Status of the Phmc-code

T. Chiarappa

October 2006

Even if a final self-consistent (independent on external softwares) code is still not achieved, in this note I would like to report the status of the PHMC algorithm, describing the way to run the program and the meaning of the related files. Furthermore, I present some tests within the various sections and finally I discuss a ToDo List.

For my purpose, I will take profit of previous notes. Those people who followed the work will recognise several repetitions, needed however to discuss in a possibly satisfactory way some new topics or developments.

1 Setup and Formulæ: still another introduction

Studying many non-perturbative QCD properties, we can consider two quark pairs, a light (l) mass degenerate one (u and d flavours) and a heavier (h) mass non-degenerate one (s and c flavour). We will adopt the tmLQCD formulation for our Monte Carlo simulations, whose action is

$$S = S_G[U] + \bar{\psi}_l \; D_l[U] \; \psi_l + \bar{\psi}_h \; D_h[U] \; \psi_h \quad , \quad \psi_l = [u,d] \quad , \quad \psi_h = [s,c] \; , \tag{1}$$

where S_G is a suitable pure gauge action.
While for the light sector an Hmc algorithm has been shown to work very well, we will mainly concentrate on an algorithm for the heavy quarks sector and its possible coupling to the existing Hmc.

Let me remind briefly the formulæ for the non-degenerate Dirac operator, suitable for describing the effects of the strange and charm quarks, (s,c).

$$D_h = \Big[\; \gamma\tilde{\nabla} + W_{cr} \; \Big] \, 2\kappa_{cr} + i\bar{\mu}\gamma_5\tau_3 + \bar{\epsilon}\tau_1 \tag{2}$$

$$W_{cr} = -\frac{a}{2}\nabla^*\nabla + M_{cr} \quad , \quad M_{cr} = \frac{1}{2\kappa_{cr}} - 4 \; . \tag{3}$$

In eq. (2), a chiral rotation and a scaling of the quark fields and the mass term by $2\kappa_{cr}$ have been performed. The scalar terms, $\bar{\mu}$ and $\bar{\epsilon}$ are the twisted mass and the mass splitting terms, respectively; note further that the Pauli matrices operate only in flavour space.
Resembling all together for later purposes, in lattice spacing units:

$$S = S_G[U] + \bar{\psi}_{h,x} \left[\delta_{x,y} \left(1 + i\gamma_5\bar{\mu}\tau_3 + \bar{\epsilon}\tau_1\right) - \kappa \sum_{\mu=\pm1}^{\pm4} \delta_{x,y+\mu}(1+\gamma_\mu)U_{y,\mu} \right] \psi_{h,y} \tag{4}$$

Properties as $O(a)$-improvement are obtained in tmLQCD at maximal twist, which can be achieved by setting the hopping parameter κ to a sensible estimate of κ_{cr}.
For convenience we work with the γ_5 hermitian partner of the above Dirac operator

$$\begin{aligned} Q = \gamma_5 D_h &= \begin{bmatrix} \tilde{Q} + i\bar{\mu} & \gamma_5\bar{\epsilon} \\ \gamma_5\bar{\epsilon} & \tilde{Q} - i\bar{\mu} \end{bmatrix} = \begin{bmatrix} Q_+ & \epsilon\gamma_5 \\ \epsilon\gamma_5 & Q_- \end{bmatrix} , \\ \tilde{Q} &= \gamma_5 \left[\gamma\tilde{\nabla} - \frac{a}{2}\nabla^*\nabla + \frac{1}{2\kappa} - 4 \right] 2\kappa \ . \end{aligned} \tag{5}$$

.

In a path integral framework, the Gaussian integration over the Grassman variables yields the determinant of the Dirac operator, not altered by the above γ_5 multiplication. Finally, we will stochastically evaluate the determinant of the Dirac operator in the heavy quark mass sector on a 2-spinors object Φ_h, as:

$$\det[Q] \ \Leftrightarrow \ \int \mathcal{D}\Phi_h \ e^{-\Phi_h^\dagger (QQ^\dagger)^{-1/2}\Phi_h} \ \simeq \ \int \mathcal{D}\Phi_h \ e^{-\Phi_h^\dagger P(QQ^\dagger)\Phi_h} \ , \qquad \Phi_h = \begin{pmatrix} \phi_{up} \\ \phi_{dn} \end{pmatrix} . \tag{6}$$

2 Intervals and Polynomials

Since in eq. (6) the Dirac operator, $QQ^\dagger$, is a two-flavour non-degenerate matrix, we cannot rely on the usual Hmc. Therefore we are forced to adopt a polynomial approximation:

$$P = P_{n,\epsilon}(S) = \frac{1}{\sqrt{S}} \left[1 + R_{n,\epsilon}(S) \right] \quad , \quad S = QQ^\dagger, \tag{7}$$

where, in contrast to the previous notes [1] and [2], starting from eq. (5) I suppressed EO-preconditioning indices (*hats*), for simplicity.
In eq. (7) the polynomial is implicitly intended as optimised to approximate the function in the normalised interval $[\epsilon, 1]$, where the lower extrema is determined as in eq. (15). The degree, n, has been determined iteratively as described in the following.

The polynomial approximations, eqs. (7) and (12), are based on Chebyshev polynomials and are constructed following the Clenshaw recursion relations. This procedure allows to get roundoff under control as well as to employ truncated polynomials from pre-calculated N_{max} Chebyshev coefficients, d_j.
Instead of the criterion explained in [4], the polynomial degree is determined by an iterative process until a stringent stopping criterion, involving only the sum of the unused coefficients, is fulfilled.
Comparing to the criterion discussed in [4], the achieved precisions goes as [1]

$$\sum_{j=n+1}^{N_{max}} |d_j| \sim \delta \quad , \quad \frac{||(PSP-1)r_h||^2}{||2r_h||^2} \sim \delta^2 \quad , \quad r_h = \text{random 2-spinors} \ . \tag{8}$$

Additionally, we also check that the second formula in eq. (8) evaluated on the real number ϵ, the most critical point in the approximation interval, is of the same order of

[1] The behaviour is expected to be valid for relatively large lattice extent. On small lattices, toy models, the criterion turns out to be even more severe.

magnitude as δ.
The accuracy parameter δ is now handled as an input parameter called `acc_Pfirst`, with default value 10^{-2}.

The polynomial in eq. (7) is playing a crucial rôle, since it enters also the Molecular Dynamic. To this aim, we will adopt the product representation in terms of its complex roots z_j:

$$P_{n,\epsilon}(S)\Phi_h = \left[\prod_{j=1}^{n} c_j(S - z_j)\right]\Phi_h = B(S)B(S)^\dagger\ \Phi_h\ , \tag{9}$$

$$B(S)\ \Phi_h = \sqrt{c_n}(Q\tau_1 - r_n)\cdot\ldots\cdot\sqrt{c_1}(Q\tau_1 - r_1)\ \Phi_h \quad , \quad r_k^* = r_{2n+1-k}\ , \tag{10}$$

where it was already assumed that the polynomial degree is even [2], such as to express the product over the n complex conjugate z_j as running over $2n$ appropriately ordered square roots of roots $r_j = \sqrt{z_j}$, see [7]. This is possible thanks to the hermiticity properties of $Q\tau_1$ (not Q itself). It was also proved that the coefficients c_j remain constant in the whole approximation interval, are independent on the roots and equal to each other, hence they can be merged together into a unique $C^n = \prod c_i$.
Following [7], we adopt a Bit-Reversal ordering of the roots z_j to get roundoff under control in eq. (10).

