1 Multigrid for Poisson: $A\underline{u} = \underline{f}$.

- Multigrid methods are some of the fastest available solution strategies for multidimensional PDEs.
- In practice, they are most commonly used as preconditioners within Krylov-subspace projection (KSP) methods, such as conjugate gradient iteration or GMRES, which gives them speed and robustness.
- Within a KSP, the multigrid solver returns $\underline{z}$ in the preconditioning step, *Solve*

$$M_g \underline{z} \;=\; \underline{r}. \tag{1}$$

- In the following, we'll think about solving $A\underline{u} = \underline{f}$ as being equivalent to $M_g\underline{z} \;=\; \underline{r}$, with recognition that our initial guess will be $\underline{u}_0 = 0$ and that $\underline{u}$ (which approximates the error in the context of a KSP) will satisfy homogeneous boundary conditions.

- *There are two essential components to a multigrid solver,*
  1. **Error Smoothing**, and
  2. **Coarse-Grid Correction**

- We'll start with smoothing.

2 Smoothing

- Consider the following fixed-point iteration for $A\underline{u} = \underline{f}$, with $\underline{u}_0 = 0$:

```
for $k = 1 : m$
    $\underline{u}_k \;=\; \underline{u}_{k-1} \;+\; \omega M^{-1}(\underline{f} - A\underline{u}_{k-1})$        (2)
end
```

- Here, M is alternately referred to as a *smoother* or *preconditioner.*
- Most frequently, M =diag(A).
- We also sometimes refer to the whole loop as a smoother.
- And, in the KSP, we refer to the whole multigrid part as a preconditioner (of the KSP).
- Let $\underline{r}_k \;:=\; \underline{f} - A\underline{u}_k \;\equiv\; A(\underline{u} - \underline{u}_k) \;=\; A\underline{e}_k$.
- The *residual*, $\underline{r}_k$, is A times the *error*, $\underline{e}_k := \underline{u} - \underline{u}_k$. Always.
- Solving $A\underline{u} = \underline{b}$ amounts to finding the error.

- Suppose we have a known guess (or *iterate*), $\underline{u}_k$.
- Then $\underline{e}_k := \underline{u} - \underline{u}_k \Longleftrightarrow \underline{u} = \underline{u}_k + \underline{e}_k$.
- If we find $\underline{e}_k$, we're done.
- The equation for $\underline{e}_k$ is: $$A\underline{e}_k \;=\; \underline{r}_k \;:=\; \underbrace{\underline{f} - A\underline{u}_k}_{computable}. \tag{3}$$
- It is easy to compute the rhs, but solving $A\underline{e}_k = \underline{r}_k$ seems as difficult as solving $A\underline{u} = \underline{b}$.
- We shall see, however, that we can infer certain properties about $\underline{e}_k$ that will make it relatively easy to solve.
- Specifically, we will design the fixed point iteration to try to make $\underline{e}_k$ as *smooth* as possible.
- At that point, we can approximate it on a coarser mesh, which will mean that it is less expensive to compute.

- Let's look more closely at the smoothing step,

$$\underline{u}_k = \underline{u}_{k-1} + \omega M^{-1}(\underline{f} - A\underline{u}_{k-1})$$

- For now, think of $M = I$ or, almost equivalently, as a modified system with $\tilde{A} = M^{-1}A$ and $\tilde{\underline{f}} = M^{-1}\underline{f}$, such that

$$\underline{u}_k = \underline{u}_{k-1} + \omega(\tilde{\underline{f}} - \tilde{A}\underline{u}_{k-1}). \qquad (4)$$

- We'll drop the ˜ for now, but will re-introduce M later.
- The iteration (4) is known as *Richardson iteration.*
- We'll see that it is analogous to Euler-Forward timestepping.

- Recall the unsteady heat equation,

$$\frac{\partial u}{\partial t} = \nabla^2 u + f \implies \frac{d\underline{u}}{dt} = -A\underline{u} + \underline{f}, \qquad (5)$$

for which the unsteady part decays in time, with the particular property that the high wavenumber components decay rapidly, while the low wavenumber components decay more slowly.

- For small enough timestep sizes, Euler-Forward timestepping

$$\frac{\underline{u}^n - \underline{u}^{n-1}}{\Delta t} = -A\underline{u}^{n-1} + \underline{f} \tag{6}$$

$$\Longrightarrow \quad \underline{u}^n = \underline{u}^{n-1} + \Delta t(\underline{f} - A\underline{u}^{n-1}), \tag{7}$$

  yields solutions with this same property.

- To see this behavior, let's discuss how timestepping is relevant to solving $A\underline{u} = \underline{f}$.
- Define $\underline{u}$ as the solution to $A\underline{u} = \underline{f}$.
- We find an equation for the *error*, $\underline{e}^n := \underline{u} - \underline{u}^n$, as follows.

$$\begin{array}{rcccl} +\,(\;\underline{u} & = & \underline{u} & + & \Delta t(\underline{f} - A\underline{u})) \\ -\,(\;\underline{u}^n & = & \underline{u}^{n-1} & + & \Delta t(\underline{f} - A\underline{u}^{n-1})) \\ \hline \underline{e}^n & = & \underline{e}^{n-1} & - & \Delta t A\underline{e}^{n-1}. \end{array}$$

- If we start with $\underline{u}^n = 0$, this equation is nothing other than EF applied to the homogeneous ODE for the error,

$$\frac{d\underline{e}}{dt} = -A\underline{e}, \quad \underline{e}(t=0) = \underline{u}. \tag{8}$$

- The error decays from its initial value, $\underline{e}^0 = \underline{u}$ towards 0.

We've seen this characteristic behavior in earlier discussions of timestepping.

