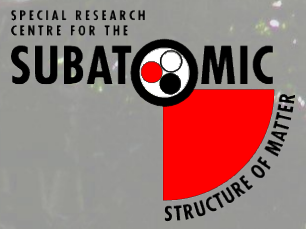
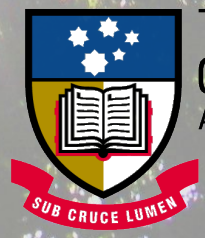
Polynomial Filtering for a Single Fermion Flavour in Lattice QCD

Waseem Kamleh and Michael J. Peardon

CSSM & University of Adelaide

T(r)opical QCD II,
September 26th – October 1st, 2010





THE UNIVERSITY
OF ADELAIDE
AUSTRALIA

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Introduction

- The dominant expense of Lattice QCD is generating dynamical gauge field configurations.
- Hybrid Monte Carlo (HMC) is still the most used algorithm for generating dynamical configurations.
- There have been many improvements to the basic HMC algorithm developed over the years.
- A partial list of "multi-scale" type algorithmic improvements:
  - Domain Decomposition method (Luscher).
  - Mass Preconditioning (Hasenbuch).
  - Polynomial Filtering (Peardon & Sexton, Peardon & WK).
- We begin by reviewing the fundamentals of the HMC algorithm.

Hybrid Monte Carlo Review

- The lattice is embedded in a Hamiltonian system by the addition of a fictitious "simulation" time τ, along with an additional fictitious field P which are the conjugate momenta to U,

$$\mathcal{H}(P,U) = \sum_{x,\mu} \frac{1}{2} \operatorname{Tr} P_\mu(x)^2 + S[U].$$

- The conjugate momenta $P_\mu(x)$ are drawn from a Gaussian distribution.
- In this way, the Hamiltonian $\mathcal{H}$ is constructed so that after path integration the expectation values of observables are unaltered.
- Where are the fermions?

Hybrid Monte Carlo Review

- The lattice is embedded in a Hamiltonian system by the addition of a fictitious "simulation" time τ, along with an additional fictitious field P which are the conjugate momenta to U,

$$\mathcal{H}(P,U) = \sum_{x,\mu} \frac{1}{2} \operatorname{Tr} P_\mu(x)^2 + S[U].$$

- The conjugate momenta $P_\mu(x)$ are drawn from a Gaussian distribution.
- In this way, the Hamiltonian $\mathcal{H}$ is constructed so that after path integration the expectation values of observables are unaltered.
- Where are the fermions?

Hybrid Monte Carlo Review

- The lattice is embedded in a Hamiltonian system by the addition of a fictitious "simulation" time τ, along with an additional fictitious field P which are the conjugate momenta to U,

$$\mathcal{H}(P,U) = \sum_{x,\mu} \frac{1}{2} \operatorname{Tr} P_\mu(x)^2 + S[U].$$

- The conjugate momenta $P_\mu(x)$ are drawn from a Gaussian distribution.
- In this way, the Hamiltonian $\mathcal{H}$ is constructed so that after path integration the expectation values of observables are unaltered.
- Where are the fermions?

Hybrid Monte Carlo Review

- The lattice is embedded in a Hamiltonian system by the addition of a fictitious "simulation" time τ, along with an additional fictitious field P which are the conjugate momenta to U,

$$\mathcal{H}(P,U) = \sum_{x,\mu} \frac{1}{2} \operatorname{Tr} P_\mu(x)^2 + S[U].$$

- The conjugate momenta $P_\mu(x)$ are drawn from a Gaussian distribution.
- In this way, the Hamiltonian $\mathcal{H}$ is constructed so that after path integration the expectation values of observables are unaltered.
- Where are the fermions?

Fermion Determinant

- The fermion fields ψ and $\bar{\psi}$ are Grassmannian, hence to perform simulations we need to integrate them out. We have that

$$\det D_{\mathrm{w}} = \int \mathcal{D}\bar{\psi}\mathcal{D}\psi \, e^{-\int d^4x \bar{\psi}(x) D_{\mathrm{w}} \psi(x)}.$$

- Define the effective action as

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] - \ln \det D_{\mathrm{w}}[U].$$

- We can write the action for full QCD in terms of bosonic fields ϕ using the identity

$$\det M = \frac{1}{\det M^{-1}} = \int \mathcal{D}\phi^\dagger \mathcal{D}\phi \, e^{-\int d^4x \, \phi^\dagger(x) M^{-1} \phi(x)}.$$

Fermion Determinant

- The fermion fields ψ and $\bar{\psi}$ are Grassmannian, hence to perform simulations we need to integrate them out. We have that

$$\det D_{\mathrm{w}} = \int \mathcal{D}\bar{\psi}\mathcal{D}\psi \, e^{-\int d^4x \bar{\psi}(x) D_{\mathrm{w}} \psi(x)}.$$

- Define the effective action as

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] - \ln \det D_{\mathrm{w}}[U].$$

- We can write the action for full QCD in terms of bosonic fields ϕ using the identity

$$\det M = \frac{1}{\det M^{-1}} = \int \mathcal{D}\phi^\dagger \mathcal{D}\phi \, e^{-\int d^4x \, \phi^\dagger(x) M^{-1} \phi(x)}.$$

Fermion Determinant

- The fermion fields ψ and $\bar{\psi}$ are Grassmannian, hence to perform simulations we need to integrate them out. We have that

$$\det D_{\mathrm{w}} = \int \mathcal{D}\bar{\psi}\mathcal{D}\psi \, e^{-\int d^4x \bar{\psi}(x) D_{\mathrm{w}} \psi(x)}.$$

- Define the effective action as

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] - \ln \det D_{\mathrm{w}}[U].$$

- We can write the action for full QCD in terms of bosonic fields ϕ using the identity

$$\det M = \frac{1}{\det M^{-1}} = \int \mathcal{D}\phi^\dagger \mathcal{D}\phi \, e^{-\int d^4x \, \phi^\dagger(x) M^{-1} \phi(x)}.$$

Pseudofermions

- The convergence of the Gaussian integral in ϕ is only guaranteed for Hermitian positive definite (Hpd) M.
- For Wilson-type fermions, D_{w} is a complex matrix, but $\det D_{\mathrm{w}}$ is real and positive.
- So, we can define $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$, and M will be Hpd with $\det M = \det D_{\mathrm{f}}^2$.
- Then the *pseudofermionic* effective action for full QCD with two flavours of degenerate quarks is

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] + \int d^4x \, \phi^{\dagger}(x) M^{-1}[U] \phi(x).$$

- The pseudofermion fields $\phi(x)$ are drawn from a Gaussian distribution.

Pseudofermions

- The convergence of the Gaussian integral in ϕ is only guaranteed for Hermitian positive definite (Hpd) M.
- For Wilson-type fermions, D_{w} is a complex matrix, but $\det D_{\mathrm{w}}$ is real and positive.
- So, we can define $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$, and M will be Hpd with $\det M = \det D_{\mathrm{f}}^2$.
- Then the *pseudofermionic* effective action for full QCD with two flavours of degenerate quarks is

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] + \int d^4x \, \phi^{\dagger}(x) M^{-1}[U] \phi(x).$$

- The pseudofermion fields $\phi(x)$ are drawn from a Gaussian distribution.

Pseudofermions

- The convergence of the Gaussian integral in ϕ is only guaranteed for Hermitian positive definite (Hpd) M.
- For Wilson-type fermions, D_{w} is a complex matrix, but $\det D_{\mathrm{w}}$ is real and positive.
- So, we can define $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$, and M will be Hpd with $\det M = \det D_{\mathrm{f}}^2$.
- Then the *pseudofermionic* effective action for full QCD with two flavours of degenerate quarks is

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] + \int d^4x \, \phi^{\dagger}(x) M^{-1}[U] \phi(x).$$

- The pseudofermion fields $\phi(x)$ are drawn from a Gaussian distribution.

Pseudofermions

- The convergence of the Gaussian integral in ϕ is only guaranteed for Hermitian positive definite (Hpd) M.
- For Wilson-type fermions, D_{w} is a complex matrix, but $\det D_{\mathrm{w}}$ is real and positive.
- So, we can define $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$, and M will be Hpd with $\det M = \det D_{\mathrm{f}}^2$.
- Then the *pseudofermionic* effective action for full QCD with two flavours of degenerate quarks is

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] + \int d^4x \, \phi^{\dagger}(x) M^{-1}[U] \phi(x).$$

- The pseudofermion fields $\phi(x)$ are drawn from a Gaussian distribution.

Pseudofermions

- The convergence of the Gaussian integral in ϕ is only guaranteed for Hermitian positive definite (Hpd) M.
- For Wilson-type fermions, D_{w} is a complex matrix, but $\det D_{\mathrm{w}}$ is real and positive.
- So, we can define $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$, and M will be Hpd with $\det M = \det D_{\mathrm{f}}^2$.
- Then the *pseudofermionic* effective action for full QCD with two flavours of degenerate quarks is

$$S_{\mathrm{eff}}[U] = S_{\mathrm{g}}[U] + \int d^4x \, \phi^{\dagger}(x) M^{-1}[U] \phi(x).$$

- The pseudofermion fields $\phi(x)$ are drawn from a Gaussian distribution.

Hybrid Monte Carlo

- The Hybrid Monte Carlo algorithm creates a Markov chain by alternately performing two steps:
  - A Molecular Dynamics (MD) integration to generate a new configuration $(U \to U', P \to P')$.
  - A Metropolis accept/reject step on the proposed configuration (U', P').
- The accept/reject step is based upon the change in the Hamiltonian

$$\rho(U \to U', P \to P') \propto e^{-\Delta\mathcal{H}}.$$

- The Molecular Dynamics integration takes place along a trajectory which for sufficiently small integration step sizes $\Delta\tau$ approximately conserves $\mathcal{H}$, hence yielding high acceptance rates.

Hybrid Monte Carlo

- The Hybrid Monte Carlo algorithm creates a Markov chain by alternately performing two steps:
  - A Molecular Dynamics (MD) integration to generate a new configuration $(U \to U', P \to P')$.
  - A Metropolis accept/reject step on the proposed configuration (U', P').
- The accept/reject step is based upon the change in the Hamiltonian

$$\rho(U \to U', P \to P') \propto e^{-\Delta\mathcal{H}}.$$

- The Molecular Dynamics integration takes place along a trajectory which for sufficiently small integration step sizes $\Delta\tau$ approximately conserves $\mathcal{H}$, hence yielding high acceptance rates.

Hybrid Monte Carlo

- The Hybrid Monte Carlo algorithm creates a Markov chain by alternately performing two steps:
  - A Molecular Dynamics (MD) integration to generate a new configuration $(U \to U', P \to P')$.
  - A Metropolis accept/reject step on the proposed configuration (U', P').
- The accept/reject step is based upon the change in the Hamiltonian

$$\rho(U \to U', P \to P') \propto e^{-\Delta \mathcal{H}}.$$

- The Molecular Dynamics integration takes place along a trajectory which for sufficiently small integration step sizes $\Delta\tau$ approximately conserves $\mathcal{H}$, hence yielding high acceptance rates.

