Computational Methods (practice) - Lecture 4

Peter Boyle (BNL, Edinburgh)

- Monte Carlo Integration
- Euclidean path integral and HMC
- Pseudofermions
- Writing and running an HMC
- Odd flavours

Monte Carlo Integration

- Integration

$$\int_U f(U)dU = Vol \times \langle f \rangle \qquad ; \qquad Vol = \int_U dU$$

- Monte Carlo Integration (x_i uniform over compact domain)

$$\langle f \rangle = \frac{1}{N} \sum_i f(x_i)$$

- Importance sampling: draw x_i with positive normalised probability density $P(x_i)$

$$\langle f \rangle = \frac{1}{N} \sum_i \frac{f(x_i)}{P(x_i)}$$

- If $|f(x_i)| \propto P(x_i)$ this may converge better.
- Variance reduction: if $\tilde{f}$ is a good, cheap approximation for f

$$\langle f \rangle = \frac{1}{N \times M} \sum_i \frac{\tilde{f}(x_i)}{P(x_i)} + \frac{1}{N} \sum_j \frac{f(x_j) - \tilde{f}(x_j)}{P(x_j)}$$

Euclidean Path Integral

- Pure gauge path integral

$$\frac{1}{Z}\int_U e^{-S_G[U]}\mathcal{O}(U)dU$$

- Importance sample: seek to distribute gluon configuratoins according to

$$P(U) = \frac{e^{-S_G[U]}}{\int_U e^{-S_G[U]}dU}$$

- Calculate observables on each *configuration*

$$\langle\mathcal{O}\rangle = \frac{1}{N}\sum_i \mathcal{O}(U_i)$$

- $\Rightarrow$ Markov chain monte carlo:
  - 10^{10} degrees of freedom(!)
  - Sharply probability weight
  - Variance of $\mathcal{O}(U)$ determines how many samples are required.
  - 100-2000 samples typically good for 1% scale statistical errors
- This is observable dependent...

Markov chains

- Sequence of states generated by *transition probability* $M(X \to X')$ from X to X'
  Rule depends only on X.
- Usually composed of proposal and acceptance probablities

$$M(X \to X') = P_p(X \to X')P_{acc}(X \to X')$$

- Design rule $M(X \to X')$ to yield *desired* equilibrium probability distribution after many transitions$P_{eq}(X)$
- P_{eq} must map to itself under of the transition rule:

$$P_{eq}(X') = \sum_X P_{eq}(X)M(X \to X')$$

- An ergodic update satisfying this *and* is a contration mapping leads to Markov transitions converge on the desired equilibrium.

(Clear pedagogical review: Anthony Kennedy Nara lectures 2006)

Metropolis algorithms

- Detailed balance property:

$$P_{eq}(X)M(X \to X') = P_{eq}(X')M(X' \to X)$$

- Sum over X to obtain

$$\sum_X P_{eq}(X)M(X \to X') = \sum_X P_{eq}(X')M(X' \to X) = P_{eq}(X')$$

- So P_{eq} is a *fixed point* of the Markov process!
- We can sample any probability distribution we desire with such an update.

Metropolis algorithms

- Make the update combine proposal and acceptance probablities

$$M(X \to X') = P_p(X \to X')P_{acc}(X \to X')$$

- detailed balance

$$P_{eq}(X)P_p(X \to X')P_{acc}(X \to X') = P_{eq}(X')P_p(X' \to X)P_{acc}(X' \to X),$$

- is satisfied with the Metropolis acceptance probability,

$$P_{acc}(X \to X') = \mathrm{Min}(1, \frac{P_{eq}(X')P_p(X' \to X)}{P_{eq}(X)P_p(X \to X')})$$

- either $P_{acc}(X \to X') = 1$, or $P_{acc}(X' \to X)$; considering cases leads to trivial proof.
- Simplifies if $P_p(X' \to X) = P_p(X \to X')$ (reversible, area preserving constraint)

Basis of most Markov Chain Monte Carlo

QCD path integral

- Partition function becomes a real, statistical mechanical probability weight

$$Z = \int d\bar{\psi} d\psi dU e^{-S_G[U] - S_F[\bar{\psi}, \psi, U]}$$

- Dirac differential operator represented via discrete derivative approximations: sparse matrix
- Use pseudofermion approach to replace with Gaussian integral $\sqrt{\pi\lambda} = \int dt e^{-t^2/\lambda}$

$$\int \mathcal{D}\bar{\psi} \mathcal{D}\psi e^{-\bar{\psi}(x) A_{xy} \psi(y)} = \det A$$

$$\pi\lambda = \int d\phi_r e^{-\phi_r \frac{1}{\lambda} \phi_r} \int d\phi_i e^{-\phi_i \frac{1}{\lambda} \phi_i} = \int d\phi^* d\phi e^{-\phi^* \frac{1}{\lambda} \phi}$$