In the notes [1] and [3] it was already pointed out the possibility to handle a second polynomial approximation, in the determination of both the pseudofermion as well as the final Hamiltonian. It has been shown that the analytically correctness of the algorithm depends exactly on this second polynomial.
Even if the determination of the pseudofermion is not unique, in [3] it was claimed that a suitable solution is given by

$$\Phi_h = \tilde{P}\ B^\dagger\ Q\ r_h \quad , \quad r_h = \text{random 2-spinors} \tag{11}$$

$$\tilde{P} = \tilde{P}_{\tilde{n},\tilde{\epsilon}}(S) = \frac{1}{P_{n,\epsilon}(S)\ \sqrt{S}}\left[1 + \tilde{R}_{\tilde{n},\tilde{\epsilon}}(S)\right], \tag{12}$$

correctly distributed and such as to approximate the exponent in eq. (6), through [3]

$$\left|\left(\tilde{P}\ B^\dagger\ Q\right)^{-1}\Phi_h\right|^2 = \Phi_h^\dagger\ \left(1 + \tilde{R}_{\tilde{n},\tilde{\epsilon}}\right)^{-1} P_{n,\epsilon}(S)\ \left(1 + \tilde{R}_{\tilde{n},\tilde{\epsilon}}\right)^{-1}\ \Phi_h. \tag{13}$$

Similarly to eq. (8), we construct this second polynomial approximation out of a precalculated N_{max} Chebyshev coefficients $\tilde{d}_j$, determining iteratively the degree $\tilde{n}$ once the following stopping criterion is fulfilled:

$$\sum_{j=n+1}^{N_{max}} |\tilde{d}_j| \sim \tilde{\delta} \quad , \quad \frac{||(\tilde{P}PSP\tilde{P} - 1)r_h||^2}{||2r_h||^2} \sim \tilde{\delta}^2 \quad , \quad r_h = \text{random 2-spinors}\ . \tag{14}$$

[2] Note moreover that an even degree forbids the zeros of the polynomial to sit on the real axis inside the approximation interval.

[3] Maybe superfluous: note however that Q does not commute neither with $Q^\dagger$ nor with P. Since $P, \tilde{P}, B, B^\dagger$ are linear combination of S, they obviously commute with each other.

Again, the accuracy parameter $\tilde{\delta}$ is treated as an input parameter, `acc_Ptilde`, with default value 10^{-4}.
As already mentioned and discussed in the note [3], $\tilde{P}$ is unavoidable in order to render the algorithm correct, entering both the computation of the final Hamiltonian as well as the computation of the reweighting correction.

With this property in mind, two different approximation intervals can be considered: a conservatively safe and fixed one ($[\tilde{\epsilon}, 1]$), and a variable one ($[\epsilon, 1]$), with the implicit convention that $\tilde{\epsilon} \leq \epsilon$.
The determination of the conservative $\tilde{\epsilon}$ has been performed by evaluating smallest and largest eigenvalues (λ^{cons} and Λ^{cons}) [4] on a uniformly distributed random gauge configuration (by which I mean a gauge configuration $U = e^{i\alpha^a \omega_a}$ with ω_a flat randomly distributed in the group interval $[0, 2\pi]$), checking that the corresponding new files `start.c/h-NEW` are copied to `start.c/h`. Empirically, it has been proved on a moderately small lattice, $8^3 \times 16$, that the eigenvalues (λ_j and Λ_j) evaluated on successively updated configurations (j) fulfils the expected condition: $\lambda_j > \lambda^{cons}$ and $\Lambda_j < \Lambda^{cons}$. This characteristic is expecting to be preserved for even larger lattices while it is hardly valid for toy models, since the larger the volume the higher the probability to find different eigenvalues.
We therefore define

$$\tilde{\lambda} = \lambda^{cons} \quad , \quad \tilde{\Lambda} = \Lambda^{cons} \quad , \quad \tilde{\epsilon} = \frac{\tilde{\lambda}}{\tilde{\Lambda}} \quad , \quad S \to S^{norm} = S \, \frac{1}{\tilde{\Lambda}} \tag{15}$$

Note moreover that the computationally time consuming evaluation of the conservative eigenvalues can be avoided if the, so far only empirically proved, following relations are fulfilled

$$\lambda^{cons} \propto 2 \, (\bar{\mu}^2 - \bar{\epsilon}^2) \quad , \quad \Lambda^{cons} \propto \frac{1}{2 \, \sqrt{\bar{\mu}^2 + \bar{\epsilon}^2}} \, . \tag{16}$$

The proportionality is intended as a tentative values for the conservative eigenvalues: the constant value 2 can be for instance increased by another $10 \div 20$ %. Obviously, in order to really gain CPU-time, these values should not yield a too pessimistic approximation interval [5]. Since this point surely requires some experience, I think it is more reliable the full $\tilde{\lambda}$, $\tilde{\Lambda}$ computation.
Recall in fact that the eigenvalues have to be evaluated only once at the beginning of the thermalisation; moreover, for moderately or large lattices, one could also rely the eigenvalue computation on a single uniformly distributed random gauge configuration, while for small lattices an average over several eigenvalues must be considered.

Once a conservative interval has been determined, together with a high degree polynomial $\tilde{P}$, we are free to choose a less accurate polynomial P in a smaller interval $[\epsilon, 1]$. The advantage consists in performing the very time demanding Molecular Dynamics with a polynomial relatively low degree.

[4] To this aim we implemented the Jacobi-Davidson procedure for a *bispinor* structure. See note [2] for descriptions, discussions and tests.

[5] Due to the formula for $\tilde{\epsilon}$, for fixed $\tilde{\delta}$ the polynomial degree $\tilde{n}$ is rapidly increasing, requiring therefore a *safe* estimate for the proportionality factor.

Inspecting the eqs. (20) and (21), an analytically exact algorithm requires to fix the two intervals and degrees, $\tilde{\epsilon}, \epsilon$ and $\tilde{n}, n$.
However, one could invest some time at the very beginning (while thermalising) in order to find the best setup for the polynomial P, adjusting ϵ and n. Once this optimal setup has been found, the code should be used first to thermalise to the correctly distributed configuration (expected to be quite fast) and finally for production.
Aiming at the best setup, a new input parameter `recev` has been introduced, deciding after how many trajectories the eigenvalues λ and Λ have to be recomputed. Consequently, a new polynomial P together with the necessary new roots have to be newly determined.
The value of `recev` is possibly to be chosen from empirical experience, since the mentioned evaluation of P and related roots can be rather time consuming, especially if external softwares are employed.

