

Algorithms

MITP - Summer school

July 30, 2025

J. Finkenrath

Outline - Part 1

- Part 0
 - Markov Chain Monte Carlo
 - Hybrid Monte Carlo algorithm
- Part 1
 - Linear solvers, Krylov subspace solvers
 - Preconditioners, smoothers and coarse grid
- Part 3
 - Fermions in simulations
 - ...

parts are based on Gustavo Ramirez-Hidalgo's Lattice Practice 2024 and Yousef Saad book 'Iterative Methods for Sparse Linear systems'

Linear systems in lattice QCD

Most computing time in lattice QCD is spend in solving linear equation of the type

$$D \cdot x = b$$

⇒ it is of utmost importance to find efficient /most efficient solvers

Solving the Dirac equation is required

- during MCMC simulations
- calculation of propagators

Discretatization of Dirac operator and use cases

- D is sparse (e.g. Wilson) or dense (e.g. overlap)
- a solution is only required for one right hand sides (rhs) or many

this influence the choice of the linear solver

Discretization of the Dirac operator

Discretisations by covariant finite-differences

$$d_\mu \psi_x = \frac{1}{a} (U_\mu(x - a\hat{\mu})\psi(x - a\hat{\mu}) - (U_\mu^\dagger(x)\psi(x - a\hat{\mu})))$$

Wilson discretization (adding Wilson term):

$$D_W = \sum_{\mu}^4 (\gamma_\mu d_\mu + a^{-1} d_\mu^2) \in \mathbb{C}^{12L_s^3 L_t \times 12L_s^3 L_t}$$

Typical discretizations yield linear systems $Dx = b$ with:

- D is **non-hermitian** ($A \neq A^\dagger$), yet $(\gamma_5 D)^\dagger = \gamma_5 D$
- $\text{spec}(D)$ is in the right half complex-plane, so that the matrix is **positive definite**
 $x^\dagger A x > 0, \quad x \neq 0$
- D is **very large** (e.g. small lattice $32^3 \times 64$ has 25 M unknowns)
- D is **sparse**, i.e. contains only next neighbor couplings, that's ~ 100 non-zeros per row

Matrix-vector operations are *relative* cheap $O(L_s^3 L_t) = O(V)$

Lattice QCD solver libraries come (usually) with highly optimized implementations for $D \cdot x$ (Dslash-operator)

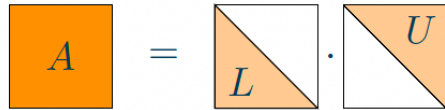
Direct methods

Idea: Solve linear system $Ax = b$ by row-/column-manipulations

Usually based on factorizing the system matrix A

- Methods based on Gaussian elimination

- $A = LU$: LU factorization


$$\boxed{A} = \boxed{L} \cdot \boxed{U}$$

- $A = LDL^{\dagger}$: Collesky factorization (A hermitian)

Direct methods are very **expensive** ($O(n^3)$ for dense matrices)

- methods exploiting sparsity exists, reducing complexity but still have a large memory and computational footprint

In general **iterative methods** are more efficient

Iterative solvers

Given:

$$Ax = b$$

with solutions \hat{x} and matrix is a A sparse matrix

Find: Approximations

$$x^{(k)}, \quad k = 1, 2, \dots, \quad \text{such that } x^{(k)} \rightarrow \hat{x}$$

1. How do we measure convergence of $x^{(k)} \rightarrow \hat{x}$?
 - require a "computable" measures ("stopping criteria")
 - is this possible by having a monotonic convergence ?
1. How we can find an **iterate** $x^{(k)}$ such that
 - the iterative process converges, namely $x^{(k)} \rightarrow \hat{x}$?
 - we can define a "simple" update formula for $x^{(k+1)}$?
 - each iteration requires only the one (or a few) operation of A on a vector
 - minimal application of $Dslash = Ax$, which requires $O(V)$ operations

How do we measure convergence ?

