer $i$, which is part of the `Entity` data structure. When terms are multiplied together, the factors $\phi_{jk}$ are absorbed into the amplitude $f$ of the resulting monomial.

In the HPsrc code, we represent a spin matrix $W$ as a defined type, `spinor`, which is encoded as a double array

$$(n; i_1, \ldots, i_n; w_1, \ldots, w_n) \equiv \sum_{k=1}^{n} w_k\Gamma_{i_k}. \tag{34}$$

It turns out to be significantly more efficient for the order of terms in this array to not necessarily match a standard order of basis elements $\{\Gamma_i\}$, hence the use of the index array $i_k$. In particular, this allows us to omit basis elements with zero coefficient.

Arithmetic operations have been overloaded to act appropriately on objects of this type, including implementation of the multiplication table. During inversion, an additional function argument, `short_spinor`, is used to employ the more efficient expression in Eq. (33).

## 3. Even more realistic fermionic actions

Section 2 summarised the general method that was described in Ref. [17]. In general, fermionic actions are much more complicated than gluonic actions and several algorithmic improvements are needed to efficiently calculate the associated Feynman rules. We stress that by efficient we mean speed-ups of at least an order of magnitude.

The algorithms described in this section can be used independently or together, with the choice configurable at runtime of the code. All of these features have also been implemented using Taylor-valued variables (as per Section 2.5) to provide automatic differentiation of Feynman rules.

### 3.1. Summands and factors

In many cases an action has a block-like structure that we can exploit to make the evaluation of the reduced vertices more efficient. This is particularly useful in the case of NRQCD and mNRQCD actions, which can be heuristically written as $\bar{\psi}(1 - ABCA)\psi$. Such an action can be expanded directly in the HiPPy code but the extremely large number of monomials makes this is inefficient (or impossibly slow and memory-hungry). It is clear why: there is often little scope for monomial compression between blocks. For instance, in (m)NRQCD, blocks $AB$ and $CA$ live on different timeslices of the lattice, and no compression is possible when combining them.

Instead, we recognise that the blocks are combined in a gauge covariant manner, so that in $AB$, for instance, each contour in $B$ starts where a contour in $A$ finishes. Summing over the start/end location of each block we obtain a convolution of terms and can use the convolution theorem to construct the overall reduced vertex (i.e. the momentum-space Fourier transform) from those of the individual blocks.

We refer to the action as being a sum over terms that we call “summands”. In the above example, there are two. Each summand is the convolution of a number of “factors”, with one factor in the first summand and four in the second.

**Table 1**
The elements $P$ in the set of ordered partitions, $P^{(r)}$, of the ordered set of the first $r$ integers, $\{1, 2, \ldots, r\}$, for $r = 1$, 2 and 3.

| $r$ | $P$ | $\lvert P\rvert$ |
|---|---|---|
| 1 | $\{\{1\}\}$ | 1 |
| 2 | $\{\{1, 2\}\}$ | 1 |
| | $\{\{1\}, \{2\}\}$ | 2 |
| 3 | $\{\{1, 2, 3\}\}$ | 1 |
| | $\{\{1, 2\}, \{3\}\}$ | 2 |
| | $\{\{1\}, \{2, 3\}\}$ | 2 |
| | $\{\{1\}, \{2\}, \{3\}\}$ | 3 |

The overall reduced vertex $Y_{F,r}$ is the sum of the reduced vertices for each summand. For each summand, $Y_{F,r}$ is calculated by combining the reduced vertices $Y^{(k)}_{F,r}$ for each of $N$ factors that make up that summand, $k = 1 \ldots N$. We generate these by expanding each factor of the action separately in the HiPPy code, with the convolution then carried out in the HPsrc code.

Here we give the method for constructing $Y_{F,r}$ for a summand with $N$ factors for general $r$. Expressions for specific $r = 0 \ldots 3$ are given in Appendix A.

In giving the general expression, we first establish some useful notation. Consider the ordered set of the first $r$ integers: $\{1, 2, \ldots, r\}$. We can form an ordered partition of this set: $\{P_1, P_2, \ldots, P_z\}$, where the cardinality $z \equiv |\{\ldots\}|$ is the number of elements in the partition. The set of all such partitions we denote $P^{(r)}$. For instance, the partitions $P^{(r)}$ for $r = 1$, 2, 3 are shown in Table 1. Note that we do not consider unordered partitions (e.g., $\{\{1, 3\}, \{2\}\}$) because the gauge fields are explicitly ordered in the paths (Wilson lines) making up the action.

The general reduced vertex for a summand with $N$ factors is then:

$$Y_{F,r}(\boldsymbol{p}, \boldsymbol{q}; \boldsymbol{k}_1, \mu_1; \ldots; \boldsymbol{k}_r, \mu_r) = \sum_{P \in P^{(r)}} \sum_{1 \leqslant n_1 < n_2 < \cdots < n_{|P|} \leqslant N} \left\{ \prod_{Q \in P} \left[ \left( \prod_{k=n_{i-1}+1}^{n_i - 1} Y^{(k)}_{F,0}(\boldsymbol{p}_i, -\boldsymbol{p}_i) \right) \times Y^{(n_i)}_{F,|Q|}(\boldsymbol{p}_i, -\boldsymbol{p}_{i+1}; \boldsymbol{k}_{Q_1}, \mu_{Q_1}; \ldots; \boldsymbol{k}_{Q_{|Q|}}, \mu_{Q_{|Q|}}) \right] \left( \prod_{k=n_{|P|}+1}^{N} Y^{(k)}_{F,0}(-\boldsymbol{q}, \boldsymbol{q}) \right) \right\} \tag{35}$$

where $i = 1 \ldots |P|$ is the position of element $Q$ in the ordered set $P$ and $n_0 = 0$. Momentum is conserved in the diagram, so $\boldsymbol{p}_1 = \boldsymbol{p}$ and subsequent $\boldsymbol{p}_{i+1} = \boldsymbol{p}_i + \boldsymbol{k}_Q$ (with $i$ defined as above). Here $\boldsymbol{k}_Q$ refers to the summed momenta for the set $Q$ (e.g. $\boldsymbol{k}_{\{1,2\}} \equiv \boldsymbol{k}_1 + \boldsymbol{k}_2$), implying $\boldsymbol{p}_{|P|} = -\boldsymbol{q}$.

### 3.2. Two-level actions

Fermion actions often use fattened links to reduce discretisation errors in numerical simulations. By fattening or smearing a link, we will, in general, end up with an $N \times N$ complex colour matrix $M$, which can be expanded in powers of the gauge field $A_\mu$. It is often the case that this matrix is reunitarised by projecting it back onto the gauge group $SU(N)$ or, more usually, simply back onto the related group $U(N)$. Fattened links can then be further fattened, in an iterative procedure. An example of this is the HISQ action.

To complement these numerical simulations, we need to do perturbative calculations using the same actions. We confine our attention here to the HISQ action (and simpler variants of the same form, for testing), with an iterated, two-level smearing procedure with an intermediate reunitarisation:

$$U^{\mathrm{HISQ}} = (F_{\mathrm{ASQ}'} \circ P_{U(3)} \circ F_{\mathrm{FAT7}})[U] \tag{36}$$

where $U = \exp(gA)$ is the unsmeared gauge field, $P_{U(3)}$ denotes the polar projection onto $U(3)$ (as used in simulations, and *not* $SU(3)$ [47]), and the FAT7 and ASQ′ (a slightly modified version of ASQ) smearings are defined in Ref. [23].

Straightforward application of the expansion algorithm above is theoretically possible but practically unfeasible: the number of monomials is huge and the memory requirements of the HiPPy code quickly become excessive. We get around this by taking advantage of the two-level structure inherent in the definition of the action and that the intermediate reunitarisation. This allows us to express the partially-smeared gauge field as a member of the associated gauge Lie algebra, allowing us to split the derivation and subsequent application of the Feynman rules into two steps.

In the first step, the Feynman rules for the outer (or "top") layer, the ASQ′ action, are derived in the same way as before. Representing the reunitarised (and so far uncalculated) FAT7R smeared links by a new, Lie-algebra-valued gauge potential, $B_\mu$:

$$U^{\mathrm{FAT7R}}_\mu(x) = (P_{U(3)} \circ F_{\mathrm{FAT7}})[U_\mu] = e^{B_\mu(x + \frac{1}{2}\hat{\mu})}, \tag{37}$$

we can use the HiPPy code to expand the ASQ′ action in terms of $B_\mu$ as before. We obtain similar position-space contributions

$$V_r = \frac{g^r}{r!} \sum_i f^{\mathrm{ASQ}'}_{r;i} \bar{\psi}(x_{r;i}) B_{\mu_1}(v_{r;i,1}) \cdots B_{\mu_r}(v_{r;i,r}) \Gamma_{r;i} \psi(y_{r;i}) \tag{38}$$

that Fourier transform to give monomials of the usual form.