Due to the roots problems discussed in the next section, the lines in the code involving `recev` have been commented. Additionally its default value is set to be 10^{+5} (and must be set to ∞ in the production runs).

3 Root evaluation

This is exactly the point where the code is still affected by some troubles.
I developed a routine computing the roots taking profit of the implemented library CLN [6], but so far, unsuccessfully.
The basic algorithm follows the Laguerre method [6] for polynomial roots computations, adapted to polynomials constructed via Clenshaw recursion relations. Additionally, I have incorporated the real coefficients properties in the deflation process.
The Laguerre method is considered safe for what concern convergence to the desired solution and, at least in my case, it also turn to be very fast in the computation of the first root z_1 (I. Montvay, using a standard representation as in eq. (32), claims it is very fast for all the roots they are looking for).
Since the roots comes in conjugate pairs, I developed a double deflation procedure to produce another real coefficients polynomial, Q, of degree $n' = n - 2$

$$Q_{n',\epsilon}(s) = \frac{P_{n,\epsilon}(s)}{(s - z_1)(s - z_2)} = \frac{P_{n,\epsilon}(s)}{(s - z_1)(s - z_1^\star)} \quad , \quad s \in \mathcal{C}$$

The polynomial Q is constructed out of coefficients b_k determined as follows

$$\begin{aligned} P_{n,\epsilon}(s) &= (s - z_1)(s - z_1^\star)\ Q_{n',\epsilon}(s) \\ P_{n,\epsilon}(s) &= (s^2 - 2As + D)\ Q_{n',\epsilon}(s) \quad , \quad A = \Re(z_1) \quad , \quad D = |z_1|^2 \\ \frac{1}{2}d_0T_0 + \sum_{j=1}^{n} d_jT_j &= (s^2 - 2As + D)\ \left[\frac{1}{2}b_0T_0 + \sum_{k=1}^{n'=n-2} b_kT_k\right] , \end{aligned} \tag{17}$$

[6] CLN = Class Library for Numbers handles number classes with extended large precision. It requires to use the `g++` C++ compiler.

where T_k are the k-degree Chebyshev polynomials. By means of the recursions

$$T_{k+1} = 2s\ T_k - T_{k-1} \quad \Rightarrow \quad s\ T_k = \frac{1}{2}\left(T_{k+1} + T_{k-1}\right)$$

the real coefficients b_k can be found iteratively comparing term by term the coefficients of the T_k on the left- and right- hand side of eq. (17), obtaining

$$\begin{aligned}
b_{n'} &= b_{n-2} = 4 \cdot d_n \\
b_{n'-1} &= b_{n-3} = 4 \cdot [d_{n-1} + A \cdot b_{n-2}] \\
b_{n'-2} &= b_{n-4} = 4 \cdot \left[d_{n-2} + A \cdot b_{n-3} - \left(D + \frac{1}{2}\right) \cdot b_{n-2}\right] \\
b_{n'-3} &= b_{n-5} = 4 \cdot \left[d_{n-3} + A \cdot (b_{n-4} + b_{n-2}) - \left(D + \frac{1}{2}\right) \cdot b_{n-3}\right] \\
b_{n'-4} &= b_{n-6} = 4 \cdot \left[d_{n-4} + A \cdot (b_{n-5} + b_{n-3}) - \left(D + \frac{1}{2}\right) \cdot b_{n-4}\right] - b_{n-2} \\
\ldots &= \ldots \\
b_{n'=0} &= b_{n-(n-2)} = 4 \cdot \left[d_{n-(n-2)} + A \cdot \left(b_{n-(n-3)} + b_{n-(n-1)}\right) - \left(D + \frac{1}{2}\right) \cdot b_{n-(n-2)}\right] - b_{n-(n-4)}
\end{aligned} \tag{18}$$

These coefficients however undergoes two problems:

1. two additional equations did **not** match with the complete list above

$$\begin{aligned}
\left(D + \frac{3}{4}\right) \cdot b_{n'=1} - A \cdot b_{n'=0} &= d_1 - \frac{1}{4} b_{n'=3} + A \cdot b_{n'=2} \\
\frac{1}{2}\left(D + \frac{1}{2}\right) \cdot b_{n'=0} - A \cdot b_{n'=1} &= \frac{1}{2} d_0 - \frac{1}{4} b_{n'=2}
\end{aligned}$$

2. the properties characterising the d_j, see for instance eq. (5.8.7) in [5], seems to be **not** preserved. Inspecting for instance eq. (18) for b_{n-2}, there is a discrepancy between the derived coefficient, i), and the one as if it would be computed exactly, ii),

$$\begin{aligned}
i) \qquad & 4 \cdot \frac{2}{n+1} \sum_{k=1}^{n+1} f\left[\cos\left(\frac{\pi(k-\frac{1}{2})}{n+1}\right)\right] \cos\left(\frac{\pi n (k-\frac{1}{2})}{n+1}\right) \\
ii) \qquad & \frac{2}{n-1} \sum_{k=1}^{n-1} f\left[\cos\left(\frac{\pi(k-\frac{1}{2})}{n-1}\right)\right] \cos\left(\frac{\pi (n-2)(k-\frac{1}{2})}{n-1}\right)
\end{aligned}$$

The file where the above equations has been coded is called `roots_clenshaw.c/h`.

For these unsolved problems we rely so far on an external Mathematica notebook, which computes the polynomial roots by a nested damped Newton's, secant and Brent's method. Beside being very slow, it is based moreover on the *naive* Chebyshev approximation (linear combination of Chebyshev polynomials, no Clenshaw): roundoff errors are controlled by working symbolically (means with very high precision).

As a cross effect, I discover a difference between the two polynomial representations for P in eqs. (7) and (9). Evaluating the scalar product between two 2-spinors, η and $\eta' = P\eta$, we expect the scalar product to be a real number, since P is a linear combination of the hermitian operator S.
This is the case when we choose η as a random 2-spinors r_h, while the relation is not satisfied when we consider η as derived from a mixed application of B and $\tilde{P}$

$$\langle \eta, P\ \eta \rangle = \begin{pmatrix} 2.7870571922 \cdot 10^{+03} - 2.3980817332 \cdot 10^{-14}\ i \\ 2.8499859650 \cdot 10^{+03} + 1.6264767311 \cdot 10^{-14}\ i \end{pmatrix} \quad , \quad \eta = r_h$$

$$\langle \eta, P\ \eta \rangle = \begin{pmatrix} 9.0779290403 \cdot 10^{+04} + 7.3660649446 \cdot 10^{+01}\ i \\ 9.0964902205 \cdot 10^{+04} - 7.3660649446 \cdot 10^{+01}\ i \end{pmatrix} \quad , \quad \eta = \tilde{P}\ B^\dagger\ Q r_h = \Phi_h$$

4 Updates

The update has been implemented with the purpose to couple the Hmc contributions to the Phmc contributions. Therefore, a new file called `hybrid_nondegenerate.c/h` has been created containing the main routines evaluating the forces for the non-degenerate case, shortly the Phmc contribution.
The correct sequence of roots used in the force evaluation has been tested as well as the normalisation factors, as described in the note [4]. Finally, the mentioned forces are coupled to the Hmc contributions simply calling already existing routines.
I should remark however that since the Phmc forces computation involves a further multiplication by the overall constant C, it is very important in the `leap_frog_ND` routine to respect the order of the `fermion_momenta` and `fermion_momenta_ND` calls.

