ALGORITHMS FOR LATTICE QCD

Kostas Orginos
William and Mary / JLab

OUTLINE

- Lattice formulation of QCD
- Computation
  - Configuration generation
    - Hybrid Monte Carlo
      - Reweighting
  - Correlation functions
    - Linear Solvers
- Outlook

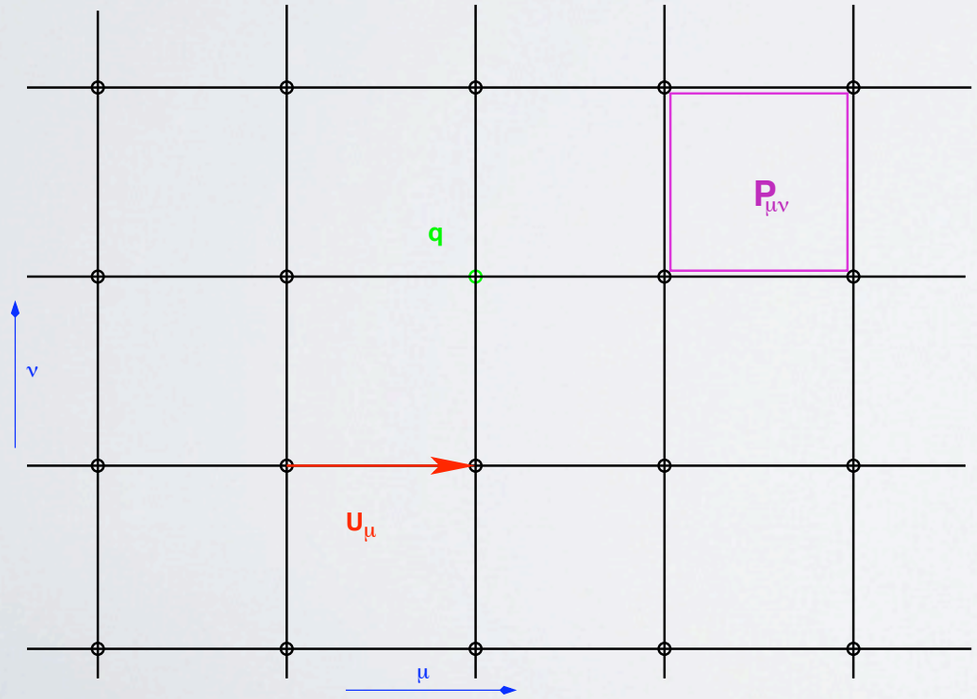
LATTICE QCD

In continuous Euclidian space:

$$\mathcal{Z} = \int \mathcal{D}q\mathcal{D}\bar{q}\mathcal{D}A_\mu \, e^{-S[\bar{q},q,A_\mu]}$$

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}q\mathcal{D}\bar{q}\mathcal{D}A_\mu \, \mathcal{O}(\bar{q},q,A_\mu) \, e^{-S[\bar{q},q,A_\mu]}$$

Lattice regulator:



Gauge sector:

$$U_\mu(x) = e^{-iaA_\mu(x+\frac{\hat{\mu}}{2})}$$

Fermion sector:

Things get nasty!

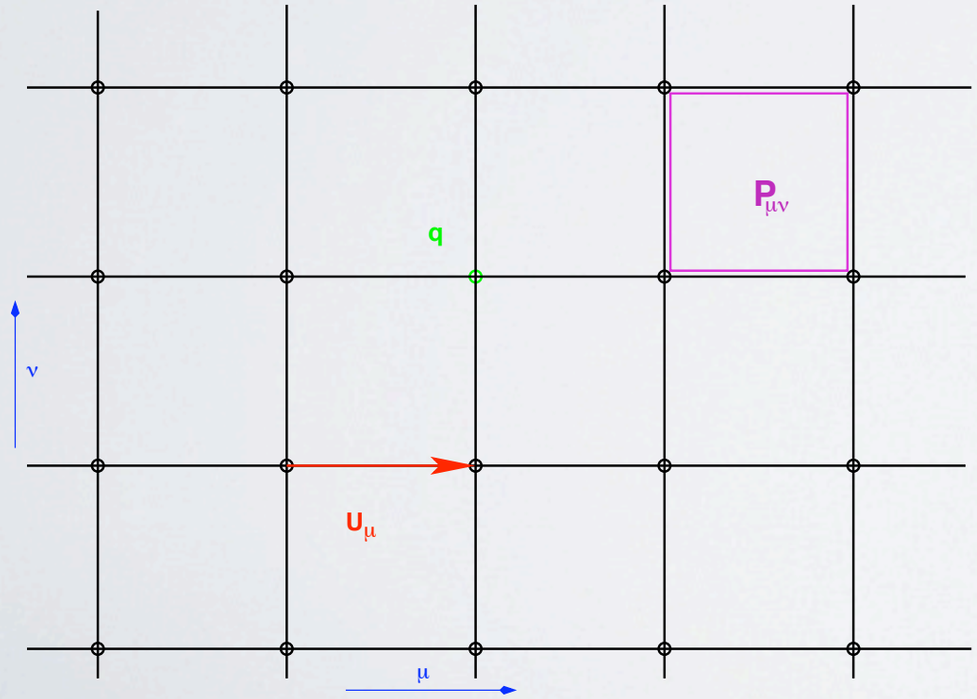
LATTICE QCD

In continuous Euclidian space:

$$\mathcal{Z} = \int \mathcal{D}q\mathcal{D}\bar{q}\mathcal{D}A_\mu \, e^{-S[\bar{q},q,A_\mu]}$$

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}q\mathcal{D}\bar{q}\mathcal{D}A_\mu \, \mathcal{O}(\bar{q},q,A_\mu) \, e^{-S[\bar{q},q,A_\mu]}$$

Lattice regulator:



Gauge sector:

$$U_\mu(x) = e^{-iaA_\mu(x+\frac{\hat{\mu}}{2})}$$

Fermion sector:

Things get nasty!

Fermion doubling

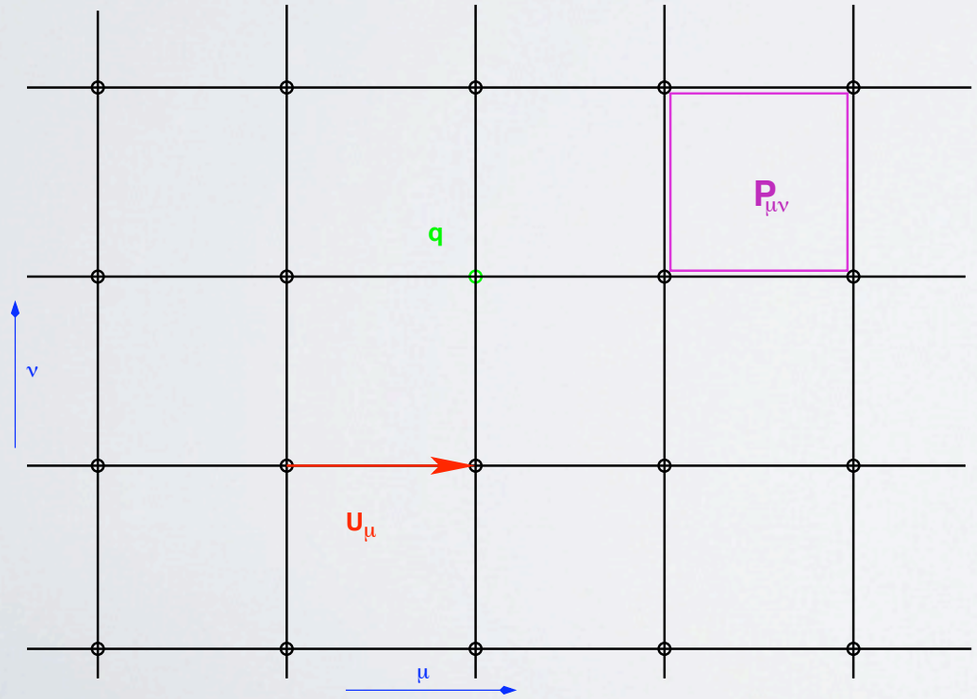
LATTICE QCD

In continuous Euclidian space:

$$\mathcal{Z} = \int \mathcal{D}q\mathcal{D}\bar{q}\mathcal{D}A_\mu \, e^{-S[\bar{q},q,A_\mu]}$$

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}q\mathcal{D}\bar{q}\mathcal{D}A_\mu \, \mathcal{O}(\bar{q},q,A_\mu) \, e^{-S[\bar{q},q,A_\mu]}$$

Lattice regulator:



Gauge sector:

$$U_\mu(x) = e^{-iaA_\mu(x+\frac{\hat{\mu}}{2})}$$

Fermion sector:

Things get nasty!

Fermion doubling

Chiral symmetry breaking

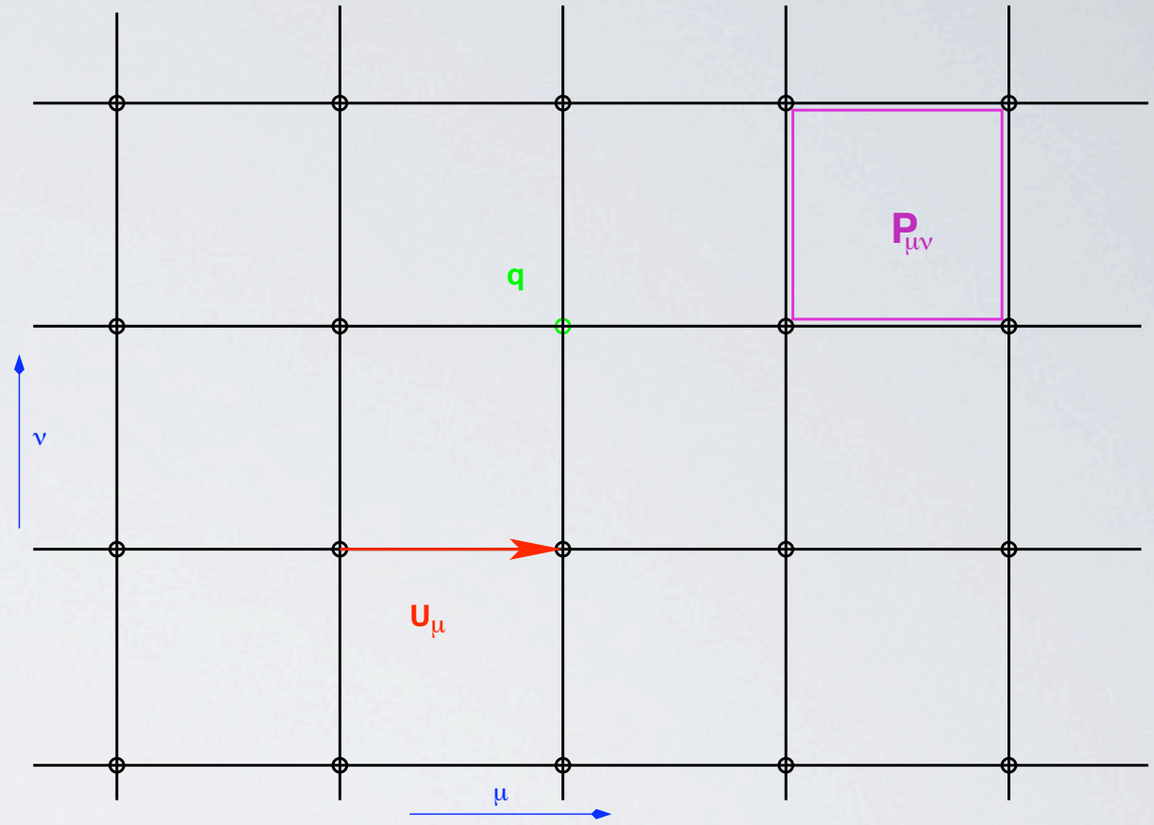
Gauge sector:

$$S_g(U) = \beta \sum_p \left(1 - \frac{1}{3}\mathrm{ReTr}U_p\right) \longrightarrow \frac{1}{4}F_{\mu\nu}^2$$

Fermion sector:

$$S_f(\bar{q}, q, U) = \bar{q}D(U)q$$

- D(U) sparse matrix
- Wilson fermions
- Kogut-Susskind fermions
- Domain Wall
- Overlap: Not a sparse matrix



$$\mathcal{Z} = \int \mathcal{D}[U]\ \mathcal{D}[\bar{\psi}]\mathcal{D}[\psi]\ e^{-\bar{\psi}D(U)\psi - S_g(U)}$$

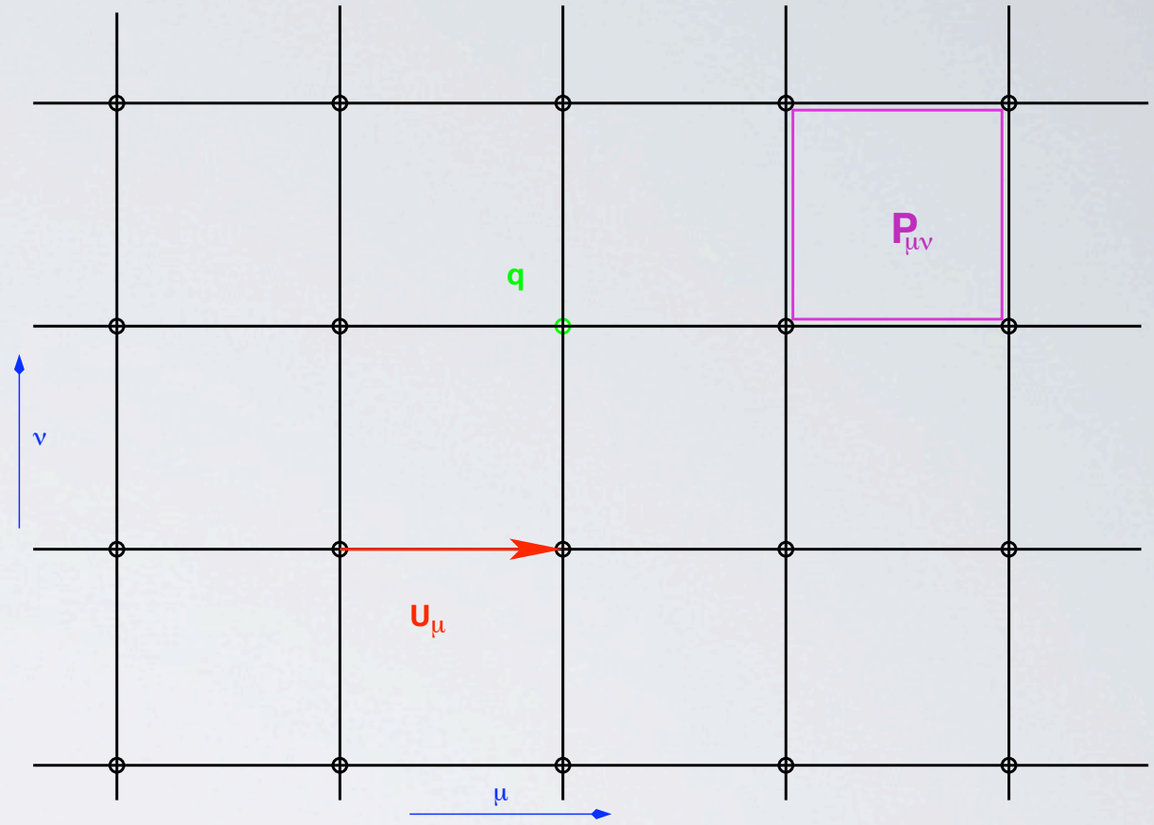
Gauge sector:

$$S_g(U) = \beta \sum_p \left(1 - \frac{1}{3}\mathrm{ReTr}U_p\right) \longrightarrow \frac{1}{4}F_{\mu\nu}^2$$

Fermion sector:

$$S_f(\bar{q}, q, U) = \bar{q}D(U)q$$

- D(U) sparse matrix
- Wilson fermions
- Kogut-Susskind fermions
- Domain Wall
- Overlap: Not a sparse matrix



$$\mathcal{Z} = \int \mathcal{D}[U] \ \det(D(U)) \ \ e^{-S_g(U)}$$

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \prod_{\mu,x} dU_\mu(x) \;\; \mathcal{O}[U, D(U)^{-1}] \;\; \det \left(D(U)^\dagger D(U) \right)^{n_f/2} \;\; e^{-S_g(U)}$$

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \prod_{\mu,x} dU_\mu(x) \;\; \mathcal{O}[U, D(U)^{-1}] \; \det\left(D(U)^\dagger D(U)\right)^{n_f/2} \;\; e^{-S_g(U)}$$

Monte Carlo integration
Hybrid Monte Carlo: No determinant evaluation

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \prod_{\mu,x} dU_\mu(x) \; \mathcal{O}[U, D(U)^{-1}] \; \det\left(D(U)^\dagger D(U)\right)^{n_f/2} \; e^{-S_g(U)}$$

Matrix Inversion: Iterative Solvers

$$D(U)\chi = \psi$$

Monte Carlo integration
Hybrid Monte Carlo: No determinant evaluation

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \prod_{\mu,x} dU_\mu(x) \; \mathcal{O}[U, D(U)^{-1}] \; \det\left(D(U)^\dagger D(U)\right)^{n_f/2} \; e^{-S_g(U)}$$

Matrix Inversion: Iterative Solvers

$$D(U)\chi = \psi$$

Monte Carlo integration
Hybrid Monte Carlo: No determinant evaluation

- Solution of linear system: significant CPU time
- HMC needs matrix inversions
  - Continuously changing U
  - Fixed right hand side ψ
- Correlation functions:
  - Fixed U
  - Large number of orthogonal right hand sides ψ

WHAT DOES IT TAKE?

- Hadronic scale
  - characteristic length ~ $\Lambda_{qcd} \sim 220\text{MeV}$ $\quad 1\text{fm} = 1 \times 10^{-13}\text{cm}$
- The lattice spacing $\quad a \ll 1\text{fm}$
- The lattice size $\quad La \gg 1\text{fm}$
- Reasonable choices $\quad a = .1\text{fm}, \ \ldots \ La = 3\text{fm}$
- Degrees of freedom

$$2 \times 3 \times 4 \times 32^4 = 2.5 \times 10^7$$

flavor, color, spin, space time

Algorithm scaling: $\quad \sim \frac{1}{a^7} \qquad \sim \frac{1}{m_q^{2.5}}$

HYBRID MONTE CARLO

Duane, Kennedy, Pendleton, Roweth, Phys. Lett. B195, 216 (1987)

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \prod_{\mu,x} dU_\mu(x) \;\; \mathcal{O}[U, D(U)^{-1}] \;\; \det\left(D(U)^\dagger D(U)\right)^{n_f/2} \;\; e^{-S_g(U)}$$

Generate gauge fields with probability:

$$P(U) = \det(D(U))^{n_f} e^{-S_g(U)}$$

Then expectation values become averages:

$$\langle \mathcal{O} \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}(U_i, \frac{1}{D(U_i)})$$

Need an update of U that:

- Detailed Balance
- Ergodicity
- Avoids the computation of the determinant

The two flavor case:

$$D(U)^\dagger = \gamma_5 D(U) \gamma_5 \longrightarrow \det(D(U))^2 = \det(D(U)^\dagger D(U))$$

Using bosonic fields:

$$\det(D(U)^\dagger D(U)) = \int d\phi^\dagger d\phi e^{-\phi^\dagger \frac{1}{D^\dagger(U) D(U)} \phi}$$

Add conjugate momenta to the gauge fields with gaussian action:

$$P_\mu(x) \leftrightarrow U_\mu(x) \qquad S_p = \frac{1}{2} \sum_{\mu,x} P_\mu(x)^2$$

Hamiltonian evolution: $\{P, U\} \leftrightarrow \{P', U'\}$

$$\mathcal{H} = \frac{1}{2} \sum_{\mu,x} P_\mu(x)^2 + S_g(U) + \phi^\dagger \frac{1}{D(U)^\dagger D(U)} \phi$$

In continuous fictitious evolution time:

$$\dot{U} = \frac{\partial \mathcal{H}}{\partial P} \qquad \dot{P} = -\frac{\partial \mathcal{H}}{\partial U}$$

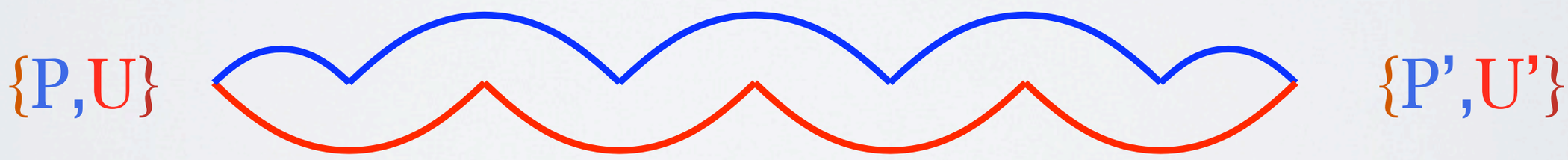
The algorithm satisfies:

- Detailed balance
- Ergodicity

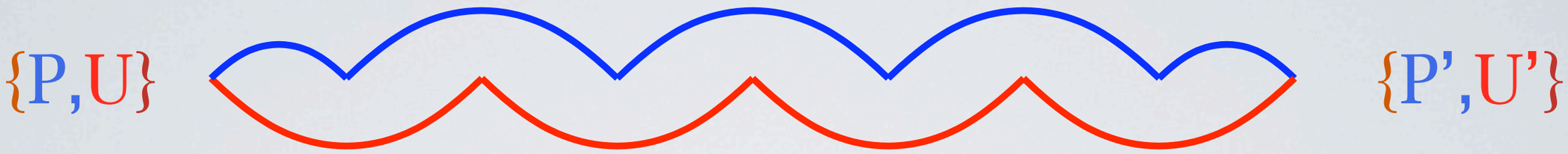
Need numerical reversible integration algorithm

Leapfrog Integrator

Omelyan Integrator [deForcrand and Takaishi Phys.Rev. E73 (2006) 036706]



Metropolis accept reject to correct energy violations

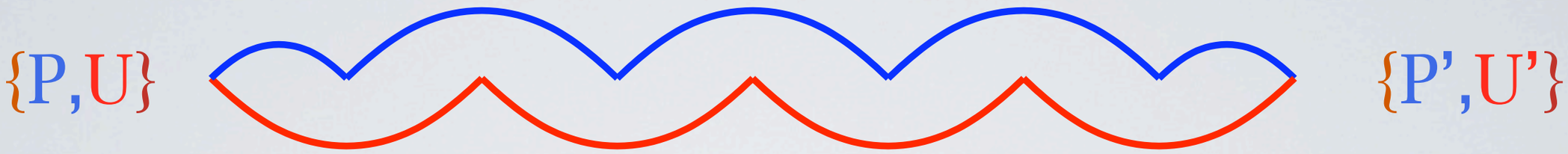


Large time step → Efficiency

Large Force → Small time step

Use multiple time steps. Isolate sources of large force and evolve them at smaller time steps.

[J. C. Sexton and D. H. Weingarten, Nucl. Phys. B380, 665 (1992).]



Large time step → Efficiency

Large Force → Small time step

Use multiple time steps. Isolate sources of large force and evolve them at smaller time steps.

[J. C. Sexton and D. H. Weingarten, Nucl. Phys. B380, 665 (1992).]

Example: Gauge action generates larger force than the fermion action

MULTIPLE TIME STEPS

[J. C. Sexton and D. H. Weingarten, Nucl. Phys. B380, 665 (1992).]

- Split up the Hamiltonian

$$\mathcal{H} = \frac{1}{2}\sum_{x,\mu} P_\mu(x)^2 + S_1 + S_2$$

$$T_U(\epsilon): \quad U \longrightarrow \quad e^{i\epsilon P}U$$

$$T_P^1(\epsilon): \quad P \longrightarrow \quad P + \epsilon F_1 \qquad T_P^2(\epsilon): \quad P \longrightarrow \quad P + \epsilon F_2$$

- Two evolutions

$$T_1 = T_P^1(\epsilon_1/2)T_U(\epsilon_1)T_P^1(\epsilon_1/2)$$

$$T_2 = T_P^1(\epsilon_2/2)T_1^{N_1}(\epsilon_1)T_P^1(\epsilon_2/2)$$

- Full trajectory **τ**:

$$[T_2]^{N_2}$$

- Time steps fulfill: $\mathbf{N_2=\tau/\varepsilon_2}$ $\mathbf{N_1=\varepsilon_1/\varepsilon_2}$

MULTIPLE TIME STEPS

[J. C. Sexton and D. H. Weingarten, Nucl. Phys. B380, 665 (1992).]

- Split up the Hamiltonian

$$\mathcal{H} = \frac{1}{2}\sum_{x,\mu} P_\mu(x)^2 + S_1 + S_2$$

$$T_U(\epsilon): \quad U \longrightarrow \quad e^{i\epsilon P}U$$

$$T_P^1(\epsilon): \quad P \longrightarrow \quad P + \epsilon F_1 \qquad T_P^2(\epsilon): \quad P \longrightarrow \quad P + \epsilon F_2$$

- Two evolutions

$$T_1 = T_P^1(\epsilon_1/2)T_U(\epsilon_1)T_P^1(\epsilon_1/2)$$

$$T_2 = T_P^1(\epsilon_2/2)T_1^{N_1}(\epsilon_1)T_P^1(\epsilon_2/2)$$

- Full trajectory **τ**:

$$[T_2]^{N_2}$$

- Time steps fulfill: $\mathbf{N_2=\tau/\varepsilon_2}$ $\mathbf{N_1=\varepsilon_1/\varepsilon_2}$

MULTIPLE TIME STEPS

[J. C. Sexton and D. H. Weingarten, Nucl. Phys. B380, 665 (1992).]