Given: Iterate $x^{(k)}$ in the k th iteration

- Using the **error**: $e^{(k)} = \hat{x} - x^{(k)} = A^{-1}b - x^{(k)}$

$$x^{(k)} \rightarrow \hat{x} \implies \|e^{(k)}\| \rightarrow 0$$

in most cases the error is **not** readily computable (*solution not known*)

- Using the **residual**: $r^{(k)} = b - Ax^{(k)}$

$$x^{(k)} \rightarrow \hat{x} \implies \|r^{(k)}\| \rightarrow 0$$

The residual is a computable quantity.

Note that:

$$r^{(k)} = b - Ax^{(k)} = A\hat{x} - Ax^{(k)} = Ae^{(k)}$$

from now on, we define $x^{(0)} = 0$

How do we find a suitable iterate $x^{(k)}$

Task: Given b find x , such that $Ax = b$ or

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad i = 1, \dots, n$$

Start: Solver for x_i for each i . Note, that if our residual $r = b - Ax$ gets smaller, we are closer to our solution

\implies **Idea:** Set entry of $r_i^{(k+1)}$ to zero: $(b - Ax)_i = 0$

- **Jacobi iteration** for $i = 1, \dots, n$

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^n a_{ij}x_j^{(k)} \right)$$

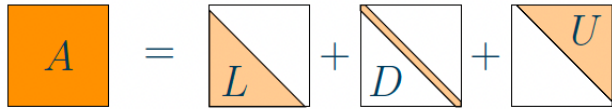
- **Gauss-Seidel iteration**

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^n a_{ij}x_j^{(k)} \right)$$

This sets entry of $r_i^{(k+1)}$ to zero using also updated previous values of iterate x

Splitting methods

Splitting methods use the **additive decomposition** of $A = L + D + U$


$$A = L + D + U$$

- Jacobi: $x^{(k+1)} = x^{(k)} + D^{-1}r^{(k)}$
- Gauss-Seidel: $x^{(k+1)} = x^{(k)} + (D + L)^{-1}r^{(k)}$
- SOR: $x^{(k+1)} = x^{(k)} + (\frac{1}{\omega}D + L)^{-1}r^{(k)}$ (Successive Over Relaxation)

General splitting method: $A = M + N$ (recall: error is given by $e^k = \hat{x} - x^k = A^{-1}r^{(k)}$)

$$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)} \quad \implies \quad e^{(k+1)} = e^{(k)} - M^{-1}Ae^{(k)}$$

Convergent if $\|I - M^{-1}A\| < 1$

- splitting methods are often used as preconditioners

Projection methods

Idea: Find solution within a smaller subspace. **Lets defined spaces K and $L \in \mathbb{C}^n$**

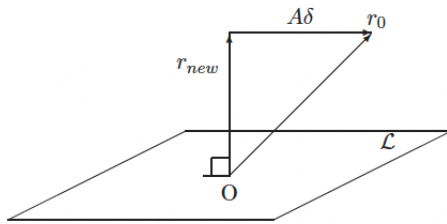
Now, our iterate \tilde{x} and its residual \tilde{r} fullfills

$$\tilde{x} \in K \quad \text{and} \quad \tilde{r} = b - A\tilde{x} \perp L$$

Note, that we want to exploit the initial guess, such that $\tilde{x} \in x_0 + K$

We can write $\tilde{x} = x_0 + \delta$ and for the residual follows

$$b - A(x_0 + \delta) = r_0 - A\delta \perp L$$



- Gauss-Seidel can be defined by a projection step with $K = L = \text{span}(e_i)$

Note: $r \perp L$ is called Petrov-Galerkin condition

One-Dimensional Projection Processes

One dimensional projection processes:

$$K = \text{span}(v) \quad \text{and} \quad L = \text{span}(w)$$

with two vectors v and w . By imposing the Petrov-Galerkin condition, it follows:

$$x \leftarrow x + \alpha v \quad \text{and} \quad r - A\delta \perp w$$

gives

$$\alpha = \frac{(r, w)}{Av, w}$$

Steepest Descent converges if A is positive symmetric. We set $v = r$ and $w = r$. It follows

$$r \leftarrow b - Ax$$

$$\alpha \leftarrow (r, r)/(Ar, r)$$

$$x \leftarrow x + \alpha r$$

requires one matrix-vector application per iteration

One-Dimensional projections

One dimensional projection processes we set the spaces to

$$K = \text{span}(v) \quad \text{and} \quad L = \text{span}(w)$$

Steepest Descent: $v = r, w = r$

- converges for positive symmetric matrices
- each step minimizes $\|x - \hat{x}\|_A^2$ in the direction $-\nabla f$