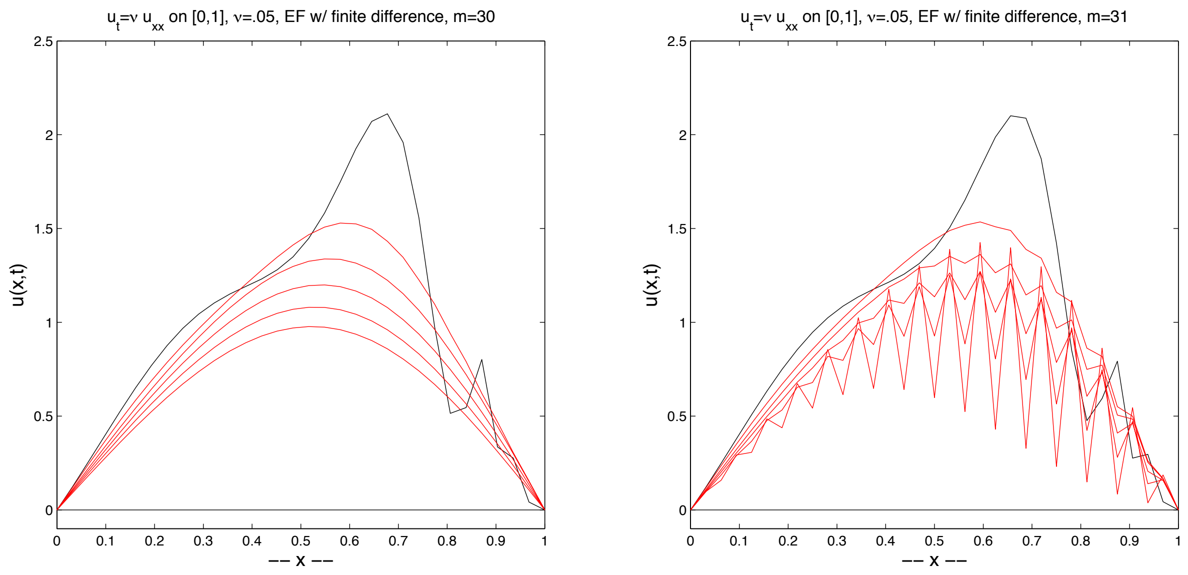


Figure 1: Euler-Forward solution behavior for unsteady 1D heat equation at two spatial resolutions, same Δt.

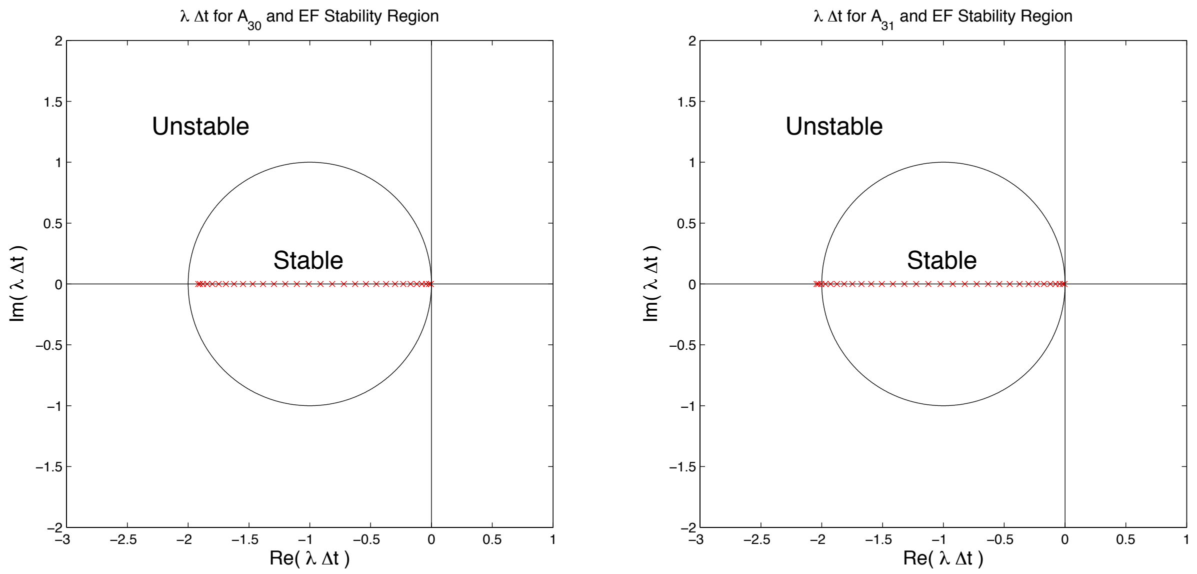


Figure 2: Stability region for Euler forward timestepping and the scaled eigenvalues, $\Delta t \lambda$ for $L = -\nu A$ with $\nu = .05$ and A the m-point 2nd-order finite difference approximation to $-u_{xx}$: (left) m=30, (right) m=31.

- In the figure on the upper left, the nonsmooth IC rapidly evolves to a smooth one, provided that $|\lambda \Delta t| < 2$, which is the bound for the EF stability region.
- If we exceed this bound, however, the solution blows up, as illustrated in the top right.
- So, the prefactor Δt in the EF timestepping depends on $\max \lambda(A)$,

$$\begin{aligned} \underline{u}^n &= \underline{u}^{n-1} + \Delta t(\underline{f} - A\underline{u}^{n-1}) \\ \Delta t &\le \frac{2}{\lambda_m}, \end{aligned}$$

  assuming $0 < \lambda_1 \le \lambda_2 \le \cdots \le \lambda_m$.
- Notice that the remark regarding smoothing applies only to the spectral components that are well within the stability region.

- Recall our earlier analysis for EF with $L = -A$ and A an $m \times m$ SPD matrix having a complete set of eigenvectors.
- For any $\underline{u} \in \mathbb{R}^m$, we can write

$$\underline{u} = \sum_k \hat{u}_k \underline{z}_k, \tag{9}$$

$$L\underline{z}_k = \mu_k \, \underline{z}_k, \tag{10}$$

  where $\mu_k(L) = -\lambda_k(A)$ are the eigenvalues of L

- Applying this decomposition to our EF timestepper, we have

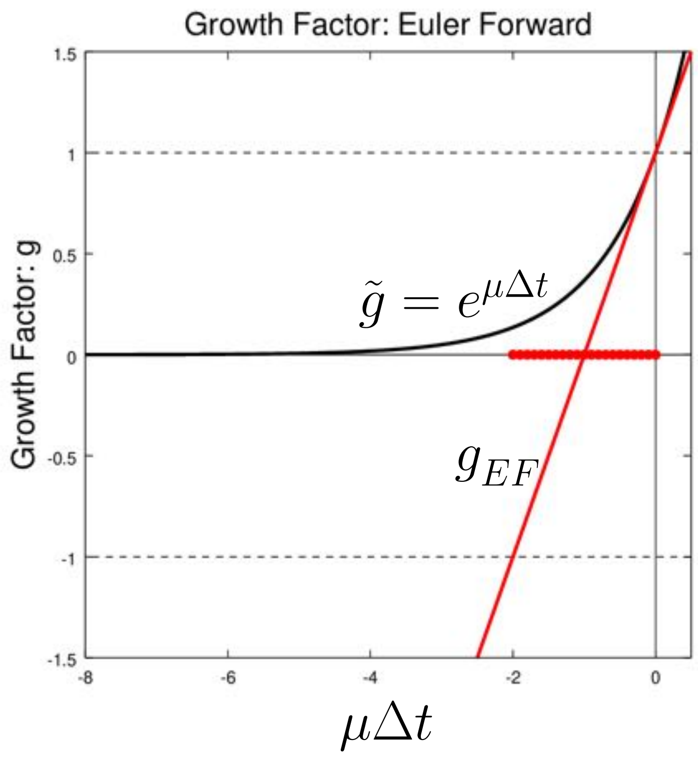
$$\hat{u}_k^n = \underbrace{[1 + \Delta t \mu_k]}_{g(\mu \Delta t)} \hat{u}_k^{n-1}, \tag{11}$$