- replace two flavour determinant with a two flavour *pseudofermion* integral

$$(\det M)^2 = (\det \gamma_5 M)^2 = \det M^\dagger M = \int \mathcal{D}\phi^* \mathcal{D}\phi e^{-\phi^*(x)(M^\dagger M)^{-1} \phi(y)}$$

Hybrid Monte Carlo

- Auxiliary Gaussian integral over conjugate momentum field $\int d\pi e^{\frac{-\pi^2}{2}}$
  Lives in Lie algbra; serves only to move U round the group Manifold

$$\int d\pi \int d\phi \int dU \quad e^{-\frac{\pi^2}{2}} e^{-S_G[U]} e^{-\phi^*(M^\dagger M)^{-1}\phi}$$

- Outer Metropolis Monte Carlo algorithm
  - Draw momenta
  - Draw pseudofermion as gaussian $\eta = M^{-1}\phi$
  - Metropolis acceptance step
- Metropolis proposal includes inner molecular dynamics at constant Hamiltonian:

$$H = \frac{\pi^2}{2} + S_G[U] + \phi^*(M^\dagger M)^{-1}\phi$$

$$\dot{U} = i\pi U \qquad ; \qquad i\dot{\pi} = -(U\nabla_U S)_{TA}$$

- Must invert $M^\dagger M$ at each timestep of evolution in MD force

$$\delta(M^\dagger M)^{-1} = -(M^\dagger M)^{-1}[(\delta M^\dagger)M + M(\delta M)](M^\dagger M)^{-1}$$

Force terms & conventions

Derive from:

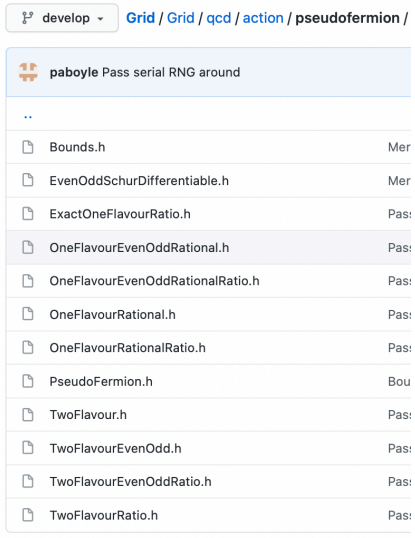
$$\dot{H} = 0 = \pi \left[\dot{\pi} + iU \cdot \nabla_U S_{TA} \right]$$

- Define *force* as the rate of change of momentum and is calculated by differentiating action with respect to each element of each link (and multiplying by U_μ ; leaving untraced).
- Grid "derivative" functions have some notable conventions. These simplify code but make somewhat non-obvious
- All code bases I have read have such quirks(!)
  1. $p_\mu = ip^a \tau^a$, removes factor of i in exponentiation
  2. Real actions (which we need for importance sampling) depend on U and $U^\dagger$ in a symmetric way.
  3. "really" should calculate dS/dU and $dS/dU^\dagger$ but this is a factor of two.
  4. "actually" calculate dS/dU and take the traceless anti-hermitian part: leaves an implicit factor of two in force included in integrator.
- Finite timestep of U is performed in the Lie algebra: $U' = e^{i\pi\, dt} U$, keeping U on the group manifold
- Force terms reduce to simple application of the product and chain rule, bearing in mind the rules of matrix differentiation.
- Code is typically pretty well commented and self documenting

Pseudofermion options

https://github.com/paboyle/Grid/tree/develop/Grid/qcd/action/pseudofermion

- Recall $\det M = \det M_{pc} \det M_{ee}$
- Require symmetric positive definite Gaussian integral (*not* just positive $\det M$)
- Two degenerate flavours is easy: $\phi^\dagger M^\dagger M \phi$
- Odd flavours harder: take square root of $\phi^\dagger M^\dagger M \phi$ in HPD fashion
- Polynomial HMC (Chebyshev) and Rational HMC (c.f. num rep)
- *Require positivity of one flavour determinant*
  - Guaranteed for DWF with positive mass, but not for Wilson (expected for sufficiently large mass, exceptional configurations are light mass)
  - https://arxiv.org/abs/2003.13359 : *In the simulation of QCD with 2+1 flavors of Wilson fermions, the positivity of the fermion determinant is generally assumed. We present evidence that this assumption is in general not justified and discuss the consequences of this finding.*
  - For DWF exact one flavour algorithm also
- RHMC efficient due to *multi-shift solvers*