Minimal Residual (MR) Iteration: $v = r, w = Ar$

- converges for positive definite matrices
- each step minimizes $f(x) = \|b - Ax\|_2$ in the direction r

Residual Norm Steepest Descent: $v = A^\dagger r, w = Ar$

- converges for non-singular matrices
- each step minimizes $f(x) = \|b - Ax\|_2$ in the direction $-\nabla f$

Generalization of projection methods:

x_m is an solution from the affine subspace $x_0 + K_m$ by imposing the Petrov-Garlerkin condition

$$b - Ax_m \perp L_m$$

A Krylov subspace method is based on the Krylov subspace

$$K_m(A, r_0) = \text{span}(r_0, Ar_0, \dots, A^{m-1}r_0)$$

Note, an approximated solution from K_m can be written as a polynome of order $m - 1$

$$A^{-1}b \approx x_m = x_0 + q_{m-1}(A)r_0$$

- the inverse of the matrix can be written as a matrix function
- useful for proof/understand convergence etc.

Arnoldi methods

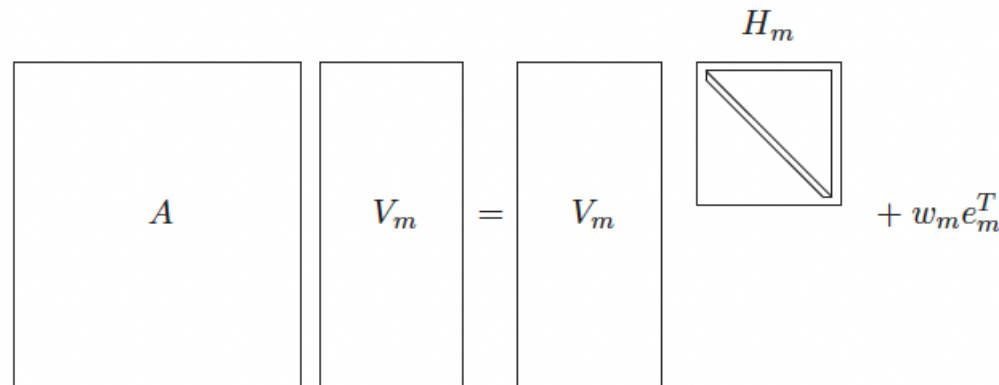
Arnoldi's method is an orthogonal projection onto K_m

- Arnoldi method is introduced as a method to reduce a dense matrix on a Hessenberg matrix:

Its based on writing:

$$AV_m = V_m H_m + w_m e_m^T = V_{m+1} \bar{H}_m$$

with



The diagram illustrates the matrix equation $AV_m = V_m H_m + w_m e_m^T$. It consists of several components: a large square matrix labeled A , a vertical rectangle labeled V_m , an equals sign, another vertical rectangle labeled V_m , a small square matrix labeled H_m with a diagonal line inside, and the term $+ w_m e_m^T$.

the coefficients $h[i][j]$ of the **Hessenberg** matrix H_m can be calculated via Gram-Schmidt orthogonalization

Generalized Minimum Residual Method (GMRES) is a projection method based on

$$K = K_m \quad \text{and} \quad L = AK_m$$

so that the residual is minimized over the space $x_0 + K_m$

Basic algorithm:

```

r0 = b-A*x0, beta = || r0 ||, v1=r0/beta # Starting conditions
for j=1, ..., m do
    w = A *v[j]                # Increase Krylov space j → j+1

    for i=1, ..., j do          # Arnoldi Gram-Schmidt orthogonalization
        h[i,j] = (w,v[j])      # calculate elements of Hessenberg matrix
        w = w-h[i,j]*v[j]      # orthogonalize w wrt v[j]
    end for

    h[j+1,j] = ||w||            # lower diagonal of Hessenberg
    v[j+1] = w/h[j+1,j]
end for

Define Vm =[v[1], .. v[m]], H={h[i,j]}
Solve ym = argmin || beta*e1 - H*y || #Impose Petrov-Garlekin condition (can be done via LU)
x0=x0+Vm*ym      # update iterate
    
```

if x_0 is not sufficient, restart GMRES (if m too large breakdown of orthogonality)

