State-of-the-art numerical solution of large Hermitian eigenvalue problems

Andreas Stathopoulos

Computer Science Department and Computational Sciences Cluster

College of William and Mary

Acknowledgment: National Science Foundation

The problem

Find `numEvals` eigenvalues $\tilde{\lambda}_i$ and corresponding eigenvectors $\tilde{x}_i$

$$\boxed{A\tilde{x}_i = \tilde{\lambda}_i\tilde{x}_i, \quad i = 1 : \texttt{numEvals}}$$

A is large, sparse, symmetric

$$N = O(10^6 - 10^8)$$

Applications: materials, structural, data mining, SVD, QCD, ...

QCD

Accelerate linear systems with multiple right hand sides

Low rank approximation of matrices

Only possible through iterative methods

Power method: the fundamental iterative method

Given initial guess v_0, the iteration

```
for $i = 1, 2, \ldots$
    $t = Av_{i-1}$
    $v_i = t / \|t\|$
```

converges to the largest modulus eigenpair $(\tilde{\lambda}_N, \tilde{x}_N)$, i.e.,

$$\frac{A^i v_0}{\|A^i v_0\|} \longrightarrow \tilde{x}_N, \quad \text{with rate} \quad \frac{\tilde{\lambda}_{N-1}}{\tilde{\lambda}_N}$$

$+$ Requires only matrix-vector multiplications

– Only for largest eigenpair

– Slow!

Krylov methods: the prevailing technique

Krylov space consists of the span of all power iterates:

$$\begin{aligned} \mathcal{K}_{m,v} &= span\left\{v, Av, A^2v, \ldots, A^{m-1}v\right\} \\ &= \{p(A)v : \forall p \text{ polynomial of degree} < m\} \end{aligned}$$

Compute V an orthonormal basis for $\mathcal{K}_{m,v}$ (for numerical stability)

Compute approximations through Rayleigh-Ritz:

$$x_i = Vy_i, \quad \text{where} \quad V^T AV y_i = \lambda_i y_i$$

Arnoldi: the above process for non-symmetric matrices

Lanczos: a special case of Arnoldi for symmetric matrices

Krylov methods: the prevailing technique

$$\begin{aligned}\mathcal{K}_{m,v} &= span\left\{v, Av, A^2v, \ldots, A^{m-1}v\right\} \\ &= \{p(A)v : \forall p \text{ polynomial of degree} < m\}\end{aligned}$$

+ Approximating extreme eigenpairs

+ Converges trivially in N steps

+ Optimal approximations over all polynomials

– Convergence rate depends on relative separation of eigenvalues

– Slow for clustered eigenvalues and large sizes

– $O(Nm^2)$ orthogonalization cost, $O(mN)$ storage

Krylov ideal for linear systems

$$\boxed{Ax = b}$$

Conjugate Gradient (CG) uses a 3-term recurrence to build $\mathcal{K}_{m,v}$ and update the approximate solution.

- $O(Nm)$ cost and $O(3N)$ storage
- minimizes $\|error\|_A$ at every step
- Preconditioning with $M^{-1} \approx A^{-1}$ also easy: $M^{-1}Ax = M^{-1}b$ (PCG)

Krylov ideal for linear systems

$$\boxed{Ax = b}$$

Conjugate Gradient (CG) uses a 3-term recurrence to build $\mathcal{K}_{m,v}$ and update the approximate solution.

- $O(Nm)$ cost and $O(3N)$ storage
- minimizes $\|error\|_A$ at every step
- Preconditioning with $M^{-1} \approx A^{-1}$ also easy: $M^{-1}Ax = M^{-1}b$ (PCG)

Note: The action of M^{-1} could be an iterative method itself!

Lanczos problems

- Lanczos 3-term recurrence still requires $O(Nm)$ storage
- Unlike CG, orthogonality is important in Lanczos

$$\Downarrow$$

- Restarting to limit the basis size destroys optimality
- Preconditioning is not obvious ($M^{-1}Ax = \lambda M^{-1}x$ not an eigenproblem)

Goal: Use PCG to derive nearly optimal eigenmethods with smaller bases

Generalized Davidson: Eigenvalue Preconditioning

Let $r = Ax - \lambda x$ the residual of an approximate eigenpair (λ, x)

Arnoldi/Lanczos: expand basis V by r.

Generalized Davidson: expands by the preconditioned r:

```
Generalized Davidson
repeat
    $V = [V,\ M^{-1}r]$                          append preconditioned residual
    $V^*AVy = \lambda y, \quad x = Vy$            Rayleigh Ritz
    $x = x/\|x\|$                                normalize
    $r = Ax - \lambda x$                         new residual
until $\|r\| < \varepsilon$
```

No 3-term recurrence, more expensive step, but much faster convergence

Inverse iteration (inverse power method)

Given initial guess v_0, the iteration

```
for $i = 1, 2, \ldots$
    $t = (A - \sigma I)^{-1} v_{i-1}$
    $v_i = t / \|t\|$
```

converges to the eigenpair closest to σ

+ The closer σ is to $\tilde{\lambda}_k$ the faster the outer convergence rate $\frac{\tilde{\lambda}_k - \sigma}{\tilde{\lambda}_{k-1} - \sigma}$

– A direct factorization of A may be prohibitive

– An iterative method for the linear system may take long to converge

Rayleigh Quotient Iteration

Given initial guess v_0:

```
for $i = 1, 2, \ldots$
    $t = (A - \sigma I)^{-1} v_{i-1}$
    $v_i = t / \|t\|$
    $\boxed{\sigma = v_{i-1}^T A v_{i-1}}$
```

All the characteristics of Inverse Iteration but also:

$+$ converges to the eigenpair **cubically**!!

$-$ if v_0 not close to the required eigenvector it may misconverge

Rayleigh Quotient Iteration

Given initial guess v_0:

```
for $i = 1, 2, \ldots$
    $t = (A - \sigma I)^{-1} v_{i-1}$
    $v_i = t / \|t\|$
    $\boxed{\sigma = v_{i-1}^T A v_{i-1}}$
```

All the characteristics of Inverse Iteration but also:

$+$ converges to the eigenpair **cubically**!!

$-$ if v_0 not close to the required eigenvector it may misconverge

Eigenproblem: constrained minimization of Rayleigh quotient $\lambda = \frac{\mathbf{x}^\mathbf{T}\mathbf{A}\mathbf{x}}{\mathbf{x}^\mathbf{T}\mathbf{x}}$

RQI equivalent to Newton on the unit-sphere manifold

Inexact RQI, Newton, and Jacobi-Davidson

$(A-\sigma I)t = v_{i-1}$ must be solved accurately enough for RQI to converge

However, inexact (truncated) Newton does not require high accuracy

Newton:	$x_{i+1} = x_i - Hess(x_i)^{-1}\nabla(x_i)$	computes correction
RQI:	$x_{i+1} = (A-\sigma I)^{-1}x_i$	updates approximation

$$\Downarrow$$

inexact RQI not exactly inexact Newton!

Inexact RQI, Newton, and Jacobi-Davidson

$(A-\sigma I)t = v_{i-1}$ must be solved quite accurately for RQI to converge

However, inexact (truncated) Newton does not require high accuracy

Newton:	$x_{i+1} = x_i - Hess(x_i)^{-1}\nabla(x_i)$	computes correction
RQI:	$x_{i+1} = (A-\sigma I)^{-1}x_i$	updates approximation

$$\Downarrow$$

inexact RQI not exactly inexact Newton!

Note $\nabla(x) = -r = -(Ax-\lambda x)$ the residual of (λ, x). Thus the correction δ to x:

Jacobi-Davidson: $(I-xx^T)(A-\eta I)(I-xx^T)\delta = r$ computes correction

JD ⇔ inexact (truncated) Newton

Equivalence in the general case

Let $M \approx A - \sigma I$ a preconditioner

Both GD/JD solve approximately the correction equation:

Generalized Davidson as $\delta = M^{-1} r$

Jacobi Davidson as $\delta = M^{-1}|_{x^\perp}\, r$

Both GD/JD not single vector iterations, they build a space!

GD, JD $\Longleftrightarrow$ subspace-accelerated inexact Newton

A different view: Quasi-Newton approaches

Mild non-linearity:

- Nonlinear CG is competitive [Bradbury & Fletcher, ’66, Others]
- Better: locally optimal LOBPCG [D’yakonov ’83, Knyazev, ’91, ’01]

$$\boxed{x_{i+1} = \text{Rayleigh\_Ritz}\left(\, x_{i-1},\ x_i,\ M^{-1} r_i \,\right)}$$

A different view: Quasi-Newton approaches

Mild non-linearity:

- Nonlinear CG is competitive [Bradbury & Fletcher, ’66, Others]
- Better: locally optimal LOBPCG [D’yakonov ’83, Knyazev, ’91, ’01]

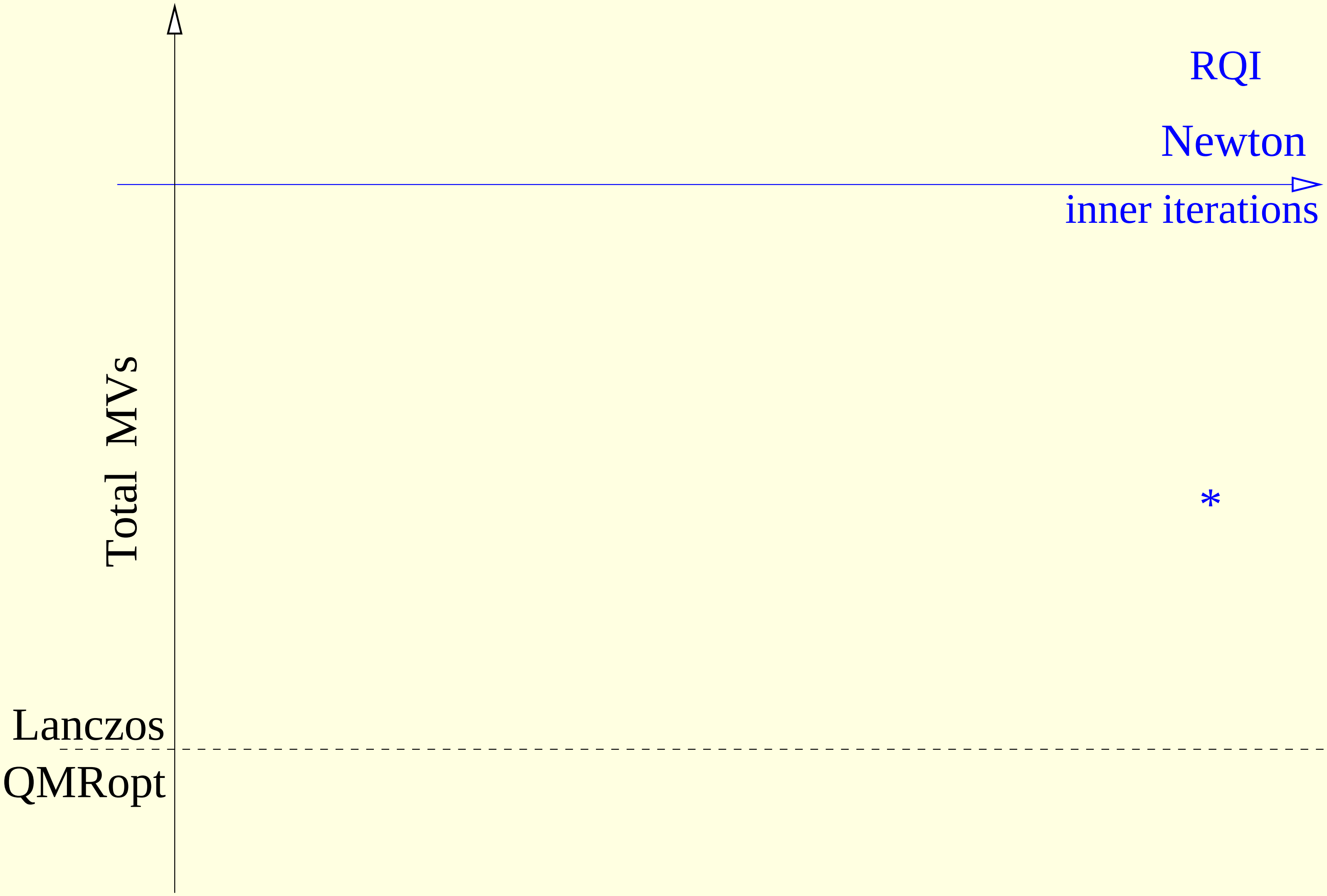
$$\boxed{x_{i+1} = \text{Rayleigh\_Ritz}\left(\, x_{i-1},\ x_i,\ M^{-1} r_i \,\right)}$$

- Subspace acceleration and recurrence restarting in GD [Murray et al., ’92]
- GD(k,m)+1: Restart with $[x_{i-1},\ x_i^1, \ldots, x_i^k]$ [AS ’98, ’99]

Direct analogy to limited memory quasi Newton methods:

GD+1 accelerates LOBCPG ⟷ Broyden accelerates Nonlinear CG

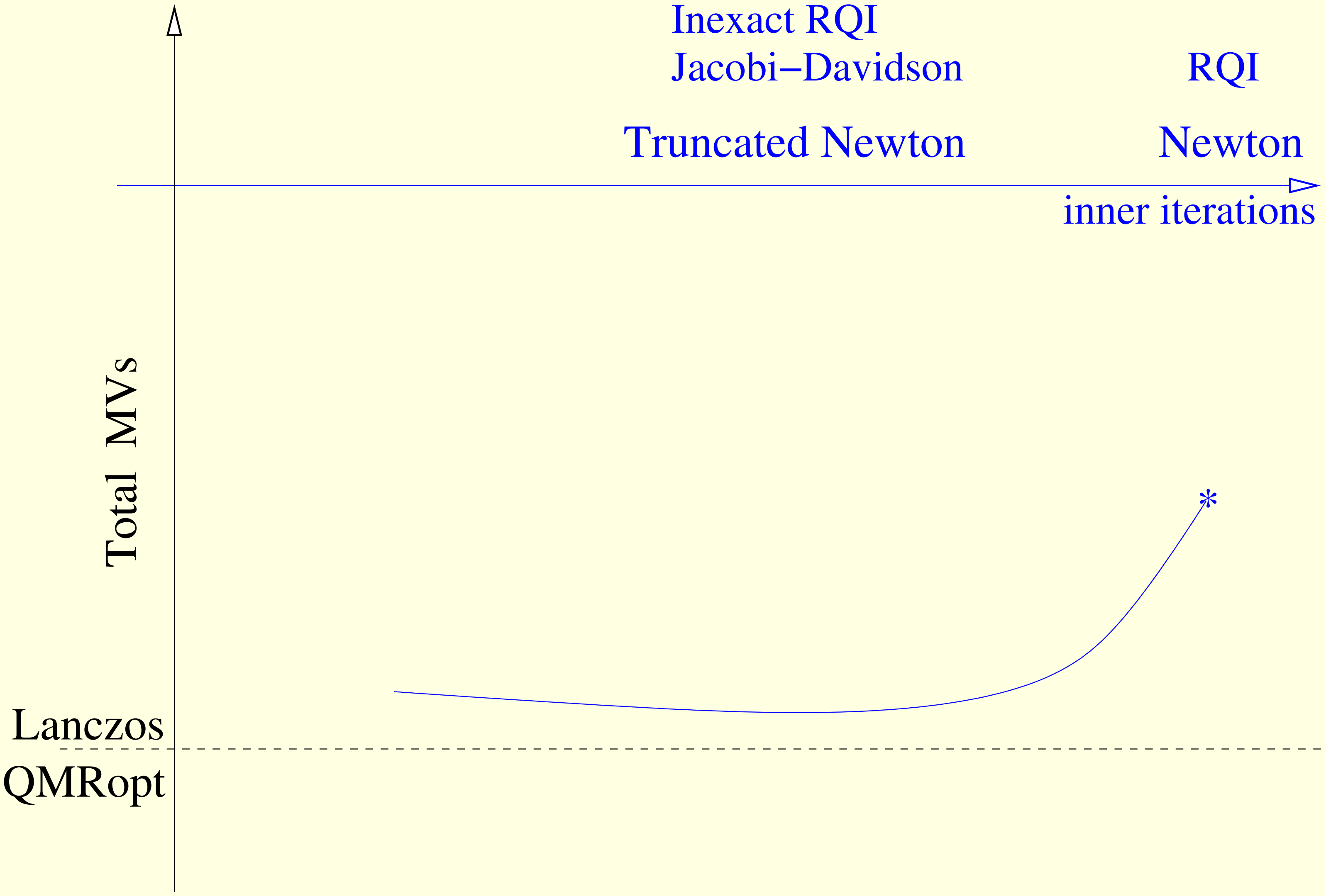
So what is optimal? Work unit: Matrix-Vector



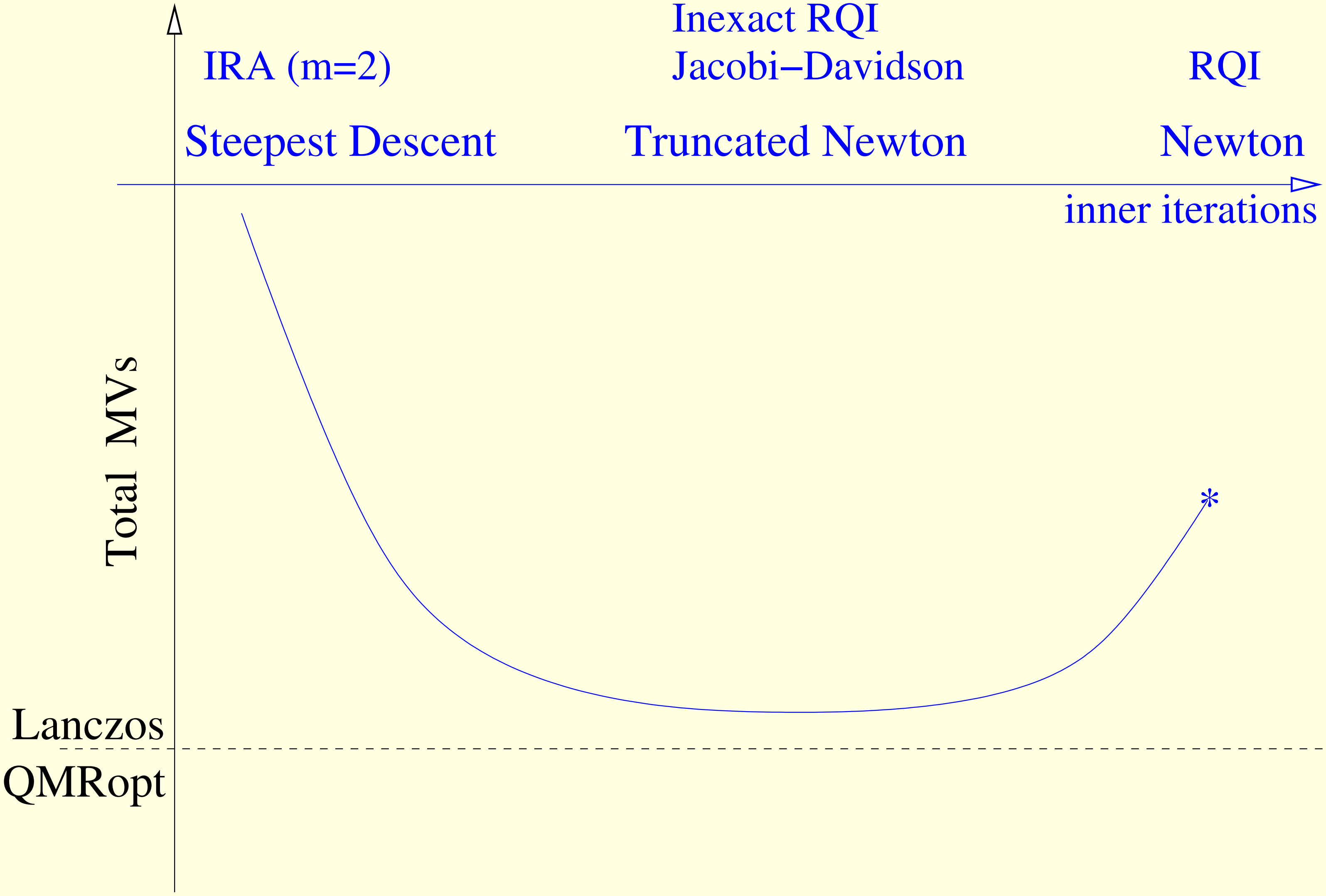
Optimal: Unrestarted Lanczos or QMRopt, QMR solving $(A - \tilde{\lambda} I)x = 0$

So what is optimal?

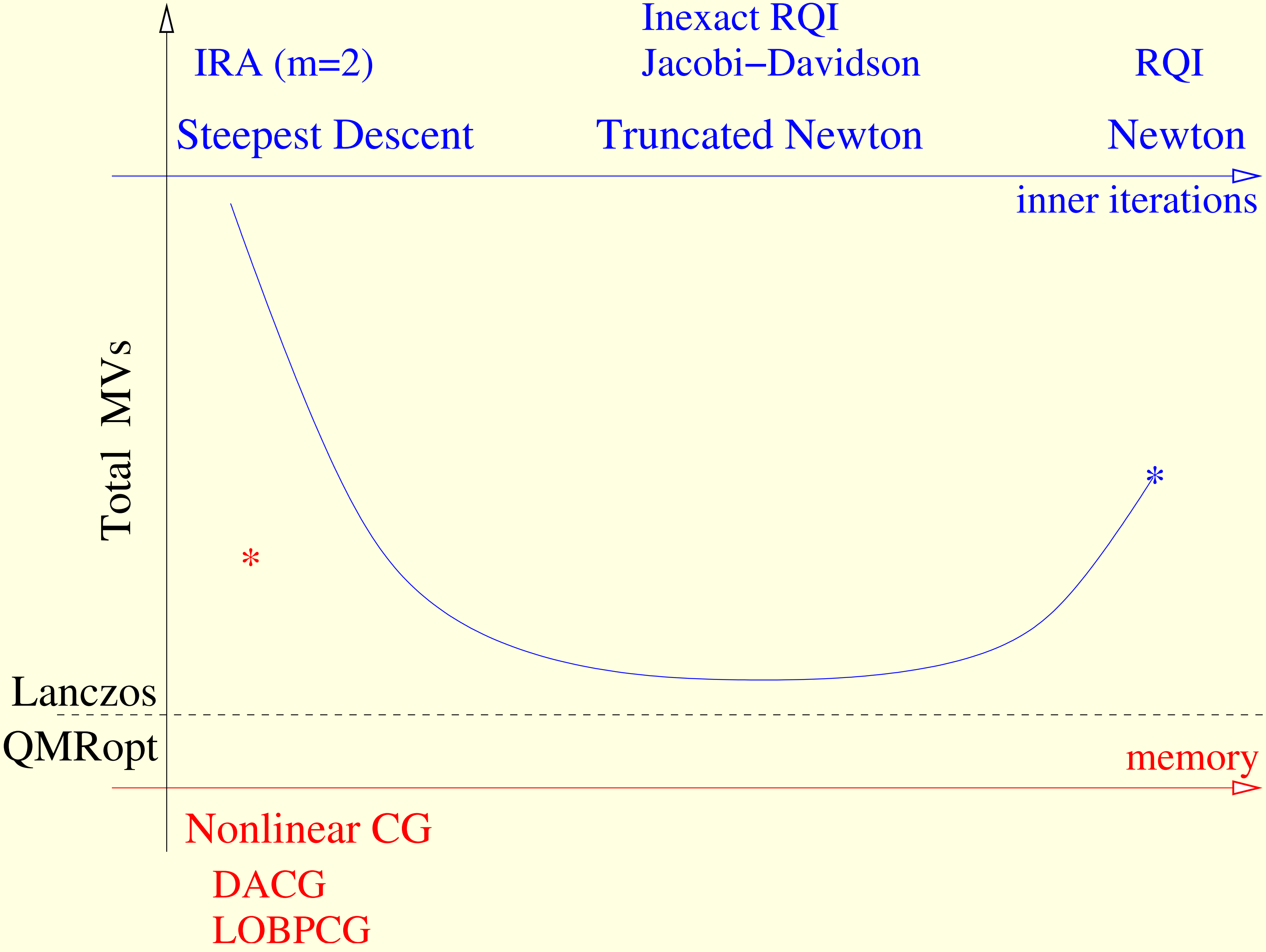
Work unit: Matrix-Vector



So what is optimal? Work unit: Matrix-Vector



So what is optimal? Work unit: Matrix-Vector



IRA (m=2)