Hybrid Monte Carlo

- The Hybrid Monte Carlo algorithm creates a Markov chain by alternately performing two steps:
  - A Molecular Dynamics (MD) integration to generate a new configuration $(U \to U', P \to P')$.
  - A Metropolis accept/reject step on the proposed configuration (U', P').
- The accept/reject step is based upon the change in the Hamiltonian

$$\rho(U \to U', P \to P') \propto e^{-\Delta \mathcal{H}}.$$

- The Molecular Dynamics integration takes place along a trajectory which for sufficiently small integration step sizes $\Delta\tau$ approximately conserves $\mathcal{H}$, hence yielding high acceptance rates.

Hybrid Monte Carlo

- The Hybrid Monte Carlo algorithm creates a Markov chain by alternately performing two steps:
  - A Molecular Dynamics (MD) integration to generate a new configuration $(U \to U', P \to P')$.
  - A Metropolis accept/reject step on the proposed configuration (U', P').
- The accept/reject step is based upon the change in the Hamiltonian

$$\rho(U \to U', P \to P') \propto e^{-\Delta \mathcal{H}}.$$

- The Molecular Dynamics integration takes place along a trajectory which for sufficiently small integration step sizes $\Delta\tau$ approximately conserves $\mathcal{H}$, hence yielding high acceptance rates.

Molecular Dynamics

- Start by requiring that the Hamiltonian be conserved along the trajectory

$$\frac{d\mathcal{H}}{d\tau} = 0.$$

- Then derive the discretised equations of motion,

$$U_\mu(x, \tau + \Delta\tau) = U_\mu(x, \tau) \exp\left(i\Delta\tau P_\mu(x, \tau)\right),$$

$$P_\mu(x, \tau + \Delta\tau) = P_\mu(x, \tau) - U_\mu(x, \tau) \frac{\delta S}{\delta U_\mu(x, \tau)}.$$

- The derivative of the action with respect to the gauge fields is known as the *force term*,

$$F_\mu(x) = \frac{\delta S}{\delta U_\mu(x)}.$$

Molecular Dynamics

- Start by requiring that the Hamiltonian be conserved along the trajectory

$$\frac{d\mathcal{H}}{d\tau} = 0.$$

- Then derive the discretised equations of motion,

$$U_\mu(x, \tau + \Delta\tau) = U_\mu(x, \tau) \exp\left(i\Delta\tau P_\mu(x, \tau)\right),$$

$$P_\mu(x, \tau + \Delta\tau) = P_\mu(x, \tau) - U_\mu(x, \tau) \frac{\delta S}{\delta U_\mu(x, \tau)}.$$

- The derivative of the action with respect to the gauge fields is known as the *force term*,

$$F_\mu(x) = \frac{\delta S}{\delta U_\mu(x)}.$$

Molecular Dynamics

- Start by requiring that the Hamiltonian be conserved along the trajectory

$$\frac{d\mathcal{H}}{d\tau} = 0.$$

- Then derive the discretised equations of motion,

$$U_\mu(x, \tau + \Delta\tau) = U_\mu(x, \tau) \exp\left(i\Delta\tau P_\mu(x, \tau)\right),$$

$$P_\mu(x, \tau + \Delta\tau) = P_\mu(x, \tau) - U_\mu(x, \tau) \frac{\delta S}{\delta U_\mu(x, \tau)}.$$

- The derivative of the action with respect to the gauge fields is known as the *force term*,

$$F_\mu(x) = \frac{\delta S}{\delta U_\mu(x)}.$$

Splitting The Action

- Lets split our action into its gauge field and pseudofermion field components, $S = S_{\mathrm{g}} + S_{\mathrm{pf}}$, where

$$S_{\mathrm{g}} = \beta \sum_{x,\mu<\nu} \frac{1}{3} \operatorname{Re} \operatorname{Tr}(1 - U_{\mu\nu}(x)),$$

$$S_{\mathrm{pf}} = \sum_{x} \phi^{\dagger}(x) (D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}})^{-1} \phi(x),$$

  and D_{w} is the Wilson fermion matrix.
- Each of the terms in the action induces a force term.
- The size of the force term is the dominant factor in determining what step size $\Delta\tau$ is need for a given acceptance probability ρ_{acc}.
- As the quark mass becomes lighter, the size of the pseudofermion force term increases.

Splitting The Action

- Lets split our action into its gauge field and pseudofermion field components, $S = S_{\mathrm{g}} + S_{\mathrm{pf}}$, where

$$S_{\mathrm{g}} = \beta \sum_{x,\mu<\nu} \frac{1}{3} \operatorname{Re} \operatorname{Tr}(1 - U_{\mu\nu}(x)),$$

$$S_{\mathrm{pf}} = \sum_{x} \phi^{\dagger}(x)(D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}})^{-1} \phi(x),$$

  and D_{w} is the Wilson fermion matrix.
- Each of the terms in the action induces a force term.
- The size of the force term is the dominant factor in determining what step size $\Delta\tau$ is need for a given acceptance probability ρ_{acc}.
- As the quark mass becomes lighter, the size of the pseudofermion force term increases.

Splitting The Action

- Lets split our action into its gauge field and pseudofermion field components, $S = S_{\mathrm{g}} + S_{\mathrm{pf}}$, where

$$S_{\mathrm{g}} = \beta \sum_{x,\mu<\nu} \frac{1}{3} \operatorname{Re} \operatorname{Tr}(1 - U_{\mu\nu}(x)),$$

$$S_{\mathrm{pf}} = \sum_{x} \phi^{\dagger}(x)(D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}})^{-1} \phi(x),$$

  and D_{w} is the Wilson fermion matrix.
- Each of the terms in the action induces a force term.
- The size of the force term is the dominant factor in determining what step size $\Delta\tau$ is need for a given acceptance probability ρ_{acc}.
- As the quark mass becomes lighter, the size of the pseudofermion force term increases.

Splitting The Action

- Lets split our action into its gauge field and pseudofermion field components, $S = S_{\mathrm{g}} + S_{\mathrm{pf}}$, where

$$S_{\mathrm{g}} = \beta \sum_{x,\mu<\nu} \frac{1}{3} \operatorname{Re} \operatorname{Tr}(1 - U_{\mu\nu}(x)),$$

$$S_{\mathrm{pf}} = \sum_{x} \phi^{\dagger}(x) (D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}})^{-1} \phi(x),$$

  and D_{w} is the Wilson fermion matrix.
- Each of the terms in the action induces a force term.
- The size of the force term is the dominant factor in determining what step size $\Delta\tau$ is need for a given acceptance probability ρ_{acc}.
- As the quark mass becomes lighter, the size of the pseudofermion force term increases.

Leapfrog Integration

- The MD equations of motion induce corresponding time evolution operators,

$$V_T(\Delta\tau) : \{U(\tau), P(\tau)\} \rightarrow \{U(\tau + \Delta\tau), P(\tau)\},$$

$$V_S(\Delta\tau) : \{U(\tau), P(\tau)\} \rightarrow \{U(\tau), P(\tau + \Delta\tau)\}.$$

- The simplest MD integration scheme is the leapfrog

$$V(\Delta\tau) = V_S(\frac{\Delta\tau}{2}) V_T(\Delta\tau) V_S(\frac{\Delta\tau}{2}).$$

- MD Integration trajectories typically have unit length, and hence as the step size $\Delta\tau$ decreases, the number of integration steps increases.

Leapfrog Integration

- The MD equations of motion induce corresponding time evolution operators,

$$V_T(\Delta\tau) : \{U(\tau), P(\tau)\} \rightarrow \{U(\tau + \Delta\tau), P(\tau)\},$$

$$V_S(\Delta\tau) : \{U(\tau), P(\tau)\} \rightarrow \{U(\tau), P(\tau + \Delta\tau)\}.$$

- The simplest MD integration scheme is the leapfrog

$$V(\Delta\tau) = V_S(\frac{\Delta\tau}{2}) V_T(\Delta\tau) V_S(\frac{\Delta\tau}{2}).$$

- MD Integration trajectories typically have unit length, and hence as the step size $\Delta\tau$ decreases, the number of integration steps increases.

Leapfrog Integration

- The MD equations of motion induce corresponding time evolution operators,

$$V_T(\Delta\tau) : \{U(\tau), P(\tau)\} \rightarrow \{U(\tau + \Delta\tau), P(\tau)\},$$

$$V_S(\Delta\tau) : \{U(\tau), P(\tau)\} \rightarrow \{U(\tau), P(\tau + \Delta\tau)\}.$$

- The simplest MD integration scheme is the leapfrog

$$V(\Delta\tau) = V_S(\frac{\Delta\tau}{2}) V_T(\Delta\tau) V_S(\frac{\Delta\tau}{2}).$$

- MD Integration trajectories typically have unit length, and hence as the step size $\Delta\tau$ decreases, the number of integration steps increases.

Pseudofermion Force

- Each time we act with $V_S(\Delta\tau)$ we need to evaluate the pseudofermion force term,

$$F_{\rm pf} = \frac{\delta S_{\rm pf}}{\delta U}.$$

- This involves inverting the fermion matrix, and hence is expensive!
- However, for split actions $S = S_1 + S_2$ we can use a multiple time scale integration scheme *(nested leapfrog)*,

$$V(\Delta\tau) = V_2(\frac{\Delta\tau}{2}) \left[V_1(\frac{\Delta\tau}{m}) \right]^m V_2(\frac{\Delta\tau}{2}),$$

$$V_1(\Delta\tau) = V_{S_1}(\frac{\Delta\tau}{2}) V_T(\Delta\tau) V_{S_1}(\frac{\Delta\tau}{2}), \quad V_2 = V_{S_2}(\Delta\tau).$$

Pseudofermion Force

- Each time we act with $V_S(\Delta\tau)$ we need to evaluate the pseudofermion force term,

$$F_{\rm pf} = \frac{\delta S_{\rm pf}}{\delta U}.$$

- This involves inverting the fermion matrix, and hence is expensive!
- However, for split actions $S = S_1 + S_2$ we can use a multiple time scale integration scheme *(nested leapfrog)*,

$$V(\Delta\tau) = V_2(\frac{\Delta\tau}{2}) \left[V_1(\frac{\Delta\tau}{m}) \right]^m V_2(\frac{\Delta\tau}{2}),$$

$$V_1(\Delta\tau) = V_{S_1}(\frac{\Delta\tau}{2}) V_T(\Delta\tau) V_{S_1}(\frac{\Delta\tau}{2}), \quad V_2 = V_{S_2}(\Delta\tau).$$

Pseudofermion Force

- Each time we act with $V_S(\Delta\tau)$ we need to evaluate the pseudofermion force term,

$$F_{\rm pf} = \frac{\delta S_{\rm pf}}{\delta U}.$$

- This involves inverting the fermion matrix, and hence is expensive!
- However, for split actions $S = S_1 + S_2$ we can use a multiple time scale integration scheme *(nested leapfrog)*,

$$V(\Delta\tau) = V_2(\frac{\Delta\tau}{2}) \left[V_1(\frac{\Delta\tau}{m}) \right]^m V_2(\frac{\Delta\tau}{2}),$$

$$V_1(\Delta\tau) = V_{S_1}(\frac{\Delta\tau}{2}) V_T(\Delta\tau) V_{S_1}(\frac{\Delta\tau}{2}), \quad V_2 = V_{S_2}(\Delta\tau).$$

Multiple Time Scales

- Multiple time scale integration is effective when the force term F_1 due to S_1 is cheap to evaluate compared to F_2 (that of S_2).
- However, as the step-size for S_2 is larger, we also require that the size of the force term for S_2 is relatively small compared to that of S_1.
- The gauge force F_g is cheap compared to the pseudofermion force F_{pf}, and at heavy quark masses $F_g > F_{pf}$, but at light quark masses the UV fluctuations in the pseudo fermion force become too large for multiple time scales to be effective.
- This is where polynomial filtering steps in.