To complete the derivation of the HISQ Feynman rules, we also need to know the expansion of $B_\mu$ in terms of the original gauge potential $A_\mu$. To obtain this, we write inner (or "bottom") layer, the FAT7-smeared link, as $F_{\mathrm{FAT7}}[U] = M = HW$, where $H^\dagger = H$ and $W \in U(3)$ (we suppress Lorentz and lattice site indices in the following). We again use the HiPPy code to obtain an expansion

$$M = c[\mathbf{1} + a_\mu A_\mu + a_{\mu\nu} A_\mu A_\nu + \cdots], \tag{39}$$

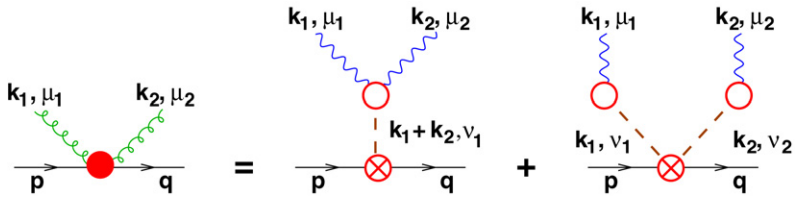


**Fig. 1.** A graphical representation of the reduced vertex $Y_{F,r}$ for a two-level action, with $r = 2$ as in Eq. (B.2). Solid circles represent the $Y_{F,r}$, whilst crossed and open circles represent $Z_{F,r}$ and $X_{F,r}$ respectively. Top-level, fattened gluons $B_\mu$ are represented by dashed (brown) lines, whilst bottom-level, unfattened gluons $A_\mu$ are shown as wavy (blue) lines. The two sub-diagrams represent the two partition contributions listed in Eq. (B.2). All momenta are incoming to the vertex. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

where $c$ is a normalisation constant which, for simplicity in the text, we assume has been rescaled to unity. The notation here is, for instance,

$$a_{\mu\nu} A_\mu A_\nu = \sum_{x,y} a_{\mu\nu}(x,y) A_\mu\left(x + \frac{1}{2}\hat{\mu}\right) A_\nu\left(y + \frac{1}{2}\hat{\nu}\right). \tag{40}$$

Then unitarity of $W$ implies that $R \equiv MM^\dagger = H^2$ and hence $W = R^{-1/2}M$ using the expansion

$$R^{-1/2} = \left(1 + (R-1)\right)^{-1/2} = 1 - \frac{1}{2}(R-1) + \frac{3}{8}(R-1)^2 + \cdots. \tag{41}$$

Rearranging the result as $W = \exp(B)$, i.e.

$$B = \log(W) = (W-1) - \frac{1}{2}(W-1)^2 + \cdots \tag{42}$$

finally yields the desired expansion of $B$. These formulæ are implemented in this form in the HiPPy code.

Given this, we can now numerically reconstruct the HISQ Feynman rules for any given set of momenta from Eq. (38) by a convolution of the ASQ′ Feynman rules of Eq. (38) with the expansion of $B_\mu$ in terms of $A_\mu$, summing up all the different ways in which the gluons $A_\mu$ going into the vertex could have come from the fields $B_\mu$ appearing in Eq. (38).

In more detail, we now separately have the expansions of the ASQ′ action in terms of the fattened gauge fields $B$, and of $B$ in terms of the unfattened gauge field $A$. To obtain the correct reduced vertex, we must find all the ways that we can get unfattened gluons of the correct Lorentz polarisations ("directions"). In doing this, we bear in mind that $B_\mu$ contains, in principle, gauge fields $A_\nu$ in all directions and not just $\nu = \mu$. Below we give an expression for the reduced vertex for general $r$. Explicit formulæ for $r \leqslant 3$, as implemented in the HPsrc code, are given in Appendix B. Using the partitions $P^{(r)}$ as before, the reduced vertex for a two-level action is:

$$Y_{F,r}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\ldots;\boldsymbol{k}_r,\mu_r) = \sum_{P\in P^{(r)}} \sum_{\nu_1,\ldots,\nu_{|P|}} Z_{F,|P|}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_{P_1},\nu_1;\ldots;\boldsymbol{k}_{P_{|P|}},\nu_{|P|}) \prod_{Q\in P} X^{\nu_i}_{F,|Q|}(\boldsymbol{k}_{Q_1},\mu_{Q_1};\ldots;\boldsymbol{k}_{Q_{|Q|}},\mu_{Q_{|Q|}}), \tag{43}$$

where $i = 1\ldots|P|$ is again the position of element $Q$ in the ordered set $P$. In the algebra reduced vertex $X$, the momenta $\boldsymbol{k}_{Q_i}$ are each one of the arguments to the overall reduced vertex $Y$. As before, in the field reduced vertex $Z$, $\boldsymbol{k}_{P_j}$ refers to the summed momenta for the partition $P_j$.

In Fig. 1 we represent this expansion graphically for $r = 2$. Solid circles represent the $Y_{F,r}$, whilst crossed and open circles represent $Z_{F,r}$ and $X_{F,r}$ respectively. Top-level, fattened gluons $B_\mu$ are represented by dashed (brown) lines, whilst bottom-level, unfattened gluons $A_\mu$ are shown as wavy (blue) lines. The two sub-diagrams represent the two partition contributions listed in Eq. (B.2).

As we shall discuss later, this partitioning translates naturally into blocks of code. For certain calculations, symmetries of the Feynman diagram will lead to the contributions of some of these blocks being zero. We can then improve the performance of the code by commenting them out in these circumstances. For instance, in the "tadpole" diagram in the one-loop fermion self energy, we can remove the term in $Y_{F,2}$ that is proportional to $X^{\nu_1}_{F,2}$ [48]. Similarly, in the calculation of the radiative corrections to the gauge action [34], we can suppress the term proportional to $X^{\nu_1}_{F,3}$ in the "octopus" diagram.

We can further optimise the calculation of the reduced vertices $Y_{F,r}$ by reusing terms $X^\nu_{F,r}$ that appear multiple times in the expressions. For instance, in Eq. (B.3) in Appendix B we can reuse $X^\nu_{F,1}(\boldsymbol{k}_1,\mu_1)$ and $X^\nu_{F,1}(\boldsymbol{k}_3,\mu_3)$.

For testing it is useful to define a simpler variant of the smearings described on p. 3 of Ref. [49], which we call "FAT3". It is composed just of a central link and adjacent 3-staples with weights $c_1 = \frac{1}{2}$, $c_3 = \frac{1}{12}$.

### 3.2.1. $Z_{F,r}$: the field reduced vertex

The field reduced vertex essentially calculates the Feynman rule for obtaining $r$ *fattened* gluons in particular directions $\mu_1,\ldots,\mu_r$. The effect of fattening is treated separately in the $X_{F,r}$.

$Z_{F,r}$ is composed of a sum of $n_r$ monomials, each representing a different way of obtaining the required gluons from the contours comprising the action:

$$Z_{F,r}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\ldots;\boldsymbol{k}_r,\mu_r)=\frac{1}{r!}\sum_{n=1}^{n_r}\Gamma_n f_n\exp\left(\frac{1}{2}\left(\boldsymbol{p}\cdot\boldsymbol{x}+\boldsymbol{q}\cdot\boldsymbol{y}+\sum_{j=1}^{r}\boldsymbol{k}_j\cdot\boldsymbol{v}_{n,j}\right)\right), \tag{44}$$

where $\Gamma_n$ is a spin matrix.

Note that if there is no fattening of the underlying gluons, this is precisely the reduced vertex $Y_{F,r}$ described in Eq. (17), if we incorporate the $1/r!$ factor that was in Eq. (14).

*3.2.2. $X_{F,r}$: the algebra reduced vertex*

If the gluons are fattened, the effect of this is contained in the reduced vertices $X_{F,r}$. These represent how we would choose $r$ unfattened gluons in directions $\mu_1,\ldots,\mu_r$ from a fattened link in the $\nu$ direction:

$$X^{\nu}_{F,r}(\boldsymbol{k}_1,\mu_1;\ldots;\boldsymbol{k}_r,\mu_r)=\frac{1}{r!}\sum_{n=1}^{n_r}f_n\exp\left(\frac{1}{2}\sum_{j=1}^{r}\boldsymbol{k}_j\cdot\boldsymbol{v}'_{n,j}\right). \tag{45}$$

Note that there is no spin matrix in $X$, so the order of the multiplication in the expression for $Y$ does not matter.

The position vectors $\boldsymbol{v}'_{n,i}\equiv\boldsymbol{v}_{n,i}-\boldsymbol{e}_\nu$ refer to the position of the unfattened link relative to the start of the fattened. As such, we need to remove the half-link shift that was implicit in the definition of $\boldsymbol{v}_{n,i}$ as the *centre* of the link in the basic expansion algorithm.

The start and end sites of the contour $\boldsymbol{x}$ and $\boldsymbol{y}$ do not appear in $X$, making $X^{\nu}_{F,r=0}=1$ for $r=0$.

Of course, the actual values of $n_r$, $f_n$ and $\{v_{n,r}\}$ will be different in Eqs. (44) and (45), as they come from different smearing prescriptions.

### 3.3. Hardwired reunitarisation

We described above how splitting a fermion action such as HISQ into two parts simplifies the generation of the Feynman rules. This allowed us to exploit the reunitarisation of the inner FAT7R smeared links allows them to be expressed in terms of a new gauge potential $B$ (suppressing the Lorentz index).