Krylov-solver methods

GMRES: $K = K_m$ and $L = AK_m$

- requires calculation of $A * v$ in each iteration
- converges for non-hermitian positive definite matrices
- mathematical equivalent to **GCR**

FOM: $K = K_m$ and $L = K_m$ (Full Orthogonalization Method)

- converges for positive definite hermitian matrices
- mathematical equivalent to **CG** (Conjugate Gradient) solver
 - CG is an elegant form which utilize the orthogonality of hermitian matrices. Hessenberg matrix becomes tri-diagonal resulting from the three-term Lanczos recurrence
 - CG is often the standart algorithm, but need a hermitian/symmetric matrix

$$Dx = b \quad \Rightarrow \quad D^\dagger Dx = Db$$

more variants / Zoo of Krylov subspace solvers like **BiCGstab**, **Block Krylov** solvers, communication avoiding variants ..

Optimal methods

Kryloc subspace methods are all-duty solvers

- "Optimal" methods for any application
- fast (short-recurrence) solvers for many applications, like CG
- Convergence depends on conditioning of A
 - in case of Conjugate Gradient solver

$$\|e^{(k)}\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|e^{(0)}\|_A, \quad \kappa = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

How to improve convergence of Krylov subspace methods ?

1. Preconditioning
2. Deflation

Scaling issues in Numerical Simulations

Numerical simulation of partial differential equations (PDEs)

$$L\psi = \phi$$

Discretization of L on lattice with spacing α yields

$$Lx = f$$

- Depending on PDE order and order of discretiation

$$\kappa(L) \sim \alpha^{-\sigma}, \quad \sigma \in \mathbb{N}^+$$

- Increasing accuracy of discretization ($\alpha \rightarrow 0$)

$$\kappa(L) \rightarrow \infty \quad (\alpha \rightarrow 0)$$

Performance of Krylov methods deteriorates when $\alpha \rightarrow 0$

—→ critical slowing down of linear solvers

Preconditioning

Idea: Improve conditioning of A in $Ax = b$

- instead of solving $Ax = b$ consider solving

$$S_l A S_r y = S_l b$$

$$x = S_r y$$

with preconditioners S_l, S_r , so that $\kappa(S_l A S_r) \ll \kappa(A)$

Consider the cases

- $S = I: \implies SA = A$ original setting
- $S = A^{-1}: \implies SA = I$ and $\kappa(SA) = 1$ (ideal)
- $S = A^\dagger: \implies SA = A^\dagger A$ hermitian but $\kappa(SA) = \kappa(A)^2$

In order to speed up convergence the preconditioner S should

- S should approximate A^{-1} and be cheap/ or lead to a good iteration count vs. work trade-off

Preconditioning - GMRES

Preconditioned GMRES

```
r0 = S*(b-A*x0), beta = || r0 ||, v1=r0/beta # Starting conditions, modified by an application of S
for j=1, ..., m do
    w = S* A *v[j]                # Increase Krylov space j → j+1
                                   # modified by application of S

    for i=1, ..., j do             # Arnoldi Gram-Schmidt orthogonalization
        h[i,j] = (w,v[j])          # calculate elements of Hessenberg matrix
        w = w-h[i,j]*v[j]         # orthogonalize w wrt v[j]
    end for

    h[j+1,j] = ||w||               # lower diagonal of Hessenberg
    v[j+1] = w/h[j+1,j]
end for

Define Vm =[v[1], .. v[m]], H={h[i,j]}
Solve ym = argmin ||beta*e1 - H*y|| #Impose Petrov-Garlekin condition (can be done via LU)
x0=x0+Vm*ym      # update iterate
```

- if S is constant, GMRES can be easily adapted
- possible to modify also other solvers like **CG**, but here *hermiticity* has to be satisfied

Selection of preconditioners

Aims for the construction of preconditioners S

1. $S \approx A^{-1}$ to get speed-up
2. S should be computational efficient, iteration count reduction vs work trade off

Classes of preconditioners:

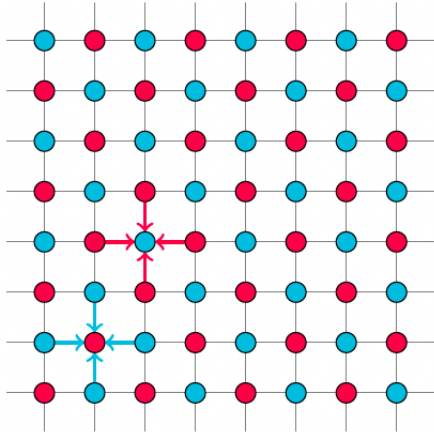
- splitting-based preconditioners
- structural preconditioners
- multi-grid preconditioners
- domain decomposition preconditioners ...

Recap: **Splitting methods**

- Jacobi: $x^{(k+1)} = x^{(k)} + D^{-1}r^{(k)}$ with $S = D^{-1}$
- Gauss-Seidel: $x^{(k+1)} = x^{(k)} + (D + L)^{-1}r^{(k)}$ with $S = (D + L)^{-1}$
- SOR: $x^{(k+1)} = x^{(k)} + (\frac{1}{\omega}D + L)^{-1}r^{(k)}$ with $S = (\frac{1}{\omega}D + L)^{-1}$

Odd-even preconditioning

Discretizations on lattice with next neighbor coupling



Nodes are odd or even

Now with $S_c = A_{ee} - A_{eo}A_{oo}^{-1}A_{oe}$, the solution of $Ax = b$ is given by

Odd-Even Reduction

$$\begin{aligned} y_o &= A_{oo}^{-1}b_o \\ \text{Solve } S_c x_e &= b_e - A_{eo}y_o \\ x_o &= y_o - A_{oo}^{-1}A_{oe}x_e \end{aligned}$$

Ordering by odd-even

$$A = \begin{bmatrix} A_{oo} & A_{oe} \\ A_{eo} & A_{ee} \end{bmatrix}$$

with diagonal A_{oo} and A_{ee}

- A_{oo}^{-1} , A_{ee}^{-1} trivial
- odd and even decoupled

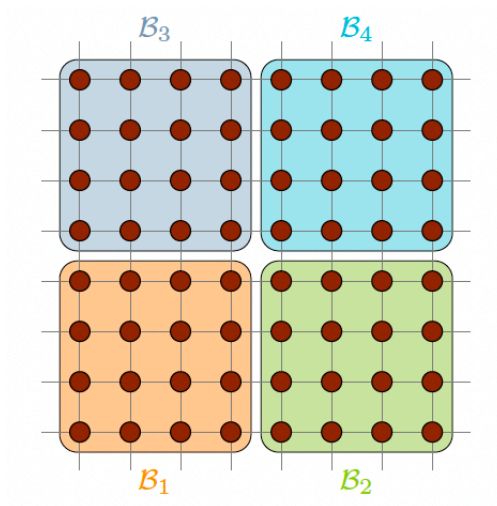
Solve first **even** then **odd**

- Iteratively solving $S_c x_e = b_e - A_{eo}y_o \implies$ odd-even preconditioner
- If A has constant diagonal $\kappa(S_c) < \kappa(A) \implies S_c$ is easier than solving A
- Since A_{oo}^{-1} is cheap (diagonal). \Rightarrow Cost for $S_c \approx$ Cost for A

Domain Decomposition

Idea:

- Split the computational domain into subdomains B_i
- Solve system iteratively on each subdomain



- Canonical injection I_j

$$I_j e_i = e_i^{B_j}$$

- Restriction of x onto B_j

$$x^{B_j} = I_j^\dagger x$$

- Restriction of A onto B_j

$$A^{B_j} = I_j^\dagger A I_j$$

Additive and Multiplicative Schwarz

Additive Schwarz:

Solve each block independently from each other:

```
for k=0,1,... do
  r[k] = b - A*x[k]
  for j=1,2,...,nb do
    #applied on Block j
    x[k+1] = x[k] + inv(Bj)r[k]
  end for
end for
```

- Block Jacobi-method
- embarrassingly parallel

Schwarz methods in general are:

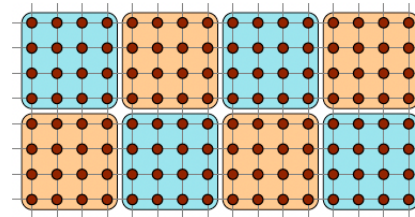
- data parallel
- computational parallel

Multiplicative Schwarz

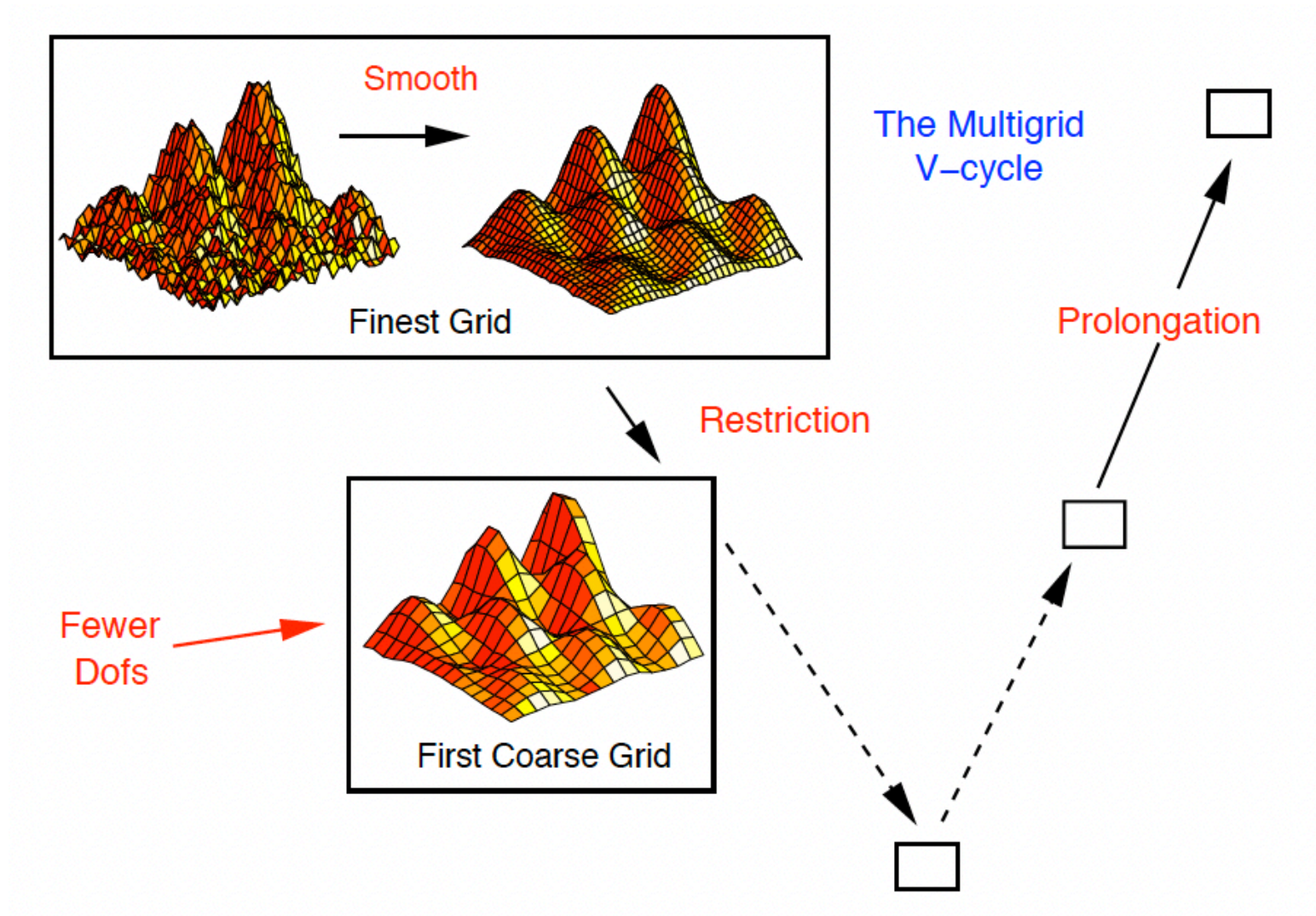
Update residual during iteration

```
for k=0,1,... do
  for j=1,2,...,nb do
    # update residual before each block app
    r = b - Ax
    #applied on Block j
    x[k+1] = x[k] + inv(Bj)r[k]
  end for
end for
```