Multiple Time Scales

- Multiple time scale integration is effective when the force term F_1 due to S_1 is cheap to evaluate compared to F_2 (that of S_2).
- However, as the step-size for S_2 is larger, we also require that the size of the force term for S_2 is relatively small compared to that of S_1.
- The gauge force F_g is cheap compared to the pseudofermion force F_{pf}, and at heavy quark masses $F_g > F_{pf}$, but at light quark masses the UV fluctuations in the pseudo fermion force become too large for multiple time scales to be effective.
- This is where polynomial filtering steps in.

Multiple Time Scales

- Multiple time scale integration is effective when the force term F_1 due to S_1 is cheap to evaluate compared to F_2 (that of S_2).
- However, as the step-size for S_2 is larger, we also require that the size of the force term for S_2 is relatively small compared to that of S_1.
- The gauge force F_g is cheap compared to the pseudofermion force F_{pf}, and at heavy quark masses $F_g > F_{pf}$, but at light quark masses the UV fluctuations in the pseudo fermion force become too large for multiple time scales to be effective.
- This is where polynomial filtering steps in.

Multiple Time Scales

- Multiple time scale integration is effective when the force term F_1 due to S_1 is cheap to evaluate compared to F_2 (that of S_2).
- However, as the step-size for S_2 is larger, we also require that the size of the force term for S_2 is relatively small compared to that of S_1.
- The gauge force F_{g} is cheap compared to the pseudofermion force F_{pf}, and at heavy quark masses $F_{\mathrm{g}} > F_{\mathrm{pf}}$, but at light quark masses the UV fluctuations in the pseudo fermion force become too large for multiple time scales to be effective.
- This is where polynomial filtering steps in.

Polynomial Filtering

- We can use a polynomial filter $\mathcal{P} = \mathcal{P}(M)$ to separate the ultraviolet and infrared physics in the pseudofermion force,

$$S_{\text{poly}} = \chi^\dagger \mathcal{P} \chi,$$

$$S_{\text{pf}} = \phi^\dagger (M\mathcal{P})^{-1} \phi.$$

- Recall $M = D_{\text{w}}^\dagger D_{\text{w}}$ is Hermitian positive definite.
- As S_{poly} is fast to evaluate compared to S_{pf} we split the action in the following way,

$$S_1 = S_{\text{g}} + S_{\text{poly}}, \quad S_2 = S_{\text{pf}}.$$

- If $\mathcal{P} \approx 1/z$ then it will capture the short-distance physics in S_{poly} and act as a UV filter in S_{pf}.

Polynomial Filtering

- We can use a polynomial filter $\mathcal{P} = \mathcal{P}(M)$ to separate the ultraviolet and infrared physics in the pseudofermion force,

$$S_{\text{poly}} = \chi^\dagger \mathcal{P} \chi,$$

$$S_{\text{pf}} = \phi^\dagger (M\mathcal{P})^{-1} \phi.$$

- Recall $M = D_{\text{w}}^\dagger D_{\text{w}}$ is Hermitian positive definite.
- As S_{poly} is fast to evaluate compared to S_{pf} we split the action in the following way,

$$S_1 = S_{\text{g}} + S_{\text{poly}}, \quad S_2 = S_{\text{pf}}.$$

- If $\mathcal{P} \approx 1/z$ then it will capture the short-distance physics in S_{poly} and act as a UV filter in S_{pf}.

Polynomial Filtering

- We can use a polynomial filter $\mathcal{P} = \mathcal{P}(M)$ to separate the ultraviolet and infrared physics in the pseudofermion force,

$$S_{\text{poly}} = \chi^\dagger \mathcal{P} \chi,$$

$$S_{\text{pf}} = \phi^\dagger (M\mathcal{P})^{-1} \phi.$$

- Recall $M = D_{\text{w}}^\dagger D_{\text{w}}$ is Hermitian positive definite.
- As S_{poly} is fast to evaluate compared to S_{pf} we split the action in the following way,

$$S_1 = S_{\text{g}} + S_{\text{poly}}, \quad S_2 = S_{\text{pf}}.$$

- If $\mathcal{P} \approx 1/z$ then it will capture the short-distance physics in S_{poly} and act as a UV filter in S_{pf}.

Polynomial Filtering

- We can use a polynomial filter $\mathcal{P} = \mathcal{P}(M)$ to separate the ultraviolet and infrared physics in the pseudofermion force,

$$S_{\text{poly}} = \chi^\dagger \mathcal{P} \chi,$$

$$S_{\text{pf}} = \phi^\dagger (M\mathcal{P})^{-1} \phi.$$

- Recall $M = D_{\text{w}}^\dagger D_{\text{w}}$ is Hermitian positive definite.
- As S_{poly} is fast to evaluate compared to S_{pf} we split the action in the following way,

$$S_1 = S_{\text{g}} + S_{\text{poly}}, \quad S_2 = S_{\text{pf}}.$$

- If $\mathcal{P} \approx 1/z$ then it will capture the short-distance physics in S_{poly} and act as a UV filter in S_{pf}.

Fermionic Determinant

- Note that the fermionic determinant is unchanged by the introduction of the polynomial filter.
- To see this we note that

$$\int \mathcal{D}\phi^\dagger \mathcal{D}\phi \mathcal{D}\chi^\dagger \mathcal{D}\chi \, e^{-\int d^4x \, \chi^\dagger \mathcal{P}\chi + \phi^\dagger(x)(M\mathcal{P})^{-1}\phi(x)}$$

$$= \frac{\det \mathcal{P}}{\det(M\mathcal{P})} = \frac{1}{\det M^{-1}} = \det M$$

- In the limit as the order of the polynomial $n \to \infty$ we have $M\mathcal{P} \to 1$ and $\mathcal{P}(M) \to M^{-1}$.
- In this way the polynomial term can reproduce as much or as little of the pseudofermion term as we want.

Fermionic Determinant

- Note that the fermionic determinant is unchanged by the introduction of the polynomial filter.
- To see this we note that

$$\int \mathcal{D}\phi^\dagger \mathcal{D}\phi \mathcal{D}\chi^\dagger \mathcal{D}\chi \, e^{-\int d^4x \, \chi^\dagger \mathcal{P}\chi + \phi^\dagger(x)(M\mathcal{P})^{-1}\phi(x)}$$

$$= \frac{\det \mathcal{P}}{\det(M\mathcal{P})} = \frac{1}{\det M^{-1}} = \det M$$

- In the limit as the order of the polynomial $n \to \infty$ we have $M\mathcal{P} \to 1$ and $\mathcal{P}(M) \to M^{-1}$.
- In this way the polynomial term can reproduce as much or as little of the pseudofermion term as we want.

Fermionic Determinant

- Note that the fermionic determinant is unchanged by the introduction of the polynomial filter.
- To see this we note that

$$\int \mathcal{D}\phi^\dagger \mathcal{D}\phi \mathcal{D}\chi^\dagger \mathcal{D}\chi \, e^{-\int d^4x \, \chi^\dagger \mathcal{P}\chi + \phi^\dagger(x)(M\mathcal{P})^{-1}\phi(x)}$$

$$= \frac{\det \mathcal{P}}{\det(M\mathcal{P})} = \frac{1}{\det M^{-1}} = \det M$$

- In the limit as the order of the polynomial $n \to \infty$ we have $M\mathcal{P} \to 1$ and $\mathcal{P}(M) \to M^{-1}$.
- In this way the polynomial term can reproduce as much or as little of the pseudofermion term as we want.

Fermionic Determinant

- Note that the fermionic determinant is unchanged by the introduction of the polynomial filter.
- To see this we note that

$$\int \mathcal{D}\phi^\dagger \mathcal{D}\phi \mathcal{D}\chi^\dagger \mathcal{D}\chi \, e^{-\int d^4x \, \chi^\dagger \mathcal{P}\chi + \phi^\dagger(x)(M\mathcal{P})^{-1}\phi(x)}$$

$$= \frac{\det \mathcal{P}}{\det(M\mathcal{P})} = \frac{1}{\det M^{-1}} = \det M$$

- In the limit as the order of the polynomial $n \to \infty$ we have $M\mathcal{P} \to 1$ and $\mathcal{P}(M) \to M^{-1}$.
- In this way the polynomial term can reproduce as much or as little of the pseudofermion term as we want.

Chebyshev Filter

- An effective UV filter is the n^{th} order Hermitian Chebyshev polynomial approximation to $1/z$,

$$\mathcal{P}_n(z) = a_n \prod_{k=1}^{n} (z - z_k) \approx \frac{1}{z},$$

  where we set $\theta_k = \frac{2\pi k}{n+1}$ to obtain the roots

$$z_k = \lambda[\frac{1}{2}(1+\epsilon)(1-\cos\theta_k) - i\sqrt{\epsilon}\sin\theta_k].$$

- The normalisation is defined by $z_0 = \frac{1}{2}(1+\epsilon)$, with

$$a_n = \frac{1}{z_0 \prod_{k=1}^{n}(z_0 - z_k)}.$$

- The approximation is good between $[\epsilon, 1]$, so we rescale with

$$\lambda = 1 + 8\kappa.$$

Chebyshev Filter

- An effective UV filter is the n^{th} order Hermitian Chebyshev polynomial approximation to $1/z$,

$$\mathcal{P}_n(z) = a_n \prod_{k=1}^{n} (z - z_k) \approx \frac{1}{z},$$

  where we set $\theta_k = \frac{2\pi k}{n+1}$ to obtain the roots

$$z_k = \lambda[\frac{1}{2}(1+\epsilon)(1-\cos\theta_k) - i\sqrt{\epsilon}\sin\theta_k].$$

- The normalisation is defined by $z_0 = \frac{1}{2}(1+\epsilon)$, with

$$a_n = \frac{1}{z_0 \prod_{k=1}^{n}(z_0 - z_k)}.$$

- The approximation is good between $[\epsilon, 1]$, so we rescale with

$$\lambda = 1 + 8\kappa.$$

Chebyshev Filter

- An effective UV filter is the n^{th} order Hermitian Chebyshev polynomial approximation to $1/z$,

$$\mathcal{P}_n(z) = a_n \prod_{k=1}^{n} (z - z_k) \approx \frac{1}{z},$$

  where we set $\theta_k = \frac{2\pi k}{n+1}$ to obtain the roots

$$z_k = \lambda[\frac{1}{2}(1+\epsilon)(1-\cos\theta_k) - i\sqrt{\epsilon}\sin\theta_k].$$