Nonetheless, the Feynman rules for $B$ (as calculated in $X_{F,r}$) are still too complicated to derive for $r\geqslant 3$. The reason is clear if we look at the formula for the reunitarisation. Using Eq. (39), the reunitarised field is $W=(MM^\dagger)^{-1/2}M$, which we can express as an element of the algebra $B=\log W$:

$$\begin{aligned}
&B\equiv b_\mu A_\mu+b_{\mu\nu}A_\mu A_\nu+b_{\mu\nu\sigma}A_\mu A_\nu A_\sigma+\cdots,\\
&b_\mu=a_\mu,\\
&b_{\mu\nu}=\frac{1}{2}\left(a_{\mu\nu}-a^\dagger_{\mu\nu}\right),\\
&b_{\mu\nu\sigma}=\frac{1}{2}\left(a_{\mu\nu\sigma}-a^\dagger_{\mu\nu\sigma}\right)-\frac{1}{4}\left(a_\mu\left[a_{\nu\sigma}+a^\dagger_{\nu\sigma}\right]+\left[a_{\mu\nu}+a^\dagger_{\mu\nu}\right]a_\sigma\right)+\frac{1}{3}a_\mu a_\nu a_\sigma.
\end{aligned} \tag{46}$$

If there are $n$ monomials in $a_\mu$, there will be at least $n^3$ in $b_{\mu\nu\sigma}$ arising from the term $a_\mu a_\nu a_\sigma$. There will be some compression, but this will at best only reduce this number by a factor of around 2. For the simpler smearing FAT3, $n=19$ and already the HiPPy code struggles to produce $b_{\mu\nu\sigma}$ for FAT3R. For FAT7, $n=135$ and direct reunitarisation to produce FAT7R links using the HiPPy code is out of the question.

The alternative is to use the HiPPy code to produce the $\{a\}$ and to do the reunitarisation in the HPsrc code by hardwiring the formulæ in Eq. (46). As before, the $X_{F,r}$ algebra reduced vertices are based on expansion coefficients $\{b\}$. The inputs from the HiPPy code, however, implement the coefficients $\{a\}$. We use these to build a set of algebra reduced vertices $W_{F,r}$ and Eq. (46) gives the relation between the reunitarised $X_{F,r}$ and the $W_{F,r}$:

$$\begin{aligned}
X^{\nu}_{F,1}(\boldsymbol{k}_1,\mu_1)&=W^{\nu}_{F,1}(\boldsymbol{k}_1,\mu_1),\\
X^{\nu}_{F,2}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2)&=\frac{1}{2}\left(W^{\nu}_{F,2}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2)-W^{\nu}_{F,2}(\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_1,\mu_1)\right),\\
X^{\nu}_{F,3}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3)&=\frac{1}{2}\left(W^{\nu}_{F,3}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3)+W^{\nu}_{F,3}(\boldsymbol{k}_3,\mu_3;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_1,\mu_1)\right)\\
&\quad-\frac{1}{4}\left(W^{\nu}_{F,1}(\boldsymbol{k}_1,\mu_1)\left[W^{\nu}_{F,2}(\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3)+W^{\nu}_{F,2}(\boldsymbol{k}_3,\mu_3;\boldsymbol{k}_2,\mu_2)\right]\right.\\
&\quad\left.+\left[W^{\nu}_{F,2}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2)+W^{\nu}_{F,2}(\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_1,\mu_1)\right]W^{\nu}_{F,1}(\boldsymbol{k}_3,\mu_3)\right)\\
&\quad+\frac{1}{3}W^{\nu}_{F,1}(\boldsymbol{k}_1,\mu_1)W^{\nu}_{F,1}(\boldsymbol{k}_2,\mu_2)W^{\nu}_{F,1}(\boldsymbol{k}_3,\mu_3).
\end{aligned} \tag{47}$$

## 4. Description of the software

### 4.1. The HiPPy code

The HiPPy code is used to Taylor expand lattice actions, producing output files encoding the monomials making up the reduced vertices that can later be used by the HPsrc code to evaluate Feynman diagrams and integrals.

The code is written in Python, for which interpreters are freely available for a wide range of computational platforms [50].

*4.1.1. Installation and compatibility*

The HiPPy code can be run on any machine which has Python version 2.5.x installed on it. It is also expected to work with any version 2.x, but is not (yet) compatible with version 3.x. Only the packages supplied in a standard installation of Python are required.

Installation consists of unpacking the source files in the chosen location.

*4.1.2. Testing*

A set of unit test programs are contained in subdirectory `tests`. They are run from the command line with no options and the code is expected to pass all of these. No input files are needed for the tests.

Whilst the unit testing does not cover each code feature separately, those in `test_class_field.py` do cover most features combined.

*4.1.3. Main programs*

The HiPPy code consists of the following main programs that make use of the core routines described below.

A full list of options for each of the main programs can be obtained by running them from the command line with option `--help`.

`sample_link.py`. Used to generate monomials from the algebra of the group for a single fattened link in the specified direction. These are used to evaluate $X_{F,r}$ in Eq. (45) or $W_{F,r}$ in Eq. (47) in the HPsrc code.

The fattening style is specified using `--bottom_style`. The output filenames have the form `algebra_%s%i_qq*.in`, where `%s` is replaced by the name of the fattening style and `%i` by the direction of the link (with value `parameters.D` mapped to zero). Wildcard `*` is $r$ copies of the character "`g`" (up to maximum `parameters.R`).

For monomials that will be used in Eq. (47) (with hardwired reunitarisation carried out in the HPsrc code), the option `--fld_as_alg` should be specified. In this case `%s` is a concatenation of character "`F`" and the name of the fattening.

If an undefined fattening style is specified with `--bottom_style`, a list of acceptable styles is printed.

`sample_naive.py`. Used to generate monomials for the naive (or, equivalently, staggered) lattice Dirac action using fattened links. These are used to evaluate $Y^{(k)}_{F,r}$ in Eq. (35) in the HPsrc code.

The fattening style(s) are specified using `--top_style` and `--bottom_style`, and should not include fattening already carried out at the algebra stage above. The output filenames have the form `vertex_%s_qq*.in`, where `%s` is replaced by the name of the top fattening style and `*` is $r$ copies of the character "`g`". For complicated fattening prescriptions, there is likely to be some filename clashes, so care should be exercised.

If an undefined fattening style is specified with `--top_style`, a list of acceptable styles is printed.

`sample_glue.py`. Used to generate monomials for a range of simple gauge actions, specified by a command line argument. As described above, these vertices have been (partially) symmetrised.

*4.1.4. Example of use*

As an example, here are the commands one would use to generate the four-dimensional HISQ reduced vertex files using a two-level action with hardwired reunitarisation and mass $am = 0.2$:

```
python sample_link.py --bottom_style fat7 --fld_as_alg 1
python sample_link.py --bottom_style fat7 --fld_as_alg 2
python sample_link.py --bottom_style fat7 --fld_as_alg 3
python sample_link.py --bottom_style fat7 --fld_as_alg 4
python sample_naive.py --top_style asq_for_hisq 0.2
```

which creates files `algebra_Ffat7%i_qq*.in` and `vertex_asq_for_hisq_qq*.in`. The same Feynman rules can be generated using

```
python sample_naive.py --top_style asq_for_hisq
   --bottom_style fat7r 0.2
```

but this generates *far* more monomials, which soon becomes prohibitive as $r$ increases, as discussed above.

For HISQ fermions, there is the option to set the mass-dependent parameter $\varepsilon$ to zero for small $am$. This is done using a command line option `--hisq_eps_zero` for `sample_naive.py`.

For main programs to calculate Feynman rules for other actions, e.g., NRQCD, interested readers should contact the authors.

*4.1.5. Core routines*

These main programs make use of the following core routines:

`parameters.py`. Defines physical and numerical parameters associated with the expansion. If the monomials require complex-valued amplitudes, e.g., for mNRQCD actions, this is enabled here.

`template.py`. Defines some link fattening prescriptions that can be used to define actions. Each fattening is a set of paths which do not need to be gauge covariant, so three-link Naik terms can be included, for instance. Instructions for adding new fattenings are included in this file.

`spin_multiplication.py`. Implements the implicit representations of the spin algebras described in Section 2.6. Running this from the command line displays the multiplication table from Eq. (28) for either Pauli or Dirac spin matrices.

`wilson.py`. The main expansion engine that converts a collection of Wilson lines into a Taylor expansion in the form of reduced vertices built from monomials. The collection of Wilson lines is defined by

```
thepath = [ [c1,c2,...],[p1,p2,...] ]
```

where `c1`, `c2` etc. are the amplitude of each contour, the path of which is defined by a set of signed integers. For instance, a plaquette would be `p1=[1,2,-1,-2]`.

As an example of a complete path, a two-dimensional naive fermion action would be defined by a path:

```
thepath = [ [0.5,-0.5,0.5,-0.5,m],[ [1],[-1],[2],[-2],[] ] ]
```

The expansion is not symbolic, so the mass `m` must take a specific numerical value.