Let me remind again that so far the coupling between Phmc and Hmc forces contributions is not automatised, but it is imposed by hand. It would probably be wiser to insert a flag to decide whether to consider this coupling ($N_f = 2 + 1 + 1$ flavours) or to apply the Phmc alone ($N_f = 1 + 1$).
Furthermore, I have implemented only one integration scheme, namely the Leap-Frog scheme, adapted for the non-degenerate case. In view of a possible merging of the Phmc with the latest developments in the Hmc, others integrations schemes for non-degenerate quark masses have possibly to be coded.

5 Final Hamiltonian

Once an updated configuration has been obtained, the program performs a Metropolis Accept/Reject test. As explained in [3], in shorthand notation,

$$H^{start} = \left| \left(\tilde{P} B^\dagger Q \right)^{-1} \Big|_{U_{start}}\ \Phi_h \right|^2 = r_h^\dagger r_h \tag{19}$$

$$H^{end} = \left| \left(\tilde{P} B^\dagger Q \right)^{-1} \Big|_{U_{end}}\ \Phi_h \right|^2 = \Phi_h^\dagger B B^\dagger\ \chi_h \tag{20}$$

where

$$\chi_h = \left(1+\tilde{R}\right)^{-2}\Phi_h = \left(\tilde{P}PSP\tilde{P}\right)^{-1}\Phi_h = \left(1-A+A^2-...\right)\Phi_h \quad , \quad A = \tilde{P}PSP\tilde{P}-1 \; . \tag{21}$$

While the starting Hamiltonian is straightforward, the final Hamiltonian in eq. (20) is very dependent on the magnitude of $\tilde{R}$, hence on the accuracy `acc_Ptilde`.
The algorithm would be exact if we could calculate the whole series in eq. (21). In practise however, it can be argued that only few terms contributes significantly.
Thus we adopt a further accuracy parameter, δ_H, to stop the series when

$$|H^{(n)} - H^{(n-1)}| < \delta_H \quad , \quad H^{(0)} = \Phi_h^\dagger B B^\dagger \Phi_h \; , \tag{22}$$

while $H^{(n)}$ involves n applications of A.
The evaluation of eq. (20) turns to be very demanding, indeed. However, taking profit of the hermiticity of the polynomials, we can calculate each contribution involving an application of A at almost half of the time, relying on the following sequence

$$\varphi^{(0)} = B^\dagger\Phi_h \;\; : \quad H^{(0)} = \left|\varphi^{(0)}\right|^2$$

$$\varphi^{(1)} = Q^\dagger P\tilde{P}\varphi^{(0)} \;\; : \quad H^{(1)} = H^{(0)} + H^{(0)} - \left|\varphi^{(1)}\right|^2 = 2H^{(0)} - \left|\varphi^{(1)}\right|^2$$

$$\varphi^{(2)} = \tilde{P}PQ\varphi^{(1)} \;\; : \quad H^{(2)} = H^{(1)} + H^{(0)} - 2H^{(1)} + \left|\varphi^{(2)}\right| = H^{(0)} - H^{(1)} + \left|\varphi^{(2)}\right|^2$$

.....

$$\varphi^{(n)} = \left\{ \begin{array}{ll} Q^\dagger P\tilde{P} \; , & \text{odd-}n \\ \tilde{P}PQ \; , & \text{even-}n \end{array} \right\} \varphi^{(n-1)} \;\; : \quad H^{(n)} = H^{(n-1)} + \sum_{j=0}^{n-1} \left(\begin{array}{c} n \\ j \end{array} \right) \; H^{(j)} + (-1)^n \left|\varphi^{(n)}\right|^2$$

Let me remark that the accuracy parameter δ_H has yet not been handled as an input parameter, but set, until now, to be equal to $\tilde{\delta}$, since the algorithm precision is governed by `acc_Ptilde`.

Moreover, the introduced roots problems reflects itself also in the behaviour of the corrections to the final Hamiltonian.
The following table illustrates the corrections on toys model, volume = 4^4 and fixed algorithm accuracy $\tilde{\delta}$. We set $\epsilon = \tilde{\epsilon}$.

$\tilde{\epsilon}$	δ	n	$\tilde{\delta}$	$\tilde{n}$	$\mid H^{(1)} - H^{(0)} \mid$	$\mid H^{(2)} - H^{(1)} \mid$	$\mid H^{(3)} - H^{(2)} \mid$
0.0043	10^{-1}	30	10^{-5}	73	$3.352193 \cdot 10^{-5}$	$1.340765 \cdot 10^{-4}$	$3.016974 \cdot 10^{-4}$
0.0043	10^{-2}	48	10^{-5}	70	$2.746273 \cdot 10^{-5}$	$1.098600 \cdot 10^{-4}$	$2.471645 \cdot 10^{-4}$
0.028	10^{-2}	20	10^{-5}	27	$3.567308 \cdot 10^{-2}$	$1.424006 \cdot 10^{-1}$	$3.210541 \cdot 10^{-1}$
0.028	10^{-5}	36	10^{-5}	36	$1.834568 \cdot 10^{-2}$	$7.325222 \cdot 10^{-2}$	$1.651101 \cdot 10^{-1}$
0.028	10^{-9}	62	10^{-9}	62	$1.398236 \cdot 10^{-3}$	$5.591353 \cdot 10^{-3}$	$1.258412 \cdot 10^{-2}$
0.028	10^{-5}	36	10^{-9}	52	$6.437184 \cdot 10^{-4}$	$2.574702 \cdot 10^{-3}$	$5.793465 \cdot 10^{-3}$
0.028	10^{-2}	20	10^{-9}	54	$1.083810 \cdot 10^{-3}$	$4.334610 \cdot 10^{-3}$	$9.754292 \cdot 10^{-3}$

From these results it seems that the following conclusions can be drawn

1. there is a strong dependency on the approximation interval: the smaller $\tilde{\epsilon}$ the smaller the first correction;

2. the dependency on the accuracy parameter $\tilde{\delta}$ is appreciable;

3. the dependency on the accuracy parameter δ appears to be almost negligible;

4. large n involves much more apparent problems related to the root computation (notice the *anomaly* of the 5^{th} row);

It is reasonable to expect that the higher the correction the more rounding errors can affect the calculations.
What is **not** convincing however is that this happens so fast, already by quite large energy differences [7].

