Residual Correction

Suppose you have a linear solver: $S$ where

$$S(b) \approx A^{-1}b.$$ 

Residual correction methods try to overcome problems with the solver by using it iteratively.
Algorithmic Sketch

\[ x = x_0 \]
\[ r = b - Ax \]
\[ \text{while } \|r\| > \tau \text{ do} \]
\[ e_s = S(r) \]
\[ x \leftarrow x + e_s \]
\[ r = b - Ax \]
\[ \text{end while} \]

When \( S(b) = A^{-1}b \) you converge in one iteration because

\[ e_s = A^{-1}r = A^{-1}(b - Ax) = x^* - x = e. \]
Suppose $S$ is a single precision solve?

$$S(b) = U_s^{-1} L_s^{-1} b$$

where $L_s$ and $U_s$ are the single precision LU factors of $A$.

```
AS=single(A);
[LS, US]=lu(AS);
```

will do the job.
The rules and the facts

- **Rule:** computed residuals in full precision
- **Fact:** relative error from solve will be single precision at best
- **Fact:** cost of LU should be reduced by $1/8$
- **Fact + Rule:** you have to pay attention to the language-dependent conversions from single to double and back.
To compute $S(b)$ you need get prepared

- Convert $A$ to single to get $A_s$
- Compute the single precision LU factorization $L_sU_s = A$

You use the preparation by

- Compute $x_s = U^{-1}_sL^{-1}_s b$ in single.
- Convert $x_s$ to double to get $x$.
- Compute $r = b - Ax$ in double
- Compute $e_s = U^{-1}_sL^{-1}_s r$ in single
  You will probably not have to convert $r$ to single to do this
- Convert $e_s$ to double to get $e_d$
- Compute $x \leftarrow x + e_d$ in double.
Some Matlab

Preparation:

\[
AS = \text{single}(A);
[LS, US] = \text{lu}(AS);
\]

Computing the correction:

\[
xs = US \backslash (LS \backslash b);
xd = \text{double}(xs);
rd = b - A \times xd;
es = US \backslash (LS \backslash rd);
ed = \text{double}(es);
xd = xd + ed;
\]
The iteration

\[
\begin{align*}
AS &= \text{single}(A); \\
[LS,US] &= \text{lu}(AS); \\
xS &= US \backslash (LS \backslash b); \\
xd &= \text{double}(xs); \quad rd = b - A \times xd; \\
\text{while } \text{norm}(rd, \infty) > 1.d-13 \\
\quad es &= US \backslash (LS \backslash rd); \\
\quad ed &= \text{double}(es); \\
\quad xd &= xd + ed; \\
\quad rd &= b - A \times xd; \\
\text{end}
\end{align*}
\]
Some Results: Method of Synthetic Solutions

Begin with

\[ n=100; \]
\[ A=\text{rand}(n,n); \ x_e=\text{rand}(n,1); \ b=A*x_e; \]

\[ \kappa_2(A) \approx 1900. \]

<table>
<thead>
<tr>
<th>relative error</th>
<th>relative residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.77e-05</td>
<td>1.52e-07</td>
</tr>
<tr>
<td>1.61e-11</td>
<td>2.25e-13</td>
</tr>
<tr>
<td>2.02e-14</td>
<td>1.02e-16</td>
</tr>
</tbody>
</table>

Analysis via Back-of-the-Envelope:

Suppose $\delta_s$ is the relative error the single precision computations.

- So $x = U_s^{-1}L_s^{-1}b = x^* - e$,
- $e = O(\delta_s)$,
- and $e_d = U_s^{-1}L_s^{-1}r = e + \|e\|O(\delta_s) = e + O(\delta_s^2)$.
- Which tells me that the correction is

$$x = x_d + e_d = x + e + O(\delta_s^2) = x^* + O(\delta_s^2)$$

So if $\delta_s = 10^{-k}$, you get $k$ figures with every iteration.
What’s $\delta_s$?

- $\epsilon_s \approx 10^{-8}$, single precision roundoff?
- $\kappa(A)\epsilon_s \approx 1900 \times 10^{-8} \approx 10^{-5}$?

Looks like $\delta_s = \kappa(A)\epsilon_s$. 
A different perspective

Is this a stationary iterative method? Is the fixed point map

\[ K(x) = x + S(x) \equiv x + l^d_s U^{-1}_s L^{-1}_s l^s_d (b - Ax) \]

So \( K \) is linear if the maps

- \( l^d_s \) conversion from single to double
  No problem: puts the same number is a bigger bucket
- \( l^s_d \) conversion from double to single
  This rounding, just as linear as floating point addition

Bottom line: it’s close enough to linear.
What’s the iteration matrix?