- Block-Gauss-Seidel-method
- Sequential (\rightarrow coloring)
- SAP is using only two kind of blocks (red-black)



Multigrid



(Algebraic) Multigrid

Given

- $Ax = b$
- Iterative method S ("smoother")

Needed

- Hierarchy of systems of $L + 1$ levels (finest $l = 0$, coarsest level $l = L$)

$$A_l x_l = b_l, \quad l = 0, \dots, L$$

- inter-grid transfer operators

Projection from l to $l+1$: $p^{l,l+1} : \mathbb{C}^{n_{l+1}} \rightarrow \mathbb{C}^{n_l}$

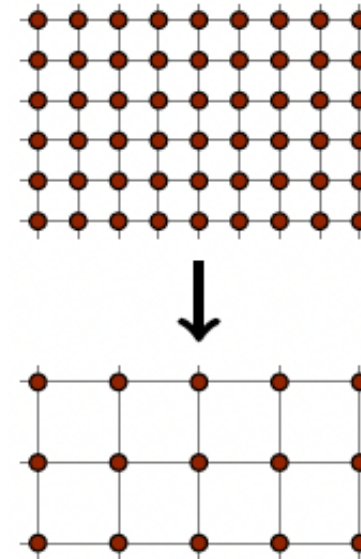
Restriction from $l+1$ to l : $p^{l+1,l} : \mathbb{C}^{n_l} \rightarrow \mathbb{C}^{n_{l+1}}$

Smoother: "High modes"

$$S_l : \mathbb{C}^{n_l} \rightarrow \mathbb{C}^{n_l}$$

Interpolation: "Low modes"

$$p^{l,l+1} : \mathbb{C}^{n_{l+1}} \rightarrow \mathbb{C}^{n_l}$$



Generic Multigrid Algorithm - $MG_l(A_l, b_l)$

Preconditioner with smoothing and coarse-grid correction

```
if l=L then
    x_L = inv(A_L) b_L
else
    x_l = 0
    for i=1, ... v1 do
        x_l = Sl(x_l, b_l)    # Pre-smoothing
    end for

    x_l+1 = MG(A_l+1, R_l, l+1(b_l - Ax_l))
    x_l = x_l + P_l, l+1 * x_l+1    # Coarse-grid corrections

    for i=1, ... v2 do
        x_l = Sl(x_l, b_l)    # Post-smoothing
    end for
end if
```

The preconditioner might be an iterative process by itself

- S will change in every iteration. There is no longer a Krylov subspace defined by

$$K_l(SA, b) = (n, SA b, (SA)^2 b, \dots, (SA)^{k-1} b)$$

→ algorithmic have to be modified, namely flexible

Flexible GMRES

Preconditioner similar to before, but the projection of the iterate has to be modified Z_m

```
while not converged do
  r0 = S(b-A*x0), beta = ||r0||, v1=r0/beta # modified starting conditions
  for j=1, ...,m do
    z[j] = Sj * y[j]      # application of Preconditioner
    w = A *z[j]           # Increase Krylov space j → j+1

    for i=1, ...,j do      # Arnoldi Gram-Schmidt orthogonalization
      h[i,j] = (w,v[j])    # calculate elements of Hessenberg matrix
      w = w-h[i,j]*v[j]    # orthogonalize w wrt v[j]
    end for

    h[j+1,j] = ||w||       # lower diagonal of Hessenberg
    v[j+1] = w/h[j+1,j]
  end for

  Define Zm =[z[1], ..z[m]], H={h[i,j]} # Use Z here instead of v
  Solve ym = argmin ||beta*e1 - H*y|| #Impose Petrov-Garlekin condition (can be done via LU)
  x0=x0+Zm*ym # update iterate
end while
```