Inexact RQI

Jacobi–Davidson

RQI

Steepest Descent

Truncated Newton

Newton

inner iterations

Total MVs

Lanczos

QMRopt

memory

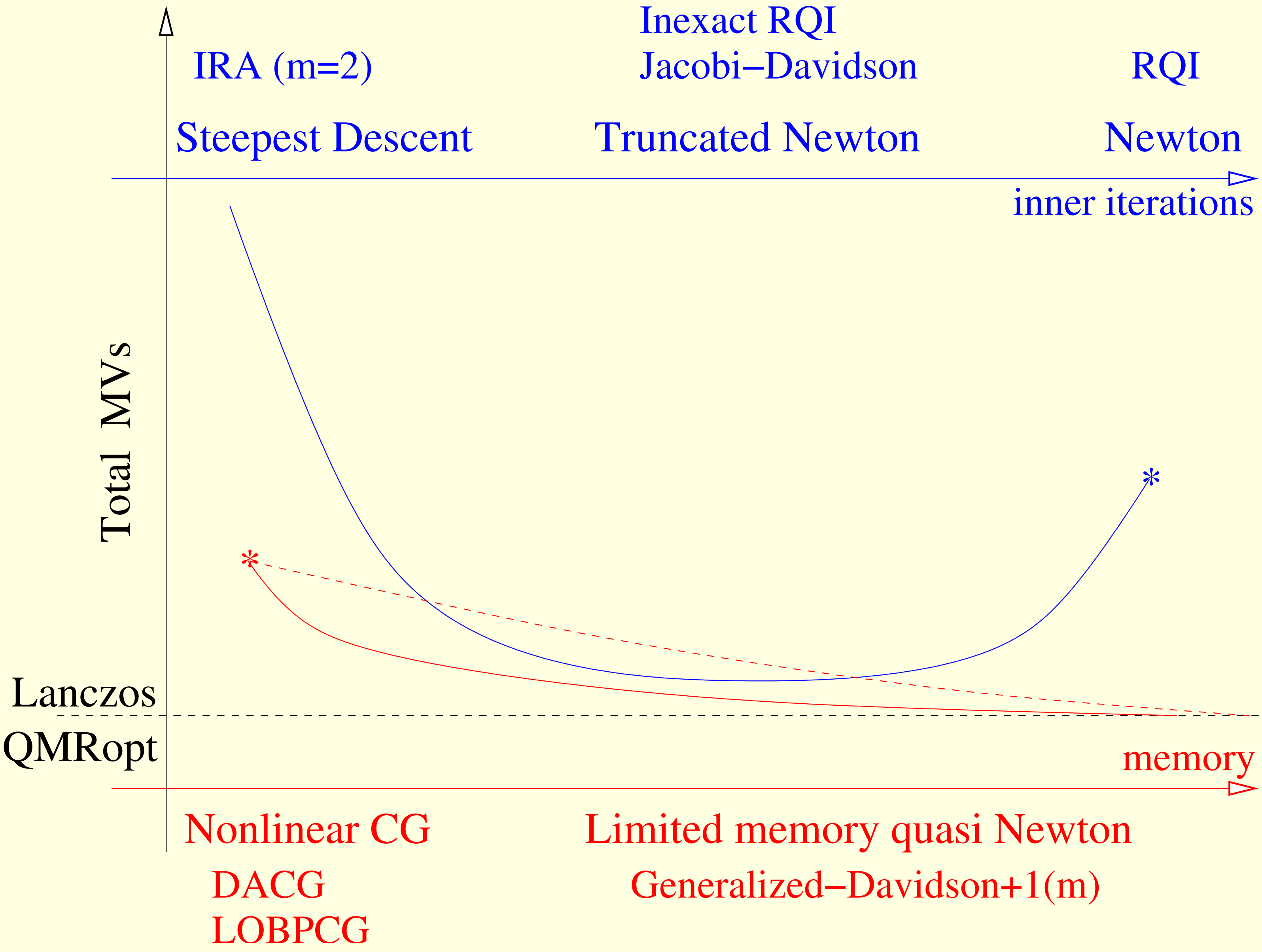
Nonlinear CG

DACG

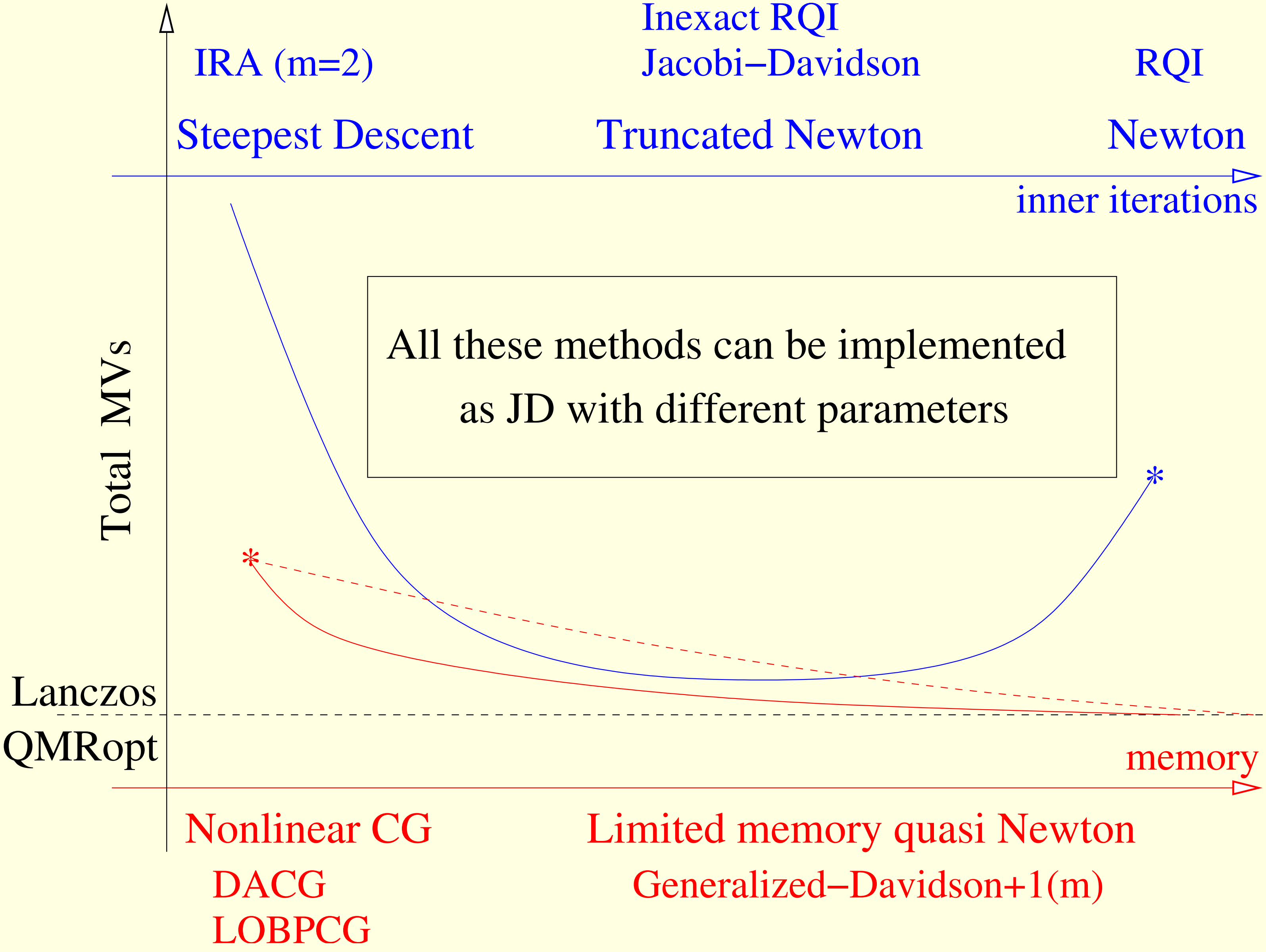
LOBPCG

So what is optimal?

Work unit: Matrix-Vector



So what is optimal? Work unit: Matrix-Vector



Inexact RQI
Jacobi−Davidson

IRA (m=2)

RQI

Steepest Descent

Truncated Newton

Newton

inner iterations

All these methods can be implemented as JD with different parameters

Total MVs

Lanczos

QMRopt

memory

Nonlinear CG

DACG

LOBPCG

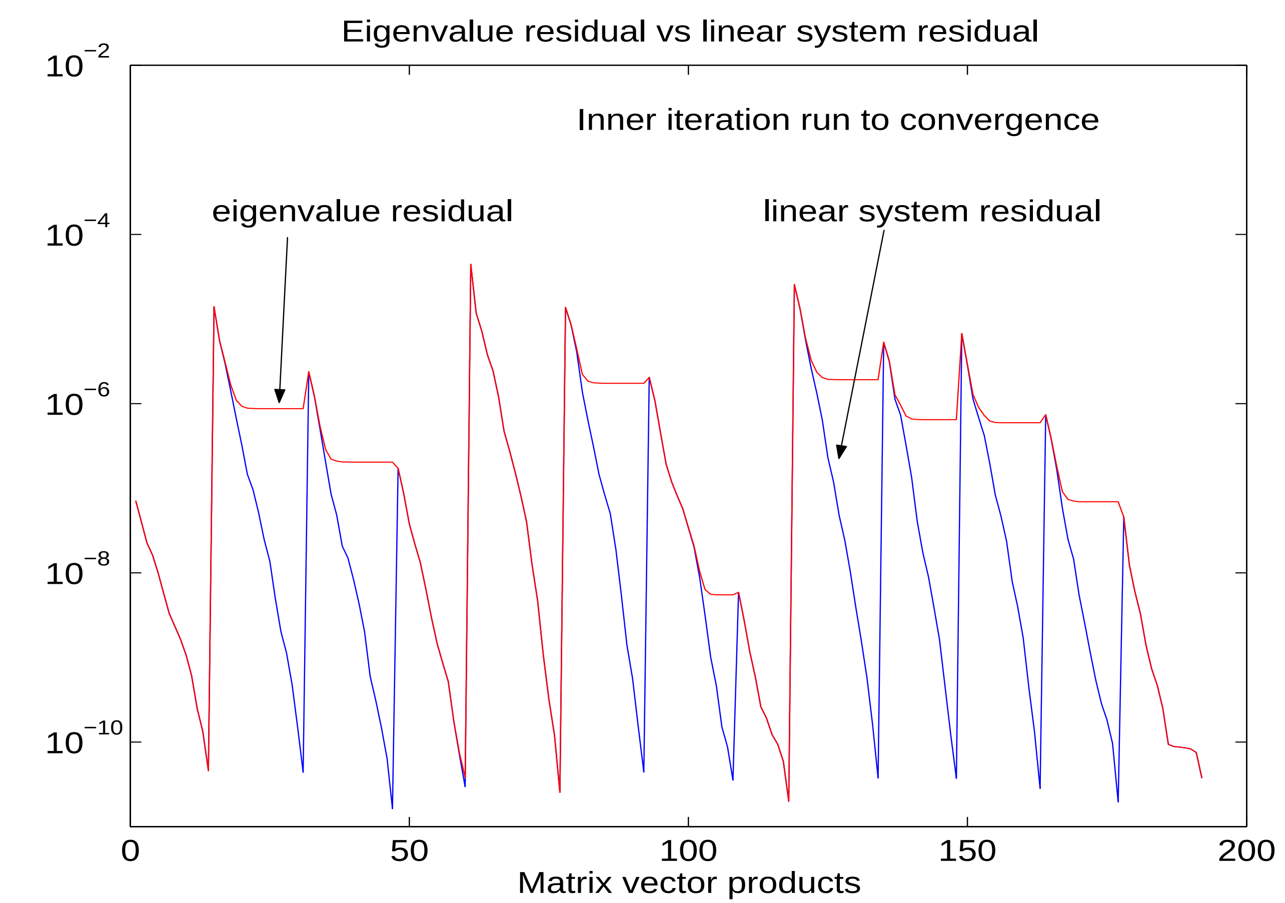
Limited memory quasi Newton

Generalized−Davidson+1(m)

Stopping the inner JD iteration (Notay 02)

$$(I - xx^T)(A - \eta I)(I - xx^T)\delta = r$$

$$\|r_{eigen}\|^2 = \underset{\displaystyle 0}{\underset{\downarrow}{\|r_{Linear}\|^2}}/f + \underset{\text{Inverse iteration residual}}{\underset{\downarrow}{\|g_k\|^2}}$$



Our JDQMR extension to JDCG

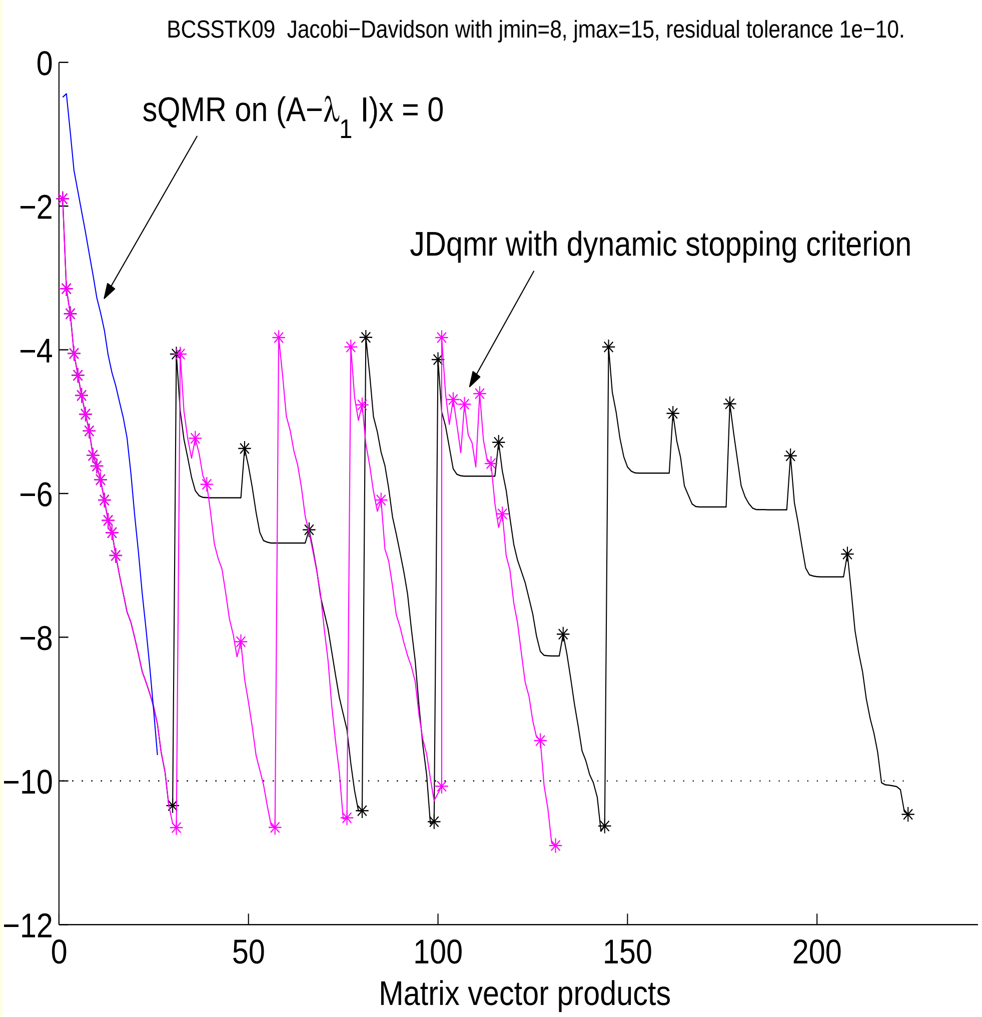
Based on symmetric QMR [Freund & Nachtigal 94] with right preconditioning

JDQMR new features

1. Can use indefinite preconditioners
2. Works for interior eigenpairs
3. Residual convergence smooth
4. Better stopping criteria

JDQMR improves robustness and efficiency

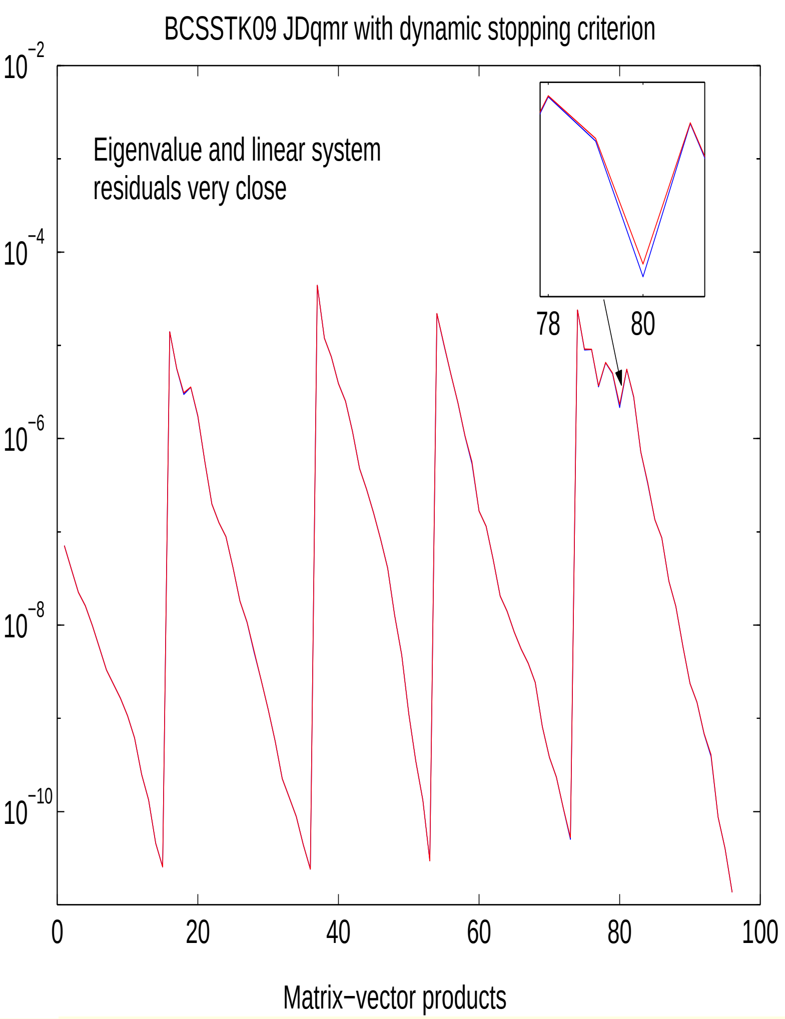
JDQMR reduces wasted iterations



BCSSTK09 Jacobi−Davidson with jmin=8, jmax=15, residual tolerance 1e−10.

