Introduction to Iterative Methods to solve linear systems

A good reference - We will follow some of their presentation

Matrix Computations, Gene H. Golub and Charles F. Van Loan, The John's Hopkins Press 1989

$A x = b$

Notation - Cap. is a matrix
vector is lowercase
terms lower case with subscripts
Scalars - greek

Jacobi and Gauss-Seidel iteration - quite obvious -

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{23} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad \begin{matrix} \text{---(1)} \\ \text{---(2)} \\ \text{---(3)} \end{matrix}$$

from (1) $x_1 = (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11}$
from (2) $x_2 = (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22}$
from (3) $x_3 = (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}$

Suppose you have an current x - Lets call it $x^{(k)}$
Then an improved one, $x^{(k+1)}$ is

$x_1^{(k+1)} = (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11}$
$x_2^{(k+1)} = (b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)})/a_{22}$
$x_3^{(k+1)} = (b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)})/a_{33}$

This is the Jacobi iteration - for $A_{n\times n}$

given $x^{(0)}$

$k=0$

converged = false

While converged = false

Do $i=1$ to n

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

end {do}

$k = k+1$

If $||x^{(k)} - x^{(k+1)}|| < \varepsilon$ then set converged = true

converged = true

end {if}

end {while}

Gauss - Seidel iteration - Same form as Jacobi - just always use the most current values

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

To be able to present these methods a bit more formally - useful for analysis and comparisons of methods - we consider various forms of matrix decomposition.

(We saw a multiplicitive one with direct solves - expensive to compute!

Will start here with one this is trivial to get.

Sum Decomposition

$A = L + D + U$ L - Lower triang., D - diag., U - upper tri.

$$L = \begin{bmatrix} 0 & & & & \\ a_{21} & 0 & & & \\ a_{31} & a_{32} & 0 & & \\ \vdots & & & \ddots & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & 0 \end{bmatrix}, \quad D = \begin{bmatrix} a_{11} & & & & \\ & a_{22} & & 0 & \\ & & a_{33} & & \\ & 0 & & \ddots & \\ & & & & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ & 0 & a_{23} & \cdots & a_{2n} \\ & & \ddots & & \vdots \\ & 0 & & & a_{n-1,n} \\ & & & & 0 \end{bmatrix}$$

With this notation –

Jacobi Iteration

$$M_J x^{(k+1)} = N_J x^{(k)} + b$$

$$M_J = D, \quad N_J = -(L+U)$$

Gauss - Seidel

$$M_G x^{(k+1)} = N_G x^{(k)} + b$$

$$M_G = (D+L), \quad N_G = -U$$

These are members of a class of iterative methods

$$M x^{(k+1)} = N x^{(k)} + b$$

where $A = M(-N)$

To be a useful iterative method
M must be "easy" to "solve"

The convergence of $x = A^{-1}b$ depends on the the eigenvalues of $M^{-1}N$

In particular one wants to consider the spectral radius of the $M^{-1}N$

The spectral radius of an nxn matrix G is
$$\rho(G) = \max\{|\lambda|, \lambda \in \lambda(G)\}$$
where $\lambda(G)$ are the eigenvalues of G

If M is non-singular (M^{-1} exists) and $\rho(M^{-1}N) < 1$ the iterative method converges for any $x^{(0)}$

It can be shown that:
Gauss-Seidel converges for positive definate A.
That is not to say it converges quickly, or even converges with finite precision math.

If $\rho(M^{-1}N)$ is close to 1 convergence will be very slow - Procedure can be modified to speed the convergence

Successive Over-Relaxation
introduce relaxation factor ω (a scalar)
The modified core expression is now

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

$$M_\omega x^{(k+1)} = N_\omega x^{(k)} + \omega b$$

$$M_\omega = D + \omega L \quad , \quad N_\omega = (1-\omega)D - \omega U$$

$$\omega \in]0, 2[$$

Symmetric Successive Over-Relaxation (SSOR)

$G = M^{-1} N$ is not symmetric for any A

Consider a backwards SOR step

For $i = n$ to 1 by -1

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

Backwards SOR

$$\tilde{M}_\omega = D + \omega U, \quad \tilde{N}_\omega = (1-\omega) D - \omega L$$

With a symmetric A (the case where we would care most about "symmetry" of the iteration) we have $U = L^T$

Thus $\tilde{M}_\omega = M_\omega^T$, $\tilde{N}_\omega = N_\omega^T$

The SOR step is then two ½ steps

$$M_\omega x^{(k+1/2)} = N_\omega x^{(k)} + \omega b$$
$$M_\omega^T x^{(k+1)} = N_\omega^T x^{(k+1/2)} + \omega b$$

There are additional methods to potentially accelerate convergence

Chebyshev - semi iterative methods that does fitting of polynomial basis.

Descent Methods

§ 10.2 of Golub and Van Loan - Matrix Computations

Consider minimization of the following quadratic form

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

$b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$ — A - positive definite, A - symmetric

at minimum we have

$$\frac{\partial \phi}{\partial x} = 0 = Ax - b \Rightarrow \text{our original problem } Ax = b$$

note at minimum $x = A^{-1}b$

$$\phi(x)_{min} = \frac{1}{2}(A^{-1}b)^T A A^{-1} b - (A^{-1}b)^T b$$

$$= \frac{1}{2} b^T A^{-T} A A^{-1} b - b^T A^{-1} b$$

$$= -\frac{1}{2} b^T A^{-1} b$$

Need a method to find minimum -

Given a current point on the "surface" for x_c, we would like to move is a direction that takes us toward the minimum.

One nice sounding option is the direction with largest decrease - that defines The method of steepest descent where we go in direction of largest negative gradient

$$-\nabla \phi(x) = -\left[2\left(\frac{1}{2}Ax\right) - b\right] = b - Ax$$

This is also equal to the "residual vector" when we plug in a value for $x = x_c$

$$r_c = b - Ax_c$$

(of course if we were at the minimum this is zero since we have $x = A^{-1}b$ and $r_c = 0$

Lets assume for moment that
- We are looking for a minimum and x_c is not it
- We have a "quadratic" positive definite A

In that case there is an α such that

$$\phi(x_c + \alpha r_c) \leq \phi(x_c)$$

Simply says that we can move in the direction of the negative gradient and have function decrease.