Integrators

- Integrators should be area preserving to keep phase space density same $\Rightarrow$ symplectic integrators
- Integrators can be nested (Sexton-Weingarten) to integrate different action fragments on different timescales
  - e.g. Gauge force is nearly free to evaluate
- Integrators can be nested (Sexton-Weingarten) to integrate different action fragments on different timescales
- Leapfrog, Omelyan, Force Gradient give different orders of integration error in δt
- Peak determinant force can be reduced using a series of *Hasenbusch determinant ratios*
- Tuning determinant factoring and timesteps is laborious and empirical
- Practical recommendation:
  - Factor determinants and place all on same timescale, use Force Gradient
  - Balance forces between pseudofermion factors
  - Increase fermion timestep until acceptance drops to 90%
  - Integrate gauge action on a sufficiently fine resolution that delta Hamiltonian is independent of gauge timestep

Pseudofermion code example

https://github.com/paboyle/Grid/blob/develop/Grid/qcd/action/pseudofermion/TwoFlavourRatio.h

```
FermionOperator<Impl> & NumOp;// the basic operator
FermionOperator<Impl> & DenOp;// the basic operator

OperatorFunction<FermionField> &DerivativeSolver;
OperatorFunction<FermionField> &ActionSolver;
OperatorFunction<FermionField> &HeatbathSolver;

FermionField PhiOdd;   // the pseudo fermion field for this trajectory
FermionField PhiEven;  // the pseudo fermion field for this trajectory
```

Pseudofermion code example

https://github.com/paboyle/Grid/blob/develop/Grid/qcd/action/pseudofermion/TwoFlavourRatio.h

```
////////////////////////////////////////////////////
// S = phi^dag V (Mdag M)^-1 Vdag phi
////////////////////////////////////////////////////
virtual RealD S(const GaugeField &U) {

  NumOp.ImportGauge(U);
  DenOp.ImportGauge(U);

  FermionField X(NumOp.FermionGrid());
  FermionField Y(NumOp.FermionGrid());

  MdagMLinearOperator<FermionOperator<Impl> ,FermionField> MdagMOp(DenOp);

  NumOp.Mdag(Phi,Y);              // Y= Vdag phi
  X=Zero();
  ActionSolver(MdagMOp,Y,X);      // X= (MdagM)^-1 Vdag phi
  DenOp.M(X,Y);                  // Y=  Mdag^-1 Vdag phi

  RealD action = norm2(Y);

  return action;
};
```

Pseudofermion code example

https://github.com/paboyle/Grid/blob/develop/Grid/qcd/action/pseudofermion/TwoFlavourRatio.h

```
////////////////////////////////////////////////////
// dS/du = phi^dag dV (Mdag M)^-1 V^dag  phi
//       - phi^dag V (Mdag M)^-1 [ Mdag dM + dMdag M ]  (Mdag M)^-1 V^dag  phi
//       + phi^dag V (Mdag M)^-1 dV^dag  phi
////////////////////////////////////////////////////
virtual void deriv(const GaugeField &U,GaugeField & dSdU) {

  NumOp.ImportGauge(U);
  DenOp.ImportGauge(U);

  MdagMLinearOperator<FermionOperator<Impl> ,FermionField> MdagMOp(DenOp);

  FermionField  X(NumOp.FermionGrid());
  FermionField  Y(NumOp.FermionGrid());
  GaugeField   force(NumOp.GaugeGrid());

  //Y=Vdag phi
  //X = (Mdag M)^-1 V^dag phi
  //Y = (Mdag)^-1 V^dag  phi
  NumOp.Mdag(Phi,Y);              // Y= Vdag phi
  X=Zero();
  DerivativeSolver(MdagMOp,Y,X);      // X= (MdagM)^-1 Vdag phi
  DenOp.M(X,Y);                  // Y=  Mdag^-1 Vdag phi

  // phi^dag V (Mdag M)^-1 dV^dag  phi
  NumOp.MDeriv(force , X, Phi, DaggerYes );  dSdU=force;

  // phi^dag dV (Mdag M)^-1 V^dag  phi
  NumOp.MDeriv(force , Phi, X ,DaggerNo  );  dSdU=dSdU+force;

  //    -    phi^dag V (Mdag M)^-1 Mdag dM   (Mdag M)^-1 V^dag  phi
  //    -    phi^dag V (Mdag M)^-1 dMdag M   (Mdag M)^-1 V^dag  phi
  DenOp.MDeriv(force,Y,X,DaggerNo);   dSdU=dSdU-force;
  DenOp.MDeriv(force,X,Y,DaggerYes);  dSdU=dSdU-force;

  dSdU *= -1.0;
};
```