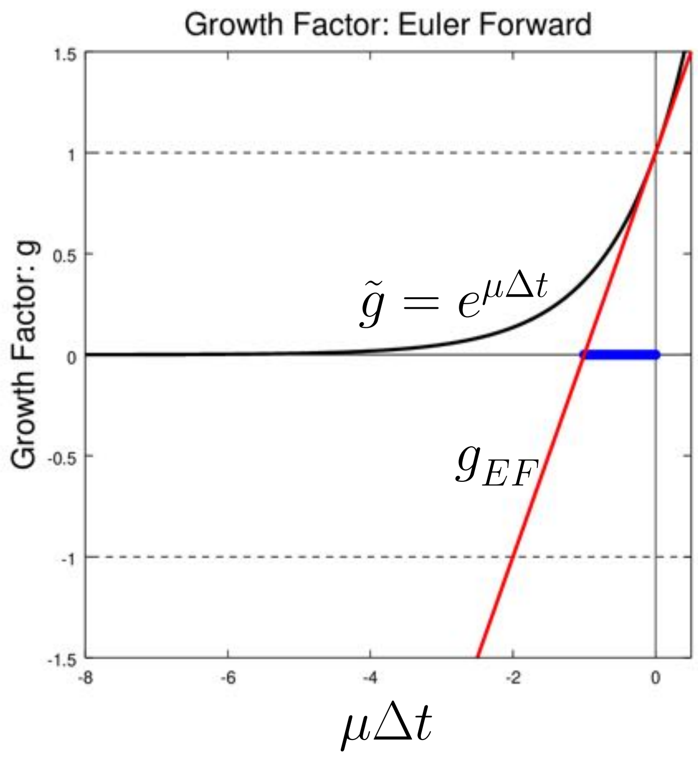
  where $g = g_{EF}(\mu \Delta t)$ is the *growth factor* for EF.

- We have a similar growth factor for our exact (analytical) timestepper.
- If we do not discretize in time, the Fourier coefficients satisfy

$$\hat{u}_k^n = \underbrace{e^{\mu \Delta t}}_{\tilde{g}(\mu \Delta t)} \hat{u}_k^{n-1}. \tag{12}$$

- Consider the case when $\mu(L) = -\lambda(A)$ is negative real (corresponding to $\lambda(A) > 0$).
- We plot g_{EF} and $\tilde{g}$ below for two different distributions of $\mu \Delta t$.





- In the right figure, we show a distribution of $\mu_k \Delta t$ in red where the largest (in magnitude) is $\mu_m \Delta t = -2$.
- For this case, the growth rate will be $g_{EF}(-2) = -1$, which means that high-frequency error will *not decay*.

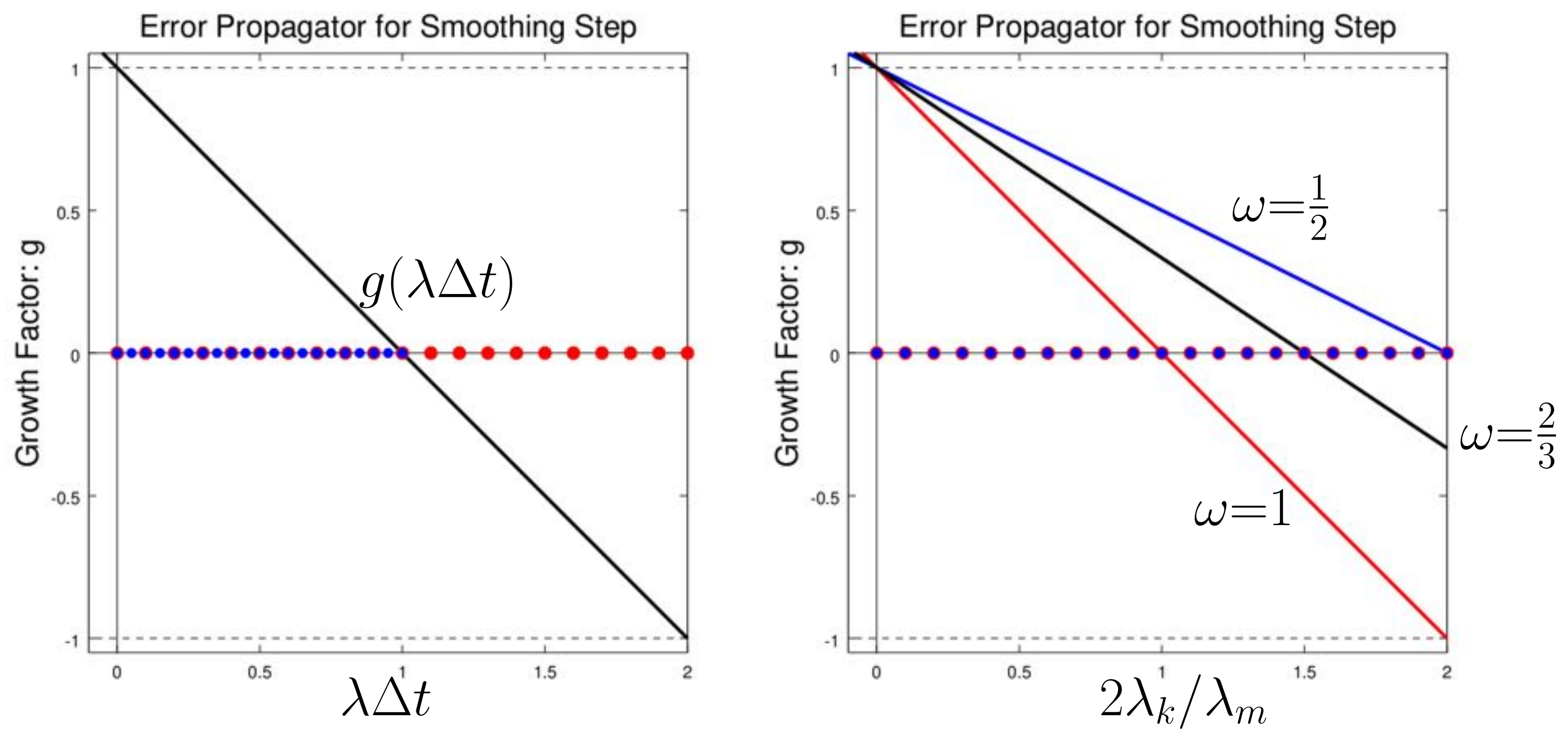
- In the figure on the right, we illustrate a case with the same eigenvalue distribution, μ_k, but with a value of Δt such that $\mu_m \Delta t = -1$.
- For this case, $g_{EF}(\mu_m \Delta t) = 0$, which means that the high-frequency error is anihilated after a single timestep.
- Paradoxically, taking a *smaller* timestep leads to more rapid decay of the high-frequency error.
- We refer to this process as under-relaxed Jacobi (or Richardson) iteration.