The goal is to select α to minimize $\phi(x_c + \alpha r_c)$

$$\phi(x_c + \alpha r_c) = \frac{1}{2}(x_c + \alpha r_c)^T A (x_c + \alpha r_c) - (x_c + \alpha r_c)^T b$$

$$= \frac{1}{2} x_c^T A x_c + x_c^T A \alpha r_c + \frac{1}{2}\alpha^2 r_c^T A r_c - x_c^T b - \alpha r_c^T b$$

For minimum we are looking for $\partial\phi / \partial\alpha = 0$

$$0 = 0 + x_c^T A r_c + \alpha r_c^T A r_c - r_c^T b$$

note $x_c^T A r_c = r_c^T A^T x_c = r_c^T A x_c$ (scalar^T = scalar, $A = A^T$)

$$0 = r_c^T (A^T x_c - b) + \alpha r_c^T A r_c$$

recall $r_c = b - Ax_c$

$$0 = -r_c^T r_c + \alpha r_c^T A r_c$$

$$\alpha = \frac{r_c^T r_c}{r_c^T A r_c}$$

This is the method of steepest descent

The iteration process:

$k=0, \quad x_0 = 0 \quad , r_0 = b$

while $\|r_k\| \neq 0$

$k = k+1$

$\alpha_k = r_{k-1}^T r_{k-1} \Big/ r_{k-1}^T A r_{k-1}$

$x_k = x_{k-1} + \alpha r_{k-1}$

$r_k = b - A x_k$

end while

This method converges if (recall that min is $-\frac{1}{2} b^T A^{-1} b$)

$$\phi(x_k) + \tfrac{1}{2} b^T A^{-1} b \leq \left(1 - \frac{1}{\kappa_2(A)}\right)\left(\phi(x_{k-1}) + \tfrac{1}{2} b^T A^{-1} b\right)$$

$$\kappa_2(A) = \frac{\lambda_1(A)}{\lambda_n(A)}$$

however if $\kappa_2(A)$ is large (such that $\left(1 - \frac{1}{\kappa_2(A)}\right) \approx 1$)
convergence is very slow - gets stuck going back and forth

This can happen in problems of interest.

Therefore we want more general search direction selections

$\Rightarrow$ Consider a set of directions - $\{p_1, p_2, p_3 \ldots\}$ that do not necessarily correspond to $\{r_0, r_1, r_2 \ldots\}$

(note shift of 1 in the subscripts)

Lets consider minimizing WRT these new (and yet to be defined) directions

For a particular direction we want to set $\partial \phi / \partial \alpha_k = 0$ for $\phi(x_{k-1} + \alpha_k p_k)$

This yields $$\alpha_k = \frac{p_k^T r_{k-1}}{p_k^T A p_k}$$

substituting in we have

$$\phi(x_{k-1} + \alpha_k p_k) = \phi(x_{k-1}) - \tfrac{1}{2}(p_k^T r_{k-1})^2 p_k^T A p_k$$

To ensure a reduction in ϕ we need to require p_k not be orthogonal to r_{k-1}. The resulting procedure:

$k=0, x_0 = 0, r_0 = b$

while $|r_k| > \varepsilon$ (small value - zero in the limit)

$k = k+1$

Choose a direction p_k such that $p_k^T r_{k-1} \neq 0$

$\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$

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

$r_k = b - A x_k$

end{while}

On to the selection of directions p_k —

A-Conjugate Search Directions

Consider: Set of linearly independent p_i with the property that each x_k is such that it yields $\min_{x \in \text{Span}(p_1, p_2, \ldots p_k)} \phi(x)$ ____ (1)

Under the assumption of precise calculations (which we never have) this would ensure convergence to the correct solution in a finite number of steps - n of them since this requirement forces $Ax_n = b$

To find the p_k for this case requires solving the minimization problem defined by eq. (1). For a direction p_k we want to solve

$$\min_{\alpha} \phi(x_{k-1} + \alpha p_k)$$

which leads to the solution of the k-dimensional problem of eq (1). To do this lets define the matrix of search directions -

$$P_k = [p_1, p_2, \cdots p_k] \in \mathbb{R}^{n \times k}$$

if $x \in \text{range}(P_k)$ then $x = P_{k-1} y + \alpha p_k$ for some $y \in \mathbb{R}^{k-1}$ and $\alpha \in \mathbb{R}$, with that it can be shown that

$$\phi(x) = \phi(P_{k-1} y) + \alpha y^T P_{k-1}^T A p_k + \frac{\alpha^2}{2} p_k^T A p_k - \alpha p_k^T b$$

Note that the term $\alpha y^T P_{k-1}^T A p_k$ couples everything making find min. $\phi(x)$ nasty

We can uncouple if we were to require

$P_{k-1}^T A p_k = 0$ ⇐ says p_k is A-conjugate to P_{k-1}

Under these conditions it can be shown that

$$\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$$

With this the resulting procedure is

$k=0$; $x_0=0$, $r_0=b$
while $|r_k| > \epsilon$
  $k = k+1$
  Choose $p_k \in \text{span}(Ap_1, Ap_2 \cdots Ap_{k-1})$ such that $p_k^T r_{k-1} \neq 0$
  (that is A-conjugate to P_{k-1})

$$\alpha_k = p_k^T r_{k-1} / p_k^T A p_k$$

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

end while

We still need at least one effective methods to select the set of A-Conjugate directions. Some properties used to construct useful directions.

For $k \geq 2$ the directions p_k satisfy

$$p_k = r_{k-1} - A P_{k-1} z_{k-1}$$

where z_{k-1} solves $\min_z \| r_{k-1} - A P_{k-1} z \|_2$
(This is given without proof.)