It is assumed that all links are of the same, fattened style. The fattening can contain any number of iterated styles defined in `template.py`, given as a list `top_style` interpreted from right to left.

At the lowest level of this iteration, a further single level of fattening can be specified using `bottom_style`. Uniquely, a reunitarisation by projection onto $U(N)$ can be imposed after this fattening by appending character "r" to the end of the name of the style (again, chosen from the definitions in `template.py`).

`class_entity.py`. Each monomial in the reduced vertex is encoded as an instance of this class, a description of which can be found in Section 4 of Ref. [17]. Translation invariance is exploited so that all paths are assumed to start at the origin, $\boldsymbol{x} = \mathbf{0}$. If this is not the case (e.g., for open boundary conditions in the temporal direction for Schrödinger functional calculations [38,39]), then an attribute `start_site` would need to be introduced to this class and appropriate changes made to ensure gauge covariance in the multiplication and in other places.

`class_field.py`. The collection of monomials defining the reduced vertices returned by `wilson.path()` are stored as a dictionary in an instance of this class. A fuller description can be found in Section 4 of Ref. [17].

`class_algebra.py`. Very similar in form to `class_field.py`, this class contains the reduced vertices associated with the algebra in Eq. (42).

`mathfns.py`. contains a collection of methods not tied to any particular class.

### 4.2. *The* HPsrc *code*

The HPsrc code uses the input files from the HiPPy code to create Feynman rules, and combines these to evaluate Feynman integrals by exact mode summation or statistical estimation using the Vegas algorithm [14,15].

#### 4.2.1. *Installation and compatibility*

The HPsrc code requires a standards-compliant Fortran95 compiler with support for minimal CPP directives (chiefly `#ifdef` and `#ifndef`). The command `make` is optional but useful. An appropriate MPI wrapper for the compiler will be needed for creating a parallel version of the executable. No additional libraries are required.

The code is known to compile and execute correctly with the following compilers: Intel ifort, Portland pgf90, NAG f95 and GNU gfortran on Unix-based systems (the first two also with MPI on parallel machines) and Silverfrost/Salford ftn95 and Lahey-Fujitsu lf95 on Microsoft Windows systems using cygwin.

To install the code, the source should be unpacked in a suitable directory and the file `Makefile_details` (which contains machine-dependent information) edited to point to the correct compiler.

It is a good idea to execute `make clean` before compiling the code. A given target `this_executable` can be compiled using the command

```
make this_executable
```

Use the command `make` to see a list of possible executables, or examine `Makefile`.

#### 4.2.2. HPsrc *filename conventions*

Main programs are named `main_*.F90` and compile to form executables with the same filestem and extension `.$(EXE)` as specified by `Makefile_details`. Typically `$(EXE) = x`, but certain compilers expect `$(EXE) = exe`.

A Fortran module `this` is typically contained in a file named `mod_this.F90`.

Input files containing information that should be specified at compile time are named with extension `.i`. The command `make clean` must be executed before recompiling if any of these files are changed. These files typically contain unavoidable CPP directives and the number of these has been kept to a minimum.

Input files that are read at runtime are have names with extension `.in`. Most input files have a filename that reflects that of the module that reads them.

*4.2.3. Testing*

The code has been rigorously tested as a whole in a number of scientific calculations, being compared with identical calculations carried out using different programs.

In addition, various manual and automatic tests are routinely carried out to verify the continued correctness of the code. The manual tests are located in subdirectory `tests/`.

*Automatic differentiation.* Without a templating option in Fortran, each vertex is effectively coded twice: once as a complex-valued object and once as a Taylor-valued object that includes the automatic derivatives.

The code exploits this by performing a finite difference of the first to compare with the derivatives included in the second. To avoid multiplication of errors in higher-order difference operators, the tests are carried out as follows. First, the non-Taylor version is compared with the zero-order derivative from the Taylor version. If this agrees, a first-order, symmetric difference is made of the zero-order derivative in the Taylor in direction $\tau$. We check that, for all $\tau$, this agrees (within a specified tolerance) with the analytic first order derivative in the Taylor object. If so, we form first-order differences of the analytic first-order derivatives and compare these with the analytic second order derivatives and so on.

These comparisons are done at runtime when the vertex modules are first initialised.

This test checks not only that the automatic differentiation (and associated overloading of operators) is working correctly, but also that there are no coding inconsistencies between the Taylor and non-Taylor versions of the vertices. It cannot, of course, detect algorithmic errors that have been consistently coded in both versions. An additional, lower level test of the TaylUR package is provided by `test_taylors.x`.

`test_compare_vertices.x`. Can be used to check that the Feynman rules are the same for two different implementations of the same quark action. This provides a rigorous check of the algorithms in `mod_vertex_qq*.F90`. Having compiled `test_compare_vertices.x`, a Python script `test_compare_vertices.py` can be executed to complete various independent checks of the two-level action decomposition, the hardwired reunitarisation and the summand/factor division of the action. The last is based on an NRQCD action and also provides a strong test of the implementation of the Pauli spin algebra.

*Tests of* Vegas. The Vegas code is automatically tested at runtime, evaluating the area under a two-dimensional, normalised Gaussian as well as a contour integral. These tests can also be run using `test_vegas.x`.

*Lower level* HPsrc *tests.* `test_spinors.x` tests type `spinor` defined in Section 2.6. In addition, `test_pauli_dirac_spinors.x` tests the embedding of Pauli two-spinors into Dirac 4-spinors. `test_print_vertices.x` prints the values of the fermionic vertices for a random set of momenta. The same random number seed is used each time, so running the code for different `vertex_qq_composite.in` on the same machine will yield outputs that can be directly compared.

*4.2.4. Main programs*

To illustrate the use of the HPsrc code, we provide three sample programs with Makefile targets:

`quark_sigma.x`. Calculates the one-loop renormalisation of ASQTAD improved staggered fermions. To do this, it evaluates the self-energy Feynman diagrams and their derivatives.

`nrqcd_sigma.x`. Performs a similar calculation for heavy NRQCD fermions.

`nflwimp.x`. Calculates the HISQ fermion contribution to the one-loop radiative corrections to the Lüscher–Weisz gauge action, as per Refs. [34,35].

*4.2.5. Core routines*

This subsection describes modules that are needed to use the vertex modules. If MPI is to be used, the file `use_mpi.i` must be changed prior to compilation. Similarly, if monomials require complex amplitudes, e.g., for mNRQCD calculations, this is enabled in `use_complex_amplitudes.i`.

`mod_num_parm.F90`. Numerical parameters that are used by most of the other modules. Most parameters in this module will not need to be changed. It also optionally reads from file `paths.in` the pathnames of the directories holding the vertex and algebra files generated by the HiPPy code.

`mod_phys_parm.F90`. Physical parameters used in the calculation. The code has only been tested for number of dimensions `Ndir` $= 4$ and colours `Ncol` $= 3$. Other choices will require careful testing before use. Values for many parameters can be changed at runtime by editing file `phys_parm.in`. Twisted boundary conditions are selected by setting the number of twisted directions `ntwisted_dirs` to 2, 3 or 4. Value 0 gives periodic boundary conditions. For finite lattices, the lattice size is specified in `latt_size(0:Ndir-1)`, with the first component the temporal one. If momenta are to be squashed $k_\mu \to k_\mu - \alpha_\mu \sin(k_\mu)$ [18], the squash factors $\alpha_\mu$ in each direction are specified in `mom_squash(0:Ndir-1)`. The anisotropy metric `anis` is only supported currently for gluonic vertices.

`mod_momenta.F90`. Defines type `mom` which encodes momenta. Each instance contains the momentum components, the twist vector (only relevant for twisted boundary conditions) and a route variable that is used in automatic differentiation.

`mod_taylors.F90`. Defines type `taylor`, as described in Section 2.5.

`mod_spinors.F90`. Defines types `spinor` and `tayl_spinor`, as described in Section 2.6.

`mod_matrices.F90`. Defines types `mat4x4`, used to explicitly represent objects such as the gluon propagator in Lorentz index space. A corresponding Taylor-valued type `tayl_mat4x4` is also defined.

`mod_colour.F90`. Calculates the Clebsch–Gordan factors for both gluonic and fermionic vertices, for periodic and twisted boundary conditions. Colour factors for periodic boundary condition are pre-computed and stored in the compile-time files `tr_cols_N.i` and `mat_cols_N.i`. For twisted boundary conditions, the Clebsch–Gordan factors are computed as needed.

`mod_mod_mpi.F90`. The code can either be run as a scalar code on a single processor or as a parallel code using MPI. To achieve this with the minimum of compile-time code modifications (either by hand or using CPP directives), `mod_mod_mpi.F90` contains a set of dummy MPI subroutines. Whether these dummy routines are used or not is controlled by the single CPP directive in file `use_mpi.i`. Note that `mod_mod_mpi.F90` is the only file that reads `use_mpi.i`. Only the MPI commands we use are included in `mod_mod_mpi.F90`. We do not know of any externally maintained library of dummy MPI commands for use in such situations. If this were to exist, it could easily be used to replace this file.