3 From Euler Forward to Smoothing

- Let's turn the preceding plots around so that the x-axis is $\lambda_k(A)$ and focus solely on the EF case.
- The growth factor is then

$$g(\lambda \Delta t) \;=\; 1 \;-\; \lambda \Delta t, \tag{13}$$

which we plot on left below on the left for the same two eigenvalue distributions considered in the preceding figures.



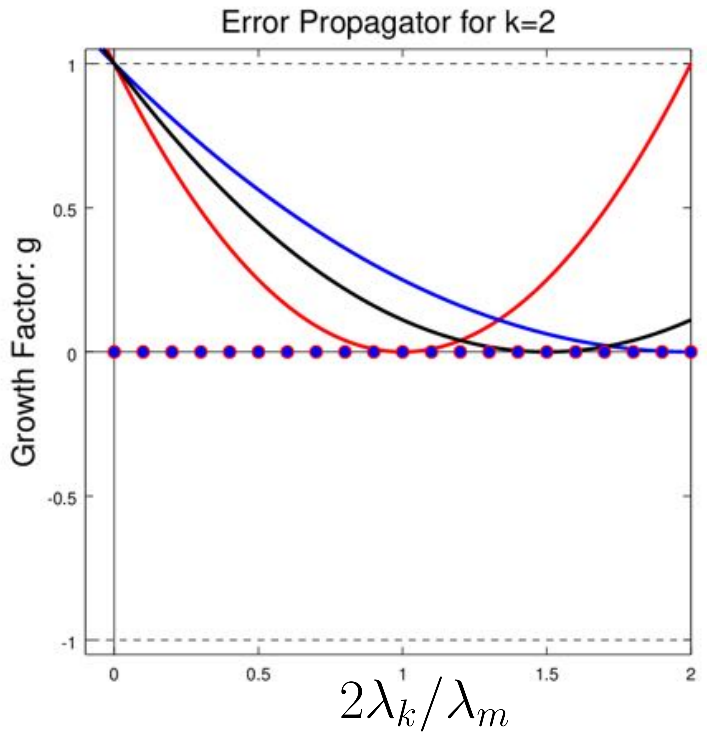
- In the right figure, we have rescaled the x-axis to be $2\lambda_k/\lambda_m$.
- The growth factor in this case becomes

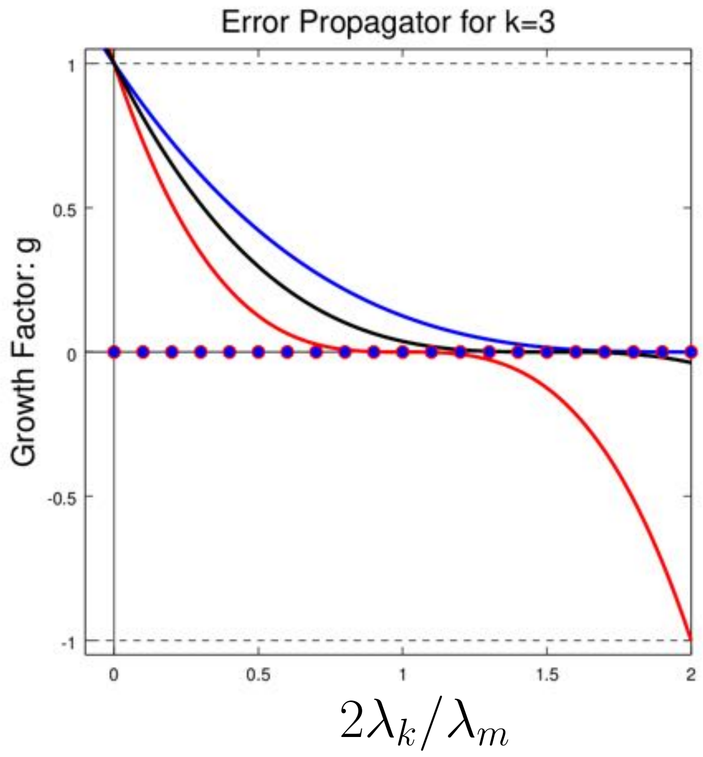
$$g_\omega(\lambda_k) \;=\; 1 \;-\; \omega\frac{2\lambda_k}{\lambda_m}. \tag{14}$$

- The eigenvalue distributions on the x-axis no longer depend on Δt, but the growth curves do, with

$$\Delta t \;=\; \omega\,\frac{2}{\lambda_m}. \tag{15}$$

- Here, $\omega = 1$ corresponds to the largest stable Δt (i.e., the *nonsmoothing case* corresponding to the red dots in the left figure)
- $\omega = 1/2$ corresponds to $\Delta t = 1/\lambda_m$, which rapidly anihilates the high wavenumber error (i.e., *smooths* the solution).
- A third case, $\omega = \frac{2}{3}$ is generally more optimal than $\omega = \frac{1}{2}$.
- With this choice, each round of the smoothing iteration reduces the error in the upper half of the spectrum by 1/3.
- The figures below illustrate what happens to the error distribution after two and three rounds of smoothing.



