

ISC 4933/5935
Iterative and Direct Solvers for Linear Systems
Lecture Notes

Bryan Quaife and Eitan Lees

Spring, 2019

Contents

1	Introduction to Linear Systems	3
1.1	Operation Counts	5
1.2	LU Decomposition	8
1.3	Spectral Decomposition	9
1.4	Schur Compliment	10
1.5	Woodbury and Sherman-Morrison Identities	11
2	Geometric Multigrid	13
2.1	Finite Difference for Elliptic PDEs	13
2.2	Fixed Point Methods for Linear Systems	14
2.3	Properties of weighted Jacobi	20
2.4	Two-Grid Multigrid	24
2.5	Recursive Multigrid	28
2.6	Multigrid Cycles	31
2.7	Multigrid in \mathbb{R}^2	33
3	Randomized SVD	35
3.1	The Singular Value Decomposition (SVD)	36
3.2	Low Rank Approximations	37
3.3	Randomized Algorithms	40
3.4	Interpolative Decomposition	46
3.5	Gram-Schmidt Process	47
3.6	The Fixed-Precision Problem	48
4	Structured Matrices	49
4.1	Recursive Off Diagonal Partitioning	50
4.2	Operations with Partitioned Matrices	51
4.3	Kernel Equations	55

4.4	Hierarchical Semi-Separable Decomposition	59
5	Krylov Methods	67
5.1	Making $A\vec{x} = \vec{b}$ a Minimization Problem	68
5.2	Method of Steepest Decent and Line Search	69
5.3	A better choice for the search direction	73
5.4	Convergence	77
6	Preconditioners	79
6.1	Domain Decomposition	81
6.1.1	Discretization Coupling	83
6.1.2	Additive and Multiplicative Schwarz	87
6.1.3	Two-Scale Domain Decomposition	89
	References	90

1 Introduction to Linear Systems

A linear system is a set of equations of the form

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

We are typically given a_{ij} and one of x_j or b_i . Our job is to find the other. In matrix form,

$$A\vec{x} = \vec{b}$$

where $A \in \mathbb{R}^{m \times n}$, $\vec{x} \in \mathbb{R}^n$, $\vec{b} \in \mathbb{R}^m$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Linear systems arise when we

- Discretize a linear differential equation or integral equation.
- Linearizing a nonlinear differential equation or a nonlinear integral equation.
- Interpolating data with basis functions such as polynomials, wavelets, Fourier expansions.
- Optimizing an objective function $f(\vec{x})$.
- Statistical operations such as linear regression.
- Solve an inverse problem.
- Data sciences.

With $A \in \mathbb{R}^{m \times n}$, we have 3 cases

- 1) $m < n$
- 2) $m > n$
- 3) $m = n$

If $m < n$, we have fewer equations than unknowns. Such a system typically has infinitely many solutions, and such a system is said to be *underdetermined*. It can also have no solutions.

EX:

$x_1 + x_2 + 2x_3 = 0$	$x_1 + x_2 - 2x_3 = 0$
$x_1 - 2x_2 + 3x_3 = 1$	$2x_1 + 2x_2 - 4x_3 = 7$
infinitely many solutions	no solutions

Underdetermined problems arise, for example, in inverse problems. Here, there is typically some kind of field that is unknown, and we only have a few measurements. For example in seismic, the distribution of the formations underground are predicted from only a few available measurements.

If we have $m > n$, we have more equations than unknowns. Such a system typically has no solution, and such a system is said to be *overdetermined*. It can also have 1 solution or infinitely many solutions.

EX:

$x + y = 1$	$x + y = 1$	$x + y = 1$
$2x + 2y = 2$	$2x + 2y = 2$	$x - y = 2$
$3x + 3y = 3$	$3x + 4y = 3$	$3x - y = 4$
infinitely many solutions	one solutions	no solutions

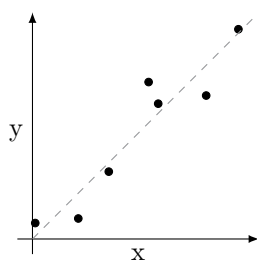
Overdetermined problems arise, for example, in interpolation. Suppose we are given data points (x_i, y_i) , $i = 1, \dots, m$ and we seek a linear function interpolating this data. Suppose the interpolant is $f(x) = c_0 + c_1x$

$$c_0 + c_1x_1 = y_1$$

$$c_0 + c_1x_2 = y_2$$

\vdots

$$c_0 + c_1x_m = y_m$$



Let's focus on the case $m = n$ and suppose that $A\vec{x} = \vec{b}$ has a unique solution for every \vec{b} . The three main tasks to be done are

- 1) Compute $A\vec{x}$ for some $\vec{x} \in \mathbb{R}^n$
- 2) Solve $A\vec{x} = \vec{b}$ for some $\vec{b} \in \mathbb{R}^n$
- 3) Decompose A as

$A = LU$	Lower-Upper Decomposition
$A = QR$	where Q is orthogonal ($Q^T = Q^{-1}$) and R is upper triangular
$A = U\Sigma V^T$	Singular Value Decomposition where U, V are orthogonal and Σ is diagonal
$A = VDV^{-1}$	Eigenvalue Decomposition
$A = A_{(:,J)}X$	Interpolative Decomposition

1.1 Operation Counts

Performing each of these tasks requires some number of floating point operations (flops). For example to compute $A\vec{x} = \vec{y}$ we would do the following For each i ,

```

for  $i = 1, \dots, n$  do
   $y_i = 0$ 
  for  $j = 1, \dots, n$  do
     $y_i = y_i + a_{ij}x_j$ 
  end for
end for

```

computing y_i requires n additions and n multiplications. This must be done for each $i = 1, \dots, n$

\Rightarrow We require $n \cdot 2n = 2n^2$ flops

We are interested in the trend of the number of flops. Therefore, the 2 doesn't really matter. We write the following

\Rightarrow The number of flops to compute $A\vec{x}$ is $\mathcal{O}(n^2)$

The way to interpret the significance is to ask how many more flops are required if n is doubled. For the problem of size n , we require Cn^2 operations. If n is doubled, we require $C(2n)^2 = 4Cn^2$

\Rightarrow The increase in the amount of flops is $\frac{4Cn^2}{Cn^2} = 4$

As we might expect, solving $A\vec{x} = \vec{b}$ is more expensive than doing matrix-vector multiplication. Suppose we use Gaussian elimination. Then, solving $A\vec{x} = \vec{b}$ is equivalent to reducing A to

1. Reduced row echelon form
2. Row echelon form and use back substitution

Let's find the complexity (number of flops/operations) to reduce A to row echelon form. Suppose row swaps are not required

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Replace row 2 with

$$\frac{a_{21}}{a_{11}} \times (\text{row 1}) - (\text{row 2})$$

This requires $\mathcal{O}(n)$ flops

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

We next replace row 3 with

$$\frac{a_{31}}{a_{11}} \times (\text{row 1}) - (\text{row 3})$$

This requires $\mathcal{O}(n)$ flops

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Continuing in this manner for all rows,
we require $\mathcal{O}(n)$ operations $n - 1$ times
 $\Rightarrow \mathcal{O}(n^2)$ operations

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Moving to the 2nd column, we use a_{22} as
a pivot to make $a_{i2} = 0$ for $i = 3, \dots, n$

The first step is to replace row 3 with

$$\frac{a_{32}}{a_{22}}(\text{row 2}) - (\text{row 3})$$

This requires $\mathcal{O}(n - 1)$ operations.

Repeating for each row means we require $\mathcal{O}((n - 1)(n - 2))$ operations to make the second column to be in the row echelon form. Continuing for each column, we require

$$\begin{array}{ll}
\mathcal{O}(n^2) & \text{operations for column 1} \\
\mathcal{O}((n-1)^2) & \text{operations for column 2} \\
\mathcal{O}((n-2)^2) & \text{operations for column 3} \\
\vdots & \\
\mathcal{O}(1^2) & \text{operations for column } (n-1)
\end{array}$$

\Rightarrow The total cost of reducing A to row echelon form is

$$\begin{aligned}
Cn^2 + C(n-1)^2 + C(n-2)^2 + \dots + C2^2 + C1^2 &= C \sum_{j=1}^n j^2 \\
&= C \frac{n(n+1)(2n+1)}{6} = \mathcal{O}(n^3) \text{ flops}
\end{aligned}$$

Reducing A further to reduce row echelon form requires another $\mathcal{O}(n^3)$ operations. Alternatively, we can solve

$$U\vec{x} = \vec{b}$$

where U is the row echelon form of A , and \vec{b} has also been updated by the Gaussian elimination procedure.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

$$x = \frac{b_n}{u_{nn}}$$

$$u_{(n-1)(n-1)}x_{n-1} + u_{(n-1)n}x_n = b_{n-1} \quad \Rightarrow \quad x_{n-1} = \frac{b_{n-1} - u_{(n-1)n}x_n}{u_{(n-1)(n-1)}}$$

\vdots

$$x_k = \frac{b_k - \sum_{j=k+1}^n u_{kj}x_j}{u_{kk}} \quad k = n, n-1, \dots, 3, 2, 1$$

Solving for x_k requires $\mathcal{O}(n-k)$ operations to compute the summation. Therefore, solving for \vec{x} requires

$$\mathcal{O}(1 + 2 + 3 + \dots + n) = \mathcal{O}\left(\frac{n(n+1)}{2}\right) = \mathcal{O}(n^2)$$

Therefore, it is more efficient to reduce A to row echelon form and apply back substitution than it is to reduce A to reduced row echelon form. However, since reducing A to row echelon form requires $\mathcal{O}(n^3)$ operations, the overall cost of solving $A\vec{x} = \vec{b}$ is $\mathcal{O}(n^3)$.

1.2 LU Decomposition

An alternative approach to solve $A\vec{x} = \vec{b}$ is to decompose A into factors, where each factor is more efficient to work with. For instance, suppose we decompose A as

$$A = LU$$

where L is lower triangular and U is upper triangular. Then to solve $A\vec{x} = \vec{b}$, we can instead solve

$$LU\vec{x} = \vec{b}$$

Letting $\vec{y} = U\vec{x}$, this is equivalent to solving

$$\begin{aligned} L\vec{y} &= \vec{b} \\ U\vec{x} &= \vec{y} \end{aligned}$$

Solving each of these linear systems requires $\mathcal{O}(n^2)$ flops using backward/forward substitution. However, we need to find L and U . Unfortunately, the LU decomposition uses Gaussian elimination which we know requires $\mathcal{O}(n^3)$ operations.

One instance where it makes sense to use LU is when solving $A\vec{x}_k = \vec{b}_k$ for many right hand sides \vec{b}_k . We can then do the following. If there are M right

Find L, U with $A = UL$	$\mathcal{O}(n^3)$
for $k = 1, 2, 3, \dots$ do	
Solve $L\vec{y} = \vec{b}$	$\mathcal{O}(n^2)$
Solve $U\vec{x} = \vec{y}$	$\mathcal{O}(n^2)$
end for	

hand sides, this requires $\mathcal{O}(n^3 + Mn^2)$ operations, while applying Gaussian elimination requires $\mathcal{O}(Mn^3)$ operations.

Another way to solve $A\vec{x} = \vec{b}$ is to use Cramer's Rule. Suppose $A\vec{x} = \vec{b}$. Then, the i^{th} entry of \vec{x} is

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i is the matrix A with it's i^{th} column replaced with \vec{b} . Therefore, to find \vec{x} , we have to compute $n+1$ determinates. The method you learnt in linear algebra is called the Laplace expansion. For $n = 3$, this is

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

\Rightarrow We have to compute 3 2×2 determinates.

If $n = 4$, we have to compute 4 3×3 determinates, or $4 \times 3 = 12$ 2×2 determinates. For an arbitrary n , we have to compute

$$\begin{array}{l} n \quad (n-1) \times (n-1) \quad \text{determinates} \\ \parallel \\ n(n-1) \quad (n-2) \times (n-2) \quad \text{determinates} \\ \parallel \quad \vdots \\ n(n-1)(n-2) \dots (3) \quad 2 \times 2 \quad \text{determinates} \end{array}$$

\Rightarrow We require $\mathcal{O}(n!)$ operations

Fortunately, computing $\det(A)$ is not an NP-hard problem. We can instead factor A as $A = LU$ and use the property of determinates.

$$\begin{aligned} \det(A) &= \det(LU) = \det(L) \times \det(U) \\ &= \prod_{k=1}^n l_{kk} \times \prod_{k=1}^n u_{kk} \end{aligned}$$

1.3 Spectral Decomposition

Another task that requires a determinant is the eigenvalue/eigenvector decomposition. Given $A \in \mathbb{R}^{n \times m}$, an eigenvalue λ and a corresponding eigenvector $\vec{x} \neq 0$ satisfies

$$A\vec{x} = \lambda\vec{x} \Rightarrow A\vec{x} = \lambda I\vec{x} \Rightarrow (A - \lambda I)\vec{x} = \vec{0}$$

Since $\vec{x} \neq 0$, we require that $A - \lambda I$ is not invertible

$$\det(A - \lambda I) = 0$$

This is always a polynomial of degree n , $p(\lambda)$, and we require $p(\lambda) = 0$. Suppose

$$\begin{aligned} A &= \begin{bmatrix} -2 & -4 & 2 \\ -2 & 1 & 2 \\ 4 & 2 & 5 \end{bmatrix} \Rightarrow \det(A - \lambda I) = \begin{vmatrix} -2-\lambda & -4 & 2 \\ -2 & 1-\lambda & 2 \\ 4 & 2 & 5-\lambda \end{vmatrix} \\ &\Rightarrow (-2-\lambda) \begin{vmatrix} 1-\lambda & 2 \\ 2 & 5-\lambda \end{vmatrix} + 4 \begin{vmatrix} -2 & 2 \\ 4 & 5-\lambda \end{vmatrix} + 2 \begin{vmatrix} -2 & 1-\lambda \\ 4 & 2 \end{vmatrix} = 0 \\ &\Rightarrow (-2\lambda)(\lambda^2 - 6\lambda + 1) + 4(2\lambda - 18) + 2(-4\lambda - 8) = 0 \\ &\Rightarrow -\lambda^3 + 4\lambda^2 + 27\lambda - 90 = 0 \\ &\Rightarrow (\lambda - 3)(\lambda + 5)(\lambda - 6) = 0 \\ &\Rightarrow \text{The eigenvalues are } \lambda = 3, -5, 6 \end{aligned}$$

To find the eigenvectors, say the one corresponding to $\lambda = 3$

$$(A - \lambda I)\vec{x} = \vec{0} \Rightarrow (A - 3I)\vec{x} = \vec{0}$$

$$\begin{bmatrix} -5 & -4 & 2 \\ -2 & -2 & 2 \\ 4 & 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & -1 \\ -5 & -4 & 2 \\ 2 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & -1 \\ 0 & 1 & -3 \\ 0 & -1 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & -1 \\ 0 & 1 & -3 \\ 0 & 0 & 0 \end{bmatrix}$$

$\Rightarrow x_3$ is free. Say $x_3 = 1$.

$$\begin{aligned} x_2 &= 3x_3 \Rightarrow x_2 = 3 \\ x_1 &= -x_2 + x_3 = -3 + 1 = -2 \\ \Rightarrow \vec{x} &= \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix} \end{aligned}$$

We can easily check that $A\vec{x} = 3\vec{x}$

To find all eigenvalues we must

- 1) Compute an $n \times n$ determinate
- 2) Find all roots, which may be complex, of a degree n polynomial

To find all of the eigenvectors, we must perform n linear solves, each of size $n \times n$. Ignoring the root finding problem, this requires at least $n \cdot \mathcal{O}(n^3) = \mathcal{O}(n^4)$ operations. To find the roots of $p(\lambda)$, we must use an iterative solver if $n \leq 5$ (thanks to Galois). Therefore, we must use an iterative method such as *Newton's method*.

Every algorithm we've considered so far is what your Linear Algebra instructor taught you, but they are all impractical for large n .

1.4 Schur Complement

We often solve linear systems with a natural partitioning

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} = \begin{bmatrix} \vec{b}_x \\ \vec{b}_y \end{bmatrix}$$

where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{m \times n}$, $D \in \mathbb{R}^{m \times m}$, $\vec{x} \in \mathbb{R}^n$, $\vec{y} \in \mathbb{R}^m$, $\vec{b}_x \in \mathbb{R}^n$, $\vec{b}_y \in \mathbb{R}^m$,

If D is invertible, then the 2nd row tells us

$$C\vec{x} + D\vec{y} = \vec{b}_y \Rightarrow \vec{y} = D^{-1}(\vec{b}_y - C\vec{x})$$

Substituting this expression into the 1st equation

$$\begin{aligned} A\vec{x} + B\vec{y} &= \vec{b}_x \Rightarrow A\vec{x} + BD^{-1}(\vec{b}_y - C\vec{x}) = \vec{b}_x \\ &\Rightarrow (A - BD^{-1}C)\vec{x} = \vec{b}_x - BD^{-1}\vec{b}_y \end{aligned}$$

The matrix $A - BD^{-1}C$ is called the *Schur complement* of the block D .

Similarly, if A is invertible, then

$$\begin{aligned} A\vec{x} + B\vec{y} &= \vec{b}_x \Rightarrow \vec{x} = A^{-1}(\vec{b}_x - B\vec{y}) \\ C\vec{x} + D\vec{y} &= \vec{b}_y \Rightarrow CA^{-1}(\vec{b}_x - B\vec{y}) + D\vec{y} = \vec{b}_y \\ &\Rightarrow (D - CA^{-1}B)\vec{y} = \vec{b}_y - CA^{-1}\vec{b}_x \end{aligned}$$

The matrix $D - CA^{-1}B$ is called the *Schur complement* of the block A . Assuming D and its Schur complement $(A - BD^{-1}C)$ are invertible, we can solve for \vec{x} by

1) Form $D^{-1}C$ and $D^{-1}\vec{b}_y$
($m + 1$) applications of D^{-1}

2) Solve

$$(A - BD^{-1}C)\vec{x} = \vec{b}_x - BD^{-1}\vec{b}_y = \vec{b}$$

($n \times n$) linear solve.

Since D is $m \times m$ and $A - BD^{-1}C$ is $n \times n$, these are smaller linear systems than the one to find \vec{x} and \vec{y} simultaneously.

1.5 Woodbury and Sherman-Morrison Identities

Consider the matrix equation

$$\begin{bmatrix} A & U \\ V & -C^{-1} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix}$$

where $A \in \mathbb{R}^{n \times n}$, $U \in \mathbb{R}^{n \times k}$, $V \in \mathbb{R}^{k \times n}$, $C \in \mathbb{R}^{k \times k}$, $X \in \mathbb{R}^{n \times n}$, $Y \in \mathbb{R}^{k \times n}$, $I \in \mathbb{R}^{n \times n}$, $0 \in \mathbb{R}^{k \times n}$, Typically $k \ll n$.

We can use the Schur complement to find X

$$VX - C^{-1}Y = 0 \Rightarrow Y = CVX$$

Using the first equation

$$\begin{aligned} AX + UY &= I \Rightarrow AX + UCVX = I \\ &\Rightarrow (A + UCV)X = I \\ &\Rightarrow X = (A + UCV)^{-1} \end{aligned}$$

That is, X is the inverse of $A + UCV$.

Next, we compute the Schur complement of the $(1, 1)$ block

$$AX + UY = I \Rightarrow X = A^{-1}(I - UY)$$

Substituting into the second equation

$$\begin{aligned}
VX - C^{-1}Y &= 0 \Rightarrow VA^{-1}(I - UY) - C^{-1}Y = 0 \\
&\Rightarrow \underbrace{(-VA^{-1}U - C^{-1})Y}_{\text{Schur complement of the (1,1) block}} = -VA^{-1} \\
&\Rightarrow Y = (C^{-1} + VA^{-1}U)^{-1}VA^{-1}
\end{aligned}$$

Substituting this Y back into the first equation

$$\begin{aligned}
AX + U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} &= I \\
\Rightarrow AX &= I - U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \\
\Rightarrow X &= A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}
\end{aligned}$$

Recalling that $X = (A + UCV)^{-1}$, we have

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

This is called the *Woodbury identity*.

It is useful if

- 1) k is sufficiently small
- 2) A fast way to apply A^{-1} .

To apply the Woodbury identity, we require

$k + 1$ applications of A^{-1}

$2 \times k$ linear solves

This identity is useful since we often have a solver for $A\vec{x} = \vec{b}$, and then we need to invert a low rank modification of A .

The most extreme case is if $k = 1$. We have $\vec{u}, \vec{v} \in \mathbb{R}^{n \times 1}$. We can assume $c = 1$ since any other non-zero value for c can be rolled into \vec{u} (or \vec{v})

$$\begin{aligned}
\Rightarrow (A + \vec{u}\vec{v}^T)^{-1} &= A^{-1} - A^{-1}\vec{u}\underbrace{(1 + \vec{v}^T A^{-1}\vec{u})}_{\mathbb{R}}^{-1}\vec{v}^T A^{-1} \\
&= A^{-1} - \frac{A^{-1}\vec{u}\vec{v}^T A^{-1}}{1 + \vec{v}^T A^{-1}\vec{u}}
\end{aligned}$$

This is the *Sherman-Morrison* identity. Sometimes it is called the *SMW* or *Sherman-Morrison-Woodbury* identity.

2 Geometric Multigrid

A popular iterative method for numerically solving differential equations is known as the *Multigrid Method*. We build up to the method by first exploring a variety of simpler methods. Many of the notes from this section are from the book by Briggs, Henson, and McCormick, *A Multigrid Tutorial, 2nd Edition*.

2.1 Finite Difference for Elliptic PDEs

Consider the BVP

$$\begin{aligned} -u''(x) + \sigma u(x) &= f(x) & x \in (0, 1), \sigma \geq 0 \\ u(0) &= u(1) = 0 \end{aligned}$$

Discretize at the n interior points

$$x_i = ih, \quad i = 1, \dots, n \quad \text{and} \quad h = \frac{1}{n+1}$$

Using the standard centered difference for $u''(x)$,

$$u''(x_i) \approx \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1}))}{h^2},$$

we can form an approximate solution to our BVP by solving

$$\begin{aligned} -\left(\frac{u_0^0 - 2u_1 + u_2}{h^2}\right) + \sigma u_1 &= f_1 \quad (= f(x_1)) \\ -\left(\frac{u_1 - 2u_2 + u_3}{h^2}\right) + \sigma u_2 &= f_2 \\ &\vdots \\ -\left(\frac{u_{n-1} - 2u_n + u_{n+1}^0}{h^2}\right) + \sigma u_n &= f_n \end{aligned}$$

$$\Rightarrow \frac{1}{h^2} \begin{bmatrix} 2 + \sigma h^2 & -1 & & 0 \\ -1 & 2 + \sigma h^2 & -1 & \\ & \ddots & \ddots & \ddots \\ 0 & & -1 & 2 + \sigma h^2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

Therefore, there is a need to develop efficient solvers for the matrix

$$A = \begin{bmatrix} 2 + \sigma h^2 & -1 & & 0 \\ -1 & 2 + \sigma h^2 & -1 & \\ & \ddots & \ddots & \ddots \\ 0 & & -1 & 2 + \sigma h^2 \end{bmatrix}$$

One way to solve this linear system with $\mathcal{O}(n)$ operations is the *Thomas algorithm*. However, this method only works if A is tridiagonal. Therefore, it can not be applied to problems with

- 1) Wider stencils (higher-order)
- 2) Different boundary conditions
- 3) More than 1 spatial dimension

2.2 Fixed Point Methods for Linear Systems

Suppose we are solving $A\vec{u} = \vec{b}$. A fixed point method is an iterative method of the form

$$\vec{u}^{k+1} = f(\vec{u}^k)$$

such that, if $\vec{u}^k \rightarrow \vec{u}$, then $\vec{u} = f(\vec{u})$ and $A\vec{u} = \vec{b}$. Before giving examples of iterative methods, we describe the norm, error and the residual.