After k iterations of the procedure we have (with some manipulation)

$$r_j = r_{j-1} - \alpha A p_j$$

$$p_j^T r_j = 0$$

$$\text{Span}\{p_1, p_2, \dots p_j\} = \text{Span}\{r_0, r_1, \dots r_{j-1}\} = \text{Span}\{b, Ab, \dots A^{j-1}b\}$$

note $A^{j-1} = \overbrace{AA\cdots A}^{j-1 \text{ of them}}$

It can also be shown that the residuals $\{r_0, r_1, \dots r_{k-1}\}$ are mutually orthogonal

The last ingredient is showing that p_k can be written as a linear combination of p_{k-1} and r_{k-1}

Using the result that $p_k^T A p_k = 0$ and $p_{k-1}^T r_{k-1} = 0$ we have

$$p_k = r_{k-1} + \beta_k p_{k-1} \quad \text{with} \quad \beta_k = \frac{p_{k-1}^T A r_{A-1}}{p_{k-1}^T A p_{k-1}}$$

and

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

This gives the 1st version of a concrete Conjugate Gradient Method

$k=0,\ x_0=0,\ r_0=b$
while $|r_k| > \epsilon$
  $k = k+1$
  if $k=1$ then
    $p_1 = r_0$
  else
    $\beta_k = -p_{k-1}^T A r_{k-1} / p_{k-1}^T A p_{k-1}$
    $p_k = r_{k-1} + \beta_k p_{k-1}$
  end {if, then, else}
  $\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$
  $x_k = x_{k-1} + \alpha p_k$
  $r_k = b - A x_k$
end {while}

note for later - three Matrix-vector products
$A r_{k-1}$
$A p_{k-1}$
$A x_k$

This can be made more computationally efficient by using identities that replace vector^T-matrix-vector products with vector^T-vector products

for example from $\alpha_k = \dfrac{r_{k-1}^T r_{k-1}}{p_k^T A p_k}$ we have

$$\alpha_{k-1} p_{k-1}^T A p_{k-1} = r_{k-2}^T r_{k-2}$$

Also
$$r_{k-1}^T r_{k-1} = -\alpha_{k-1} r_{k-1}^T A p_{k-1}$$

This yields the Conjugate Gradient algorithm for $A \in \mathbb{R}^{n \times n}$, a symmetric positive matrix and $b \in \mathbb{R}^n$ we want to solve $Ax = b$

$k = 0$, $x_0 = 0$, $r_0 = b$
while $|r_k| > \epsilon$
  $k = k + 1$
  if $k = 1$ then
    $p_1 = r_0$
  else
    $\beta_k = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$
    $p_k = r_{k-1} + \beta_k p_{k-1}$

Note- only 1 matrix vector product- Ap_k → you would not do Ap_k twice or $r_{k-1} r_{k-1}$ twice

  end {if, then, else}
  $\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$
  $x_k = x_{k-1} + \alpha_k p_k$
  $r_k = r_{k-1} - \alpha_k A p_k$
end {while}

For real problems preconditioning is needed to make the system diagonally dominate

Preconditioned Conjugate Gradient
Transform original system $Ax = b$ into preconditioned one, $\tilde{A}\tilde{x} = \tilde{b}$ which is solved by CG method.

Consider

$\tilde{A} = C^{-1} A C^{-1}$, $\tilde{x} = C x$, $\tilde{b} = C^{-1} b$

C - symmetric, pos. def.

note: $C^{-1} A C^{-1} C x = C^{-1} b \Rightarrow C^{-1} A x = C^{-1} b \Rightarrow Ax = b$

$x = C^{-1} \tilde{x}$

Want $\tilde{A}$ to be better than A to solve and, as will be shown, want C^2 must be "easy to get"

Look at CG on transformed system

$k=0$, $\tilde{x}_0 = 0$, $\tilde{r}_0 = \tilde{b}$

while $|r_k| > \epsilon$

$k = k+1$

if $k=1$ then

$\tilde{p}_1 = \tilde{r}_0$

else

$\beta_k = \tilde{r}_{k-1}^T \tilde{r}_{k-1} / \tilde{r}_{k-2}^T \tilde{r}_{k-2}$

$\tilde{p}_k = \tilde{r}_{k-1} + \beta_k \tilde{p}_{k-1}$

end {if, then, else}

$\alpha_k = \tilde{r}_{k-1}^T r_{k-1} / \tilde{p}_k^T C^{-1} A C^{-1} \tilde{p}_k$

$\tilde{x}_k = \tilde{x}_{k-1} + \alpha_{k-1} \tilde{p}_k$

$\tilde{r}_k = \tilde{r}_{k-1} - \alpha C^{-1} A C^{-1} \tilde{p}_k$

end while

To get to a form we want we will use:

$\tilde{x}_k = C x_k$

$\tilde{r}_k = C^{-1} r_k$

$\tilde{b} = C^{-1} b$

$\tilde{p}_k = C p_k$

Using these we can write a version of CG that looks rather hasty

$k=0$, $Cx_0 = 0$, $C^{-1}r_0 = C^{-1}b$
while $C^{-1}r_k > \epsilon$
$k = k+1$
if $k=1$ then
$Cp_1 = C^{-1}r_0$
else
$\beta_k = (C^{-1}r_{k-1})^T(C^{-1}r_{k-1}) / (C^{-1}r_{k-2})^T(C^{-1}r_{k-2})$
$Cp_k = C^{-1}r_{k-1} + \beta_k Cp_{k-1}$
end {if, then, else}
$\alpha_k = (C^{-1}r_{k-1})^T(C^{-1}r_{k-1}) / (Cp_k)^T C^{-1}AC^{-1}(Cp_k)$
$Cx_k = Cx_{k-1} + \alpha_k Cp_k$
$C^{-1}r_k = C^{-1}r_{k-1} - \alpha_k (C^{-1}AC^{-1})Cp_k$