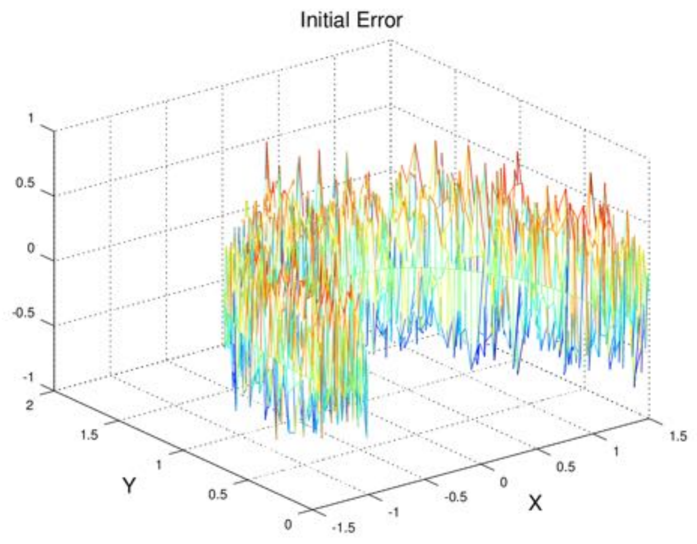


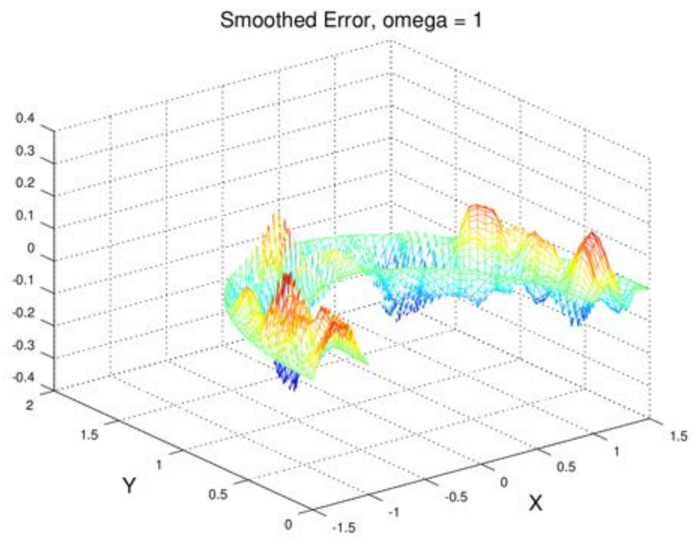
- It is clear from the black curve that $\omega = \frac{2}{3}$ does a better job of suppressing the error in the mid-range of the spectrum, albeit at the cost of less error reduction than $\omega = \frac{1}{2}$ at the very high end.

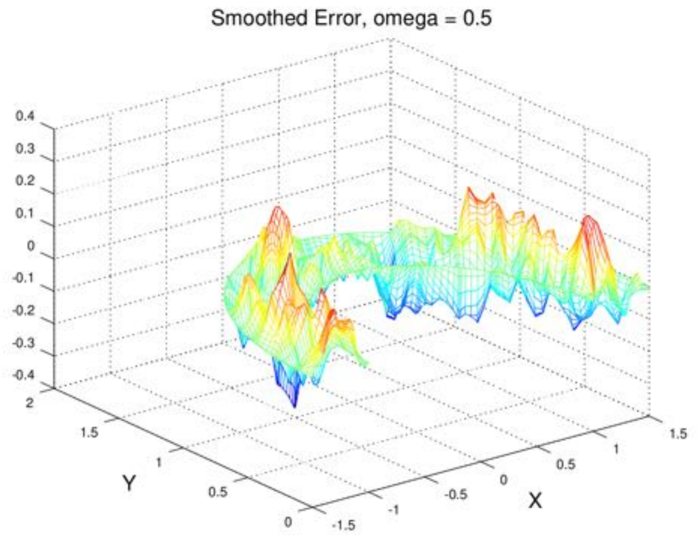
- Let's look at the smoothing idea in the context of the SEM.
- Here, λ reflects the eigenvalues of $D^{-1}A$, where D =diag(A).
- The `smoother.m` file reads,

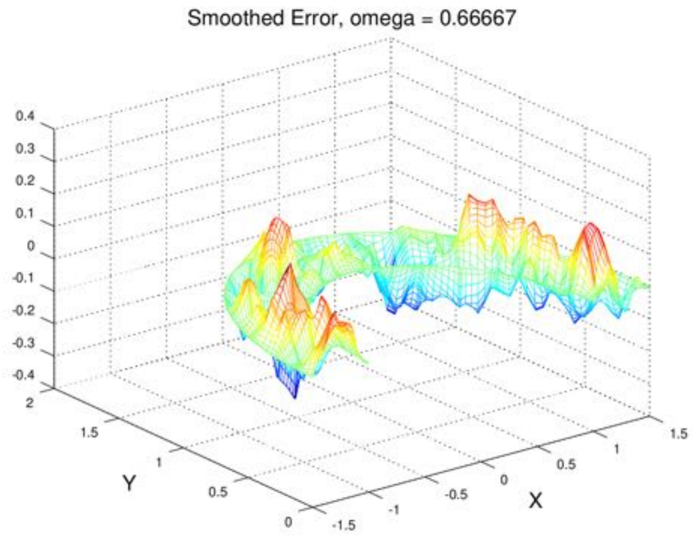
```
function [u]=smoother...
   (Fl,lam_max,omega,m,b0,nu,M,Q,Bl,Grr,Grs,Gss,Dh,Di,ifnull);

omega = omega * (2 / lam_max); %% Rescale omega

r = Fl;
z = Di.*(M.*qqt(Q,r));    %% diagonal preconditioner
u = omega*z;              %% initial guess for x is 0.

for iter=2:m;             %% Apply m rounds of Jacobi smoothing
   r = Fl - axl(u,b0,nu,Bl,Grr,Grs,Gss,Dh);
   z = Di.*(M.*qqt(Q,r));
   u = u + omega*z;
end;
```

- Here, $\underline{u}$ is the current guess,
  $\underline{r} := \underline{f} - A\underline{u}$ is the residual, and
  $\underline{z} = M^{-1}\underline{r}$ is the search direction.
- The update step is thus $\underline{u} = \underline{u} + \omega\, M^{-1}(\underline{f} - A\underline{u})$.
- The following figures show the results of $k = 50$ rounds of smoothing for $\omega = 1$, $\frac{1}{2}$, and $\frac{2}{3}$, along with the initial error, for a 2D SEM example.