6 Reweighting

Even if this part has yet not been coded, let me recall the first reweighting correction as mentioned in eq. (14) of note [3]. Due to the polynomial approximation, there is a discrepancy between the gauge configuration distributions we will achieve, *Code*, and the target one, *Aim*, namely

$$Code: \det\left[\left(1+\tilde{R}\right)P^{-1}\left(1+\tilde{R}\right)\right] \quad , \quad Aim: \det\left[\sqrt{S}\right]$$

The correction factor $W_h[U]$, compensating for this discrepancy, will then be given by

$$W_h[U] = \det\left[\sqrt{S}P\left(1+\tilde{R}\right)^{-2}\right] = \det\left[\sqrt{S}P\left(1+\tilde{R}\right)^{-1}\right] \det\left[\left(1+\tilde{R}\right)^{-1}\right]$$
$$W_h[U] = W_h^{(1)}[U]\cdot W_h^{(2)}[U] = \det\left[\tilde{P}^{-1}\right] \det\left[\left(1+\tilde{R}\right)^{-1}\right] ,$$

where I implicitly assumed that the operators S, P and $\tilde{P}$ are evaluated on the last accepted configuration. If the correction due to W_h turns out to be so small to be effectively neglected, one can choose to evaluate the reweighting each $10 \to 100$ configurations only as a test.
For the precision of $\tilde{P}$ we are planning to achieve, we claim that eventually only the first term, $W_h^{(1)}$, will contribute. In this case the reweighting factor will consist only on the stochastic evaluation of

$$W_h^{(1)}[U] = \det[\tilde{P}^{-1}] \simeq \int D\eta_1 \; e^{-\eta_1^\dagger \eta_1} \; \exp\left[\eta_1^\dagger\left(1-\tilde{P}\right)\eta_1\right] \tag{23}$$

To this purpose, we only have to produce another polynomial

$$\bar{P} = 1 - \tilde{P}$$

[7] I have honestly to admit that when these corrections were coded, similar tests turned out to be much more satisfactory. Unfortunately I lost the corresponding files.

to be applied to a large ($O(20 \div 50)$?) number of random pseudofermions, η_1.
The last equation should not scared since this third polynomial is straightforwardly constructed out of the $\tilde{d}_j$ coefficients, just replacing $\bar{d}_0 = 2 - \tilde{d}_0$ and $\bar{d}_i = -\tilde{d}_i$, $i \neq 0$, as straightforward referring at the expression of $\tilde{P}$ in eq. (17).
Following this procedure one continues getting again rounding error under control.
Note moreover that eq. (23) can be evaluated *off-line*, namely externally on the running code.

7 Notation

In order to run a performance test between the Phmc code developed by the Hamburg (HH) group and our code, one should notice the different notation.
Let me therefore repeat the formulæ for the action, at least in the heavy sector, given by the HH-group as in eqs. (8), (9), and (6) of [8]:

$$S_{HH} = S_G[U] + \bar{\chi}_x \left[\delta_{x,y} \left(\mu_k + i\gamma_5\tau_1 a\mu_\sigma + \tau_3 a\mu_\delta \right) - \frac{1}{2} \sum_{\mu=\pm 1}^{\pm 4} \delta_{x,y+\mu}(1+\gamma_\mu) U_{y,\mu} \right] \chi_y \quad , \quad (24)$$

where the scalar term μ_k is related to the critical mass, while μ_σ and μ_δ are proportional to the input mass parameter $\bar{\mu}$ and $\bar{\epsilon}$ respectively.
Contrary as in eqs. (2), (3) and (4), the matter fields in eq. (24) are **not** scaled. Furthermore, the HH matter fields, χ, are related through a pure flavour transformation, something like a projection, to *our* matter fields, ψ.
All in one:

$$\psi_h = \frac{1}{\sqrt{2}} (1 - i\tau_2) \ \frac{1}{\sqrt{2\kappa}} \ \chi \quad , \quad \bar{\psi}_h = \frac{1}{\sqrt{2\kappa}} \ \bar{\chi} \ \frac{1}{\sqrt{2}} (1 + i\tau_2) \ , \quad (25)$$

Thus, the scalar terms in eqs. (24) and (4) are related as follow:

$$\mu_k = 2\kappa \ aM_{cr} \quad , \quad \bar{\mu} = 2\kappa \ \mu_\sigma \quad , \quad \bar{\epsilon} = 2\kappa \ \mu_\delta \quad . \quad (26)$$

As far as the pure run is concerned, we have only to convert the input mass parameters, $\bar{\mu}$ and $\bar{\epsilon}$, as described in eq. (26).

Obviously it would be nice to unify the notation, choosing the best one would however require some discussion.

8 Tests

In this section, I will report about the results comparison between the our code and the HH-group's one.

On a 16×8^3 lattice, The HH-group used its code with the following inputs:

$$\beta = 3.30 \quad , \quad C1 = -0.08333333 \quad , \quad \kappa = 0.170$$
$$\mu_\sigma^l = 0.01 \quad , \quad \mu_\delta^l = 0.0 \quad , \quad \mu_\sigma^h = 0.325 \quad , \quad \mu_\delta^h = 0.275 \ ,$$

where the mass superscripts l, h distinguish the light from the heavy quark masses. With respect to eq. (24), we concentrate only on μ_σ^h and μ_δ^h.
The results the HH-group obtained are

$$\text{Plaq} \ = 0.4831 \pm 0.0002 \quad , \quad \tau_{int} = 19.679 \quad , \quad \sigma = 0.002437 \ ,$$

where the "naive" variance σ has been calculated assuming that the configuration after every trajectory is statistically independent. The calculated autocorrelations starts at 2000, while the trajectories used for calculating autocorrelations are $5601 = Nmeas$.
The reported error has been calculated from

$$\text{Error} \ = \sigma \cdot \sqrt{\frac{2\tau_{int}}{Nmeas}} \tag{27}$$

For a comparison, we follow eq. (26) to set the mass parameters, thermalise for around 2000 trajectories Once we calculate the mass parameters as in eq. (26), we start the runs mainly on parallel architectures. Discarding the first 1104 trajectories for supposed thermalisation, we use $Nmeas = 5500$ and a JackKnife method to obtain

$$\text{Plaq} \ = 0.4838 \pm 0.00016 \tag{28}$$

which is **not** consistent with the HH-result.
Note however that the above results have been obtained using "wrong" roots, namely while Mathematica is evaluated exactly n Chebyshev coefficients, the C-code is handling n out of N_{max}, leading therefore to a slightly (but sufficiently) different polynomial construction.
On a chronologically later stage it has been proved the above statement, showing that the founded roots are not roots for the polynomial constructed as in our C-code.

The motivation of using a truncated polynomial is mainly due to have a direct control over the polynomial approximation. In fact, a high (2000) polynomial degree using 2000 coefficients approximates, say, perfectly the argument function. As shown in eq. (8), keeping only n out of N_{max} coefficients guarantees the desired level of accuracy.
Not the same (or at least not that simply) can be inferred by evaluating and using exactly only n coefficients.
Nevertheless, since we are interested in a rather modest accurate polynomial approximation for P, we can even lower the approximation confidence by **imposing** to evaluate only n coefficients (means $N_{max} = n$).
By this method, the Mathematica and C-code coefficients are the same, as will be shown later, and the corresponding roots are correct.