Define $M = C^2$ (will be symmetric pos. def given C pos def)
let z_k be the solution to $Mz_k = r_k$
that is $z_k = M^{-1}r_k = (C^2)^{-1}r_k$
thus $Cp_1 = C^{-1}r_0 \Rightarrow C^{-1}Cp_1 = C^{-1}C^{-1}r_0 = M^{-1}r_0 = z_0$)

$$(C^{-1}r_k)^T(C^{-1}r_k) = r_k^T C^{-T}C^{-1}r_k = r_k^T M^{-1}r_k = r_k^T z_k$$

$$C(C^{-1}r_k) = C\left(C^{-1}r_{k-1} - \alpha_k (C^{-1}AC^{-1})Cp_k\right)$$

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

Using these we have the Preconditioned Conjugate Gradient

Preconditioned Conjugate Gradient

$k=0,\ x_0=0,\ r_0=b$

while $|r_k| > \epsilon$

Solve $M z_k = r_k$ ← Need this to be "easy" to solve

$k=k+1$

if $k=1$

$p_1 = z_0$

else

$\beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$

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

end {if, then, else}

$\alpha_k = r_{k-1}^T z_{k-1} / p_k^T A p_k$

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

$r_k = r_k - \alpha_k A p_k$

end {while}

Some properties

$r_j^T M^{-1} r_i = 0 \quad i \neq j$

$p_j^T (C^{-1} A C^{-1}) p_i = 0 \quad i \neq j$ } can be shown

$r_{k-2}^T z_{k-2} = z_{k-2}^T M z_{k-2}$ is never zero with pos. def. M

There are a number of approaches to define preconditioners

Incomplete Cholesky is a good one – but quite expensive

Various forms of diag. and "physics" based methods exist.

Multigrid Solution Techniques

"A Multigrid Tutorial", W.L. Briggs, V.E. Henson, S.F. McCormick, SIAM, 2000.

We will focus on just the basics looking at a 1-D case

Want to solve $Ax = b$

given an approximation to x, denoted v

the error and residual are

$$e = x - v$$

$$r = b - Av \quad , \quad Av = b - r$$

note: $Ax = b$

$$-(Av = b - r)$$

$$A(x - v) = r \Rightarrow Ae = r$$

A multigrid method will build on a selected basic iterative method that will be used to perform "relaxation iterations on different levels of the "mesh".

Consider the simplest method - Jacobi

with $A = L + D + U$, D-diag., L-Lower tri, U-upper tri.

The basic form:

$$v^{(i+1)} = -D^{-1}(L+U)v^{(i)} + D^{-1}b = R_J v^{(i)} + D^{-1}b$$

With relaxation

$$v^{(i+1)} = [(1-\omega)I + \omega R_J]v^{(i)} + \omega D^{-1}b = R_\omega v^{(i)} + \omega D^{-1}b$$

To see the basics of how a multigrid method works we will look at an example of $x=0$ (the desired exact soln) on a mesh of piecewise liner elements (n-elements; n+1 nodes)

For our discretization we will use a set of Fourier modes and we will note very different rates of convergence to the $x=0$ value based on the number of $\frac{1}{2}$ sin waves we start with. We see the more sin waves the faster the convergence

$\Rightarrow$ The higher the frequency of the error (WRT the mesh size) the faster the convergence.

An analysis will give complete insight into this. Note our iteration methods is:

$$v^{(i+1)} = Pv^{(i)} + g$$

note: $x = Px + g \Leftarrow$ converged soln.

$$(x - v^{(i+1)}) = P(x - v^{(i)}) + g - g \Rightarrow e^{(i+1)} = Pe^{(i)}$$

$$e^{(1)} = Pe^{(0)}$$
$$e^{(2)} = Pe^{(1)} = P(Pe^{(0)}) = P^2e^{(0)}$$
$$e^{l} = P^l e^{(0)} \quad \text{after } l \text{ iterations}$$

It can be shown that (in appropriate norms)

$$\|e^l\| \leq \|P\|^l \|e^0\|$$

thus if $\|P\| < 1$ the method will converge

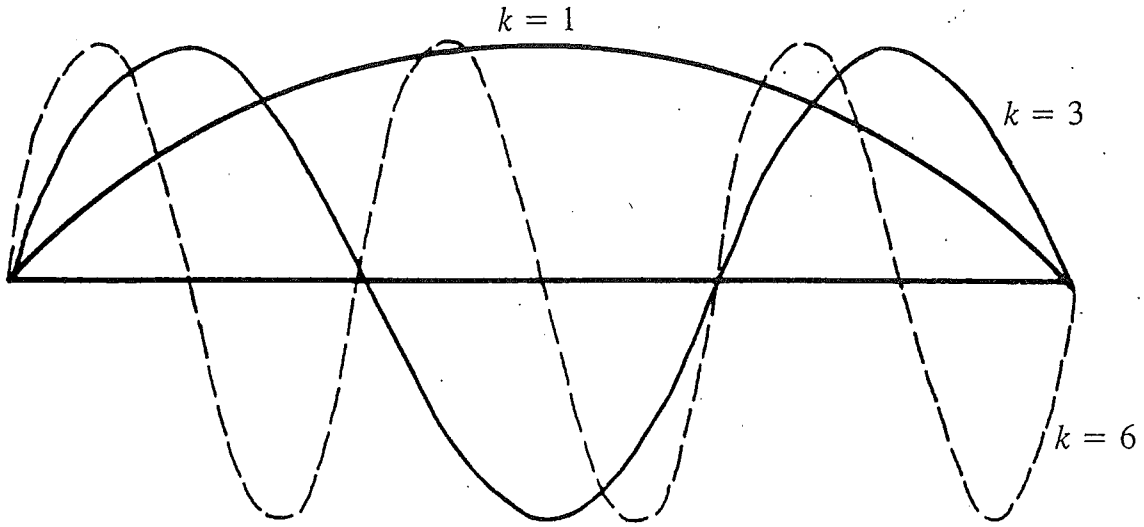
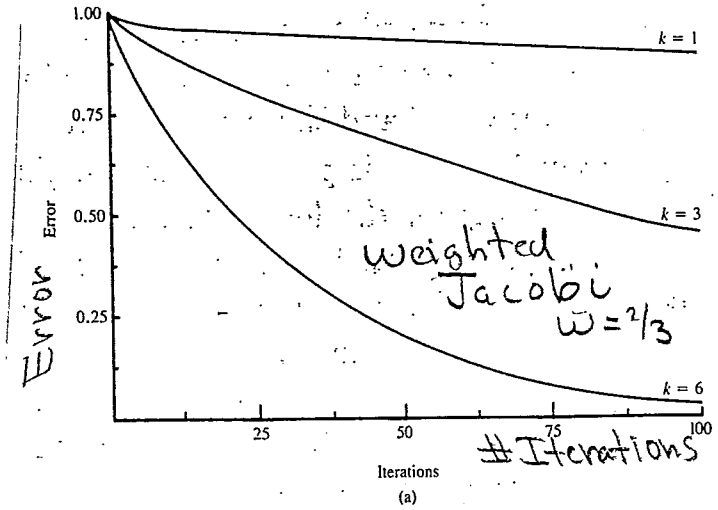