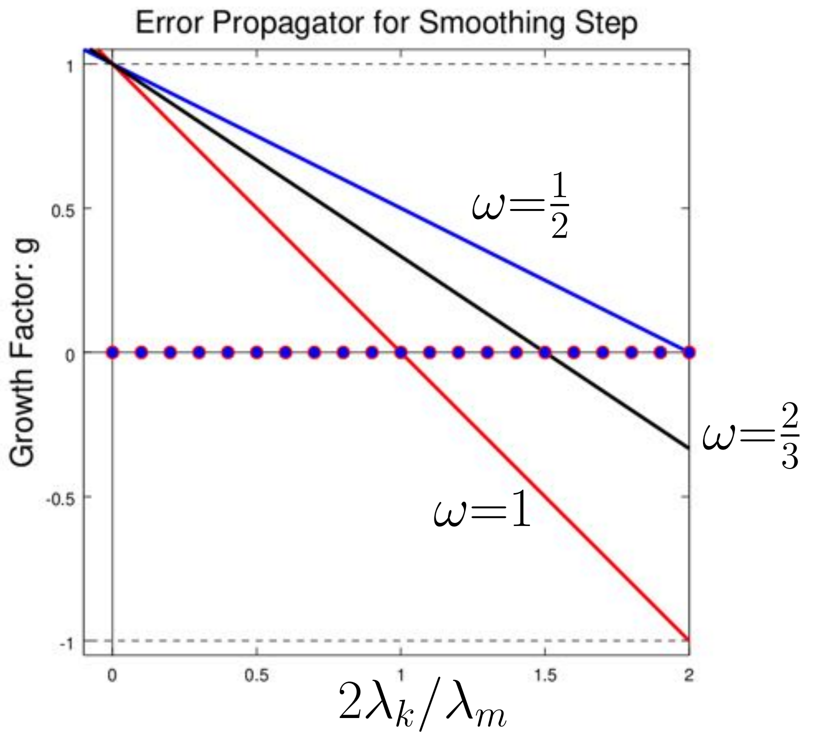


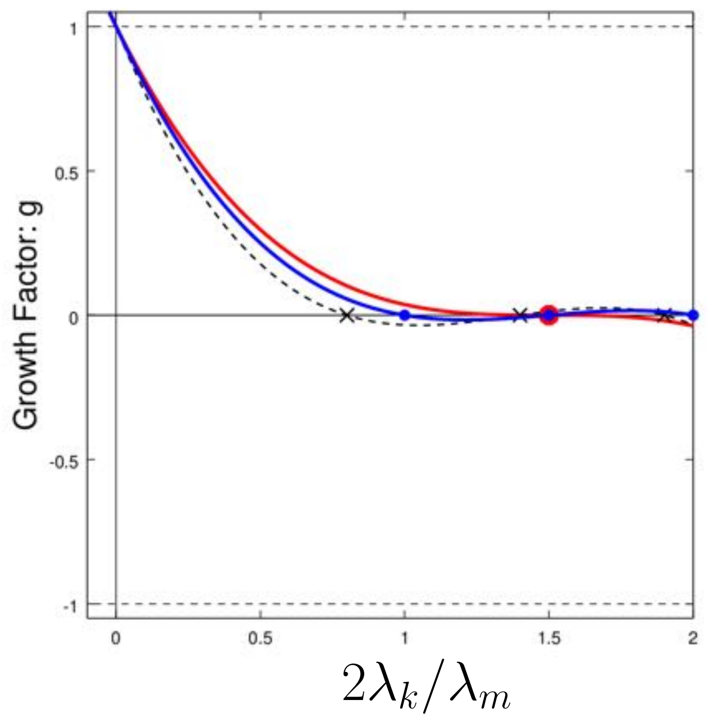


- The solution (initial error), upper left, is a random field with the correct boundary conditions.
- The upper right figure clearly shows that the choice $\omega=1$ fails to smooth the solution.
- The lower left figure shows that $\omega=\frac{1}{2}$ leads to an excellent smoother.
- However, the case $\omega=\frac{2}{3}$, lower right, leads to more overall error reduction.

4 Chebyshev Smoothing

- A significant advance in multigrid methods has resulted from recognizing that smoothing can be more effective if one chooses different values of ω on each iteration.
- The idea is to spread out the zero crossings such that the error is suppressed over a broader range, rather than just focusing on error reduction in the neighborhood of the principal root.
- Below, we plot the $k = 1$ plots for our three choices of ω on the left.
- On the right we contrast smoothing after 3 rounds with our optimized $\omega = \frac{2}{3}$ (red) and the case with $\omega_k = \frac{1}{2}, \frac{2}{3}$, and 1, for round k=1, 2, and 3 (blue).





- Overall, the blue curve shows broader error suppression than the red curve.
- Another distribution, seen as the black dashed line, shows better suppression using roots [0.9, 1.4, 1.9].
- This observation leads to the natural question, *Can we further optimize the choice of the roots of our kth-order polynomial?*
- Remarkably, the answer has its roots in spectral methods!
- Namely, if you wish to minimize the maximum of a polynomial over a given interval, subject to some nontrivial scaling, then answer is typically a scaled and translated *Chebyshev* polynomial.
- There are a variety of choices, but recent work by James Lottes has developed an algorithm using *optimized 4th-kind Chebyshev polynomials*, which is given below.

```
function [x]=cheby4(Fl,lam_max,omega,k,b0,nu,M,Q,Bl,Grr,Grs,Gss,Dh,Di,ifnull);
%% Apply kth-order optimized 4th-kind Chebyshev
%% (from Malachi Phillips dissertation,2023).
lmax_i = 1./lam_max;
beta=Beta(k,:);
r = Fl;
x = 0*r;
d = (lmax_i*4./3.)*(Di.*(M.*qqt(Q,r)));%Diagonal preconditioner
for i=1:k-1;
   x = x + beta(i)*d;
   r = r - axl(d,b0,nu,Bl,Grr,Grs,Gss,Dh);
   s1= (2*i-1)/(2*i+3); s2=lmax_i*(8*i+4)/(2*i+3);
   d = s1*d + s2*Di.*(M.*qqt(Q,r));
end;
x = x + beta(k)*d;
```

- The algorithm requires an estimate of $\lambda_m(M^{-1}A)$, which is readily obtained from power iteration or from running CG in "Lanczos" mode, which can given estimates of the extreme eigenpairs.