We therefore repeat a similar comparison on a smaller lattice, 4^4, with the same parameters. The HH-outcomes are

$$\text{Plaq } = 0.5339 \pm 0.0002 \quad , \quad \tau_{int} = 1.276 \quad , \quad \sigma = 0.01030 \ ,$$

measured over a total of 7731 trajectories.
Our results, discarding again 2000 trajectories are

$$\text{Plaq } = 0.5305 \pm 0.0004 \tag{29}$$

Only for completeness, I report also the analysis of the data obtained by using different coefficients (see discussion above for the 16×8^3 lattice)

$$\text{Plaq } = 0.5320 \pm 0.0005 \tag{30}$$

The correctness of the above method (coherent Mathematica vs. C-code coefficient calculation) reflect itself also in

$$\Delta H = \left\{ \begin{array}{c} 0.00026 \pm 0.0003 \\ 0.3937 \pm 0.0005 \end{array} \right\} \quad \Rightarrow \quad \exp(-\Delta H) = \left\{ \begin{array}{c} 1.000235 \pm 0.000342 \\ 0.6753 \pm 0.0003 \end{array} \right\} \ , \tag{31}$$

where the (second) first line refers to a (un-)coherent evaluation of Chebyshev coefficients.

9 ToDo List

In this section, I will present some checks of possible implementation. The discussion is mostly related to the possibilities to overcome the roots-problem, insisting on evaluating the Chebyshev coefficients d_j differently in Mathematica or in the C-code.
It sounds a little bit anachronistically as compared to the results presented in the previous section, but the outcomes in Sect. (8) have been obtained in the latest days of my work.
Therefore, one could probably skip the following paragraphs as non-interesting. Nevertheless I think they can furnish some insight of the problems encountered by insisting on using non-consistently Mathematica and the C-code.
Additionally, the following checks could yield some idea for further implementations of the code.
Finally, I should apologise for having discussed the subjects in a rather fuzzy way, not following a precise criterion.

9.1 Parallel dimensionality

So far I was able to run the code **only** on a 1- and 2- parallel dimensions setup. Carsten suggests to look for possible dimension-n division anywhere in the new files.

9.2 CLN

Beside the file `roots_clenshaw.c/h` containing the code for the Laguerre method and double deflation procedure, possibly suitable for polynomials with real coefficients, the CLN library can be useful in the construction of the *standard* representation:

$$P = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_{n-1} x^{n-1} + a_n \cdot x^n \ . \tag{32}$$

The evaluation of the coefficients a_i from a polynomial expressed in the Chebyshev basis, as given in the first equation of (17), will require in fact an extended calculation accuracy.
In the following I will give you my formulæ for these coefficients even if a CLN routine is yet not ready.

In order to give you the possibility to crosscheck my formulæ for these coefficients, let me remind the construction of P à la Clenshaw, starting with the Chebyshev coefficients, d_k. Restricting the attention to a real argument, x, for simplicity, one first has to change variable, $x \to y$, in order to fit the $[-1, 1]$ approximation interval, and then iterate the following

$$\begin{aligned}
& x \in [\epsilon, 1] \qquad x \to y = Ax - B = \frac{2}{1-\epsilon} x - \frac{1+\epsilon}{1-\epsilon} \qquad y \in [-1, 1] \\
& v_{n+2} = 0 \quad , \quad v_{n+1} = 0 \\
& v_j = 2A \ x \ v_{j+1} - 2B \ v_{j+1} - v_{j+2} + d_j \qquad , \qquad j = n-1, n-2, ..., 1 \\
& P \equiv 2A \ x \ v_1 - 2B \ v_1 - v_2 + \frac{1}{2} d_0 \ \ ,
\end{aligned} \tag{33}$$

assuming the polynomial degree being n.

Following this procedure for few examples ($n = 4, 6, 8, 10$), I was able to extrapolate analytically the following formulæ for the coefficients a_0

$$\begin{aligned}
& a_0 = \frac{1}{2} d_0 + \sum_{j=1}^{n/2} (-1)^j \, d_{2j} \ + \ B \left\{ \sum_{i=1}^{n} \left[(2B)^{i-1} \ \sum_{j=1}^{j_{max}} (-1)^{j+i-1} \ d_{2j+i-2} \cdot s_j^{(i)} \right] \right\} \\
& j_{max} \equiv \mathrm{INT} \left(\frac{n-i+1}{2} \right) \qquad ; \qquad s_{j=0}^{(i)} = 0 \quad \forall i = 1, ..., n \\
& s_j^{(i=1)} = (2j-1) \quad ; \quad s_j^{(i=2)} = s_{j-1}^{(i=2)} + s_j^{(i=1)} \quad ; \quad ... \quad ; \quad s_j^{(i=n)} = s_{j-1}^{(i=n)} + s_j^{(i=n-1)} \ ,
\end{aligned}$$

while for the remaining a_k

$$\begin{aligned}
& a_k = 2^{k-1} A^k \ \sum_{i=1}^{n-k+1} \left[\ l_i^{(k)} \, (2B)^{i-1} \ \sum_{j=1}^{j_{max}} (-1)^{j+1} \ d_{k+2j-3+1} \cdot s_j^{(k+i-1)} \right] \\
& j_{max} \equiv \mathrm{INT} \left(\frac{n-k+3-i}{2} \right) \\
& l_i^{(k=1)} = 1 \quad ; \quad l_i^{(k=2)} = l_{i-1}^{(k=2)} + l_i^{(k=1)} \quad ; \quad ... \quad ; \quad l_i^{(k=n)} = l_{i-1}^{(k=n)} + l_i^{(k=n-1)} \ ,
\end{aligned}$$

The numerical difficulties involved in the computation of the coefficients a_k causes the simple C-program to crash. My experience shows that the main (or only first ?) problem concerns with the evaluation of the overall coefficients $s_j^{(i)}$ and $l_i^{(k)}$, due to roundoff as well as machine precision. Expecting n to be of order 50, it is easy that such coefficients did **not** fit the `double` or `long double` structures allowed by C.

To be done: It would therefore be suitable to convert the C file (which I gave the misleading name `clenshaw_coef.c/h`, sorry) into a CLN one, and then adopt the (HH) well tested Laguerre method.

9.3 Mathematica

As referred at the end of Sect. (3), it seems to arise a difference between the polynomial in the Chebyshev-, eq. (7), and in the monomial basis, eq. (9).
There are many possible parts where this problem could arise, since chronologically

1. compute $N_{max} = 2000$ Chebyshev coefficients, d_j and construct P using n coefficients out of them, by means of Clenshaw recursion relation, see eq. (33);

2. giving to Mathematica the degree n and the approximation interval $[\epsilon, 1]$, let it compute exactly the n coefficients and then the roots z_k of the polynomial applied to real variable. Order the roots in Bit-Reversal mode;

3. write the roots r_k and the matching overall constant $C = (\prod_i c_i)^{1/n}$ as in eq. (10) into a file (relying on a finite accuracy, say values with 24 digits);

4. let these values be read by the C-code and perform the many multiplications involving the polynomial in non-trivial structure (2-spinors), as discussed in [4].