Suppose $A\vec{u} = \vec{b}$ and A is invertible. If we form an approximate solution \vec{v} of \vec{u} , then the error is

$$\vec{e} = \vec{u} - \vec{v}$$

To quantify how close \vec{v} is to \vec{u} , we use a norm. The most common norms are

$$\|\vec{e}\| = \|\vec{e}\|_2 = \left(\sum_{i=1}^n e_i^2 \right)^{1/2} \quad \text{and} \\ \|\vec{e}\|_\infty = \max_{1 \leq i \leq n} |e_i|$$

In general, we don't know the error \vec{e} since we don't know the exact solution \vec{u} . Instead, we can compute the *residual*

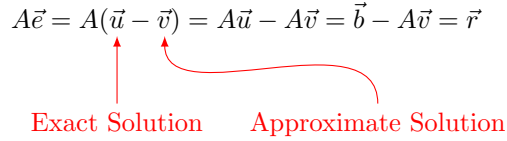
$$\vec{r} = \vec{b} - A\vec{v}$$

Then

$$\vec{r} = 0 \Leftrightarrow \vec{b} - A\vec{v} = 0 \Leftrightarrow A\vec{v} = \vec{b} \Leftrightarrow \vec{v} = \vec{u} \\ \Updownarrow \\ \vec{e} = 0$$

However, it is possible for $\|\vec{r}\|$ to be small, but non-zero, while $\|\vec{e}\|$ is large. Such a linear systems are called ill-conditioned.

An important relationship between the error \vec{e} and the residual \vec{r} is the following

$$A\vec{e} = A(\vec{u} - \vec{v}) = A\vec{u} - A\vec{v} = \vec{b} - A\vec{v} = \vec{r}$$


Exact Solution Approximate Solution

Therefore, we can improve on an approximate solution \vec{v} by

- 1) Computing the residual $\vec{r} = \vec{b} - A\vec{v}$
- 2) Approximately solve $A\vec{e} = \vec{r}$
- 3) Replace \vec{v} with $\vec{v} + \vec{e}$

We first need to decide how to approximately solve $A\vec{u} = \vec{b}$. This can be done with a fixed point iterative method such as

- 1) Jacobi and weighted Jacobi
- 2) Gauss-Seidel
- 3) SOR (Successive over relaxation)

Each of these methods can be described in terms of how they iterate on \vec{u}^k as a vector, or the components \vec{u}_j^k , $j = 1, \dots, n$.

In matrix-form, we write

$$A = D - L - U$$

where D has only diagonal terms, L only has terms below the main diagonal, and U only has terms above the main diagonal.

Ex

$$\begin{array}{c} \text{D} \qquad \qquad \qquad \text{L} \qquad \qquad \qquad \text{U} \\ \begin{bmatrix} 1 & -1 & 2 \\ 3 & 2 & -4 \\ -5 & 6 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ -5 & 6 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -1 & 2 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{aligned} A\vec{u} = \vec{b} &\Rightarrow (D - L - U)\vec{u} = \vec{b} \\ &\Rightarrow D\vec{u} = \vec{b} + (L + U)\vec{u} \\ &\Rightarrow \vec{u} = D^{-1} \left(\vec{b} + (L + U)\vec{u} \right) \end{aligned}$$

Therefore, we can define the iterative method

$$\vec{u}^{k+1} = D^{-1} \left(\vec{b} + (L + U)\vec{u}^k \right), \quad k = 1, 2, \dots$$

If this iteration converges to some \vec{u}^* , then $A\vec{u}^* = \vec{b}$ by construction.

We can also write this iteration as follows.

$$\begin{bmatrix} a_{11} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ \vdots \\ \vdots \\ u_n^{k+1} \end{bmatrix} = \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & 0 & \cdots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & 0 \end{bmatrix} \begin{bmatrix} u_1^k \\ \vdots \\ \vdots \\ u_n^k \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

$$\Rightarrow u_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} u_j^k \right), \quad i = 1, \dots, n$$

Note how u_i^{k+1} requires evaluating a sum of size $n - 1$. Therefore, the cost of applying this method requires $\mathcal{O}(n(n - 1)) = \mathcal{O}(n^2)$ operations per iterations. This iterative method is called the *Jacobi method*.

Weighted Jacobi slightly modifies Jacobi by using some of the Jacobi iterate and some of the previous iterate.

$$\vec{u}^{k+1} = \omega D^{-1} \left(\vec{b} + (L + U)\vec{u}^k \right) + (1 - \omega)\vec{u}^k \quad \omega \neq 0$$

or

$$u_i^{k+1} = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} u_j^k \right) + (1 - \omega) u_i^k, \quad i = 1, \dots, n$$

If it converges to some \vec{u}^* , then

$$\begin{aligned} \vec{u}^* &= \omega D^{-1} \left(\vec{b} + (L + U) \vec{u}^* \right) + (1 - \omega) \vec{u}^* \\ \Rightarrow D \vec{u}^* &= \omega \left(\vec{b} + (L + U) \vec{u}^* \right) + (1 - \omega) D \vec{u}^* \\ \Rightarrow D \vec{u}^* &\stackrel{0}{=} \omega \left(\vec{b} + (L + U) \vec{u}^* \right) + (1 - \omega) D \vec{u}^* \\ \Rightarrow D \vec{u}^* &= \left(\vec{b} + (L + U) \vec{u}^* \right) \Rightarrow A \vec{u}^* = \vec{b} \end{aligned}$$

Gauss-Seidel leaves the matrix $L + D$ on the left side, but moves U to the right side

$$\begin{aligned} A \vec{u} &= \vec{b} \\ \Rightarrow (D - L - U) \vec{u} &= \vec{b} \\ \Rightarrow (D - L) \vec{u} &= \vec{b} + U \vec{u} \end{aligned}$$

This motivated the fixed-point iterative method

$$\vec{u}^{k+1} = (D - L)^{-1} \left(\vec{b} + U \vec{u}^k \right)$$

Since $D - L$ is lower triangular applying $(D - L)^{-1}$ requires $\mathcal{O}(n^2)$ operations using back substitution. In component form

$$u_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{k+1} - \sum_{j=i+1}^n a_{ij} u_j^k \right), \quad i = 1, \dots, n$$

If one “implements” Jacobi, but overwrites u_i^k with u_i^{k+1} when looping over i , they are actually applying Gauss-Seidel. Gauss-Seidel also requires $\mathcal{O}(n^2)$ operations per iteration if A is dense.

Finally *successive over-relaxation*, (SOR), writes $A \vec{u} = \vec{b}$ as

$$\begin{aligned} (\omega D - \omega L - \omega U) \vec{u} &= \omega \vec{b}, \quad \omega > 1 \\ \Rightarrow (D - \omega L) \vec{u} &= \omega \vec{b} + \omega U \vec{u} - (\omega - 1) D \vec{u} \end{aligned}$$

This motivates the iterative method

$$\vec{u}^{k+1} = (D - \omega L)^{-1} \left(\omega \vec{b} + \omega U \vec{u}^k - (\omega - 1) D \vec{u}^k \right)$$

Similar to Gauss-Seidel, we have

$$\vec{u}^{k+1} = (1 - \omega) u_i^k + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{k+1} - \sum_{j=i+1}^n a_{ij} u_j^k \right) \quad i = 1, \dots, n$$

SOR also requires $\mathcal{O}(n^2)$ operations per iteration.

SOR to Gauss-Seidel is as weighted Jacobi is to Jacobi

Each of these methods may or may not converge. The convergence depends on the iterations matrix. For example, in Jacobi

$$\vec{u}^{k+1} = D^{-1} \left(\vec{b} + (L + U) \vec{u}^k \right), \quad k = 1, 2, \dots$$

The iteration matrix is $C = D^{-1}(L + U)$.

The reason C is important is that

$$\begin{aligned} \vec{u}^k &= D^{-1} \left(\vec{b} + (L + U) \vec{u}^{k-1} \right) \\ &= D^{-1} \vec{b} + C \vec{u}^{k-1} \\ &= D^{-1} \vec{b} + C \left(D^{-1} \vec{b} + C \vec{u}^{k-2} \right) \\ &= D^{-1} \vec{b} + C D^{-1} \vec{b} + C^2 \vec{u}^{k-2} \\ &= D^{-1} \vec{b} + C D^{-1} \vec{b} + C^2 D^{-1} \left(\vec{b} + (L + U) \vec{u}^{k-3} \right) \\ &= D^{-1} \vec{b} + C D^{-1} \vec{b} + C^2 D^{-1} \vec{b} + C^3 \vec{u}^{k-3} \\ &\vdots \end{aligned}$$

Therefore, as $k \rightarrow \infty$, we take larger powers of C . Assuming C has an eigenvalue/eigenvector decomposition

$$C = V \Lambda V^{-1}$$

where V is a matrix of eigenvectors, and Λ is a diagonal matrix filled with the corresponding eigenvalues.

$$\begin{aligned}
\Rightarrow C^k &= (V\Lambda V^{-1}) (V\Lambda V^{-1})^{\text{\textcolor{red}{k times}}} \dots (V\Lambda V^{-1}) \\
&= V\Lambda (V^{-1}V) \Lambda (V^{-1}V) \dots (V^{-1}V) \Lambda V^{-1} \\
&= V\Lambda^k V^{-1} = V \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix} V^{-1}
\end{aligned}$$

and this converges to the 0 matrix if the eigenvalues of C are all less than 1 in magnitude.

Definition

If C has eigenvalues $\lambda_1, \dots, \lambda_n$, we define the *spectral radius* of C to be

$$\rho(C) := \max\{|\lambda_1|, \dots, |\lambda_n|\}$$

Therefore, Jacobi will converge is

$$\rho(D^{-1}(L+U)) < 1$$

For weighted Jacobi

$$\vec{u}^{k+1} = \omega D^{-1}(\vec{b} + (L+U)\vec{u}^k) + (1-\omega)I\vec{u}^k$$

So the iteration matrix is

$$\begin{aligned}
C &= \omega D^{-1}(L+U) + (1-\omega)I \\
&= \omega D^{-1}(L+U-D+D) + (1-\omega)I \\
&= \omega D^{-1}(-A+D) + (1-\omega)I \\
&= -\omega D^{-1}A + \omega I + I - \omega I \\
&= I - \omega D^{-1}A
\end{aligned}$$

Therefore, weighted Jacobi converges if

$$\rho(I - \omega D^{-1}A) < 1$$

Recall that we are interested in solving $A\vec{u} = h^2 \vec{f}$ where

\vec{u} - approximate solution of our BVP at n equispaced interior discretization points

\vec{f} - the right hand side of the BVP at the same discretization points

$$A = \begin{bmatrix} 2 + \sigma h^2 & -1 & & & \\ & -1 & 2 + \sigma h^2 & -1 & \\ & & \ddots & \ddots & \\ 0 & & & -1 & 2 + \sigma h^2 \end{bmatrix}$$

For this matrix

$$D = \begin{bmatrix} 2 + \sigma h^2 & & \\ & \ddots & \\ & & 2 + \sigma h^2 \end{bmatrix}, -L = \begin{bmatrix} 0 & & \\ -1 & \ddots & \\ 0 & & -1 \end{bmatrix}$$

$$-U = -L^T$$

For this matrix, Jacobi is

$$u_i^{k+1} = \frac{1}{2 + \sigma h^2} (h^2 f_i + u_{i-1}^k + u_{i+1}^k), \quad i = 1, \dots, n$$

and weighted Jacobi

$$u_i^{k+1} = \frac{\omega}{2 + \sigma h^2} (h^2 f_i + u_{i-1}^k + u_{i+1}^k) + (1 - \omega)u_i^k, \quad i = 1, \dots, n$$

Note that applying 1 step of Jacobi or weighted Jacobi requires $\mathcal{O}(n)$ operations! This is because A is banded.

2.3 Properties of weighted Jacobi

Let's further analyze how weighted Jacobi performs when solving $A\vec{u} = h^2\vec{f}$ where A is the tridiagonal linear system. We can consider the case

$$\vec{f} = \vec{0} \text{ and } \vec{u} \text{ is random.}$$

From our numerical experiment, we saw that for some values of ω , the most oscillatory elements of the error decay quicker than the elements with very few oscillations.

Let's consider the case $\sigma = 0$

$$\begin{aligned} f_i &= 0 \quad \forall i = 1, \dots, n \\ u_i^k &= \sin\left(\frac{ij\pi}{n}\right) \quad i = 1, \dots, n \quad \text{for some } j = 1, \dots, n \end{aligned}$$

Then,

$$\begin{aligned} u_i^{k+1} &= \frac{\omega}{2 + \sigma h^2} (h^2 f_i + u_{i-1}^k + u_{i+1}^k) + (1 - \omega) u_i^k \\ &= \frac{\omega}{2} \left(\sin\left(\frac{(i-1)j\pi}{n}\right) + \sin\left(\frac{(i+1)j\pi}{n}\right) \right) + (1 - \omega) \sin\left(\frac{ij\pi}{n}\right) \end{aligned}$$

Using the identity

$$\sin(x + y) = \sin x \cos y + \cos x \sin y$$

$$\begin{aligned} u_i^{k+1} &= \frac{\omega}{2} \left(\sin\left(\frac{ij\pi}{n}\right) \cos\left(\frac{-j\pi}{n}\right) + \cancel{\cos\left(\frac{ij\pi}{n}\right) \sin\left(\frac{-j\pi}{n}\right)} \right. \\ &\quad \left. + \sin\left(\frac{ij\pi}{n}\right) \cos\left(\frac{j\pi}{n}\right) + \cancel{\cos\left(\frac{ij\pi}{n}\right) \sin\left(\frac{j\pi}{n}\right)} \right) \\ &\quad + (1 - \omega) \sin\left(\frac{ij\pi}{n}\right) \\ &= \omega \sin\left(\frac{ij\pi}{n}\right) \cos\left(\frac{j\pi}{n}\right) + (1 - \omega) \sin\left(\frac{ij\pi}{n}\right) \\ &= \left(1 - \omega + \omega \cos\left(\frac{j\pi}{n}\right) \right) \sin\left(\frac{ij\pi}{n}\right) \\ &= \left(1 - \omega + \omega \cos\left(2\left(\frac{j\pi}{2n}\right)\right) \right) \sin\left(\frac{ij\pi}{n}\right) \end{aligned}$$

Using the identity,

$$\cos(2\theta) = 1 - 2\sin^2 \theta = \cos^2 \theta - \sin^2 \theta$$

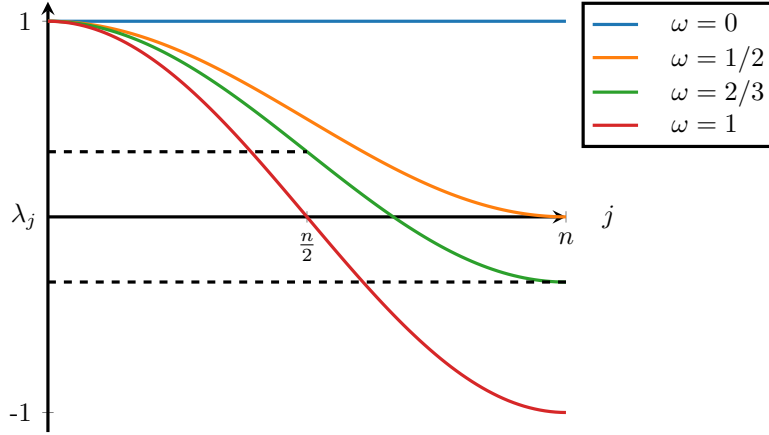
$$\begin{aligned}
u_i^{k+1} &= \left(1 - \omega + \omega \left(1 - 2 \sin^2 \left(\frac{j\pi}{2n} \right) \right) \right) \sin \left(\frac{ij\pi}{n} \right) \\
&= \left(1 - 2\omega \sin^2 \left(\frac{j\pi}{2n} \right) \right) \sin \left(\frac{ij\pi}{n} \right)
\end{aligned}$$

$\Rightarrow \sin \left(\frac{ij\pi}{n} \right) \quad i = 1, \dots, n$ is an eigenvector of weighted Jacobi with eigenvalue

$$1 - 2\omega \sin^2 \left(\frac{j\pi}{2n} \right) \quad \text{for all } j = 1, \dots, n.$$

Therefore, each time we apply weighted Jacobi to the j^{th} eigenvector, the error is reduced by a factor of

$$\lambda_j = 1 - 2\omega \sin^2 \left(\frac{j\pi}{2n} \right)$$



In multigrid, we want the most oscillatory elements of the error to be reduced the fastest. We identify high frequency terms with eigenvectors with $j > n/2$ and low frequency as eigenvectors with $j \leq n/2$

If we choose

$\omega = 1$: The $n/2$ frequency is eliminated after 1 iteration, but the highest oscillatory terms are damped slowly.

$\omega = 1/2$: The highest frequencies are eliminated very quickly, but the frequencies near $n/2$ decay slowly

If we make the $n/2^{\text{nd}}$ and the n^{th} eigenvalue the same in magnitude, then all frequencies in between will be less than this value in absolute value.

$$\begin{aligned}
\lambda_{n/2} &= -\lambda_n \Rightarrow \lambda_{n/2} + \lambda_n = 0 \\
&\Rightarrow \left(1 - 2 \sin^2 \left(\frac{\mathcal{N}/2 \cdot \pi}{2\mathcal{N}}\right)\right) + \left(\left(1 - 2 \sin^2 \left(\frac{\mathcal{N}\pi}{2\mathcal{N}}\right)\right)\right) = 0 \\
&\Rightarrow 1 - 2\omega \sin^2 \left(\frac{\pi}{4}\right) + 1 - 2\omega \sin^2 \left(\frac{\pi}{2}\right) = 0 \\
&\Rightarrow 1 - 2\omega \left(1/\sqrt{2}\right) + 1 - 2\omega(1)^2 = 0 \\
&\Rightarrow 2 - 3\omega = 0 \Rightarrow \omega = 2/3
\end{aligned}$$

We have only considered what happens if we apply weighted Jacobi to an eigenvector. But what happens if we apply it to a general vector \vec{u}^k . Since the eigenvectors are all linearly independent, there exists numbers c_j with

$$\begin{aligned}
\vec{u}_i^k &= \sum_{j=1}^n c_j \sin \left(\frac{ij\pi}{n}\right) \\
\Rightarrow \vec{u}_i^{k+1} &= \sum_{j=1}^n c_j \lambda_j \sin \left(\frac{ij\pi}{n}\right)
\end{aligned}$$

Therefore, the error will decay quickly in the high frequencies. However, once these frequencies are sufficiently small, the error will all be in the low frequencies. Since weighted Jacobi converges slowly at low frequencies, the iteration will significantly slow down.

How do we reduce the error in the low frequencies?

The key thing to recognize is that the notion of high and low frequency depend on the resolution. For example:

$$u_i = \sin \left(\frac{i \cdot 6 \cdot \pi}{n}\right) \quad i = 1, \dots, n$$

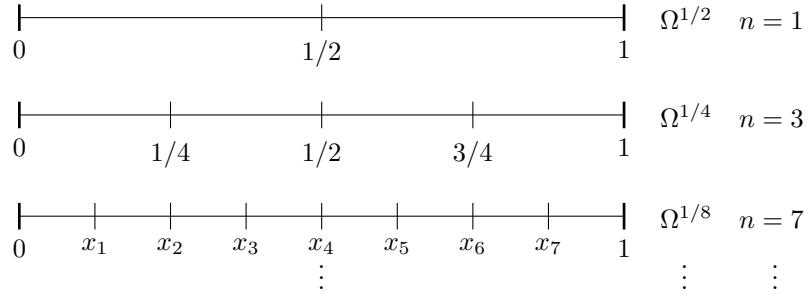
is considered to be low frequency if $n = 16$, but it is high frequency if $n = 8$. There fore we only need a method to solve $A\vec{u} = h^2 \vec{f}$ at multiple resolutions. That is, on multiple grids.

2.4 Two-Grid Multigrid

Some new notation we need is $\Omega = (0, 1)$ and Ω^h is the grid formed by dividing Ω into intervals of size h



We choose h to be of the form $h = 1/(n + 1)$ for some n that is a power of 2 minus 1.



A^h is the linear system at the grid resolution of Ω^h . For us $A^h \in \mathbb{R}^{n \times n}$

Ex:

$$A^{1/4} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \in \mathbb{R}^{3 \times 3}$$

$$A^{1/8} = \begin{bmatrix} 2 & -1 & & & & & & \\ -1 & 2 & -1 & & & & & 0 \\ & -1 & 2 & -1 & & & & \\ & & -1 & 2 & -1 & & & \\ & & & -1 & 2 & -1 & & \\ & 0 & & & -1 & 2 & -1 & \\ & & & & & -1 & 2 & \end{bmatrix} \in \mathbb{R}^{7 \times 7}$$

$\vec{u}^h, \vec{f}^h, \vec{r}^h, \vec{e}^h$ are all vectors on the grid Ω^h . Therefore, they are in \mathbb{R}^n

Instead of solving $A^h \vec{u}^h = h^2 \vec{f}^h$, lets redefine A^h to include the $1/h^2$ term so that we are solving

$$A^h \vec{u}^h = \vec{f}^h$$

A two-grid multigrid iteration works as follows. Start with an initial guess \vec{v}^h

- ① • Apply weighted Jacobi (smooth) to $A^h \vec{v}^h = \vec{f}^h$ with initial guess \vec{v}^h
 - Compute the residual

$$\vec{r}^h = \vec{f}^h - A^h \vec{v}^h$$

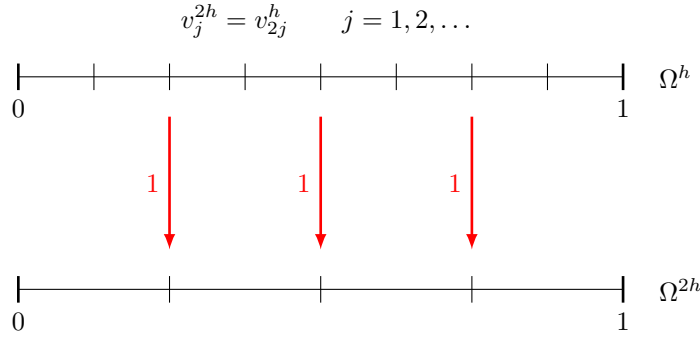
- ② • Move the residual to grid Ω^{2h} to form \vec{r}^{2h}
- ④ • Solve $A^{2h} \vec{e}^{2h} = \vec{r}^{2h}$
- ③ • Move the error to grid Ω^h to form \vec{e}^h
 - Correct the solution $\vec{v}^h = \vec{v}^h + \vec{e}^h$
- ① • Apply weighted Jacobi with the initial guess \vec{v}^h to form a smoother \vec{v}^h .