sQMR on (A−λ_1 I)x = 0

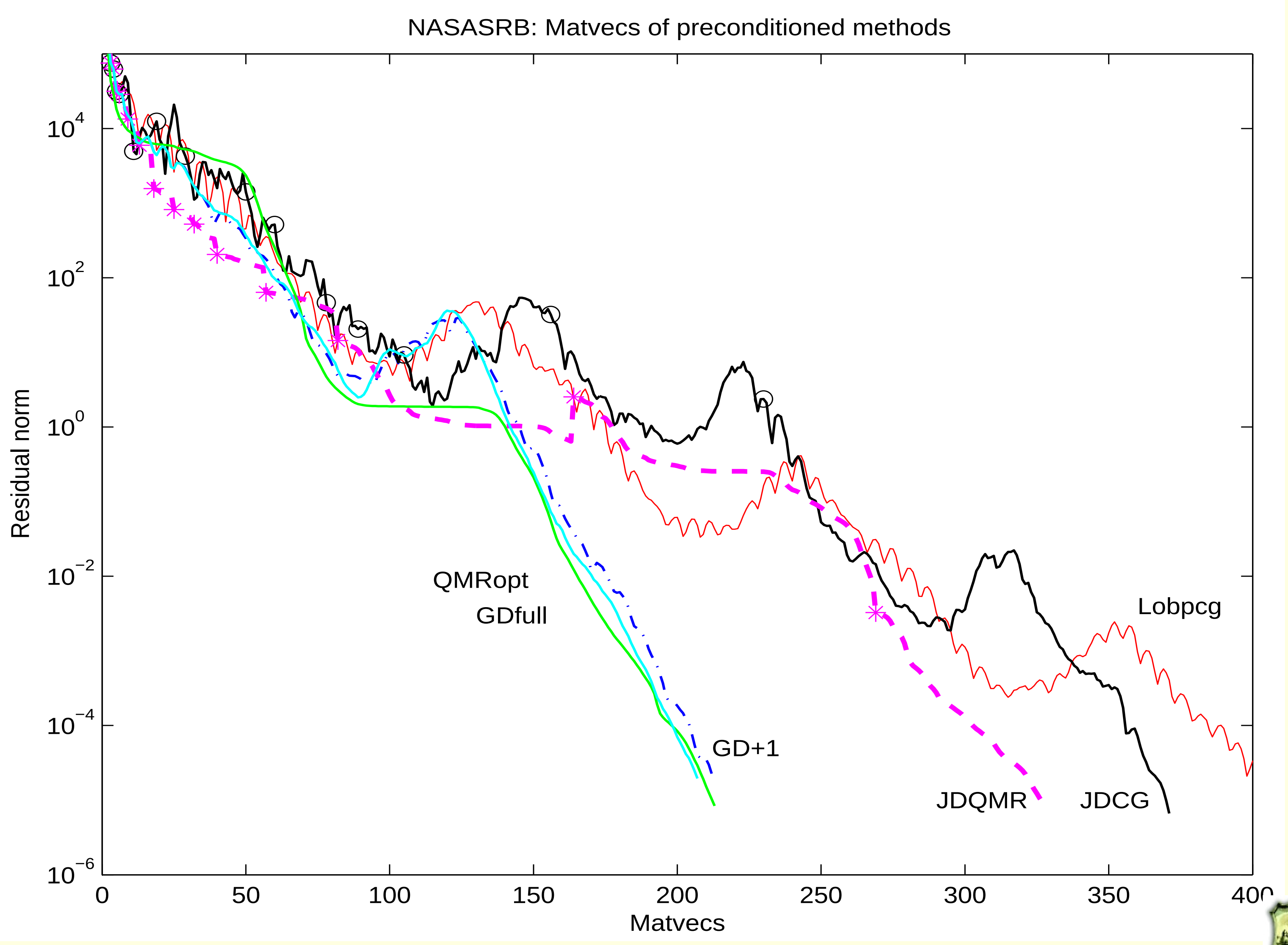
JDqmr with dynamic stopping criterion



BCSSTK09 JDqmr with dynamic stopping criterion

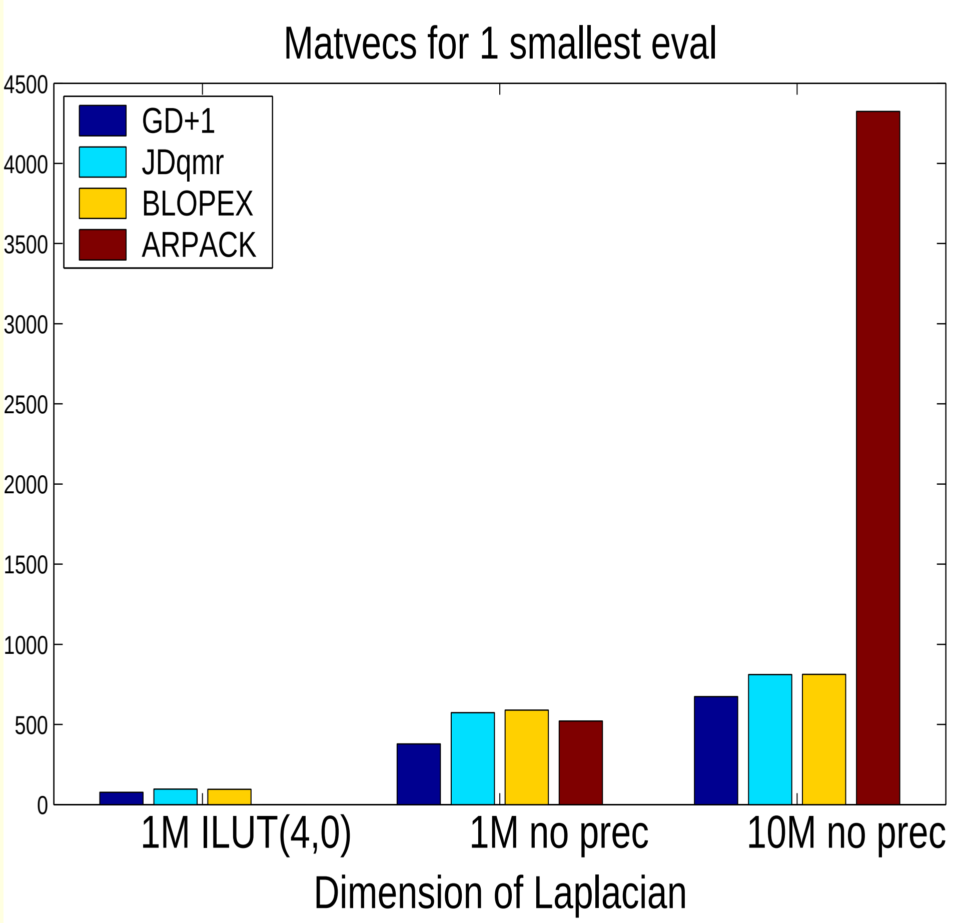
Eigenvalue and linear system residuals very close

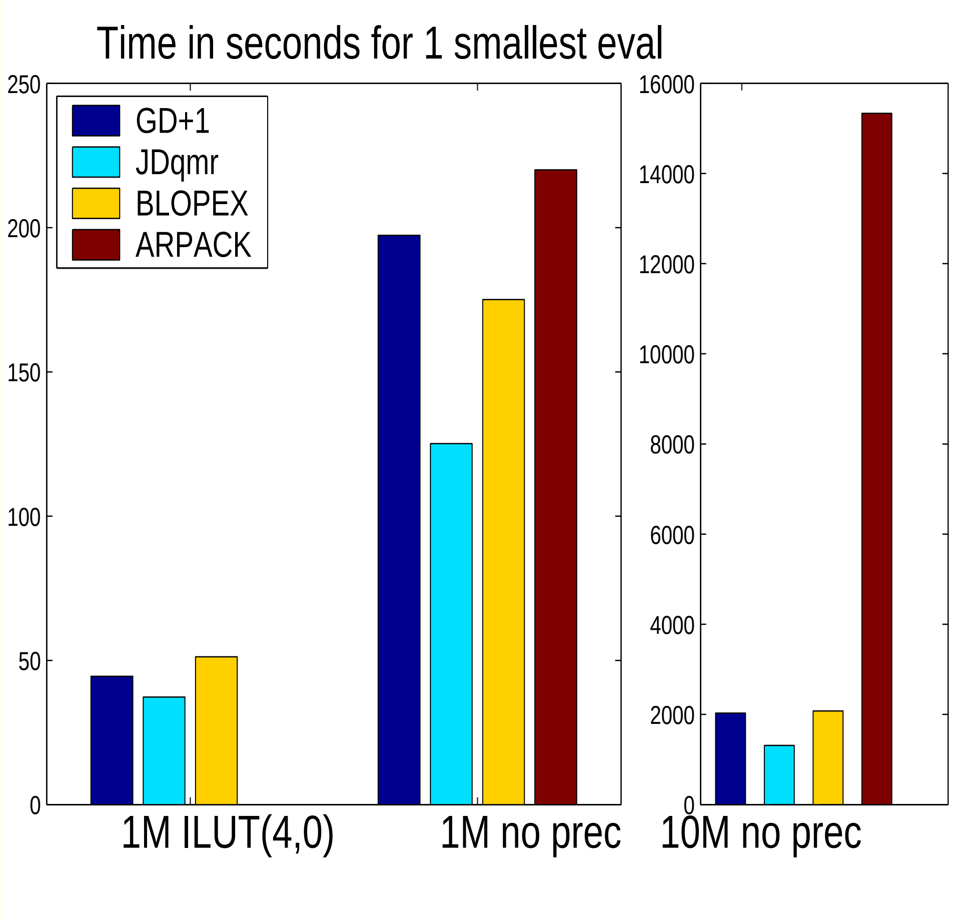
One eigenvalue with preconditioning NASASRB: Note the plateaus



Sample runs GD+k: minMatvecs JDQMR: minTime

1 smallest eigenvalue of ∇^2 for large matrices

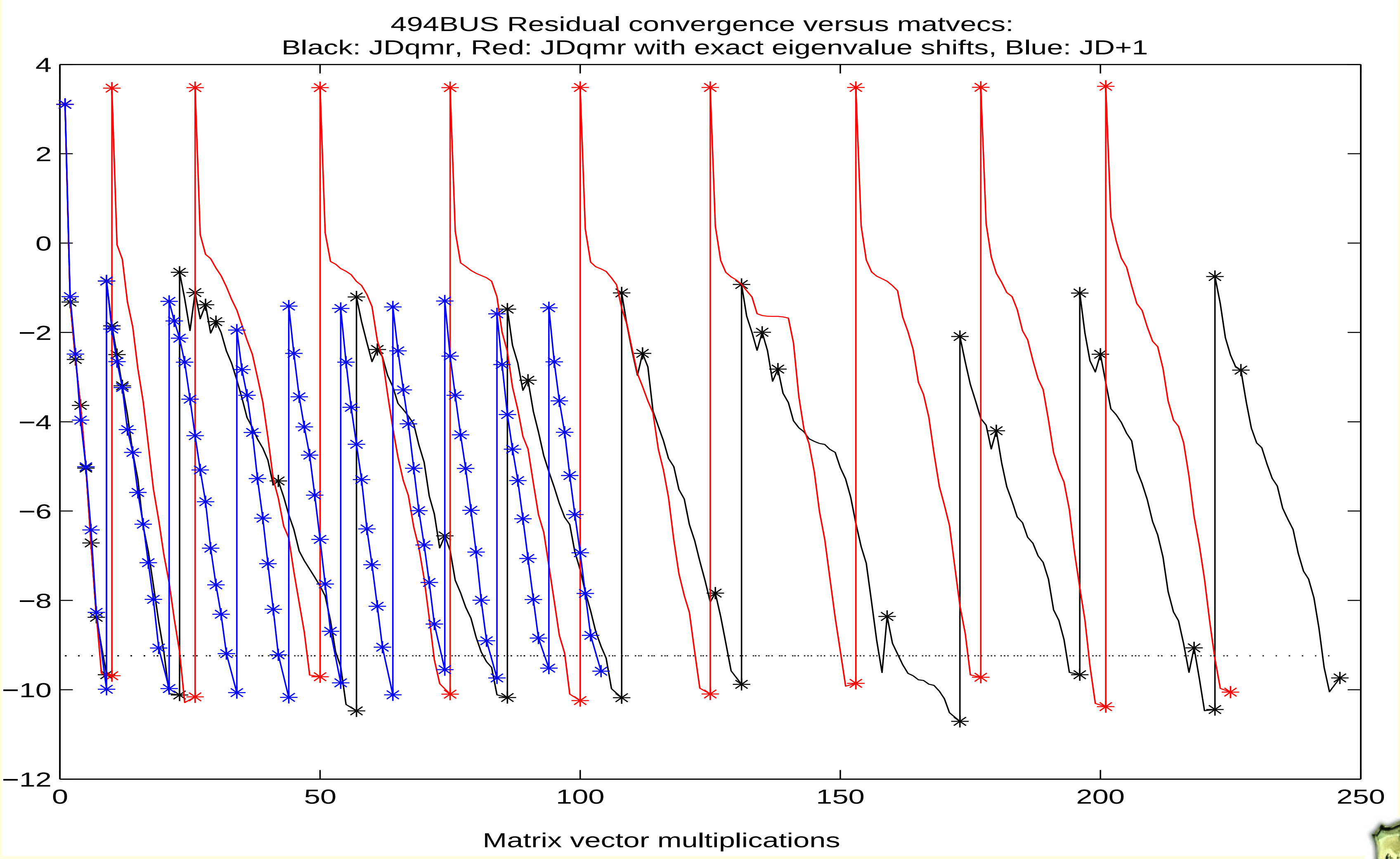




ARPACK for 1M: 525 Matvecs, 220 seconds

What is optimal for many eigenvalues?