- It also requires some precomputed values of β_i, which are provided in functional and tabular form in recent articles.
- Here is the table used in our matlab code:

```
Beta=[ ...
1.12500000000000                0                0                0                0                0                0;
1.02387287570313 1.26408905371085                0                0                0                0                0;
1.00842544782028 1.08867839208730 1.33753125909618                0                0                0                0;
1.00391310427285 1.04035811188593 1.14863498546254 1.38268869241000                0                0                0;
1.00212930146164 1.02173711549260 1.07872433192603 1.19810065292663 1.41322542791682                0                0;
1.00128517255940 1.01304293035233 1.04678215124113 1.11616489419675 1.23829020218444 1.43524297106744                0;
1.00083464397912 1.00843949430122 1.03008707768713 1.07408384092003 1.15036186707366 1.27116474046139 1.45186658649364];
```

- We'll look at the impact of this smoother momentarily.

5 Coarse-Grid Correction

- Once the *error* is smooth, we can return to our modified problem, *Solve* $A\underline{e}_k = \underline{r}_k$.
- Given that this problem arises in the context of an outer iteration, we do not need to solve it exactly.
- Since $e_k(\mathbf{x})$ is a smooth function, we can represent the solution on a coarser grid.
- In the SEM context with polynomial order N, we might consider approximating $\underline{e}_k$ (actually, $e_k(\mathbf{x})$) with polynomial order $N_c = N/2$ or $N-2$, say.
- The natural approach is to use a Galerkin approximation.
- Let $\underline{\tilde{e}}_c := J\underline{e}_c$ be the coarse-grid approximation to $\underline{e}_k$.
- Here, J is an interpolation matrix from the coarse space (order $N_c < N$) to the fine space (order N).
- The variational problem is, *Find* $\underline{\tilde{e}}_c \in \mathcal{R}(J)$ *such that, for all* $\underline{v} \in \mathcal{R}(J)$,

$$\underline{v}^T A \underline{\tilde{e}}_c \;=\; \underline{v}^T \underline{r}_k \;\equiv \underline{v}^T A \underline{e}_k. \qquad (16)$$

- With $\underline{v} = J\underline{v}_c$ and $\underline{\tilde{e}}_c := J\underline{e}_c$, this becomes *Find* $\underline{e}_c \in \mathbb{R}^{n_c}$ *such that, for all* $\underline{v}_c \in \mathbb{R}^{n_c}$,

$$\underline{v}_c^T J^T A J \underline{e}_c \;=\; \underline{v}_c^T J^T \underline{r}_k. \qquad (17)$$

or simply, with $A_c := J^T A J$,

$$A_c \underline{e}_c \;=\; J^T \underline{r}_k \;=:\; \underline{r}_c. \qquad (18)$$

- It is easy to interpolate on an elment-by-element basis, so application of J and J^T is *completely local.*
- Unfortunately, $A_c = J^T A J$ loses the tensor-product structure of A.
- We therefore replace it with A_{N_c}, the system matrix associated with the lower polynomial degree, N_c.
- Once we solve $A_c \underline{e}_c = \underline{r}_c$, we interpolate to the fine mesh and add this correction to the current (smoothed) iterate:

  Coarse-Grid Correction: $\underline{u}_k \;=\; \underline{u}_k \;+\; J\underline{e}_c, \qquad \underline{e}_c \;:=\; A_c^{-1} J^T \underline{r}_k.$
- Depending on the size, we solve the coarse grid problem either iteratively or directly.
- The most common approach is to apply multigrid to this problem as well, which changes the *two-level multigrid* algorithm we've described so far into an actual *multigrid* method.

- In the following, examples we revisit our SEM matlab example using two-level multigrid with $N = 7$, $N_c = 5$, and either 5 rounds of damped Jacobi smoothing $\omega = \frac{2}{3}$ or optimized 4th-kind Chebyshev smoothing with $k = 6$.
- Note that the work for $k = 6$ is effectively the same as 5 rounds of damped Jacobi smoothing.

```
     Damped Jacobi, omega=2/3          Optimized 4th-kind Chebyshev
       e_smooth      e_coarse                e_smooth      e_coarse
 1    6.3368e-01   2.5709e-01         1    5.7602e-01   2.6874e-01
 2    2.0752e-01   1.7723e-01         2    1.9069e-01   1.0420e-01
 3    1.5486e-01   1.2586e-01         3    9.0264e-02   4.2108e-02
 4    1.1029e-01   1.0912e-01         4    3.9747e-02   1.9548e-02
 5    9.3898e-02   8.0521e-02         5    1.8402e-02   1.1508e-02
 6    6.8317e-02   7.0165e-02         6    1.0927e-02   5.5164e-03
 7    5.9945e-02   5.3564e-02         7    5.2486e-03   3.3857e-03
 8    4.5350e-02   4.5779e-02         8    3.2528e-03   1.6877e-03
 9    3.8958e-02   3.5762e-02         9    1.6222e-03   1.0741e-03
10    3.0253e-02   3.0143e-02        10    1.0269e-03   5.4384e-04
```

- For each smoother choice, the table shows the error after the smoothing step and after the coarse grid correction for each MG step.
- We can see that, for the same work, the 4th-kind Chebyshev significantly outperforms, *by 3*×, standard Jacobi smoothing.
- Although not discussed here, the 4th-kind Chebyshev is more robust than red-black Gauss-Seidel smoothing for geometries that have high aspect-ratio cells.

  (Also, Gauss-Seidel is not amenable to matrix-free formulations.)
- Importantly, this preconditioner will be wrapped with a KSP, which will significantly accelerate its performance.

- Another observation: For this example, setting a tighter coarse-solve tolerance significantly improves the Chebyshev case, but not the Jacobi case.
- This observation leads to the idea that one can adjust the tolerance for the coarse (PCG) solver to be a fraction of the initial coarse-PCG residual, as illustrated by the use of `nrc`, the 2-norm of the coarse residual–normalized by the domain volume, in the following code snippet.