Throughout the code we are (or try to be) careful that input/output files are only opened by one processor, with information then shared using MPI commands.

`mod_vegasrun.F90`. Provides an implementation of the Vegas algorithm, based on the original code of Peter Lepage [14,15], who also helped convert the code to a parallel version using MPI. On parallel machines, the processors are divided into farms, each of which does an independent Vegas estimate of the integral. Within the farm, all processors carry out the same calculation, interacting only to share the work of evaluating the integrand at each of the Monte Carlo points. The number of processors per farm is controlled in file `vegas_parm.in`, and must be a factor of the number of processors. Farming has the advantage of avoiding potentially slow global communications on large clusters.

An improved parallel implementation is currently being prepared as part of a DEISA DECI-funded project [51].

#### *4.2.6. Gluonic vertex modules*

Feynman rules $V_{G,r}$ for the gluonic vertices are implemented in `mod_vertex_*.F90`, where `*` is $r$ (the number of gluons) copies of character "g". Currently vertices for $r \leqslant 5$ have been implemented.

The modules for $r \geqslant 1$ are called `vertex_*` and have a very similar structure. The top level routine is `vert_*(k,lorentz,colour)`, which takes as arguments the momenta, Lorentz and colour indices and returns the complete *symmetrised* Feynman rule for that vertex as a complex number. The colour array is ignored if we are using twisted boundary conditions. The module assumes that the monomials written by the HiPPy code have been partially symmetrised as described above. The remaining symmetrisation over the distinct cosets is carried out automatically in the top level routine. The reduced vertex is calculated from the monomials in `yvertex_*()`.

This code structure is repeated using instances of type `taylor` instead of complex numbers, providing analytic derivatives of the Feynman rules. The top level routine in this case is called `taylor_vert_*()`. As detailed above, coding the algorithm twice in this way is exploited in the runtime testing of the code.

The remainder of the `mod_vertex_*.90` files is taken up with initialisation code (which reads in all the vertex files at runtime) and testing routines.

Only `mod_vertex_gg.F90` differs from this overall structure. This calculates the gluon propagator. In this case, the top-level routine is `gluon_prop(k,colour)`. The propagator is simultaneously calculated for all Lorentz index combinations, packaged as a $D \times D$ matrix implemented as derived type `mat4x4`.

The choice of gauge is controlled by variable `Galpha`, specified in runtime input file `vertex_gg_parm.in`. Value `1` corresponds to Feynman gauge and `0` to Landau gauge. As detailed in Ref. [28], the inverse propagator does not exist in Landau gauge, so an intermediate gauge parameter `Gbeta` should be used. `gauge_family = 'landau'` is the usual choice, where the gauge correction is based on the four-vector $\boldsymbol{k}$. Changing this to `'coulomb'` uses just the spacelike components instead, but this option is not fully tested.

As evaluation of the gluon propagator can be an expensive part of a perturbative calculation, the propagators for various gluon actions are hardwired in `mod_vertex_gg.F90`. Their use is specified by changing variable `action` in `vertex_gg_parm.in` from `'python'` to appropriate other character strings which are listed in subroutine `vertex_gg_params()`.

A gluon mass can be added to the propagator. Its square is `gluon_mass2`, specified in `vertex_gg_parm.in`.

Again, this code is repeated using automatic derivatives packaged instead in a `taylor`-valued matrix of type `tayl_mat4x4`. The remainder of the module contains runtime tests and initialisation routines.

Associated with the gluon modules are modules calculating the Feynman rules associated with the Fadeev–Popov ghost fields and the Haar measure. These are implemented as hardwired formulæ in `mod_vertex_hhstar.F90` (for $r \leqslant 2$) and in `mod_vertex_meas_gg.F90` (only for $r = 2$).

#### *4.2.7. Fermion vertex modules*

Feynman rules $V_{F,r}$ for the fermionic vertices (the *unsymmetrised* version of Eq. (15)) are calculated in the files `mod_vertex_qq*.F90`, where `*` is $r$ (the number of gluons) copies of character "g". Currently vertices for $r \leqslant 3$ have been implemented. The modules for $r \geqslant 1$ are called `vertex_qq*` and have a very similar structure. The top level routine is `vert_qq*(k,lorentz,colour)`, which takes as arguments the momenta, Lorentz and colour indices and returns the complete Feynman rule for that vertex as an instance of type `spinor`. The convention is that `k(1:r+2)` is $(/\boldsymbol{q}, \boldsymbol{p}, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r/)$. The colour array has the same ordering, but is ignored if we are using twisted boundary conditions.

Internally, this function calculates the reduced vertex using `yvertex_qq*()` and multiplies on the appropriate Clebsch–Gordon factor, calculated using `mod_colour.F90`.

`yvertex_qq*()`. Implements the summands and factors decomposition described in Section 3.1. The reduced vertices for each factor are calculated in `yvertex_qq*_partial()`. The sum over partitions $\sum_{P\in P^{(r)}}$ naturally translates into a block structure for this routine, the order of which follows that of the expressions in Appendix A.

`yvertex_qq*_partial()`. Implements the two-level field/algebra algorithm as per Section 3.2. The "upper-level", fattened, field vertices $Z_{F,r}$ are calculated in `yvertex_qq*_partial_noalg()`, combining monomials that have been read into the module at runtime. The "lower-level", algebra vertices $X_{F,r}$ are calculated in `yvertex_qq*_algebra()`. Once more, the sum over partitions translates into a block structure for the code, and the order reflects the expressions in Appendix B. As discussed earlier, for certain Feynman diagrams we can comment out some blocks on symmetry grounds.

`yvertex_qq*_algebra()`. Calculates $X_{F,r}$ in terms of the monomial-calculated $W_{F,r}$ from `y_qq*_alg_basic()`, as detailed in Section 3.3. There is the option to apply no reunitarisation (setting $X_{F,r} = W_{F,r}$), or the hardwired projection described in the text. Additional projection methods, such as "stouting" [52], could also be implemented here.

This code structure is repeated using instances of type `tayl_spinor` instead of type `spinor`, providing analytic derivatives of the Feynman rules. The top level routine in this case is called `taylor_vert_qqg*()`. As detailed above, coding the algorithm twice in this way is exploited in the runtime testing of the code.

The remainder of the `mod_vertex_qq*.F90` files is taken up with initialisation code (which reads in all the vertex files at runtime) and testing routines.

Only `mod_vertex_qq.F90` differs from this overall structure. This calculates the fermion propagator. In this case, the top-level routine is `quark_prop(k,colour)` where the momentum specified is $\boldsymbol{q}$ (which flows in the direction of the fermion arrow). `inv_quark_prop()` calculates the two-point function for the fermions and implements the summands and factors decomposition described in Section 3.1. The reduced vertex for each factor is calculated in `inv_quark_prop_partial()`. This is the lowest-level routine, as there is no algebra contribution for $r = 0$. These routines are followed in the code by type `tayl_spinor` versions and by initialisation and tests.

#### 4.2.8. Runtime specification of Feynman rules

The HPsrc code allows for the use of multiple fermion types, each with their own Feynman rules constructed in their own way. The details of all of this are specified at runtime by the file `vertex_qq_composite.in`, which is read by all the `mod_vertex_qq*.F90` files. The input takes the form of a NAMELIST. A typical example is given in Appendix C.

## 5. Conclusions

The accuracy of simulations of lattice QCD (and other field theories) has been markedly improved by the use of Symanzik- and radiatively-improved actions and measurement operators. In addition to the calculation of the radiative corrections, this improvement programme also requires a concomitant effort in calculating associated renormalisation parameters that are vital to the continuum and chiral extrapolations. Lattice perturbation theory is an essential tool in these calculations. For some quantities we can use other techniques, in particular non-perturbative renormalisation (NPR) [1,2], high-$\beta$ simulations [10,12] and stochastic techniques [13]. In cases where these can be used, lattice perturbation theory provides a valuable check of their results and, in the case of high-$\beta$ simulations, important constraints on the coefficients of the low powers of $\alpha_s$. When there is complicated mixing of operators, which happens in a lot of physically relevant measurements, statistical errors lead to the failure of the NPR and high-$\beta$ techniques and lattice perturbation theory is the only alternative. Stochastic perturbation theory is also very expensive for theories with dynamical fermions.

Lattice perturbation theory for improved actions is complicated: not only does it include many vertices not present in the continuum, the Feynman rules also contain a very large number of terms. These complications vanish, of course, in the continuum limit as the lattice spacing $a$ tends to zero. The point is, however, that lattice perturbation theory is used to correct for the missing momentum modes on a discrete lattice, so the presence of these complications are central to its utility. Many of the complicating terms violate Lorentz symmetry and are mathematically complicated. This makes analytic evaluation of Feynman integrals impossible in almost all cases. It is therefore crucial that we develop a robust computational method for deriving and then using these Feynman rules in perturbation theory. This method must be efficient as well as accurate.