- Red: QMRopt (exact eigenvalues as shifts)
- Black: JDQMR nearly optimal
- Blue: subspace accelerated GD+1 better than optimal ?



JD: projecting the locked vectors X $\neq$ projecting the Ritz vector x

$$(I - XX^T)\ \ (A - \eta I)K^{-1}\ \ \big(I - X(X^TK^{-1}X)^{-1}X^TK^{-1}\big)\ \ \ t = -r$$

Left projection

$\Downarrow$

needed for definiteness

Skew Right projection

$\Downarrow$

is it needed?

- Orthogonalization requirements dominate for *nev* $\gg$, so

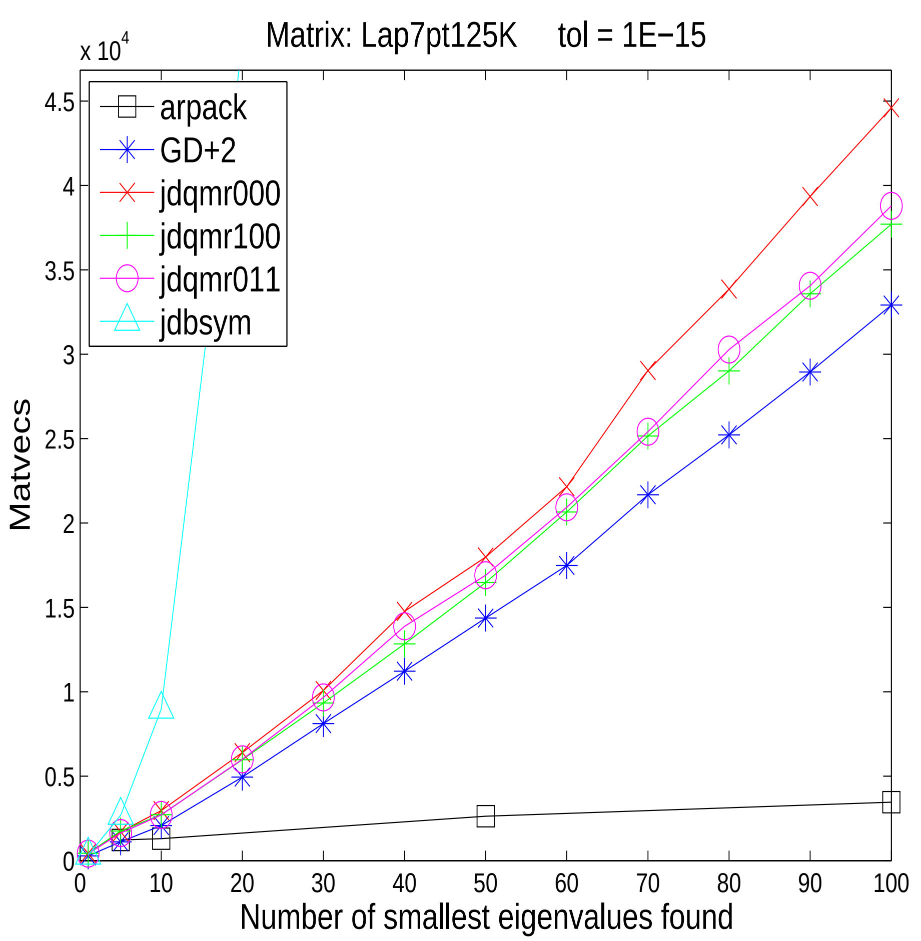
Solve:

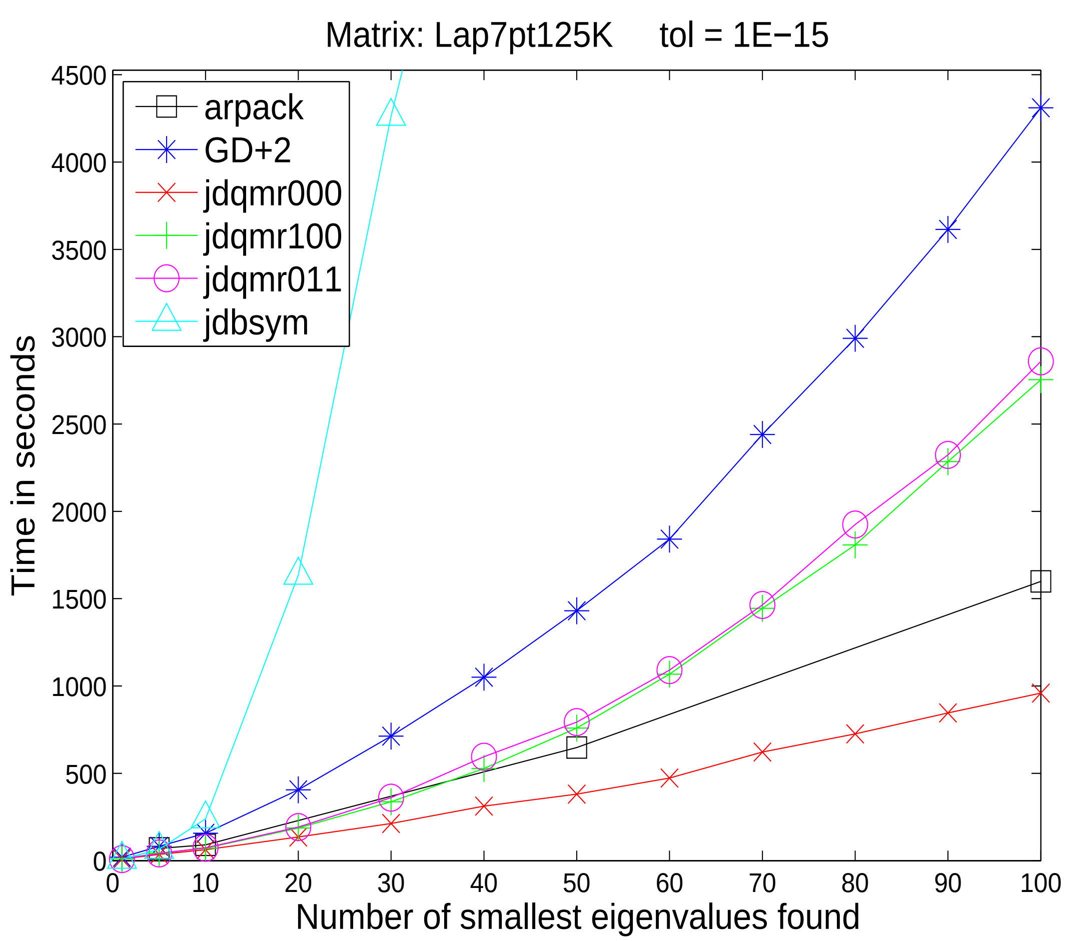
$$\begin{array}{rll} (A - \eta I) & t = -r \ , & \text{w/o preconditioning} \\ (I - XX^T)(A - \eta I)K^{-1} & t = -r \ , & \text{with preconditioning} \end{array}$$

Other choices JDQMR-(Left,Skew,Right) (111), (000), (101), (011), (100)