The 4 algorithms we require are

- 1) A smoother, like weighted Jacobi
- 2) A restriction operator to move from grid Ω^h to Ω^{2h}
- 3) A prolongation (interpolation) operator to move from grid Ω^{2h} to Ω^h
- 4) A solver for $A^{2h} \vec{e}^{2h} = \vec{r}^{2h}$

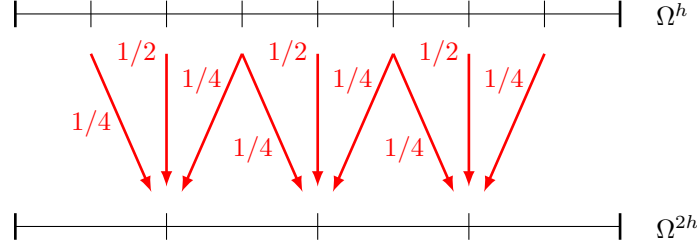
We must also decide how many times to smooth initially (pre-smoothing) and how many times to smooth after correcting \vec{v}^h with $\vec{v}^h + \vec{e}^h$ (post-smoothing). We'll denote these parameters as ν_1 , and ν_2 . Typically values for both ν_1 and ν_2 are 0, 1, or 2.

There are several choices for the restriction operator. The easiest is *interjection* where

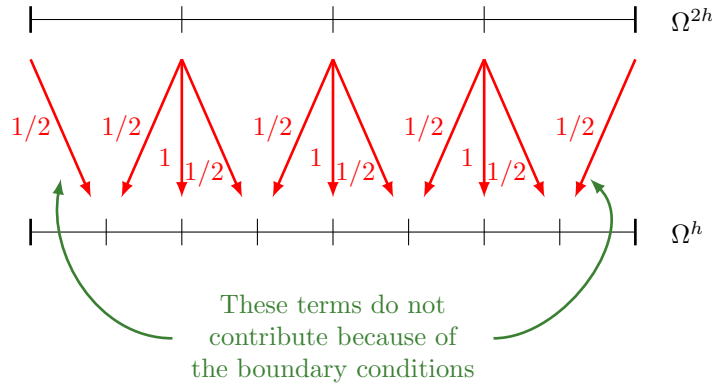


The problem with injection is that it losses information that is available at half the points of Ω^h . A better choice is *full weighting* where

$$v_j^{2h} = \frac{1}{4} (v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h)$$



For prolongation, we typically use a linear interpolant at the new grid points



We introduce the following notation

$\text{Smoother}(\vec{v}^h, \vec{f}^h) \rightarrow$ Apply 1 step of the smoother (Ex. weighted Jacobi) to the linear system $A^h \vec{u}^h = \vec{f}^h$ with initial guess \vec{v}^h

$\mathcal{I}_h^{2h} \rightarrow$ restrict a vector on grid Ω^h to a vector on Ω^{2h}

$\mathcal{I}_{2h}^h \rightarrow$ prolong a vector on grid Ω^{2h} to a vector on Ω^h

Therefore, our two-grid multigrid scheme is the following. Given an initial guess of \vec{v}^h of $A^h \vec{u} = \vec{f}^h$

- $\vec{v}^h = \text{Smoother}(\vec{v}^h, \vec{f}^h) \quad k = 1, \dots, \nu_1$
- $\vec{r}^h = \vec{f}^h - A^h \vec{v}^h$
- $\vec{r}^{2h} = \mathcal{I}_h^{2h} \vec{r}^h$
- Solve $A^{2h} \vec{e}^{2h} = \vec{r}^{2h}$ for \vec{e}^{2h}
- $\vec{e}^h = \mathcal{I}_{2h}^h \vec{e}^{2h}$
- $\vec{v}^h = \vec{v}^h + \vec{e}^h$
- $\vec{v}^h = \text{Smoother}(\vec{v}^h, \vec{f}^h) \quad k = 1, \dots, \nu_2$

Note that this is not a solver, but instead is an iterative method that takes an initial guess \vec{v}^h of $A^h \vec{u} = \vec{f}^h$, and hopefully returns a more accurate solution.

We can calculate the number of flops to apply this two-grid cycle once. The main steps are

- 1) Apply smoother $\nu_1 + \nu_2$ times. Since A is sparse (tridiagonal), this costs $\mathcal{O}((\nu_1 + \nu_2)n)$ operations.
- 2) Compute the residual $\vec{r}^h = \vec{f}^h - A^h \vec{v}^h$. This costs $\mathcal{O}(n)$ operations.
- 3) Apply the restriction and prolongation operators. Each of these steps requires visiting each of the n points once, and doing $\mathcal{O}(1)$ operations at each point. Therefore, prolongation and restriction cost $\mathcal{O}(n)$ operations.
- 4) Solve $A^{2h} \vec{e}^{2h} = \vec{r}^{2h}$. This cost depends on how we solve this system. It actually doesn't even have to be solved exactly.
 - a) Solve with Gaussian elimination/ LU / QR /etc. with a cost of $\mathcal{O}\left(\left(\frac{n}{2}\right)^3\right) = \mathcal{O}(n^3/8)$ operations.
 - b) Solve approximately with Jacobi/SOR/Gauss-Siedell/etc. with a cost of $\mathcal{O}(n \cdot n_{\text{iter}})$ where the iterative method is applied n_{iter} times.
 - c) TBD

Suppose we did Gaussian elimination at the coarse grid solve. Then the cost is

$$\mathcal{O}\left(\underbrace{\nu_1 + \nu_2}_{\text{pre-smoothing}} + \underbrace{1 + 1 + 1}_{\text{post-smoothing}} + \underbrace{1}_{\text{Compute } \vec{r}^h} + \underbrace{1}_{\mathcal{I}_h^{2h}} + \underbrace{1}_{\mathcal{I}_{2h}^h} + 1)n + \frac{n^3}{8}\right) \text{ operations}$$

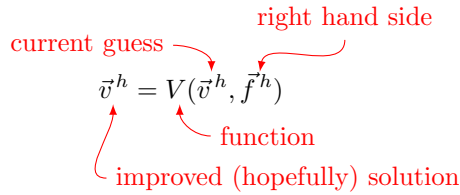
However, doing Gaussian eliminations to $A^h \vec{u} = \vec{f}^h$ requires $\mathcal{O}(n^3)$ operations. Therefore, if not too many two-grid cycles are required, our two-grid algorithm can out perform Gaussian elimination.

2.5 Recursive Multigrid

Regarding option c), we note that we are solving

$$A^{2h} \vec{e}^{2h} = \vec{r}^{2h}$$

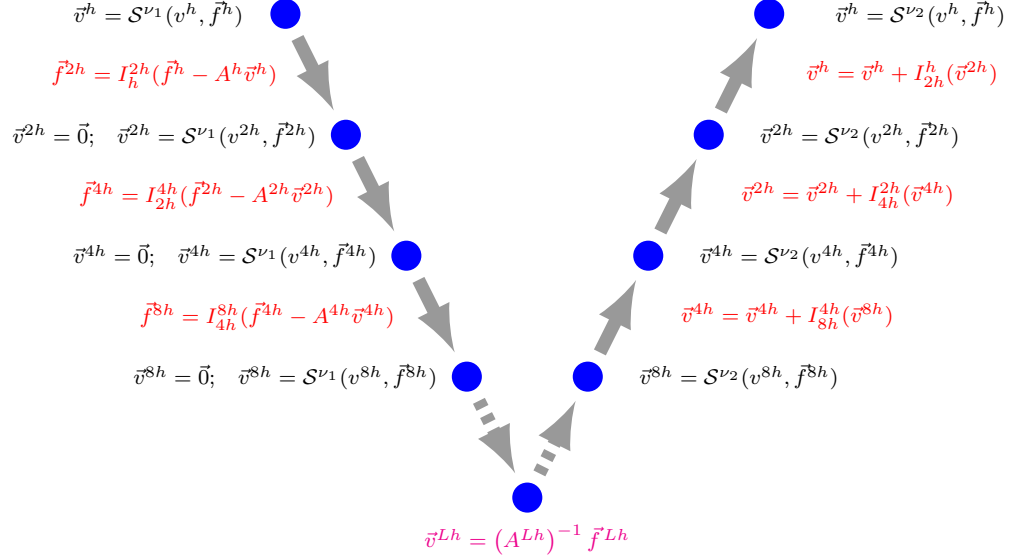
This is just another linear system, albeit a smaller one, that can be solved with a two-grid cycle. That is we have the following algorithm that takes in an initial guess \vec{v}^h and a right hand side \vec{f}^h and returns a better solution of $A^h \vec{u} = \vec{f}^h$.



Algorithm 1 V-cycle

$$\begin{aligned}
 \vec{v}^h &= \text{Smoother}(\vec{v}^h, \vec{f}^h) & k = 1, \dots, \nu_1 \\
 \vec{f}^{2h} &= \mathcal{I}_h^{2h}(\vec{f}^h - A^h \vec{v}^h) \\
 \vec{v}^{2h} &= \vec{0} \\
 \vec{v}^{2h} &= \text{Smoother}(\vec{v}^{2h}, \vec{f}^{2h}) & k = 1, \dots, \nu_1 \\
 \vec{f}^{4h} &= \mathcal{I}_{2h}^{4h}(\vec{f}^{2h} - A^{2h} \vec{v}^{2h}) \\
 \vec{v}^{4h} &= \vec{0} \\
 \vec{v}^{4h} &= \text{Smoother}(\vec{v}^{4h}, \vec{f}^{4h}) & k = 1, \dots, \nu_1 \\
 \vec{f}^{8h} &= \mathcal{I}_{4h}^{8h}(\vec{f}^{4h} - A^{4h} \vec{v}^{4h}) \\
 &\vdots \\
 \vec{v}^{Lh} &= (A^{Lh})^{-1} \vec{f}^{Lh} \\
 &\vdots \\
 \vec{v}^{4h} &= \vec{v}^{4h} + \mathcal{I}_{8h}^{4h} \vec{v}^{8h} \\
 \vec{v}^{4h} &= \text{Smoother}(\vec{v}^{4h}, \vec{f}^{4h}) & k = 1, \dots, \nu_2 \\
 \vec{v}^{2h} &= \vec{v}^{2h} + \mathcal{I}_{4h}^{2h} \vec{v}^{4h} \\
 \vec{v}^{2h} &= \text{Smoother}(\vec{v}^{2h}, \vec{f}^{2h}) & k = 1, \dots, \nu_2 \\
 \vec{v}^h &= \vec{v}^h + \mathcal{I}_{2h}^h \vec{v}^{2h} \\
 \vec{v}^h &= \text{Smoother}(\vec{v}^h, \vec{f}^h) & k = 1, \dots, \nu_2
 \end{aligned}$$

Again, this algorithm is not a solver. It returns a more accurate solution of $A^h \vec{u} = \vec{f}^h$. We can visualize the algorithm as follows.



Because of the shape of this picture we call the algorithm $\vec{v}^h \leftarrow V(\vec{v}^h, \vec{f}^h)$ a *V-cycle*

Notice how at each level, we are simply applying another V-cycle, just with fewer levels. Thus the algorithm can be written very compactly using recursion:

Algorithm 2 V-cycle $\vec{v}^h = V(\vec{v}^h, \vec{f}^h)$

```

 $\vec{v}^h = \text{Smoother}(\vec{v}^h, \vec{f}^h) \quad k = 1, \dots, \nu_1$ 
 $\vec{f}^{2h} = \mathcal{I}_h^{2h}(\vec{f}^h - A^h \vec{v}^h)$ 
if  $\Omega^{2h}$  is the coarsest grid then
  Solve  $A^{2h} \vec{v}^{2h} = \vec{f}^{2h}$ 
else
   $\vec{v}^{2h} = \vec{0}$ 
   $\vec{v}^{2h} = V(\vec{v}^{2h}, \vec{f}^{2h})$ 
end if
 $\vec{v}^h = \vec{v}^h + \mathcal{I}_{2h}^h \vec{v}^{2h}$ 
 $\vec{v}^h = \text{Smoother}(\vec{v}^h, \vec{f}^h) \quad k = 1, \dots, \nu_2$ 

```

When applying a V-cycle with ν_1 pre-smoothing and ν_2 post-smoothing steps, we call it a $V(\nu_1, \nu_2)$ -cycle.

Let's estimate the number of flops required to apply the $V(\nu_1, \nu_2)$ -cycle. The four main costs come from:

- 1) Apply the smoother at all levels

- 2) Compute the residual
- 3) Prolongation and restriction
- 4) Course-grid solve

We ignore updating the solution

$$\text{ie. } \vec{v}^h = \vec{v}^h + \mathcal{I}_{2h}^h \vec{v}^{2h}$$

we also ignore prolongation and restriction. Suppose there are L levels. That is, we apply the smoother on the grids

$$\Omega^h, \Omega^{2h}, \Omega^{4h}, \dots, \Omega^{2^{L-1}h}$$

On grid Ω^h , the smoother requires $\mathcal{O}(n)$ operations.

On grid Ω^{2h} , the smoother requires $\mathcal{O}\left(\frac{n}{2}\right)$ operations.

\vdots

On grid $\Omega^{2^{L-1}h}$, the smoother requires $\mathcal{O}\left(\frac{n}{2^{L-1}}\right)$ operations.

Therefore, one smoother per level requires

$$\begin{aligned} \mathcal{O}\left(n + \frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{2^{L-1}}\right) &= \mathcal{O}\left(n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^{L-1}}\right)\right) \\ &= \mathcal{O}\left(n\left(\frac{1 - \left(\frac{1}{2}\right)^L}{1 - \frac{1}{2}}\right)\right) \\ &= \mathcal{O}\left(2n\left(1 - \frac{1}{2^L}\right)\right) \end{aligned}$$

\Rightarrow The cost of pre-smoothing and post-smoothing is

$$\mathcal{O}\left(2(\nu_1 + \nu_2)\left(1 - \frac{1}{2^L}\right)n\right) \quad \text{operations}$$

The cost of computing all of the residuals is

$$\mathcal{O}\left(2\left(1 - \frac{1}{2^L}\right)n\right) \quad \text{operations}$$

Solving the coarse-grid equation requires no more than

$$\mathcal{O}\left(\left(\frac{n}{2^{L-1}}\right)^3\right) \text{ operations}$$

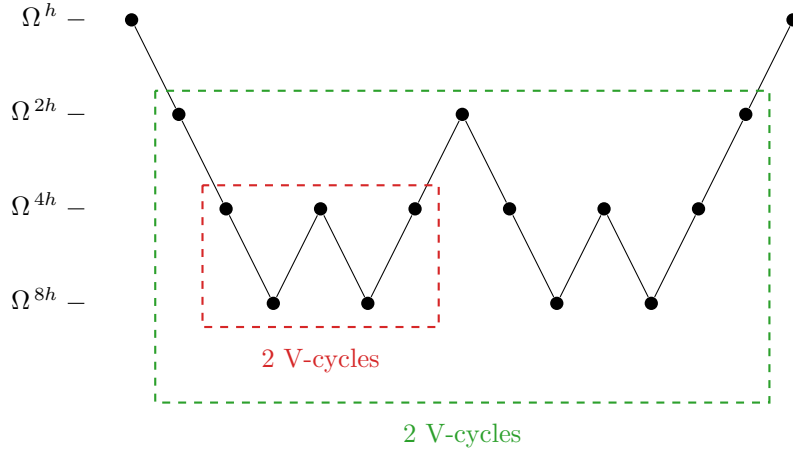
We would like to choose a value for L so that $\frac{n}{2^{L-1}}$ is order 1.

$$\begin{aligned} \Rightarrow \frac{n}{2^{L-1}} &= \mathcal{O}(1) \\ \Rightarrow n &= \mathcal{O}(2^{L-1}) \\ \Rightarrow \log_2 n &= \mathcal{O}(L-1) \\ \Rightarrow L &= \mathcal{O}(\log_2 n) \end{aligned}$$

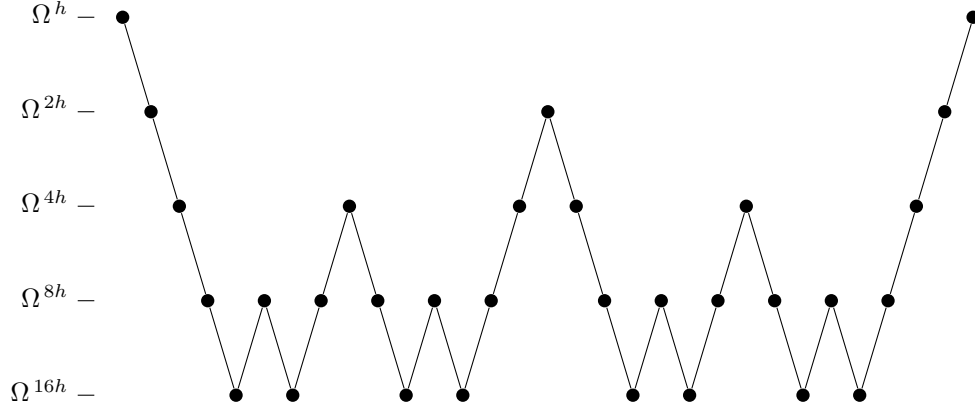
With this choice for L , the smoothing and residual operations cost $\mathcal{O}(n)$ flops and the coarse-grid solve requires $\mathcal{O}(1)$ operations.

2.6 Multigrid Cycles

The V-cycle recursively performs one V-cycle per level. Alternatively, we could apply $\mu \in \mathbb{N}$ V-cycles per level. If $\mu = 2$, the result is a W-cycle. The schedule of the grids are as follows.



If we add another level the W-cycle would look like



Once you have a good recursive V-cycle code, Implementing a W-cycle is trivial.

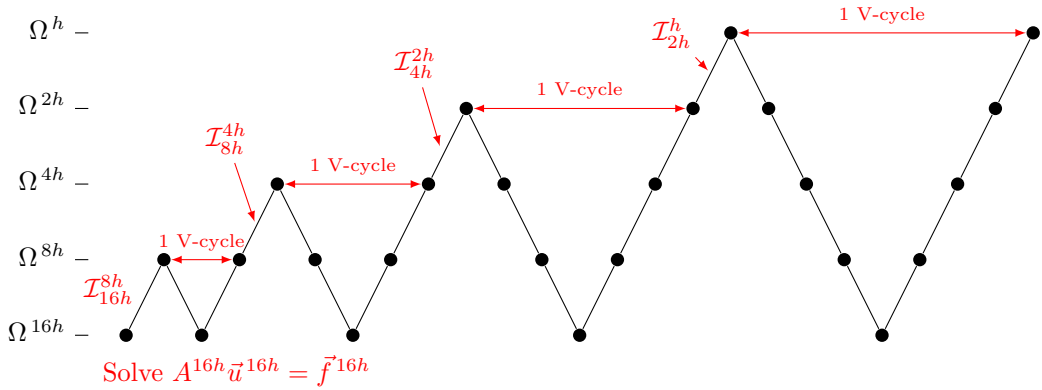
$$\begin{array}{ccc}
 \text{V-Cycle} & & \text{W-Cycle} \\
 \vec{v}^{2h} = 0 & \text{Changes} & \vec{v}^{2h} = 0 \\
 \vec{v}^{2h} = V(\vec{v}^{2h}, \vec{f}^{2h}) & \text{to} & \vec{v}^{2h} = V(\vec{v}^{2h}, \vec{f}^{2h}) \\
 & & \vec{v}^{2h} = V(\vec{v}^{2h}, \vec{f}^{2h})
 \end{array}$$

The W-cycle costs more per iterations, but it does a better job of eliminating intermediate frequencies. This is beneficial in \mathbb{R}^2 and \mathbb{R}^3 , but in \mathbb{R}^1 , more times than not, you can not out perform a V(1,1)-cycle.

The final multigrid cycle we'll consider is called *full multigrid* or FMG. It is motivated by the fact that we should choose a good initial guess. We can choose an initial guess at the coarsest grid by solving.

$$\vec{u}^{Lh} = (A^{Lh})\vec{f}^{Lh}$$

However, we need to move this initial guess to the grid Ω^h . We could just prolong the whole way up but this would not be very, accurate. Instead we prolong once, perform a V-cycle, and repeat until we're at the finest grid. We then perform our standard V-cycle.



Some generalizations are

- 1) Do W-cycles instead of V-cycles.
- 2) Do more than one V-cycle or W-cycle starting at the finest grid.
- 3) Do more than 1 V-cycle or W-cycle when forming the initial guess.

2.7 Multigrid in \mathbb{R}^2

Multigrid is not very practical for solving

$$\begin{aligned} u'' &= f & x \in (0, 1) \\ u(0) &= u(1) = 0 \end{aligned}$$

with second-order centered differences. Instead, we should solve this system with the Thomas algorithm in $\mathcal{O}(n)$ operation. However, in \mathbb{R}^2 or higher, we are solving

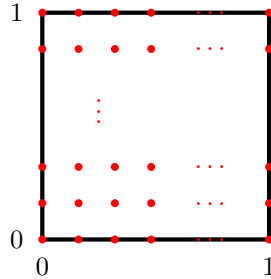
$$\begin{aligned} \Delta u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) & (x, y) \in (0, 1) \\ u &= 0 & (x, y) \in \partial(0, 1) \end{aligned}$$

After discretizing this problem with centered difference, multigrid becomes much more powerful.

The linear system to solve

$$f_{i,j} = \frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2} \quad \begin{matrix} i=1,\dots,n \\ j=1,\dots,n \end{matrix}$$

Where $\Delta x = \Delta y = h = \frac{1}{n+1}$ and $U_{i,j} \approx u(x_i, y_j)$ with $x_i = ih$, $y_j = jh$



Note that we have removed the boundary points from the linear system. We use a lexicographic ordering of $U_{i,j}$

$$\frac{1}{h^2} \begin{bmatrix} \begin{array}{ccc|ccc} -4 & 1 & & 0 & 1 & \\ 1 & -4 & 1 & & 1 & 0 \\ 0 & & \ddots & 1 & & \\ \hline 1 & & & 0 & -4 & 1 \\ & 1 & & 0 & 1 & \\ 0 & & \ddots & 1 & & \\ \hline 0 & & & 0 & 1 & \\ & & 1 & & -4 & 1 \\ & & & 1 & & 0 \\ \hline & & & & 1 & \\ & & & & & 0 \\ & & & & & 1 \end{array} & \begin{bmatrix} U_{11} \\ U_{21} \\ \vdots \\ U_{n1} \\ U_{12} \\ U_{22} \\ \vdots \\ U_{n2} \\ U_{1n} \\ U_{2n} \\ \vdots \\ U_{nn} \end{bmatrix} = \begin{bmatrix} f_{11} \\ f_{21} \\ \vdots \\ f_{n1} \\ f_{12} \\ f_{22} \\ \vdots \\ f_{n2} \\ f_{1n} \\ f_{2n} \\ \vdots \\ f_{nn} \end{bmatrix} \end{bmatrix}$$

If we define

$$T = \begin{bmatrix} -4 & 1 & & 0 \\ 1 & -4 & 1 & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -4 \end{bmatrix} \in \mathbb{R}^{n \times n} \quad I = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} \in \mathbb{R}^{n \times n}$$

then

$$A = \frac{1}{h^2} \begin{bmatrix} T & I & & \\ I & T & I & \\ & I & T & I \\ & & \ddots & \ddots & 0 \\ 0 & & & I & T & I \\ & & & I & T \end{bmatrix}$$

We are trying to solve

$$\frac{1}{h^2} (U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}) = f_{i,j}$$

The Jacobi iteration is

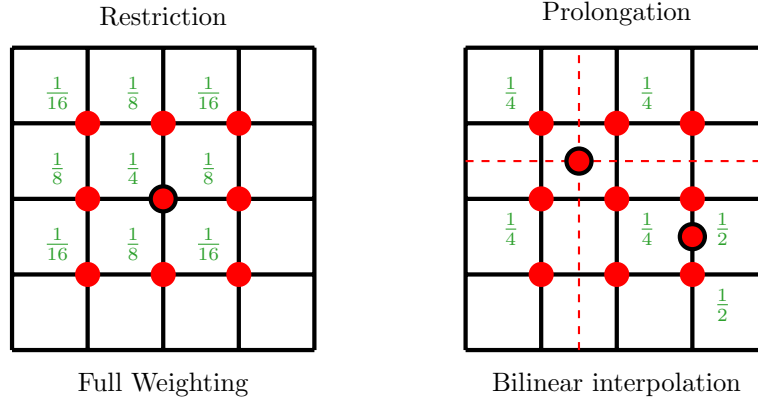
$$U_{i,j}^{n+1} = \frac{1}{4} (U_{i+1,j}^n + U_{i-1,j}^n + U_{i,j+1}^n + U_{i,j-1}^n - h^2 f_{i,j})$$

The weighted Jacobi is

$$U_{i,j}^{n+1} = \frac{\omega}{4} (U_{i+1,j}^n + U_{i-1,j}^n + U_{i,j+1}^n + U_{i,j-1}^n - h^2 f_{i,j}) + (1 - \omega)U_{i,j}^n$$

Using a similar argument as we did in \mathbb{R}^1 , the optimal choice for ω is $\omega = 4/5$.