- Split up the Hamiltonian

$$\mathcal{H} = \frac{1}{2}\sum_{x,\mu} P_\mu(x)^2 + S_1 + S_2$$

$$T_U(\epsilon): \quad U \longrightarrow \quad e^{i\epsilon P}U$$

$$T_P^1(\epsilon): \quad P \longrightarrow \quad P + \epsilon F_1 \qquad T_P^2(\epsilon): \quad P \longrightarrow \quad P + \epsilon F_2$$

- Two evolutions

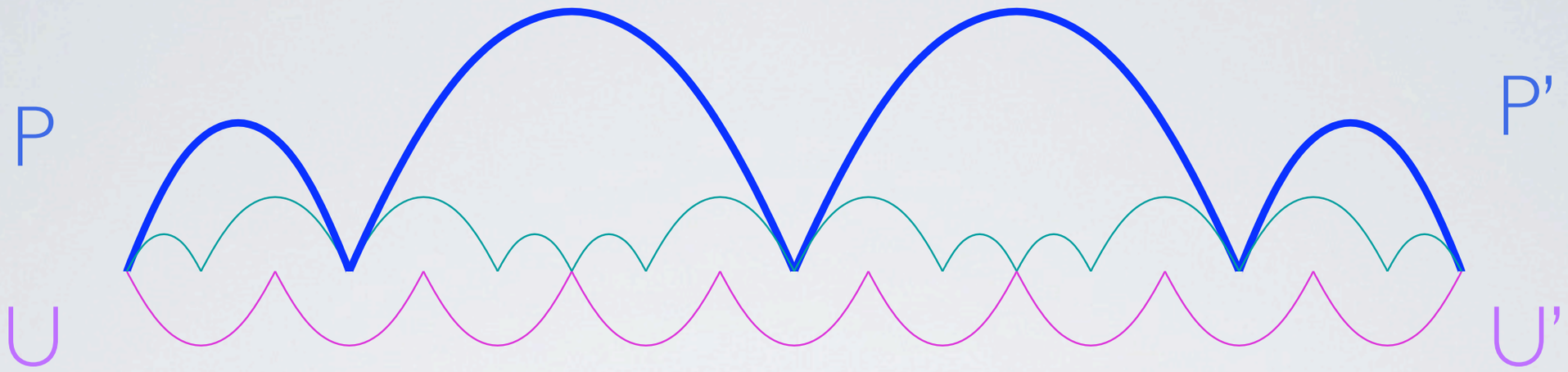
$$T_1 = T_P^1(\epsilon_1/2)T_U(\epsilon_1)T_P^1(\epsilon_1/2)$$

$$T_2 = T_P^1(\epsilon_2/2)T_1^{N_1}(\epsilon_1)T_P^1(\epsilon_2/2)$$

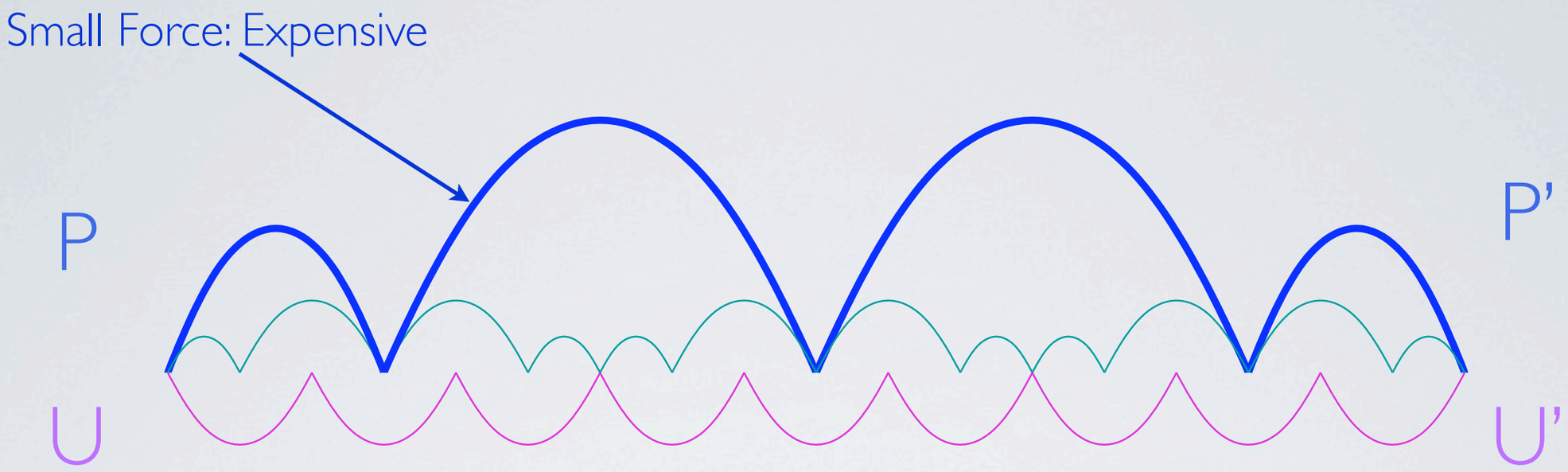
- Full trajectory **τ**:

$$[T_2]^{N_2}$$

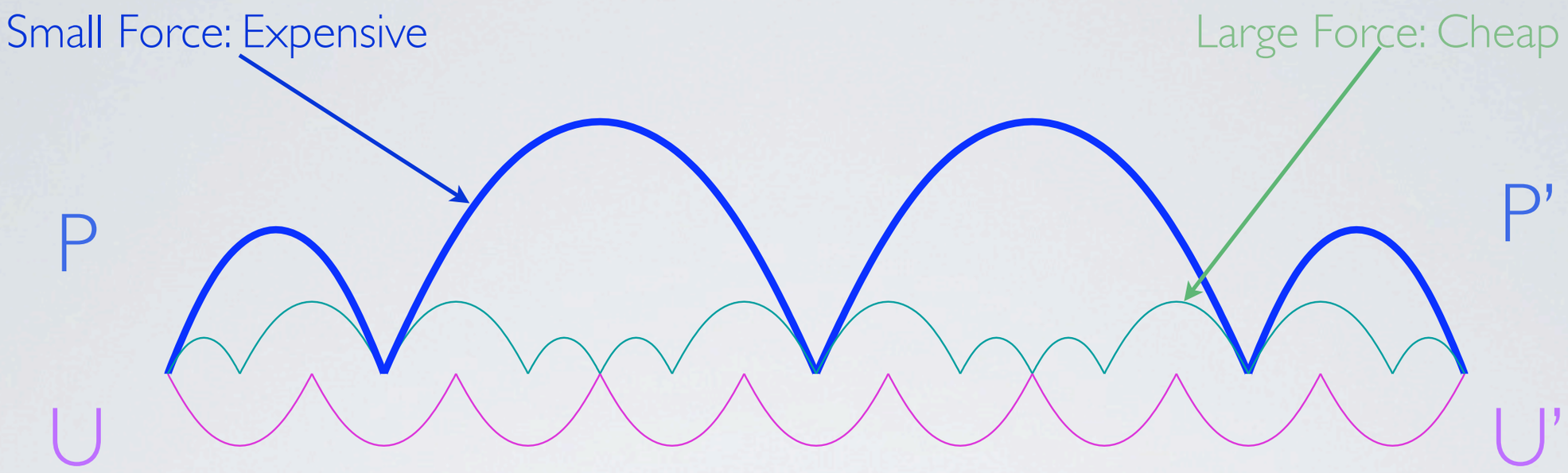
- Time steps fulfill: $\mathbf{N_2=\tau/\varepsilon_2}$ $\mathbf{N_1=\varepsilon_1/\varepsilon_2}$



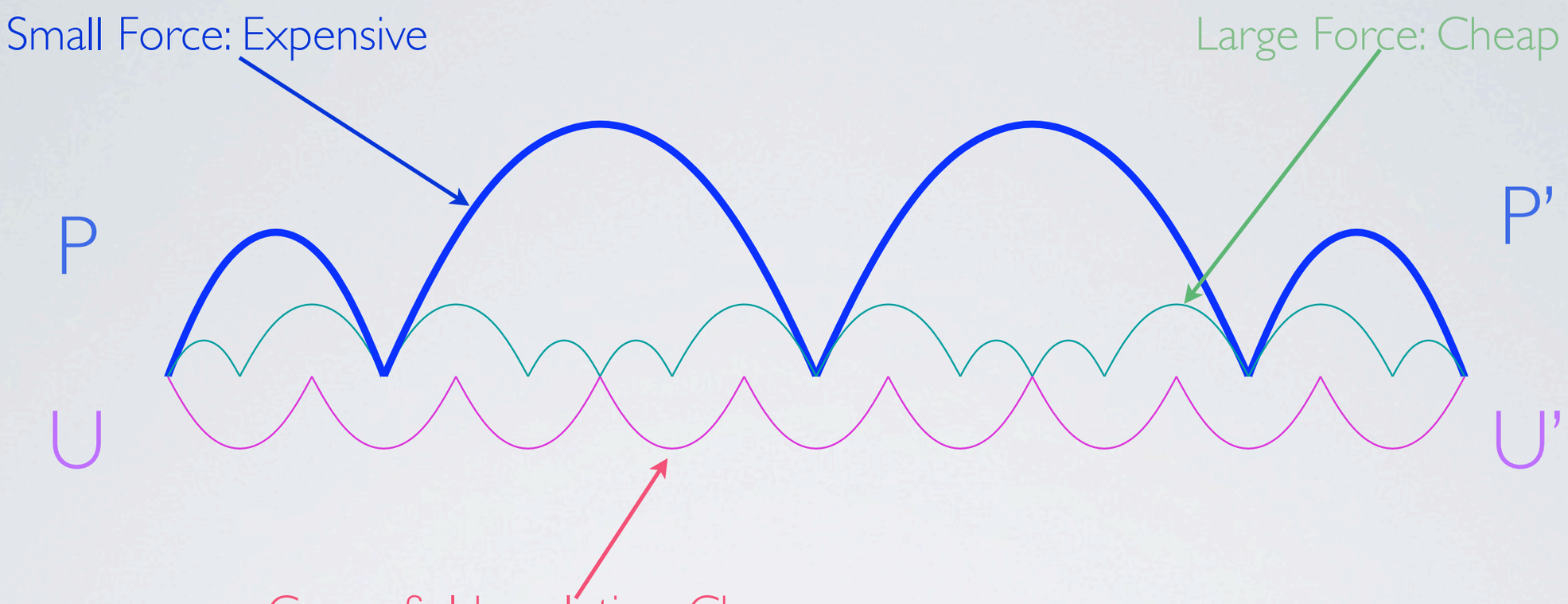
- This allows fast evolution
- Small energy violation
- Large acceptance



- This allows fast evolution
- Small energy violation
- Large acceptance



- This allows fast evolution
- Small energy violation
- Large acceptance



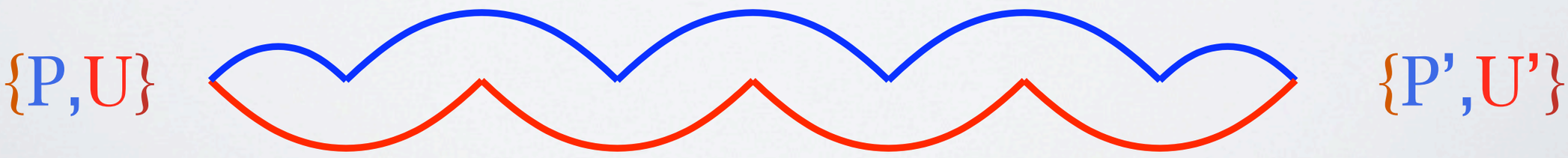
- This allows fast evolution
- Small energy violation
- Large acceptance

THE FERMION FORCE

Most challenging

Need to solve: $$\chi = \frac{1}{D^\dagger(U)D(U)}\phi$$

- Harder as the quark mass gets smaller
- Fermion force dominates at small quark masses
- Chronological inversion [Brower, Ivanenko, Levi, KO Nucl.Phys. B484 (1997)]



PRECONDITIONED HMC

Idea: Split up the fermion force

$$\det\left(D(U)^\dagger D(U)\right) = \det(M(U)^\dagger M(U))\det\left(\frac{1}{M(U)^\dagger} D(U)^\dagger D(U) \frac{1}{M(U)}\right)$$

- Use two boson fields (pseudo-fermions)
- *M(U)* Preconditioner that generates cheap but large force
- The correction term the gives small force
- Preconditioner need not be good solver preconditioner

PRECONDITIONED HMC

- UV spectrum of D(U): Large force

- IR spectrum of D(U): Small force

HEAVY MASS PRECONDITIONING

[M. Hasenbusch Phys.Lett. B519 (2001) 177-182]

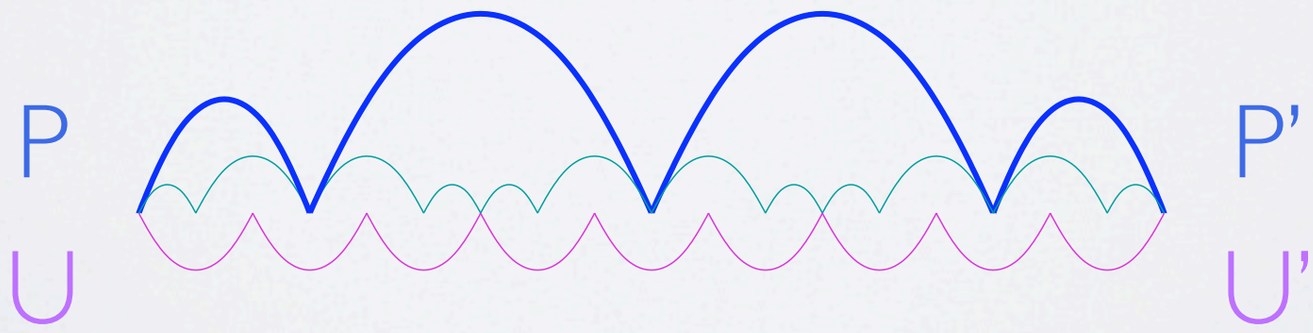
$$\det\left(D(U)^\dagger D(U)\right) = \det(M(U)^\dagger M(U))\det\left(\frac{1}{M(U)^\dagger} D(U)^\dagger D(U) \frac{1}{M(U)}\right)$$

$$\chi' = \frac{1}{M^\dagger(U)[m_h]M(U)[m_h]}\phi'$$

Cheap Large force: small time step

$$\chi = M^\dagger(U)[m_h]\frac{1}{D^\dagger(U)[m_l]D(U)[m_l]}M(U)[m_h]\phi$$

Expensive small force: Large time step



POLYNOMIAL FILTERING

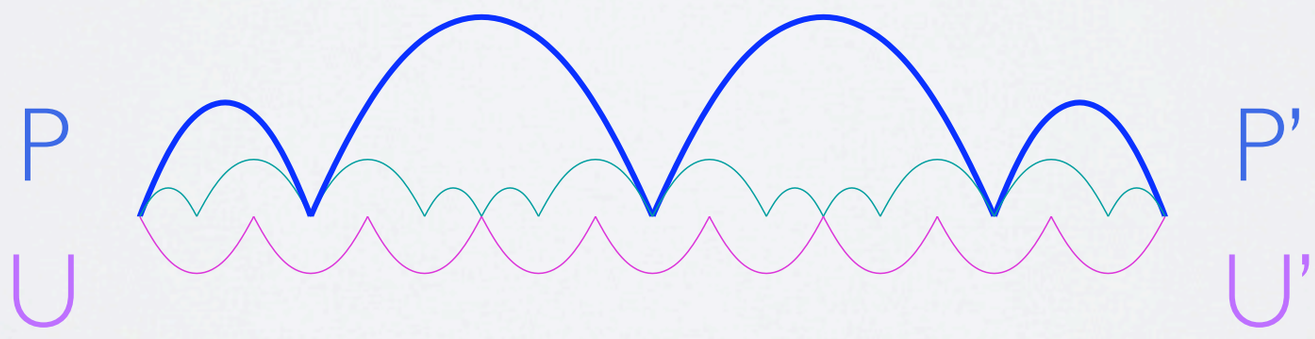
M. Peardon J. Sexton LATTICE 2002 hep-lat/0209037
W. Kamleh M. Peardon POS(LAT2005)106

Use Chebyshev polynomial approximation to the UV spectrum

$$M(U)^{-1} = P(\lambda, D^\dagger D)$$

Approximation good in [λ, 1] λ~.3

Polynomium degree is small (n ~ 16)
Most of the Fermion force comes from this limited part of the spectrum
Most eigenvalues are in this range!
Force calculation is cheap (No matrix inversion needed)



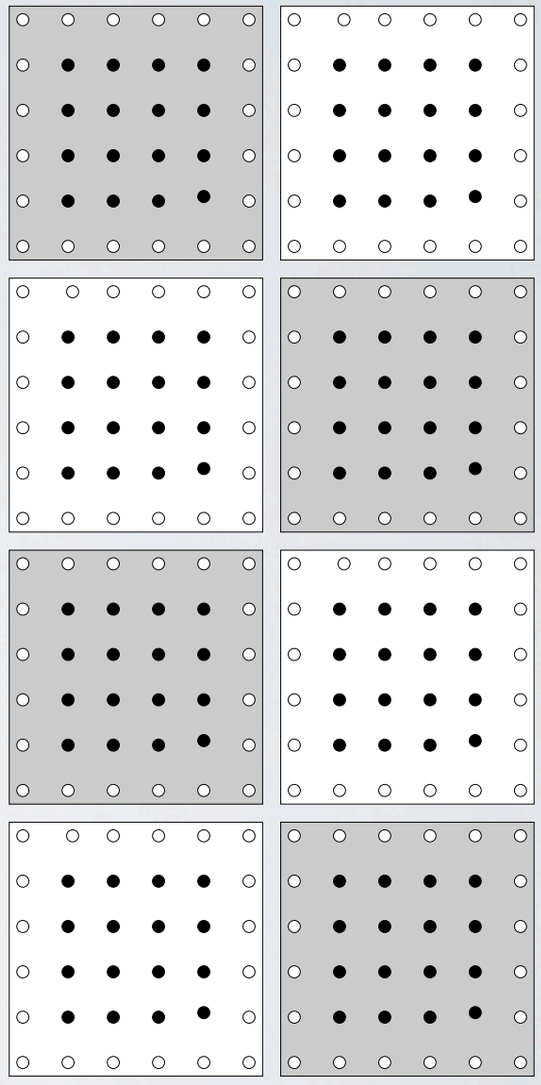
Golub Ruiz Touhmi 2005: Use this preconditioner for multiple right hand sides

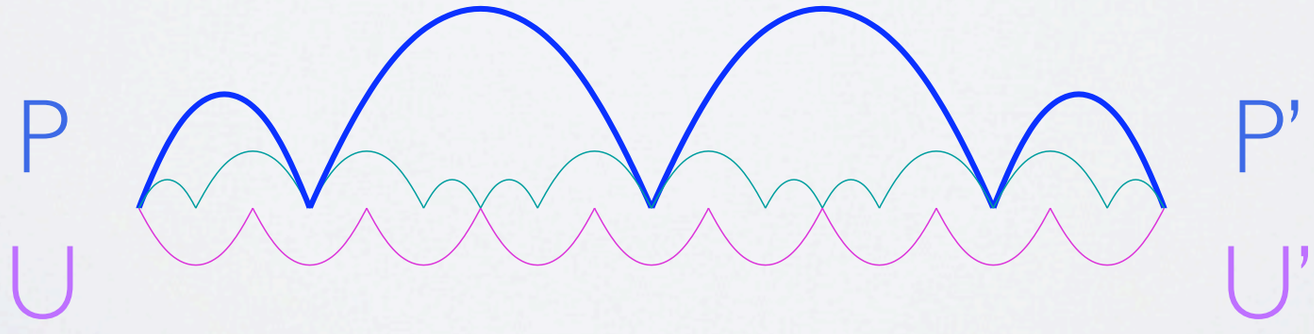
Schwarz-Preconditioner

DDHMC

[M. Luscher Comput.Phys.Commun. 165 (2005) 199–220]

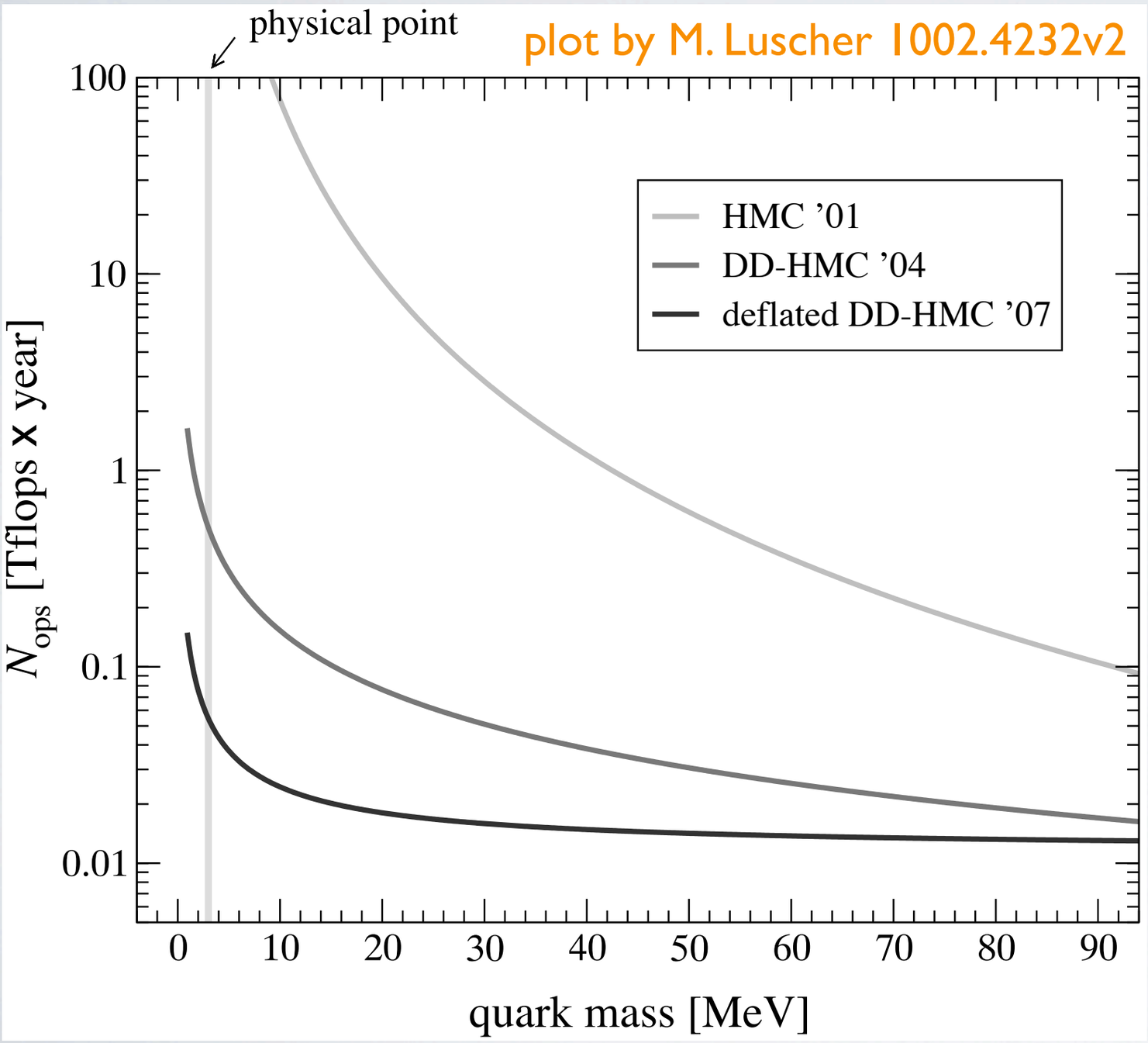
- M(U): No links to neighboring Blocks
- M(U): UV physics
- Correction term needs noise on the surface only
- Correction term: IR physics
- Factor of ~10 speed up at small quark masses





DEFLATED-DD-HMC

[Luscher 0710.5417v1]



- Uses an GCR with an AMG preconditioner
  - [Luscher 0706.2298v4]
- Chronology reduces the refresh of the preconditioner
- Nearly removes critical slowing down
- Lattice $32^3 \times 64$ a = 0.08fm

RATIONAL HMC

[Clark and Kennedy]

$$\det(D(U)^\dagger D(U)) = \det(R(D(U)^\dagger D(U))^{-2})$$

$$R(x) = \frac{P_n(x)}{Q_m(x)} = \sum_{k=1}^{m} \frac{a_k}{x + b_k} \rightarrow \frac{1}{x^{1/2}}$$