Scaling numEvals no preconditioner

Laplace 7point, 125K, Tol = 1e-15

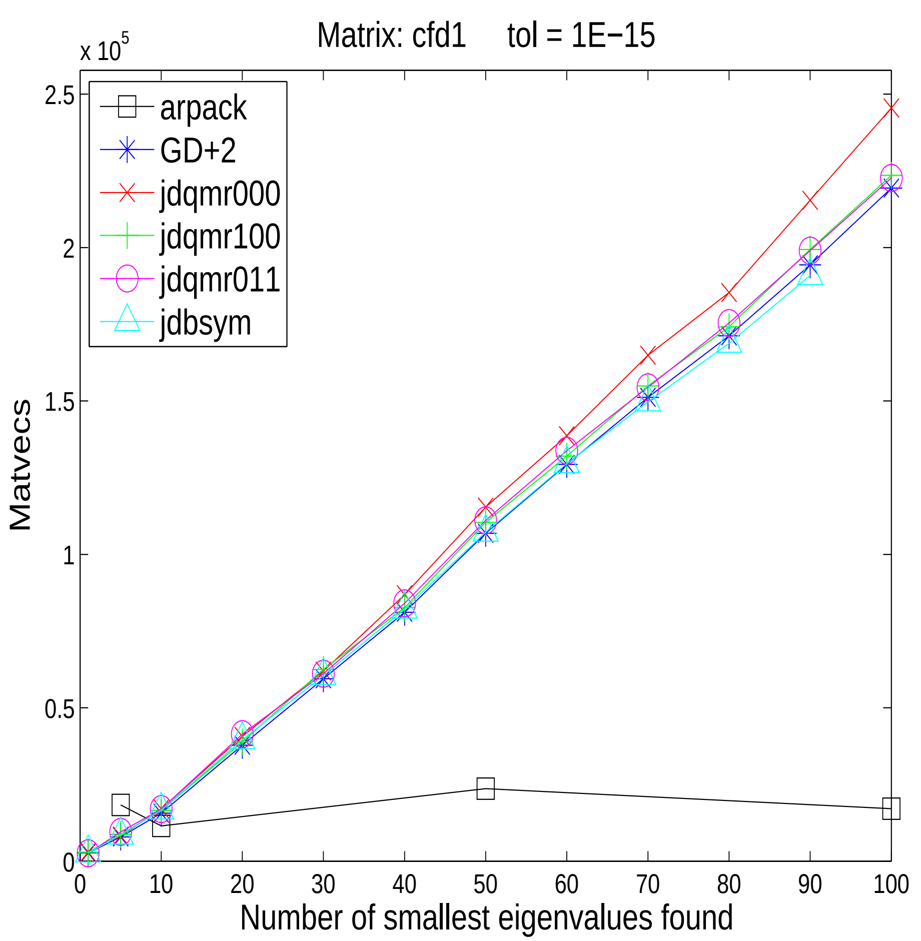


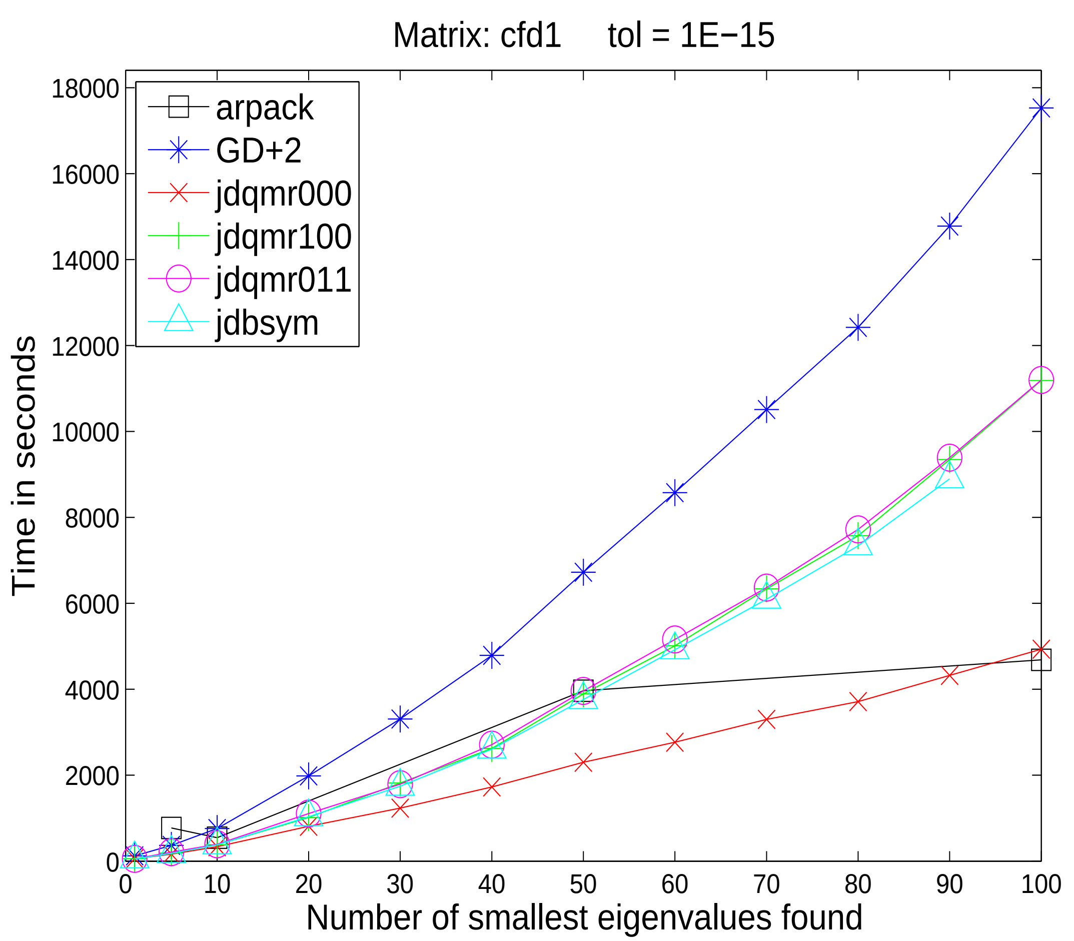


JDQMR-000 fastest among all PRIMME variants and ARPACK

Scaling numEvals   no preconditioner

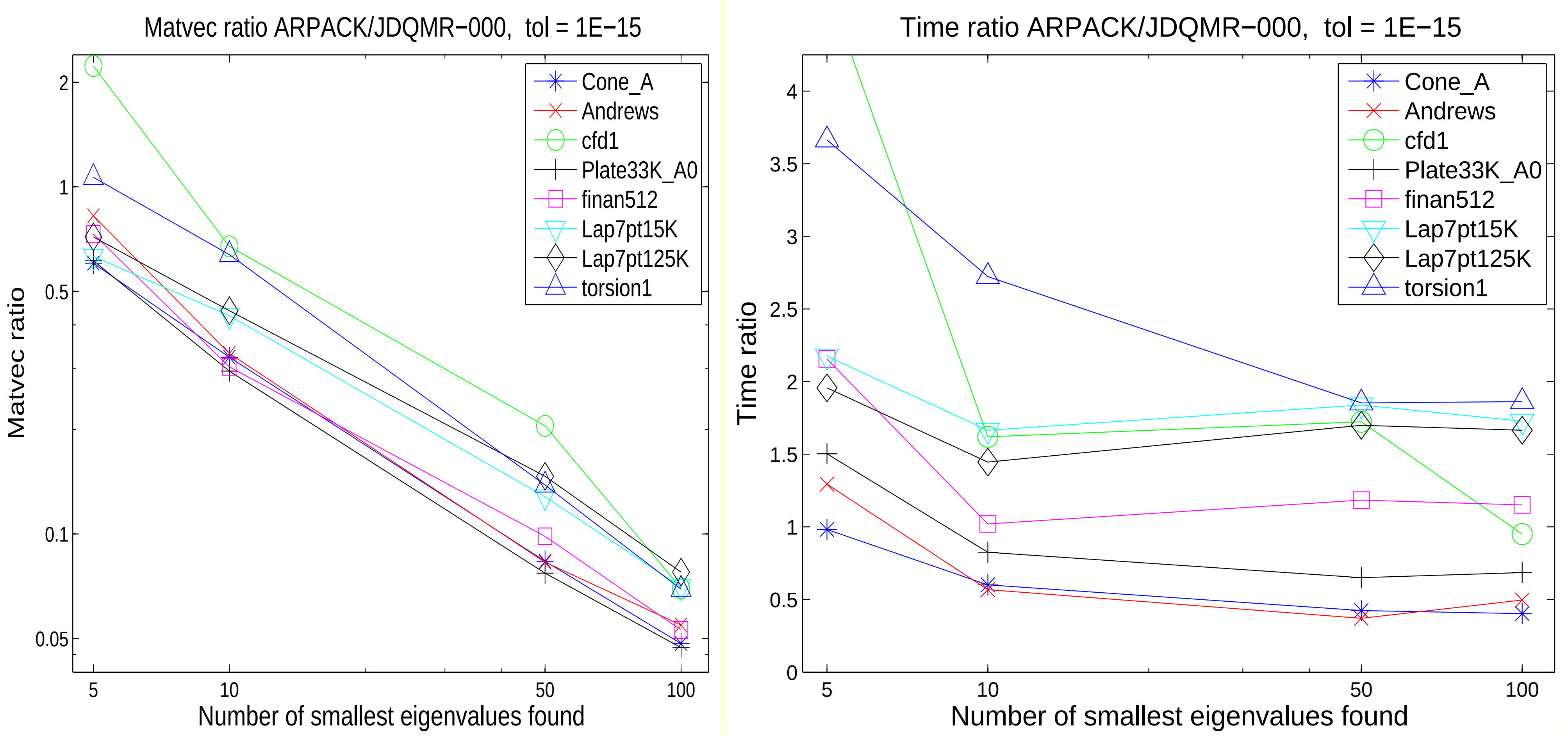
cfd1, 70K, 26 nonzeros/row   Tol = 1e-15





ARPACK *eventually* better for *large numEvals* and *denser matrices*

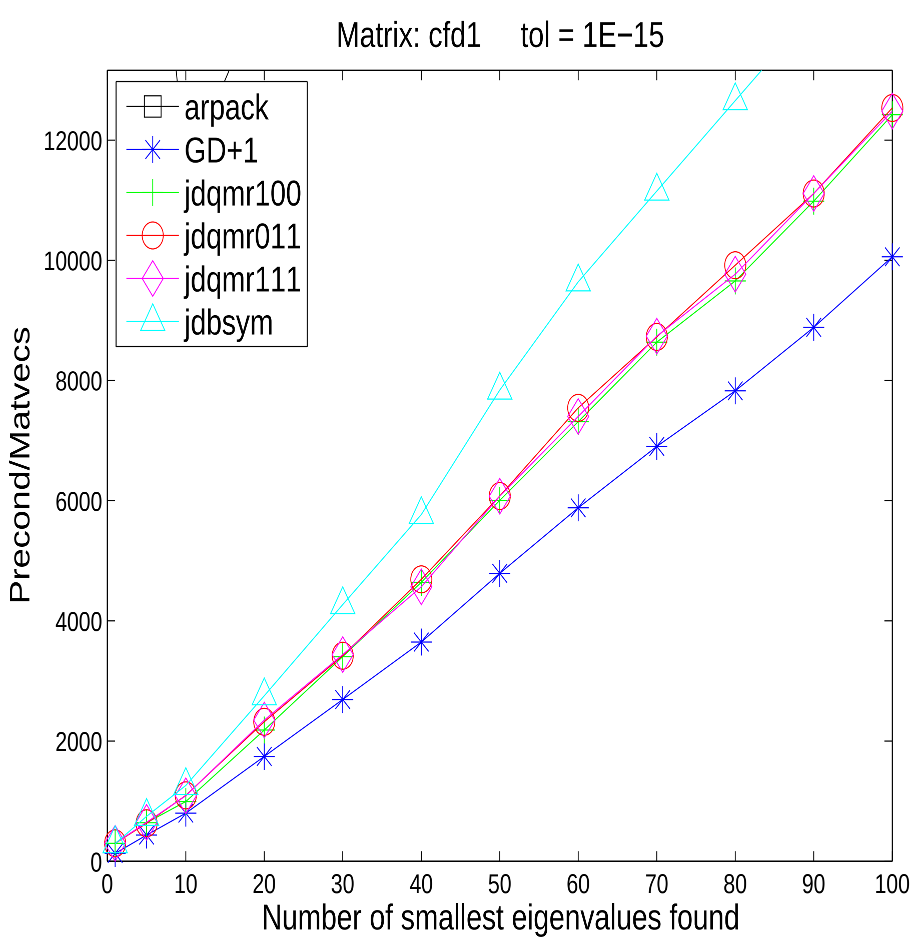
Ratio: ARPACK / JDQMR-000 for 8 matrices

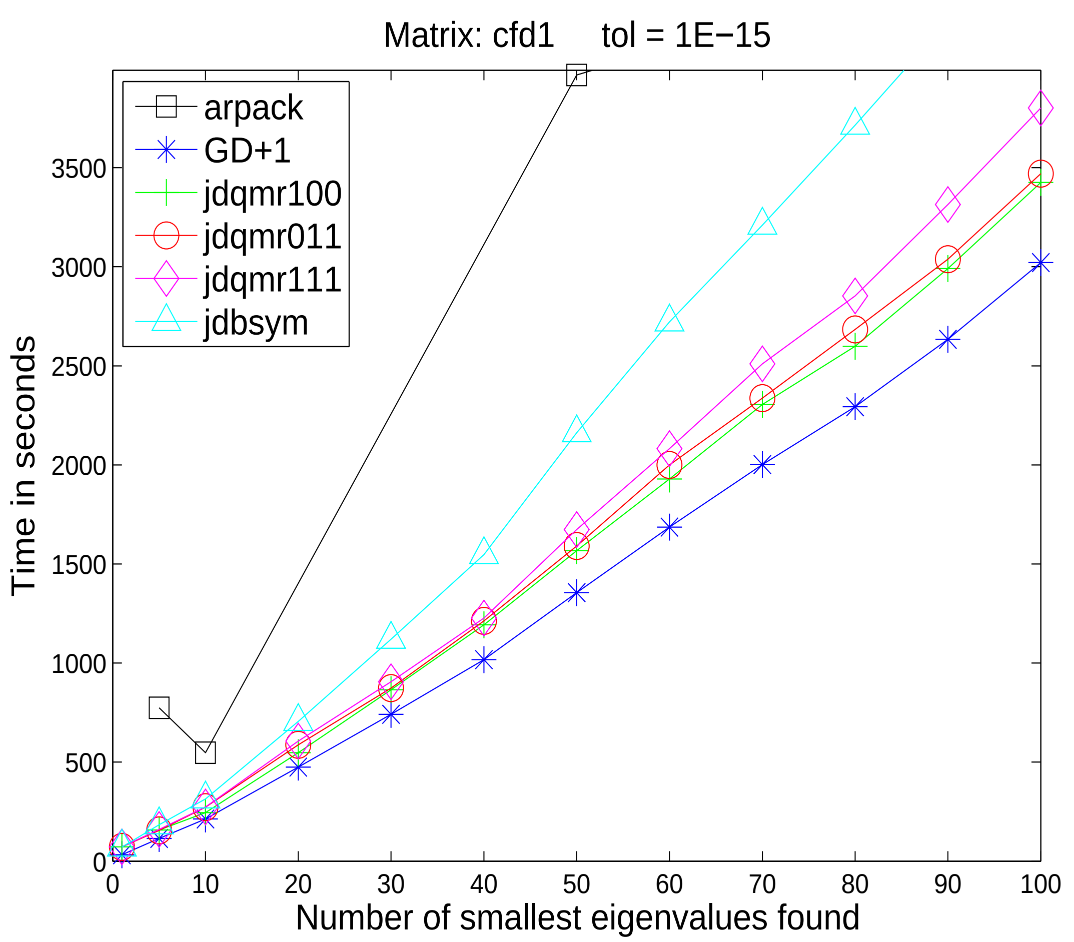


JDQMR-000 faster for `numEvals < 10`. Asymptotically depends on sparsity

Scaling numEvals ILUT(80,1e-4)

cfd1 70K, Tol = 1e-15

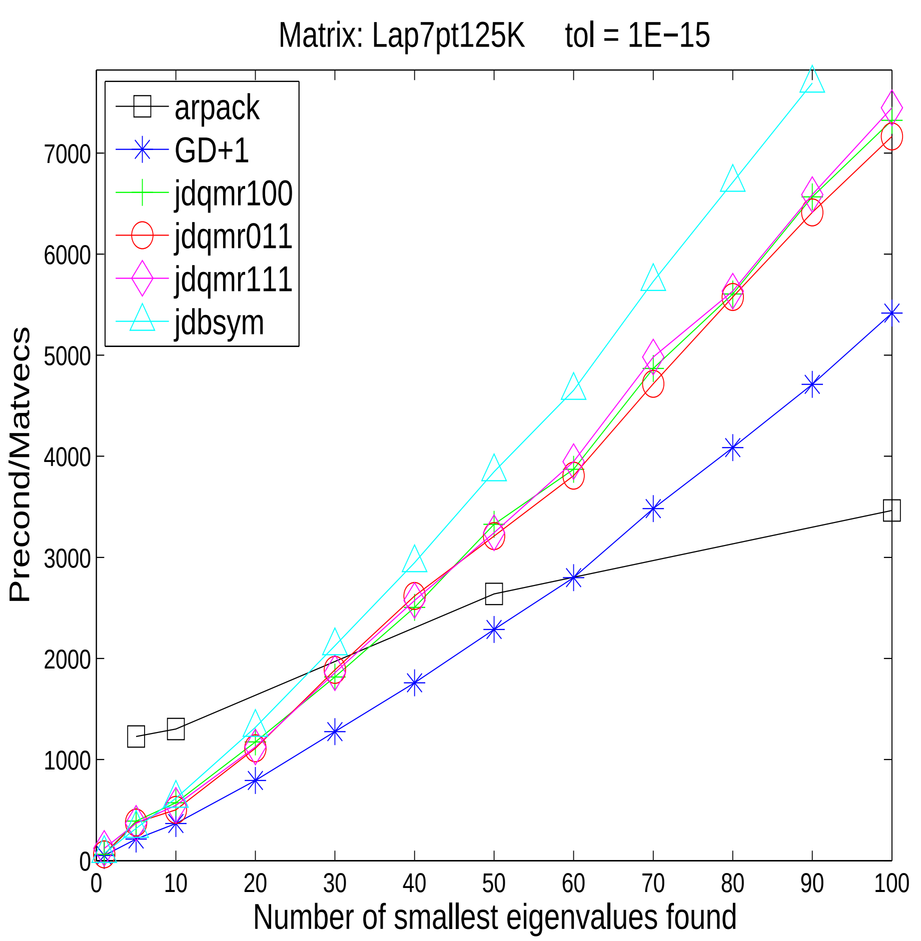


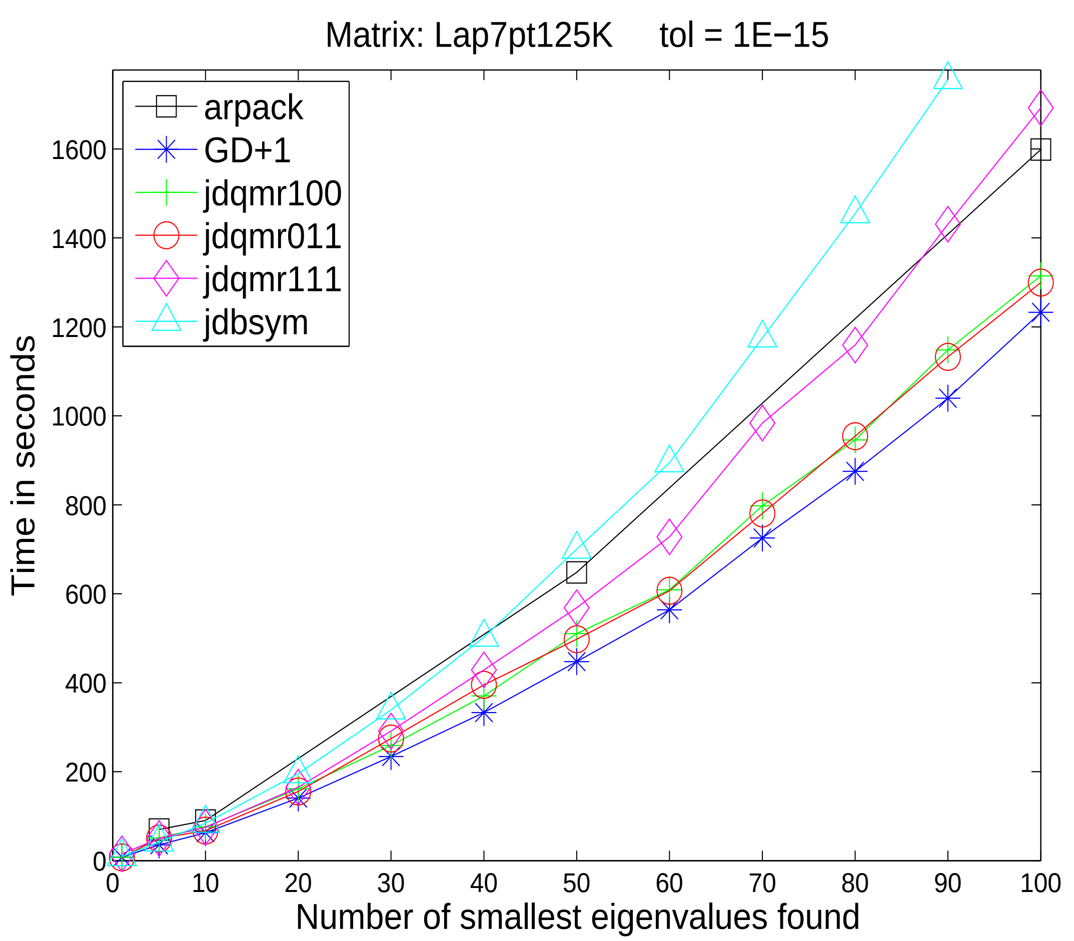


Q-projectors have no effect convergence of JDQMR

Scaling numEvals ILUT(20,1e-6)