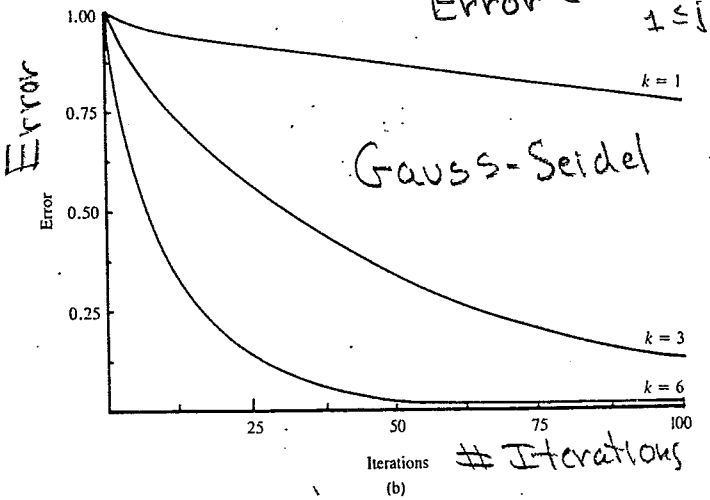
FIG. 4. The modes $v_j = \sin\left(\frac{jk\pi}{N}\right)$, $0 \le j \le N$, with wavenumbers $k = 1, 3, 6$. The kth mode consists of $k/2$ full sine waves on the interval.

$n = 64$

$k < n$



Error = $\max_{1 \le j \le N} |e_j|$



This is basically saying we need

$$\lim_{l \to \infty} \|P\|^l = 0$$

We have this if $\rho(P) < 1$, $\rho(P) = \max_i |\lambda_i(P)|$

spectral radius

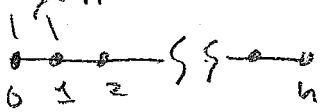
For the model 1-D problem we looked at

$$\lambda_k(P_w) = 1 - \frac{w}{2}\lambda_k(A)$$

and $\lambda_k(A) = 4\sin^2\left(\frac{k\pi}{2n}\right)$ $1 \le k \le n-1$

$$\lambda_k(P_w) = 1 - 2w\sin^2\left(\frac{k\pi}{2n}\right)$$

for unit length domain $h = \frac{1}{n}$



The lowest frequency (smoothness) mode dictates convergence look at λ_1

$$\lambda_1 = 1 - 2w\sin^2\left(\frac{\pi}{2n}\right) = 1 - 2w\sin^2\left(\frac{\pi h}{2}\right) \approx 1 - \frac{w\pi^2 h^2}{2}$$

(using $\sin\left(\frac{\pi h}{2}\right) \approx \frac{\pi h}{2}$ for small h)

As h gets smaller $\lambda \to 1$

The finer the mesh - the more difficult the convergence of the iterative method (when portions of the error have are "smooth" with respect to h.)

Additional analysis shows that w has little influence on convergence of the smooth modes However, it is important to getting convergence of higher modes