Since the discussed troubles did not seem to emerge within the Mathematica notebook, I start checking the values of the coefficients d_j as evaluated in Mathematica and the C-program, choosing $n = 48$ and $\epsilon = 0.0025$. The results are presented in the following table, where I denoted by C-exact the calculation of exactly $0 \to 48$ coefficients by the C-code, while Trunc stands for the calculation of the first $0 \to 48$ coefficients out of 2000.

k	Math	C-exact	Trunc C	Trunc Math
0	5.58200914260652	5.58200914260654	5.58206453293626	5.58206453293624
1	-3.04473013842613	-3.04473013842614	-3.04478583614984	-3.04478583614983
...	...	...	...	...
19	-0.16354383654914	-0.16354383654914	-0.16375361893866	-0.16375361893866
20	0.14450811986551	0.14450811986551	0.14474062315254	0.14474062315253
...	...	...	...	...
47	-0.00231417824140	-0.00231417824137	-0.00650832665357	-0.00650832665357
48	0.00114977193761	0.00114977193761	0.00582954388001	0.00582954388001

The evident almost equality between the Math and the C-exact coefficients reflects itself in a negligible difference between the two polynomial representations of P, as calculated by Mathematica on real variables [8].
On the contrary, it is interesting to look at the second table illustrating the differences and the variances between the different ways to determine the coefficients d_j. The 3^{th}

	Math vs. Trunc C		Variance: Math vs. C	
k	Difference	Variance	Exact	Trunc
0	0.00005539032974	0.00000992300950	$0.0 \cdot 10^{-14}$	$0.0 \cdot 10^{-14}$
1	0.00005569772371	0.0000182931561	$0.0 \cdot 10^{-14}$	$0.0 \cdot 10^{-14}$
...	...	...	...	...
19	0.00020978238952	0.00128272880191	$3.0 \cdot 10^{-14}$	$0.0 \cdot 10^{-14}$
20	0.00023250328703	0.00160892887709	$1.0 \cdot 10^{-14}$	$2.0 \cdot 10^{-14}$
...	...	...	...	...
47	0.00419414841217	1.81237051543798	$982 \cdot 10^{-14}$	$40 \cdot 10^{-14}$
48	0.00467977194240	4.07017408348795	$473 \cdot 10^{-14}$	$42 \cdot 10^{-14}$

and 4^{th} columns have been reported just to show the (same) order of magnitude between the Math and the C methods, both for exact as well as for truncated coefficients. Let me recall that the table shows the difference and variances between the coefficients. Even reminding that higher coefficients, say $d_{j>30}$ are expect to contribute much less than the first ones, say $d_{j<10}$, I strongly believe that the 1^{st} and 2^{nd} columns of the table reflects an existing problem.
However it is yet not proved how these sometimes large discrepancies effects the hole computations of P in the two representations [9].
The following two possible ways to overcome the problem turned out to fail:

- Read in Mathematica the coefficients as produced by the C-program (the Trunc C columns). **Fails** since Mathematica is expecting to handle a much higher accuracy.
- Produce with Mathematica itself n coefficients out of 2000. **Fails**, more correctly it runs **eternally**, probably for the same low-accuracy reasons.

To be done: Search for some more tests checking for the correctness of my hypothesis above. However, my wish would be to turn to CLN and have a (fast !) self-containing code.

Before going on, it should be mentioned that the Mathematica notebook evaluates the overall constant C as

$$\bar{C} = \frac{P(\bar{s})}{\prod_{j=1}^{n}(\bar{s} - z_j)} \quad , \quad P(\bar{s}) = \frac{1}{2} d_0 T_0(\bar{s}) + \sum_{j=1}^{n} d_j T_j(\bar{s}) \quad , \quad d_j \in \text{Math}$$
$$\Rightarrow \quad C = \bar{C}^{1/n} \, , \tag{34}$$

[8] I do not expect that applications on 2-spinors will change this behaviour.
[9] I think that in principle, discrepancies cancellations can happen during the multiplication, due to the alternating sign of the coefficients.

where the polynomial, P, is evaluated exactly on few chosen real point, $\bar{s}$, in the approximation interval. Believing these points, this overall constant turns out to remain constant in the approximation interval.
However this is **not** the definition of the constant we really need.

For our purposes, we should replace in eq. (34) the exact polynomial evaluation with the widely used Clenshaw recursion relations, together with the corresponding truncated d_j. In this way however, we enter the discussed problems concerning the discrepancy between the d_j's, see previous table.

Insisting on working with Mathematica, inserting Clenshaw recursions and truncated coefficients, it happens that the claimed overall constant **did not** remain constant within the approximation interval, as illustrated in the following table. The correspond-

	$n=8$			$n=18$			$n=48$		
		P vs. $BB^\dagger$			P vs. $BB^\dagger$			$(P$ vs. $BB^\dagger)\cdot 10^{-2}$	
$\bar{s}$	C_8	| Diff. |	Var. %	C_{18}	| Diff. |	Var. %	C_{48}	| Diff. |	Var. %
ϵ	2.9561	2.02350	16.18	3.1544	0.48712	3.32	3.33944350	0.6592	4.325
0.336	2.8234	0.29820	21.03	3.1434	0.04925	2.93	3.33940988	0.0088	0.508
0.5	2.9394	0.19799	12.30	3.1439	0.03669	2.66	3.33943271	0.0392	2.773
0.734	2.8154	0.21562	23.80	3.1509	0.01629	1.38	3.33939139	0.0369	3.165
1.0	2.9533	0.19084	15.56	3.1547	0.03685	3.53	3.33944545	0.0461	4.604
$\bar{s}=0.5$	$C=2.8916$			$C=3.1484$			$C=3.33941341$		

ing reference value as evaluated using eq. (34) for the 2 settings are reported in the last row.
It should be remarked that the last three columns ($n=48$ case) contains values calculated with less precision: in order to evaluate the Clenshaw recursions I have reduced the (truncated) Chebyshev coefficients, d_j, to a precision of 36 digits.
On the contrary, the values in the other columns refer to a toy models, with polynomial degree $n=8,18$ and have been evaluated using the full Mathematica precision.
Insisting on using this external software, the same analysis for an appropriate degree, say $n=48$, needs ages to run till the end. On one hand this is due to the computation of the $N_{max}=2000$ coefficients, from which the code takes only the first n; on the other hand, the Clenshaw loop discussed in eq. (33) seems to progressively slow down, until it stacks [10].

To be done: One should therefore take care of:
- check whether the discrepancy between the overall constants as evaluated in the two prescribed model persists also for larger degrees;
- check whether the last, correct, C remains really constant in the approximation interval, i.e: varying $\bar{s}$;
- since it enters n multiplication in the Molecular Dynamics, write the constant C with large precision to a file, `normierungLocal.dat`;
- similarly, write the roots r_j with large number of digits in a separate file

[10] The determination of $BB^\dagger$ as the determinant of a diagonal $n\times n$ matrix is not requiring a too long time.

`Square_root_BR_roots.dat`.