- Very accurate
- Generated using Remez algorithm
- Multi shift solver (Real roots $b_k > 0$)
- Can be used to work with odd number of flavors (strange quark)

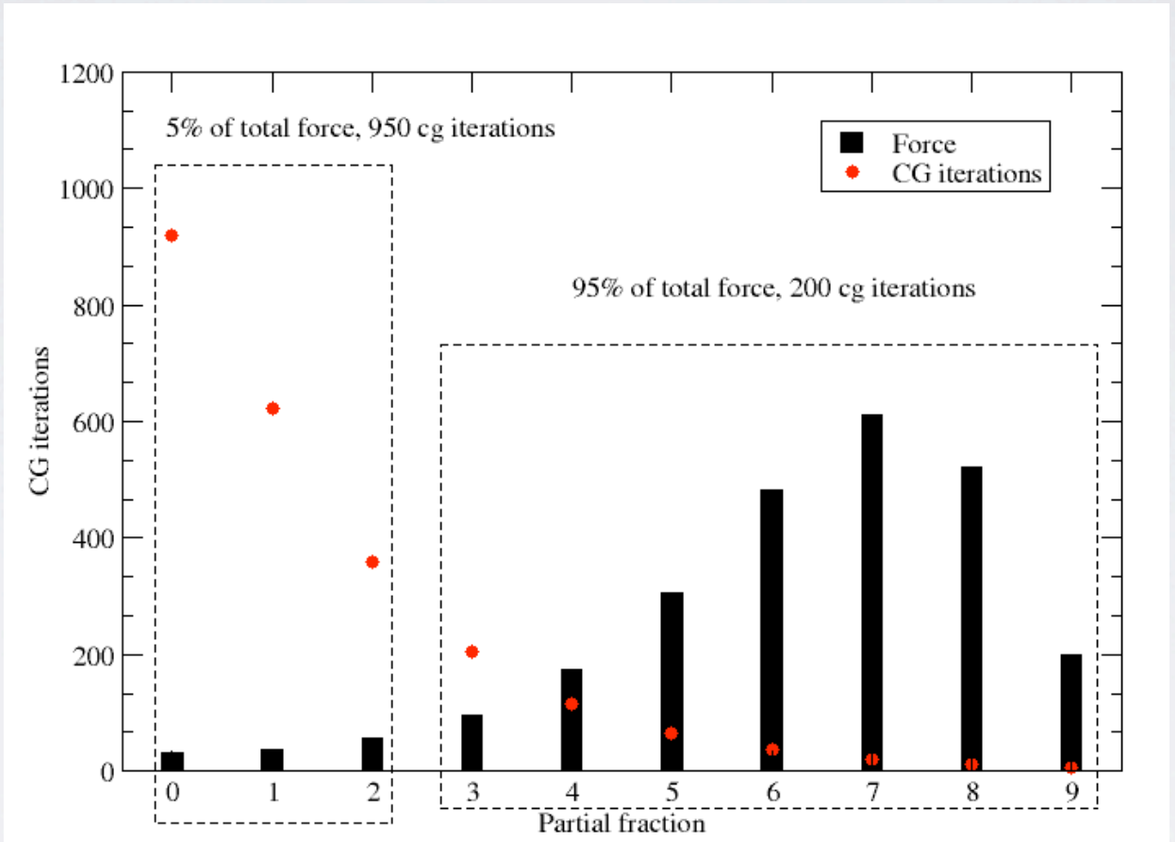
RHMC AND NTH ROOT TRICK

$$\det(D(U)^\dagger D(U)) = \det([D(U)^\dagger D(U)]^{1/n})^n$$

- Use different pseudo fermions for each factor
- Force 1/n the original
- Evolution can go faster

RHMC MULTI TIME SCALE

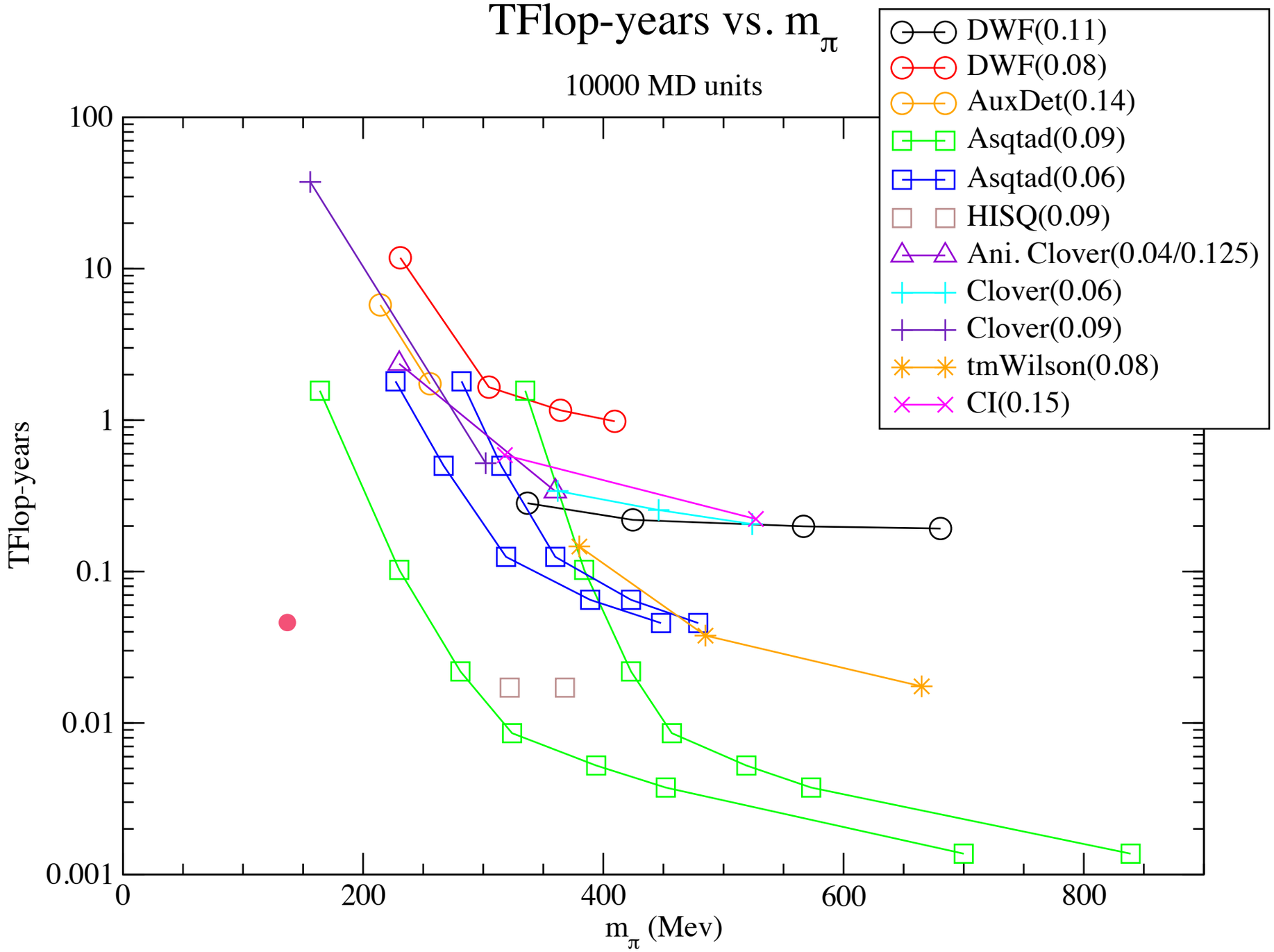
[Clark, deForcrand, Kennedy PoS LAT2005 (2005) 115]



* Use different time scale small and large roots.
* Force 1/n the original

WORLD HMC PERFORMANCE

figure by C. Jung arXiv:001.0941v1



● 2 flavor Clover Defl. - DDHMC arXiv:1002.4232v2

ISOSPIN BREAKING

- Gauge field configurations are generated with the degenerate up and down quark masses
- In nature $m_{up} \sim 2$ MeV and $m_{down} \sim 5$ MeV
- Precision calculations will need to non-degenerate light quark masses
- Nuclear physics: Fine tuned
- We need an efficient way to do calculations to slightly vary parameters in the action

REWEIGHTING

- Reweighting is a method used to perform calculations using an ensemble that does not have the action parameters we want

- Gauge configurations are generated with $m_{up} = m_{down}$

- Observables with $m_{up} \neq m_{down}$ can be calculated

Starting ensemble

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}[U] \ \det(D^\dagger(U) D(U))^{N_f/2} \ \mathcal{O}(D(U)^{-1}, U) \ \ e^{-S_g(U)}$$

$$\mathcal{Z} = \int \mathcal{D}[U] \ \det(D(U) D^\dagger(U))^{N_f/2} \ \ e^{-S_g(U)}$$

Target ensemble

$$\langle \mathcal{O} \rangle' = \frac{1}{\mathcal{Z}'} \int \mathcal{D}[U] \ \det(D'^\dagger(U) D'(U))^{N_f/2} \ \mathcal{O}(D'(U)^{-1}, U) \ \ e^{-S_g(U)}$$

$$\mathcal{Z}' = \int \mathcal{D}[U] \ \det(D'(U) D'^\dagger(U))^{N_f/2} \ \ e^{-S_g(U)}$$

Modify the fermion action

$$\langle \mathcal{O} \rangle' = \frac{1}{\mathcal{Z}'} \int \mathcal{D}[U]\, e^{\frac{N_f}{2}\left[Tr\log(D'^{\dagger}(U)D'(U)) - Tr\log(D^{\dagger}(U)D(U))\right]}\, \mathcal{O}(D'(U)^{-1}, U)\; e^{-S(U)}$$

$$\mathcal{Z}' = \frac{1}{\mathcal{Z}} \int \mathcal{D}[U]\, e^{\frac{N_f}{2}\left[Tr\log(D'^{\dagger}(U)D'(U)) - Tr\log(D^{\dagger}(U)D(U))\right]}\; e^{-S(U)}$$

Computational task:

Evaluate the trace log of a sparse positive definite matrix

- Use pseudofermions just like HMC
  - Compute inverses of the Dirac Matrix
- Use Gaussian quadrature [Golub & Meurant '93; Bai, Fahey & Golub '96]
  - Lanczos iteration
  - Converges faster than solving a linear system (with ex. CG)

Hassenfratz et. al. arXiv:0805.2369 ;
RBC arXiv:1011.0892 ;
PACS-CS arXiv:0911.2561

GAUSSIAN QUADRATURE

[Golub & Meurant '93; Bai, Fahey & Golub '96]

$$Tr\log(A) \approx \frac{1}{N}\sum_{k=1}^{N} \eta_k^\dagger \log(A)\eta_k$$

η are vectors whose components are random Z_4 noise

Gaussian quadrature evaluates $\eta_k^\dagger \log(A)\eta_k$

$$\eta^\dagger f(A)\eta = \eta^\dagger Q^\dagger f(\Lambda) Q\eta = u^\dagger f(\Lambda) u = \sum_i u_i^* f(\lambda_i) u_i$$

With Q the eigenvector matrix and λ_i the eigenvalues of A

$$I[f] = \eta^\dagger f(A)\eta = \sum_i u_i^* f(\lambda_i) u_i = \int_a^b d\lambda \sum_{i=1}^n u_i^* u_i \delta(\lambda - \lambda_i) f(\lambda) = \int_a^b d\mu(\lambda) f(\lambda)$$

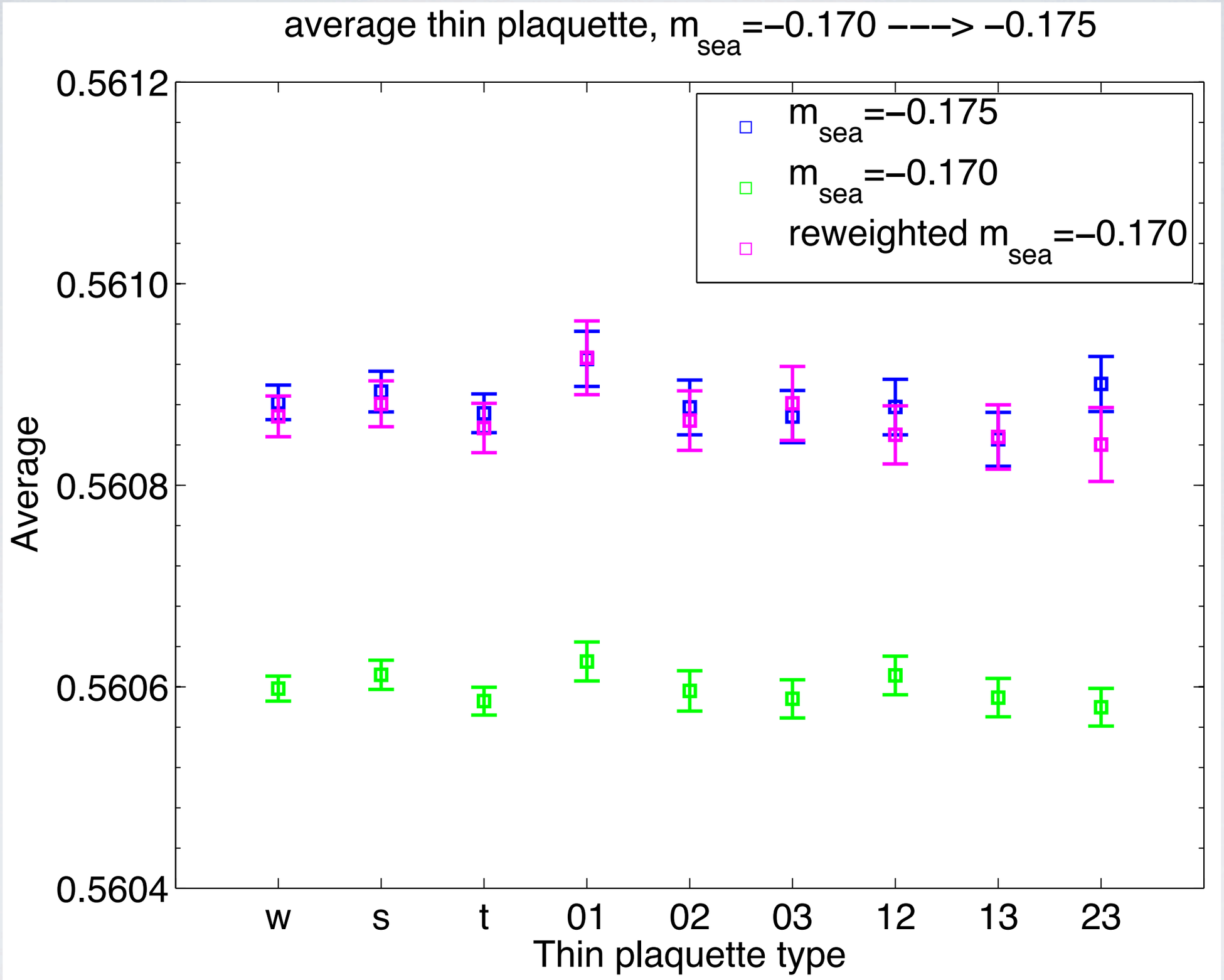
$$\mu(\lambda) = \begin{cases} 0, & \text{if} \quad \lambda < a = \lambda_1 \\ \sum_j^i u_j^* u_j, & \text{if} \quad \lambda_i \leq \lambda < \lambda_{i+1} \\ \sum_j^n u_j^* u_j, & \text{if} \quad b = \lambda_n \leq \lambda \end{cases}$$

To calculate the integral use Gaussian Quadrature integration with the orthogonal polynomial defined by the Lanczos recursion relation

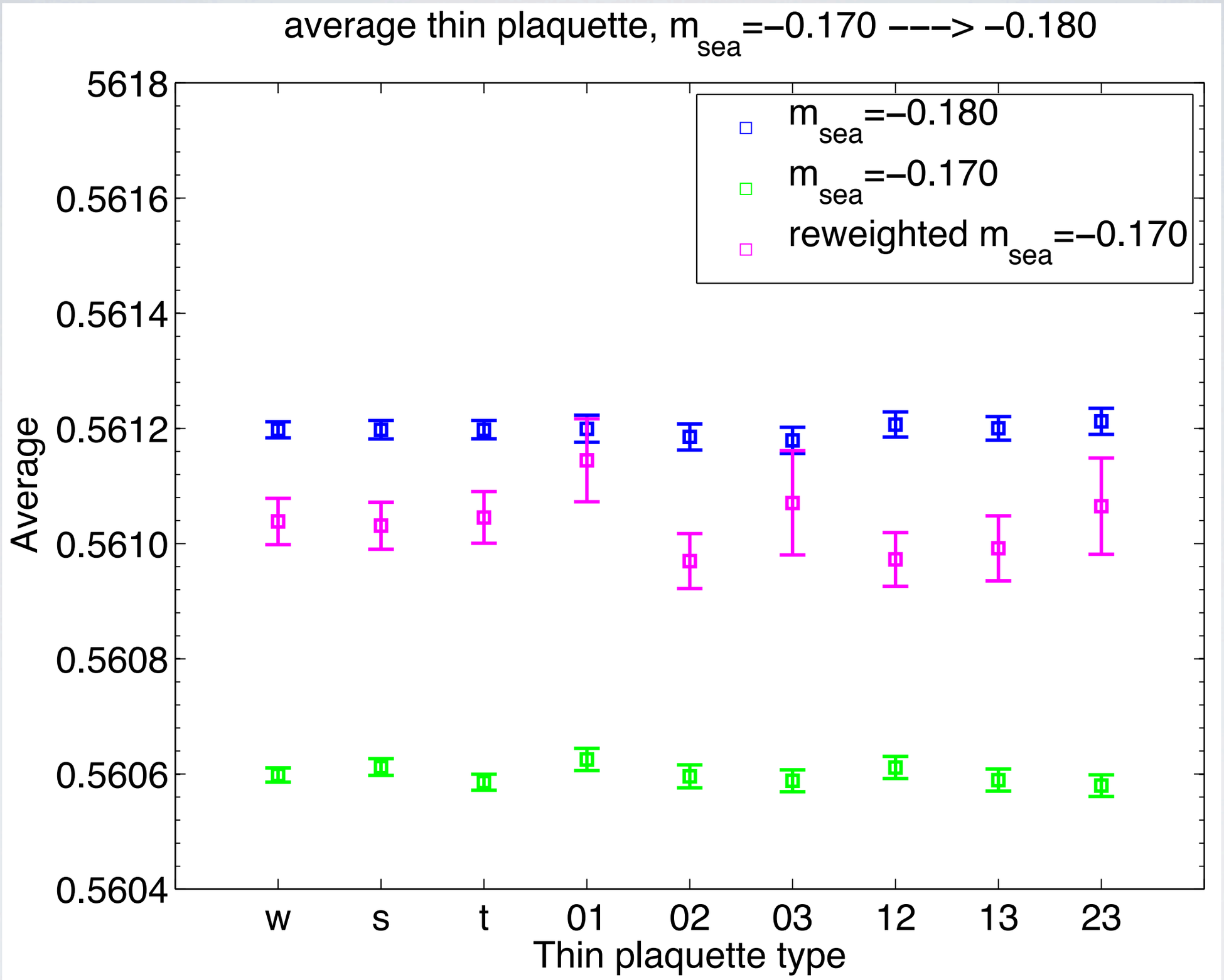
$$I[f] \approx \sum_i^k \omega_i^2 f(\theta_i)$$

θ_i are the eigenvaluesand ω_i the squares of the first elements of the normalized eigenvectors of the Lanczos matrix T_k

We apply this method to reweighting: [A. Rehim W. Detmold KO]



plot by A. Rehim

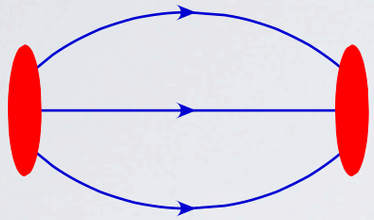


plot by A. Rehim

REWEIGHTING

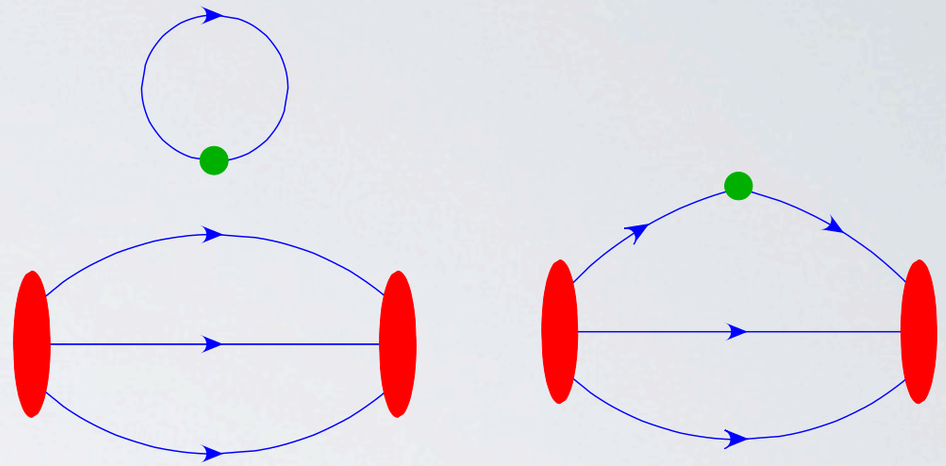
- Is already very useful and may become even more so in the near future
- Linear algebra methods for determining the reweighting factor work well
- Reweighted observables agree with exact results provided that the shifts in the action parameters are small
- Can we find further improvements? Do multi-grid like approaches exist?