In addition to a smoother, we require a prolongation and restriction operators.



3 Randomized SVD

Many matrices that appear in scientific contexts have a low rank structure which can be exploited. In recent years, the field of data science has also found low rank approximations useful for their analysis. See the paper by Udell and Townsend, “Why Are Big Data Matrices Approximately Low Rank?” for more on the topic. The large size of data matrices can prohibit the direct calculation low rank structure, but randomized methods can generate strong approximations. For a detailed analysis see the paper by Martinsson, Rokhlin, and Tygert, “A randomized algorithm for the decomposition of matrices” . Also Martinsson’s website contains many slides and video lectures on the topic.

3.1 The Singular Value Decomposition (SVD)

The SVD is arguably the most important matrix decomposition in linear algebra. Unlike the eigendecomposition, LU , and others, the SVD always exists and the matrix need not be square. A SVD of $A \in \mathbb{R}^{m \times n}$ is

$$A = U\Sigma V^T$$

Where $U \in \mathbb{R}^{(m \times m)}$ and $V \in \mathbb{R}^{(n \times n)}$ are orthogonal and $\Sigma \in \mathbb{R}^{(m \times n)}$ is a diagonal matrix with only non-negative entries.

Definition

A matrix $U \in \mathbb{R}^{(m \times m)}$ is orthogonal if

- 1) The columns/rows are orthogonal
- 2) $U^T U = U U^T = I$
- 3) $U^{-1} = U^T$

The diagonal entries of Σ are denoted by $\sigma_1, \sigma_2, \dots, \sigma_p$ where $p = \min(m, n)$. We always order the columns of U , rows of V , and diagonal of Σ so that $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \dots \geq \sigma_p \geq 0$

Having the SVD of a matrix is very useful. With an SVD, we can form the pseudo inverse of A

$$A^\dagger = V\Sigma^\dagger U^T$$

where Σ^\dagger is the transpose of Σ and invert the non-zero entries

Ex

$$\Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \Sigma^\dagger = \begin{bmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

With the SVD, one can also find the best rank r matrix that approximates A with $r \leq \min(m, n)$

$$\tilde{A} = U\tilde{\Sigma}V^T$$

where $\tilde{\Sigma}$ is the matrix Σ with only the first r singular values

$$\Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad r = 2 \Rightarrow \tilde{\Sigma} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This matrix \tilde{A} is the minimizer of

$$\{ \|A - B\|_2 \mid \text{rank}(B) = r \}$$

We can write our SVD using outer products as follows. Let $p = \min(m, n)$

$$A = U \Sigma V^T$$

$$A = \begin{bmatrix} \begin{array}{c} \uparrow \\ \vec{u}_1 \\ \downarrow \end{array} & \begin{array}{c} \uparrow \\ \vec{u}_2 \\ \downarrow \end{array} & \dots & \begin{array}{c} \uparrow \\ \vec{u}_m \\ \downarrow \end{array} \end{bmatrix} \begin{bmatrix} \sigma_1 & & & 0 \\ & \sigma_2 & & \\ 0 & & \ddots & \\ & & & \sigma_p \end{bmatrix} \begin{bmatrix} \begin{array}{c} \leftarrow \vec{v}_1 \rightarrow \end{array} \\ \begin{array}{c} \leftarrow \vec{v}_2 \rightarrow \end{array} \\ \vdots \\ \begin{array}{c} \leftarrow \vec{v}_n \rightarrow \end{array} \end{bmatrix}$$

$$A = \sum_{i=1}^p \sigma_i \vec{u}_i \vec{v}_i^T$$

Unfortunately, computing the SVD directly requires $\mathcal{O}(mn^2)$ operations if $m \geq n$. For many practical problems, this is too expensive. Fortunately, many matrices in statistics, data sciences, PDEs, and more are *numerically low rank*.

3.2 Low Rank Approximations

Given a tolerance ϵ , a matrix A has numerical rank k if there exists a rank k matrix with

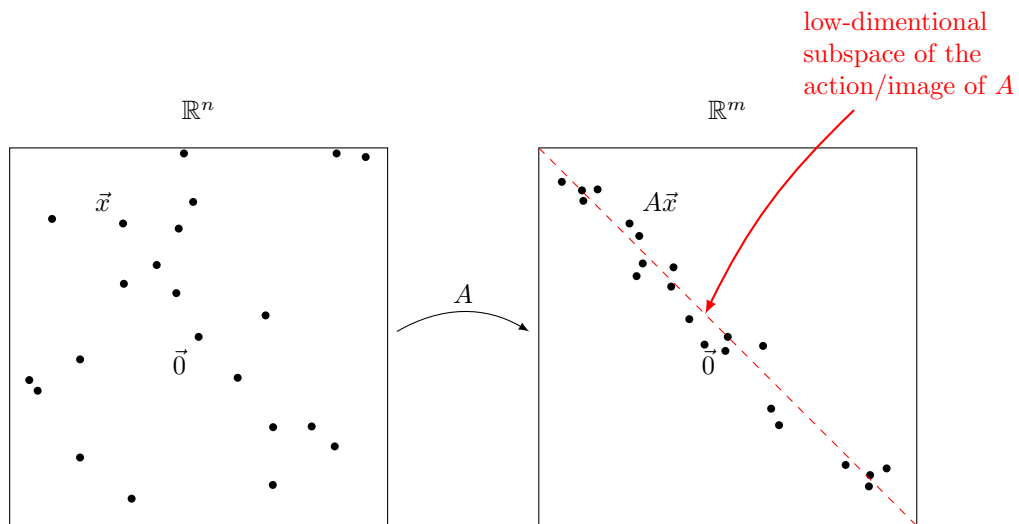
$$\|A - \tilde{A}\| \leq \epsilon$$

A is numerically low rank if k is sufficiently small.

A very nice property to have is if the numerical rank of a matrix A does not grow with the size of A , where the size of A depends on how the problem is discretized.

To compute a low-rank approximation of $A \in \mathbb{R}^{m \times n}$, two computational tasks must be performed.

- (A) A low-dimensional subspace that captures the action of matrix A must be formed.



(B) The matrix A must be restricted to this subspace, and then a standard decomposition, such as an SVD is computed for the reduced matrix.

$$\begin{array}{c}
 \begin{array}{ccc}
 & n & \mathbb{R}^n \quad \mathbb{R}^m \\
 A = & m & \left[\begin{array}{c} \\ \\ \\ \end{array} \right] \left[\begin{array}{c} \\ \\ \\ \end{array} \right] = \left[\begin{array}{c} \\ \\ \\ \end{array} \right]
 \end{array} \\
 \\
 \begin{array}{ccc}
 & n & \mathbb{R}^n \quad \mathbb{R}^k \\
 B = & k & \left[\begin{array}{c} \\ \\ \\ \end{array} \right] \left[\begin{array}{c} \\ \\ \\ \end{array} \right] = \left[\begin{array}{c} \\ \\ \\ \end{array} \right]
 \end{array}
 \end{array}$$

More formally, task (A) is equivalent to the following:

Compute an approximate basis for the range of A . This is equivalent to finding a matrix $Q \in \mathbb{R}^{m \times k}$ with orthonormal columns such that

$$A \approx QQ^T A$$

This matrix $QQ^T A$ is what we called \tilde{A} . We would also like Q to have as few columns as possible. Ficen such a Q , we have

$$QQ^T A = \tilde{A}$$

Also,

$$A\vec{x} = Q(Q^T A\vec{x}) \in \text{Range}(Q)$$

$$\Rightarrow \text{Range}(A) \approx \text{Range}(Q)$$

Ex

$$A = \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & -2 \\ 2 & 0 & 1 \end{bmatrix}$$

The rank of A is 2 and

$$\text{Range}(A) = \text{span} \left\{ \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\}$$

Let's make the following choice for Q

$$Q = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{6} \\ 1/\sqrt{3} & -2/\sqrt{6} \\ 1/\sqrt{3} & 1/\sqrt{6} \end{bmatrix}$$

$$\begin{aligned} QQ^T &= \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{6} \\ 1/\sqrt{3} & -2/\sqrt{6} \\ 1/\sqrt{3} & 1/\sqrt{6} \end{bmatrix} \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{6} & 1/\sqrt{3} \\ 1/\sqrt{6} & -2/\sqrt{6} & 1/\sqrt{6} \end{bmatrix} \\ &= \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \end{bmatrix} \end{aligned}$$

$$\begin{aligned}\Rightarrow QQ^T A &= \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & -2 \\ 2 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & -2 \\ 2 & 0 & 1 \end{bmatrix} = A\end{aligned}$$

That is, $A = QQ^T A$.

Given this matrix Q , we can perform task (B). For example, suppose we find a matrix $Q \in \mathbb{R}^{m \times k}$ with $A \approx QQ^T A$. Then we can do the following 3 steps

- 1) Let $B = Q^T A$
- 2) Compute the SVD of B which is

$$B = \tilde{U} \Sigma V^T$$

- 3) Let $U = Q \tilde{U}$

Then,

$$\begin{aligned}U \Sigma V^T &= Q \tilde{U} \Sigma V^T \\ &= QB \\ &= QQ^T A \approx A\end{aligned}$$

That is, $U \Sigma V^T$ is an approximate (truncated) SVD of A . The big question is

How do we find $Q \in \mathbb{R}^{m \times k}$?

3.3 Randomized Algorithms

In this last example, I chose the optimal 3×2 matrix Q so that $\|A - QQ^T A\|$ is minimized. In fact, it was 0. In particular,

$$A = \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & -2 \\ 2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1/\sqrt{6} & -1/\sqrt{3} & -1/\sqrt{2} \\ 2/\sqrt{6} & -1/\sqrt{3} & 0 \\ -1/\sqrt{6} & -1/\sqrt{3} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 & 0 \\ 0 & \sqrt{6} & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1/\sqrt{3} & -1/\sqrt{2} & 1/\sqrt{6} \\ 1/\sqrt{3} & -1/\sqrt{2} & -1/\sqrt{6} \\ -1/\sqrt{3} & 0 & -2/\sqrt{6} \end{bmatrix}$$

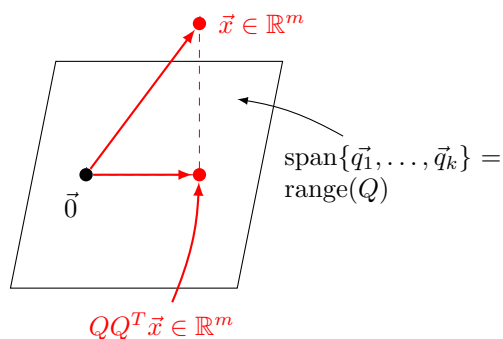
Letting Q be the left singular vectors of A with a non-zero singular value, we have

$$Q = \begin{bmatrix} 1/\sqrt{6} & 1/\sqrt{3} \\ -2/\sqrt{6} & 1/\sqrt{3} \\ 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix}$$

Last week, we looked for a matrix with orthonormal columns such that

$$A \approx QQ^T A \quad \text{with } A \in \mathbb{R}^{m \times n}, Q \in \mathbb{R}^{m \times k}$$

The matrix QQ^T is actually the matrix that does an orthogonal projection of any vector onto the space spanned by Q 's columns



Since we are computing $QQ^T A$, this is the orthogonal projection of the range of A onto the space spanned by Q 's columns. Finally to see that QQ^T is an orthogonal projection, recall that given any set of linearly independent vectors $\vec{q}_1, \dots, \vec{q}_k \in \mathbb{R}^m$, but are not necessarily orthonormal, the matrix that performs an orthogonal projection onto, the space spanned by $\vec{q}_1, \dots, \vec{q}_k$ is

$$P_Q = Q (Q^T Q)^{-1} Q^T$$

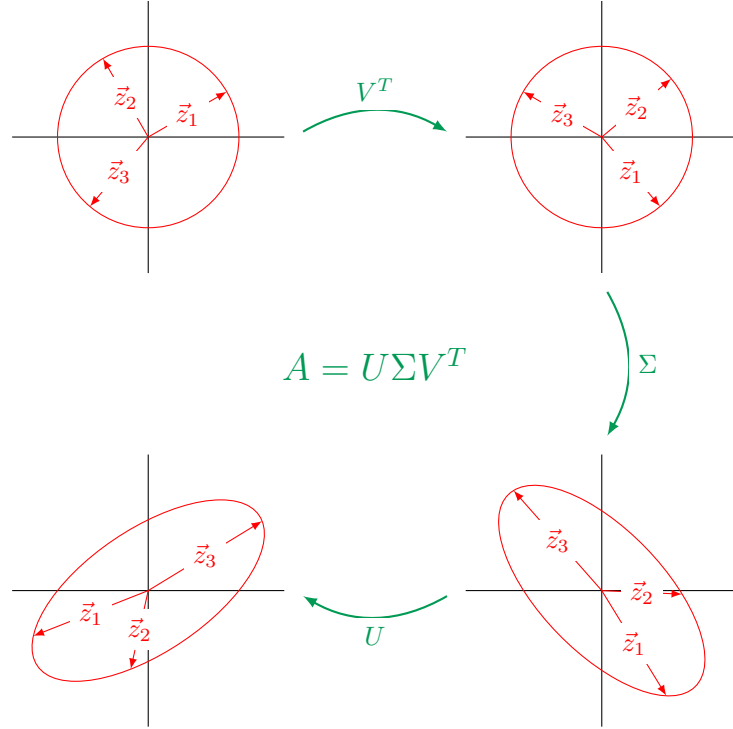
If Q has orthogonal columns, then

$$Q^T Q = \begin{bmatrix} \leftarrow \vec{q}_1^T \rightarrow \\ \leftarrow \vec{q}_2^T \rightarrow \\ \vdots \\ \leftarrow \vec{q}_k^T \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow \vec{q}_1 \downarrow \\ \uparrow \vec{q}_2 \downarrow \\ \vdots \\ \uparrow \vec{q}_k \downarrow \end{bmatrix}$$

$$Q^T Q = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = I \in \mathbb{R}^{k \times k}$$

$$\Rightarrow P_Q = QQ^T$$

Aside about visualizing the SVD:



From previous example:

$$A = \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & -2 \\ 2 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 1/\sqrt{6} & 1/\sqrt{3} \\ -2/\sqrt{6} & 1/\sqrt{3} \\ 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix} = \begin{bmatrix} \uparrow & \uparrow \\ \vec{u}_1 & \vec{u}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

That is, Q 's columns are the left singular vectors of A whose singular values are non-zero. Then

$$QQ^T A = \begin{bmatrix} \uparrow & \uparrow \\ \vec{u}_1 & \vec{u}_2 \\ \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} \leftarrow \vec{u}_1^T \rightarrow \\ \leftarrow \vec{u}_2^T \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ \vec{u}_1 & \vec{u}_2 & \vec{u}_3 \\ \downarrow & \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & 0 \end{bmatrix} \begin{bmatrix} \leftarrow \vec{v}_1^T \rightarrow \\ \leftarrow \vec{v}_2^T \rightarrow \\ \leftarrow \vec{v}_3^T \rightarrow \end{bmatrix}$$

$$\begin{aligned}
QQ^T A &= \begin{bmatrix} \uparrow & \uparrow \\ \vec{u}_1 & \vec{u}_2 \\ \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \Sigma V^T \\
\\
QQ^T A &= \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ \vec{u}_1 & \vec{u}_2 & \vec{0} \\ \downarrow & \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & 0 \end{bmatrix} \begin{bmatrix} \leftarrow \vec{v}_1^T \rightarrow \\ \leftarrow \vec{v}_2^T \rightarrow \\ \leftarrow \vec{v}_3^T \rightarrow \end{bmatrix} \\
\\
QQ^T A &= \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T = A
\end{aligned}$$

Of course, we can't use the SVD to construct Q since the goal of computing Q is to be able to compute the SVD and other matrix decompositions more efficiently. Instead we use randomized algorithms. Again, our problem is the following:

Given a matrix $A \in \mathbb{R}^{m \times n}$ and a tolerance ϵ , find a matrix $Q \in \mathbb{R}^{m \times k}$ with $k = k(\epsilon)$ and the columns of Q being orthonormal such that

$$\|A - QQ^T A\| \leq \epsilon$$

where $\|\cdot\|$ is the l^2 -matrix norm

If we find such a Q , the range of Q is approximately the k -dimensional (upto a tolerance ϵ) range of A .

The SVD provides the best such Q . Letting Q be the matrix with the first k left singular vectors of A , we have $\text{rank}(Q) = k$ and

$$\|A - QQ^T A\| = \sigma_{k+1}$$

Therefore, given a tolerance ϵ , if we could use the SVD to construct Q , we would choose k such that $\sigma_{k+1} \leq \epsilon$ and $\sigma_k > \epsilon$.

In order to calculate Q more efficiently, we start by introducing an oversampling parameter p . Assuming A has rank k , our goal is now to find Q with $k + p$ orthonormal columns such that

$$\|A - QQ^T A\| \approx \min_{\text{rank}(X)=k} \|A - X\|$$

As before, if $p = 0$, the minimizer comes from the left singular vectors of A . However, by letting $p > 0$, we have flexibility which will lead to more efficient methods.

Suppose $A \in \mathbb{R}^{m \times n}$ has rank k . We can draw a random vector $\vec{\omega} \in \mathbb{R}^n$ and compute $\vec{y} = A\vec{\omega} \in \text{range}(A)$.

We can repeat this k times.

$$\vec{y}^{(i)} = A\vec{\omega}^{(i)} \quad i = 1, 2, \dots, k$$

Because we choose $\vec{\omega}^{(i)}$ randomly, the $\vec{y}^{(i)}$ are almost guaranteed to be linearly independent, and thus span $\text{range}(A)$. Then, to produce an orthonormal basis, we apply an algorithm such as Gram–Schmidt to generate $\vec{q}_1, \vec{q}_2, \dots, \vec{q}_k$, and then

$$Q = \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ \vec{q}_1 & \vec{q}_2 & \cdots & \vec{q}_k \\ \downarrow & \downarrow & \downarrow \end{bmatrix}$$

is the matrix we need to construct. However, A is in general not rank k , but has numerical rank k . That is

$$A = B + E$$

where $\text{rank}(B) = k$ and $\|E\| \leq \epsilon$. We wish to form a basis for the range of B . Since we are computing $A\vec{\omega}$, this vector is not in the range of B . Therefore, we choose more than k vectors. In particular, we form the vectors

$$\vec{y}^{(i)} = A\vec{\omega}^{(i)} = B\vec{\omega}^{(i)} + E\vec{\omega}^{(i)} \quad i = 1, 2, \dots, k + p.$$

The perturbation $E\vec{\omega}^{(i)}$ shifts $\vec{y}^{(i)}$ from the range of B . Therefore, it is possible that $\vec{y}^{(1)}, \dots, \vec{y}^{(k)}$ does not span the range of B . However the enriched space

$$\text{span} \left\{ \vec{y}^{(i)} \mid i = 1, \dots, k + p \right\}$$

has a much better chance of spanning the range of B . Once we have chosen p , we have the beginnings of a basic algorithm to construct $Q \in \mathbb{R}^{m \times k}$ with

$$A \approx QQ^T A$$

Given a matrix $A \in \mathbb{R}^{m \times n}$, a target rank k , and an oversampling parameter p , the *fixed-rank problem* computes a $m \times (k + p)$ matrix Q whose columns are orthonormal and whose range approximates the range of A . The basic algorithm is as follows

- 1) Draw a random $n \times (k + p)$ test matrix Ω .
- 2) Compute $Y = A\Omega \in \mathbb{R}^{m \times (k+p)}$.

- 3) Construct a matrix Q whose columns form an orthonormal basis for the range of Y .

Note that we have not assumed that Y 's columns are linearly independent. Several issues that must be addressed are the following:

- How large should we take p ?
- What random matrix Ω should we pick?
- What if k is not known in advanced?
- If Y is ill-conditioned (which is likely), how do we stably orthonormalize its columns to generate Q ?
- What computational cost is incurred?

Theorem

Let $A \in \mathbb{R}^{m \times n}$ and $k \geq 2$ be a target rank and $p \geq 2$ is an oversampling parameter such that $k + p \leq \min\{m, n\}$

Then, if $\Omega \in \mathbb{R}^{n \times (k+p)}$ is a Gaussian test matrix used to generate $Y = A\Omega \in \mathbb{R}^{m \times (k+p)}$, and let Q be the matrix with orthonormal vectors that span the range of Y . Then

$$\mathbb{E} \|A - QQ^T A\| \leq \left(1 + \frac{4\sqrt{k+p}}{p-1} \sqrt{\min(m, n)}\right) \sigma_{k+1}$$

where \mathbb{E} is the expected value and σ_{k+1} is the $(k+1)^{\text{st}}$ singular value of A .

Definition

A Gaussian matrix is a matrix whose entries are all independently and identically distributed (iid) with distribution $\mathcal{N}(0, 1)$ (normal distribution with mean 0 and variance 1).

In matlab you would do the following command to generate a Gaussian matrix $\Omega = \text{randn}(n, k+p)$

A valid question to ask is what the variance of this distribution is (ie. $\|A - QQ^T A\|$). If it is large, then the estimate might not be very useful. However, one can show that if Ω is a Gaussian matrix in $\mathbb{R}^{n \times (k+p)}$, then the probability that

$$\|AQQ^T A\| \leq \left(1 + 9\sqrt{k+p} \sqrt{\min(m, n)}\right) \sigma_{k+1}$$

is at least $1 - 3p^{-p}$. Therefore, if $p = 5$, we have about an 0.1% chance that $\|AQQ^T A\|$ is larger than this bound. Therefore, we have addressed the issue about how to choose p .

3.4 Interpolative Decomposition

If we choose the columns of Ω to be one of the standard basis functions in \mathbb{R}^n

$$\vec{e}_i = (0, \dots, 0, 1, 0, \dots, 0)$$

then Ω will contain columns of A . By selecting columns, we have the following

Theorem

Let $A \in \mathbb{R}^{m \times n}$ and $k \leq \min(m, n)$. Then, there exists a k -column submatrix $C \in \mathbb{R}^{k \times n}$ of A for which

$$\|A - CC^\dagger A\| \leq \sqrt{1 + k(n - k)} \|A - A_{(k)}\|$$

where $A_{(k)}$ is the best rank k approximation of A (ie. it comes from the SVD) and \dagger is the dagger operation for the pseudoinverse

$$C^\dagger = (C^T C)^{-1} C^T$$

Finding the optimal C is an NP-hard problem. A more practical approach is to sample A 's columns randomly. But, we use the following heuristic. The columns of A with the largest norm contribute “the most” to the range of A . Therefore, we can create a distribution formed with the norms of A 's columns, and sample from this distribution.