In this paper we have described such a method. We have used a two-pronged approach, with separate programs to expand the action and derive the Feynman rules, and then to use these rules to calculate Feynman integrals. In both cases, we have developed efficient algorithms to minimise the computational expense. Indeed, without these efficient algorithms we find the calculations to be impossible for many realistic action choices. We have also presented a full, working implementation of the algorithm in the form of the HiPPy and HPsrc codes. The programming languages used for the implementations are Python and Fortran95, respectively. Python offers good list-handling and object-orientation features that are suited to the expansion of the action to derive the Feynman rules. Fortran95 is numerically efficient, an over-riding concern for the evaluation (or estimation) of expensive Feynman integrals.

We stress that our method is generically applicable. In the text, we have focused on the features of our method that make possible calculations using realistic NRQCD and HISQ fermion actions. This bias merely reflects the problems for which we have used the method most so far. We have considered other actions, both by expanding them using the HiPPy code and by hardwiring the hand-derived Feynman rules in the HPsrc vertex modules.

Particular strengths of our approach are the ability to deal with complicated colour and spin structures in actions and the breaking down of actions into simpler sub-structures for faster evaluation. We have also provided a particularly simple way of describing actions that reduces the scope for transcription errors. The comprehensive suite of tests in the code also give confidence in the accuracy of the basic components of both the HiPPy and HPsrc code suites.

As well as extending the basic functionality of the codes, e.g., to describe actions with multiply nested levels of link fattening, the codes are easily extended to a wide range of related problems in lattice simulation, as has already been done for lattice chiral perturbation theory [37] and Schrödinger functional calculations [38,39]. Another area in which the code might be used is in calculations where the lattice

gauge field is split into a fluctuating, quantum piece and an external background field [53,54]. The authors are happy to provide further details on the implementation of this, as well as additional advice on the use of the HiPPy and HPsrc codes.

**Acknowledgements**

A.H. thanks the U.K. Royal Society for financial support. The University of Edinburgh is supported in part by the Scottish Universities Physics Alliance (SUPA). G.M.v.H. is supported by the Deutsche Forschungsgemeinschaft in the SFB/TR 09. This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF; http://www.ecdf.ed.ac.uk). The ECDF is partially supported by the eDIKT initiative (http://www.edikt.org.uk). We also acknowledge support from the DEISA Extreme Computing Initiative (http://www.deisa.eu/science/deci).

## Appendix A. Explicit expressions for factors

Using Table 1, we here give explicit expressions for Eq. (35) for $Y_{F,r}$ for a summand in the action with $N$ factors, for $r = 0 \ldots 3$ (the range of $r$ implemented in the HPsrc code):

$$Y_{F,0}(\boldsymbol{p},\boldsymbol{q}) = \prod_{k=1}^{N} Y_{F,0}^{(k)}(\boldsymbol{p},\boldsymbol{q}), \tag{A.1}$$

$$Y_{F,1}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1) = \sum_{n_1=1}^{N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,1}^{(n_1)}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1) \left( \prod_{k=n_1+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right), \tag{A.2}$$

$$\begin{aligned} Y_{F,2}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2) = {} & \sum_{1\leqslant n_1\leqslant N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,2}^{(n_1)}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2) \left( \prod_{k=n_1+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right) \\ & + \sum_{1\leqslant n_1<n_2\leqslant N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,1}^{(n_1)}(\boldsymbol{p},-\boldsymbol{p}-\boldsymbol{k}_1;\boldsymbol{k}_1,\mu_1) \left( \prod_{k=n_1+1}^{n_2-1} Y_{F,0}^{(k)}(\boldsymbol{p}+\boldsymbol{k}_1,-\boldsymbol{p}-\boldsymbol{k}_1) \right) \\ & \times Y_{F,1}^{(n_2)}(\boldsymbol{p}+\boldsymbol{k}_1,\boldsymbol{q};\boldsymbol{k}_2,\mu_2) \left( \prod_{k=n_2+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right), \end{aligned} \tag{A.3}$$

$$\begin{aligned} & Y_{F,3}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3) \\ & = \sum_{1\leqslant n_1\leqslant N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,3}^{(n_1)}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3) \left( \prod_{k=n_1+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right) \\ & \quad + \sum_{1\leqslant n_1<n_2\leqslant N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,2}^{(n_1)}(\boldsymbol{p},\boldsymbol{q}+\boldsymbol{k}_3;\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2) \\ & \quad \times \left( \prod_{k=n_1+1}^{n_2-1} Y_{F,0}^{(k)}(-\boldsymbol{q}-\boldsymbol{k}_3,\boldsymbol{q}+\boldsymbol{k}_3) \right) Y_{F,1}^{(n_2)}(-\boldsymbol{q}-\boldsymbol{k}_3,\boldsymbol{q};\boldsymbol{k}_3,\mu_3) \left( \prod_{k=n_2+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right) \\ & \quad + \sum_{1\leqslant n_1<n_2\leqslant N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,1}^{(n_1)}(\boldsymbol{p},-\boldsymbol{p}-\boldsymbol{k}_1;\boldsymbol{k}_1,\mu_1) \left( \prod_{k=n_1+1}^{n_2-1} Y_{F,0}^{(k)}(\boldsymbol{p}+\boldsymbol{k}_1,-\boldsymbol{p}-\boldsymbol{k}_1) \right) \\ & \quad \times Y_{F,2}^{(n_2)}(\boldsymbol{p}+\boldsymbol{k}_1,\boldsymbol{q};\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3) \left( \prod_{k=n_2+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right) \\ & \quad + \sum_{1\leqslant n_1<n_2\leqslant N} \left( \prod_{k=1}^{n_1-1} Y_{F,0}^{(k)}(\boldsymbol{p},-\boldsymbol{p}) \right) Y_{F,1}^{(n_1)}(\boldsymbol{p},-\boldsymbol{p}-\boldsymbol{k}_1;\boldsymbol{k}_1,\mu_1) \left( \prod_{k=n_1+1}^{n_2-1} Y_{F,0}^{(k)}(\boldsymbol{p}+\boldsymbol{k}_1,-\boldsymbol{p}-\boldsymbol{k}_1) \right) \\ & \quad \times Y_{F,1}^{(n_2)}(\boldsymbol{p}+\boldsymbol{k}_1,\boldsymbol{q}+\boldsymbol{k}_3;\boldsymbol{k}_2,\mu_2) \left( \prod_{k=n_2+1}^{n_3-1} Y_{F,0}^{(k)}(-\boldsymbol{q}-\boldsymbol{k}_3,\boldsymbol{q}+\boldsymbol{k}_3) \right) Y_{F,1}^{(n_3)}(-\boldsymbol{q}-\boldsymbol{k}_3,\boldsymbol{q};\boldsymbol{k}_3,\mu_3) \left( \prod_{k=n_3+1}^{N} Y_{F,0}^{(k)}(-\boldsymbol{q},\boldsymbol{q}) \right). \end{aligned} \tag{A.4}$$

The reduced vertex from the $n$th is denoted $Y_{F,r}^{(n)}$. The formula for $r = 3$ is illustrated graphically in Fig. 2.

## Appendix B. Explicit expressions for two-level actions

To make things more concrete, we give explicit formulæ for the $r = 1$, $r = 2$ and $r = 3$ reduced vertices for a two-level action as defined in Eq. (43). This is the range of $r$ implemented in the HPsrc code. Using the partitions in Table 1 we obtain:

$$Y_{F,1}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1) = \sum_{\nu_1} Z_{F,1}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\nu_1) X_{F,1}^{\nu_1}(\boldsymbol{k}_1,\mu_1), \tag{B.1}$$

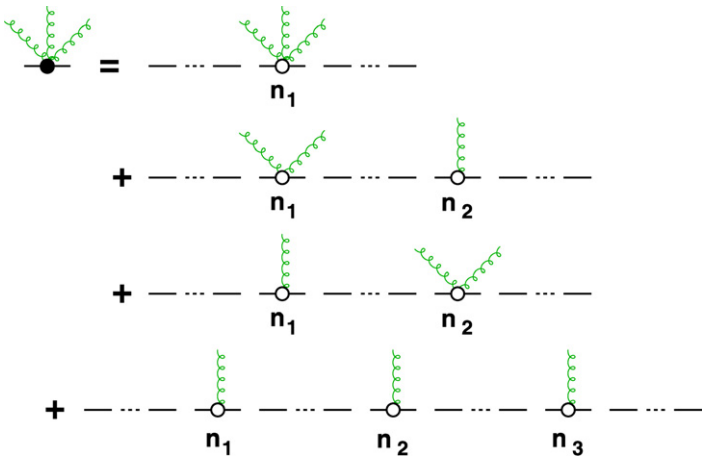


**Fig. 2.** A graphical representation of the reduced vertex $Y_{F,r}$ for $r = 3$ for a single summand of an action that is composed of a number of factors, as described mathematically in Eq. (A.4) and implemented in the HPsrc code. The four terms in Eq. (A.4) are separately shown, with $n_{1,2,3}$ showing the number of the factors from which each of the gluons are drawn. Open circles denote vertices associated with individual factors. Factors contributing no gluons are shown as "$-\cdots-$". Momentum flow in the vertices is not shown, but can be deduced from Eq. (A.4).