CORRELATION FUNCTIONS



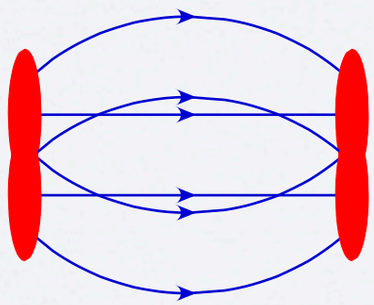
$$C_{2pt}(\vec{p},t) = \langle J_{\vec{p}}(t) J(0) \rangle$$

Spectrum



$$C_{3pt}(\vec{p},\vec{q};t,\tau) = \langle J_{\vec{p}}(t) \mathcal{O}(\vec{q},\tau) J(0) \rangle$$

Structure



$$\overset{\text{pxpxpxpxpxpx}}{C_{mp}} = \langle J_1(t) J_2(t) J_1(0) J_2(0) \rangle$$

Interactions and multi-particle Spectrum

DEFLATION

- Iterative solvers slow down when the matrix has very low eigenvalues
- Basic Idea: Project out the low modes
- Computing eigenvectors is expensive
  - 1 eigenvector roughly costs as much as solving one linear system
- Deflation can work well if the cost of constructing a sufficient basis spanning the low eigenvector space is small
- Two approaches:
  - Krylov methods: Computing the basis while solving a linear system
    - EigCG: symmetric problem [Stathopoulos and KO **arXiv:0707.0131**]
    - GMRES-DR: non-symmetric problem [Morgan, Wilcox et al **arXiv:math-ph/0405053 arXiv:0707.0502 arXiv:0707.0505**]
  - Algebraic Multi-grid/Domain decomposition [M. Clark et.al. Brannick et al arXiv:0707.4018] [Luscher arXiv:0706.2298] [Babich arXiv:1005.3043]

The eigCG algorithm

Developed with A. Stathopoulos (W&M)

Basic goals:

- Let CG do its job in solving the system
- Slowly accumulated few low eigenvectors of our matrix **A** by interrupting CG without restarting it
- Use are "recurrence" relation that improves eigenvector convergence
- Use limited memory i.e. do not store all residual vectors that CG produces

The eigCG algorithm (N_{ev}, m)

$k = 0; j = 0; x_0 = 0; r_0 = b$

while $r_k \neq 0$

$k = k + 1$

if $(k = 1)$

$p_1 = r_0$

else

$$\beta_k = \frac{r_{k-1}^\dagger r_{k-1}}{r_{k-2}^\dagger r_{k-2}}$$

$p_k = r_{k-1} + \beta_k p_{k-1}$

end

$$\alpha_k = \frac{r_{k-1}^\dagger r_{k-1}}{p_k^\dagger A p_k}$$

$x_k = x_{k-1} + \alpha_k p_k$

$r_k = r_{k-1} - \alpha_k A p_k$

end

$x = x_k$

$$v_j = \frac{r_k}{\|r_k\|}$$

update the T_j matrix

if $(j = m)$

diagonalize $T_m \rightarrow Y_m$ keep lowest N_{ev}

diagonalize $T_{m-1} \rightarrow Y_{m-1}$ keep lowest N_{ev}

QR factorize $Y_m,\ Y_{m-1} \rightarrow Q$

construct $H = Q^\dagger T_m Q$

H is $2N_{ev} \times 2N_{ev}$

diagonalize $H \rightarrow Z$

$$v_i = \sum_{n=1}^{m} (QZ)_{ni} v_n \quad \{i = 1..2N_{ev}\}$$

$j = 2N_{ev}$

rebuild T_j

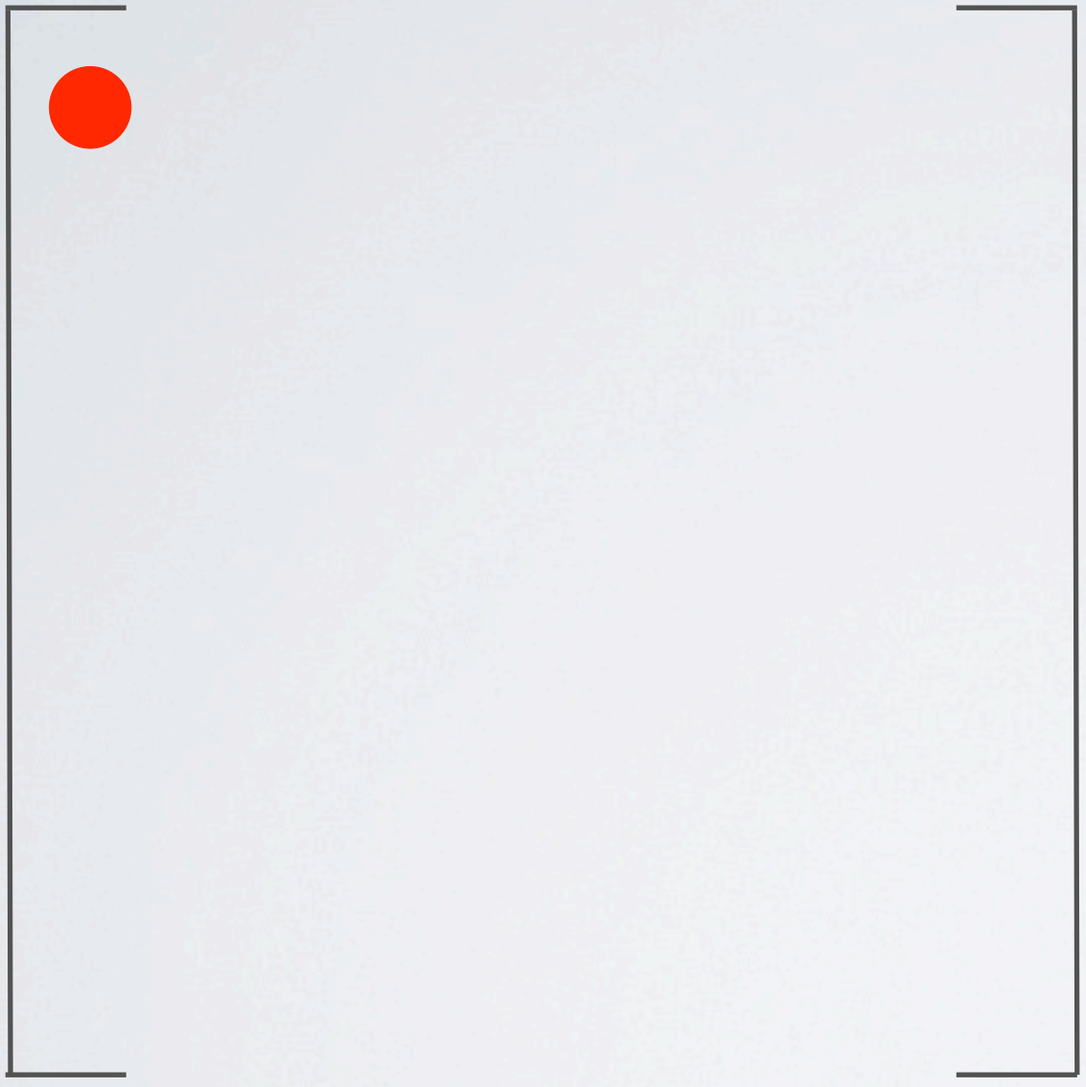
end

The eigCG algorithm

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

$N_{ev} = 2 \quad m=9$

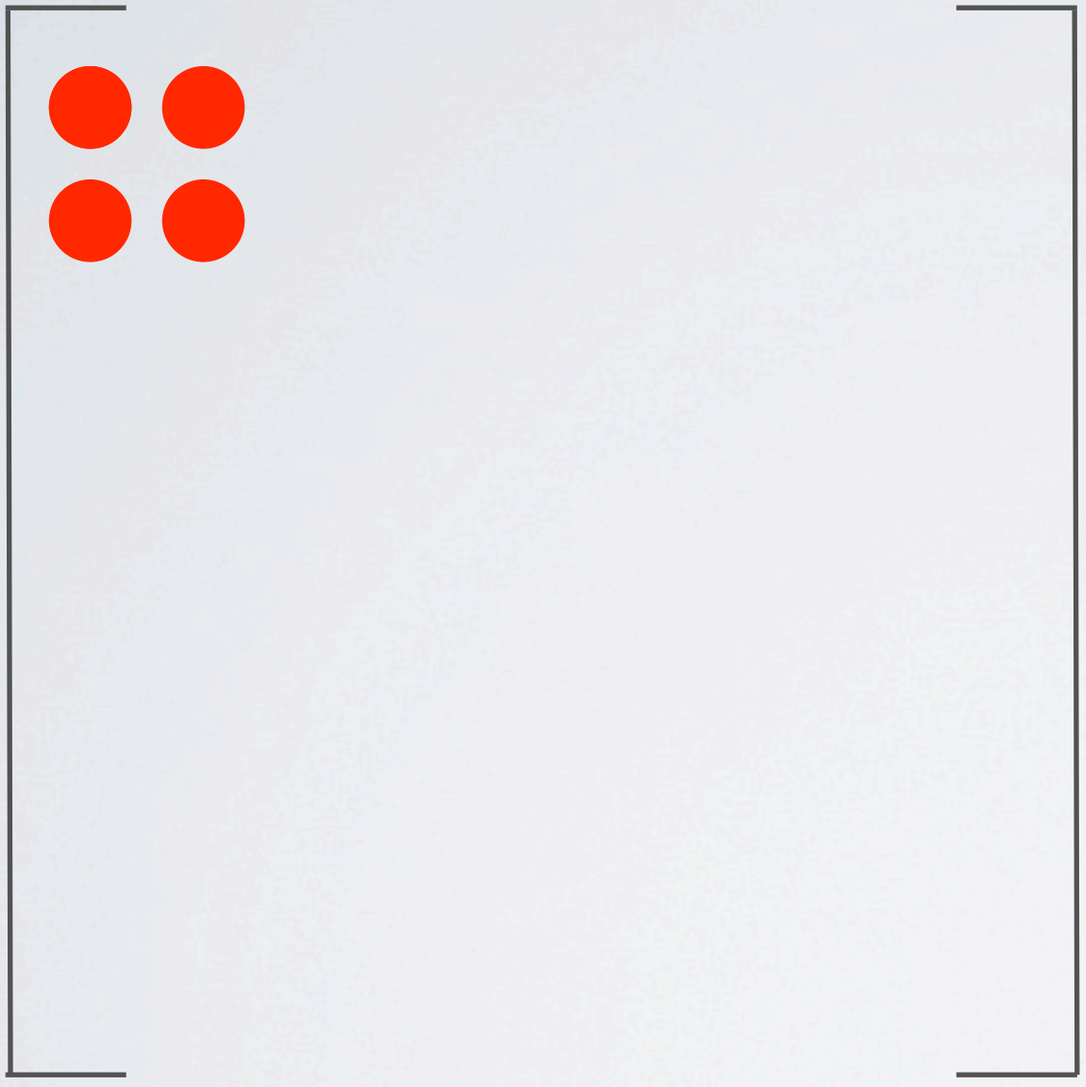
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

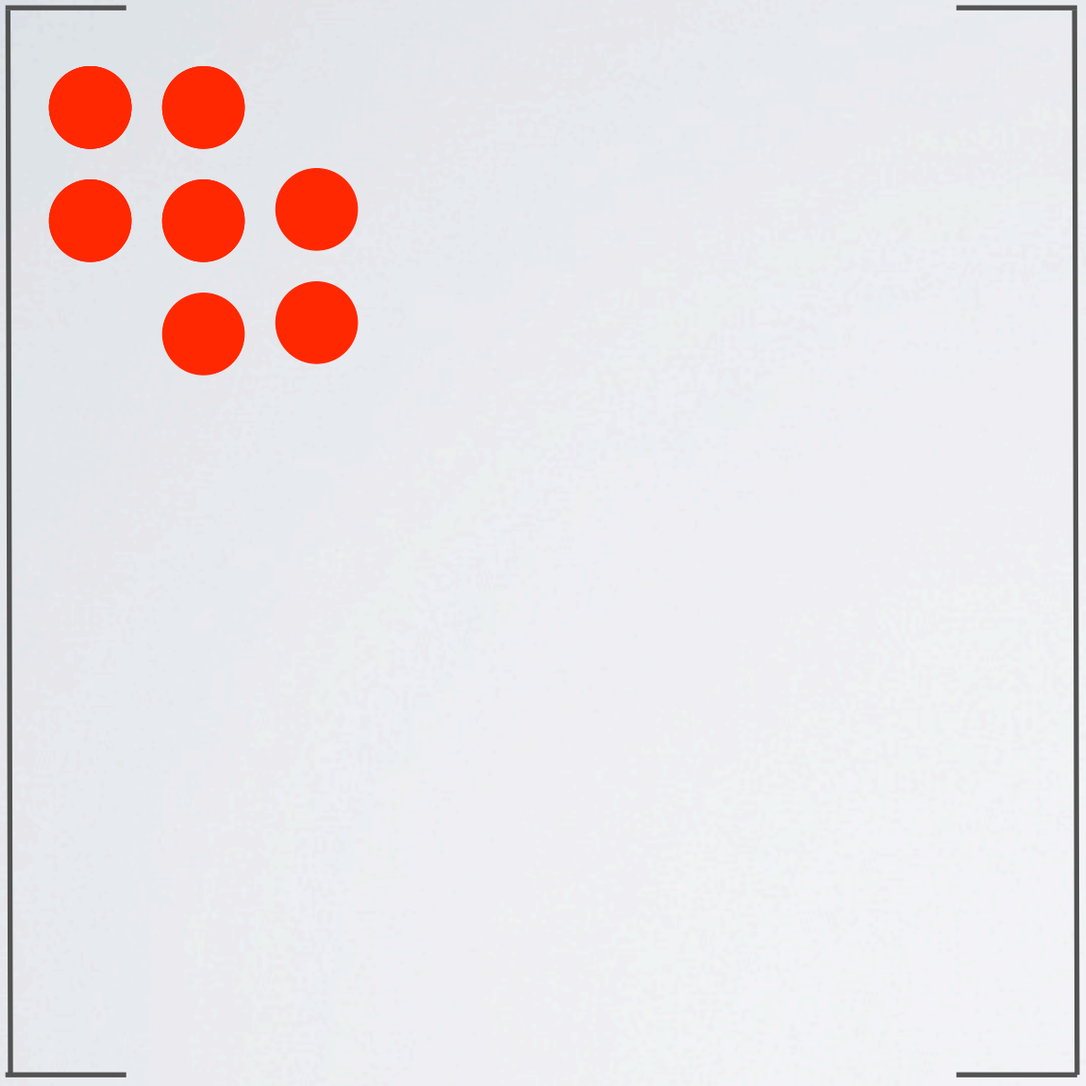
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

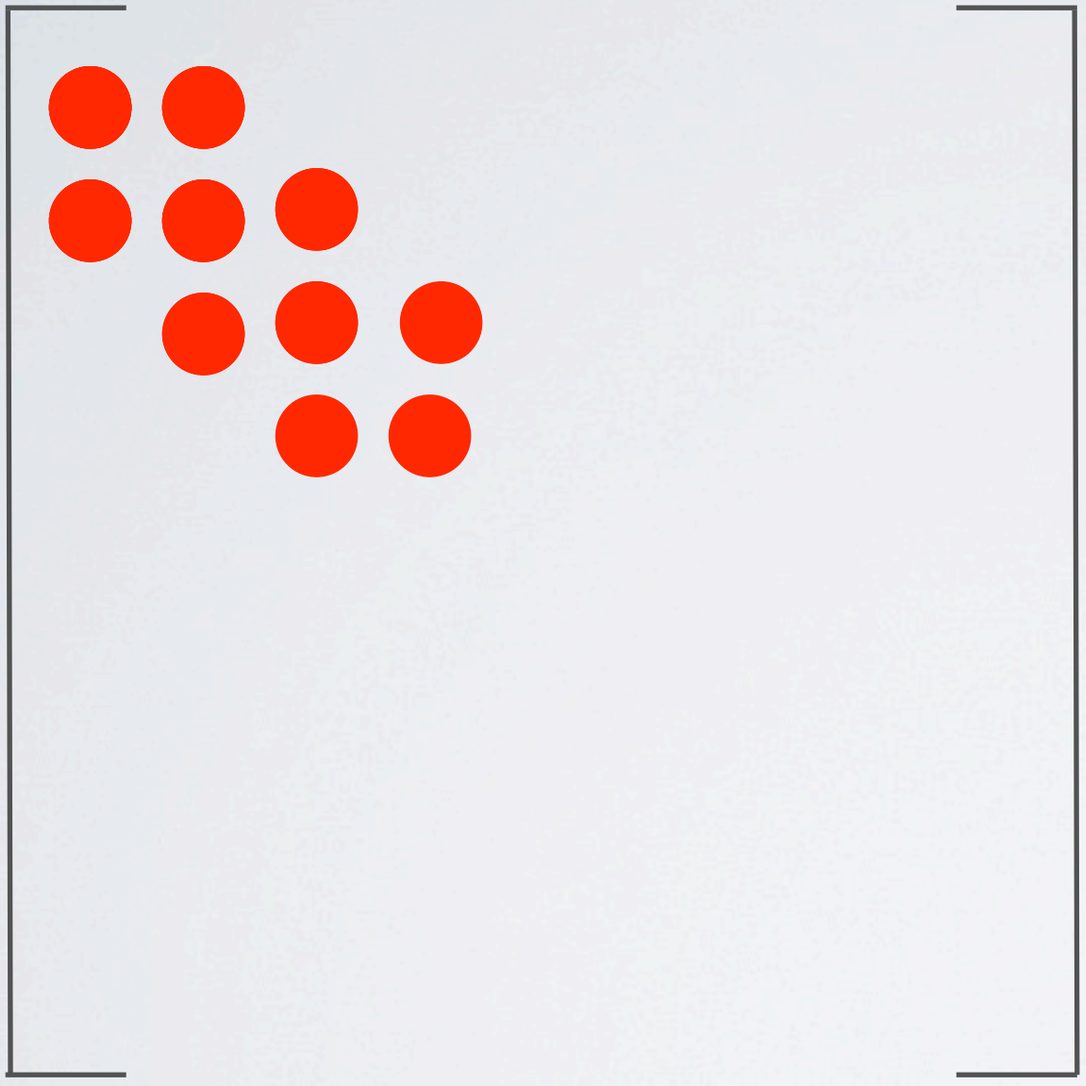
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

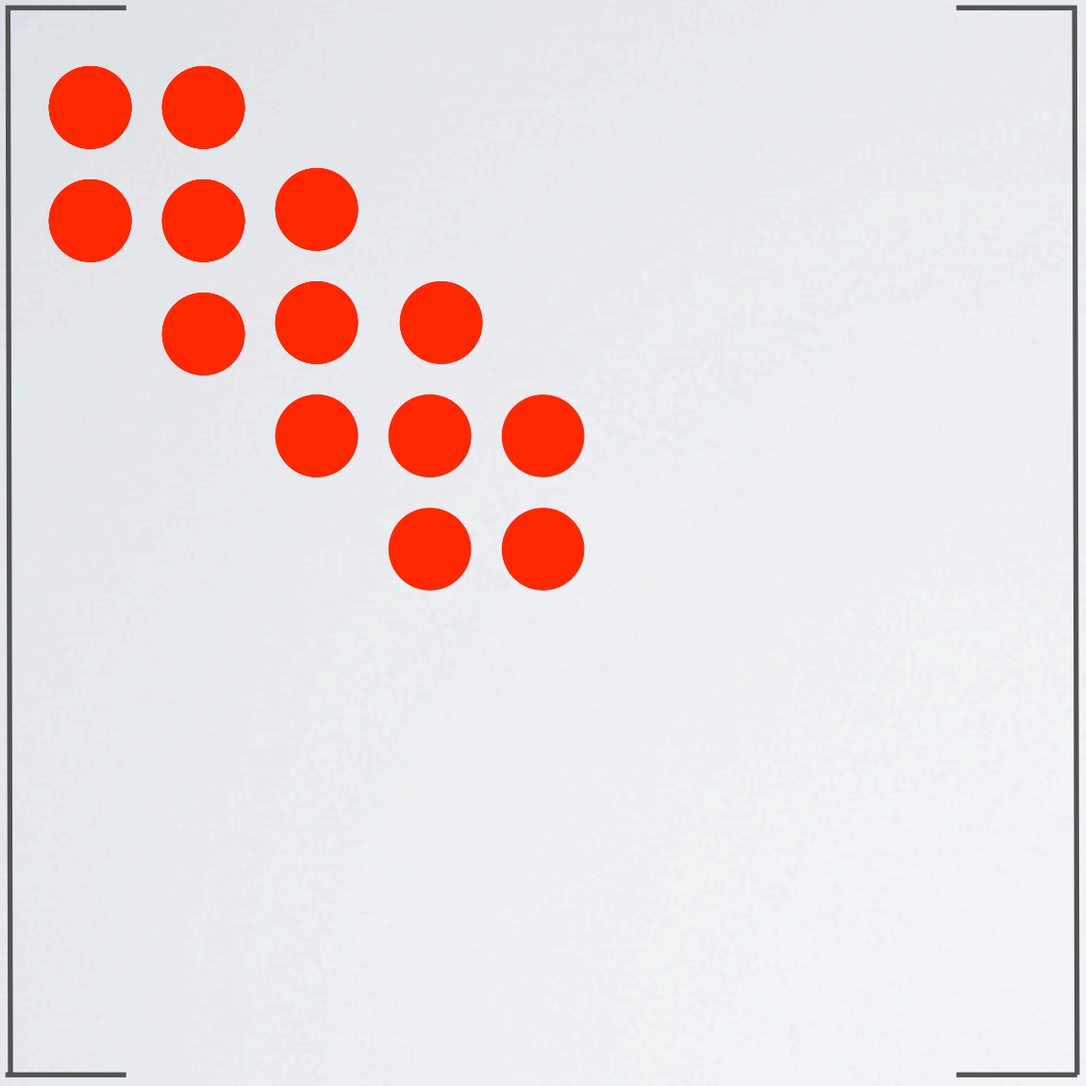
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

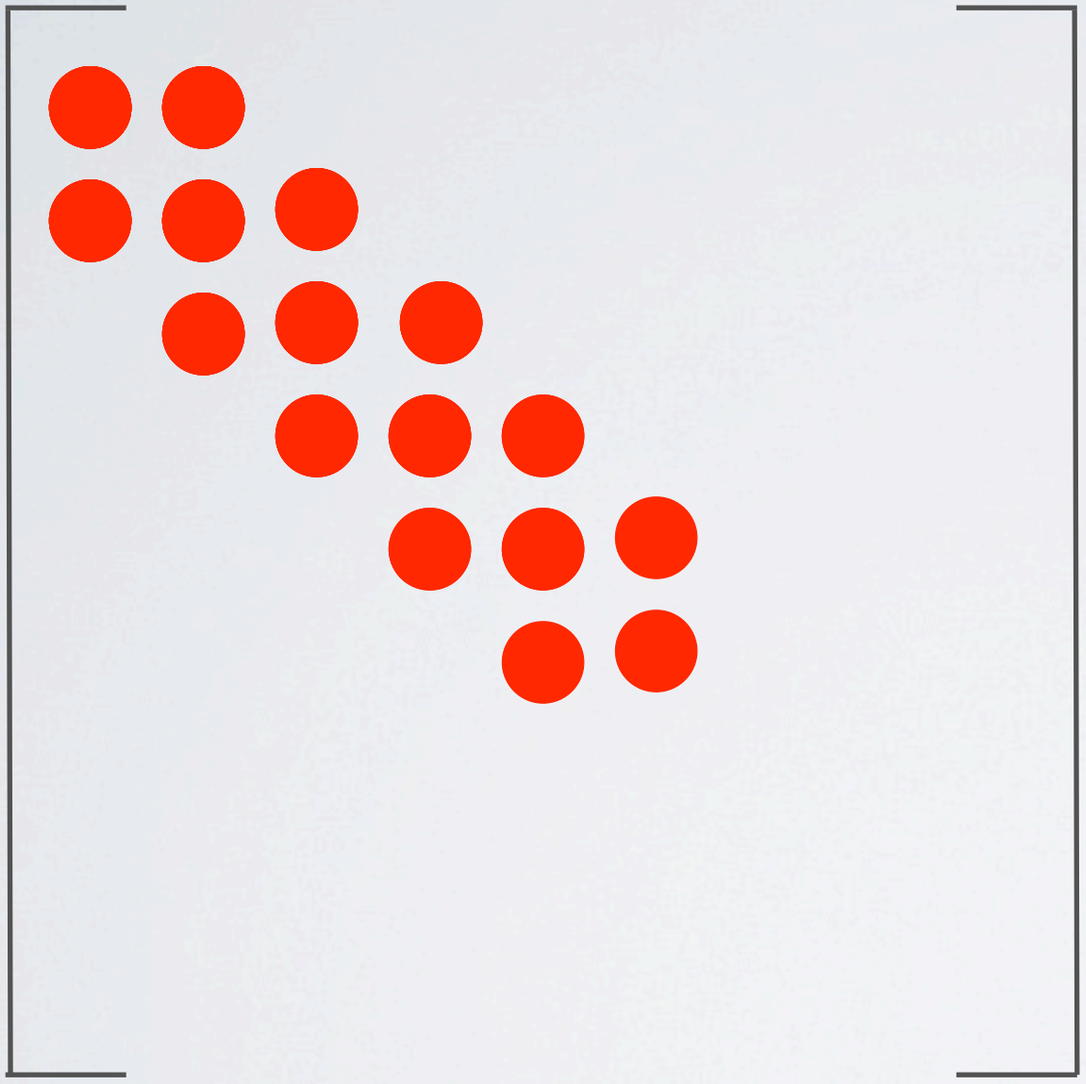
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

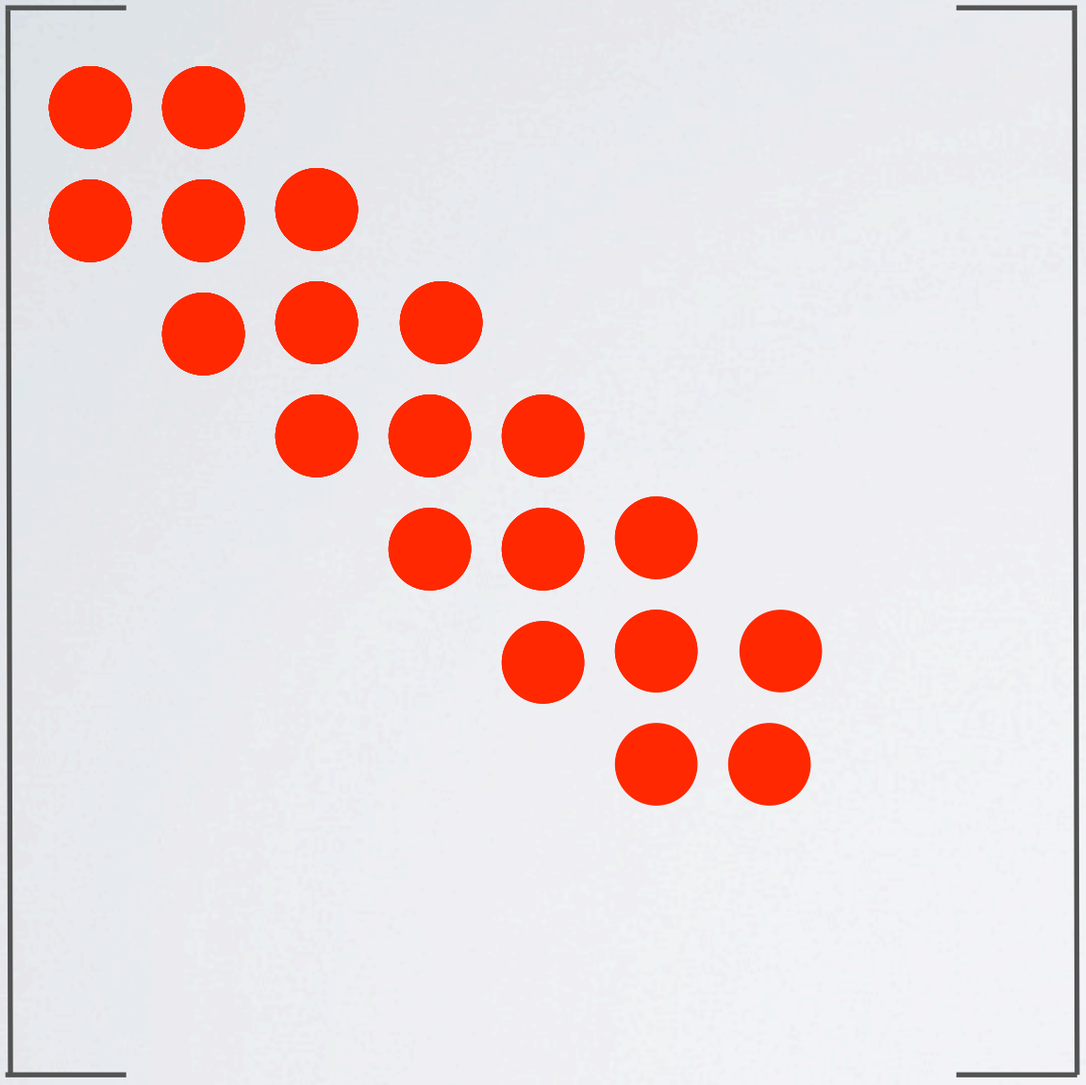
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