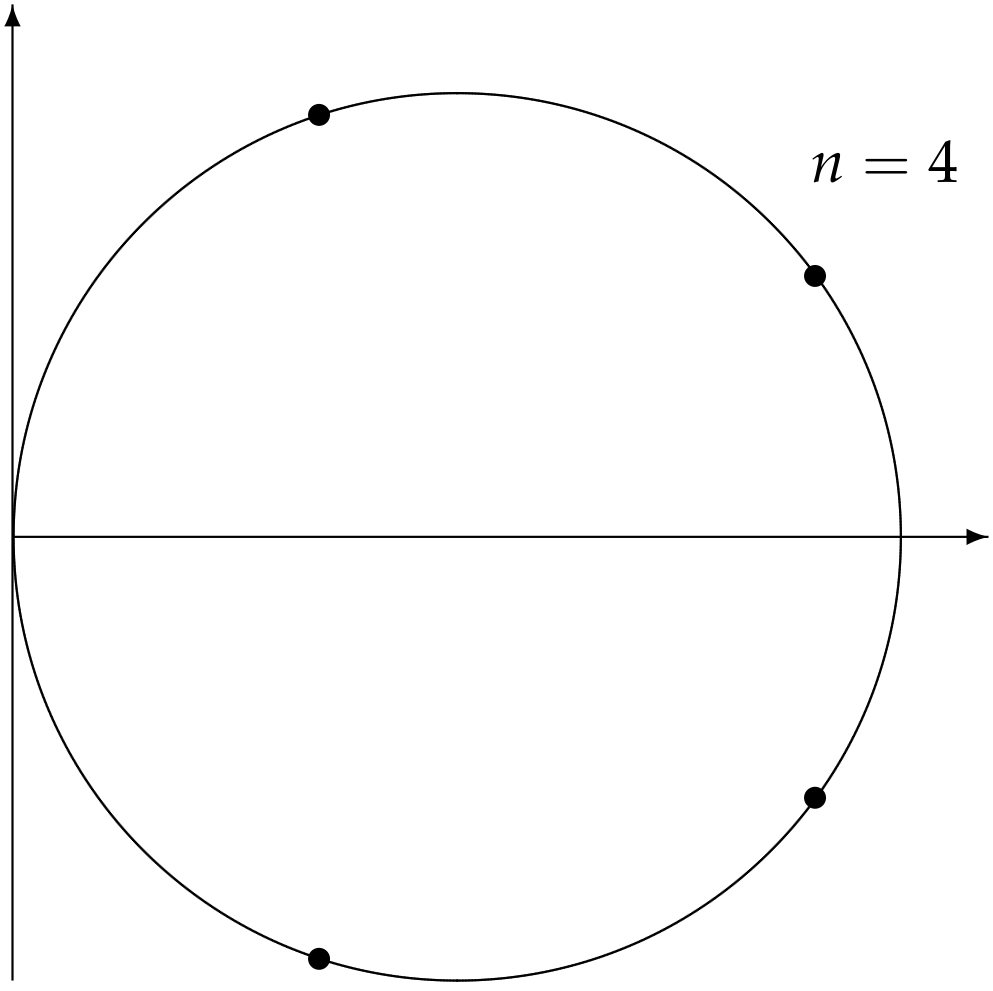
- The normalisation is defined by $z_0 = \frac{1}{2}(1+\epsilon)$, with

$$a_n = \frac{1}{z_0 \prod_{k=1}^{n}(z_0 - z_k)}.$$

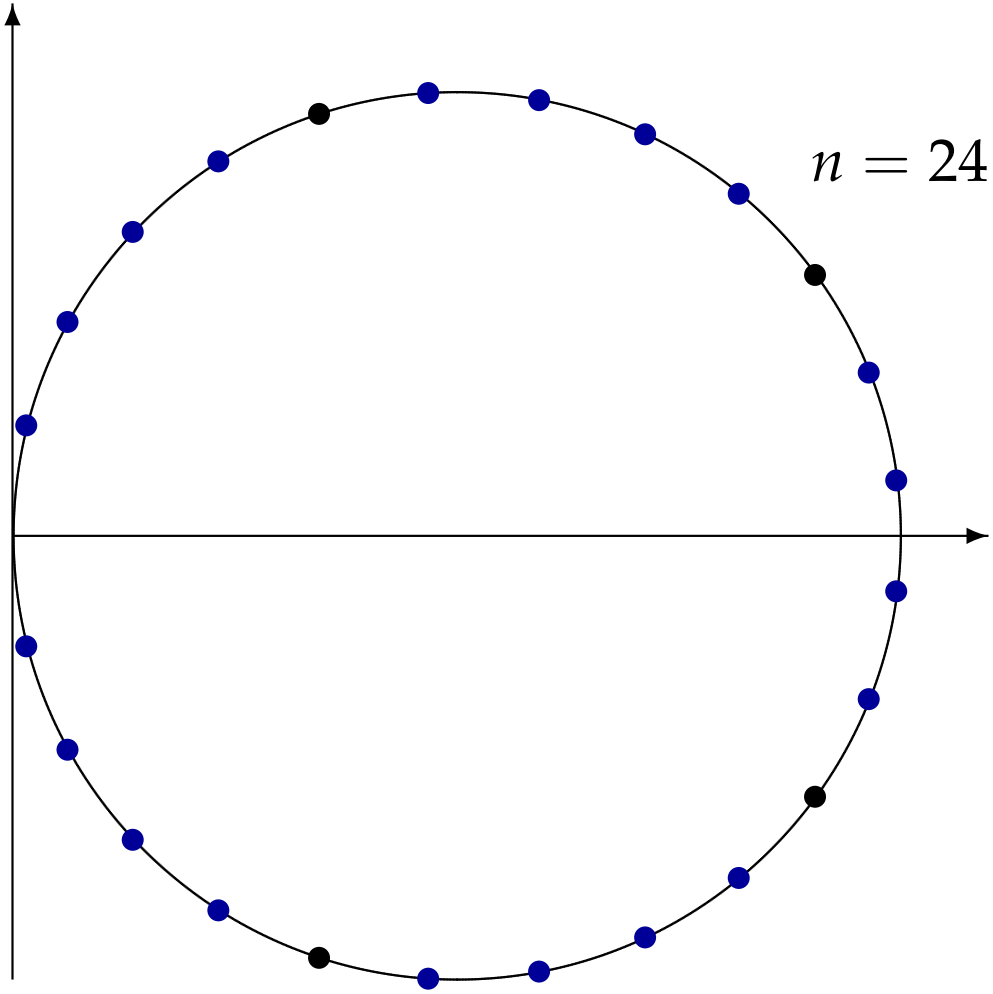
- The approximation is good between $[\epsilon, 1]$, so we rescale with

$$\lambda = 1 + 8\kappa.$$

Chebyshev Roots



Chebyshev Roots



Intermediate Filter

- Using this property of the Chebyshev polynomials, we can add an intermediate filter to our action as a bridge between the high and low energy scales, e.g.

$$S_1 = S_g \qquad S_2 = \chi^\dagger \mathcal{P}_m \chi,$$
$$S_3 = \chi^\dagger \mathcal{P}_{m|n} \chi, \quad S_4 = \phi^\dagger (M\mathcal{P}_n)^{-1} \phi.$$

- $\mathcal{P}_n$ denotes the Chebyshev polynomial of order n.
- $\mathcal{P}_{m|n}$ is defined so that $\mathcal{P}_n = \mathcal{P}_m \mathcal{P}_{m|n}$.
- This allows us to perform fermion matrix inversions even less frequently.
- This may(?) be more efficient than a single filter algorithm.
- Note: filter implementation makes use of multi-shift linear solvers for efficiency.

Intermediate Filter

- Using this property of the Chebyshev polynomials, we can add an intermediate filter to our action as a bridge between the high and low energy scales, e.g.

$$S_1 = S_g \qquad S_2 = \chi^\dagger \mathcal{P}_m \chi,$$
$$S_3 = \chi^\dagger \mathcal{P}_{m|n} \chi, \quad S_4 = \phi^\dagger (M\mathcal{P}_n)^{-1} \phi.$$

- $\mathcal{P}_n$ denotes the Chebyshev polynomial of order n.
- $\mathcal{P}_{m|n}$ is defined so that $\mathcal{P}_n = \mathcal{P}_m \mathcal{P}_{m|n}$.
- This allows us to perform fermion matrix inversions even less frequently.
- This may(?) be more efficient than a single filter algorithm.
- Note: filter implementation makes use of multi-shift linear solvers for efficiency.

Intermediate Filter

- Using this property of the Chebyshev polynomials, we can add an intermediate filter to our action as a bridge between the high and low energy scales, e.g.

$$S_1 = S_g \qquad S_2 = \chi^\dagger \mathcal{P}_m \chi,$$
$$S_3 = \chi^\dagger \mathcal{P}_{m|n} \chi, \quad S_4 = \phi^\dagger (M\mathcal{P}_n)^{-1} \phi.$$

- $\mathcal{P}_n$ denotes the Chebyshev polynomial of order n.
- $\mathcal{P}_{m|n}$ is defined so that $\mathcal{P}_n = \mathcal{P}_m \mathcal{P}_{m|n}$.
- This allows us to perform fermion matrix inversions even less frequently.
- This may(?) be more efficient than a single filter algorithm.
- Note: filter implementation makes use of multi-shift linear solvers for efficiency.

Intermediate Filter

- Using this property of the Chebyshev polynomials, we can add an intermediate filter to our action as a bridge between the high and low energy scales, e.g.

$$S_1 = S_g \qquad S_2 = \chi^\dagger \mathcal{P}_m \chi,$$
$$S_3 = \chi^\dagger \mathcal{P}_{m|n} \chi, \quad S_4 = \phi^\dagger (M\mathcal{P}_n)^{-1} \phi.$$

- $\mathcal{P}_n$ denotes the Chebyshev polynomial of order n.
- $\mathcal{P}_{m|n}$ is defined so that $\mathcal{P}_n = \mathcal{P}_m \mathcal{P}_{m|n}$.
- This allows us to perform fermion matrix inversions even less frequently.
- This may(?) be more efficient than a single filter algorithm.
- Note: filter implementation makes use of multi-shift linear solvers for efficiency.

Intermediate Filter

- Using this property of the Chebyshev polynomials, we can add an intermediate filter to our action as a bridge between the high and low energy scales, e.g.

$$S_1 = S_g \qquad S_2 = \chi^\dagger \mathcal{P}_m \chi,$$
$$S_3 = \chi^\dagger \mathcal{P}_{m|n} \chi, \quad S_4 = \phi^\dagger (M\mathcal{P}_n)^{-1} \phi.$$

- $\mathcal{P}_n$ denotes the Chebyshev polynomial of order n.
- $\mathcal{P}_{m|n}$ is defined so that $\mathcal{P}_n = \mathcal{P}_m \mathcal{P}_{m|n}$.
- This allows us to perform fermion matrix inversions even less frequently.
- This may(?) be more efficient than a single filter algorithm.
- Note: filter implementation makes use of multi-shift linear solvers for efficiency.