$$Y_{F,2}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2) = \sum_{\nu_1} Z_{F,1}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1+\boldsymbol{k}_2,\nu_1) X^{\nu_1}_{F,2}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2) + \sum_{\nu_1,\nu_2} Z_{F,2}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\nu_1;\boldsymbol{k}_2,\nu_2) X^{\nu_1}_{F,1}(\boldsymbol{k}_1,\mu_1) X^{\nu_2}_{F,1}(\boldsymbol{k}_2,\mu_2), \tag{B.2}$$

$$\begin{aligned} Y_{F,3}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3) = {} & \sum_{\nu_1} Z_{F,1}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1+\boldsymbol{k}_2+\boldsymbol{k}_3,\nu_1) X^{\nu_1}_{F,3}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3) \\ & + \sum_{\nu_1,\nu_2} Z_{F,2}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1+\boldsymbol{k}_2,\nu_1;\boldsymbol{k}_3,\nu_2) X^{\nu_1}_{F,2}(\boldsymbol{k}_1,\mu_1;\boldsymbol{k}_2,\mu_2) X^{\nu_2}_{F,1}(\boldsymbol{k}_3,\mu_3) \\ & + \sum_{\nu_1,\nu_2} Z_{F,2}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\nu_1;\boldsymbol{k}_2+\boldsymbol{k}_3,\nu_2) X^{\nu_1}_{F,1}(\boldsymbol{k}_1,\mu_1) X^{\nu_2}_{F,2}(\boldsymbol{k}_2,\mu_2;\boldsymbol{k}_3,\mu_3) \\ & + \sum_{\nu_1,\nu_2,\nu_3} Z_{F,3}(\boldsymbol{p},\boldsymbol{q};\boldsymbol{k}_1,\nu_1;\boldsymbol{k}_2,\nu_2;\boldsymbol{k}_3,\nu_3) X^{\nu_1}_{F,1}(\boldsymbol{k}_1,\mu_1) X^{\nu_2}_{F,1}(\boldsymbol{k}_2,\mu_2) X^{\nu_3}_{F,1}(\boldsymbol{k}_3,\mu_3). \end{aligned} \tag{B.3}$$

## Appendix C. Example of `vertex_qq_composite.in`

Here we provide an explicit example of the runtime file used to specify the Feynman rules used by the HPsrc code.

The calculation will use two types of quarks: relativistic HISQ and heavy NRQCD. It is assumed that the appropriate vertex and, where relevant, algebra files have been precomputed using the HiPPy code and stored in the locations specified in the HPsrc file `paths.in`.

```
&vertex_qq_composite
        no_quark_types = 2
! First quark type: HISQ
!   All RH indices set to 1 to denote first quark type
        summands(1) = 1
        factors(1:1,1) = 1
        summand_amps(1:1,1) = 1.d0
        opname(1:1,1,1) = "asq_for_hisq_"
        algname(1) = "Ffat7"
reunit_to_apply_to_alg(1) = "project"
! Second quark type: NRQCD
!   All RH indices set to 2 to denote second quark type
        summands(2) = 2
        factors(1:2,2) = 0,4
        summand_amps(1:2,2) = 1.d0,-1.d0
        opname(1:4,2,2) = "nrqcd_A","nrqcd_B","nrqcd_C","nrqcd_A"
        algname(2) = "simple"
reunit_to_apply_to_alg(2) = "none"
! N.B. no opname(:,1,2) definition because first summand has no
!   HiPPy inputs because it is a constant
/
```

That there are two quark types is specified in the NAMELIST using `no_quark_types`. Within the HPsrc code we control which set of Feynman rules we use by setting variable `quark_type` to 1 or 2 prior to calling `quark_prop()` or `vert_qqg*()`.

The first fermions are relativistic HISQ quarks. There is no splitting of the action into summands and factors, so `summands(qt)=1` and `factors(1:summands(qt),qt)=1`. Similarly, the only summand has amplitude `summand_amps(1:summands(qt),qt)=1.0d0`.

The files used to construct $Z_{F,r}$ for each factor for each summand are assumed to have been generated by the HiPPy code and stored in files named `vertex_%sqq*.in` where `%s` is replaced by appropriate text for each factor for each summand as specified in `opname(ft,sm,qt)`, where `ft` runs from 1 to `factors(sm,qt)` and `sm` runs from 1 to `summands(qt)`. In this case, the only vertex files are those with `%s` replaced by `asq_for hisq_`.

The name of the algebra file is given by `algname(qt)` (note there is no trailing underscore in this case). We specify that hardwired projection is required to relate $W_{F,r}$ in Eq. (47) to $X_{F,r}$ in Eq. (45) using `reunit_to_apply_to_alg(qt)`. The HPsrc code currently assumes that the same algebra files are used for all factors and summands of a particular quark type. Again, it is assumed that the files defining these have been pre-produced using the HiPPy code and stored in files named `algebra_%s%i_qq*.in`. `%s` is replaced by `algname(qt)` and `%i` runs from 0 to $D-1$, where $D$ is the number of dimensions.

In this explanation, we have used `qt` to denote the quark type, and other intuitive labels to specify array sizes. Unfortunately, NAMELISTs only support numbered array entries.

The second block of the NAMELIST defines an NRQCD action of the heuristic form $\bar{\psi}(1-ABCA)\psi$. There are two summands, the first with 0 factors (which gives default answer 1) and the second with 4. The minus sign in front of the second summand is specified in `summand_amps`. The first summand has no factors, so `opname` is not specified. The second summand has 4 factors, and the filenames are specified. Note that the first and last filenames are the same. The algebra is specified as before, and no hardwired reunitarisation is required.

In both cases, the correct spin algebra is specified in the headers of the input files written by the HiPPy code.

## References


[1] G. Martinelli, C. Pittori, C.T. Sachrajda, M. Testa, A. Vladikas, A General method for nonperturbative renormalization of lattice operators, Nucl. Phys. B 445 (1995) 81–108, doi:10.1016/0550-3213(95)00126-D, arXiv:hep-lat/9411010.

[2] R. Sommer, Non-perturbative QCD: Renormalization, O(a)-improvement and matching to heavy quark effective theory, lectures given at Workshop on Perspectives in Lattice QCD, Nara, Japan, 2006, arXiv:hep-lat/0611020.

[3] T. Bhattacharya, R. Gupta, W.-J. Lee, S.R. Sharpe, Order $a$ improved renormalization constants, Phys. Rev. D 63 (2001) 074505, doi:10.1103/PhysRevD.63.074505, arXiv:hep-lat/0009038.

[4] G.P. Lepage, Redesigning lattice QCD, in: Perturbative and Nonperturbative Aspects of Quantum Field Theory, Schladming Winter School, 1996, pp. 1–48, arXiv:hep-lat/9607076.

[5] S. Capitani, Lattice perturbation theory, Phys. Rep. 382 (2003) 113–302, doi:10.1016/S0370-1573(03)00211-4, arXiv:hep-lat/0211036.

[6] H.D. Trottier, Higher-order perturbation theory for highly-improved actions, Nucl. Phys. (Proc. Suppl.) 129 (2004) 142–148, doi:10.1016/S0920-5632(03)02515-5, arXiv:hep-lat/0310044.

[7] G.P. Lepage, P.B. Mackenzie, N.H. Shakespeare, H.D. Trottier, Perturbative two- and three-loop coefficients from large beta Monte Carlo, Nucl. Phys. (Proc. Suppl.) 83 (2000) 866–871, arXiv:hep-lat/9910018.

[8] F. Di Renzo, L. Scorzato, A consistency check for renormalons in lattice gauge theory: $\beta^{-10}$ contributions to the SU(3) plaquette, JHEP 0110 (2001) 038, arXiv:hep-lat/0011067.

[9] R. Horsley, P.E.L. Rakow, G. Schierholz, Separating perturbative and non-perturbative contributions to the plaquette, Nucl. Phys. (Proc. Suppl.) 106 (2002) 870–872, doi:10.1016/S0920-5632(01)01870-9, arXiv:hep-lat/0110210.

[10] H.D. Trottier, N.H. Shakespeare, G.P. Lepage, P.B. Mackenzie, Perturbative expansions from Monte Carlo simulations at weak coupling: Wilson loops and the static-quark self-energy, Phys. Rev. D 65 (2002) 094502, doi:10.1103/PhysRevD.65.094502, arXiv:hep-lat/0111028.

[11] A. Hart, R.R. Horgan, L.C. Storoni, Perturbation theory vs. simulation for tadpole improvement factors in pure gauge theories, Phys. Rev. D 70 (2004) 034501, doi:10.1103/PhysRevD.70.034501, arXiv:hep-lat/0402033.

[12] I.F. Allison, et al., Matching the bare and MSbar charm quark masses using weak coupling simulations, PoS LAT2008 (2008) 225, arXiv:0810.0285.

[13] F. Di Renzo, L. Scorzato, C. Torrero, High loop renormalization constants by NSPT: a status report, PoS LAT2007 (2007) 240, arXiv:0710.0552.

[14] G.P. Lepage, A new algorithm for adaptive multidimensional integration, J. Comput. Phys. 27 (1978) 192, doi:10.1016/0021-9991(78)90004-9.