Example showing slow & fast convergence

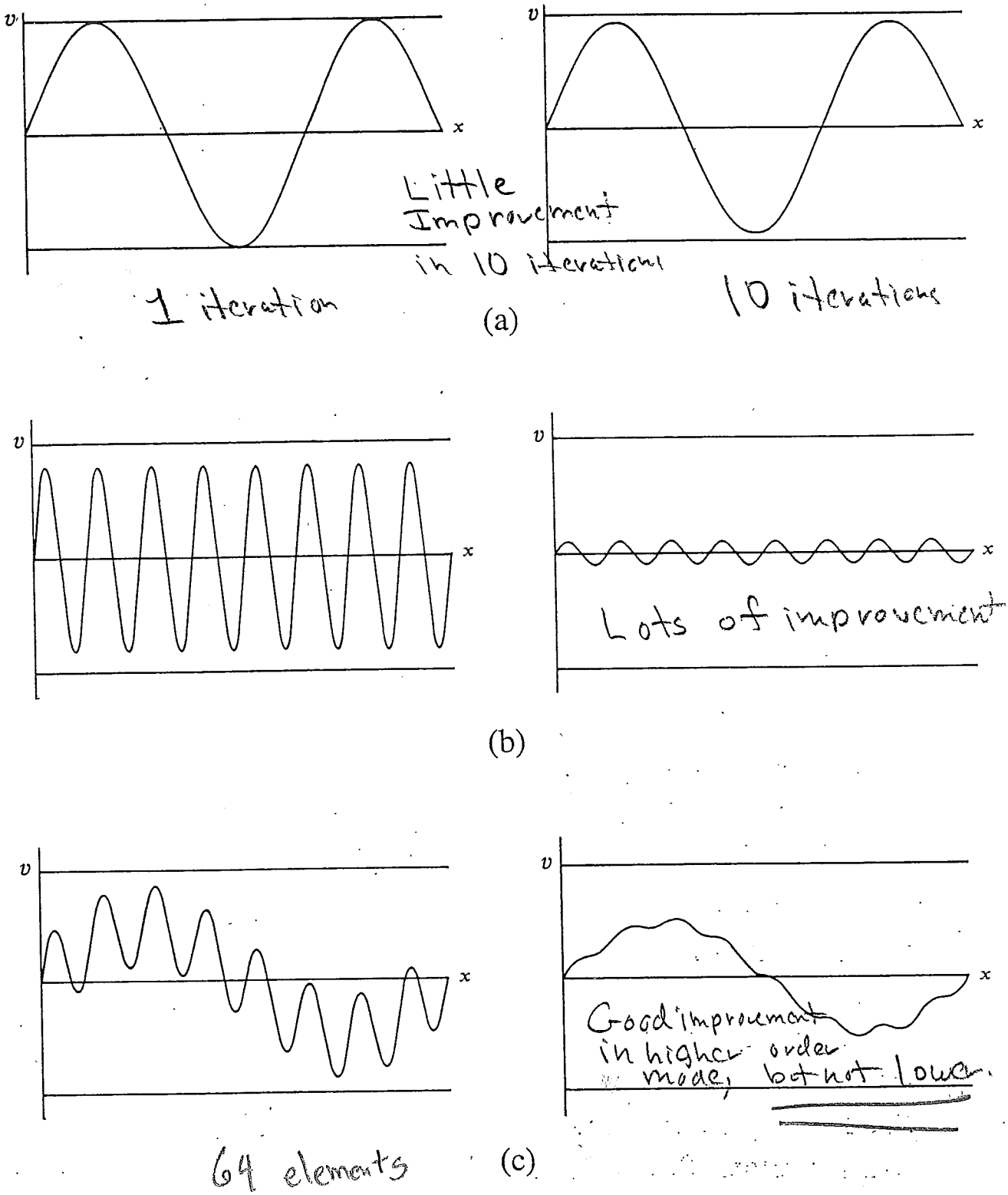


FIG. 11. The weighted Jacobi method with $\omega = 2/3$ applied to the one-dimensional model problem with $N = 64$ and with an initial guess consisting of (a) $\mathbf{w}_3$, (b) $\mathbf{w}_{16}$ and (c) a combination of $\mathbf{w}_2$ and $\mathbf{w}_{16}$. The figures show the approximation after one iteration (on the left) and after ten iterations (on the right).

The example hints at the fact that the frequency question is related to mesh size

Analysis indicates:

(A) The k^{th} mode for $1 \le k \le n/2$ on Ω^h (a mesh with element size h) becomes the k^{th} mode on Ω^{2h} (mesh with elements 2h - half as many elements). However, it is now a higher frequence WRT the element size.

(B) For $k > n/2$ there is an "aliasing phenomenon" such that the k^{th} mode becomes the $(n-k)^{th}$ mode on Ω^{2h} ⇐ The higher frequency modes on Ω^h become smoother modes on Ω^{2h}

Combining (A) and (B) with the fact that convergence is fast for "high" frequency modes indicates we want to iterate on multiple levels of the mesh -
Thus far it looks like we want to iterate first on the fine grid and to then move to coarser grids - That is one important aspect - but does not get it all because of interactions of modes, etc.

By using $Ae = r = b - Av$, we can iterate directly on the error term.

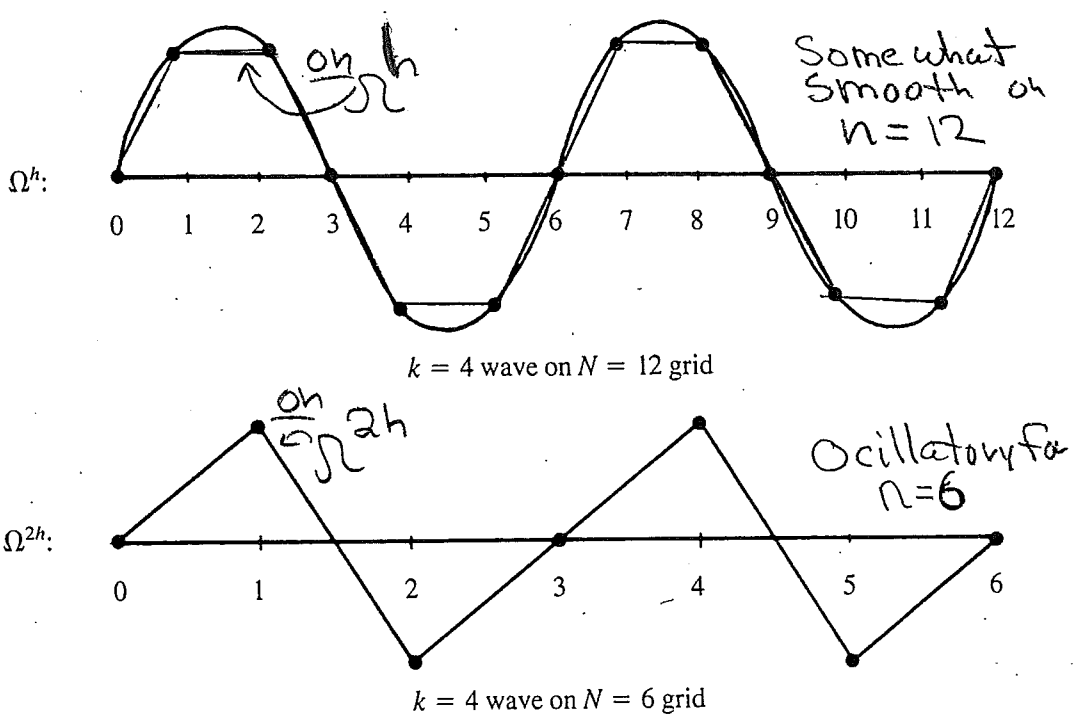


FIG. 14. A wave with wavenumber $k = 4$ on Ω^h $(N = 12)$ is projected onto Ω^{2h} $(N = 6)$. The coarse grid "sees" a wave with $k = 4$ which is more oscillatory on the coarse grid than on the fine grid.