Intermediate Filter

- Using this property of the Chebyshev polynomials, we can add an intermediate filter to our action as a bridge between the high and low energy scales, e.g.

$$S_1 = S_g \qquad S_2 = \chi^\dagger \mathcal{P}_m \chi,$$
$$S_3 = \chi^\dagger \mathcal{P}_{m|n} \chi, \quad S_4 = \phi^\dagger (M\mathcal{P}_n)^{-1} \phi.$$

- $\mathcal{P}_n$ denotes the Chebyshev polynomial of order n.
- $\mathcal{P}_{m|n}$ is defined so that $\mathcal{P}_n = \mathcal{P}_m \mathcal{P}_{m|n}$.
- This allows us to perform fermion matrix inversions even less frequently.
- This may(?) be more efficient than a single filter algorithm.
- Note: filter implementation makes use of multi-shift linear solvers for efficiency.

Integration Scheme

- With a second filter, we have four different scales, five (or more) if we are doing a 2+1 flavour simulation.
- Nested leapfrog is too cumbersome for fine tuning these scales.
- We make use of the fact that $V_i = V_{S_i}$ all commute to introduce a generalised integration scheme.
- If N_i is the number of integration steps per trajectory, then nested leapfrog requires

$$N_i | N_{i-1} \forall i$$

- The new integration scheme requires only that

$$N_i | N_1 \forall i$$

Integration Scheme

- With a second filter, we have four different scales, five (or more) if we are doing a 2+1 flavour simulation.
- Nested leapfrog is too cumbersome for fine tuning these scales.
- We make use of the fact that $V_i = V_{S_i}$ all commute to introduce a generalised integration scheme.
- If N_i is the number of integration steps per trajectory, then nested leapfrog requires

$$N_i | N_{i-1} \forall i$$

- The new integration scheme requires only that

$$N_i | N_1 \forall i$$

Integration Scheme

- With a second filter, we have four different scales, five (or more) if we are doing a 2+1 flavour simulation.
- Nested leapfrog is too cumbersome for fine tuning these scales.
- We make use of the fact that $V_i = V_{S_i}$ all commute to introduce a generalised integration scheme.
- If N_i is the number of integration steps per trajectory, then nested leapfrog requires

$$N_i | N_{i-1} \forall i$$

- The new integration scheme requires only that

$$N_i | N_1 \forall i$$

Integration Scheme

- With a second filter, we have four different scales, five (or more) if we are doing a 2+1 flavour simulation.
- Nested leapfrog is too cumbersome for fine tuning these scales.
- We make use of the fact that $V_i = V_{S_i}$ all commute to introduce a generalised integration scheme.
- If N_i is the number of integration steps per trajectory, then nested leapfrog requires

$$N_i | N_{i-1} \forall i$$

- The new integration scheme requires only that

$$N_i | N_1 \forall i$$

Integration Scheme

- With a second filter, we have four different scales, five (or more) if we are doing a 2+1 flavour simulation.
- Nested leapfrog is too cumbersome for fine tuning these scales.
- We make use of the fact that $V_i = V_{S_i}$ all commute to introduce a generalised integration scheme.
- If N_i is the number of integration steps per trajectory, then nested leapfrog requires

$$N_i | N_{i-1} \forall i$$

- The new integration scheme requires only that

$$N_i | N_1 \forall i$$

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 = j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

Generalised Leapfrog

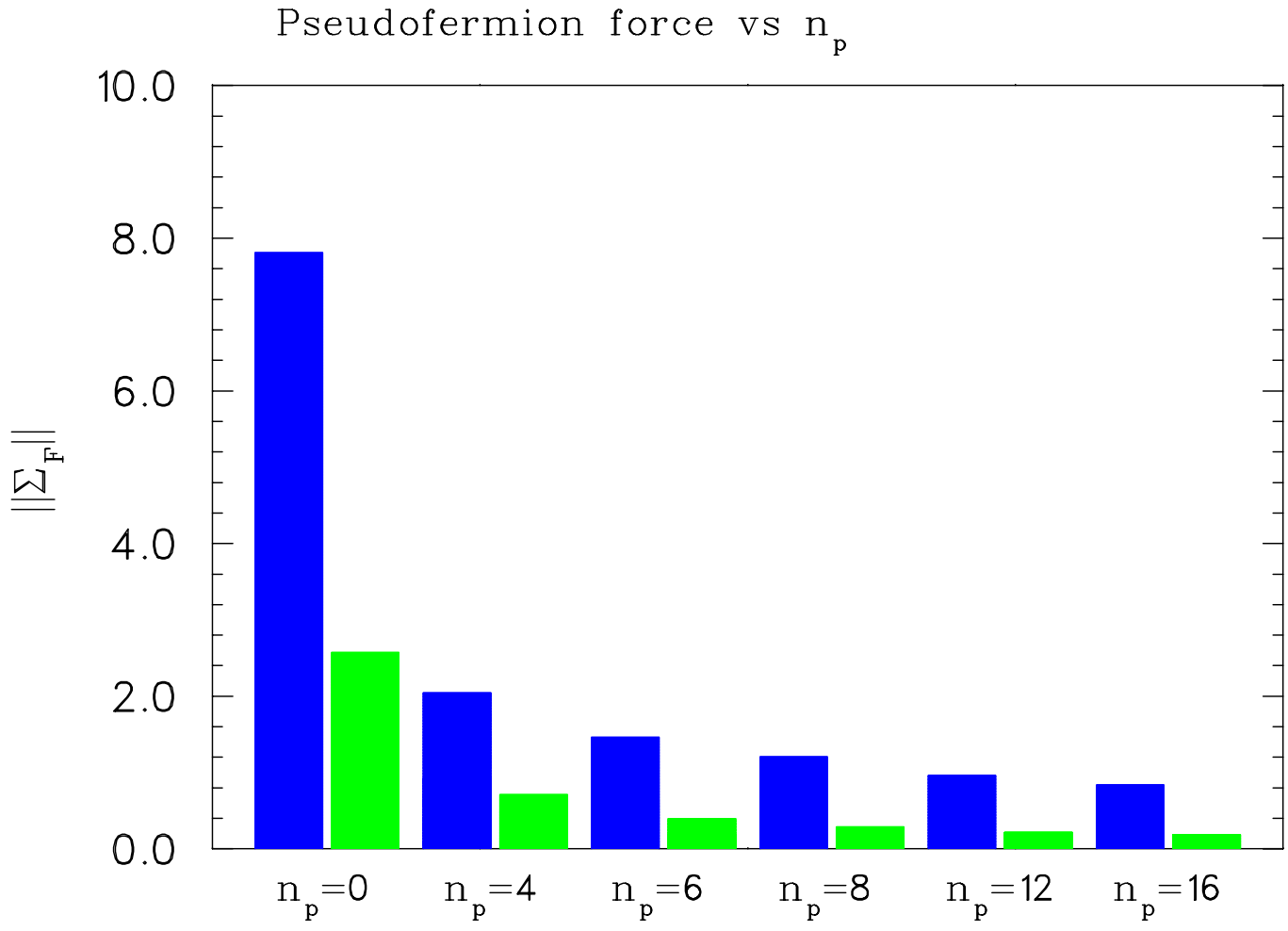
- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

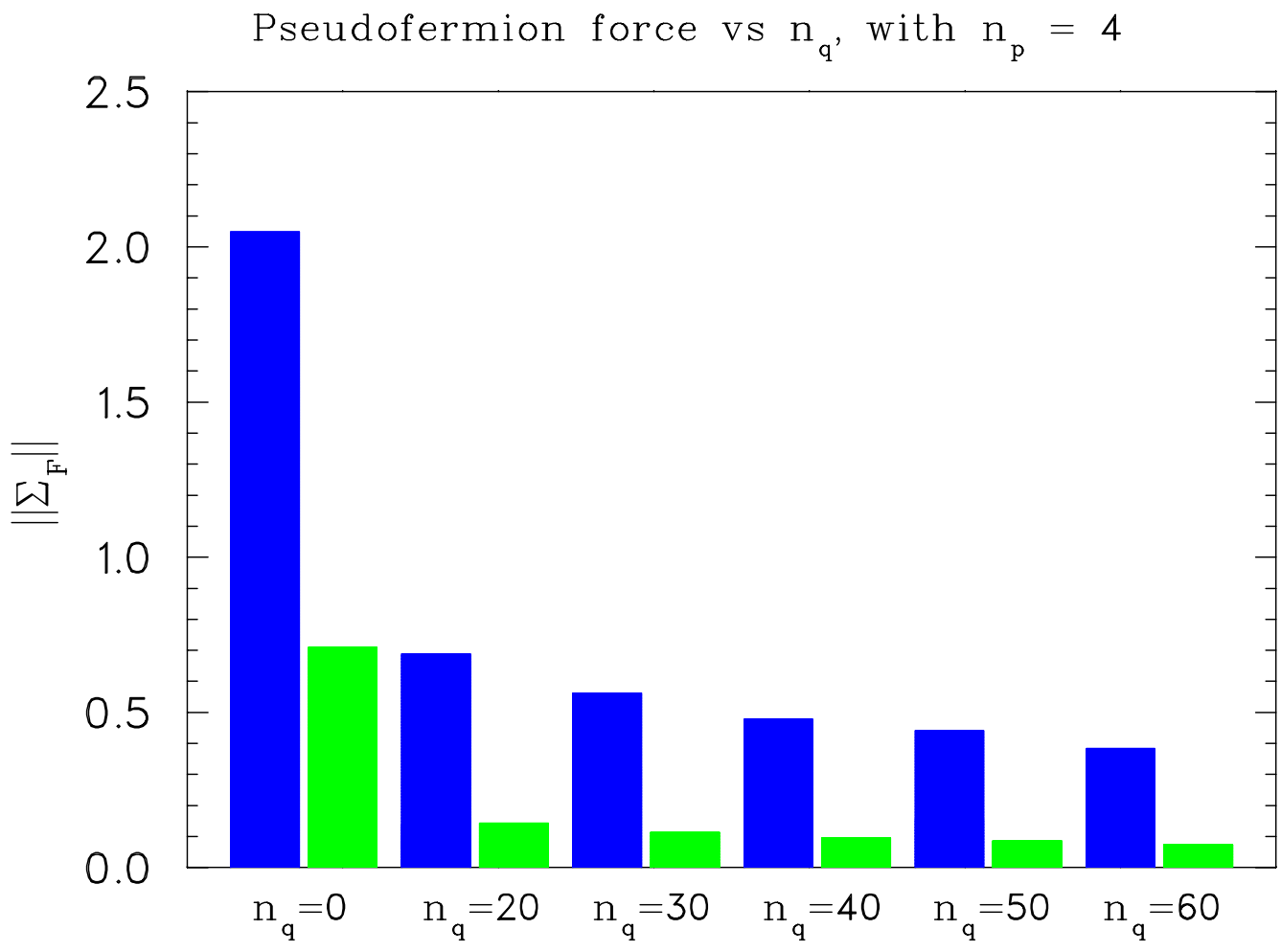
Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