This result is related to the *Interpolative Decomposition* (ID). Given a matrix $A \in \mathbb{R}^{m \times n}$ of rank k , then there exists k columns of A that span all of the columns of A . That is, there exists an index set

$$J = \{j_1, \dots, j_k\}$$

of positive integers less than m with no repetition with

$$A = A_{(:,J)} X$$

with $X_{(:,J)} = I_k$. So far, this result is obvious since all columns of A can be written as a linear combination of any k columns of A that span the range of A . What the ID guarantees that the entries of X are bounded in absolute value by 2.

Let's form all possible IDs of the matrix

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 3 & -2 & -1 \\ 0 & 1 & 2 \end{bmatrix}$$

which has rank 2. We only have 3 choices for J .

Case 1: $J = \{1, 2\}$

$$\begin{bmatrix} 2 \\ -1 \\ 2 \end{bmatrix} = 1 \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \Rightarrow A = \begin{bmatrix} 0 & 1 \\ 3 & -2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Since $|X_{ij}| \leq 2$, this is an ID.

Case 2: $J = \{2, 3\}$

$$\begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} = -2 \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ -1 \\ 2 \end{bmatrix} \Rightarrow A = \begin{bmatrix} 1 & 2 \\ -2 & -1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

This is an ID.

Case 3: $J = \{1, 3\}$

$$\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} = \frac{-1}{2} \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 2 \\ -1 \\ 2 \end{bmatrix} \Rightarrow A = \begin{bmatrix} 0 & 2 \\ 3 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & -1/2 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}$$

3.5 Gram-Schmidt Process

One of the central tasks we need to do is form an orthonormal basis for the range (column space) of the matrix $Y = A\Omega$, where Ω is some kind of random matrix. The standard way to form these vectors is with *Gram-Schmidt* (G-S). G-S forms orthogonal vectors, $\vec{q}_1, \dots, \vec{q}_{k+p}$, but they can be made orthonormal by normalizing

$$\vec{q}_i := \frac{\vec{q}_i}{\|\vec{q}_i\|} \quad i = 1, \dots, k+p$$

To form the orthogonal vectors, suppose we start with $\vec{y}_1, \dots, \vec{y}_k \in \mathbb{R}^m$. We then define $\vec{q}_1, \dots, \vec{q}_k \in \mathbb{R}^m$ as follows:

$$\begin{aligned} \vec{q}_1 &= \vec{y}_1 \\ \vec{q}_2 &= \vec{y}_2 - \frac{\vec{q}_1^T \vec{y}_2}{\vec{q}_1^T \vec{q}_1} \vec{q}_1 \\ \vec{q}_3 &= \vec{y}_3 - \frac{\vec{q}_1^T \vec{y}_3}{\vec{q}_1^T \vec{q}_1} \vec{q}_1 - \frac{\vec{q}_2^T \vec{y}_3}{\vec{q}_2^T \vec{q}_2} \vec{q}_2 \\ &\vdots \\ \vec{q}_i &= \vec{y}_i - \sum_{j=1}^{i-1} \frac{\vec{q}_j^T \vec{y}_i}{\vec{q}_j^T \vec{q}_j} \vec{q}_j \quad i = 1, 2, \dots, k. \end{aligned}$$

We can easily show that each of the $\{\vec{q}_1, \dots, \vec{q}_k\}$ are in the span of $\{\vec{y}_1, \dots, \vec{y}_k\}$

To see this, note that

$$\begin{aligned}\vec{q}_1 &\in \text{span}\{\vec{y}_1\} \\ \vec{q}_2 &\in \text{span}\{\vec{q}_1, \vec{y}_2\} \Rightarrow \vec{q}_2 \in \text{span}\{\vec{y}_1, \vec{y}_2\} \\ &\vdots \\ \vec{q}_i &\in \text{span}\{\vec{y}_1, \vec{y}_2, \dots, \vec{y}_i\}\end{aligned}$$

The proof that $\vec{q}_1, \dots, \vec{q}_k$ are orthogonal can be proved using mathematical induction.

One problem with G-S is that $\vec{q}_i^T \vec{q}_j$, $i \neq j$, is not quite 0 in finite precision because of rounding errors. This loss of orthogonality means QQ^T is no longer an orthogonal projection, and this is a requirement for the approximation

$$A \approx QQ^T A$$

These vectors $\vec{q}_1, \dots, \vec{q}_k$ can be computed in a different manner such that:

- 1) The result is identical to the classical G-S in infinite precision.
- 2) $\vec{q}_i^T \vec{q}_j$ will not potentially be large because of round-off error.

This method is known as modified G-S and it is numerically stable as opposed to classical G-S which is numerically unstable.

One nice feature of G-S is that it works even if $\vec{y}_1, \dots, \vec{y}_k$ are linearly dependent. If this is the case, say \vec{y}_i depends linearly on $\vec{y}_1, \dots, \vec{y}_{i-1}$, then G-S will return $\vec{q}_i = \vec{0}$. We simply remove \vec{y}_i from Y and skip \vec{q}_i , and continue applying G-S.

3.6 The Fixed-Precision Problem

So far, we have considered the problem of finding the best possible rank k matrix whose range approximates the range of A .

This is called the fixed-rank problem which involves finding $Q \in \mathbb{R}^{m \times k}$ with

$$\|A - QQ^T A\|$$

as small as possible. However, we typically don't have a target rank, but instead have a target tolerance. This results in the fixed precision problem:

Given a tolerance $\epsilon > 0$, find a matrix $Q \in \mathbb{R}^{m \times l}$ with orthonormal columns and l as small as possible such that

$$\|QQ^T A - A\| \leq \epsilon$$

The size of l must be at least as large as the ϵ -rank of A , but we want to keep l as close to k as possible. The idea is to generate Q one column at a time, and check $\|QQ^T A - A\| < \epsilon$ at each iteration.

The size l should only be slightly larger than the ϵ -rank of A which is k . We could simply construct columns of Q one at a time, and check if $\|QQ^T A - A\| \leq \epsilon$ at each iterations. However, this is too expensive.

Instead of computing the matrix norm at each iteration, we might expect that applying $QQ^T A - A$ to a random vector can tell us something about its norm. We will use the following result.

Theorem

Let $\{\vec{\omega}^{(i)}, i = 1, \dots, r\}$ be r standard Gaussian vectors. Then ,

$$\|QQ^T A - A\| \leq 10\sqrt{\frac{2}{\pi}} \max_{i=1, \dots, r} \|(QQ^T - I)A\vec{\omega}^{(i)}\|$$

with probability at least $1 - 10^{-r}$

Therefore, if we let $r = 4$, then by computing 4 vector norms, we have a bound on $\|QQ^T A - A\|$ with probability $1 - 10^{-4} = 99.99\%$.

This result allows us to incrementally construct Q so that $QQ^T A \approx A$. We start with $Q^{(0)}$ being the empty matrix

```

for  $i = 1, 2, \dots$  do
  Let  $\vec{\omega}^{(i)} \in \mathbb{R}^n$  be a Gaussian random vector.
  Let  $\vec{y}^{(i)} = A\vec{\omega}^{(i)}$       %  $\vec{y}^{(i)}$  is in the range of  $A$ 
  Compute  $\tilde{\vec{q}}^{(i)} = (I - Q^{(i-1)}Q^{(i-1)T})\vec{y}^{(i)}$ 
      % orthogonal projection of  $\vec{y}^{(i)}$  onto the
      % column space of  $Q^{(i-1)}$ 
  Normalize  $\vec{q}^{(i)} = \tilde{\vec{q}}^{(i)} / \|\tilde{\vec{q}}^{(i)}\|$ 
  Update  $Q^{(i)} = Q^{(i-1)}\vec{q}^{(i)}$ 
end for

```

We break the loop when we believe that $\|Q^{(l)}Q^{(l)T} A - A\| \leq \epsilon$, but how do we choose l ? Using the last Theorem, we can show that $\|(QQ^T - I)A\vec{\omega}^{(i)}\|$ can be used to estimate the matrix norm, but this is exactly $\|\tilde{\vec{q}}^{(i)}\|$. Therefore, we stop when we observe r consecutive vectors $\|\tilde{\vec{q}}^{(i)}\|$ with norms less than $\epsilon/(10\sqrt{2/\pi})$

4 Structured Matrices

In many linear systems, the singular values do not decay rapidly and the matrix has a large (or even full) ϵ -rank. However, it's off-diagonal blocks of A are often

low rank. That is, given the matrix $A \in \mathbb{R}^{n \times n}$ with n being a power of 2, then we can decompose A as

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right],$$

and the off-diagonal blocks A_{12} and A_{21} are rank k with $k \ll n$. Often this low-rank off-diagonal block structure can be applied recursively, and the new off-diagonal blocks are still rank k .

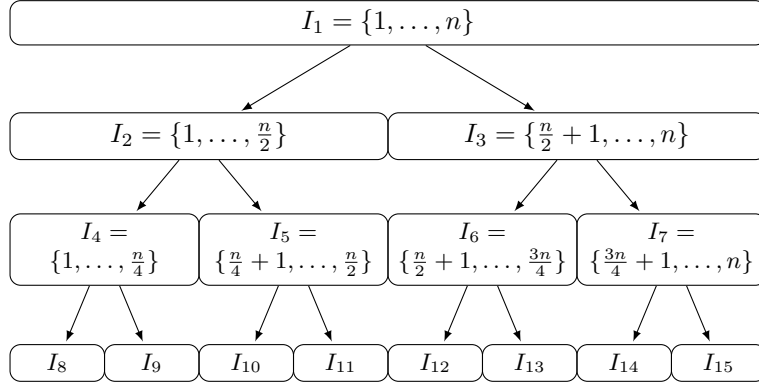
4.1 Recursive Off Diagonal Partitioning

Let's change the notation so that we can do recursive off-diagonal structured matrices. Define the index set $I_i = \{1, 2, \dots, n\}$. We let I_2 and I_3 be the first and second halves of I_1 . That is

$$I_2 = \left\{1, \dots, \frac{n}{2}\right\}$$

$$I_3 = \left\{\frac{n}{2} + 1, \dots, n\right\}$$

Continuing we have a tree structure



Then, for $\sigma, \tau \in \mathbb{N}$ we let $A_{\sigma\tau}$ be the submatrix of A with rows in the index set I_σ and columns in the index set I_τ . Then, the new labelling from the above partition is

$$A = \begin{array}{cc} & \begin{matrix} I_2 & I_3 \end{matrix} \\ \begin{matrix} I_2 \\ I_3 \end{matrix} & \left[\begin{array}{c|c} A_{22} & A_{23} \\ \hline A_{32} & A_{33} \end{array} \right] \end{array}$$

In general

$$I_\sigma \left\{ \begin{array}{c} I_\tau \\ A_{\sigma\tau} \end{array} \right\}$$

Applying the off-diagonal structure recursively,

4.2 Operations with Partitioned Matrices

If we need to compute $A\vec{x}$, this can be done as follows

$$A\vec{X} = \left\{ \begin{bmatrix} & & & \\ & 0 & & A_{2,3} \\ & & & \\ & A_{3,2} & & 0 \\ & & & \end{bmatrix} + \begin{bmatrix} & & & \\ & 0 & A_{4,5} & \\ & A_{5,4} & 0 & \\ & & & 0 \\ & & 0 & A_{6,7} \\ & & & \\ & & A_{7,6} & 0 \\ & & & \end{bmatrix} + \right.$$

$$\begin{bmatrix} \vec{x} \end{bmatrix}_{52}$$

$$A\vec{x} = \begin{bmatrix} A_{2,3}\vec{x}_3 \\ A_{3,2}\vec{x}_2 \end{bmatrix} + \begin{bmatrix} A_{4,5}\vec{x}_5 \\ A_{5,4}\vec{x}_4 \\ A_{6,7}\vec{x}_7 \\ A_{7,6}\vec{x}_6 \end{bmatrix} + \begin{bmatrix} A_{8,9}\vec{x}_9 \\ A_{9,8}\vec{x}_8 \\ A_{10,11}\vec{x}_{11} \\ A_{11,10}\vec{x}_{10} \\ A_{12,13}\vec{x}_{13} \\ A_{13,12}\vec{x}_{12} \\ A_{14,15}\vec{x}_{15} \\ A_{15,14}\vec{x}_{14} \end{bmatrix} + \begin{bmatrix} A_{8,8}\vec{x}_8 \\ A_{9,9}\vec{x}_9 \\ \vdots \\ A_{15,15}\vec{x}_{15} \end{bmatrix}$$

All of these matrices have rank k except for the ones in the last vector. Therefore, the number of flops is

$$2\frac{N}{2}k + 4\frac{N}{4}k + 8\frac{N}{8}k + 16\left(\frac{N}{16}\right)^2 = 3Nk + \frac{N^2}{16}$$

which is faster than N^2 for sufficiently large N .

If we can continue dividing diagonal blocks until the diagonal blocks are $2k \times 2k$ or smaller, the number of flops is

$$2\frac{N}{2}k + 4\frac{N}{4}k + 8\frac{N}{8}k + \dots + 16\left(\frac{N}{16}\right)^2 = 3Nk + \frac{N^2}{16}$$

For matrices with this structure, we can write a matvec recursively.

Algorithm 3 Function $b = \text{matvec}(A, x)$

if $\text{size}(x) < \text{min_size}$ **then**

$b = Ax$

else

 Split

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

 Split

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \text{matvec}(A_{11}, x_1) \\ \text{matvec}(A_{22}, x_2) \end{bmatrix}$$

end if

The matvec is somewhat straightforward. More interesting is computing A^{-1} when A has low rank off-diagonal blocks. We could use the Schur complements we learnt in week 1.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} S_{11}^{-1} & -S_{11}^{-1}A_{12}A_{22}^{-1} \\ -A_{22}^{-1}A_{21}S_{11}^{-1} & A_{22}^{-1} + A_{22}^{-1}A_{21}S_{11}^{-1}A_{12}A_{22}^{-1} \end{bmatrix}$$

where $S_{11} = A_{11} - A_{12}A_{22}^{-1}A_{21}$. What is important to note is that inverting an $N \times N$ matrix now only requires inverting $2 \frac{N}{2} \times \frac{N}{2}$ matrices.

Algorithm 4 Function $B = \text{matinv}(A)$

if $\text{size}(x) < \text{min_size}$ **then**

$b = A^{-1}$

else

Split

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$X_{22} = \text{matinv}(A_{22})$

$T_{11} = \text{matinv}(A_{11} - A_{12}X_{22}A_{21})$

Set

$$B = \begin{bmatrix} T_{11} & -T_{11}A_{12}X_{22} \\ -X_{22}A_{21}T_{11} & X_{22} + X_{22}A_{21}T_{11}A_{12}X_{22} \end{bmatrix}$$

end if

An issue with this technique is that we can not compute the (2,2) block of B until the (1,1) block of B is formed. This makes the method inherently serial. However, we have other techniques to invert a low rank correction of a matrix.

$$\begin{aligned} A &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} + \begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix} \\ &= \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} + \begin{bmatrix} 0 & U_1 \tilde{A}_{12} V_2^T \\ U_2 \tilde{A}_{21} V_1^T & 0 \end{bmatrix} \end{aligned}$$

where $U_1, U_2 \in \mathbb{R}^{n/2 \times k}$ $\tilde{A}_{12}, \tilde{A}_{21} \in \mathbb{R}^{k \times k}$ $V_1, V_2 \in \mathbb{R}^{n/2 \times k}$

$$= \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} + \begin{array}{c} \begin{matrix} n \times 2k & 2k \times 2k & 2k \times n \end{matrix} \\ \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} 0 & \tilde{A}_{12} \\ \tilde{A}_{21} & 0 \end{bmatrix} \begin{bmatrix} V_1^T & 0 \\ 0 & V_2^T \end{bmatrix} \end{array}$$

The Woodbury identity is

$$\left(D + U \tilde{A} V^T \right)^{-1} = D^{-1} - D^{-1} U \left(\tilde{A} + V^T D^{-1} U \right)^{-1} V D^{-1}$$

Applying to our matrix structure

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} - \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \times \\ \left(\begin{bmatrix} 0 & \tilde{A}_{12} \\ \tilde{A}_{21} & 0 \end{bmatrix} + \begin{bmatrix} V_1^T & 0 \\ 0 & V_2^T \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \right)^{-1} \times \\ \begin{bmatrix} V_1^T & 0 \\ 0 & V_2^T \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix}$$

Therefore, we only have to invert

- 1) $2 \times n/2 \times n/2$ matrices (A_{11}^{-1} and A_{22}^{-1})
- 2) $1 \times 2k \times 2k$ matrix (term in brackets)

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} - \begin{bmatrix} A_{11}^{-1}U_1 & 0 \\ 0 & A_{22}^{-1}U_2 \end{bmatrix} \begin{bmatrix} V_1^T A_{11}^{-1}U_1 & \tilde{A}_{21} \\ \tilde{A}_{21} & V_2^T A_{22}^{-1}U_2 \end{bmatrix}^{-1} \begin{bmatrix} V_1^T A_{11}^{-1} & 0 \\ 0 & V_2^T A_{22}^{-1} \end{bmatrix}$$

With this new formulation, we can form each of the blocks of $B = A^{-1}$ in parallel. The result is a much more practical method than the one arising from the Schuure complement.

4.3 Kernel Equations

These structured matrices arise when solving a variety of problems that involve a kernel function

$$K : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$$

Often, K is a function of the difference between its arguments, or the norm of this distance. That is

$$K(\vec{x}, \vec{y}) = K(\vec{x} - \vec{y}) = K(\|\vec{x} - \vec{y}\|)$$

Given such a kernel function, and a set of points $\{\vec{x}_i\}_{i=1}^N \in \mathbb{R}^2$, we often need to solve

$$f(\vec{x}_i) = \lambda \sigma(x_i) + \sum_{j=1}^N K(\vec{x}_i, \vec{x}_j) \sigma(\vec{x}_j) \quad i = 1, \dots, N$$

where f is a given function, λ is a given number, and σ is unknown. Sometimes, $K(\vec{x}_i, \vec{x}_j)$ is undefined and in this case, we skip the $j = i$ term.

$$f(\vec{x}_i) = \lambda \sigma(x_i) + \sum_{\substack{j=1 \\ j \neq i}}^N K(\vec{x}_i, \vec{x}_j) \sigma(\vec{x}_j) \quad i = 1, \dots, N$$

In matrix form,

$$\begin{bmatrix} f(\vec{x}_1) \\ f(\vec{x}_2) \\ \vdots \\ f(\vec{x}_N) \end{bmatrix} = \lambda \begin{bmatrix} \sigma(\vec{x}_1) \\ \sigma(\vec{x}_2) \\ \vdots \\ \sigma(\vec{x}_N) \end{bmatrix} + \begin{bmatrix} K(\vec{x}_1, \vec{x}_1) & K(\vec{x}_1, \vec{x}_2) & \dots & K(\vec{x}_1, \vec{x}_N) \\ K(\vec{x}_2, \vec{x}_1) & K(\vec{x}_2, \vec{x}_2) & \dots & K(\vec{x}_2, \vec{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(\vec{x}_N, \vec{x}_1) & K(\vec{x}_N, \vec{x}_2) & \dots & K(\vec{x}_N, \vec{x}_N) \end{bmatrix} \begin{bmatrix} \sigma(\vec{x}_1) \\ \sigma(\vec{x}_2) \\ \vdots \\ \sigma(\vec{x}_N) \end{bmatrix}$$

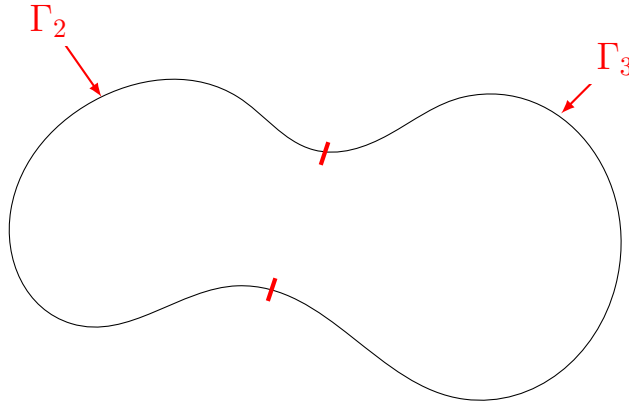
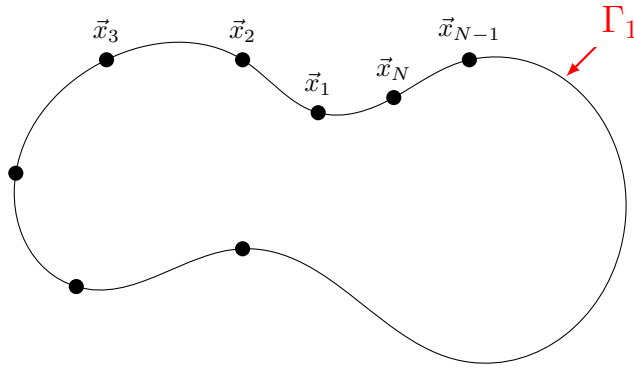
or

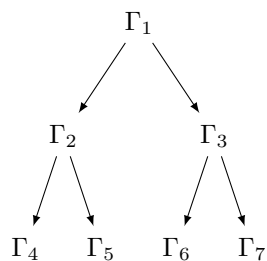
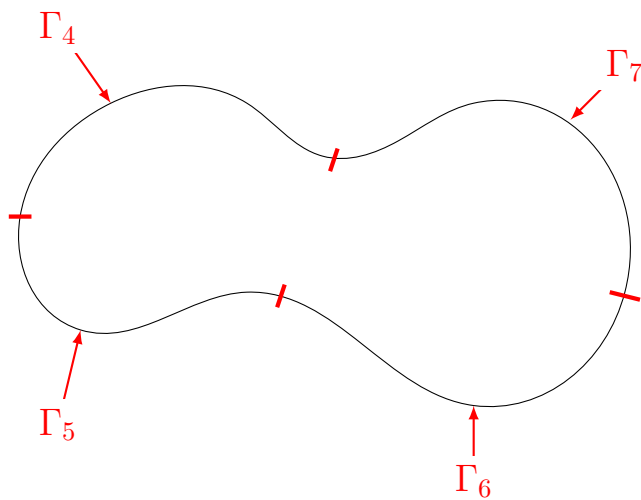
$$\vec{f} = \lambda I \vec{\sigma} + K \times \vec{\sigma}$$

or

$$\vec{f} = (\lambda I + K) \vec{\sigma} =: A \vec{\sigma}$$

Let's focus on the case where $\{\vec{x}_i\} \in \mathbb{R}^2$ are all on some closed curve in \mathbb{R}^2



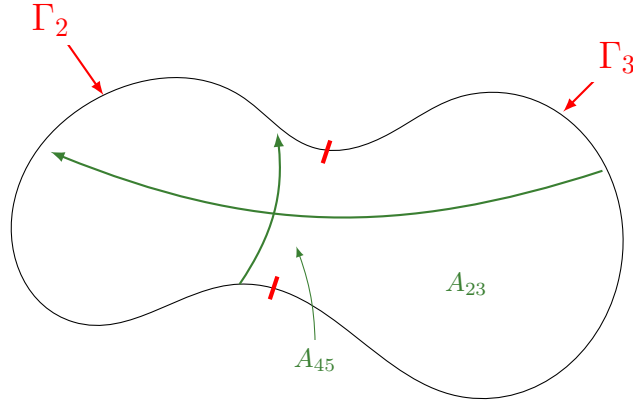
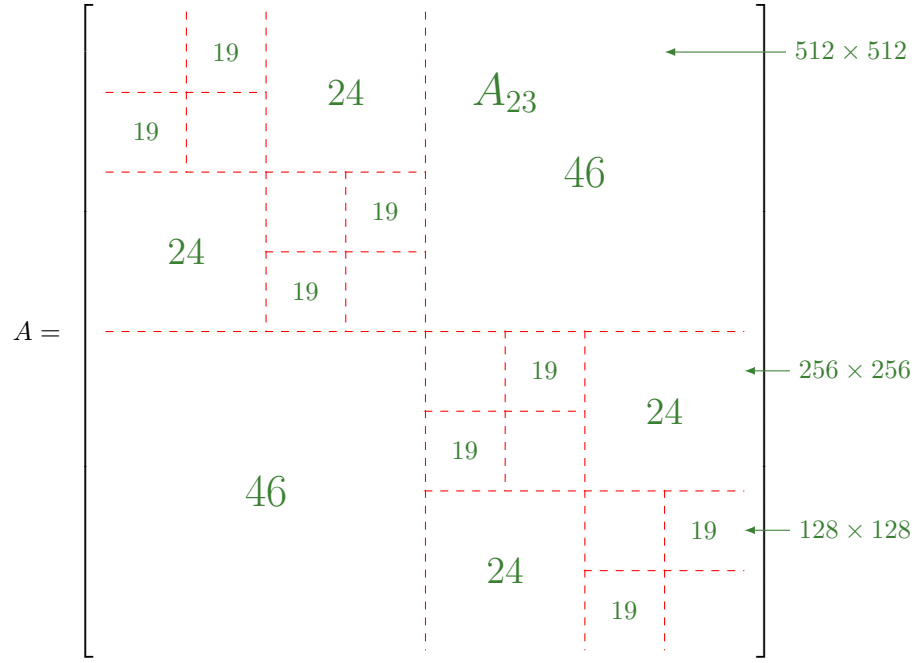


Example

Let $\lambda = 0$ and $K(\vec{x}, \vec{y}) = \log(\|\vec{x} - \vec{y}\|) = \frac{1}{2} \log(\|\vec{x} - \vec{y}\|^2)$ and let the diagonal terms be 0. That is $K(\vec{x}, \vec{x}) := 0$. Let \vec{x} be equispaced points on the five-lobed flower

$$\vec{x}(\theta) = (1 + 0.2 \cos(5\theta)) \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} \quad \theta \in [0, \pi)$$

Let's compute the 10^{-10} numerical ranks with $N = 1024$.