Pseudofermion code example

https://github.com/paboyle/Grid/blob/develop/Grid/qcd/action/pseudofermion/TwoFlavourRatio.h

```
virtual void refresh(const GaugeField &U, GridSerialRNG &sRNG, GridParallelRNG& pRNG) {

  // P(phi) = e^{- phi^dag V (MdagM)^-1 Vdag phi}
  //
  // NumOp == V
  // DenOp == M
  //
  // Take phi = Vdag^{-1} Mdag eta  ; eta = Mdag^{-1} Vdag Phi
  //
  // P(eta) = e^{- eta^dag eta}
  //
  // e^{x^2/2 sig^2} => sig^2 = 0.5.
  //
  // So eta should be of width sig = 1/sqrt(2) and must multiply by 0.707....
  //
  RealD scale = std::sqrt(0.5);

  FermionField eta(NumOp.FermionGrid());
  FermionField tmp(NumOp.FermionGrid());

  gaussian(pRNG,eta);

  NumOp.ImportGauge(U);
  DenOp.ImportGauge(U);

  // Note: this hard codes normal equations type solvers; alternate implementation needed for
  // non-herm style solvers.
  MdagMLinearOperator<FermionOperator<Impl> ,FermionField> MdagMOp(NumOp);

  DenOp.Mdag(eta,Phi);            // Mdag eta
  tmp = Zero();
  ActionSolver(MdagMOp,Phi,tmp);  // (VdagV)^-1 Mdag eta = V^-1 Vdag^-1 Mdag eta
  NumOp.M(tmp,Phi);               // Vdag^-1 Mdag eta

  Phi=Phi*scale;

};
```

Even Odd preconditioning

- We can write the Fermion determinant in terms of the red-black preconditioned operator

$$M = \begin{pmatrix} M_{ee} & M_{eo} \\ M_{oe} & M_{oo} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ M_{oe}M_{ee}^{-1} & 1 \end{pmatrix} \begin{pmatrix} M_{ee} & 0 \\ 0 & M_{oo} - M_{oe}M_{ee}^{-1}M_{eo} \end{pmatrix} \begin{pmatrix} 1 & M_{ee}^{-1}M_{eo} \\ 0 & 1 \end{pmatrix}$$

- where the Schur complement, is written as $M_{pc} = M_{oo} - M_{oe}M_{ee}^{-1}M_{eo}$.
- $\det U = \det M = 1$ and so,

$$\det M = \det M_p c \det M_{ee}$$

- Since the inverse of $M_{pc}^{\dagger}M_{pc}$ arises from a conjugate gradient solve (making the matrix HPD), and $M_{pc}^{-dagger} = M_{pc}(M_{pc}^{\dagger}M)^{-1}$, we can use both red-black solvers and a single solve in the force evaluation by using the Schur factored determinant in HMC.

Observables

Importance sampling has reduced:

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int_U e^{-S_G[U]} \mathcal{O}(U) dU \rightarrow \frac{1}{N} \sum_i \mathcal{O}(U_i)$$

- Zero momentum pion, kaon or B meson two point function

$$\sum_x \langle \bar{u} \gamma_0 \gamma_5 d(x,t) \bar{d} \gamma_0 \gamma_5 u(0,0) \rangle = \frac{1}{N} \sum_i \mathrm{trace}\{ \gamma_0 \gamma_5 M_d^{-1}(x,t;0,0) \gamma_0 \gamma_5 M_u^{-1}(0,0;x,t) \}$$

- Euclidean space $\propto Ae^{-mt}$
- Tune bare mass until interacting meson mass is correct, prefactor gives pion, kaon, B meson decay constant
- etc..

Exercise

- Quenched simulation for Gauge action of your choice
- Create a corresponding Grid HMC: cut down the Mobius DWF HMC as required
- Run it on your laptop on 8^4 and check you get close to the same plaquette as literature results
  - e.g. https://arxiv.org/pdf/hep-lat/0610075.pdf
- Check the Creutz relation: $< e^{-dH} >= 1$
- Save the gauge configurations for the next lecture(!)

Examples:
https://github.com/paboyle/Grid/blob/develop/HMC/Mobius2p1fRHMC.cc
https://github.com/paboyle/Grid/tree/develop/tests/hmc

Resources

- Kennedy, Pendleton, Roweth, Duane HMC
- Sexton-Weingarten
- Hasenbusch determinant factoring
- Kennedy, Clark RHMC

Topics not covered:

- Link smearing
- Exact one flavor algorithm (DWF)
- DDHMC
- Multigrid setup in HMC and polynomial subspace prediction
- Multishift solvers
- Polynomial HMC