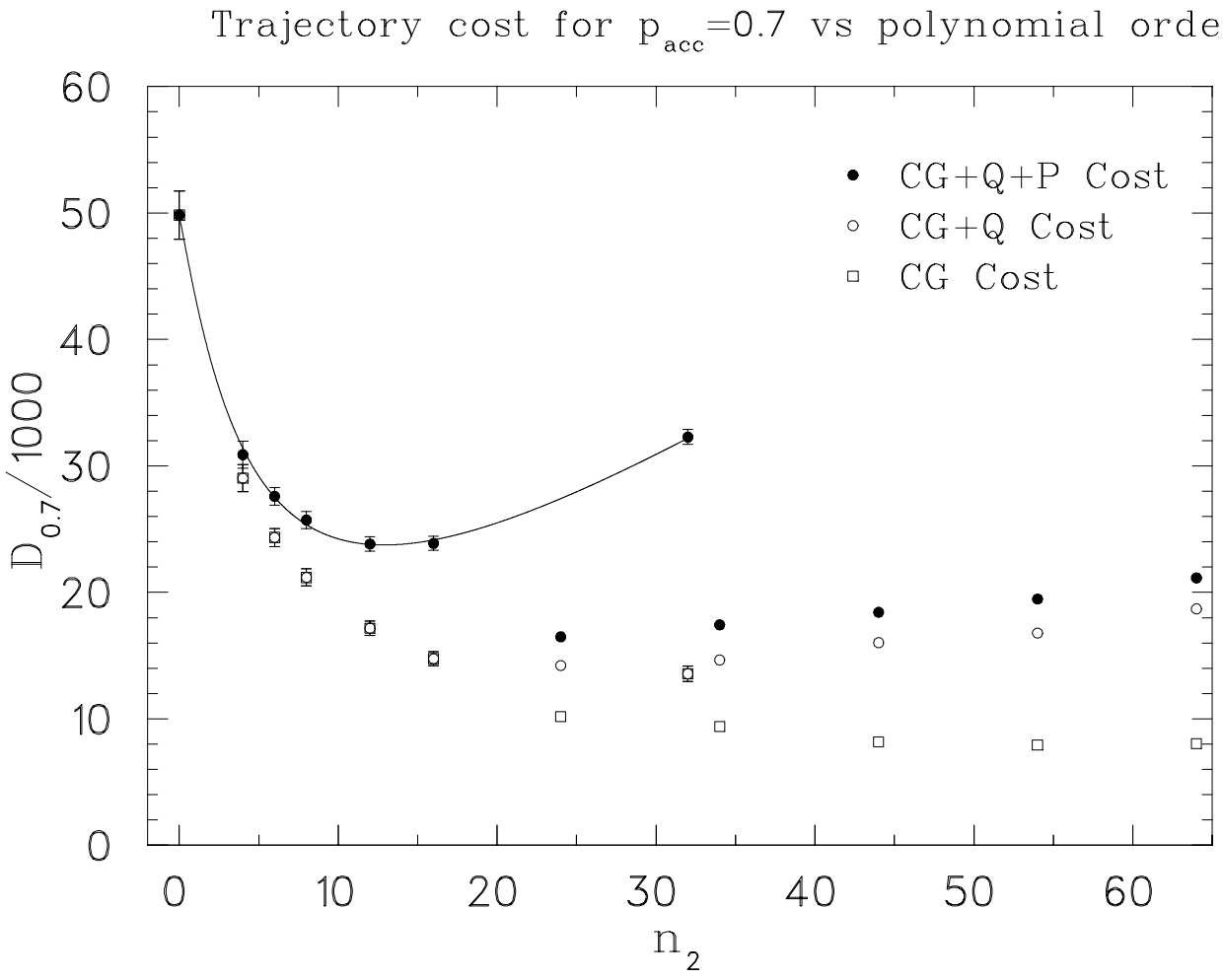
Generalised Leapfrog

- Set $N = N_1$ and set $n_i = N_1 / N_i$
- The generalised leapfrog algorithm is then:
  1. Perform an initial half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
  2. Loop over $j = 1$ to $N - 1$
     - Apply $V_T(\Delta\tau)$ to update U.
     - If $\{0 \equiv j \mod N_i\}$ apply $V_i(\Delta\tau_i)$ to update P
  3. Apply $V_T(\Delta\tau)$ to update U.
  4. Perform a final half-step $V_i(\frac{1}{2}\Delta\tau_i)$ updating P for all i.
- Can show using BCH that it has errors of $\mathcal{O}[(\Delta\tau)^3]$.
- Reduces to nested leapfrog for $N_i | N_{i-1}$.

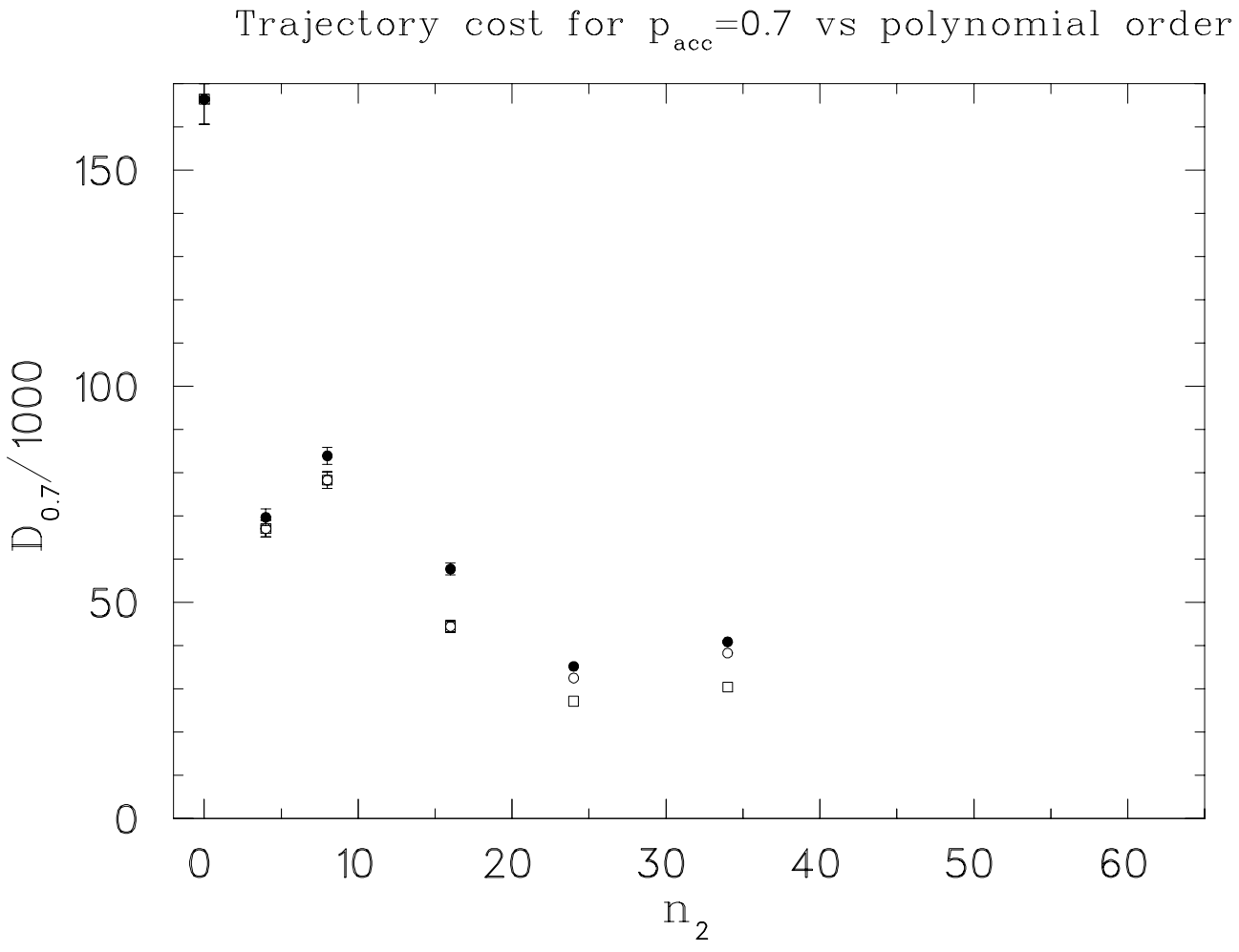

Pseudofermion force vs n_p
10.0
8.0
6.0
4.0
2.0
0.0
$||\Sigma_F||$
$n_p=0$
$n_p=4$
$n_p=6$
$n_p=8$
$n_p=12$
$n_p=16$


Pseudofermion force vs n_q, with n_p = 4
2.5
2.0
1.5
1.0
0.5
0.0
$||\Sigma_F||$
n_q=0
n_q=20
n_q=30
n_q=40
n_q=50
n_q=60

Results $\kappa = 0.1575, m_\pi = 665\text{MeV}$



Results $\kappa = 0.15825, m_\pi \approx 400\text{MeV}$



Single Flavour QCD

- Single fermion flavours can be simulated using a rational polynomial,

$$\mathcal{R}(M) = \sum \frac{a_i}{M + b_i} \approx \frac{1}{\sqrt{M}}$$

  (or some other method e.g. polynomial approx. to $1/\sqrt{M}$).
- Recall $M = D_w^\dagger D_w$ is Hermitian positive definite.
- Can we extend our polynomial filtering technique to single flavour simulations?
- Suppose we have a polynomial $\mathcal{Q}$ such that

$$\mathcal{Q}(M) \approx \frac{1}{\sqrt{M}}$$

- E.g. numerically calculate coefficients for Chebyshev approximation to $1/\sqrt{z}$, then calculate the roots...

Single Flavour QCD

- Single fermion flavours can be simulated using a rational polynomial,

$$\mathcal{R}(M) = \sum \frac{a_i}{M + b_i} \approx \frac{1}{\sqrt{M}}$$

  (or some other method e.g. polynomial approx. to $1/\sqrt{M}$).
- Recall $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$ is Hermitian positive definite.
- Can we extend our polynomial filtering technique to single flavour simulations?
- Suppose we have a polynomial $\mathcal{Q}$ such that

$$\mathcal{Q}(M) \approx \frac{1}{\sqrt{M}}$$

- E.g. numerically calculate coefficients for Chebyshev approximation to $1/\sqrt{z}$, then calculate the roots...

Single Flavour QCD

- Single fermion flavours can be simulated using a rational polynomial,

$$\mathcal{R}(M) = \sum \frac{a_i}{M + b_i} \approx \frac{1}{\sqrt{M}}$$

  (or some other method e.g. polynomial approx. to $1/\sqrt{M}$).
- Recall $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$ is Hermitian positive definite.
- Can we extend our polynomial filtering technique to single flavour simulations?
- Suppose we have a polynomial $\mathcal{Q}$ such that

$$\mathcal{Q}(M) \approx \frac{1}{\sqrt{M}}$$

- E.g. numerically calculate coefficients for Chebyshev approximation to $1/\sqrt{z}$, then calculate the roots...