Given $\sigma(\vec{x}_1), \dots, \sigma(\vec{x}_N)$, the half corresponding to the points in Γ_3 are \vec{x}_i with $i = I_3$. Then, computing $A_{23}\vec{\sigma}_{I_3}$ return the function

$$\sum_{j \in I_3} K(\vec{x}_i, \vec{x}_j) \sigma(\vec{x}_j) \quad \forall \quad i \in I_2$$

For these kernel matrices, the rank of the blocks is independent of, or nearly independent of, the problem size N . However, requiring a unique SVD for each off-diagonal block is too restrictive to develop efficient methods to construct A^{-1} .

4.4 Hierarchical Semi-Separable Decomposition

A more efficient partitioning is a *hierarchical semi-separable* (HSS) decomposition

$$A = \begin{bmatrix} D_4 & A_{45} & A_{46} & A_{47} \\ A_{54} & D_5 & A_{56} & A_{57} \\ A_{64} & A_{65} & D_6 & A_{67} \\ A_{74} & A_{75} & A_{76} & D_7 \end{bmatrix}$$

An HSS decomposition writes each of the off-diagonal blocks as

$$A_{ij} = U_i \tilde{A}_{ij} V_j^T$$

where $k \ll n$. The assumption we have made is that one matrix U_i works for compressing $A_{ij} \forall j \neq i$. Similarly, one matrix V_j^T works for compressing $A_{ij} \forall j \neq i$. That is, many of the blocks use the same left and right “singular vectors”. Note that we haven’t exploited an hierarchical structure yet. Using this decomposition

$$A = \begin{bmatrix} D_4 & U_4 \tilde{A}_{45} V_5^T & U_4 \tilde{A}_{46} V_6^T & U_4 \tilde{A}_{47} V_7^T \\ U_5 \tilde{A}_{54} V_4^T & D_5 & U_5 \tilde{A}_{56} V_6^T & U_5 \tilde{A}_{57} V_7^T \\ U_6 \tilde{A}_{64} V_4^T & U_6 \tilde{A}_{65} V_5^T & D_6 & U_6 \tilde{A}_{67} V_7^T \\ U_7 \tilde{A}_{74} V_4^T & U_7 \tilde{A}_{75} V_5^T & U_7 \tilde{A}_{76} V_6^T & D_7 \end{bmatrix}$$

$$A = \begin{bmatrix} D_4 & & & \\ & D_5 & & \\ & & D_6 & \\ & & & D_7 \end{bmatrix} + \begin{bmatrix} U_4 & & & \\ & U_5 & & \\ & & U_6 & \\ & & & U_7 \end{bmatrix} \begin{bmatrix} 0 & \tilde{A}_{45} & \tilde{A}_{46} & \tilde{A}_{47} \\ \tilde{A}_{54} & 0 & \tilde{A}_{56} & \tilde{A}_{57} \\ \tilde{A}_{64} & \tilde{A}_{65} & 0 & \tilde{A}_{67} \\ \tilde{A}_{74} & \tilde{A}_{75} & \tilde{A}_{76} & 0 \end{bmatrix} \begin{bmatrix} V_4^T & & & \\ & V_5^T & & \\ & & V_6^T & \\ & & & V_7^T \end{bmatrix}$$

$4n \times 4n$

$4n \times 4k$

$$A = \begin{bmatrix} \text{diag} & & & \\ & \text{diag} & & \\ & & \text{diag} & \\ & & & \text{diag} \end{bmatrix} + \begin{bmatrix} \text{diag} & & & \\ & \text{diag} & & \\ & & \text{diag} & \\ & & & \text{diag} \end{bmatrix} \begin{bmatrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} \begin{bmatrix} \text{diag} & & & \\ & \text{diag} & & \\ & & \text{diag} & \\ & & & \text{diag} \end{bmatrix}$$

$4k \times 4k$

$4k \times 4n$

$$= D + U \tilde{A} V^T$$

This factorization is not possible if each A_{ij} required a different U_{ij} for each j and a different V_{ij}^T for each i . Applying the Woodbury identity

$$A^{-1} = \left(D + U \tilde{A} V^T \right)^{-1} = D^{-1} - D^{-1} U \left(\tilde{A} - V^T D^{-1} U \right)^{-1} V^T D^{-1}$$

$$A^{-1} = \begin{bmatrix} \square & & & \\ & \square & & \\ & & \square & \\ & & & \square \end{bmatrix} + \begin{bmatrix} \boxed{} & & & \\ & \boxed{} & & \\ & & \boxed{} & \\ & & & \boxed{} \end{bmatrix} \begin{bmatrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} \begin{bmatrix} \boxed{} & & & \\ & \boxed{} & & \\ & & \boxed{} & \\ & & & \boxed{} \end{bmatrix}$$

$G \qquad E \qquad (\tilde{A} - \tilde{D})^{-1} \qquad F^T$

where

$$\begin{aligned} \tilde{D} &= V^T D^{-1} U \\ E &= D^{-1} \\ F &= V^T D^{-1} \\ G &= D^{-1} \end{aligned}$$

The computational cost is now 4 $n \times n$ matrix inversion and a $4k \times 4k$ matrix inverse

$$\Rightarrow \mathcal{O}(4n^3 + (4k)^3) = \mathcal{O}(4n^3 + 64k^3) = \mathcal{O}(4n^3)$$

Assuming $k \ll n$

In contrast, inverting the $4n \times 4n$ matrix costs

$$\mathcal{O}((4n)^3) = \mathcal{O}(64n^3)$$

\Rightarrow A speed up of $16\times$.

However, this can speed up even more. First, let the decompositions of $A_{ij} = U_i \tilde{A}_{ij} V_j^T$ be the interpolative decomposition (ID). That is, we find an index set J , of size k , of a low-rank matrix A_{ij}

$$A_{ij} = A_{ij}(:, J) X$$

where $X(:, J) = I_k$ and all entries of X are bounded by 2 in absolute value. We can think of X as being V_j^T . Applying an ID to A^T gives a U_i .

Example

$$A_{ij} = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 4 \\ 1 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 3 \\ 4 \\ 3 \end{bmatrix} = 1 \cdot \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 2 & 1 & 3 \end{bmatrix} + \frac{3}{5} \begin{bmatrix} 1 & 3 & 4 \end{bmatrix}$$

$$\Rightarrow A_{ij} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1/5 & 3/5 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$= U_i \tilde{A}_{ij} V_j^T$$

Note the \tilde{A}_{ij} is a submatrix of A_{ij} . Therefore, we only need to store index sets and the parts of U_i and V_j^T that are not in the index set of J .

Next we find a way to compress A . Given a vector $[f_4, f_5, f_6, f_7]^T$, finding A^{-1} is equivalent to developing a method to solve

$$A \begin{bmatrix} q_4 \\ q_5 \\ q_6 \\ q_7 \end{bmatrix} = \begin{bmatrix} f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}$$

Recall that

$$A = \begin{bmatrix} D_4 & U_4 \tilde{A}_{45} V_5^T & U_4 \tilde{A}_{46} V_6^T & U_4 \tilde{A}_{47} V_7^T \\ U_5 \tilde{A}_{54} V_4^T & D_5 & U_5 \tilde{A}_{56} V_6^T & U_5 \tilde{A}_{57} V_7^T \\ U_6 \tilde{A}_{64} V_4^T & U_6 \tilde{A}_{65} V_5^T & D_6 & U_6 \tilde{A}_{67} V_7^T \\ U_7 \tilde{A}_{74} V_4^T & U_7 \tilde{A}_{75} V_5^T & U_7 \tilde{A}_{76} V_6^T & D_7 \end{bmatrix}$$

Introduce the extended variables $\tilde{q}_i = V_i^T q_i$

Then, $A\vec{q} = \vec{f}$ is equivalent to the extended system

$$\left[\begin{array}{cccc|cccc} D_4 & & & & 0 & U_4 \tilde{A}_{45} & U_4 \tilde{A}_{46} & U_4 \tilde{A}_{47} \\ & D_5 & & & U_5 \tilde{A}_{54} & 0 & U_5 \tilde{A}_{56} & U_5 \tilde{A}_{57} \\ & & D_6 & & U_6 \tilde{A}_{64} & U_6 \tilde{A}_{65} & 0 & U_6 \tilde{A}_{67} \\ & & & D_7 & U_7 \tilde{A}_{74} & U_7 \tilde{A}_{75} & U_7 \tilde{A}_{76} & 0 \\ \hline -V_4^T & & & & I & & & \\ & -V_5^T & & & & I & & \\ & & -V_6^T & & & & I & \\ & & & -V_7^T & & & & I \end{array} \right] \begin{bmatrix} q_4 \\ q_5 \\ q_6 \\ q_7 \\ \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} = \begin{bmatrix} f_4 \\ f_5 \\ f_6 \\ f_7 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We now eliminate q_4, q_5, q_6, q_7 using a (block) Schur complement so that we are instead solving for $\tilde{q}_4, \tilde{q}_5, \tilde{q}_6, \tilde{q}_7$

$$\Rightarrow \begin{bmatrix} D_4 & & & \\ & D_5 & & \\ & & D_6 & \\ & & & D_7 \end{bmatrix} \begin{bmatrix} q_4 \\ q_5 \\ q_6 \\ q_7 \end{bmatrix} = - \begin{bmatrix} 0 & U_4 \tilde{A}_{45} & U_4 \tilde{A}_{46} & U_4 \tilde{A}_{47} \\ U_5 \tilde{A}_{54} & 0 & U_5 \tilde{A}_{56} & U_5 \tilde{A}_{57} \\ U_6 \tilde{A}_{64} & U_6 \tilde{A}_{65} & 0 & U_6 \tilde{A}_{67} \\ U_7 \tilde{A}_{74} & U_7 \tilde{A}_{75} & U_7 \tilde{A}_{76} & 0 \end{bmatrix} \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} + \begin{bmatrix} f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} -V_4^T & & & \\ & -V_5^T & & \\ & & -V_6^T & \\ & & & -V_7^T \end{bmatrix} \begin{bmatrix} D_4 & & & \\ & D_5 & & \\ & & D_6 & \\ & & & D_7 \end{bmatrix}^{-1} \cdot \left(- \begin{bmatrix} 0 & U_4 \tilde{A}_{45} & U_4 \tilde{A}_{46} & U_4 \tilde{A}_{47} \\ U_5 \tilde{A}_{54} & 0 & U_5 \tilde{A}_{56} & U_5 \tilde{A}_{57} \\ U_6 \tilde{A}_{64} & U_6 \tilde{A}_{65} & 0 & U_6 \tilde{A}_{67} \\ U_7 \tilde{A}_{74} & U_7 \tilde{A}_{75} & U_7 \tilde{A}_{76} & 0 \end{bmatrix} \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} + \begin{bmatrix} f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} \right) + I \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} V_4^T D_4^{-1} & & & \\ & V_5^T D_5^{-1} & & \\ & & V_6^T D_6^{-1} & \\ & & & V_7^T D_7^{-1} \end{bmatrix} \begin{bmatrix} 0 & U_4 \tilde{A}_{45} & U_4 \tilde{A}_{46} & U_4 \tilde{A}_{47} \\ U_5 \tilde{A}_{54} & 0 & U_5 \tilde{A}_{56} & U_5 \tilde{A}_{57} \\ U_6 \tilde{A}_{64} & U_6 \tilde{A}_{65} & 0 & U_6 \tilde{A}_{67} \\ U_7 \tilde{A}_{74} & U_7 \tilde{A}_{75} & U_7 \tilde{A}_{76} & 0 \end{bmatrix} \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} + \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} = \begin{bmatrix} V_4^T D_4^{-1} f_4 \\ V_5^T D_5^{-1} f_5 \\ V_6^T D_6^{-1} f_6 \\ V_7^T D_7^{-1} f_7 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} I & V_4^T D_4^{-1} U_4 \tilde{A}_{45} & V_4^T D_4^{-1} U_4 \tilde{A}_{46} & V_4^T D_4^{-1} U_4 \tilde{A}_{47} \\ V_5^T D_5^{-1} U_5 \tilde{A}_{54} & I & V_5^T D_5^{-1} U_5 \tilde{A}_{56} & V_5^T D_5^{-1} U_5 \tilde{A}_{57} \\ V_6^T D_6^{-1} U_6 \tilde{A}_{64} & V_6^T D_6^{-1} U_6 \tilde{A}_{65} & I & V_6^T D_6^{-1} U_6 \tilde{A}_{67} \\ V_7^T D_7^{-1} U_7 \tilde{A}_{74} & V_7^T D_7^{-1} U_7 \tilde{A}_{75} & V_7^T D_7^{-1} U_7 \tilde{A}_{76} & I \end{bmatrix} \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} = \begin{bmatrix} V_4^T D_4^{-1} f_4 \\ V_5^T D_5^{-1} f_5 \\ V_6^T D_6^{-1} f_6 \\ V_7^T D_7^{-1} f_7 \end{bmatrix}$$

We have defined \tilde{A}_{ij} for $i \neq j$, but not \tilde{A}_{ii} .

We let

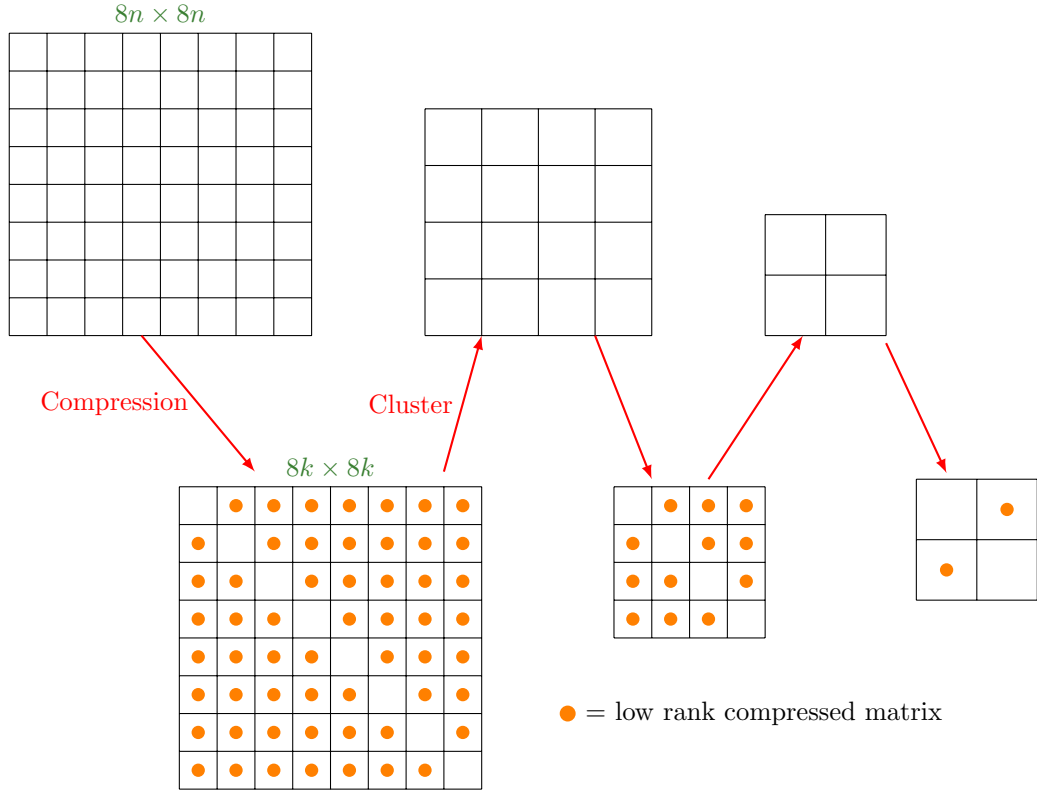
$$\tilde{A}_{ii} = (V_i^T D_i^{-1} U_i)^{-1}$$

Multiplying each row by \tilde{A}_{ii} , we have

$$\begin{bmatrix} \tilde{A}_{44} & \tilde{A}_{45} & \tilde{A}_{46} & \tilde{A}_{47} \\ \tilde{A}_{54} & \tilde{A}_{55} & \tilde{A}_{56} & \tilde{A}_{57} \\ \tilde{A}_{64} & \tilde{A}_{65} & \tilde{A}_{66} & \tilde{A}_{67} \\ \tilde{A}_{74} & \tilde{A}_{75} & \tilde{A}_{76} & \tilde{A}_{77} \end{bmatrix} \begin{bmatrix} \tilde{q}_4 \\ \tilde{q}_5 \\ \tilde{q}_6 \\ \tilde{q}_7 \end{bmatrix} = \begin{bmatrix} \tilde{f}_4 \\ \tilde{f}_5 \\ \tilde{f}_6 \\ \tilde{f}_7 \end{bmatrix}$$

where $\tilde{f}_i = \tilde{A}_{ii} V_i^T D_i^{-1} f_i$.

This is now a $4k \times 4k$ matrix for \tilde{q}_i instead of a $4n \times 4n$ matrix for q_i . Moreover, each off-diagonal block is simply a submatrix of the full matrix A . The speed up is $(\frac{n}{k})^3$. To get an asymptotically faster scheme, we recurse.



The only matrices in this decomposition that are new are the diagonal blocks. All other blocks are submatrices of the original coefficient matrix since we are using the ID.

In matrix form, we have constructed the following telescoping factorization

$$A = U^{(3)} \left(U^{(2)} \left(U^{(1)} B^{(0)} V^{(1)T} + B^{(1)} \right) V^{(2)T} + B^{(2)} \right) V^{(3)T} + D^{(3)}$$

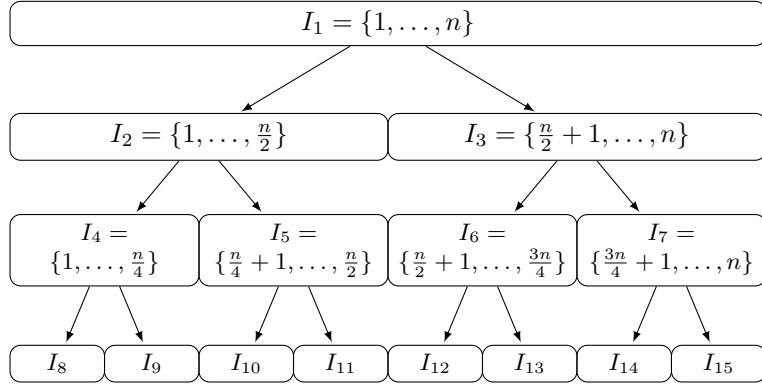
$$\begin{aligned}
&= \left[\begin{array}{cccccccc} \square & & & & & & & \\ & \square & & & & & & \\ & & \square & & & & & \\ & & & \square & & & & \\ & & & & \square & & & \\ & & & & & \square & & \\ & & & & & & \square & \\ & & & & & & & \square \end{array} U^{(3)} \right] \left(\left[\begin{array}{cccc} \square & & & \\ & \square & & \\ & & \square & \\ & & & \square \end{array} U^{(2)} \right] \left(\left[\begin{array}{c} \square \\ \square \end{array} U^{(1)} \right] \left[\begin{array}{cc} \square & \\ & \square \end{array} B^{(0)} \right] \left[\begin{array}{cc} \square & \\ & \square \end{array} V^{(1)T} \right] + \right. \\
&\quad \left. \left[\begin{array}{ccc} \square & & \\ \square & \square & \\ & \square & \square \end{array} B^{(1)} \right] \left[\begin{array}{cccc} \square & & & \\ & \square & & \\ & & \square & \\ & & & \square \end{array} V^{(2)T} \right] + \left[\begin{array}{ccccccc} \square & & & & & & \\ & \square & & & & & \\ & & \square & & & & \\ & & & \square & & & \\ & & & & \square & & \\ & & & & & \square & \\ & & & & & & \square \end{array} B^{(2)} \right] \right) \times \\
&\quad \left[\begin{array}{cccccccc} \square & & & & & & & \\ & \square & & & & & & \\ & & \square & & & & & \\ & & & \square & & & & \\ & & & & \square & & & \\ & & & & & \square & & \\ & & & & & & \square & \\ & & & & & & & \square \end{array} V^{(3)T} \right] + \left[\begin{array}{cccccccc} \square & & & & & & & \\ & \square & & & & & & \\ & & \square & & & & & \\ & & & \square & & & & \\ & & & & \square & & & \\ & & & & & \square & & \\ & & & & & & \square & \\ & & & & & & & \square \end{array} D^{(3)} \right]
\end{aligned}$$

Using our definitions of E , F , and \tilde{D} from two Fridays ago (March 15), after applying the Woodbury identity, we have

$$A^{-1} = E^{(3)} \left(E^{(2)} \left(E^{(1)} \tilde{D}^{(0)} F^{(1)T} + \tilde{D}^{(1)} \right) F^{(2)T} + \tilde{D}^{(2)} \right) F^{(3)T} + \tilde{D}^{(3)}$$

The only non-block diagonal matrix is $\tilde{D}^{(0)}$, but it is small, so can be computed directly.