Laplace 7point, 125K, Tol = 1e-15





Expensive preconditioner ⇒ fewer MVs means faster (GD+1)

Software availability

PRIMME: PReconditioned Iterative MultiMethod Eigensolver

with my Ph.D. student J.R. McCombs

- Full set of defaults for non expert users
- Full customizability for expert users
- Near optimality through GD+k and JDQMR
- Over 12 methods accessible through PRIMME.
- Parallel, high performance implementation
- C and Fortran interfaces, Matlab interface soon.

Download: `www.cs.wm.edu/~andreas`

Minimal interface – End user

```
#include "primme.h"

primme_params primme;
primme_Initialize(&primme);

primme.n = n;
primme.numEvals = 20;

primme.matrixMatvec        = MV(x,y,k)
primme.applyPreconditioner = PR(x,y,k)

primme_set_method(method, &primme);

ierr = dprimme(evals, evecs, rnorms, &primme);
```

Minimal interface – End user

```
#include "primme.h"

primme_params primme;
primme_Initialize(&primme);

primme.n = n;
primme.numEvals = 20;
```

The matrix and its size have been read.
Number of needed eigenvalues,
smallest by default

```
primme.matrixMatvec        = MV(x,y,k)
primme.applyPreconditioner = PR(x,y,k)

primme_set_method(method, &primme);

ierr = dprimme(evals, evecs, rnorms, &primme);
```

Minimal interface – End user

```
#include "primme.h"

primme_params primme;
primme_Initialize(&primme);

primme.n = n;
primme.numEvals = 20;

primme.matrixMatvec        = MV(x,y,k)
primme.applyPreconditioner = PR(x,y,k)

primme_set_method(method, &primme);

ierr = dprimme(evals, evecs, rnorms, &primme);
```

Pointers to functions for block matrix-vectors, and block precondition-vectors

Minimal interface – End user

```
#include "primme.h"

primme_params primme;
primme_Initialize(&primme);

primme.n = n;
primme.numEvals = 20;

primme.matrixMatvec          = MV(x,y,k)
primme.applyPreconditioner = PR(x,y,k)

primme_set_method(method, &primme);
```

CHOICES:

DYNAMIC
DEFAULT_MIN_TIME
DEFAULT_MIN_MATVECS
Arnoldi
GD
GD_plusK
GD_Olsen_plusK
JD_Olsen_plusK
RQI
JDQR
JDQMR
JDQMR_ETol
SUBSPACE_ITERATION
LOBPCG_OrthoBasis
LOBPCG_OrthoBasis_Window

```
ierr = dprimme(evals, evecs, rnorms, &primme);
```

The full interface – Advanced user

```
#include "primme.h"
primme_params primme;

primme.
   outputFile         = stdout              iseed                        = -1
   printLevel         = 5                   restarting.scheme            = primme_thick
   numEvals           = 10                  restarting.maxPrevRetain     = 1
   aNorm              = 1.0                 correction.precondition      = 1
   eps                = 1.0e-12             correction.robustShifts      = 1
   maxBasisSize       = 15                  correction.maxInnerIterations = -1
   minRestartSize     = 7                   correction.relTolBase        = 1.5
   maxBlockSize       = 1                   correction.convTest   = adaptive_ETolerance
   maxOuterIterations = 10000               correction.projectors.LeftQ  = 1
   maxMatvecs         = 300000              correction.projectors.LeftX  = 1
   target             = primme_smallest     correction.projectors.RightQ = 0
   numTargetShifts    = 0                   correction.projectors.SkewQ  = 0
   targetShifts       = 1.0 2.0             correction.projectors.RightX = 1
   locking            = 1                   correction.projectors.SkewX  = 1
   initSize           = 0                   matrixMatvec                 = MV(x,y,k)
   numOrthoConst      = 0;                  applyPreconditioner          = PR(x,y,k)

ierr = dprimme(evals, evecs, rnorms, &primme);
```

Minimal Fortran interface

```
include ’primme_f77.h’
integer primme
call primme_initialize_f77(primme)
call primme_set_member_f77(primme, PRIMMEF77_n, n)
call primme_set_member_f77(primme, PRIMMEF77_numEvals, 20)
call primme_set_member_f77(primme, PRIMMEF77_matrixMatvec, MV)
call primme_set_member_f77(primme, PRIMMEF77_applyPreconditioner,PR)
call primme_set_method_f77(primme, method, bytesNeeded)
call dprimme_f77(evals, evecs, rnorms, primme, ierr)
```

Similar to C

```
#include "primme.h"
primme_params primme;
primme_Initialize(&primme);
primme.n = n;
primme.numEvals = 20;
primme.matrixMatvec        = MV;
primme.applyPreconditioner = PR;
primme_set_method(method, &primme);
ierr = dprimme(evals, evecs, rnorms, &primme);
```

In QCD we solve a sequence of linear systems

Can we use these Krylov spaces to

1. obtain eigenpairs?
2. use these eigenpairs to deflate and thus accelerate subsequent systems?

For restarted GMRES(m), the variant GMRESDR(m) $\Leftrightarrow$ IRA(m)

GMRESDR computes eigenvalues while solving the system

GMRES expensive per iteration

Restarting slows convergence for linear system AND eigenvectors

Can we be more effective on CG/Lanczos?

Our recent work (with K. Orginos)

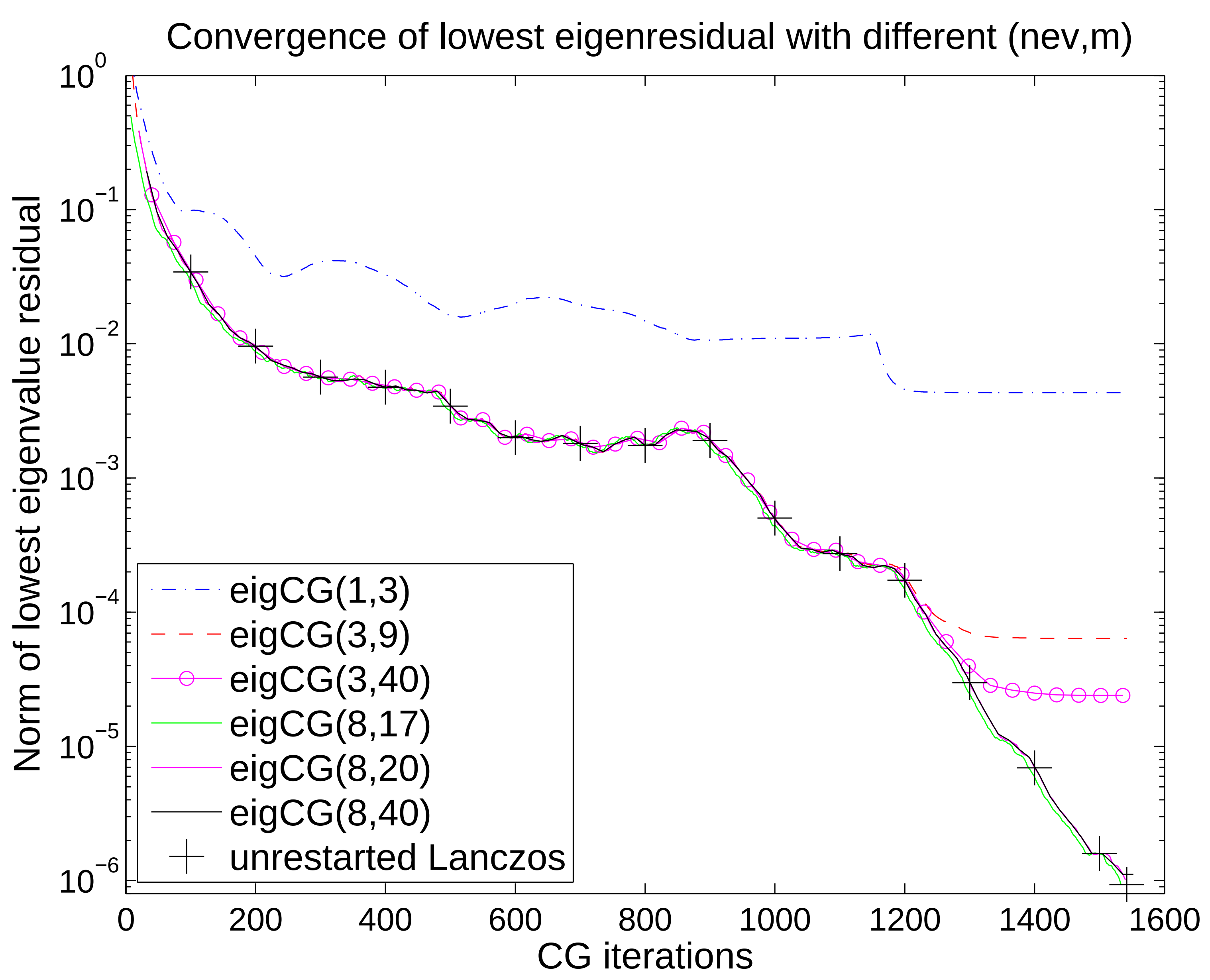
Small window of m vectors, V, keeps track of the smallest $nev < m$ eigenvectors

V is expanded by the CG residuals

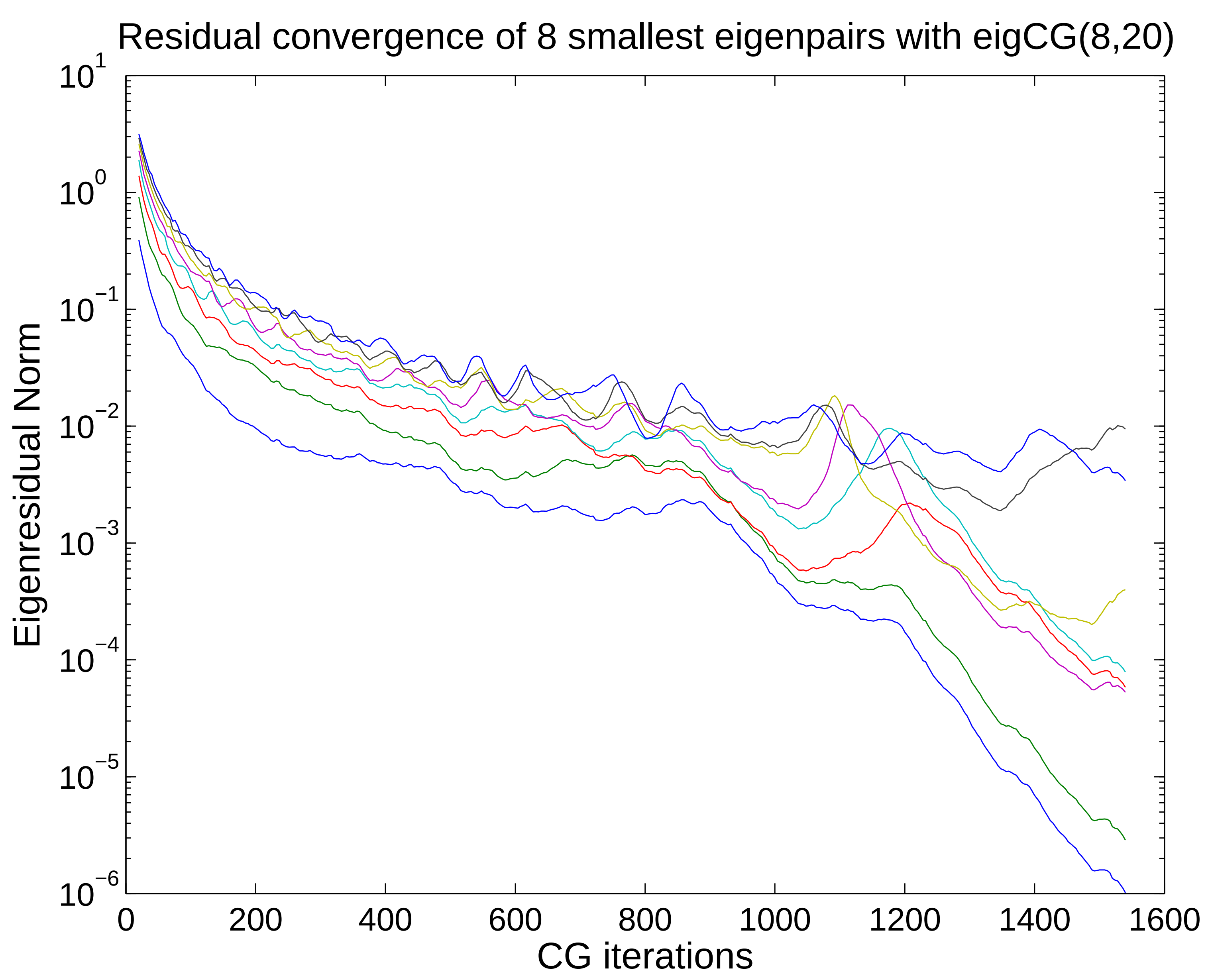
When m vectors in V, restart it as in $\mathrm{GD}(nev,m)+nev$

- CG iterates unaffected
- Records the Lanczos vector contributions to eigenvectors

Does it find eigenvalues accurately?