Single Flavour QCD

- Single fermion flavours can be simulated using a rational polynomial,

$$\mathcal{R}(M) = \sum \frac{a_i}{M + b_i} \approx \frac{1}{\sqrt{M}}$$

  (or some other method e.g. polynomial approx. to $1/\sqrt{M}$).
- Recall $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$ is Hermitian positive definite.
- Can we extend our polynomial filtering technique to single flavour simulations?
- Suppose we have a polynomial $\mathcal{Q}$ such that

$$\mathcal{Q}(M) \approx \frac{1}{\sqrt{M}}$$

- E.g. numerically calculate coefficients for Chebyshev approximation to $1/\sqrt{z}$, then calculate the roots...

Single Flavour QCD

- Single fermion flavours can be simulated using a rational polynomial,

$$\mathcal{R}(M) = \sum \frac{a_i}{M + b_i} \approx \frac{1}{\sqrt{M}}$$

  (or some other method e.g. polynomial approx. to $1/\sqrt{M}$).
- Recall $M = D_{\mathrm{w}}^{\dagger} D_{\mathrm{w}}$ is Hermitian positive definite.
- Can we extend our polynomial filtering technique to single flavour simulations?
- Suppose we have a polynomial $\mathcal{Q}$ such that

$$\mathcal{Q}(M) \approx \frac{1}{\sqrt{M}}$$

- E.g. numerically calculate coefficients for Chebyshev approximation to $1/\sqrt{z}$, then calculate the roots...

Single Flavour Polynomial Filter

- Then we can write

$$S_{\text{poly1f}} = \chi^{\dagger}_{\text{1f}} \mathcal{Q} \chi_{\text{1f}},$$

$$S_{\text{1pf}} = \phi^{\dagger}_{\text{1f}} \mathcal{R} \mathcal{Q}^{-1} \phi_{\text{1f}}.$$

- As before, the determinant is unaffected by the addition of the polynomial filter.
- Could add an intermediate filter for the single flavour as before but at the strange quark mass probably not worth it.
- Knowing roots of $\mathcal{Q}$ is needed to rewrite $\mathcal{R}\mathcal{Q}^{-1}$ as a sum over poles.
- This allows the use of efficient linear multi-shift system solvers.

Single Flavour Polynomial Filter

- Then we can write

$$S_{\text{poly1f}} = \chi^{\dagger}_{\text{1f}} \mathcal{Q} \chi_{\text{1f}},$$

$$S_{\text{1pf}} = \phi^{\dagger}_{\text{1f}} \mathcal{R} \mathcal{Q}^{-1} \phi_{\text{1f}}.$$

- As before, the determinant is unaffected by the addition of the polynomial filter.
- Could add an intermediate filter for the single flavour as before but at the strange quark mass probably not worth it.
- Knowing roots of $\mathcal{Q}$ is needed to rewrite $\mathcal{R}\mathcal{Q}^{-1}$ as a sum over poles.
- This allows the use of efficient linear multi-shift system solvers.

Single Flavour Polynomial Filter

- Then we can write

$$S_{\text{poly1f}} = \chi^{\dagger}_{\text{1f}} \mathcal{Q} \chi_{\text{1f}},$$

$$S_{\text{1pf}} = \phi^{\dagger}_{\text{1f}} \mathcal{R} \mathcal{Q}^{-1} \phi_{\text{1f}}.$$

- As before, the determinant is unaffected by the addition of the polynomial filter.
- Could add an intermediate filter for the single flavour as before but at the strange quark mass probably not worth it.
- Knowing roots of $\mathcal{Q}$ is needed to rewrite $\mathcal{R}\mathcal{Q}^{-1}$ as a sum over poles.
- This allows the use of efficient linear multi-shift system solvers.

Single Flavour Polynomial Filter

- Then we can write

$$S_{\text{poly1f}} = \chi^{\dagger}_{\text{1f}} \mathcal{Q} \chi_{\text{1f}},$$

$$S_{\text{1pf}} = \phi^{\dagger}_{\text{1f}} \mathcal{R} \mathcal{Q}^{-1} \phi_{\text{1f}}.$$

- As before, the determinant is unaffected by the addition of the polynomial filter.
- Could add an intermediate filter for the single flavour as before but at the strange quark mass probably not worth it.
- Knowing roots of $\mathcal{Q}$ is needed to rewrite $\mathcal{R}\mathcal{Q}^{-1}$ as a sum over poles.
- This allows the use of efficient linear multi-shift system solvers.

Single Flavour Polynomial Filter

- Then we can write

$$S_{\text{poly1f}} = \chi^{\dagger}_{\text{1f}} \mathcal{Q} \chi_{\text{1f}},$$

$$S_{\text{1pf}} = \phi^{\dagger}_{\text{1f}} \mathcal{R} \mathcal{Q}^{-1} \phi_{\text{1f}}.$$

- As before, the determinant is unaffected by the addition of the polynomial filter.
- Could add an intermediate filter for the single flavour as before but at the strange quark mass probably not worth it.
- Knowing roots of $\mathcal{Q}$ is needed to rewrite $\mathcal{R}\mathcal{Q}^{-1}$ as a sum over poles.
- This allows the use of efficient linear multi-shift system solvers.

Alternative Approximation

- Note that $\mathcal{R}\mathcal{Q}^{-1} \approx 1$.
- May be advantageous to simply use the Remes algorithm to get a rational approximation to

$$\tilde{\mathcal{R}}(z) \approx f(z) = \frac{1}{\sqrt{z}\mathcal{Q}(z)}$$

  and use that instead of the product $\mathcal{R}\mathcal{Q}^{-1}$.
- What are the possible advantages?
- The rational approximation to f might have improved precision for a given order.
- Smallest shift for $\tilde{\mathcal{R}}$ expressed as a sum over poles might be bigger.

  $\Longrightarrow$ Less iterations to solve.

Alternative Approximation

- Note that $\mathcal{R}\mathcal{Q}^{-1} \approx 1$.
- May be advantageous to simply use the Remes algorithm to get a rational approximation to

$$\tilde{\mathcal{R}}(z) \approx f(z) = \frac{1}{\sqrt{z}\mathcal{Q}(z)}$$

  and use that instead of the product $\mathcal{R}\mathcal{Q}^{-1}$.
- What are the possible advantages?
- The rational approximation to f might have improved precision for a given order.
- Smallest shift for $\tilde{\mathcal{R}}$ expressed as a sum over poles might be bigger.

  $\Longrightarrow$ Less iterations to solve.

Alternative Approximation

- Note that $\mathcal{R}\mathcal{Q}^{-1} \approx 1$.
- May be advantageous to simply use the Remes algorithm to get a rational approximation to

$$\tilde{\mathcal{R}}(z) \approx f(z) = \frac{1}{\sqrt{z}\mathcal{Q}(z)}$$

  and use that instead of the product $\mathcal{R}\mathcal{Q}^{-1}$.
- What are the possible advantages?
- The rational approximation to f might have improved precision for a given order.
- Smallest shift for $\tilde{\mathcal{R}}$ expressed as a sum over poles might be bigger.

  $\Longrightarrow$ Less iterations to solve.

Alternative Approximation

- Note that $\mathcal{R}\mathcal{Q}^{-1} \approx 1$.
- May be advantageous to simply use the Remes algorithm to get a rational approximation to

$$\tilde{\mathcal{R}}(z) \approx f(z) = \frac{1}{\sqrt{z}\mathcal{Q}(z)}$$

  and use that instead of the product $\mathcal{R}\mathcal{Q}^{-1}$.
- What are the possible advantages?
- The rational approximation to f might have improved precision for a given order.
- Smallest shift for $\tilde{\mathcal{R}}$ expressed as a sum over poles might be bigger.

  $\Longrightarrow$ Less iterations to solve.

Alternative Approximation

- Note that $\mathcal{R}\mathcal{Q}^{-1} \approx 1$.
- May be advantageous to simply use the Remes algorithm to get a rational approximation to

$$\tilde{\mathcal{R}}(z) \approx f(z) = \frac{1}{\sqrt{z}\mathcal{Q}(z)}$$

  and use that instead of the product $\mathcal{R}\mathcal{Q}^{-1}$.
- What are the possible advantages?
- The rational approximation to f might have improved precision for a given order.
- Smallest shift for $\tilde{\mathcal{R}}$ expressed as a sum over poles might be bigger.

$\Longrightarrow$ Less iterations to solve.

Alternative Approximation

- Note that $\mathcal{R}\mathcal{Q}^{-1} \approx 1$.
- May be advantageous to simply use the Remes algorithm to get a rational approximation to

$$\tilde{\mathcal{R}}(z) \approx f(z) = \frac{1}{\sqrt{z}\mathcal{Q}(z)}$$

  and use that instead of the product $\mathcal{R}\mathcal{Q}^{-1}$.
- What are the possible advantages?
- The rational approximation to f might have improved precision for a given order.
- Smallest shift for $\tilde{\mathcal{R}}$ expressed as a sum over poles might be bigger.

  $\Longrightarrow$ Less iterations to solve.

Another Variant

- Recall that we used the Hermitian Chebyshev approximation to $1/z$ in our two flavour polynomial filter.
- The non-Hermitian Chebyshev approximation $\mathcal{K}(z)$ to $1/z$ has the same normalisation, but slightly different roots,

$$y_k = d(1 - \cos\theta_k) + i\sqrt{d^2 - c^2}\sin\theta_k.$$

- Valid for an elliptical region in the complex plane.
- So long as the spectrum of the non-Hermitian matrix D_{w} is within this ellipse, we can write

$$\mathcal{K}(D_{\mathrm{w}}) = a_n \prod_{k=1}^{n/2} (D_{\mathrm{w}} - y_k)(D_{\mathrm{w}} - y_k^*) \approx \frac{1}{D_{\mathrm{w}}}$$

Another Variant

- Recall that we used the Hermitian Chebyshev approximation to $1/z$ in our two flavour polynomial filter.
- The non-Hermitian Chebyshev approximation $\mathcal{K}(z)$ to $1/z$ has the same normalisation, but slightly different roots,

$$y_k = d(1 - \cos\theta_k) + i\sqrt{d^2 - c^2}\sin\theta_k.$$

- Valid for an elliptical region in the complex plane.
- So long as the spectrum of the non-Hermitian matrix D_w is within this ellipse, we can write

$$\mathcal{K}(D_\mathrm{w}) = a_n \prod_{k=1}^{n/2} (D_\mathrm{w} - y_k)(D_\mathrm{w} - y_k^*) \approx \frac{1}{D_\mathrm{w}}$$

Another Variant

- Recall that we used the Hermitian Chebyshev approximation to $1/z$ in our two flavour polynomial filter.
- The non-Hermitian Chebyshev approximation $\mathcal{K}(z)$ to $1/z$ has the same normalisation, but slightly different roots,

$$y_k = d(1 - \cos\theta_k) + i\sqrt{d^2 - c^2}\sin\theta_k.$$

- Valid for an elliptical region in the complex plane.
- So long as the spectrum of the non-Hermitian matrix D_{w} is within this ellipse, we can write

$$\mathcal{K}(D_{\mathrm{w}}) = a_n \prod_{k=1}^{n/2} (D_{\mathrm{w}} - y_k)(D_{\mathrm{w}} - y_k^*) \approx \frac{1}{D_{\mathrm{w}}}$$

Another Variant

- Recall that we used the Hermitian Chebyshev approximation to $1/z$ in our two flavour polynomial filter.
- The non-Hermitian Chebyshev approximation $\mathcal{K}(z)$ to $1/z$ has the same normalisation, but slightly different roots,

$$y_k = d(1 - \cos\theta_k) + i\sqrt{d^2 - c^2}\sin\theta_k.$$

- Valid for an elliptical region in the complex plane.
- So long as the spectrum of the non-Hermitian matrix D_{w} is within this ellipse, we can write

$$\mathcal{K}(D_{\mathrm{w}}) = a_n \prod_{k=1}^{n/2} (D_{\mathrm{w}} - y_k)(D_{\mathrm{w}} - y_k^*) \approx \frac{1}{D_{\mathrm{w}}}$$

Factoring the Polynomial

- Now construct a polynomial using only half the roots (say those with positive imaginary parts),

$$\mathcal{K}_+(D_\mathrm{w}) = \sqrt{a_n}\prod_{k=1}^{n/2}(D_\mathrm{w} - y_k)$$

- We need the following two properties of determinants,

$$\det(AB) = \det A \det B$$
$$\det A^\dagger = (\det A)^*$$

- Using these we can deduce that

$$\det \mathcal{K}_+^\dagger(D_\mathrm{w})\mathcal{K}_+(D_\mathrm{w}) = \det \mathcal{K}(D_\mathrm{w}) \approx (\det D_\mathrm{w})^{-1}$$

- So long as $\det D_\mathrm{w}$ is real and positive, this is the correct weighting for a single fermion flavour.