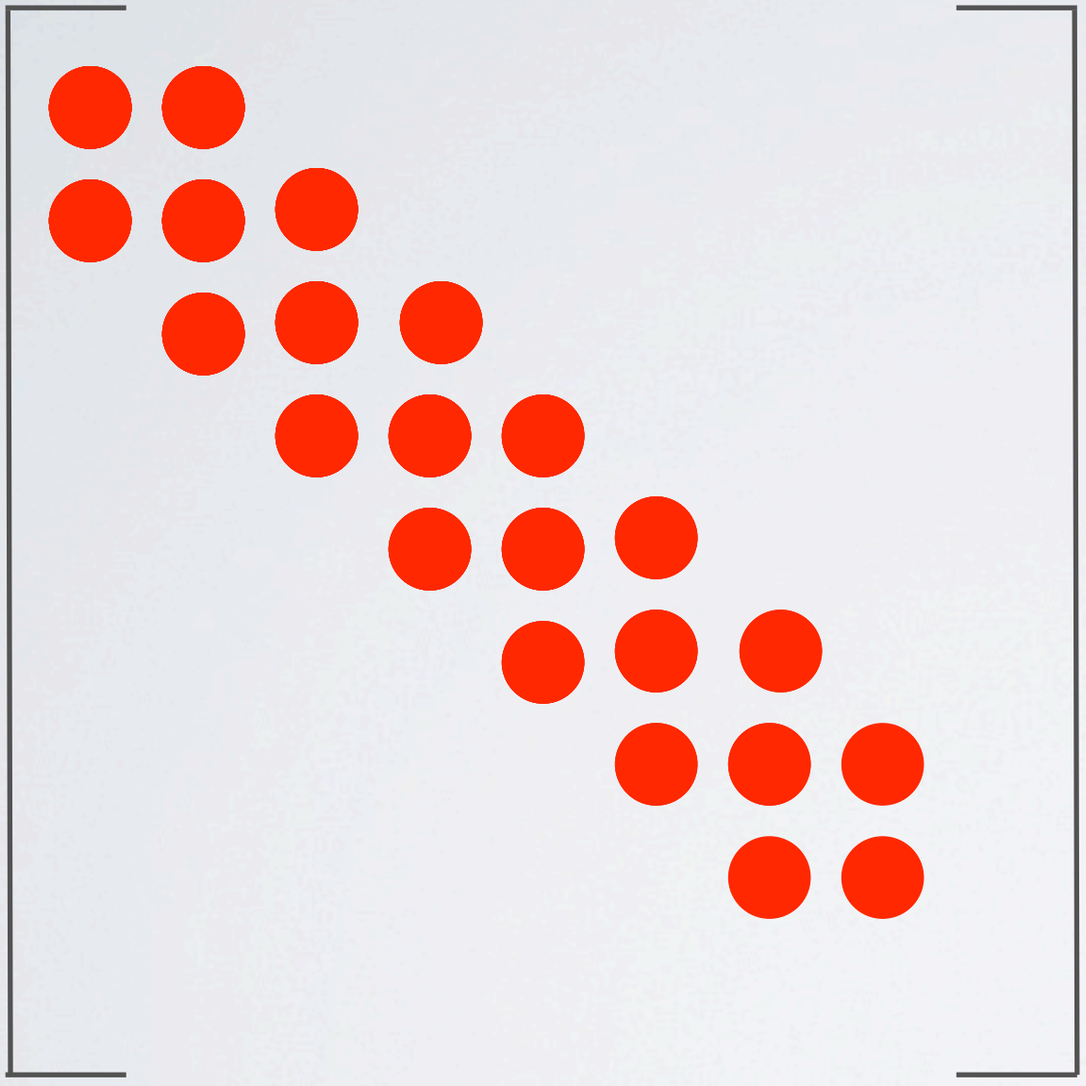
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

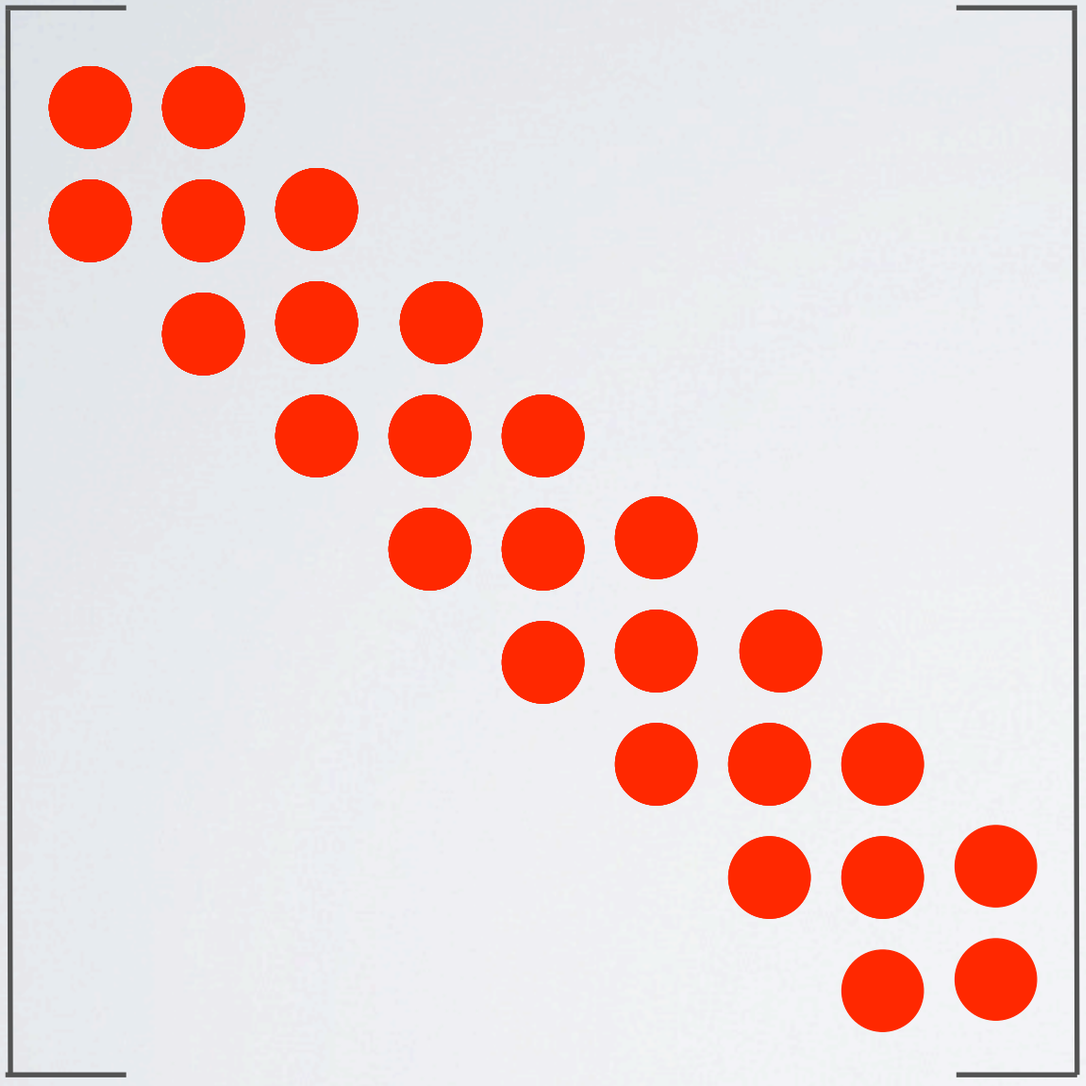
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

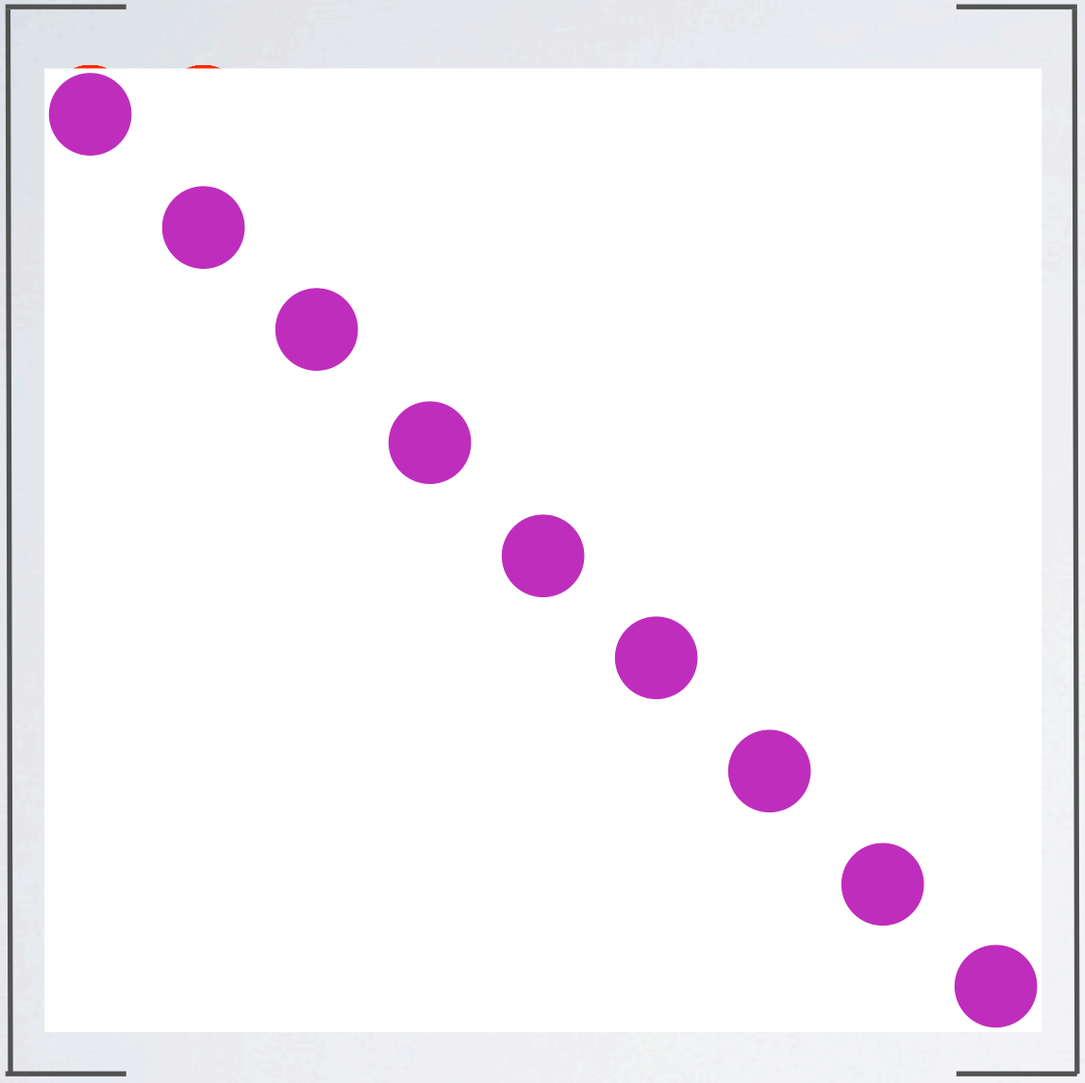
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

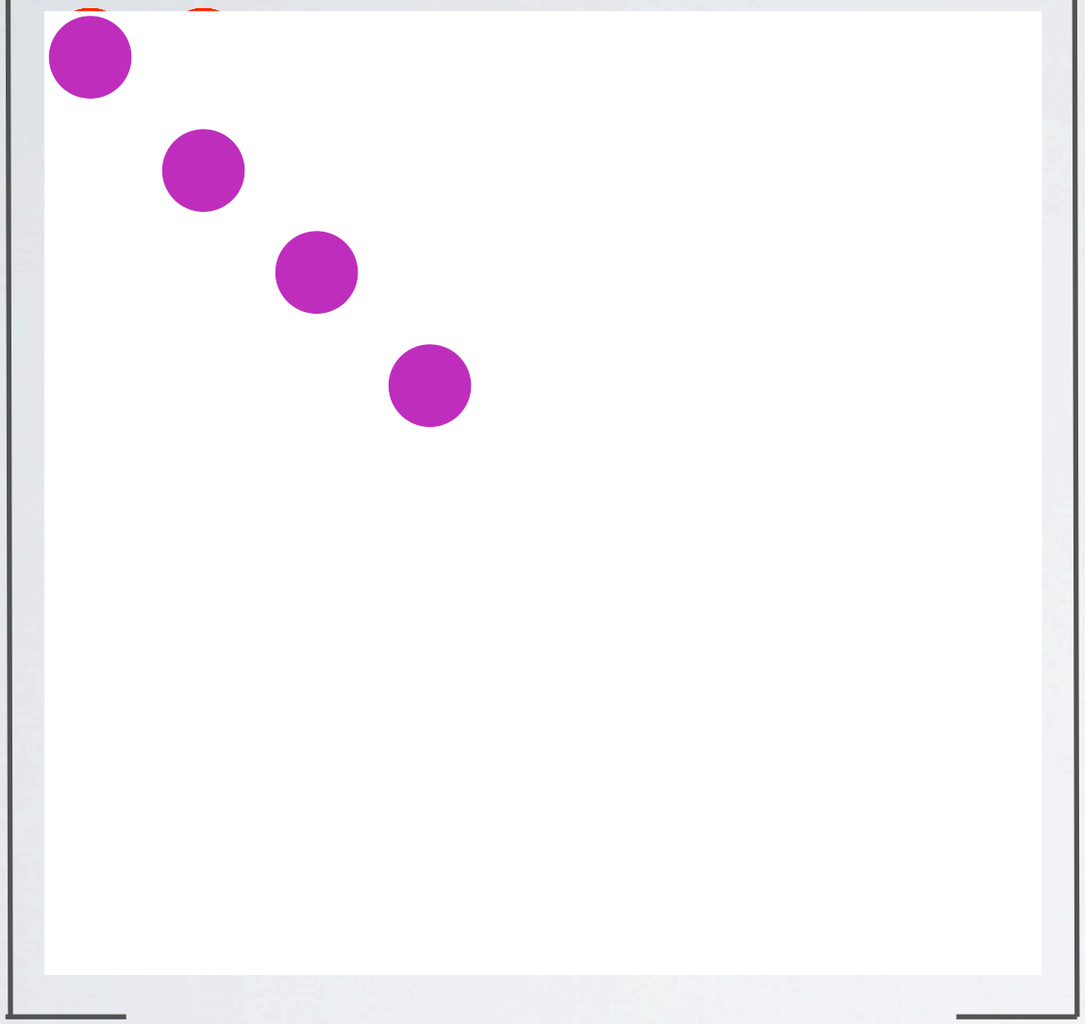
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

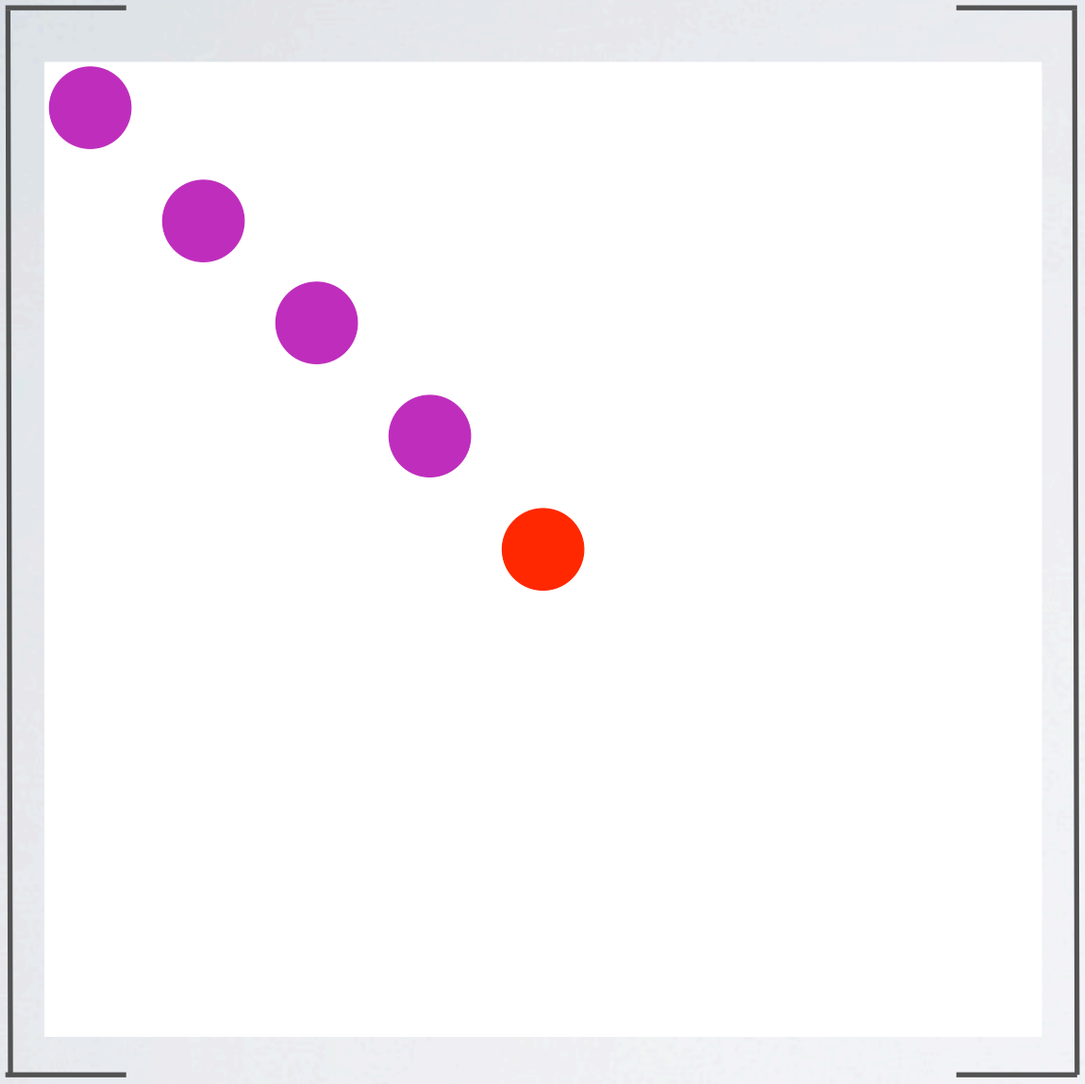
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

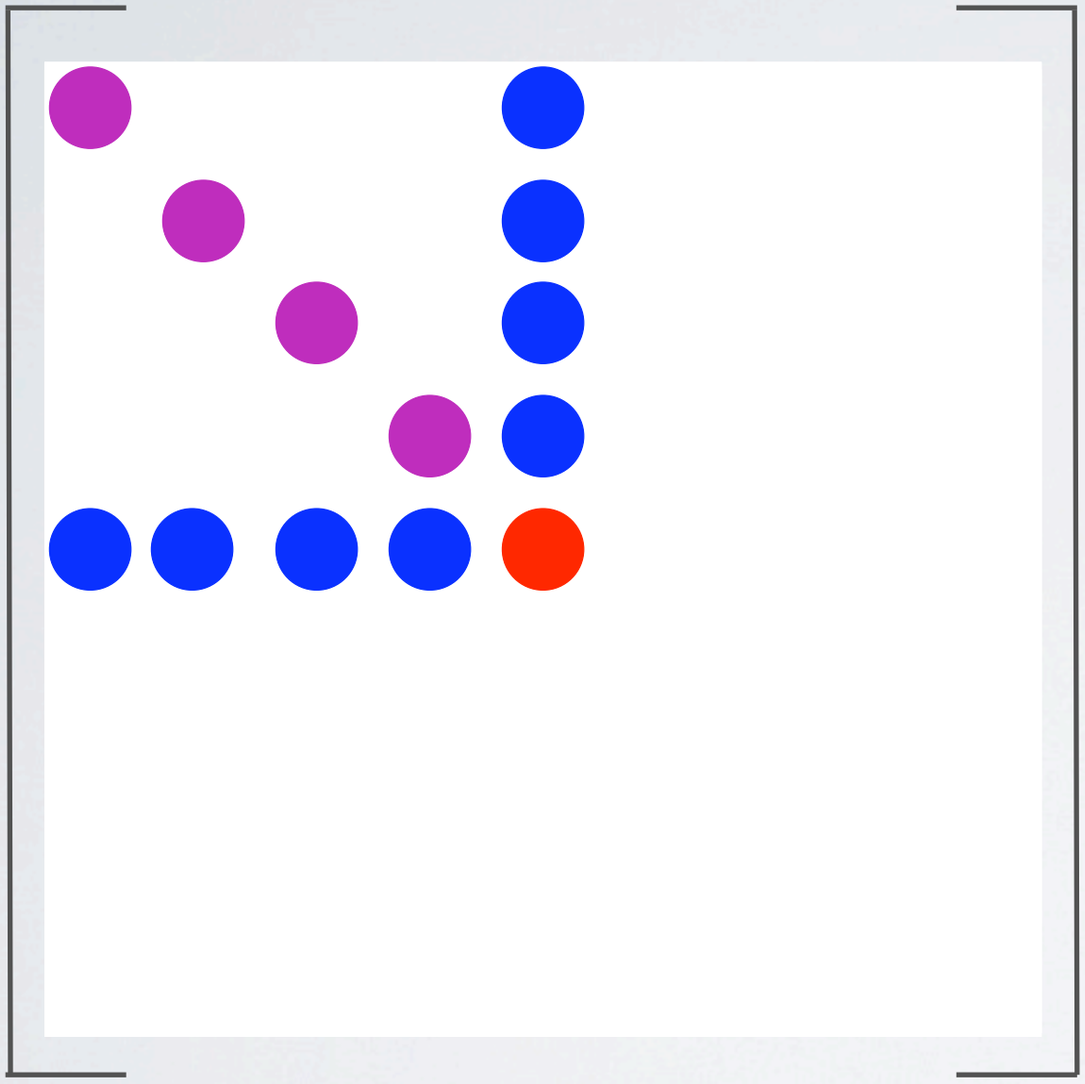
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

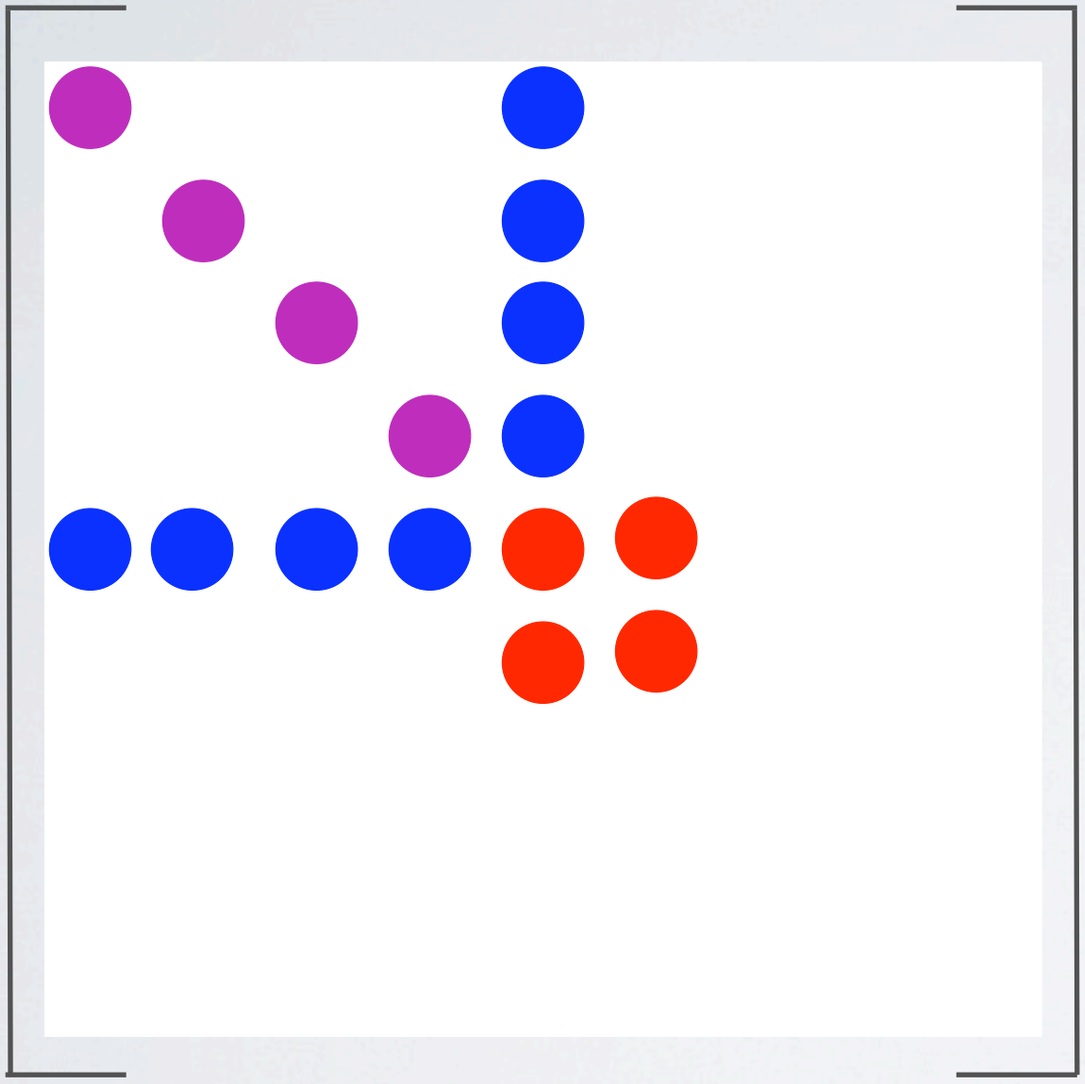
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