Lets look at a process for doing this

Some Components

1. Start on a coarse grid, Ω^c, and "relax" to get a starting guess on a finer grid. Relax on that grid. Continue the process on finer grids until you get to Ω^h (finest grid) ⇐ By its self this will not do the job based on what we have seen

2. Coarse Grid Correction:
   Relax $Ax=b$ on Ω^h to get an approx. soln v^h
   Compute residual $r = b - Av^h$
   Relax residual eq. $Ae=r$ on Ω^{2h} to obtain e^{2h}
   Correct approximation on Ω^h using e^{2h}
   $v^h \leftarrow v^h + e^{2h}$
   (Can extend to more levels)

In both cases there are options WRT the transfer of solutions between grids

Prolongation – Transfer error, e^{2h}, from Ω^{2h} to Ω^h.

$I_{2h}^h v^{2h} = v^h$

An option
Simple linear interpolation

$$v_{2j}^h = v_j^{2h}$$

$$v_{2j+1}^h = \frac{1}{2}\left(v_j^{2h} + v_{j+1}^{2h}\right) \qquad 0 \le j \le \frac{n}{2} - 1$$

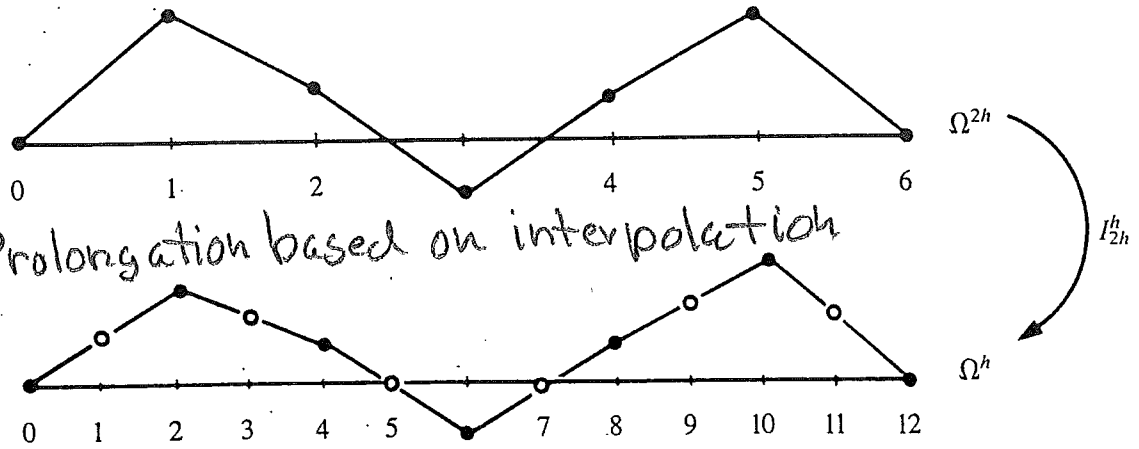


FIG. 15. Interpolation of a vector on the coarse grid Ω^{2h} to the fine grid Ω^{h}.

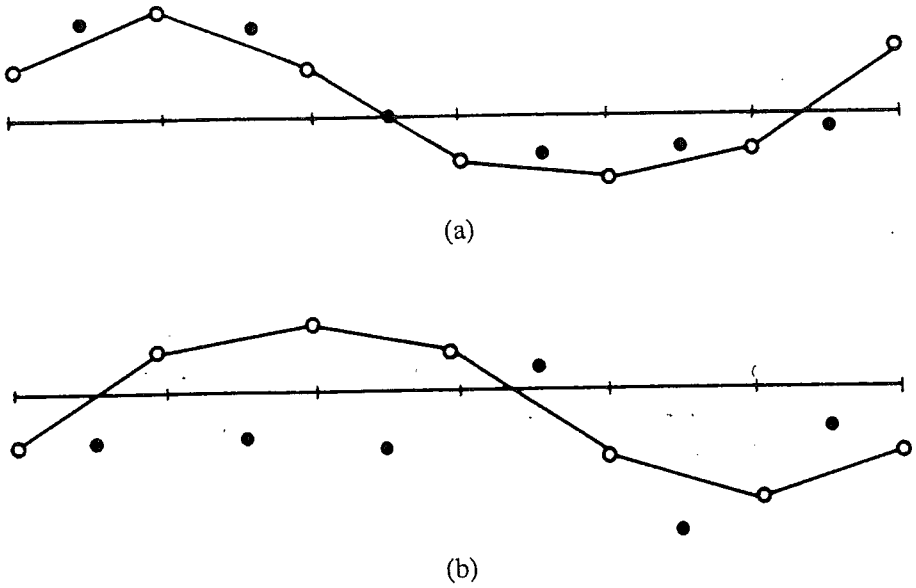


FIG. 16. (a) If the exact error on Ω^{h} (indicated by ○ and •) is smooth, an interpolant of the coarse grid approximation $\mathbf{e}^{2h}$ (indicated by ○) should give a good representation of the exact error. (b) If the exact error on Ω^{h} (indicated by ○ and •) is oscillatory, an interpolant of the coarse grid approximation $\mathbf{e}^{2h}$ (indicated by ○) may give a poor representation of the exact error.

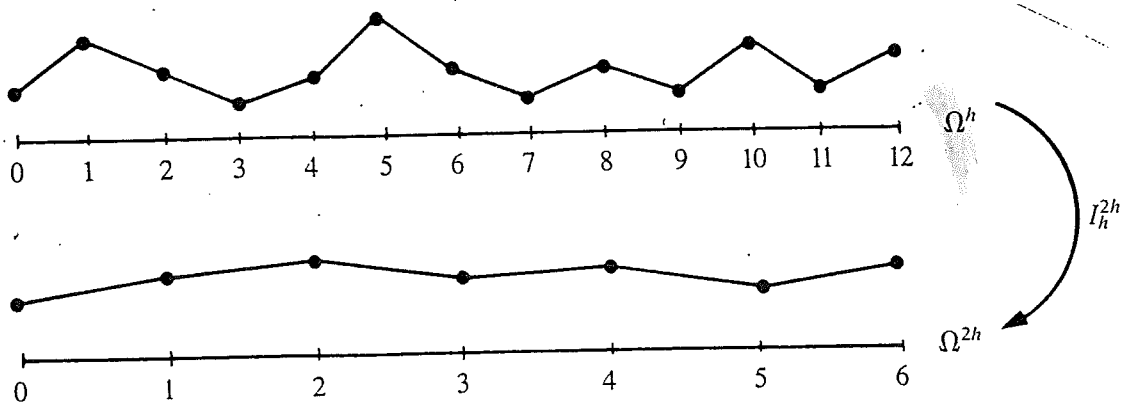
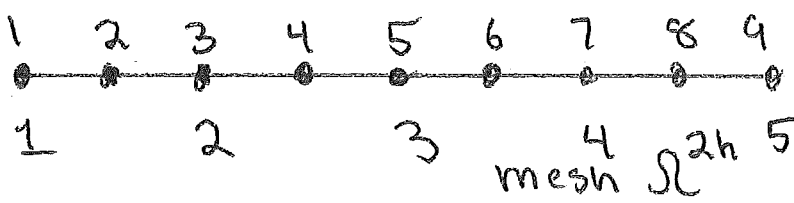