Factoring the Polynomial

- Now construct a polynomial using only half the roots (say those with positive imaginary parts),

$$\mathcal{K}_+(D_\mathrm{w}) = \sqrt{a_n}\prod_{k=1}^{n/2}(D_\mathrm{w} - y_k)$$

- We need the following two properties of determinants,

$$\det(AB) = \det A \det B$$
$$\det A^\dagger = (\det A)^*$$

- Using these we can deduce that

$$\det \mathcal{K}_+^\dagger(D_\mathrm{w})\mathcal{K}_+(D_\mathrm{w}) = \det \mathcal{K}(D_\mathrm{w}) \approx (\det D_\mathrm{w})^{-1}$$

- So long as $\det D_\mathrm{w}$ is real and positive, this is the correct weighting for a single fermion flavour.

Factoring the Polynomial

- Now construct a polynomial using only half the roots (say those with positive imaginary parts),

$$\mathcal{K}_+(D_\mathrm{w}) = \sqrt{a_n}\prod_{k=1}^{n/2}(D_\mathrm{w} - y_k)$$

- We need the following two properties of determinants,

$$\det(AB) = \det A \det B$$
$$\det A^\dagger = (\det A)^*$$

- Using these we can deduce that

$$\det \mathcal{K}_+^\dagger(D_\mathrm{w})\mathcal{K}_+(D_\mathrm{w}) = \det \mathcal{K}(D_\mathrm{w}) \approx (\det D_\mathrm{w})^{-1}$$

- So long as $\det D_\mathrm{w}$ is real and positive, this is the correct weighting for a single fermion flavour.

Factoring the Polynomial

- Now construct a polynomial using only half the roots (say those with positive imaginary parts),

$$\mathcal{K}_+(D_\mathrm{w}) = \sqrt{a_n}\prod_{k=1}^{n/2}(D_\mathrm{w} - y_k)$$

- We need the following two properties of determinants,

$$\det(AB) = \det A \det B$$
$$\det A^\dagger = (\det A)^*$$

- Using these we can deduce that

$$\det \mathcal{K}_+^\dagger(D_\mathrm{w})\mathcal{K}_+(D_\mathrm{w}) = \det \mathcal{K}(D_\mathrm{w}) \approx (\det D_\mathrm{w})^{-1}$$

- So long as $\det D_\mathrm{w}$ is real and positive, this is the correct weighting for a single fermion flavour.

The Action

- So we can the construct a polynomial filtered one-flavour action using $\mathcal{K}_+$,

$$S_{\text{poly1f}} = \chi^\dagger_{1\text{f}} \mathcal{K}^\dagger_+ \mathcal{K}_+ \chi_{1\text{f}},$$

$$S_{1\text{pf}} = \phi^\dagger_{1\text{f}} \mathcal{W}^\dagger(D_{\text{w}}) \mathcal{W}(D_{\text{w}}) \phi_{1\text{f}}.$$

- Here, $\mathcal{W}$ needs to be a rational polynomial approximation to $\{z\mathcal{K}^*_+(z)\mathcal{K}_+(z)\}^{-1}$.
- Should be able to obtain this by factoring $\mathcal{R}(z)$ the Zolotarev approximation to $1/z$ and setting

$$\mathcal{W}(z) = \mathcal{R}_+(z)\mathcal{K}^{-1}_+.$$

- At this point, it might be more efficient to take large n polynomial limit for $\mathcal{K}_+$, split it into two and do filtered polynomial HMC for the single flavour...

The Action

- So we can the construct a polynomial filtered one-flavour action using $\mathcal{K}_+$,

$$S_{\text{poly1f}} = \chi^\dagger_{1\text{f}} \mathcal{K}^\dagger_+ \mathcal{K}_+ \chi_{1\text{f}},$$

$$S_{1\text{pf}} = \phi^\dagger_{1\text{f}} \mathcal{W}^\dagger(D_{\text{w}}) \mathcal{W}(D_{\text{w}}) \phi_{1\text{f}}.$$

- Here, $\mathcal{W}$ needs to be a rational polynomial approximation to $\{z\mathcal{K}^*_+(z)\mathcal{K}_+(z)\}^{-1}$.
- Should be able to obtain this by factoring $\mathcal{R}(z)$ the Zolotarev approximation to $1/z$ and setting

$$\mathcal{W}(z) = \mathcal{R}_+(z)\mathcal{K}^{-1}_+.$$

- At this point, it might be more efficient to take large n polynomial limit for $\mathcal{K}_+$, split it into two and do filtered polynomial HMC for the single flavour...

The Action

- So we can the construct a polynomial filtered one-flavour action using $\mathcal{K}_+$,

$$S_{\text{poly1f}} = \chi^\dagger_{1\text{f}} \mathcal{K}^\dagger_+ \mathcal{K}_+ \chi_{1\text{f}},$$

$$S_{1\text{pf}} = \phi^\dagger_{1\text{f}} \mathcal{W}^\dagger(D_{\text{w}}) \mathcal{W}(D_{\text{w}}) \phi_{1\text{f}}.$$

- Here, $\mathcal{W}$ needs to be a rational polynomial approximation to $\{z\mathcal{K}^*_+(z)\mathcal{K}_+(z)\}^{-1}$.
- Should be able to obtain this by factoring $\mathcal{R}(z)$ the Zolotarev approximation to $1/z$ and setting

$$\mathcal{W}(z) = \mathcal{R}_+(z)\mathcal{K}^{-1}_+.$$

- At this point, it might be more efficient to take large n polynomial limit for $\mathcal{K}_+$, split it into two and do filtered polynomial HMC for the single flavour...

The Action

- So we can the construct a polynomial filtered one-flavour action using $\mathcal{K}_+$,

$$S_{\text{poly1f}} = \chi^\dagger_{1\text{f}} \mathcal{K}^\dagger_+ \mathcal{K}_+ \chi_{1\text{f}},$$

$$S_{1\text{pf}} = \phi^\dagger_{1\text{f}} \mathcal{W}^\dagger(D_{\text{w}}) \mathcal{W}(D_{\text{w}}) \phi_{1\text{f}}.$$

- Here, $\mathcal{W}$ needs to be a rational polynomial approximation to $\{z\mathcal{K}^*_+(z)\mathcal{K}_+(z)\}^{-1}$.
- Should be able to obtain this by factoring $\mathcal{R}(z)$ the Zolotarev approximation to $1/z$ and setting

$$\mathcal{W}(z) = \mathcal{R}_+(z)\mathcal{K}^{-1}_+.$$

- At this point, it might be more efficient to take large n polynomial limit for $\mathcal{K}_+$, split it into two and do filtered polynomial HMC for the single flavour...

Conclusions

- The use of a polynomial approximation to the inverse as a filter successfully separates the UV and IR pseudofermion dynamics.
- In combination with a generalise leapfrog algorithm we successfully reduce the cost of dynamical simulations.
- The generalised leapfrog algorithm is applicable to any multiple time scale integration scheme, far more flexible than nested leapfrog.
- Technique is not necessarily orthogonal to other improvements (e.g. DD, Hasenbuch trick).
- Polynomial filtering can also be extended to single flavour simulations.
- More results to come...

Conclusions

- The use of a polynomial approximation to the inverse as a filter successfully separates the UV and IR pseudofermion dynamics.
- In combination with a generalise leapfrog algorithm we successfully reduce the cost of dynamical simulations.
- The generalised leapfrog algorithm is applicable to any multiple time scale integration scheme, far more flexible than nested leapfrog.
- Technique is not necessarily orthogonal to other improvements (e.g. DD, Hasenbuch trick).
- Polynomial filtering can also be extended to single flavour simulations.
- More results to come...

Conclusions

- The use of a polynomial approximation to the inverse as a filter successfully separates the UV and IR pseudofermion dynamics.
- In combination with a generalise leapfrog algorithm we successfully reduce the cost of dynamical simulations.
- The generalised leapfrog algorithm is applicable to any multiple time scale integration scheme, far more flexible than nested leapfrog.
- Technique is not necessarily orthogonal to other improvements (e.g. DD, Hasenbuch trick).
- Polynomial filtering can also be extended to single flavour simulations.
- More results to come...

Conclusions

- The use of a polynomial approximation to the inverse as a filter successfully separates the UV and IR pseudofermion dynamics.
- In combination with a generalise leapfrog algorithm we successfully reduce the cost of dynamical simulations.
- The generalised leapfrog algorithm is applicable to any multiple time scale integration scheme, far more flexible than nested leapfrog.
- Technique is not necessarily orthogonal to other improvements (e.g. DD, Hasenbuch trick).
- Polynomial filtering can also be extended to single flavour simulations.
- More results to come...

Conclusions

- The use of a polynomial approximation to the inverse as a filter successfully separates the UV and IR pseudofermion dynamics.
- In combination with a generalise leapfrog algorithm we successfully reduce the cost of dynamical simulations.
- The generalised leapfrog algorithm is applicable to any multiple time scale integration scheme, far more flexible than nested leapfrog.
- Technique is not necessarily orthogonal to other improvements (e.g. DD, Hasenbuch trick).
- Polynomial filtering can also be extended to single flavour simulations.
- More results to come...

Conclusions

- The use of a polynomial approximation to the inverse as a filter successfully separates the UV and IR pseudofermion dynamics.
- In combination with a generalise leapfrog algorithm we successfully reduce the cost of dynamical simulations.
- The generalised leapfrog algorithm is applicable to any multiple time scale integration scheme, far more flexible than nested leapfrog.
- Technique is not necessarily orthogonal to other improvements (e.g. DD, Hasenbuch trick).
- Polynomial filtering can also be extended to single flavour simulations.
- More results to come...