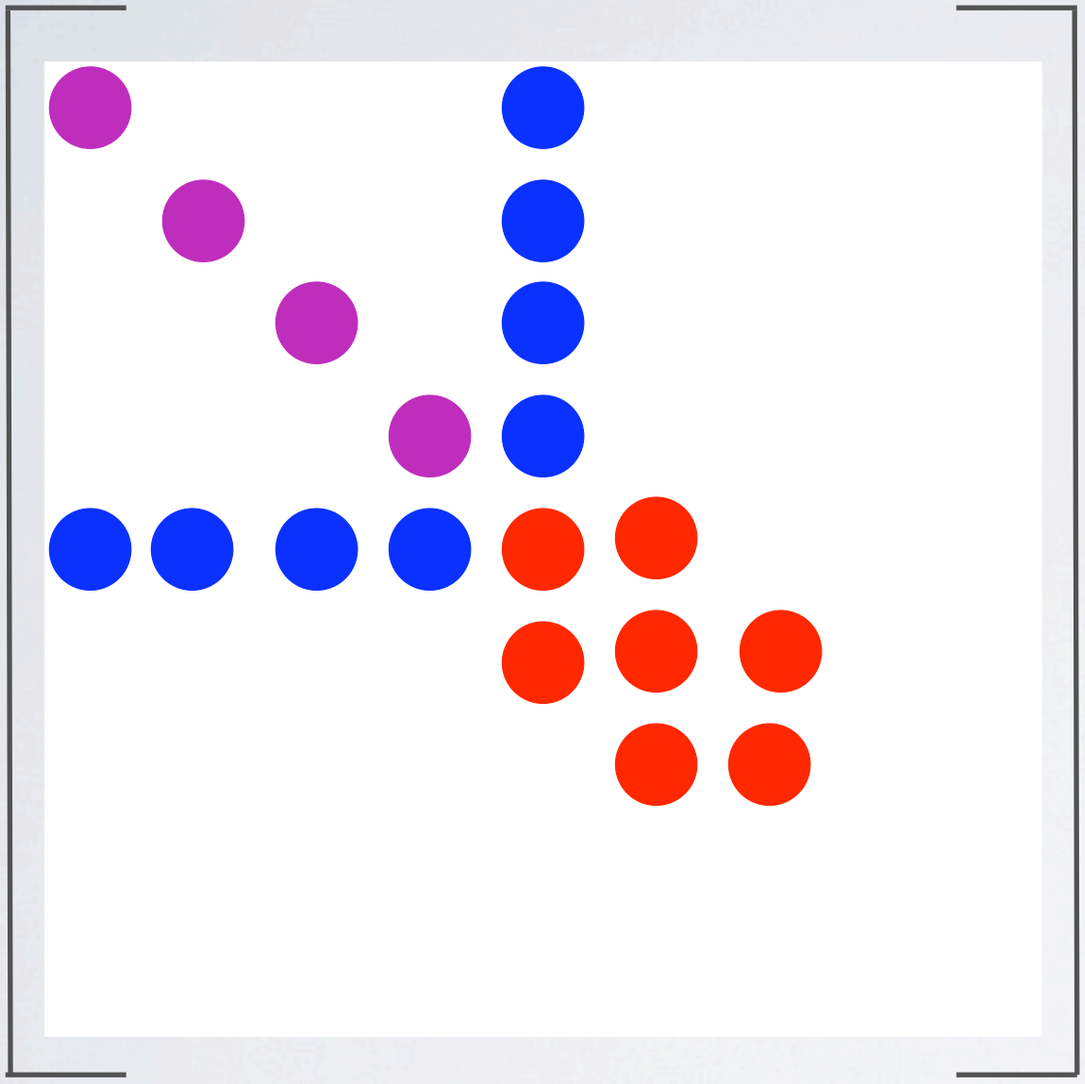
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

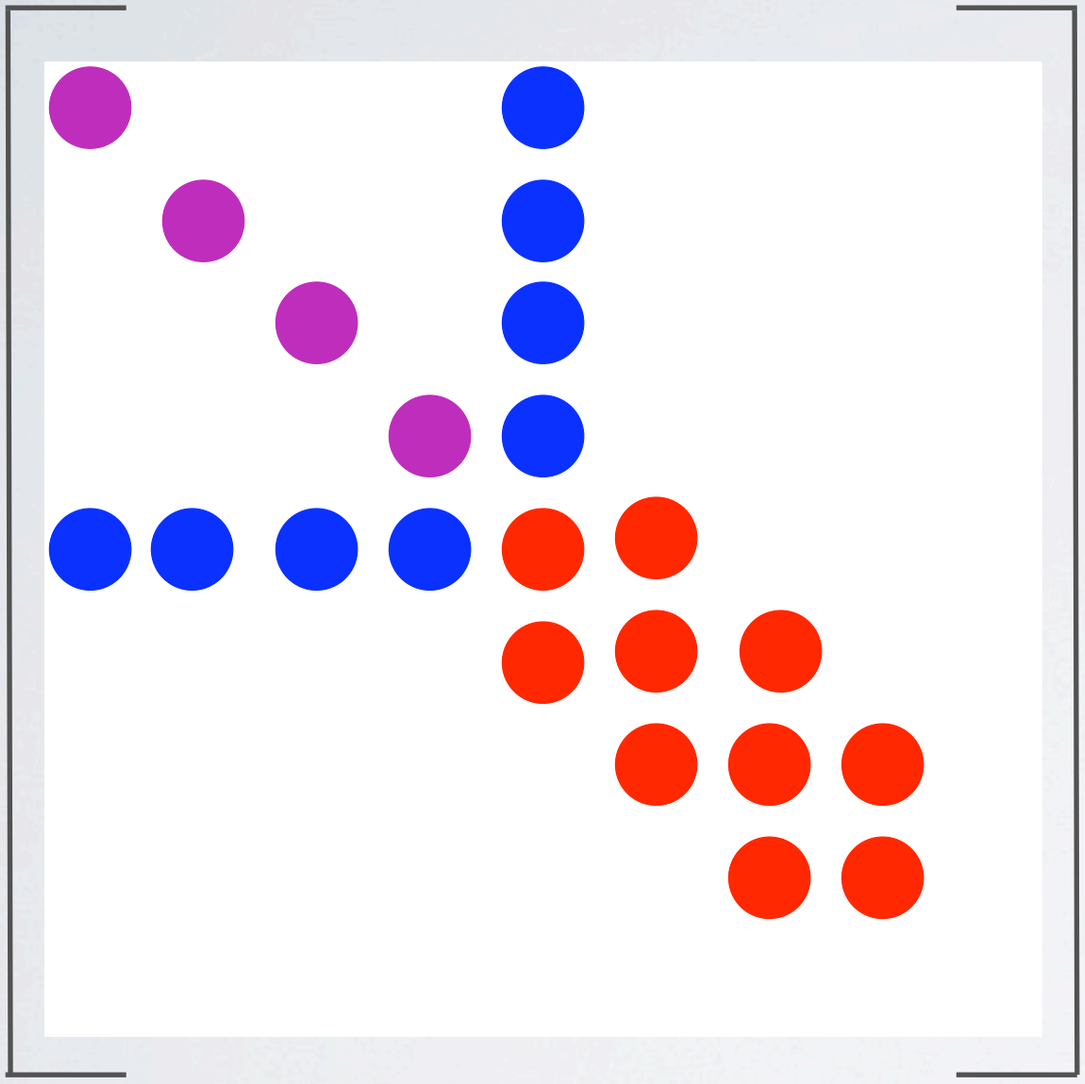
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

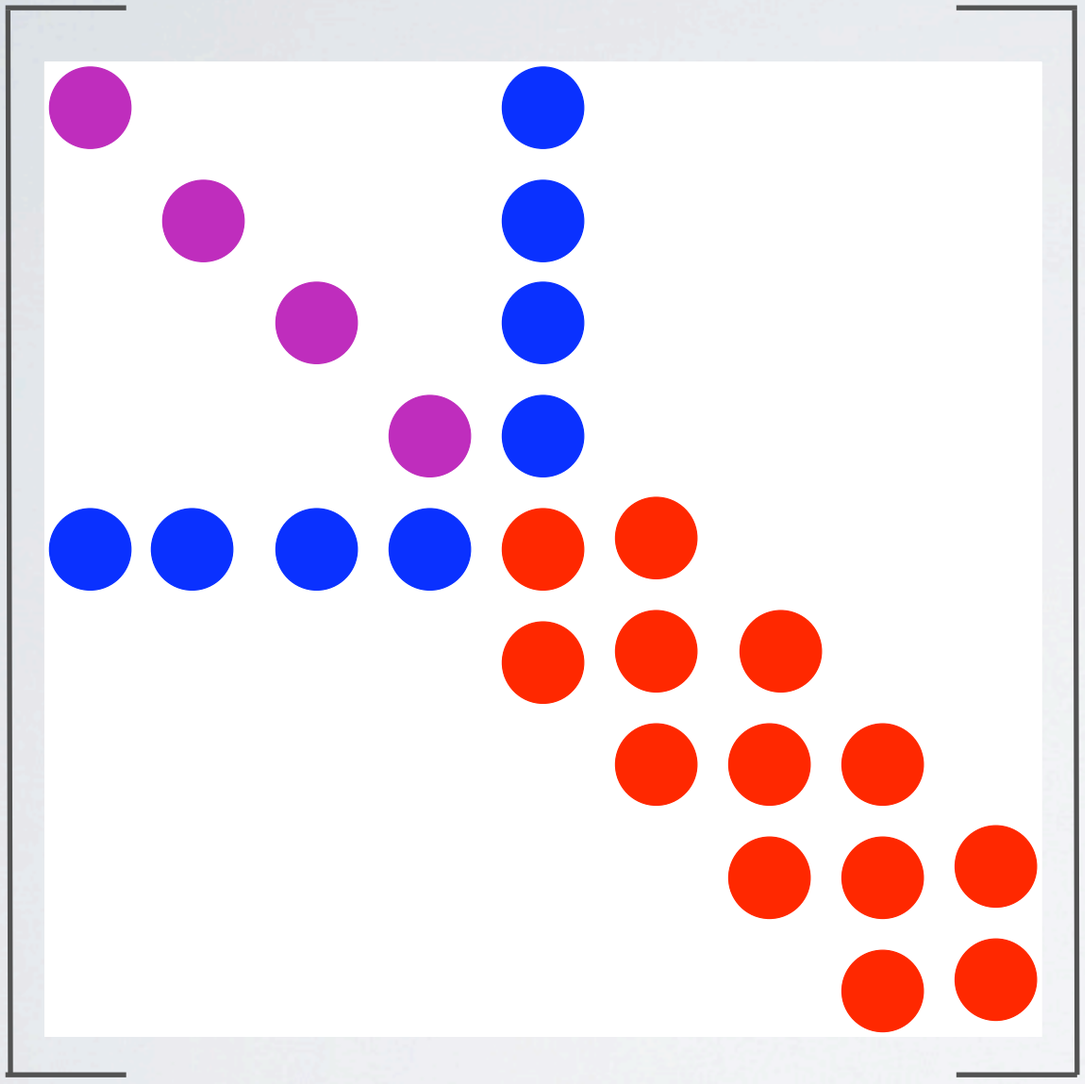
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

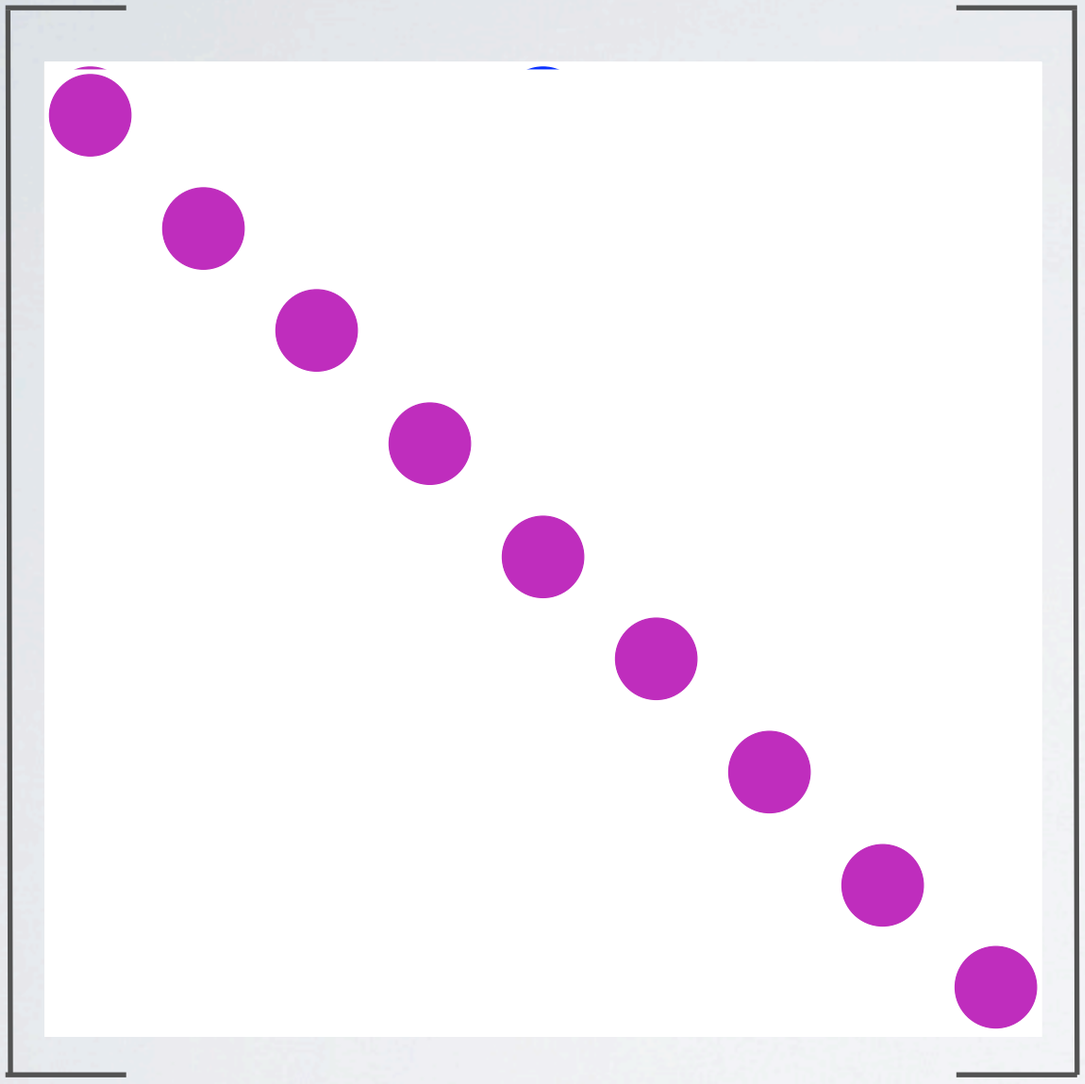
The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

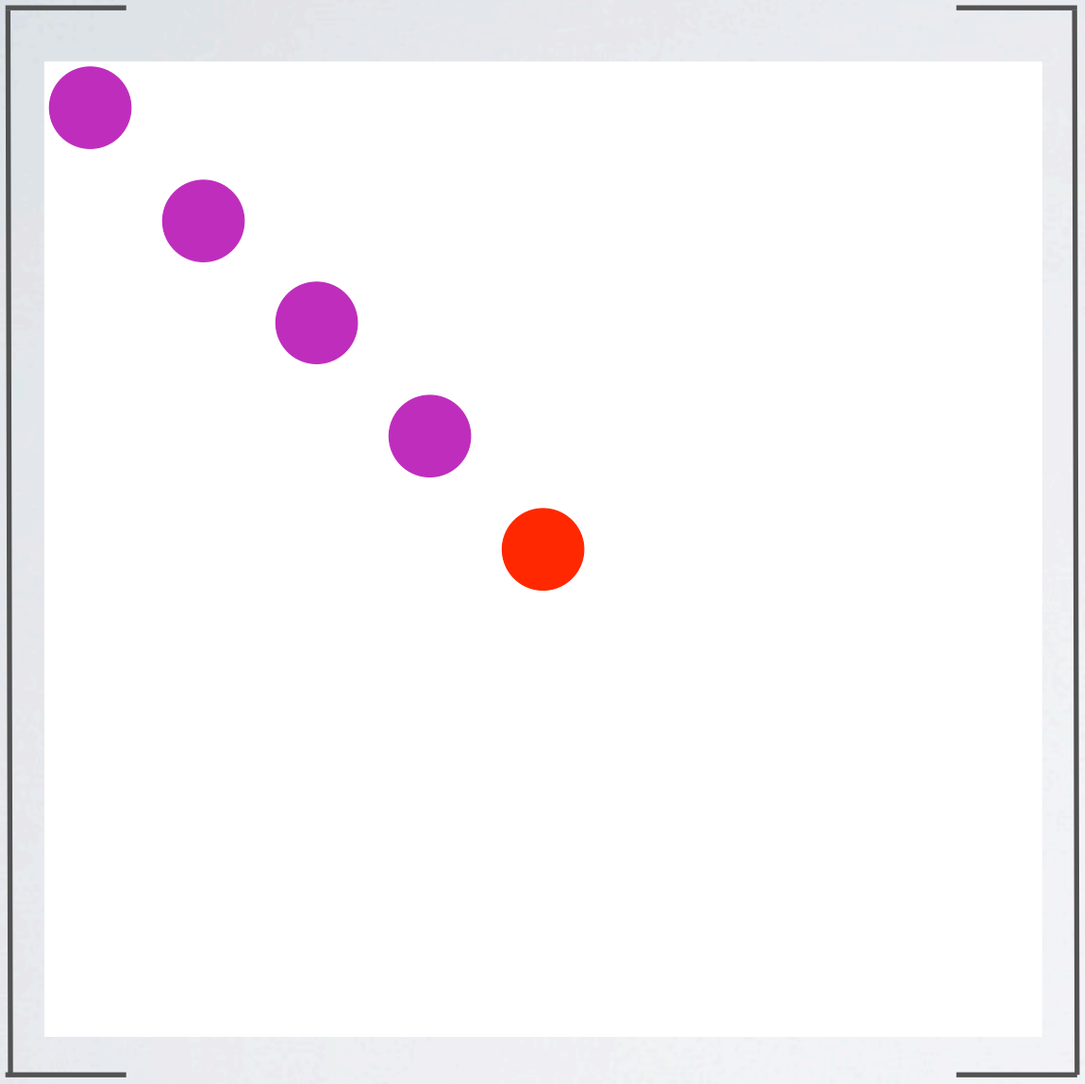
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

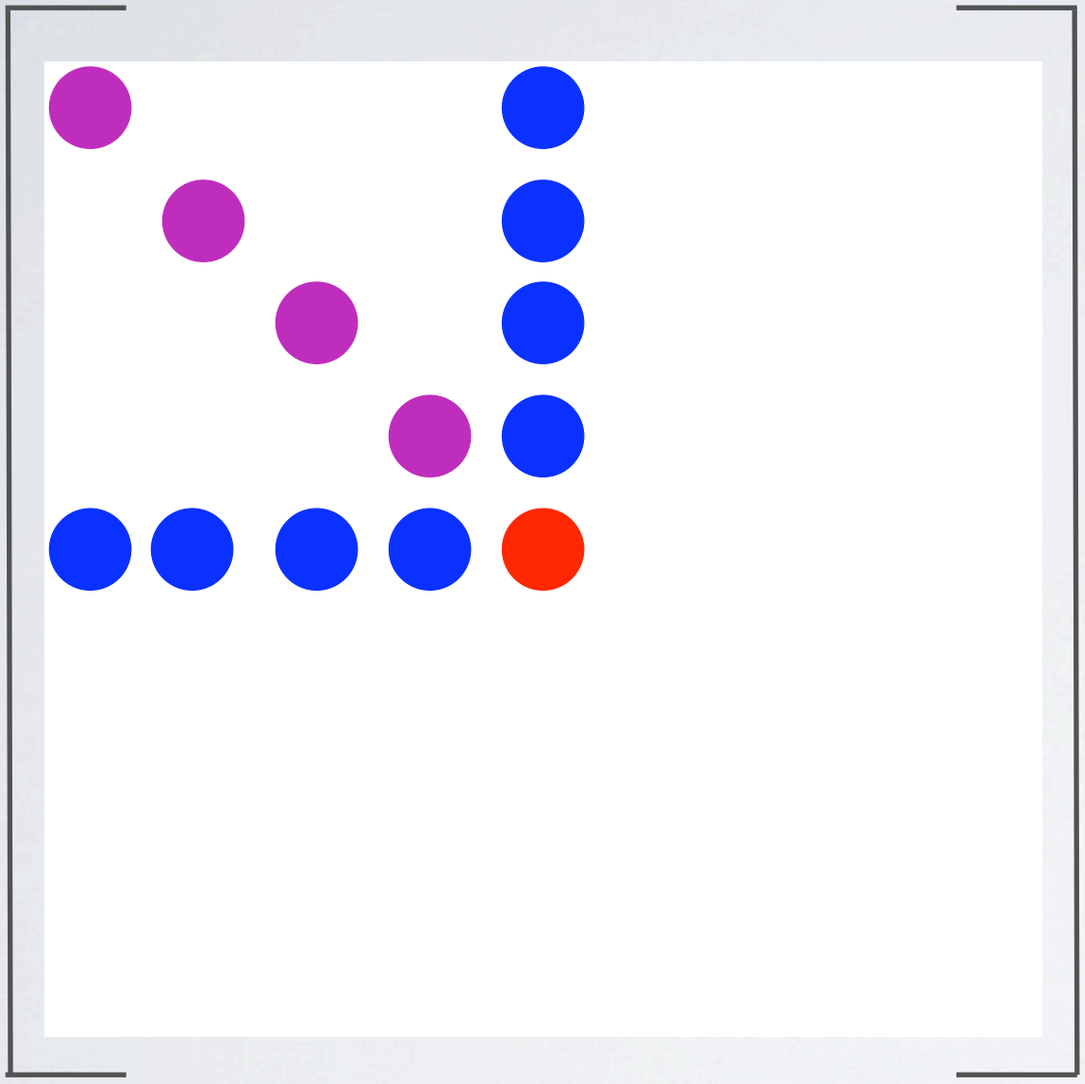
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