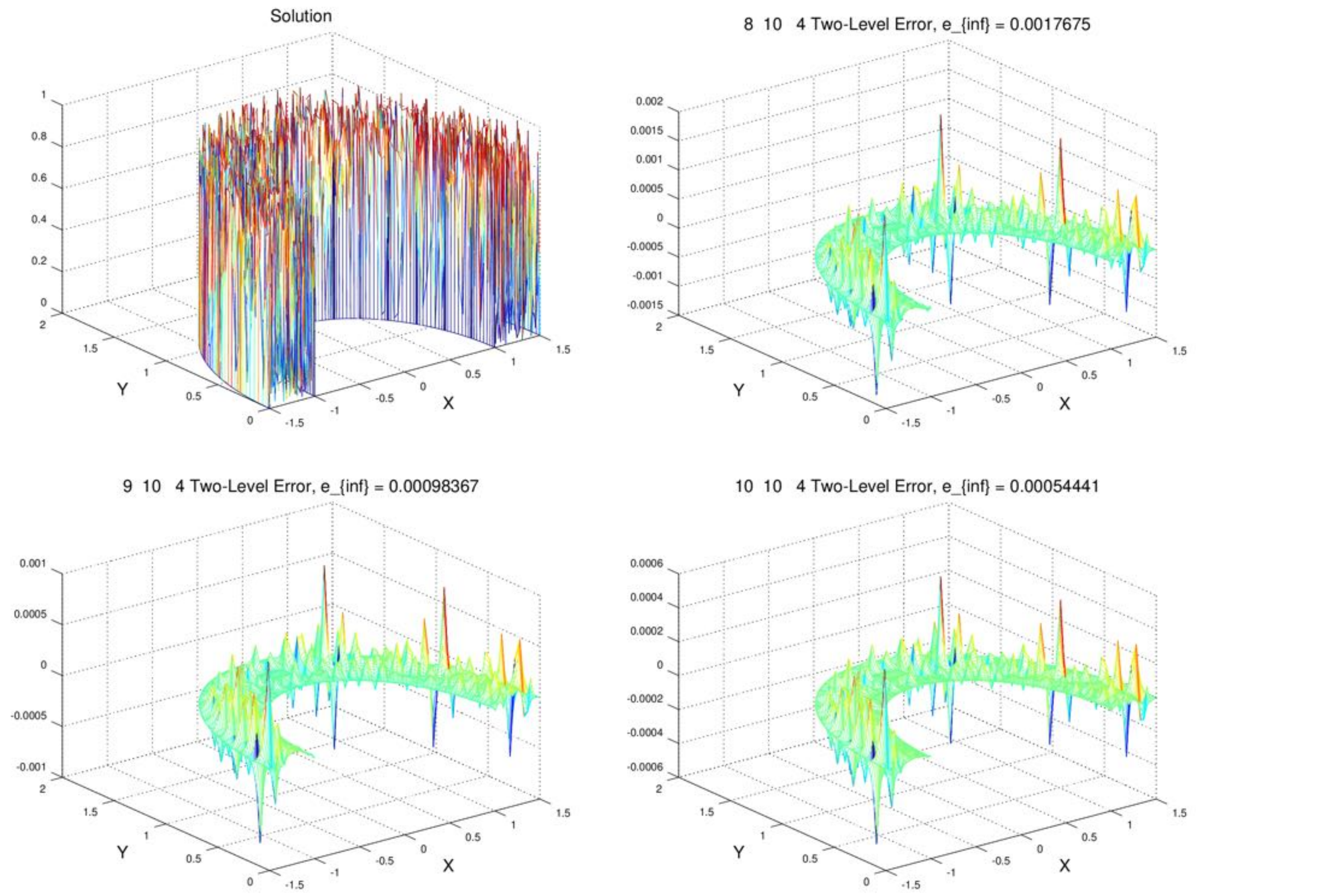
```
%% Coarse-grid correction

rl = rl-axl(Us,b0,nu,Bl,Grr,Grs,Gss,Dh);      %% Update residual
rc = tensor3(Jc',1,Jc',rl);
nrc= sqrt( sum(sum(sum(M.*dA.*(rl.^2))))/vol );
tolc=0.1*nrc;
[Ec,iterc,res,lam_crs]=...
    pcg_lambda(rc,tolc,100,b0,nu,Mc,Qc,Blc,Grrc,Grsc,Gssc,Dc,dAc,ifnull);
El = tensor3(Jc,1,Jc,Ec);

Umg = Umg + El;                                  %% Add coarse-grid correction
```

- The output for this strategy is shown below, where we see the coarse iteration counts, `kc` and the rapid convergence of Chebyshev-accelerated p-multigrid (pMG). (pMG = coarsening by reducing local polynomial order, $p = N$.)
- The driver, `slide_table.m`, and supporting scripts are posted to the Relate page. (Note that the exact solution is *random*, so that tables are a bit different whenever this demo is run.)

```
                          Damped Jacobi, omega=2/3
 N    Nc     k         e_inf(k)        smoother            tol_c     kc
10     4     1         0.35707          0.87416          5.58806     18
10     4     2         0.27208          0.31388          1.57768      9
10     4     3         0.18824          0.25511          0.83471     13
10     4     4         0.15520          0.15888          0.54382      3
10     4     5         0.13856          0.14039          0.39838      2
10     4     6         0.12369          0.12547          0.31063      2
10     4     7         0.10579          0.11221          0.25203      6
10     4     8         0.09492          0.09576          0.21026      2
10     4     9         0.08524          0.08605          0.17690      2
10     4    10         0.07666          0.07742          0.15070      2
```

```
                        Optimized 4th-kind Chebyshev
 N    Nc     k         e_inf(k)        smoother            tol_c     kc
10     4     1         0.70037          0.75592          8.17176      9
10     4     2         0.12425          0.57576          1.30327     17
10     4     3         0.03994          0.09235          0.25363     15
10     4     4         0.02072          0.02760          0.06939     19
10     4     5         0.01275          0.01350          0.02363      6
10     4     6         0.00766          0.00810          0.01022     12
10     4     7         0.00461          0.00495          0.00514      6
10     4     8         0.00263          0.00298          0.00284     15
10     4     9         0.00156          0.00168          0.00162      6
10     4    10         0.00094          0.00100          0.00095     11
```

- In the tables, `smoother` refers to the error after the smoothing step, whereas `e_inf(k)` refers to the error after the kth pMG sweep.



- The figures above show the errors for a Chebyshev pMG case with $N = 10$ and $N_c = 4$. The initial error is in the top left and the error for iterations $k = 8$, 9, and 10 in the other panels.

- We see that the error decreases for $k = 8$ to 10, but the similarity of the error distributions indicates that there is significant potential to accelerate the process by projecting the solution at each iteration to avoid resolving the same problem.