Does this find eigenvalues accurately? Identical to Lanczos



Incrementally improving accuracy and number of eigenvalues

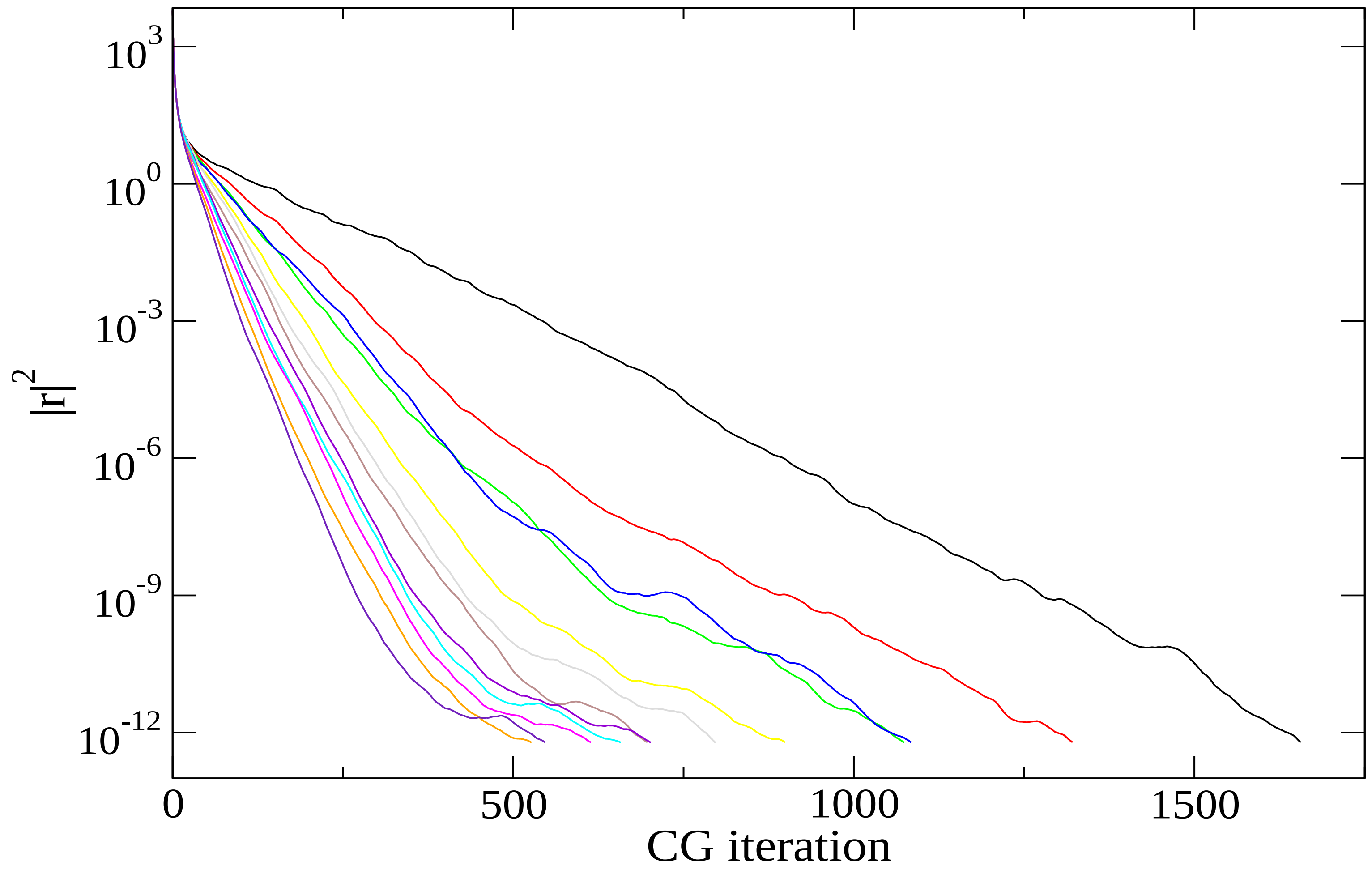
Use the CG iterations for $\tilde{k}$ subsequent RHS to improve U:

```
Incremental eigCG
$U = [\,], \Lambda = [\,]$                                  // accumulated eigenpairs
for i = 1 : $\tilde{k}$
    $x_0 = U\Lambda^{-1}U^H b_i$                            // the init-CG part
    $[x_i, V, M\,]$ = eigCG($nev, m, A, x_0, b_i$)          // eigCG with initial guess $x_0$
    $[U, \Lambda]$ = RayleighRitz($[U, V]$);
end
```

Typical values:

$$k = 100,\ \tilde{k} = 12 - 24,\ nev = 10,\ m = 40$$

Convergence improves after every new CG



A realistic experimental case

Lattice parameters:

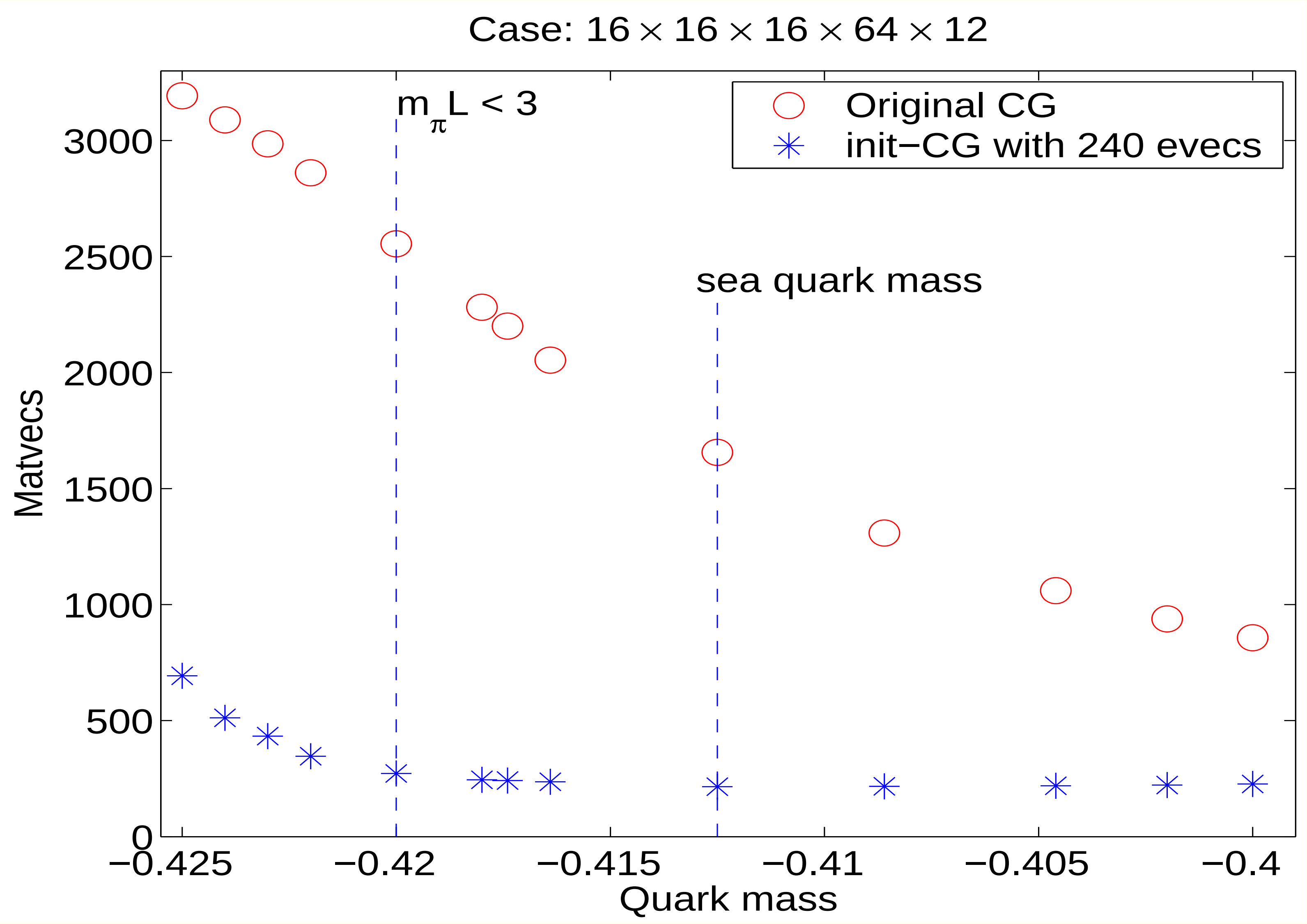
- 2 flavor Wilson fermions
- Lattice spacing $a_s = 0.1\, fm$ (spatial)
- anisotropic: $a_t = a_s/3$
- pion mass (350-400 MeV)

Two lattice sizes:

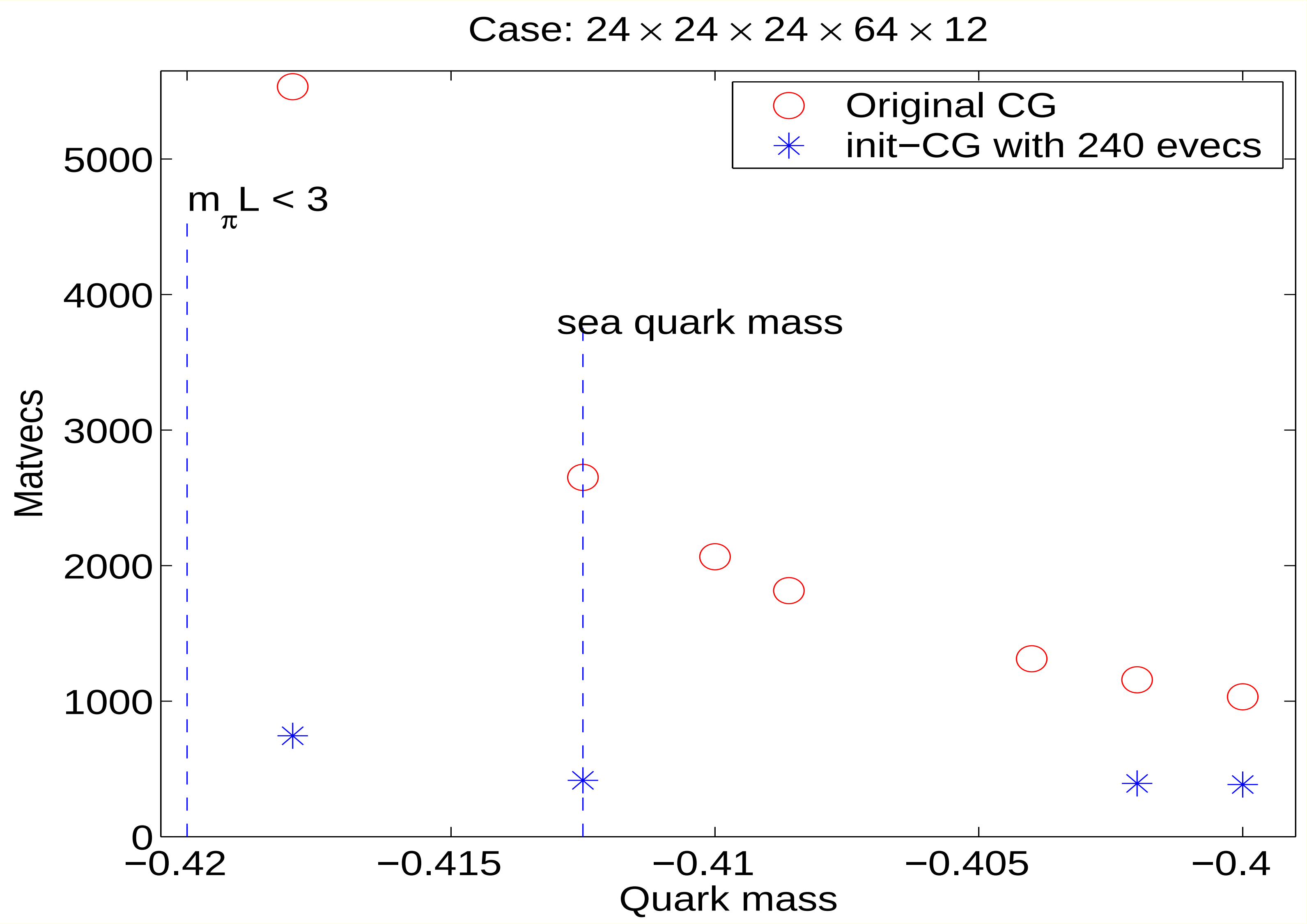
- $16^3 \times 64$ for a matrix dimension of 3.1 million
- $24^3 \times 64$ for a matrix dimension of 10.6 million

Currently running using **Chroma** at Jefferson Lab

Case: 3.1 million



Case: 10.6 million



Conclusions

- JDQMR ⇔ subspace accelerated inexact (truncated) Newton
- GD+1 ↔ subspace accelerated quasi Newton
- Near optimal for just a few eigenpairs
- Cheaper projectors possible with JDQMR for many eigenvalues

PRIMME a state of the art eigensolver

Our recent research promising for optimal, limited memory eigensolver