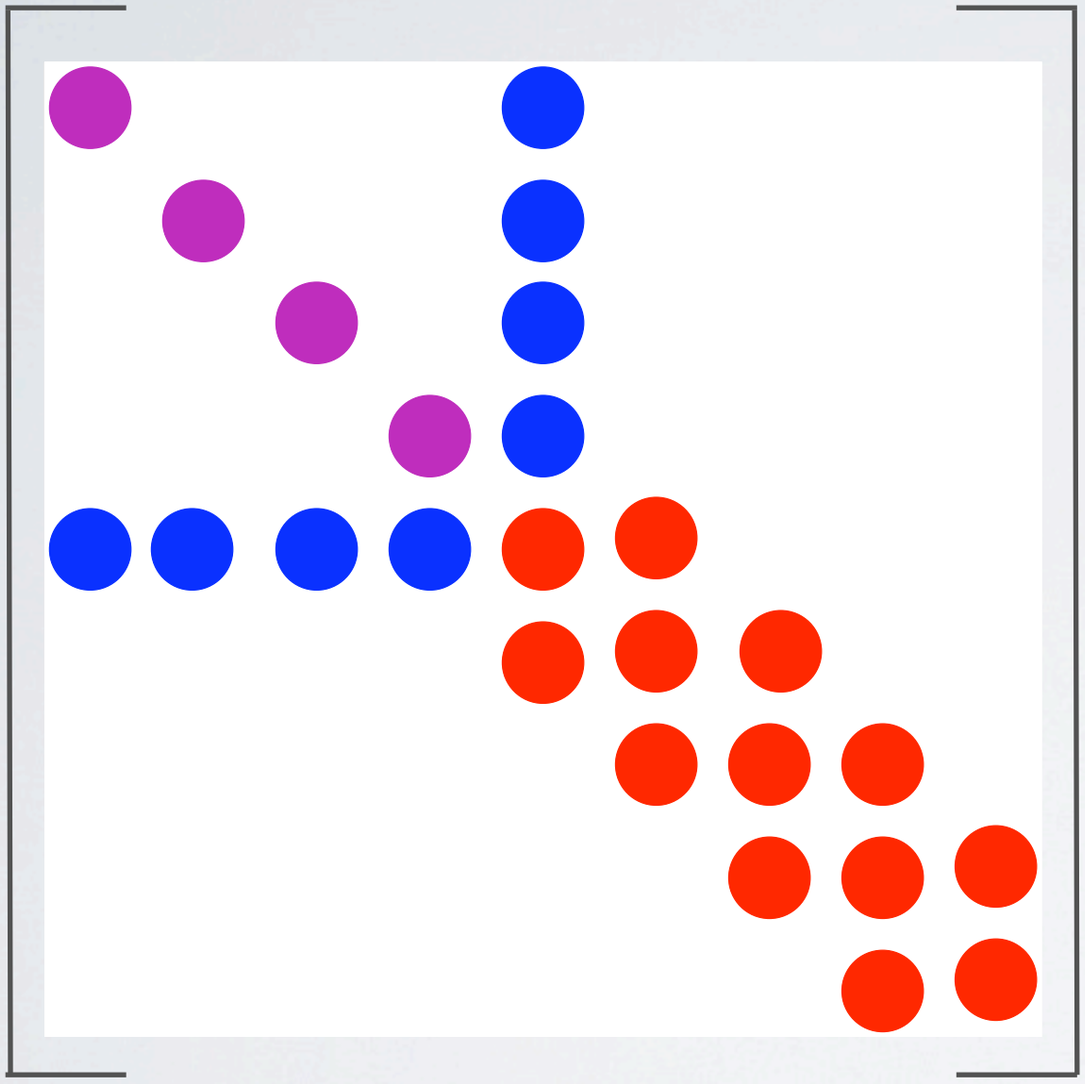
The eigCG algorithm



$N_{ev} = 2 \quad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

The eigCG algorithm



$N_{ev} = 2 \qquad m=9$

- Iterate the CG algorithm
- Save the residual vectors and fill in the tridiagonal matrix
- When max number of vectors reached: Diagonalize
- Keep only few low eigenpairs
- Continue the CG filling in the tridiagonal matrix and saving the new residual vectors

The Incremental eigCG

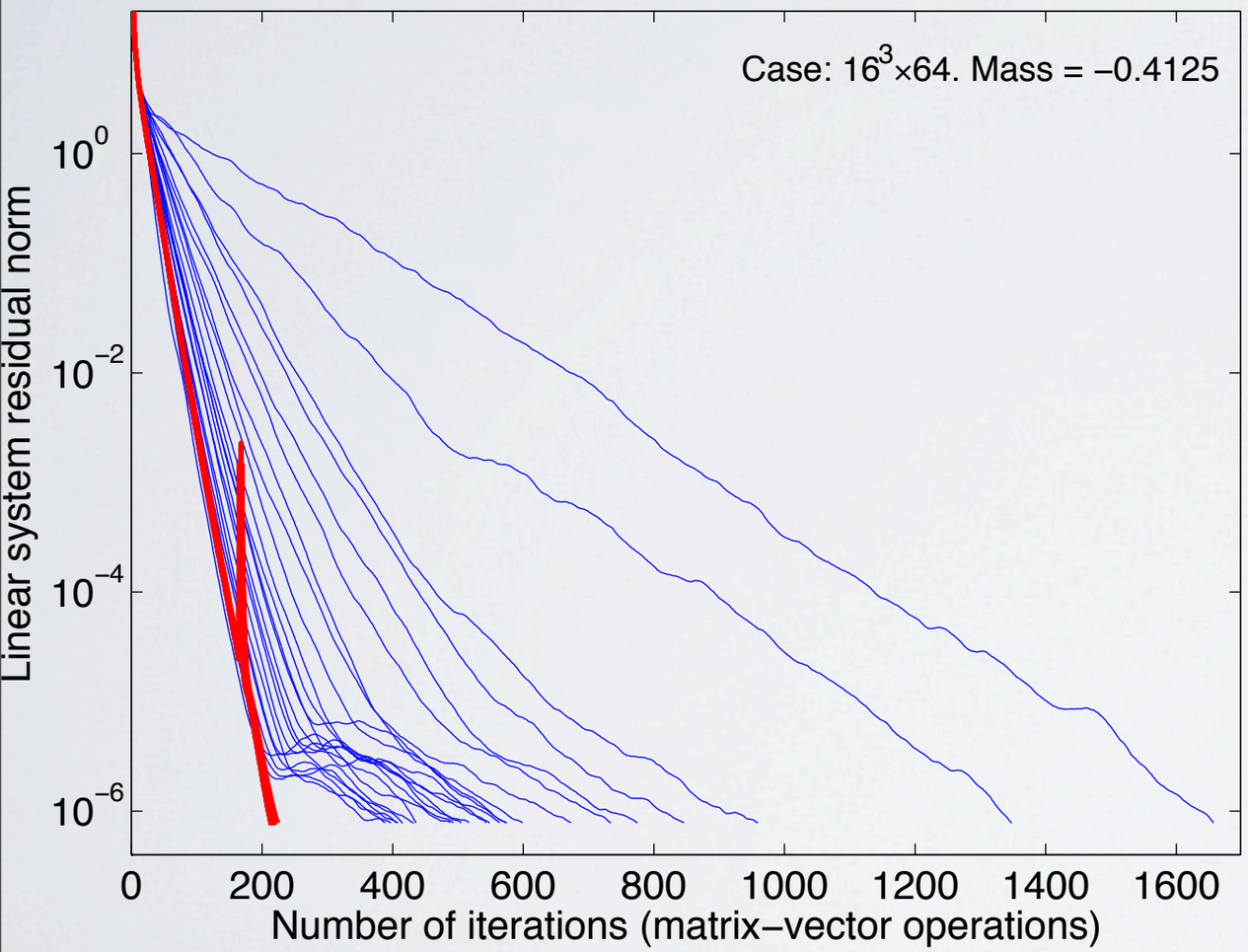
For the first s_1 right hand sides do:

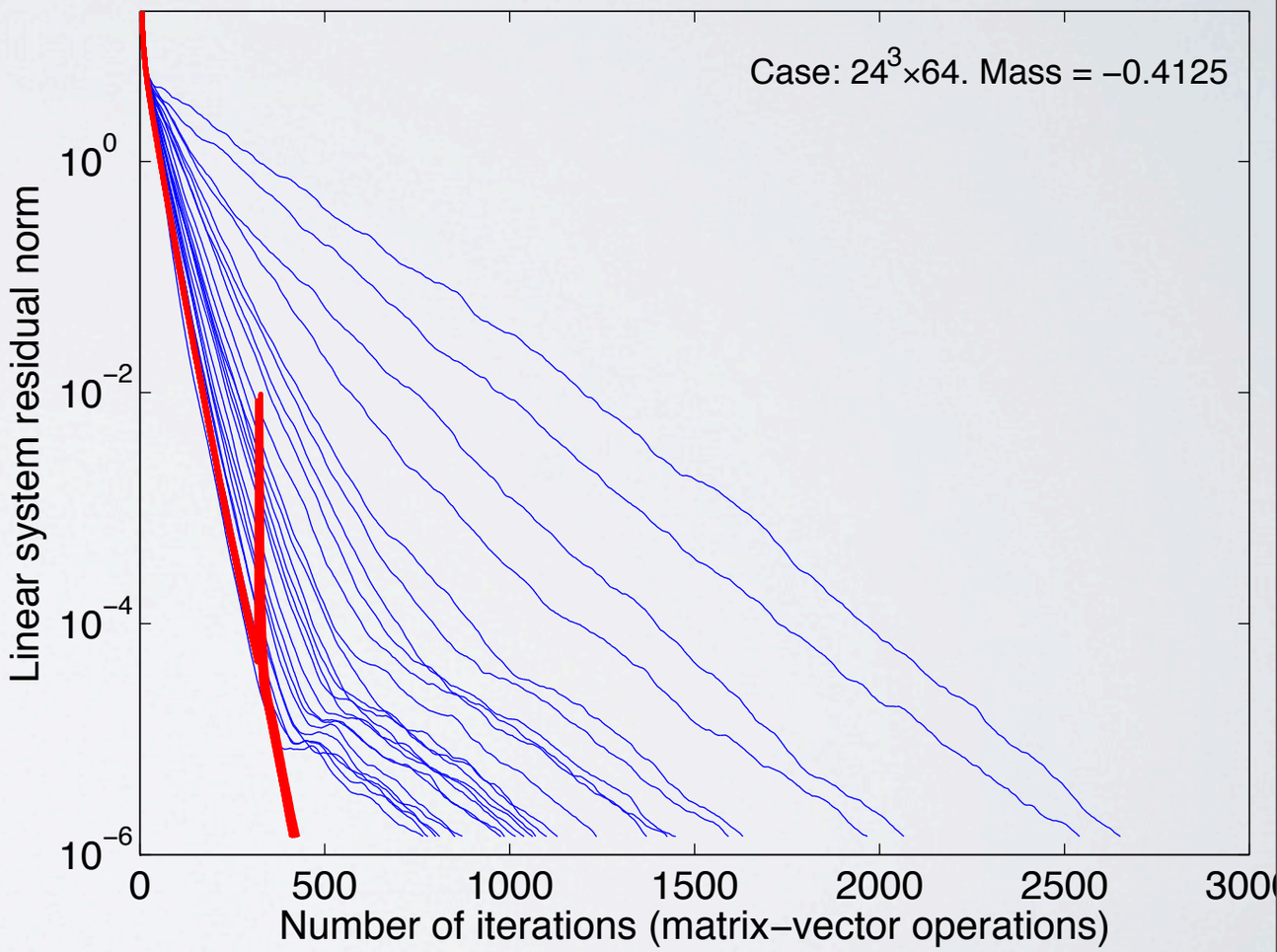
$U = [\,], H = [\,]$
for $i = 1 : s_1$
$x_0 = UH^{-1}U^H b_i$ % accumulated Ritz vectors
$[x_i, V, M\,] = \text{eigCG}(nev, m, A, x_0, b_i)$ % for s_1 initial rhs
$\bar{V} = \text{orthonormalize } V \text{ against } U$ % the init-CG part
$W = A\bar{V},\ H = \begin{bmatrix} H & U^H W \\ W^H U & \bar{V}^H W \end{bmatrix}$ % eigCG with initial guess x_0
% (Not strictly necessary)
% Add nev rows to H
Set $U = [U,\ \bar{V}]$ % Augment U
end

- Most of the cost is in Rayleigh-Ritz
- Needs n_{ev} MatVec operations and several dot products
- Ultimate size of U is determined by how much cost you can amortize for a given number of right hand sides
- Lots of the flops can be done efficiently using level 3 BLAS

EigCG for QCD

N_{ev}=10, m=100

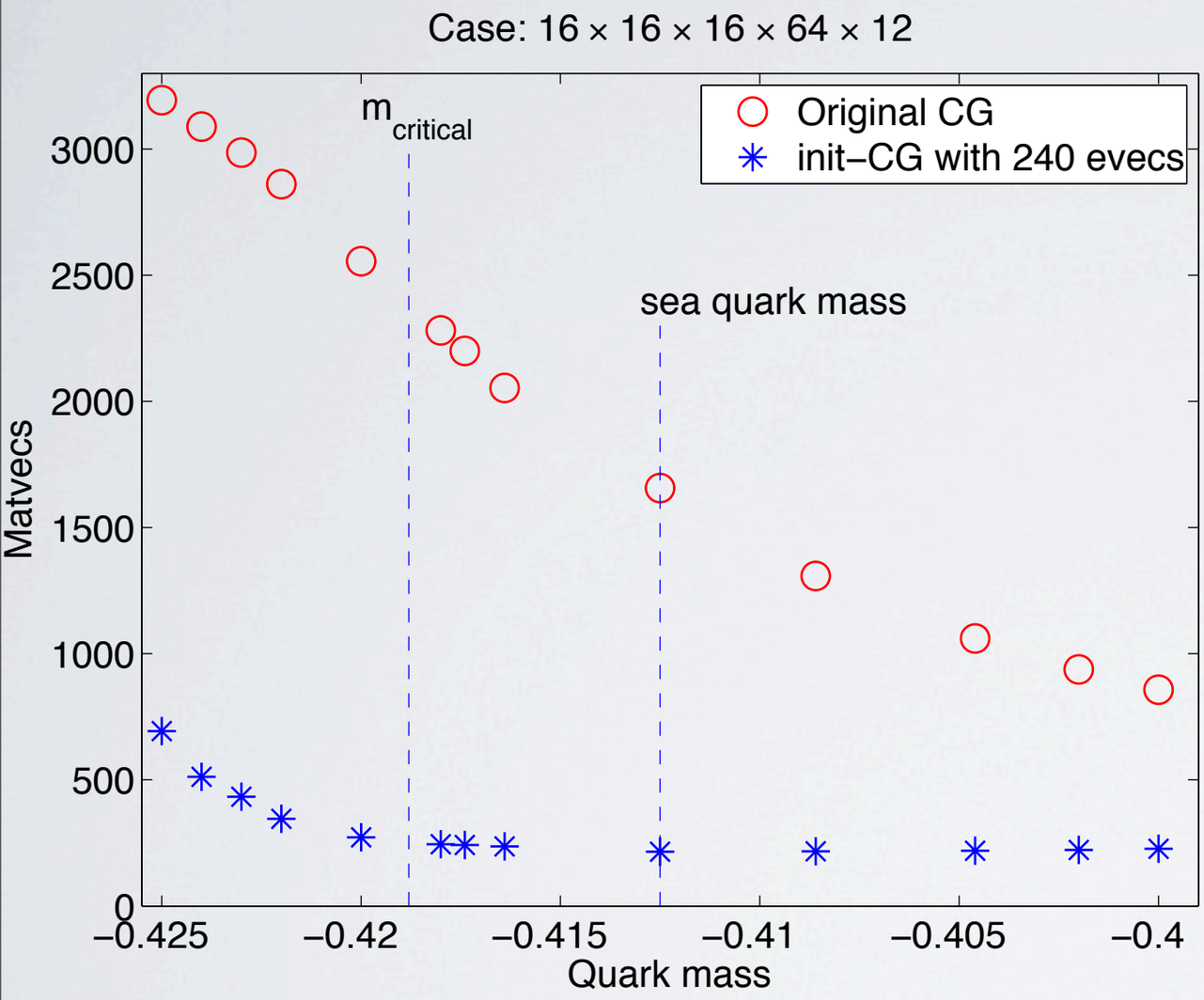


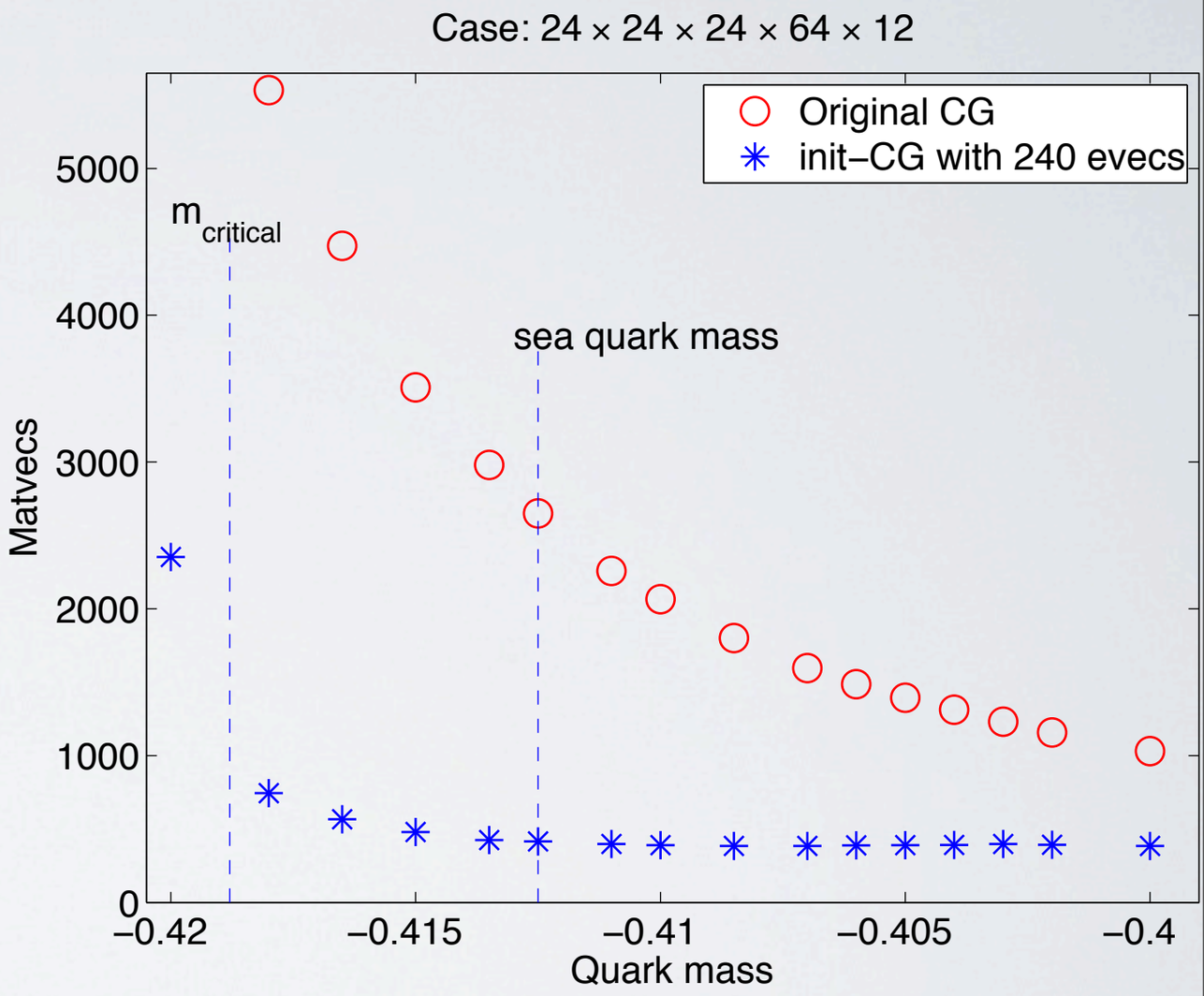


- Valence Quark mass equals to sea quark mass

Critical slowing down (or lack off)

EigCG [Stathopoulos and KO **arXiv:0707.0131**]





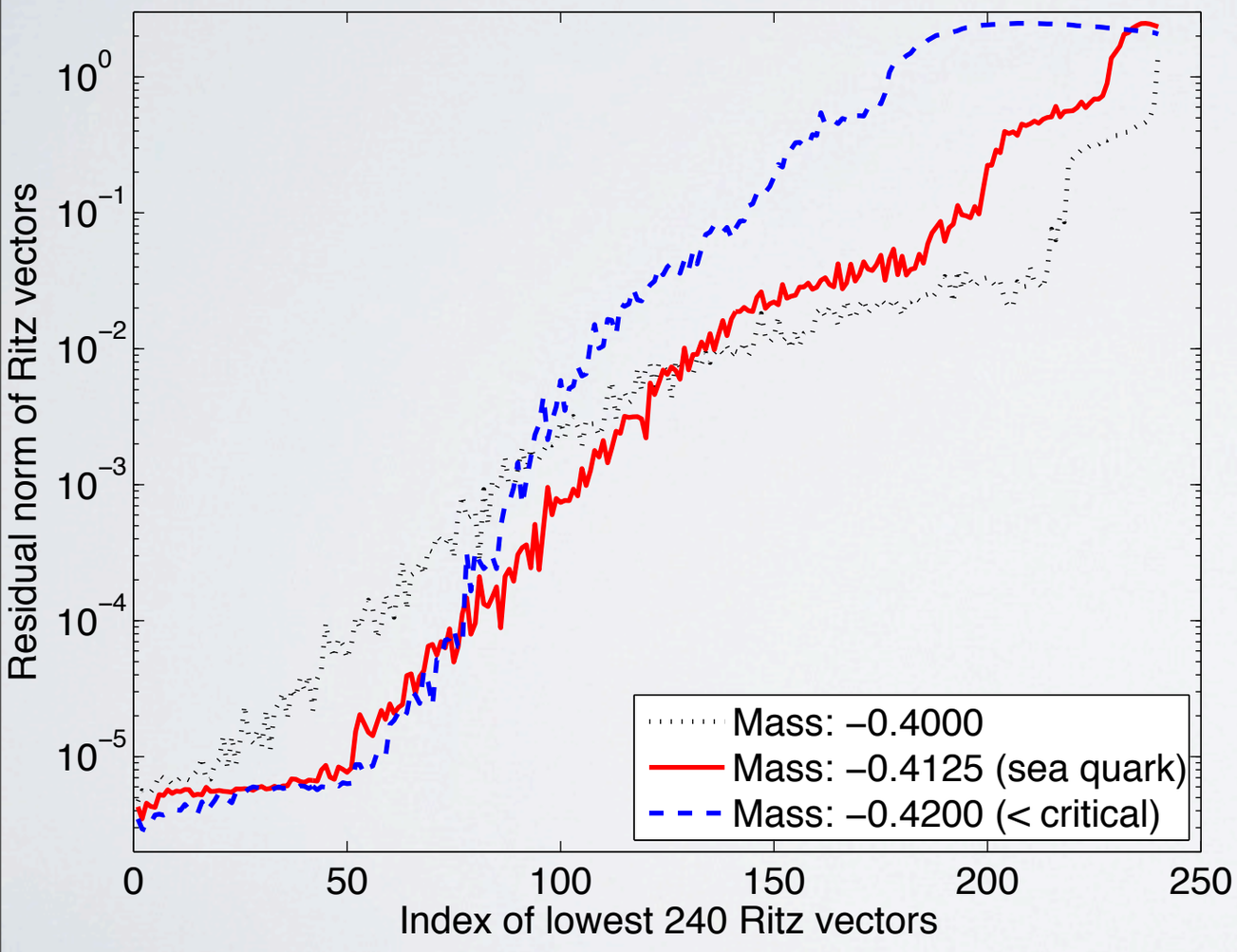
For sufficiently large number of right hand sides:

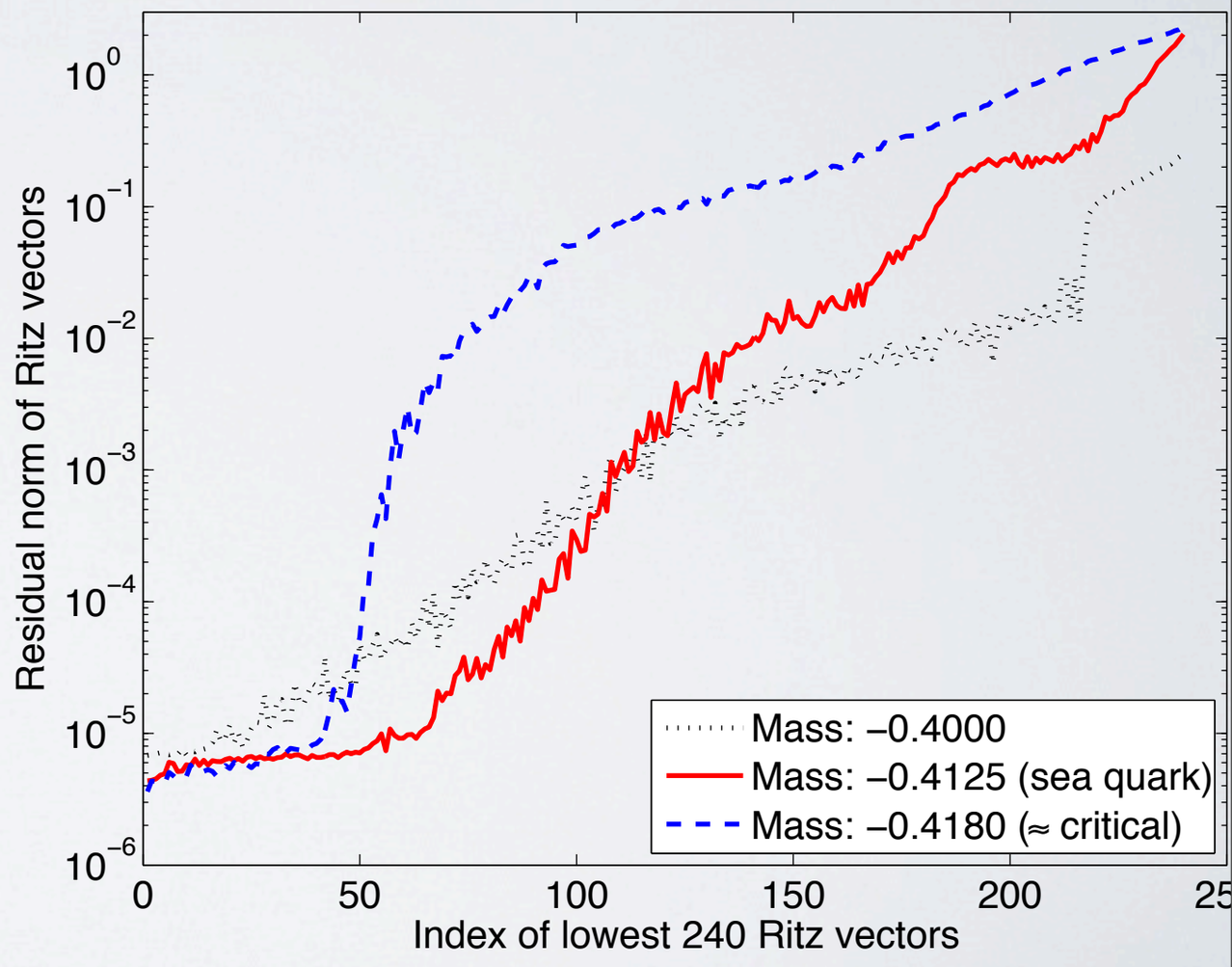
- Small volume factor of 8 speed up at $m_{sea} = m_{val}$
- Large volume factor of 6 speed up at $m_{sea} = m_{val}$