This inversion can be written with fairly simple pseudocode. Recall that we are partitioning A into blocks corresponding to a tree structure applied to our index set $\{1, \dots, n\} = I_1$



We define all elements of the tree with no descendants as a leaf. Other nodes are called parent nodes. For each leaf node, we define,

<u>Notation</u>	<u>Size</u>	<u>Role</u>
D_τ	$n \times n$	Diagonal block of $A(I_\tau, I_\tau)$
U_τ	$n \times k$	Basis for the columns of the blocks in row τ
V_τ	$n \times k$	Basis for the rows of the blocks in column τ

For each parent node:

<u>Notation</u>	<u>Size</u>	<u>Role</u>
B_τ	$2k \times 2k$	Interaction between the children of τ
U_τ	$2k \times k$	Basis for the columns of the blocks in row τ
V_τ	$2k \times k$	Basis for the rows of the blocks in column τ

Then, the pseudocode to compute A^{-1} where A is an HSS matrix is:

```

for  $l = L, L - 1, \dots, 1$  do
  for  $\tau$  at level  $l$  do
    if  $\tau$  is a leaf node then
       $X = D_\tau$ 
    else
      Let  $\sigma_1, \sigma_2$  be  $\tau$ 's children

```

$$X = \begin{bmatrix} D_{\sigma_1} & B_{\sigma_1, \sigma_2} \\ B_{\sigma_2, \sigma_1} & D_{\sigma_2} \end{bmatrix}$$

```

    end if
     $\tilde{D}_\tau = (V_\tau^T X^{-1} U_\tau)^{-1}$ 
     $E_\tau = X^{-1} U_\tau D_\tau$ 
     $F_\tau^T = D_\tau V_\tau^T X^{-1}$ 
     $G_\tau = X^{-1} - X^{-1} U_\tau D_\tau V_\tau^T X^{-1}$ 
  end for
end for

```

$$G_1 = \begin{bmatrix} D_2 & B_{23} \\ B_{32} & D_3 \end{bmatrix}^{-1}$$

Then, G_1 is the inverse of A with some percision stored in a compressed format. Once all these matrices are constructed and compressed, it is very fast to apply G_1 to lots of vectors \vec{b} .

5 Krylov Methods

Krylov methods are iterative methods to solve a linear system $A\vec{x} = \vec{b}$. They are different from fixed-point methods such as Jacobi. Given a vector $\vec{d} \in \mathbb{R}^n$, the m^{th} Krylov space is

$$\mathcal{K}_m(A, \vec{d}) = \text{span}\{\vec{d}, A\vec{d}, A^2\vec{d}, \dots, A^{m-1}\vec{d}\}$$

At the m^{th} iterate of a Krylov method, it seeks the best solution to $A\vec{x} = \vec{b}$ in $\mathcal{K}_m(A, \vec{d})$. Since $\mathcal{K}_m \subseteq \mathcal{K}_{m+1}$, the error always decreases with m . The two main Krylov methods that we'll consider:

- 1) Conjugate Gradient(CG)
- 2) Generalized Minimal Residual method (GMRES)

Varients of these methods are preconditioned versions (pCG, pGMRES), Biconjugate Gradient method (BiCG), and Biconjugate Gradient Stabilized method (BiCGSTAB).

We will begin our discussion of Krylov methods by exploring the Conjugate Gradient method. Much of these notes come from the paper by Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, where a deeper analysis is provided.

5.1 Making $A\vec{x} = \vec{b}$ a Minimization Problem

Suppose A is symmetric ($A = A^T$) and positive definite:

$$\vec{x}^T A \vec{x} > 0 \quad \forall \vec{x} \neq \vec{0}$$

These matrices are typical in many problems including finite difference and finite element methods. Given such a matrix, it is guaranteed to be invertible.

Given the linear system $A\vec{x} = \vec{b}$, we define the quadratic form (function)

$$f(\vec{x}) = \frac{1}{2} \vec{x}^T A \vec{x} - \vec{b}^T \vec{x}$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

Because A is positive definite, the function f is shaped like an n -dimensional paraboloid (ie. a bowl). Therefore it has a single minimum.

$$\nabla f = \begin{bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix} = \vec{0}$$

We have that

$$\begin{aligned}
f(\vec{x}) = f(x_1, x_2, \dots, x_n) &= \frac{1}{2} \sum_{i=1}^n x_i \sum_{j=1}^n a_{ij} x_j - \sum_{i=1}^n b_i x_i \\
\Rightarrow \frac{\partial f}{\partial x_k} &= \frac{1}{2} \sum_{j=1}^n a_{kj} x_j + \frac{1}{2} \sum_{i=1}^n x_i a_{ik} - b_k \\
&= \frac{1}{2} \sum_{j=1}^n a_{kj} x_j + \frac{1}{2} \sum_{i=1}^n a_{ik} x_i - b_k \\
&= \frac{1}{2} (A\vec{x})_k + \frac{1}{2} (A^T \vec{x})_k - b_k \\
\Rightarrow \nabla f &= \frac{1}{2} A\vec{x} + \frac{1}{2} A^T \vec{x} - \vec{b} \\
&= \frac{1}{2} A\vec{x} + \frac{1}{2} A\vec{x} - \vec{b} \\
&= A\vec{x} - \vec{b} \\
\Rightarrow \nabla f = 0 &\Leftrightarrow A\vec{x} - \vec{b} = 0 \Leftrightarrow A\vec{x} = \vec{b}
\end{aligned}$$

That is, the solution of $A\vec{x} = \vec{b}$ is at the (local) minimum of f . Therefore, if we can minimize f , we have solved $A\vec{x} = \vec{b}$.

5.2 Method of Steepest Decent and Line Search

Imagine you are in a high-dimensional bowl and your goal is to reach the bottom as fast as possible. Moreover, you can not see the bottom and can only see what is happening in a small region around you.

At each step (iteration), we have 2 choices

- 1) Which direction to go?
- 2) How far should you go in that direction?

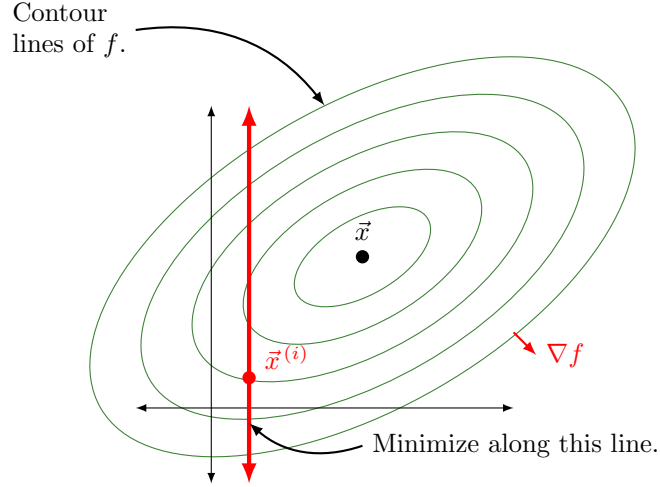
A natural choice for the direction is the direction of steepest descent. For any function f , the method of steepest descent is $\pm \nabla f$. In our case, the steepest decent is in the direction $-\nabla f$. We already know that

$$-\nabla f = \vec{b} - A\vec{x}$$

Therefore, given a current iterate $\vec{x}^{(i)}$, the new iterate should be

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + \alpha^{(i)} \vec{r}^{(i)}$$

where $\vec{r}^{(i)} = \vec{b} - A\vec{x}^{(i)}$ is the residual, and $\alpha^{(i)} \in \mathbb{R}^+$ is the step size. To find the optimal $\alpha^{(i)}$, we optimize $f(\vec{x}^{(i+1)})$. This is a line search and the role of $\alpha^{(i)}$. If $n = 2$, we can draw that is happening



To find the optimal $\alpha^{(i)}$, we optimize

$$f(\vec{x}^{(i+1)}) = f(\vec{x}^{(i)} + \alpha \vec{r}^{(i)})$$

with respect to α . This can be done by solving

$$\begin{aligned} \frac{d}{d\alpha} f(\vec{x}^{(i+1)}) &= 0 \\ \Rightarrow \frac{d}{d\alpha} f(\vec{x}^{(i)} + \alpha \vec{r}^{(i)}) &= 0 \end{aligned}$$

Claim:

$$\frac{d}{d\alpha} f(\vec{x}^{(i)} + \alpha \vec{r}^{(i)}) = \nabla f(\vec{x}^{(i)} + \alpha \vec{r}^{(i)}) \cdot \vec{r}^{(i)}$$

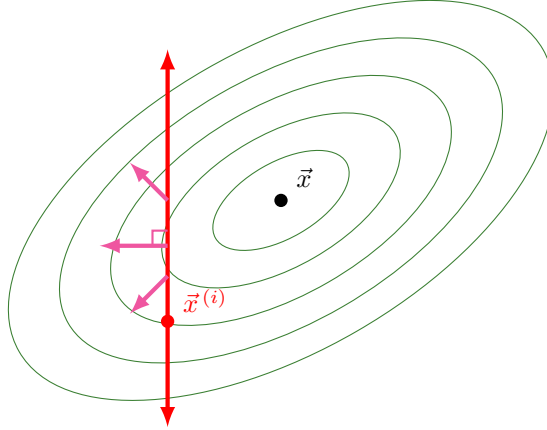
Proof:

$$\begin{aligned}
\frac{d}{d\alpha} f\left(\vec{x}^{(i)} + \alpha \vec{r}^{(i)}\right) &= \frac{d}{d\alpha} f\left(\begin{bmatrix} x_1^{(i)} + \alpha r_1^{(i)} \\ x_2^{(i)} + \alpha r_2^{(i)} \\ \vdots \\ x_n^{(i)} + \alpha r_n^{(i)} \end{bmatrix}\right) \\
&= \frac{\partial f}{\partial x_1} \Big|_{\vec{x}^{(i)} + \alpha \vec{r}^{(i)}} r_1^{(i)} + \frac{\partial f}{\partial x_2} \Big|_{\vec{x}^{(i)} + \alpha \vec{r}^{(i)}} r_2^{(i)} \\
&\quad + \dots + \frac{\partial f}{\partial x_n} \Big|_{\vec{x}^{(i)} + \alpha \vec{r}^{(i)}} r_n^{(i)} \\
&= \nabla f\left(\vec{x}^{(i)} + \alpha \vec{r}^{(i)}\right) \cdot \vec{r}^{(i)}
\end{aligned}$$

Therefore, the optimal $\alpha^{(i)}$ satisfies

$$\nabla f\left(\vec{x}^{(i)} + \alpha \vec{r}^{(i)}\right) \cdot \vec{r}^{(i)} = 0$$

That is, $\alpha^{(i)}$ should be chosen so that $\vec{r}^{(i)}$ is orthogonal to $\nabla f\left(\vec{x}^{(i+1)}\right)$



To find this value of α , since $\nabla f\left(\vec{x}^{(i+1)}\right) = -\vec{r}^{(i+1)}$, we need $\vec{r}^{(i+1)} \cdot \vec{r}^{(i)} = 0$

$$\begin{aligned}
&\Leftrightarrow \vec{r}^{(i+1)T} \vec{r}^{(i)} = 0 \\
&\Leftrightarrow \left(\vec{b} - A\vec{x}^{(i+1)} \right)^T \vec{r}^{(i)} = 0 \\
&\Leftrightarrow \left(\vec{b} - A \left(\vec{x}^{(i)} + \alpha \vec{r}^{(i)} \right) \right)^T \vec{r}^{(i)} = 0 \\
&\Leftrightarrow \left(\vec{b} - A\vec{x}^{(i)} \right)^T \vec{r}^{(i)} - \alpha \left(A\vec{r}^{(i)} \right)^T \vec{r}^{(i)} = 0 \\
&\Leftrightarrow \vec{r}^{(i)T} \vec{r}^{(i)} - \alpha \vec{r}^{(i)T} A^T \vec{r}^{(i)} = 0 \\
&\Leftrightarrow \vec{r}^{(i)T} \vec{r}^{(i)} - \alpha \vec{r}^{(i)T} A \vec{r}^{(i)} = 0 \\
&\Leftrightarrow \alpha = \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{r}^{(i)T} A \vec{r}^{(i)}}
\end{aligned}$$

Therefore the optimal α to take in the line search is

$$\alpha^{(i)} = \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{r}^{(i)T} A \vec{r}^{(i)}}$$

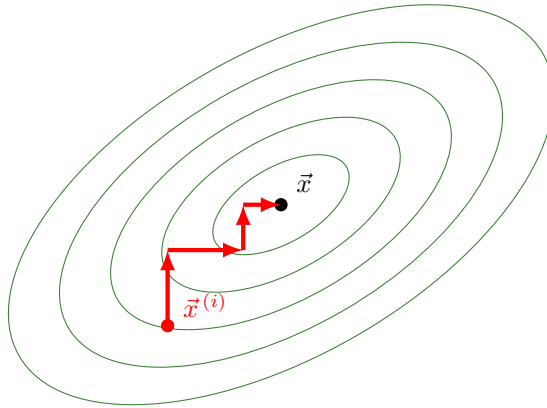
Therefore, our algorithm to solve $A\vec{x} = \vec{b}$ is

```

for  $i = 0, 1, 2, \dots$  do
   $\vec{r}^{(i)} = \vec{b} - A\vec{x}^{(i)}$ 
   $\alpha^{(i)} = \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{r}^{(i)T} A \vec{r}^{(i)}}$ 
   $\vec{x}^{(i+1)} = \vec{x}^{(i)} + \alpha^{(i)} \vec{r}^{(i)}$ 
end for

```

By choosing the optimal $\alpha^{(i)}$, we have that consecutive search directions are orthogonal. This creates a zigzag pattern as $\vec{x}^{(i)}$ converges to \vec{x} .



We define the error at iteration i to be

$$\vec{e}^{(i)} = \vec{x}^{(i)} - \vec{x}$$

To analyze the convergence of steepest descent, it is convenient to use the energy norm

$$\|\vec{e}\|_A := \sqrt{\vec{e}^T A \vec{e}}$$

Since A is spd, all its eigenvalues are real and positive, and are also the singular values of A . Letting λ_{\max} and λ_{\min} be the biggest and smallest eigenvalues, then

$$\|\vec{e}^{(i+1)}\|_A = \|\vec{e}^{(i)}\|_A \omega$$

where

$$\omega \leq \frac{\kappa - 1}{\kappa + 1}$$

where $\kappa = \lambda_{\max}/\lambda_{\min} \geq 1$ is the condition number of A . Therefore, convergence is fastest when $\kappa \approx 1$.

In general, we have

$$\begin{aligned} \|\vec{e}^{(i)}\|_A &= \|\vec{e}^{(i-1)}\|_A \omega = \|\vec{e}^{(i-2)}\|_A \omega^2 \\ &= \|\vec{e}^{(0)}\|_A \left(\frac{\kappa - 1}{\kappa + 1} \right)^i \end{aligned}$$

5.3 A better choice for the search direction

Steepest descent has a major issue. The search direction at different iterations are often very similar. It would be nice if we went exactly the correct amount in that direction so that we find the solution by the n^{th} iterate.

are A -orthogonal. That is

$$\vec{d}^{(i)T} A \vec{d}^{(j)} = 0 \quad \forall i \neq j$$

Then instead of requiring $\vec{e}^{(i+1)}$ to be orthogonal to $\vec{d}^{(i)}$, we require that $\vec{e}^{(i+1)}$ is A -orthogonal to $\vec{d}^{(i)}$. This is indeed a good choice since

$$\begin{aligned} \frac{d}{d\alpha} f(\vec{x}^{(i+1)}) &= 0 \\ \Leftrightarrow \nabla f(\vec{x}^{(i+1)})^T \frac{d}{d\alpha} \vec{x}^{(i+1)} &= 0 \\ \Leftrightarrow -\vec{r}^{(i+1)T} \vec{d}^{(i)} &= 0 \\ \Leftrightarrow -\vec{d}^{(i)T} \vec{r}^{(i+1)} &= 0 \\ \Leftrightarrow -\vec{d}^{(i)T} A \vec{e}^{(i+1)} &= 0 \end{aligned}$$

Since $A\vec{e}^{(i+1)} = A(\vec{x}^{(i+1)} - \vec{x}) = A\vec{x}^{(i+1)} - \vec{b} = -\vec{r}^{(i+1)}$

That is, if $\vec{e}^{(i+1)}$ is A -orthogonal to $\vec{d}^{(i)}$, then $\vec{x}^{(i+1)} = \vec{x}^{(i)} + \alpha^{(i)} \vec{d}^{(i)}$ minimizes f along this line search direction. To find $\alpha^{(i)}$, we require

$$\begin{aligned} \vec{d}^{(i)T} A \vec{e}^{(i+1)} &= 0 \\ \Rightarrow \vec{d}^{(i)T} A(\vec{e}^{(i)} + \alpha^{(i)} \vec{d}^{(i)}) &= 0 \\ \Rightarrow \alpha^{(i)} &= -\frac{\vec{d}^{(i)T} A \vec{e}^{(i)}}{\vec{d}^{(i)T} A \vec{d}^{(i)}} \\ \alpha^{(i)} &= -\frac{\vec{d}^{(i)T} \vec{r}^{(i)}}{\vec{d}^{(i)T} A \vec{d}^{(i)}} \end{aligned}$$

Unlike before, this $\alpha^{(i)}$ can be computed since it only involves the search directions $\vec{d}^{(i)}$ and the residual $\vec{r}^{(i)} = \vec{b} - A\vec{x}^{(i)}$

What remains is to generate n A -orthogonal vectors for the search directions. This can be done with conjugate Gram-Schmidt.

Suppose we have a set of n linearly independent vectors $\vec{u}^{(0)}, \vec{u}^{(1)}, \dots, \vec{u}^{(n-1)}$. Then, to form an A -orthogonal set of vectors, we construct $\vec{d}^{(i)}$ by subtracting components that are not A -orthogonal to the previous $\vec{d}^{(j)}$, $j < i$, vectors. That is

$$\vec{d}^{(i)} = \vec{u}^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} \vec{d}^{(k)}$$

for carefully chosen β_{ik} . To enforce that $\vec{d}^{(i)}$ is A -orthogonal to $\vec{d}^{(j)}$, $i \neq j$

$$\begin{aligned}
\vec{d}^{(i)T} A \vec{d}^{(j)} &= \vec{u}^{(i)T} A \vec{d}^{(j)} + \sum_{k=0}^{i-1} \beta_{ik} \vec{d}^{(k)T} A \vec{d}^{(j)} \\
\Rightarrow 0 &= \vec{u}^{(i)T} A \vec{d}^{(j)} + \beta_{ij} \vec{d}^{(j)T} A \vec{d}^{(j)} \\
\Rightarrow \beta_{ij} &= -\frac{\vec{u}^{(i)T} A \vec{d}^{(j)}}{\vec{d}^{(j)T} A \vec{d}^{(j)}}
\end{aligned}$$

The method of performing a line search in directions that are A -orthogonal is known as the method of conjugate directions. As of now, it's cost is $\mathcal{O}(n^3)$ which is the same as Gaussian elimination. This can be reduced by making a good choice for $\vec{u}^{(0)}, \vec{u}^{(1)}, \dots, \vec{u}^{(n-1)}$

To start to understand why this is a Krylov method, define

$$\mathcal{D}_i = \text{span} \left\{ \vec{d}^{(0)}, \vec{d}^{(1)}, \dots, \vec{d}^{(i-1)} \right\}.$$

Then at iteration i , we are minimizing

$$f(\vec{x}) = \frac{1}{2} \vec{x}^T A \vec{x} - \vec{b}^T \vec{x}$$

over the space $\vec{x}^{(0)} + \mathcal{D}_i$ since

$$\begin{aligned}
\vec{x}^{(i+1)} &= \vec{x}^{(i)} + \alpha^{(i)} \vec{d}^{(i)} \\
&= \vec{x}^{(i-1)} + \alpha^{(i-1)} \vec{d}^{(i-1)} + \alpha^{(i)} \vec{d}^{(i)} \\
&= \dots \\
&= \vec{x}^{(0)} + \alpha^{(0)} \vec{d}^{(0)} + \alpha^{(1)} \vec{d}^{(1)} + \dots + \alpha^{(i)} \vec{d}^{(i)}
\end{aligned}$$

The method of conjugate gradients (CG) used the gradients of f at each iterate to form $\vec{u}^{(0)}, \vec{u}^{(1)}, \dots, \vec{u}^{(i-1)}$. That is,

$$\vec{u}^{(i)} = \nabla f(\vec{x}^{(i)}) = \vec{r}^{(i)}$$

Then, $\vec{d}^{(i)}$ is still formed with conjugate Gram-Schmidt. We first note that

$$\begin{aligned}
\vec{r}^{(i+1)} &= -A \vec{e}^{(i+1)} \\
&= -A \left(\vec{e}^{(i)} + \alpha^{(i)} \vec{d}^{(i)} \right) \\
&= \vec{r}^{(i)} - \alpha^{(i)} A \vec{d}^{(i)}
\end{aligned}$$

\Rightarrow Each new residual $\vec{r}^{(i+1)}$ is just a linear combination of $\vec{r}^{(i)}$ and $A \vec{d}^{(i)}$

$$\begin{aligned}
\Rightarrow \mathcal{D}_i &:= \text{span} \left\{ \vec{d}^{(0)}, \vec{d}^{(1)}, \dots, \vec{d}^{(i-1)} \right\} \\
&= \text{span} \left\{ \vec{r}^{(0)}, \vec{r}^{(1)}, \dots, \vec{r}^{(i-1)} \right\} \\
&= \text{span} \left\{ \vec{r}^{(0)}, A\vec{r}^{(0)}, \dots, A^{(i-1)}\vec{r}^{(0)} \right\}
\end{aligned}$$

Which is a Krylov space.

Because we have chosen orthogonal search directions for the $\vec{u}^{(i)}$, many of the β_{ij} are in fact 0 in the conjugate Gram-Schmidt process

$$\beta_{ij} = \begin{cases} \frac{1}{\alpha^{(i-1)}} & \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{d}^{(i-1)T} A \vec{d}^{(i-1)}} & i = j + 1 \\ 0 & & i > j + 1 \end{cases}$$

Finally, defining $\beta^{(i)} := \beta_{i,i+1}$, and substituting in the appropriate value for $\alpha^{(i-1)}$, it can be shown that

$$\beta^{(i)} = \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{d}^{(i-1)T} \vec{r}^{(i-1)}} = \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{r}^{(i-1)T} \vec{r}^{(i-1)}}$$

We finally can define CG.

Algorithm 5 Conjugate Gradient Method: given $\vec{x}^{(0)}, A, \vec{b}$

```

 $\vec{d}^{(0)} = \vec{r}^{(0)} = \vec{b} - A\vec{x}^{(0)}$ 
for  $i = 0, 1, \dots, n - 1$  do
   $\alpha^{(i)} = \frac{\vec{r}^{(i)T} \vec{r}^{(i)}}{\vec{d}^{(i)T} A \vec{d}^{(i)}}$ 
   $\vec{x}^{(i+1)} = \vec{x}^{(i)} + \alpha^{(i)} \vec{d}^{(i)}$ 
   $\vec{r}^{(i+1)} = \vec{r}^{(i)} - \alpha^{(i)} A \vec{d}^{(i)}$ 
   $\beta^{(i+1)} = \frac{\vec{r}^{(i+1)T} \vec{r}^{(i+1)}}{\vec{r}^{(i)T} \vec{r}^{(i)}}$ 
   $\vec{d}^{(i+1)} = \vec{r}^{(i+1)} + \beta^{(i+1)} \vec{d}^{(i)}$ 
end for

```

5.4 Convergence

CG, and other Krylov methods, are guaranteed to converge in n iterations. However, it is especially useful if it converges in m iterations

- 1) $m \ll n$ (good)
- 2) m is independent of n (best)

In the later case, the cost of solving $A\vec{x} = \vec{b}$ with CG is proportional to the cost of a single matrix-vector multiplication (matvec) which is not worse than $\mathcal{O}(n^2)$.