The map is

$$K(x) = x + I_d^s U_s^{-1} L_s^{-1} l_d^s (b - Ax)$$

$$= (I - I_d^d U_s^{-1} L_s^{-1} l_d^s A)x + I_d^d U_s^{-1} L_s^{-1} l_d^s b \equiv M_s x + x_s$$

So it’s preconditioned Richardson iteration and the convergence rate is

$$\rho(M_s) = \rho \left( I - I_d^d U_s^{-1} L_s^{-1} l_d^s A \right).$$

How might we estimate that?
Estimating $\|M_s\|$ 

How about evaluating $\|M_s x - x\|$ for a random $x$? We already did that and got

<table>
<thead>
<tr>
<th>relative error</th>
<th>relative residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.77e-05</td>
<td>1.52e-07</td>
</tr>
<tr>
<td>1.61e-11</td>
<td>2.25e-13</td>
</tr>
<tr>
<td>2.02e-14</td>
<td>1.02e-16</td>
</tr>
</tbody>
</table>

So things look consistent, and $L_s^d U_s^{-1} L_s^{-1} I_s$ is a good approximate inverse of $A$.

Warning! Matlab does not support sparse single precision arrays.
There’s more!

Suppose you have two solvers $S_1$ and $S_2$ which are good at different things.

Suppose

- $x_+ = S_1(x_c, b)$ is a few iterations of an iterative solver
  - for $Ax = b$,
  - with $x_c$ as the incoming iterate, and
  - $x_+$ as the new iterate.

- $S_2(b)$ is a solver (iterative or direct) which returns a “converged” result for $Ae = r$.

- Given the transfer maps, these solvers could have different physics, precisions, grids, ...
Two-solver correction

You can let one correct the other via . . .

\[
\begin{align*}
x &= x_0 \\
\textbf{while} & \text{ Not happy do} \\
x_{1/2} &= S_1(x, b) \\
r &= b - Ax_{1/2} \\
e &= S_2(r) \\
x &= x_{1/2} + e \\
\textbf{end while}
\end{align*}
\]
Examples

- Iterative refinement: $S_1 = I; \ S_2 = U_s^{-1}L_s^{-1}$.
- MCSA: $S_1$ is one step of Richardson; $S_2$ is Monte Carlo solve
- Two-grid: $S_1$ is a couple steps of Jacobi, Gauss-Seidel, ... $S_2$ is a solve on a coarser grid.
- Multigrid: Like two-grid, but $S_2$ is a recursive call to MG.
Recall our sad story with Jacobi for this problem.

- We had a simple example.
- Jacobi did ok for the first few iterations and then went very, very slowly.
- It seemed to kill the high frequency error fast, and get stuck on the low frequency stuff.

Let’s look again.
Let’s solve

\[-u'' = 0, \quad u(0) = u(1) = 0,\]

with \( h = 1/101 \) and \( N = 100 \). The solution is \( u = 0 \). We will use
as an initial iterate

\[u_0 = x(1 - x) + \frac{1}{10} \sin(49\pi x)\]

We will take 100 Jacobi iterations.
Matrix Representation

\[ Au = f \]

where \( A \) is tridiagonal and symmetric

\[
A^h = \frac{1}{h^2} \begin{pmatrix}
2 & -1 & 0 & \ldots & 0, & 0 \\
-1 & 2 & -1 & \ldots & 0 & 0 \\
0 & -1 & 2 & -1 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots, & 0, & -1 & 2 & -1 \\
0 & \ldots, & \ldots, & 0 & -1 & 2
\end{pmatrix}
\]
Theorem: $A$ is symmetric positive definite. The eigenvalues are

$$\lambda_n = h^{-2} 2 \left(1 - \cos(\pi nh)\right).$$

The eigenvectors $u_n = (\xi^n_1, \ldots, \xi^n_N)^T$ are given by

$$\xi^n_i = \sqrt{2/h} \sin(ni\pi h)$$
Initial Error as Function of $x$
Part IXa: Residual Correction Methods

Final Error as Function of $x$

![Graph showing the final error as a function of $x$. The graph has a peak around $x = 0.5$ and decreases as $x$ moves towards 0 and 1.]
Final Error as Function of $x$
How’s this for a great idea?

- Start at mesh size $h$.
- Do a few Jacobi’s (one?)
  We’ll call Jacobi (or whatever) the smoother.
- Transfer residual to mesh $2h$.
- Do an exact solve of $D_{2h}^2 e_{2h} = r_{2h}$
- Send $e_{2h}$ to $h$ mesh and correct $x \leftarrow x + e_h$
And the details?