The root ordering has been proved to be correctly done by checking the oscillation (Min, Max, Ratio) during the product discussed in eq. (9)

$$\text{Min}^{(l)} = \min_{s\in[\epsilon,1]} |P^l(s)| \quad , \quad \text{Max}^{(l)} = \max_{s\in[\epsilon,1]} |P^l(s)| \quad , \quad \text{Ratio}^{(l)} = \frac{\text{Max}^{(l)}}{\text{Min}^{(l)}}$$

$$P^l(s) = P^l_{n,\epsilon}(s) = \prod_{j=1}^{l}(s - z_j) \qquad , \qquad l \in \{1, ..., n\}$$

The following table clearly shows the validity of the adopted Bit-Reversal Ordering, as discussed in [7]. The last row has been chosen as the case where the Ratio become maximal.

	Unordered			Bit Reversal for z_i			Bit Reversal for r_i		
l	Max	Min	Ratio	Max	Min	Ratio	Max	Min	Ratio
1	0.949	0.232	4.089	0.938	0.224	4.193	3.502	0.184	19.077
2	1.591	0.063	25.27	2.609	0.010	248.17	6.942	0.060	115.17
...	...	...	...	...	...	...	...	...	...
22	1416.3	0.0013	1092932.3	2.58	0.257	10.06	1.061	0.038	27.86
23	2421.5	0.0009	2630314.7	5.87	0.295	19.91	1.922	0.129	14.93
...	...	...	...	...	...	...	...	...	...
47	10.61	0.301	35.27	25.63	0.292	87.9	2.659	0.578	4.600
48	1.002	0.997	1.00	1.002	0.997	1.011	0.999	0.316	3.160
...	...	...	...	...	...	...	...	...	...
67							39.66	0.199	199.56
68							117.7	0.263	448.1
Worst	4157.3	0.0005	7977251.4	2.609	0.010	248.17	114.65	0.247	464.7

Even if everything seems to be correctly implemented, recall that it has been checked working with real values. Nevertheless, I hardly think that non-trivial structures as 2-spinors will change these outcomes.

To be done: Repeat a similar analysis for the application of S to 2-spinors Φ_h.

9.4 Hermiticity

Related to the previous two subsections, it is plausible to expect that also the problems concerning the scalar product, as at the end of Sect. (3), and the behaviour of the corrections to the final Hamiltonian, as discussed at the end of Sect. (5), will be solved once the roots computation is well tested.

To be done: It would be however interesting to monitor the behaviour of the final Hamiltonian corrections by choosing $\epsilon \neq \tilde{\epsilon}$. Probably more exhaustive tests on

the dependence on the various parameters can be performed.

9.5 Merging

This point should probably be implemented only once everything else is correctly running.
However it would be interesting to test whether the use of other inverter algorithm needed by the Jacoby-Davidson procedure can accelerate the eigenvalues evaluation. So far only the `bicgstab_complex_bi.c/h` has been modified in order to handle bispinor structures, when the serial code calls the routine `jdher_bi.c/h`. Analogously, the parallel code `pjdher_bi.c/h` can only take profit of `cg_her_bi.c/h`.
Beside the eigenvalues calculation, there is a lack on variety also of integration scheme: as already mentioned, I only implemented the `leap_frog_ND` as a routine in the file `hybrid_nondegenerate.c/h`.

10 Involved files

The files characterising the Phmc code are mainly derived from similar Hmc-files and usually carry similar names. To distinguish and/or to find the Phmc properties, a comment (`IF PHMC ... END PHMC`) has been inserted whenever new characterising lines have been added.

- `phmc_tm.c`
  The input parameters are read; the memory for the so-called bispinor structures, needed for the eigenvalues λ and Λ computation is allocated.
  Depending on the starting condition (hot, cold, continue, restart), the code computes new eigenvalues on a uniformly distributed random gauge configuration and write them into a file or it reads from a file. This is called `INOUT.data` and contains $\tilde{\lambda}, \tilde{\Lambda}$ and $\tilde{\epsilon}$; moreover, more unnecessary informations are stored in `EVS.data`.
  Once the intervals are determined, the code calls `chebyshev_polynomial_nd.c/h` evaluating the Chebyshev coefficients, d_k, and polynomial degree, n, for P; thus the same for the construction of $\tilde{P}$ calling `Ptilde_nd.c/h`.
  Known n, the code dynamically allocates the memory for the χ spinors to be used in the Molecular Dynamic (see [4]); **so far** (due to external Mathematica notebook) the $2n$ roots r_j and the overall coefficient C needed to match the two polynomial representations of P, see eq. (9) are read and not computed.
  Finally the measurement loop is started, calling the update routine `update_tm_nd.c/h`.
  Let me recall that the re-computation of the interval $[\epsilon, 1]$, of the polynomial P and its roots each *rec_ev* measurements is still commented.

- `update_tm_nd.c/h`
  After defining the pseudofermion 2-spinors as in eq. (11), the code calls for the routine evolving the configuration and evaluating the final Hamiltonian (say without corrections). Finally it performs a Metropolis Accept/Reject step.

Even here, the coupling of the contributions between the light (Hmc) and heavy (Phmc) sector is done by hand, by simply summing the energies.

- `hybrid_nondegenerate_update.c/h`
  This is the routine where the Phmc forces are evaluated and the new configuration candidate is determined. Again: in order to discard the Hmc contributions to the Phmc part has been done by hand.

Additionally, there are few more files for the implicitly mentioned routines:

- `init_bispinor_field.c/h`, `init_chi_spinor_field.c/h` and `init_chi_copy.c/h` allocate memory for bispinors and the up/down spinors, respectively;
- `start.c/h-NEW` containing the additional routine for the uniformly distributed random gauge configuration. Remember to copy these files into `start.c/h`;
- in the directory `solver` you can find few *filename*`_bi.c/h` needed for the eigenvalues computation using the bispinor structure;
- analogously, in the directory `linalg` you can find few *filename*`_bi.c/h` created initially for some test on the non-degenerate operator or used when computing the eigenvalues.

Finally, due to the still not finished code, there are similar copies of some files, like `phmc_tm.c/h-extension` or `update_tm_nd.c/h-extension` containing just some testing code. They work with the usual header file.

References

[1] T. Chiarappa, R. Frezzotti and C. Urbach, [`hep-lat/0509154`]
"A (P)HMC algorithm for N(f) = 2+1+1 flavours of twisted mass fermions".

[2] T. Chiarappa, *October 2005*
"Status of eigenvalues computation of a flavour non-degenerate Dirac operator".

[3] T. Chiarappa and R. Frezzotti, *November 2005*
"Polynomial approximations for 1+1 dynamical quarks".

[4] T. Chiarappa, *March 2006*
"Status of the Phmc-forces computation".

[5] Many, `Numerical Recipes`
"Chapter 5.8: Chebyshev Approximation".

[6] Many, `Numerical Recipes`
"Chapter 9.5: Roots of Polynomials".

[7] B. Bunk, S. Elser, R. Frezzotti and K. Jansen, [`hep-lat/9805026`]
"Ordering monomial factors of polynomials in the product representation".

[8] HH-group and al., [hep-lat/0606011]
"Numerical simulation of QCD with u, d, s and c quarks in the twisted-mass Wilson formulation".