FIG. 17. Restriction by full weighting of a fine grid vector to the coarse grid.

Consider a specific case of $n=8$



Given BC - $x_1 = x_9 = 0$

$$I_{2h}^{h} v^{2h} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} v_1^{2h} \\ v_2^{2h} \\ v_3^{2h} \\ v_4^{2h} \end{Bmatrix} = \begin{Bmatrix} v_1^h \\ v_2^h \\ v_3^h \\ v_4^h \\ v_5^h \\ v_6^h \\ v_7^h \\ v_8^h \end{Bmatrix}$$

If the exact error is smooth, the coarse grid interpolation is a good approximation. If the exact error is oscillatory, the coarse grid interpolation is poor (part of the reason to go back and forth between levels)

Restriction - Transfers from Ω^h to Ω^{2h}

Obvious option - $v_j^{2h} = v_{2j}^h$ ⇐ use fine grid values

An alternative is a (full) weighted version - an option motivated by F.D. is

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

The full weighted operator for our $n=8$ case

$$I_h^{2h} v^h = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 2 & 1 \end{bmatrix} \begin{Bmatrix} v_2^h \\ v_3^h \\ v_4^h \\ v_5^h \\ v_6^h \\ v_7^h \\ v_8^h \end{Bmatrix} = \begin{Bmatrix} v_2^{2h} \\ v_3^{2h} \\ v_4^{2h} \end{Bmatrix} = v^{2h}$$

Note: With this Selection $I_{2h}^{h} = C\left(I_{h}^{2h}\right)^{T}$
(A property typically maintained)

Note the Prolongation and Restriction operators given are for a simple "linear" element case.

Much more complex for higher order cases, and on general meshes - Lots of papers addressing these issues - Two classes of approach -

Geometric MultiGrid - Define levels with respect to levels of mesh

etc

Algebraic Multigrid
$[K]_{n\times n}\{d\} = \{F\}$ → use algebra on this system to get coarser levels

For the geometric approach consider we have multiple levels of mesh - $\Omega^{h}, \Omega^{2h}, \Omega^{4h}, \ldots \Omega^{2^{m}h}$
$\Omega^{2^{0}h}, \Omega^{2^{1}h}, \Omega^{2^{2}h}, \ldots \Omega^{2^{m}h}$

We will have the matrix for the Ω^{h} case
$A^{h}x^{h} = b^{h}$

A specific option for the coarser meshes

$$A^{2^{p}h}x^{2^{p}h} = b^{2^{p}h} \qquad p = 1(1)m$$

is to use our Restriction and Prolongation operators

$$A^{2^{p}h} = I_{2^{p-1}h}^{2^{p}h} A^{2^{p-1}h} I_{2^{p}h}^{2^{p-1}h}, \quad I_{2^{p-1}h}^{2^{p}h} = C\left(I_{2^{p}h}^{2^{p-1}h}\right)^{T}$$

A two level multigrid

① Relax μ_1 times on $A^h v^h = b^h$ on Ω^h

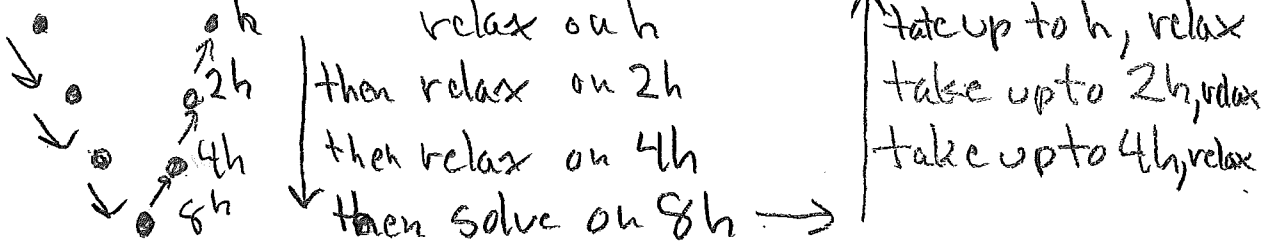
② Compute $r^{2h} = I_h^{2h}(b^h - A^h v^h)$
"Solve" $A^{2h} e^{2h} = r^{2h}$ on Ω^{2h}
Correct fine grid $v^h \leftarrow v^h + I_{2h}^h e^{2h}$

③ Relax μ_2 times on $A^h v^h = b^h$ on Ω^h

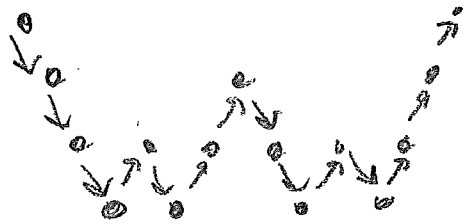
1. Converge high frequency on fine mesh
2. "Solve" for the error on the coarse grid and prolongate back to fine grid
3. Additional relaxation on fine grid to improve solution more

The course grid solution needs to be accurate for this to work well. Reducing mesh by one level will typically yield a system that is still too expensive to solve - In that case - add more levels -

for example V cycle



There are mor options



W cycle