Note: GCR doesn't need to be modified to be flexible, *flexible per definition*

- note that inexact deflation can be converted into a 2 lvl-MG with a GCR

Conclusion and outlook

- Computational cost of conventional solvers rapidly grows with $(\alpha m_q)^{-1}$, system is quickly ill conditioned
- Multi-grid solvers are practically solving this issues
 - lead to a speed up of $O(10)$ - $O(100)$

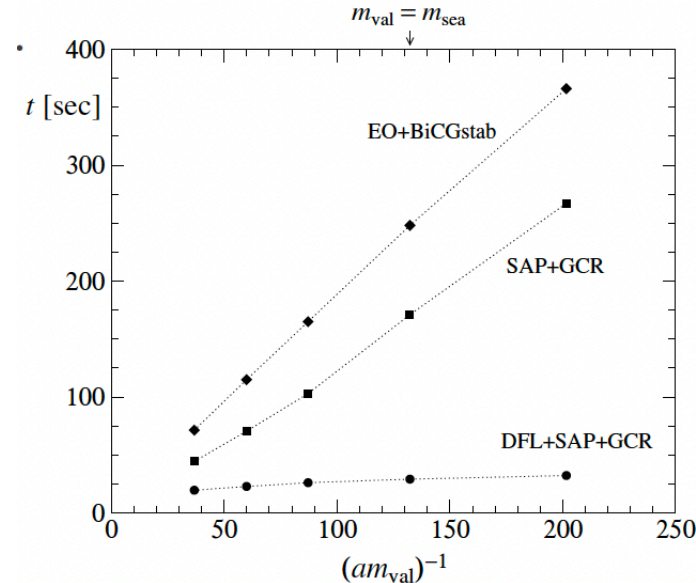
Challenges

multi-grid solver are parameter rich

- some are performance critical like
 - blocking size, projection size
- some are less performance critical
 - smoothing applications, iterations
- less scalable (coarse grid boundary surface becomes large)
- memory bound

... more ..

Usually main computational challenge:



[Luscher, 2007]

multi-grid is computational limited by

Reference

- Gustavo Ramirez Lattice Practice 2024 which is partly based also on Input by Andreas Frommer and Karsten Kahl
- Yousef Saad, Iterative Methods for Sparse Linear systems, Society for Industrial and Applied Mathematics, 2nd edition, 2003.
- A. Greenbaum, Iterative Methods for Solving Linear Systems, volume 17 of Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, 1997.
- M. Hestenes and E. Stiefels. Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards, Section B, 49, 1952.
- C. Jagels and L. Reichel. A fast minimal residual algorithm for shifted unitary matrices. Numer. Linear Algebra Appl., 1, 1994.

Notations

Linear systems of equations:

$$\sum_j^n a_{ij}x_j = b_i, \quad i = 1, \dots, n$$

$$Ax = b, \quad A \in \mathbb{C}^{n \times n}, x \in \mathbb{C}^n, b \in \mathbb{C}^n$$

Euclidian inner product:

$$\langle x, y \rangle_2 = y^\dagger x = \sum_{i=1}^n \bar{y}_i x_i$$

Adjoint A^\dagger of A w.r.t $\langle \dots \rangle_2$

$$\langle Ax, y \rangle_2 = \langle x, A^\dagger y \rangle_2$$

For complex matrices: A hermitian $\Leftrightarrow A^\dagger = A$, For real matrices: A is symmetric $\Leftrightarrow A^T = A$

A hermitian positive definite

$$A^\dagger = A \quad \text{and} \quad x^\dagger Ax > 0, \quad x \neq 0$$

Preconditioners - Summary

Preconditioning improves convergence if

$$\kappa(SA) \ll \kappa(A)$$

There is a wide variety of preconditioners available

- most of them require knowledge about A or its origins

Goals when constructing preconditioners S are

- $S \approx A^{-1}$ and S needs to be computationally affordable

In general Preconditioning makes Krylov subspace methods **more robust**

- Reducing $\kappa(A)$ helps controlling the error $e^{(k)}$, since

$$\|e\|_2 \leq c\kappa(A)^{-1}\|r\|_2$$

\implies : If $\kappa(A) \gg 1$ results based on $\|r\|_2$ can be unreliable

\implies : If $\kappa(A) \gg 1$ a preconditioner becomes necessary