- Exactly how many Jacobi’s do I need here?
- Can I do the obvious thing for intergrid transfer?
- Is this really likely to work?!?
Two-grid algorithm: notation

- $\Omega^h$: vectors corresponding to the $h$-mesh $\mathbb{R}^{n^d}$ where there are $n$ points in each direction and $d = 1, 2, 3$.
- $A^h$: discrete differential operator on $\Omega^h$.
- $I_{2h}^h$: fine-to-coarse intergrid transfer.
- $I_h^h$: coarse-to-fine intergrid transfer.
- $TG(v^h, f^h)$: a single two-grid iteration for $A^h v^h = f^h$.
- $S^\nu(v^h, f^h)$: $\nu$ iterations of the smoother.
Two-Grid Method

\[ \mathbf{v}^h \leftarrow TG(\mathbf{v}^h, \mathbf{f}^h) \]
\[ \mathbf{v}^h \leftarrow S^{\nu_1}(\mathbf{v}^h, \mathbf{f}^h) \]
\[ \mathbf{r}^h = \mathbf{f}^h - A^h \mathbf{v}^h \]
\[ \mathbf{r}^{2h} = l_h^{2h} \mathbf{r}^h \]
\[ \text{Solve}(?) \quad A^{2h} \mathbf{e}^{2h} = \mathbf{r}^{2h} \]
\[ \mathbf{e}^h = l_h^{2h} \mathbf{e}^{2h} \]
\[ \mathbf{v}^h = \mathbf{v}^h + \mathbf{e}^h \]
\[ \mathbf{v}^h \leftarrow S^{\nu_2}(\mathbf{v}^h, \mathbf{f}^h) \]
Let’s make some questionable decisions.

- One Jacobi presmooth $(\nu_1 = 1, \nu_2 = 0)$
- $I_{2h}^h$: piecewise linear interpolation.
- $I_{h}^{2h}$: restriction by injection.
function us=jacobi(u,b)
n=length(u); h=1/(n+1); h2=h*h; us=zeros(n,1);
us(2:n-1)=h2*b(2:n-1) + u(3:n) + us(1:n-2);
us(1) = h2*b(1) + u(2); us(n) = h2*b(n) + u(n-1);
us=us*.5;
Restriction by Injection

function uc = inject(uf)
% INJECT
% fine to coarse intergrid transfer by injection
% function uc = inject(uf)
% nf = length(uf);
uc = uf(2:2:nf-1);
function uf = ctof(uc)
% CTOF
% function uf = ctof(uc)
% Coarse to fine intergrid transfer
%
nc = length(uc); nf = 2*(nc+1) - 1; uf = zeros(nf,1);
% Use zero boundary conditions
uf(1) = 0.5*uc(1); uf(nf) = 0.5*uc(nc);
% Vectorize the rest.
uf(3:2:nf-1) = 0.5*(uc(1:nc-1)+uc(2:nc));
uf(2:2:nf) = uc;
That would be great, but it seems to get stuck.
Gauss-Seidel is better. Why?

The graph shows the convergence of residual norms for different values of $\nu_1$. The norms are plotted on a log scale, demonstrating exponential decay with increasing iterations.
What happened?

- We had this idea that Jacobi did great on high-frequency and poorly on low, so
- I assumed that Gauss-Seidel did the same.
- I was half right.
- It’s time to do an experiment.
  - Take one Jacobi/Gauss-Seidel iteration for $Au = 0$ with $u_0 = \sin(i\pi x)$ for $1 \leq i \leq n$
  - and see what the reduction rate is.
Results for Gauss-Seidel
Results for Jacobi
I’d like to save Jacobi because it parallelizes and vectorizes well. One fix: damped Jacobi

\[ u^{new} = (1 - \omega)u^{old} + \omega u^{jacobi} \]

For our problem it’s

\[ u_{i}^{new} = (1 - \omega)u_{i}^{old} + \omega(h^{2}b_{i} + u_{i-1}^{old} + u_{i+1}^{old})/2. \]

So what’s \( \omega \)?