The convergence of CG (i.e. size of m) is closely tied to the eigenvalues of A . Letting $\Lambda(A)$ be the spectrum of A (i.e. set of all eigenvalues of A), then

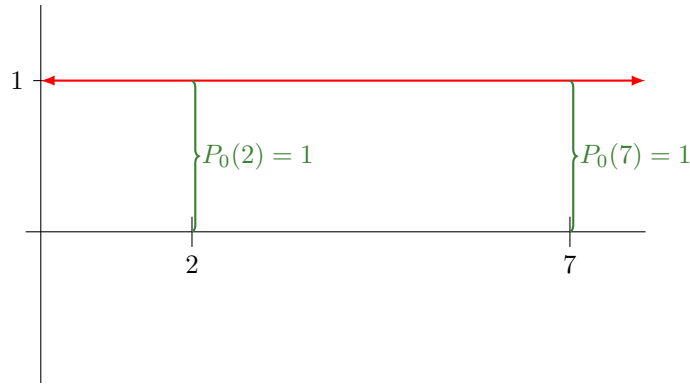
$$\|\vec{e}^{(i)}\|_A^2 \leq \min_{P_i} \left(\max_{\lambda \in \Lambda(A)} ((P_i(\lambda))^2) \right) \|\vec{e}^{(0)}\|_A^2$$

where P_i is a polynomial of degree i and $P_i(0) = 1$. That is, the error is related to how close a polynomial of degree i can be 0 at the eigenvalues and 1 at 0.

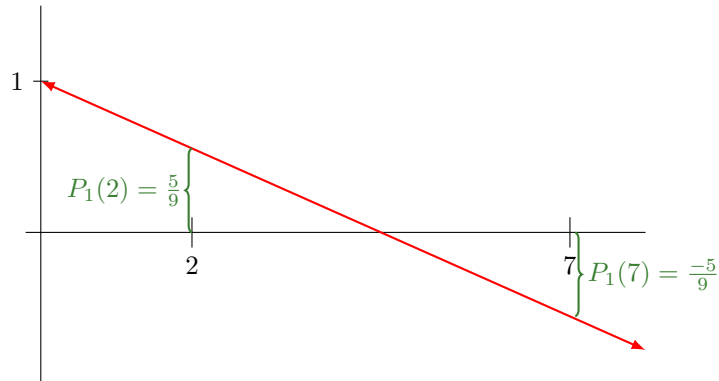
EX:

Suppose $A \in \mathcal{R}^{2 \times 2}$ has eigenvalues at $\lambda = 2$ and $\lambda = 7$. That is, $\Lambda(A) = \{2, 7\}$. Then

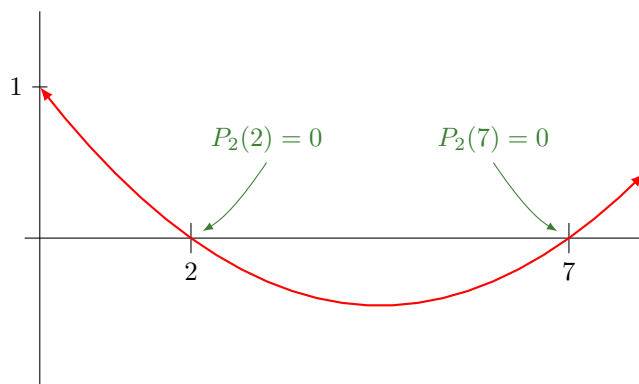
$$P_0(x) = 1$$



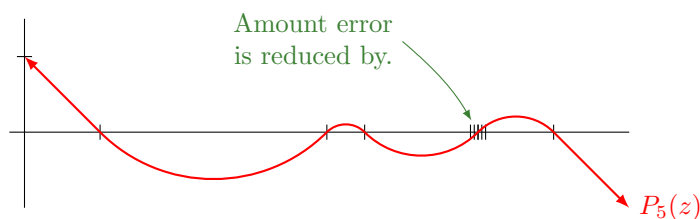
$$P_1(x) = 1 - \frac{2x}{9}$$



$$P_2(x) = \frac{(x-1)(x-7)}{14}$$



The number of CG iterations is reduced if it has multiple eigenvalues at the same location. What is also beneficial, and much more common, is the eigenvalues cluster at some location.



In a general case where A has a largest eigenvalue λ_{\max} and a smallest eigenvalue λ_{\min} , and the other eigenvalues are more or less evenly distributed in $[\lambda_{\min}, \lambda_{\max}]$

$$\|\vec{e}^{(i)}\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|\vec{e}^{(0)}\|_A$$

where $\kappa = \lambda_{\max}/\lambda_{\min} = \sigma_{\max}/\sigma_{\min}$ is the condition number of A .

In many situations, the most expensive part of CG, or another Krylov solver, is the matvecs. However, once we've reduced this cost to $\mathcal{O}(n)$ operations, either because A is sparse or has structure (KIFMM), the best of the matvec can not be reduced further with respect to complexity. Another way to further reduce the cost is to use a preconditioner.

6 Preconditioners

Let A be s.p.d. and suppose we are solving $A\vec{x} = \vec{b}$. Suppose we have some matrix P with $P \approx A$. Then, the preconditioned linear system is

$$P^{-1}A\vec{x} = P^{-1}\vec{b},$$

where P is called the preconditioner. There are two properties that are required of a good preconditioner

- 1) P^{-1} can be applied efficiently.
- 2) $P \approx A$ or $P^{-1}A \approx I$

Our second requirement for the preconditioner P is that $P \approx A$ or $P^{-1}A \approx I$. This condition can also be phrased as

- a) $\kappa(P^{-1}A) \ll \kappa(A)$ where $\kappa(A)$ is the condition number of A .
- b) The eigenvalues of $P^{-1}A$ are much more clustered than the eigenvalues of A .
- c) CG for $P^{-1}A$ requires much fewer iterations than CG applied to A .

The two most extreme “preconditioners” are $P = A$ and $P = I$. If $P = I$, then we satisfy condition 1 trivially, but not condition 2. If $P = A$, then we satisfy condition 2, but not condition 1. A good preconditioner is somewhere inbetween.

Because of the required spd property of CG, some careful algebra is required to apply CG to a preconditioned linear system. This is because, even if P and P^{-1} are symmetric, we have

$$(P^{-1}A)^T = A^T P^{-T} = AP^{-1}$$

$\Rightarrow P^{-1}A$ is symmetric iff A and P^{-1} commute.

There is a fix for this issue, and the result is the preconditioned CG (pCG) method:

Algorithm 6 Preconditioned Conjugate Gradient Method

```

 $\vec{r}^{(0)} = \vec{b} - A\vec{x}^{(0)}$ 
 $\vec{d}^{(0)} = P^{-1}\vec{b}$ 
for  $i = 0, 1, \dots, n - 1$  do
   $\alpha^{(i)} = \frac{\vec{r}^{(i)T} P^{-1} \vec{r}^{(i)}}{\vec{d}^{(i)T} A \vec{d}^{(i)}}$ 
   $\vec{x}^{(i+1)} = \vec{x}^{(i)} + \alpha^{(i)} \vec{d}^{(i)}$ 
   $\vec{r}^{(i+1)} = \vec{r}^{(i)} - \alpha^{(i)} A \vec{d}^{(i)}$ 
   $\beta^{(i+1)} = \frac{\vec{r}^{(i+1)T} P^{-1} \vec{r}^{(i+1)}}{\vec{r}^{(i)T} P^{-1} \vec{r}^{(i)}}$ 
   $\vec{d}^{(i+1)} = P^{-1} \vec{r}^{(i+1)} + \beta^{(i+1)} \vec{d}^{(i)}$ 
end for

```

If P^{-1} can be applied (matvec) efficiently, the reduction of the number of iterations of CG may be large enough to offset the cost of applying P^{-1} .

Some of the most common preconditioners are:

- 1) $P = \text{diag}(A)$
- 2) A sparse approximate inverse (SPAI) matrix minimizes $\|AT - I\|_{\mathcal{F}}$, where $\|\cdot\|_{\mathcal{F}}$ is the Frobenius norm.

$$\|A\|_{\mathcal{F}}^2 = \sum_{i=1}^n \sum_{j=1}^m a_{ij}^2$$

This minimizer is a preconditioner P^{-1} , and the minimization is done over some sparsity pattern.

$$P^{-1} = \text{argmin} \|AT - I\|_{\mathcal{F}}$$

- 3) The Incomplete LU (ILU) factorization is an approximation LU decomposition of A . One version of the ILU drops all entries of the LU factorization of A that are below some threshold. Therefore

$$A \approx LU$$

and we let $P = LU$. To apply P^{-1} , we need to solve

$$LU\vec{x} = \vec{b}.$$

This is done by solving

$$L\vec{y} = \vec{b} \quad \text{and then} \quad U\vec{x} = \vec{y}$$

- 4) The multigrid algorithm we studied approximately solves $A\vec{x} = \vec{b}$ for certain matrices A . We define the preconditioner as

$$P^{-1}\vec{b} := \text{GMG}(\vec{b})$$

where $\text{GMG}(\vec{b})$ means applying some μ -cycle with some number of pre and post smoothing steps.

- 5) Domain Decomposition

6.1 Domain Decomposition

Here, we will focus on preconditioners for solving a standard second-order discretization of the elliptic PDE

$$\begin{aligned} -\nabla \cdot (a(\vec{x}) \nabla u) &= f(\vec{x}) & \vec{x} &\in \Omega \\ u &= 0 & \vec{x} &\in \partial\Omega \end{aligned}$$

we will write the linear system as $A\vec{x} = \vec{b}$

This linear system $A\vec{x} = \vec{b}$ will be symmetric positive definite if $a(\vec{x})$ has some appropriate conditions. If we let $a(\vec{x}) = 1$, then

$$-\nabla \cdot (a(\vec{x})\nabla u) = \nabla \cdot (\nabla u) = -\Delta u = f$$

So, if you want, you can think of our PDE of interest to be the Poisson equation

$$\begin{aligned} -\Delta u &= f & \vec{x} &\in \Omega \\ u &= 0 & \vec{x} &\in \partial\Omega \end{aligned}$$

Domain Decomposition (DD) can be thought of as being a generalization of a block-Jacobi preconditioner. The block Jacobi precondition partitions the index set $I = \{1, \dots, n\}$ as

$$I = I_1 \cup I_2 \cup \dots \cup I_M$$

where each of the I_i 's are mutually disjoint. Then, the block Jacobi preconditioner is

$$P = (p_{ij})_{i,j=1,\dots,n}$$

with

$$p_{ij} = \begin{cases} a_{ij} & \text{if } i \text{ and } j \text{ are in the same index set} \\ 0 & \text{Otherwise.} \end{cases}$$

$$A = \begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ 0 & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix} \quad \begin{aligned} I_1 &= \{1, 2\} \\ I_2 &= \{3, 4\} \\ I_3 &= \{5, 6, 7\} \end{aligned}$$

$$\Rightarrow P = \begin{bmatrix} \begin{Bmatrix} 2 & -1 \\ -1 & 2 \end{Bmatrix} & & & 0 \\ & \begin{Bmatrix} 2 & -1 \\ -1 & 2 \end{Bmatrix} & & \\ & & \begin{Bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{Bmatrix} & \\ 0 & & & \end{bmatrix}$$

A major benefit of this preconditioner is that its inverse can be trivially applied in parallel. Unfortunately it isn't very useful for large n as it doesn't significantly reduce the number of CG iterations.

6.1.1 Discretization Coupling

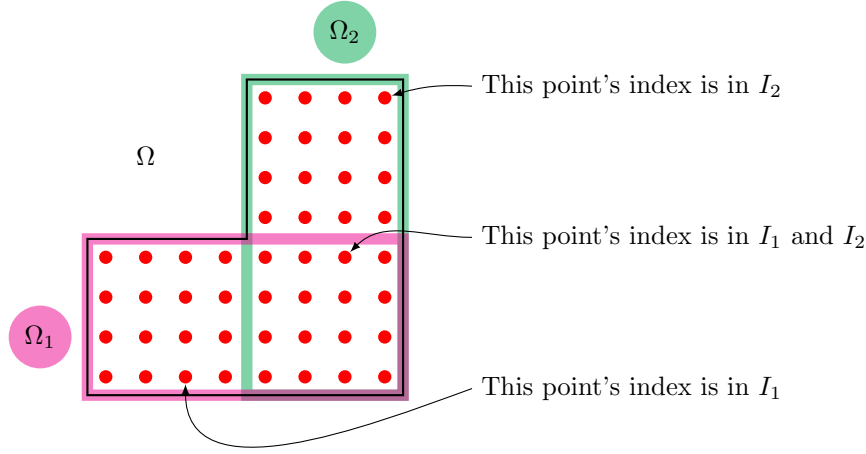
The problem is that elliptic PDEs, like the one we are solving, couple all points $\vec{x} \in \Omega$. That is if f is changed in some small region of Ω , the u changes everywhere in Ω . However, if $f(x_i)$ is changed, only part of $(P^{-1}\vec{f})$ will change. For discretizations of elliptic PDEs, it is important that the preconditioner couples all of the discretization points. One way to do this coupling is with DD. Let's write our PDE as

$$\begin{aligned} Lu &= f & \vec{x} &\in \Omega \\ u &= 0 & \vec{x} &\in \partial\Omega \end{aligned}$$

If you want, think of $L = -\Delta$. We partition Ω as $\Omega = \Omega_1 \cup \Omega_2$ with overlap. That is, $\Omega_1 \cap \Omega_2 \neq \emptyset$. Upon discretization, the points in Ω_1 are in some index set I_1 , and the points in Ω_2 are in some index set I_2 . Letting n_1 and n_2 be the number of points in Ω_1 and Ω_2 respectively, we have

$$n_1 + n_2 > n$$

where n is the total number of discretization points.



We can describe DD at the continuous level, or at the discretize level. To be applied at the discrete level, we require a restriction and prolongation operators. For each I_i , we define a $n \times n_i$ rectangular matrix R_i^T such that

$$(R_i^T \vec{x})_k = \begin{cases} x_k & \text{if } k \in I_i \\ 0 & \text{if } k \notin I_i \end{cases} \quad k = 1, \dots, n$$

Ex

If $I_1 = \{1, 2, 3, 4\}$ and $I_2 = \{4, 5, 6\}$, then

$$R_1^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad R_2^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, the matrices R_1 and R_2 are restriction operations that remove all entries from \vec{x} except those indexed in I_i . For example

$$R_1 \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad R_2 \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_6 \end{bmatrix} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

Then, the local subdomain matrices

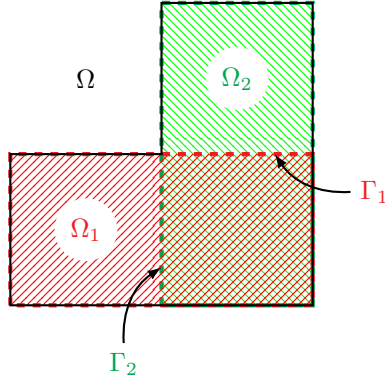
$$A_1 = R_1 A R_1^T, \quad A_2 = R_2 A R_2^T$$

$$A = \begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ 0 & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix} \quad \begin{array}{l} I_1 = \{1, 2\} \\ I_2 = \{3, 4\} \\ I_3 = \{5, 6, 7\} \end{array}$$

$$R_1 A R_1^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ 0 & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

The idea of DD is to solve discretizations of the PDE in each subdomain Ω_1 and Ω_2 . However, since boundary conditions are required on the boundaries of Ω_1 and Ω_2 , and part of these boundaries are in the interior of Ω .

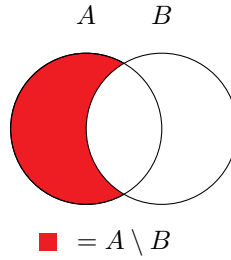


Letting Γ_1 and Γ_2 be the boundaries of Ω_1 and Ω_2 that are in the interior of Ω , the DD method starts with some u^0 and iterates by solving

$$\begin{cases} Lu_1^{k+1} = f & \vec{x} \in \Omega_1 \\ u_1^{k+1} = u_2^k \Big|_{\Gamma_1} & \vec{x} \in \Gamma_1 \\ u_1^{k+1} = 0 & \vec{x} \in \partial\Omega_1 \setminus \Gamma_1 \end{cases}$$

$$\begin{cases} Lu_2^{k+1} = f & \vec{x} \in \Omega_2 \\ u_2^{k+1} = u_1^{k+1} \Big|_{\Gamma_2} & \vec{x} \in \Gamma_2 \\ u_2^{k+1} = 0 & \vec{x} \in \partial\Omega_2 \setminus \Gamma_2 \end{cases}$$

Note the notation



With some conditions on L (coercivity), this iterative method converges in some norm to the exact solution. The convergence is geometric, meaning that

$$||u - u^k|| \leq \rho^k ||u - u_0||$$

where $\rho < 1$ depends on the choice of Ω_1 and Ω_2

6.1.2 Additive and Multiplicative Schwarz

This DD method can be written in another form by using the residual. We will do this for 2 index sets partitioning $\{1, \dots, n\}$ with overlap. At iteration k , the residual is

$$\vec{r}^k = \vec{f} - A\vec{x}^k$$

and the error satisfies

$$\vec{x} = \vec{x}^k + \vec{e}^k$$

Multiply both sides by A ,

$$\begin{aligned} A\vec{x} &= A\vec{x}^k + A\vec{e}^k \\ \Rightarrow \vec{f} &= A\vec{x}^k + A\vec{e}^k \\ \Rightarrow A\vec{e}^k &= \vec{f} - A\vec{x}^k \\ \Rightarrow A\vec{e}^k &= \vec{r}^k \Rightarrow \vec{e}^k = A^{-1}\vec{r}^k \end{aligned}$$

To update the solution at indices in I_1 , we need to restrict the residual to I_1 . Once we've solved for the error on I_1 , we prolong the error back to all the indices. We then repeat for I_2 .

$$\begin{aligned} \vec{x}^{k+1/2} &= \vec{x}^k + R_1^T A_1^{-1} R_1 (\vec{f} - A\vec{x}^k) \\ \vec{x}^{k+1} &= \vec{x}^{k+1/2} + R_2^T A_2^{-1} R_2 (\vec{f} - A\vec{x}^{k+1/2}) \end{aligned}$$

Letting

$$P_i := R_i^T A_i^{-1} R_i A, \quad i = 1, 2$$

we have

$$\begin{aligned} \vec{x}^{k+1/2} &= (I - P_1)\vec{x}^k + R_1^T A_1^{-1} R_1 \vec{f}, \\ \vec{x}^{k+1} &= (I - P_2)\vec{x}^{k+1/2} + R_2^T A_2^{-1} R_2 \vec{f} \end{aligned}$$

Therefore, the convergence of this method depends on the eigenvalues of the iteration matrix. For this algorithm, it can be found as follows.

$$\begin{aligned}\vec{x}^{k+1} &= (I - P_2) \left((I - P_1) \vec{x}^k + R_1^T A_1^{-1} R_1 \vec{f} \right) + R_2^T A_2^{-1} R_2 \vec{f} \\ &= (I - P_2)(I - P_1) \vec{x}^k + \vec{g}\end{aligned}$$

where

$$\begin{aligned}\vec{g} &= (I - P_2) R_1^T A_1^{-1} R_1 \vec{f} + R_2^T A_2^{-1} R_2 \vec{f} \\ \Rightarrow \vec{x}^{k+1} &= M \vec{x}^k + \vec{g}\end{aligned}$$

which is the iteration matrix.

Since the iteration matrix depends on a product, this method is called a *Multiplicative Schwarz iteration*.

Instead, we can use the last iterate \vec{x}^k in both subdomains. That is,

$$\begin{aligned}\vec{x}^{k+1/2} &= \vec{x}^k + R_1^T A_1^{-1} R_1 (\vec{f} - A \vec{x}^k) \\ \vec{x}^{k+1} &= \vec{x}^{k+1/2} + R_2^T A_2^{-1} R_2 (\vec{f} - A \vec{x}^k)\end{aligned}$$

Then, applying A_1^{-1} and A_2^{-1} can be done in parallel rather than sequentially. Putting the iterates together

$$\vec{x}^{k+1} = \vec{x}^k + (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2) (\vec{f} - A \vec{x}^k)$$

Therefore, the iteration matrix is

$$\begin{aligned}I - (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2) A \\ = I - P_1 - P_2\end{aligned}$$

and we call this method an *Additive Schwarz method*.

There is an alternative way to think of additive Schwarz. Assuming we are at a fixed point \vec{x} , we have

$$\begin{aligned}\vec{x} &= \vec{x} + (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2) (\vec{f} - A \vec{x}) \\ \Rightarrow (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2) A \vec{x} &= (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2) \vec{f}.\end{aligned}$$

Which is simply a preconditioned version of $A\vec{x} = \vec{f}$ with the preconditioner

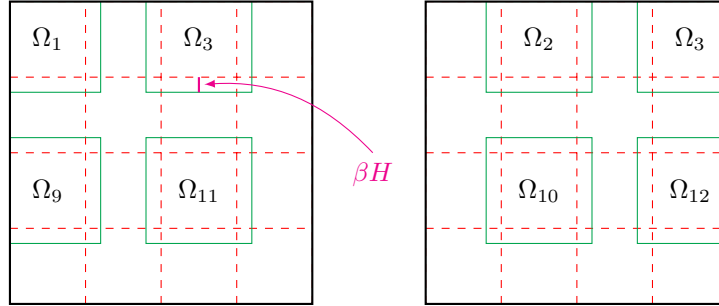
$$P^{-1} = R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2$$

A problem with DD as presented is that its convergence slows down when we have more than 2 processors (i.e. 2 subdomains). The reason is that not all the domains are talking to one another. As is, only neighboring domains communicate.

6.1.3 Two-Scale Domain Decomposition

One solution is to use a two-scale DD method. That is, one grid size h is used to solve problems on subdomains, and another grid with grid size H couples the subdomains. This gives us 2 more parameters, the grid sizes, that play a role in the multi-level DD method.

Given an elliptic PDE in \mathbb{R}^2 , we construct subdomains as follows:



An additive Schwarz preconditioner is

$$P^{-1} = M_{AS,1} = \sum_{i=1}^{16} R_i^T A_i^{-1} R_i$$

However, not all subdomains are coupled. If we define R_H^T to be an interpolation map as in GMG from the coarse grid to the fine grid. Then R_H is a restriction map. Then, a two-grid additive Schwarz preconditioner is

$$P^{-1} = M_{AS,2} = R_H^T A_H^{-1} R_H + \sum_{i=1}^{16} R_i^T A_i^{-1} R_i$$

and

$$\text{cond}(M_{AS,2}A) \leq C(1 + \beta^{-1})$$

Where C is independent of h and H .

References

- [1] William Briggs, Van Henson, and Steve McCormick. *A Multigrid Tutorial, 2nd Edition*. Jan. 2000. ISBN: 978-0-89871-462-3.
- [2] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. “A randomized algorithm for the decomposition of matrices”. In: *Applied and Computational Harmonic Analysis* 30.1 (2011), pp. 47–68. ISSN: 1063-5203. DOI: <https://doi.org/10.1016/j.acha.2010.02.003>. URL: <http://www.sciencedirect.com/science/article/pii/S1063520310000242>.
- [3] Jonathan Richard Shewchuk et al. *An introduction to the conjugate gradient method without the agonizing pain*. 1994.
- [4] M. Udell and A. Townsend. “Why Are Big Data Matrices Approximately Low Rank?” In: *SIAM Journal on Mathematics of Data Science* 1.1 (2019), pp. 144–160. DOI: 10.1137/18M1183480. eprint: <https://doi.org/10.1137/18M1183480>. URL: <https://doi.org/10.1137/18M1183480>.