[15] G.P. Lepage, Vegas: an adaptive multidimensional integration program, Cornell preprint CLNS-80/447, March 1980.

[16] J.A.M. Vermaseren, New features of FORM (announcement), arXiv:math-ph/0010025, 2000.

[17] A. Hart, G.M. von Hippel, R.R. Horgan, L.C. Storoni, Automatically generating Feynman rules for improved lattice field theories, J. Comput. Phys. 209 (2005) 340–353, doi:10.1016/j.jcp.2005.03.010, arXiv:hep-lat/0411026.

[18] M. Lüscher, P. Weisz, Efficient numerical techniques for perturbative lattice gauge theory computations, Nucl. Phys. B 266 (1986) 309, doi:10.1016/0550-3213(86)90094-5.

[19] M.A. Nobes, H.D. Trottier, G.P. Lepage, Q. Mason, Second order perturbation theory for improved gluon and staggered quark actions, Nucl. Phys. (Proc. Suppl.) 106 (2002) 838–840, doi:10.1016/S0920-5632(01)01860-6, arXiv:hep-lat/0110051.

[20] M.A. Nobes, H. Trottier, One loop renormalization of Fermilab fermions, Nucl. Phys. (Proc. Suppl.) 119 (2003) 461–463, doi:10.1016/S0920-5632(03)01586-X, arXiv:hep-lat/0209017.

[21] M.A. Nobes, H.D. Trottier, Progress in automated perturbation theory for heavy quark physics, Nucl. Phys. (Proc. Suppl.) 129 (2004) 355–357, doi:10.1016/S0920-5632(03)02580-5, arXiv:hep-lat/0309086.

[22] B. Alles, M. Campostrini, A. Feo, H. Panagopoulos, Lattice perturbation theory by computer algebra: A three loop result for the topological susceptibility, Nucl. Phys. B 413 (1994) 553–566, doi:10.1016/0550-3213(94)90632-7, arXiv:hep-lat/9301012.

[23] E. Follana, et al., Highly improved staggered quarks on the lattice, with applications to charm physics, Phys. Rev. D 75 (2007) 054502, doi:10.1103/PhysRevD.75.054502, arXiv:hep-lat/0610092.

[24] G.P. Lepage, L. Magnea, C. Nakhleh, U. Magnea, K. Hornbostel, Improved nonrelativistic QCD for heavy quark physics, Phys. Rev. D 46 (1992) 4052–4067, doi:10.1103/PhysRevD.46.4052, arXiv:hep-lat/9205007.

[25] M. Lüscher, P. Weisz, On-shell improved lattice gauge theories, Commun. Math. Phys. 97 (1985) 59, doi:10.1007/BF01206178.

[26] M. Lüscher, P. Weisz, Computation of the action for on-shell improved lattice gauge theories at weak coupling, Phys. Lett. B 158 (1985) 250, doi:10.1016/0370-2693(85)90966-9.

[27] G.M. de Divitiis, R. Petronzio, N. Tantalo, On the discretization of physical momenta in lattice QCD, Phys. Lett. B 595 (2004) 408–413, doi:10.1016/j.physletb.2004.06.035, arXiv:hep-lat/0405002.

[28] I.T. Drummond, A. Hart, R.R. Horgan, L.C. Storoni, One loop calculation of the renormalised anisotropy for improved anisotropic gluon actions on a lattice, Phys. Rev. D 66 (2002) 094509, doi:10.1103/PhysRevD.66.094509, arXiv:hep-lat/0208010.

[29] I.T. Drummond, A. Hart, R.R. Horgan, L.C. Storoni, The contribution of $\mathcal{O}(\alpha)$ radiative corrections to the renormalised anisotropy and application to general tadpole improvement schemes, Phys. Rev. D 68 (2003) 057501, doi:10.1103/PhysRevD.68.057501, arXiv:hep-lat/0307010.

[30] I.T. Drummond, A. Hart, R.R. Horgan, L.C. Storoni, Lattice perturbation theory for gluonic and fermionic actions, Nucl. Phys. (Proc. Suppl.) 119 (2003) 470–475, doi:10.1016/S0920-5632(03)01589-5, arXiv:hep-lat/0209130.

[31] A. Hart, G.M. von Hippel, R.R. Horgan, Leptonic widths of heavy quarkonia: S-Wave QCD/NRQCD matching coefficients for the electromagnetic vector annihilation current at $O(\alpha_s v^2)$, Phys. Rev. D 75 (2007) 014008, doi:10.1103/PhysRevD.75.014008, arXiv:hep-lat/0605007.

[32] A. Hart, G.M. von Hippel, R.R. Horgan, Leptonic widths of heavy quarkonia: QCD/NRQCD matching for the electromagnetic current at $O(\alpha_s v^2)$, PoS LAT2006 (2006) 098, arXiv:hep-lat/0609002.

[33] A. Hart, G.M. von Hippel, R.R. Horgan, Heavy quarkonium decays on and off the lattice, in: G. Grindhammer, K. Sachs (Eds.), Proceedings of 15th International Workshop on Deep-Inelastic Scattering and Related Subjects (DIS2007), 2007, p. 162, doi:10.3360/dis.2007.162.
[34] A. Hart, G.M. von Hippel, R.R. Horgan, Perturbative calculations for the HISQ action: the gluon action at $O(N_f\alpha_s a^2)$, PoS LATTICE2008 (2008) 046, arXiv:0808.1791.
[35] A. Hart, G.M. von Hippel, R.R. Horgan, Radiative corrections to the lattice gluon action for HISQ improved staggered quarks and the effect of such corrections on the static potential, Phys. Rev. D 79 (2009) 074008, doi:10.1103/PhysRevD.79.074008.
[36] S. Meinel, et al., Rare B decays with moving NRQCD and improved staggered quarks, PoS LAT2008 (2008) 280, arXiv:0810.0921.
[37] B. Borasoy, G.M. von Hippel, H. Krebs, R. Lewis, Automated methods in chiral perturbation theory on the lattice, PoS LAT2005 (2006) 038, arXiv:hep-lat/0509007.
[38] S. Takeda, U. Wolff, Automatic generation of vertices for the Schroedinger functional, PoS LAT2007 (2007) 257, arXiv:0709.4167.
[39] S. Takeda, Automatic generation of Feynman rules in the Schroedinger functional, Nucl. Phys. B 811 (2009) 36–65, doi:10.1016/j.nuclphysb.2008.11.022, arXiv:0808.3065.
[40] J. Foley, C. Morningstar, Tuning improved anisotropic actions in lattice perturbation theory, PoS LAT2008 (2008) 212, arXiv:0810.4477.
[41] G. 't Hooft, A property of electric and magnetic flux in nonabelian gauge theories, Nucl. Phys. B 153 (1979) 141.
[42] G. Parisi, Prolegomena to any future computer evaluation of the QCD mass spectrum, invited talk given at Summer Inst. Progress in Gauge Field Theory, Cargese, France, Sep. 1–15, 1983.
[43] R. Wohlert, Improved continuum limit lattice action for quarks, DESY 87/069, 1987.
[44] Automatic differentiation packages are discussed at http://www.autodiff.org.
[45] G.M. von Hippel, TaylUR, an arbitrary-order automatic differentiation package for Fortran 95, Comput. Phys. Comm. 174 (2006) 569–576, doi:10.1016/j.cpc.2005.12.016, arXiv:physics/0506222.
[46] G.M. von Hippel, New version announcement for TaylUR, an arbitrary-order diagonal automatic differentiation package for Fortran 95 (announcement), arXiv:0704.0274, 2007.
[47] A. Bazavov, et al., HISQ action in dynamical simulations, PoS LAT2008 (2008) 033, arXiv:0903.0874.
[48] W.-J. Lee, Perturbative improvement of staggered fermions using fat links, Phys. Rev. D 66 (2002) 114504, doi:10.1103/PhysRevD.66.114504, arXiv:hep-lat/0208032.
[49] K. Orginos, D. Toussaint, R.L. Sugar, Variants of fattening and flavor symmetry restoration, Phys. Rev. D 60 (1999) 054503, doi:10.1103/PhysRevD.60.054503, arXiv:hep-lat/9903032.
[50] Python interpreters and further information is available from http://www.python.org.
[51] DEISA Extreme Computing Initiative (DECI), http://www.deisa.eu/science/deci.
[52] C. Morningstar, M.J. Peardon, Analytic smearing of SU(3) link variables in lattice QCD, Phys. Rev. D 69 (2004) 054501, doi:10.1103/PhysRevD.69.054501, arXiv:hep-lat/0311018.
[53] M. Lüscher, P. Weisz, Background field technique and renormalization in lattice gauge theory, Nucl. Phys. B 452 (1995) 213–233, doi:10.1016/0550-3213(95)00346-T, arXiv:hep-lat/9504006.
[54] M. Lüscher, P. Weisz, Computation of the relation between the bare lattice coupling and the MS coupling in SU($N$) gauge theories to two loops, Nucl. Phys. B 452 (1995) 234–260, doi:10.1016/0550-3213(95)00338-S, arXiv:hep-lat/9505011.