- \( \omega = 1 \) is Jacobi.
- \( \omega = 0 \) is do nothing.
- The winner is \( \omega = 2/3 \) because . . .
Analysis of Damped Jacobi

Damped Jacobi is

\[ x_{n+1} = (1 - \omega)x_n + \omega D^{-1}(L + U)x_n + D^{-1}b \]

with iteration matrix \( M^h_{DJ} = (1 - \omega)I + \omega M_{JAC} \), where

\[ M^h_{JAC} = -D^{-1}(L + U). \]
We showed that the eigenvalues of $M_{JAC}$ were

$$\mu_n = 1 - (h^2/2)\lambda_n$$

with the same eigenvectors as $A^h$. Similarly

$$\mu_n = (1 - \omega) + \omega(1 - (h^2/2)\lambda_n) = 1 - (h^2/2)\omega\lambda_n$$

$$= 1 - \omega(1 - \cos(\pi nh))$$
Optimal $\omega$

Now suppose $n \geq N/2$ (high frequency), then

$$\mu_n = 1 - \omega(1 - \cos(\pi nh)) = 1 - \omega + \omega \cos(\pi nh)$$

because $\cos(\pi nh) \leq \cos(\pi/2) = 0$, so

$$1 - 2\omega \leq \mu_n \leq 1 - \omega.$$ 

Minimize $|\mu_n|$ to see that the optimal value of $\omega$ is $2/3$. So

$$|\mu_n| \leq 1/3$$ for $N/2 \leq n \leq N - 1$
Results for Damped Jacobi
Comparison

![Graph showing comparison of Damped Jacobi, Gauss-Seidel, and Jacobi methods]

- **Damped Jacobi**
- **Gauss-Seidel**
- **Jacobi**

**Wave number**

- 0
- 20
- 40
- 60
- 80
- 100
- 120
- 140

**Reduction**

- 0
- 0.1
- 0.2
- 0.3
- 0.4
- 0.5
- 0.6
- 0.7
- 0.8
- 0.9
- 1


Part IXa: Residual Correction Methods
Damped Jacobi’s the winner. Let’s solve something.
Can we do better?

One problem is that restriction by injection ignores half of the fine grid points. A better idea is restriction by full weighting

\[ u_i^{2h} = \left( u_{2i-1}^h + 2u_{2i}^h + u_{2i+1}^h \right) / 4 \]

function vcoarse = ftoc(vfine)
% FTOC fine to coarse intergrid transfer by full weighting
    nf = length(vfine);
    nc = .5*(nf+1)-1;
    vtmp = zeros(nf+2,1);
    vcoarse = .5*vfine(2:2:nf-1);
    vcoarse = vcoarse + .25*(vfine(1:2:nf-2)+vfine(3:2:nf));
Restriction by full weighting, two-grid, Damped Jacobi

\[
\frac{\| r \|}{\| r_0 \|} \quad \text{iterations: } 0, 5, 10, 15, 20, 25
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
\nu_1 & 1 & 4 & 8 \\
\hline
\| r \| / \| r_0 \| & 10^0 & 10^{-1} & 10^{-2} \\
\hline
\end{array}
\]
Something more to like. Matrix representations

\[
I^h_{2h} = \frac{1}{2}
\begin{pmatrix}
1 & 0 & \ldots & 0 & 0 & 0 \\
2 & 0 & \ldots & 0 & 0 & 0 \\
1 & 1 & \ldots & 0 & 0 & 0 \\
0 & 2 & \ldots & 0 & 0 & 0 \\
0 & 1 & 1 & \ldots & 0 & 0 \\
0 & 0 & 2 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 1 \\
0 & 0 & 0 & \ldots & 0 & 2 \\
0 & 0 & 0 & \ldots & 0 & 1 \\
0 & 0 & 0 & \ldots & 0 & 1 \\
\end{pmatrix}
\]

, \quad I^{2h}_{h} = \frac{1}{2}(I^{h}_{2h})^{T} \text{(full weighting)}
Two-Grid is a stationary iterative method

Go through the loops and

\[
M_{TG} = M_S^{\nu_2}(I - \frac{I_{2h}}{2h}(A^{2h})^{-1} \frac{I_{h}}{h} A^{h})M_S^{\nu_1}
\]

where \(M_S\) is the iteration matrix for the smoother. One standard configuration is

- Damped Jacobi smoother
- \(\nu_1 = \nu_2 = 1\)
Estimate the spectral radius

- Look at ratio of relative residuals.
- The ratios should converge to the spectral radius unless something strange happens.
- Do this for various values of $h$ to see if the spectral radius depends on $h$ or not.
Feel-good Prediction

- Each application of damped Jacobi reduces high-frequency error by $1/3$
- So, if the coarse grid correction eliminates the low-frequency errors completely . . .

then

$$\rho(M_{TG}) = 1/9.$$

Really?
We have a winner! Ratio \( = \frac{1}{9} \)
Where are we?

- **Good:**
  - TG is a stationary iterative method.
  - Solver work moved to coarse grid.
  - Convergence rate independent of $h$.

- **Bad:**
  - Coarse mesh solver is still a matrix factorization.

**Fix:** replace the coarse mesh solver with another two-grid iteration and get ...