Numerical tests ran at NERSC (franklin) and the cyclades cluster at W&M

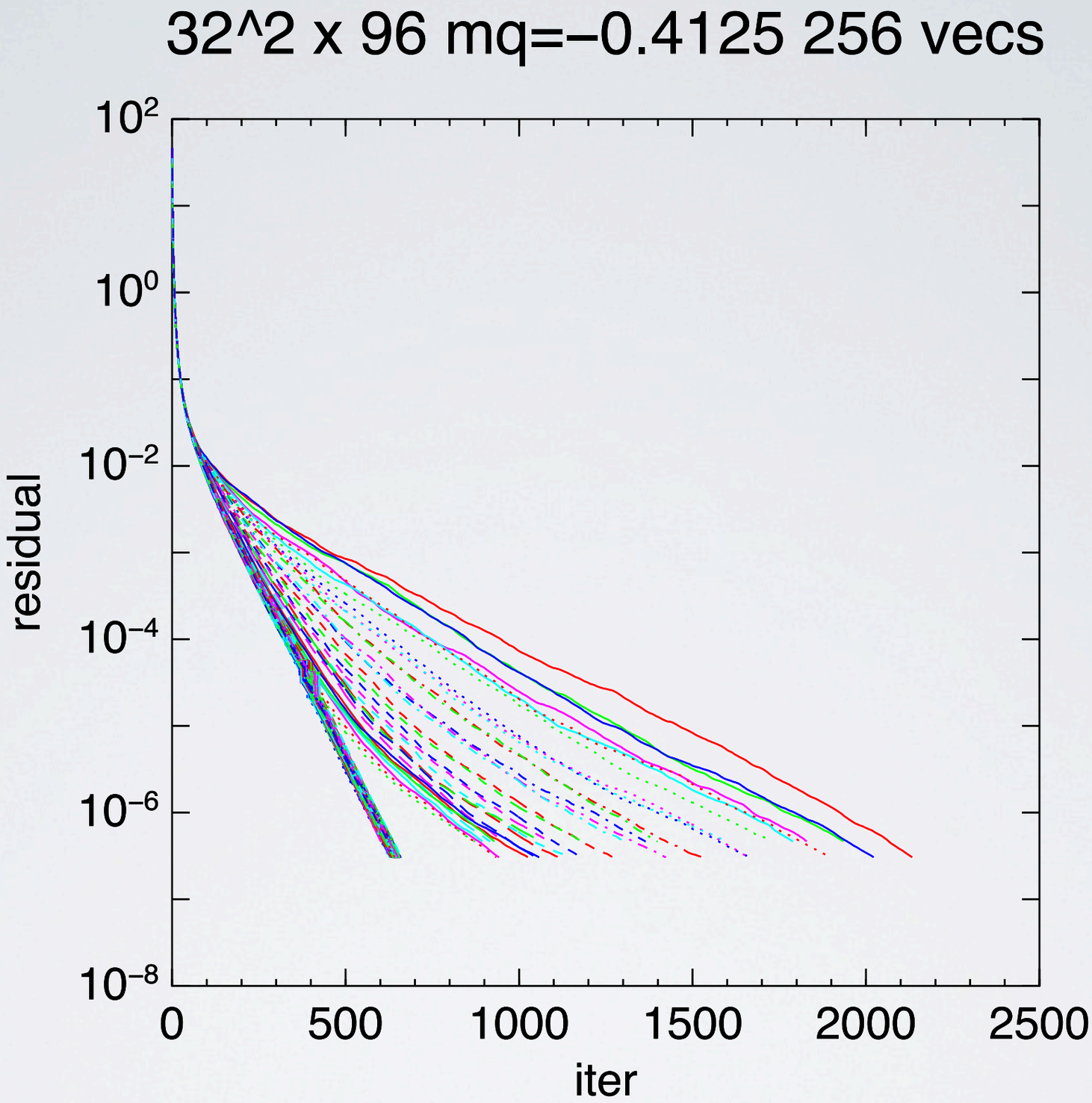
EIGENVECTOR ACCURACY

$$R = \|Ae - \lambda e\|$$

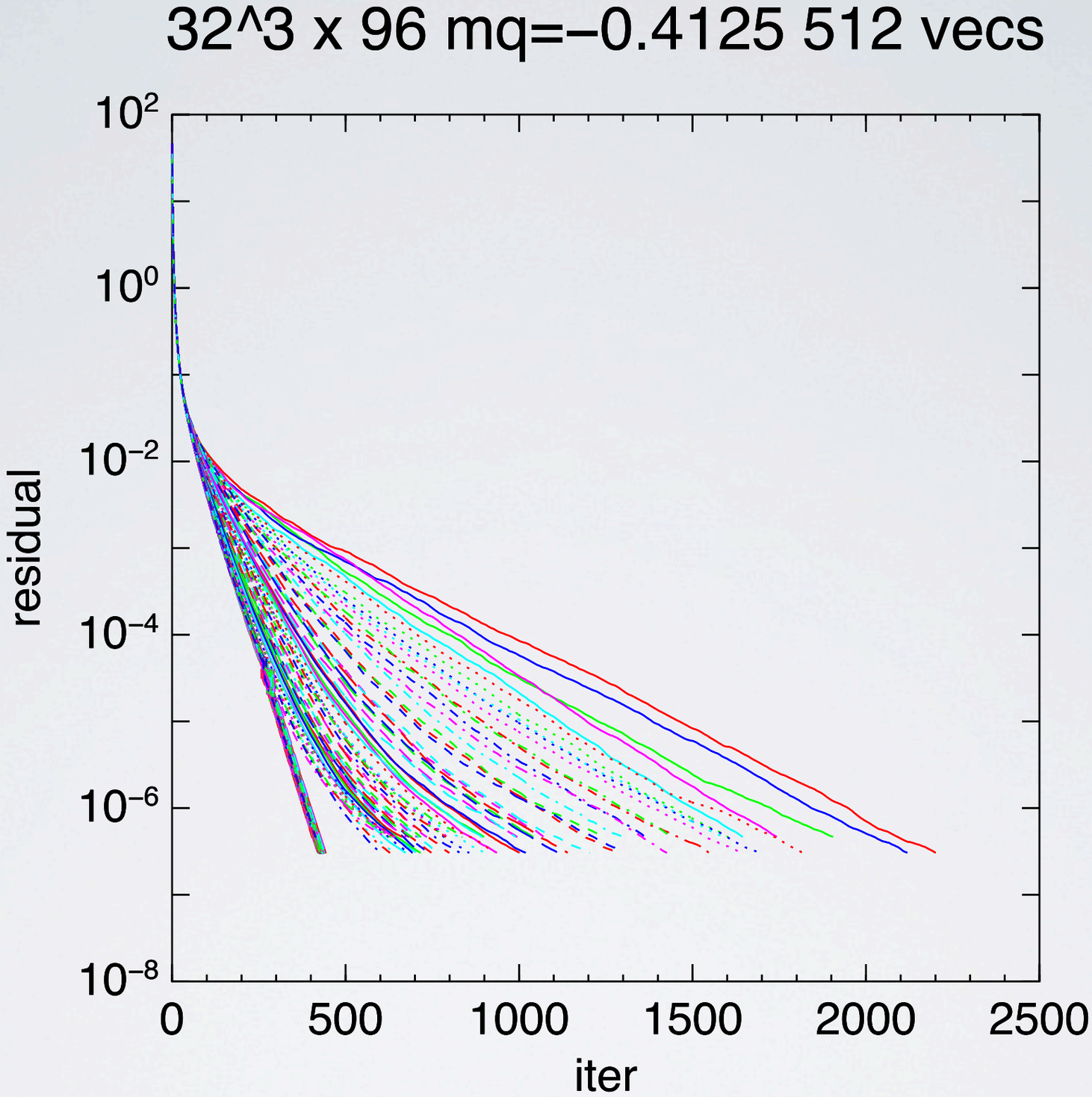




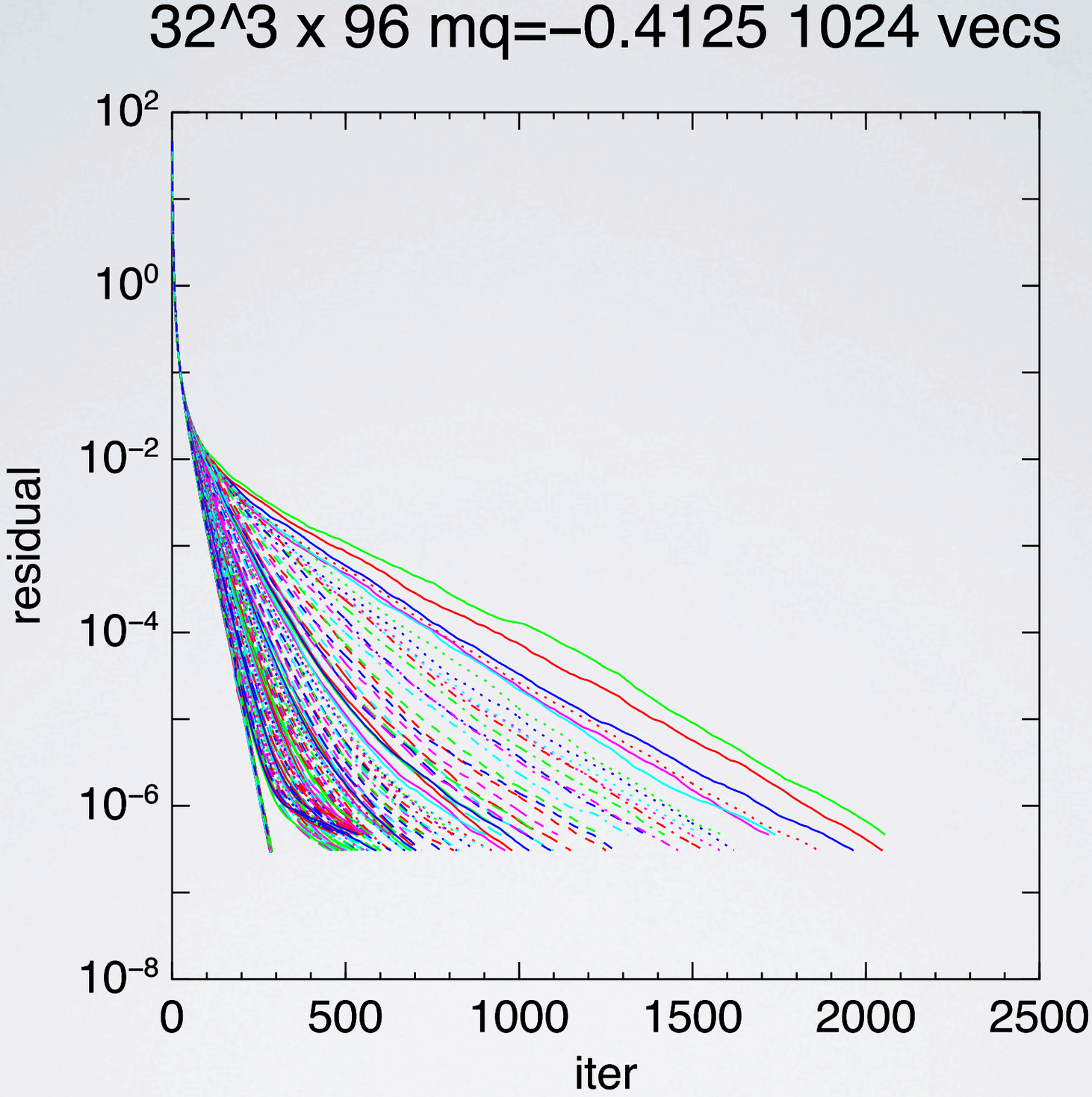
- For light masses we get O(50) eigenvectors to single precision accuracy



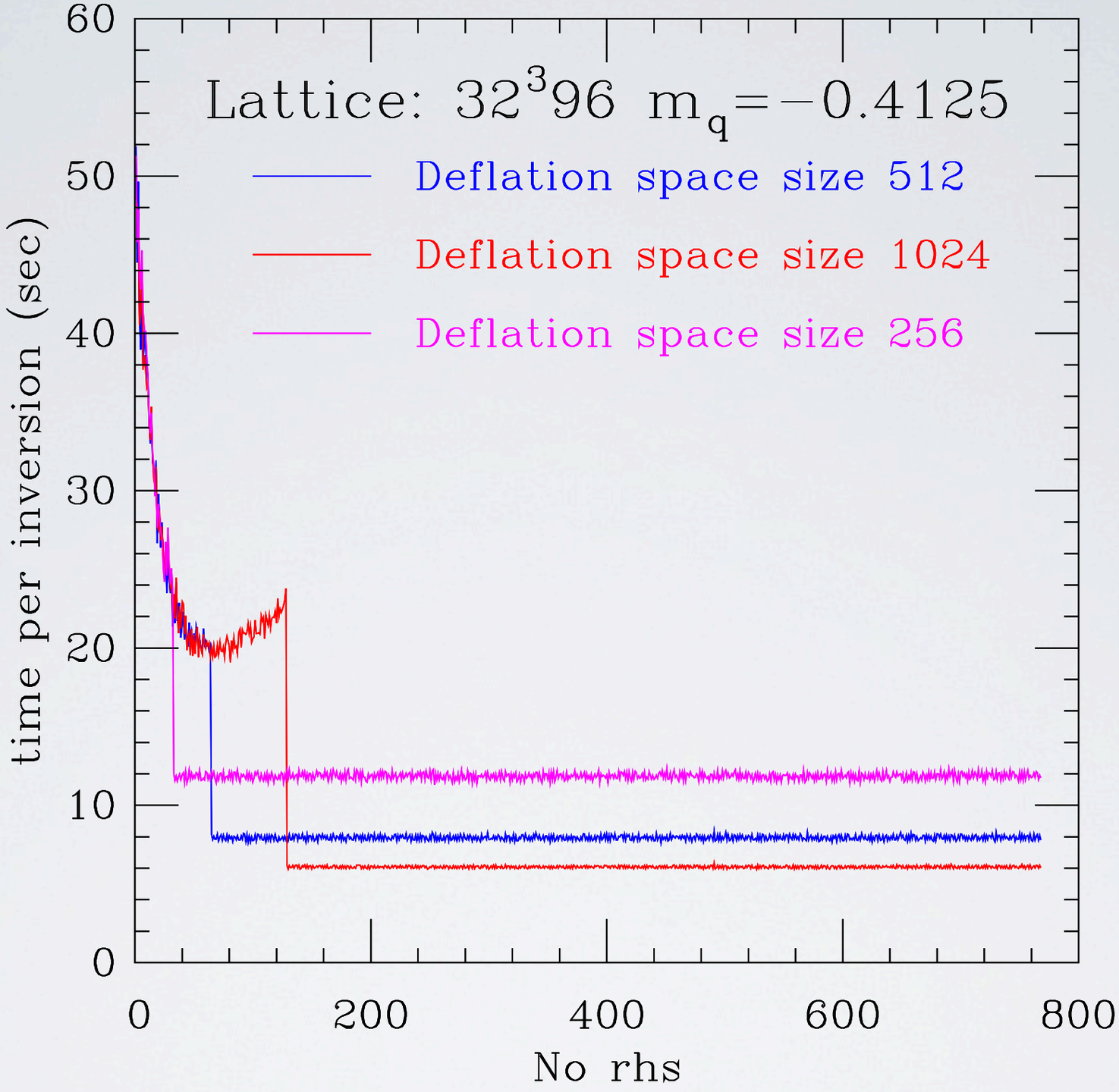
Large Lattice Tests



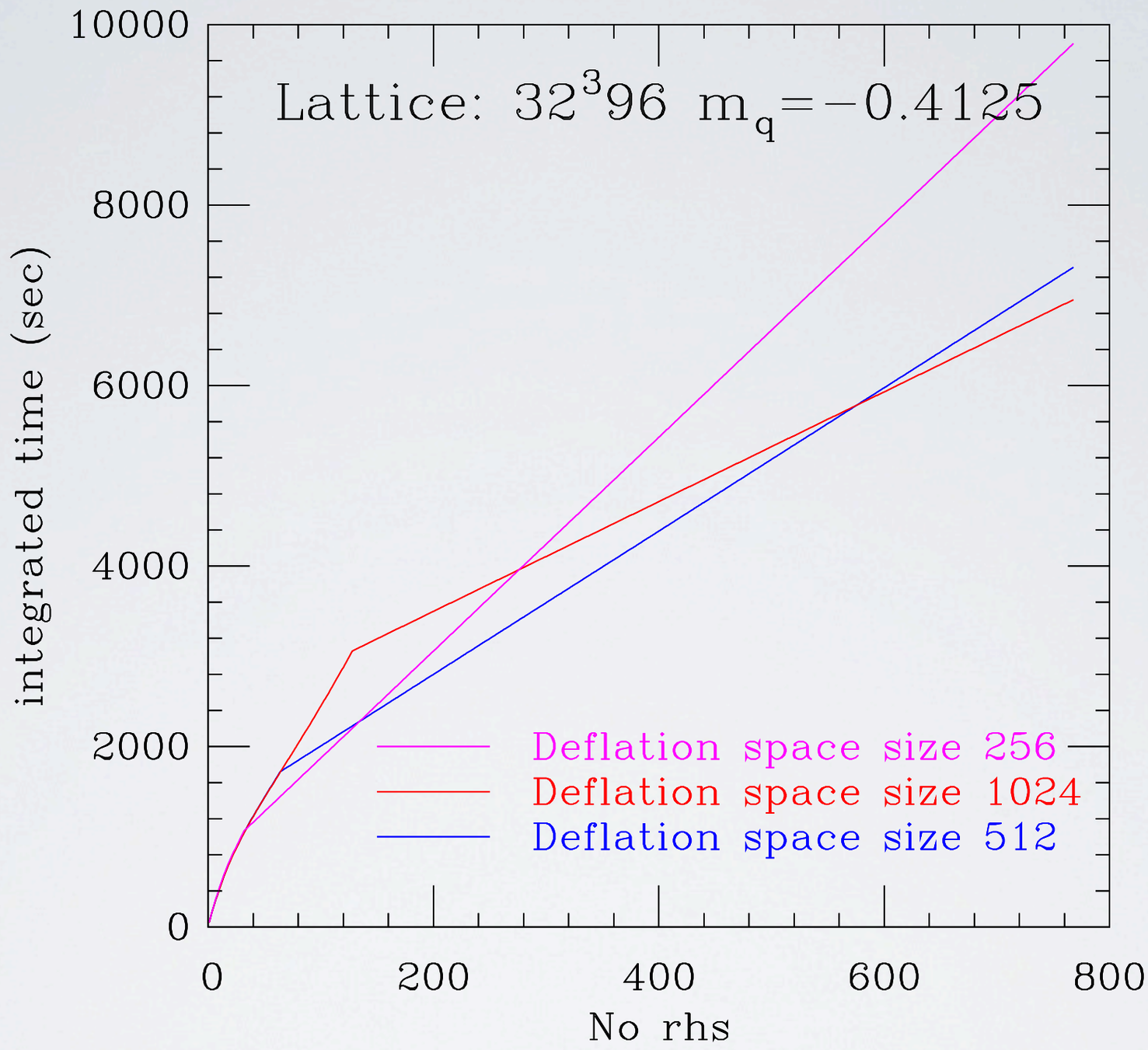
Large Lattice Tests



Large Lattice Tests



Large Lattice Tests



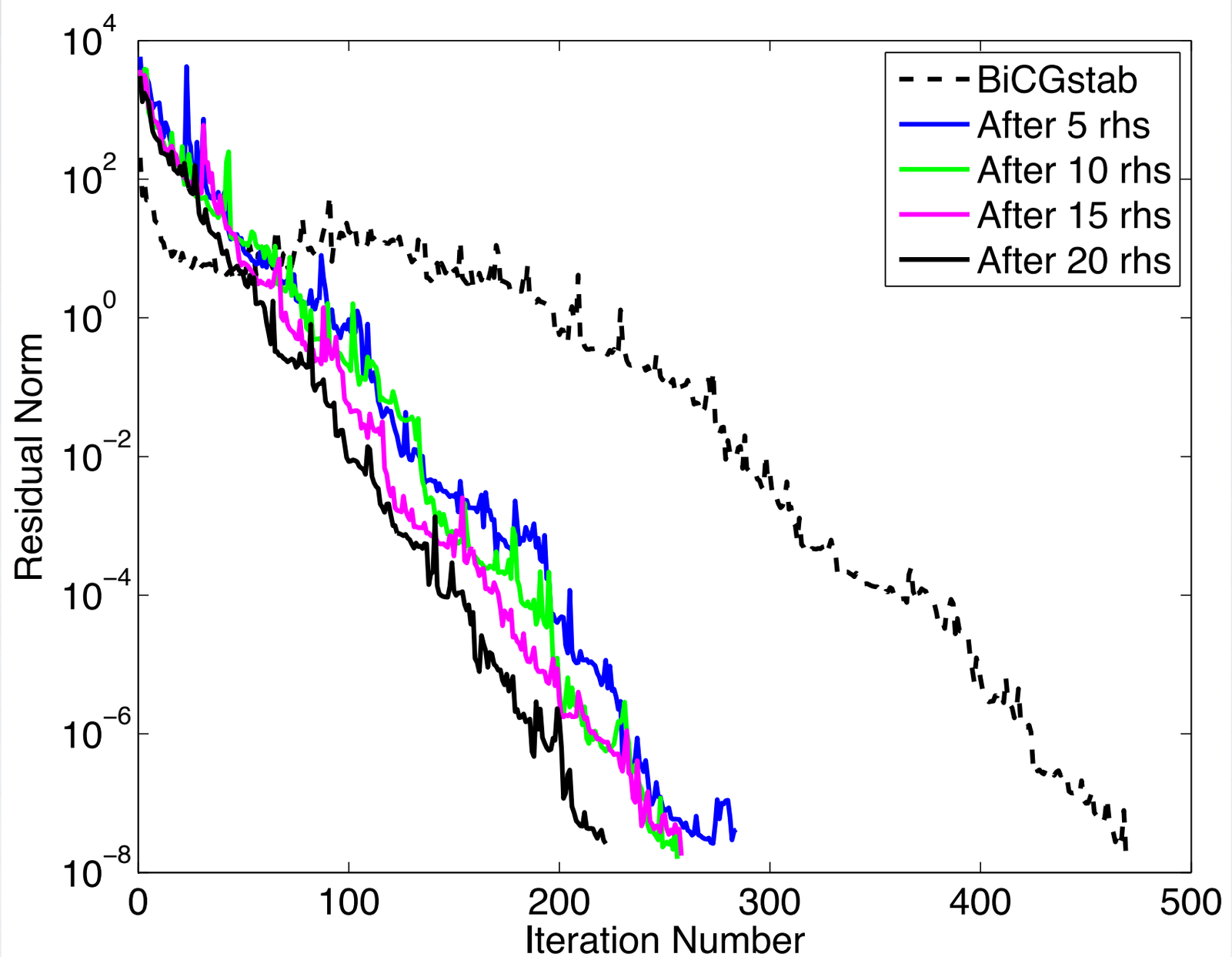
Large Lattice Tests

EigBiCG

Developed with A. Rehim A. Stathopoulos (W&M)

- Same idea as EigCG applied to BiCG
- Exploits the bi-Lanczos algorithm to built a deflation subspace while solving linear systems
- Use BiCGstab for steady state inverter
- BiCGstab has good performance for certain LQCD problems

EigBiCG on a 12^4 lattice ($N=2.5\times10^5$)



plot by A. Rehim

OTHER DEFLATION ALGORITHMS

- GMRES-DR [Morgan, Wilcox et al **arXiv:math-ph/0405053 arXiv:0707.0502 arXiv:0707.0505**]
- Luscher's Schwarz pre-conditioner (Domain-Decomposition)

  see talk by Balint Joo
- Multigrid

  see talks by Brower, Falgout, and Cohen

CONCLUSIONS

- Numerical linear algebra algorithms play an important role in Lattice QCD calculations
- Improvements by orders of magnitude have been made by careful tuning and innovative ideas
- Still a lot needs to be done
- In some cases reformulation of the problem might become more important than the implementation details of existing methodologies