Introduction to Multigrid Methods

Chapter 8: Elements of Multigrid Methods

Gustaf Söderlind

Numerical Analysis, Lund University

Textbooks: *A Multigrid Tutorial*, by William L Briggs. SIAM 1988

A First Course in the Numerical Analysis of Differential Equations, by Arieh Iserles. Cambridge 1996

Matrix-based multigrid: Theory and Applications, by Yair Shapira. Springer 2008

Multi-Grid Methods and Applications, by Wolfgang Hackbusch, 1985

1. Successive over-relaxation (SOR)

An attempt to speed up convergence in Gauss–Seidel

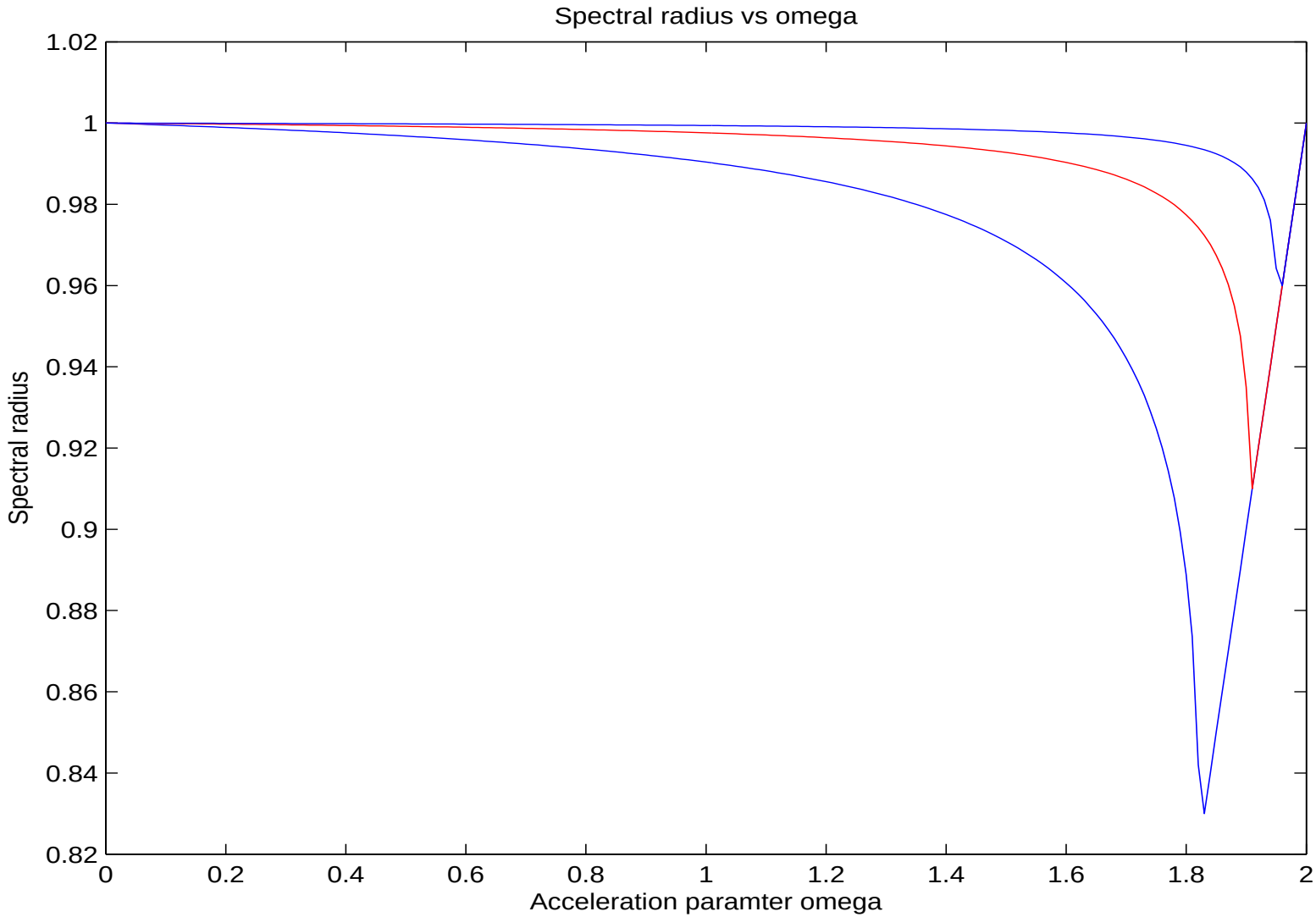
$$
\begin{aligned}
Du &= Du - \omega(Au - f) \\
Du &= Du - \omega(D - L - U)u + \omega f \\
(D - \omega L)u &= \big(\omega U + (1-\omega)D\big)u + \omega f \\
(D - \omega L)u^{m+1} &= \big(\omega U + (1-\omega)D\big)u^{m} + \omega f
\end{aligned}
$$

Iteration matrix $P_{SOR} = (D - \omega L)^{-1}\big((1-\omega)D + \omega U\big)$

Acceleration parameter $\omega = 1 \Rightarrow$ Gauss–Seidel method

Spectral radius of SOR matrix for $T_{\Delta x}$

$\rho[P_{SOR}(\omega)]$ for $N = 31, 63, 127$



Faster convergence is possible!

For 1D Poisson, the spectral radius is minimized at

$$\hat{\omega} = \frac{2}{1 + \sqrt{1 - \rho^2[P_J]}}$$

As $\rho[P_J] = \cos \pi \Delta x \approx 1 - \pi^2 \Delta x^2 / 2$ we have

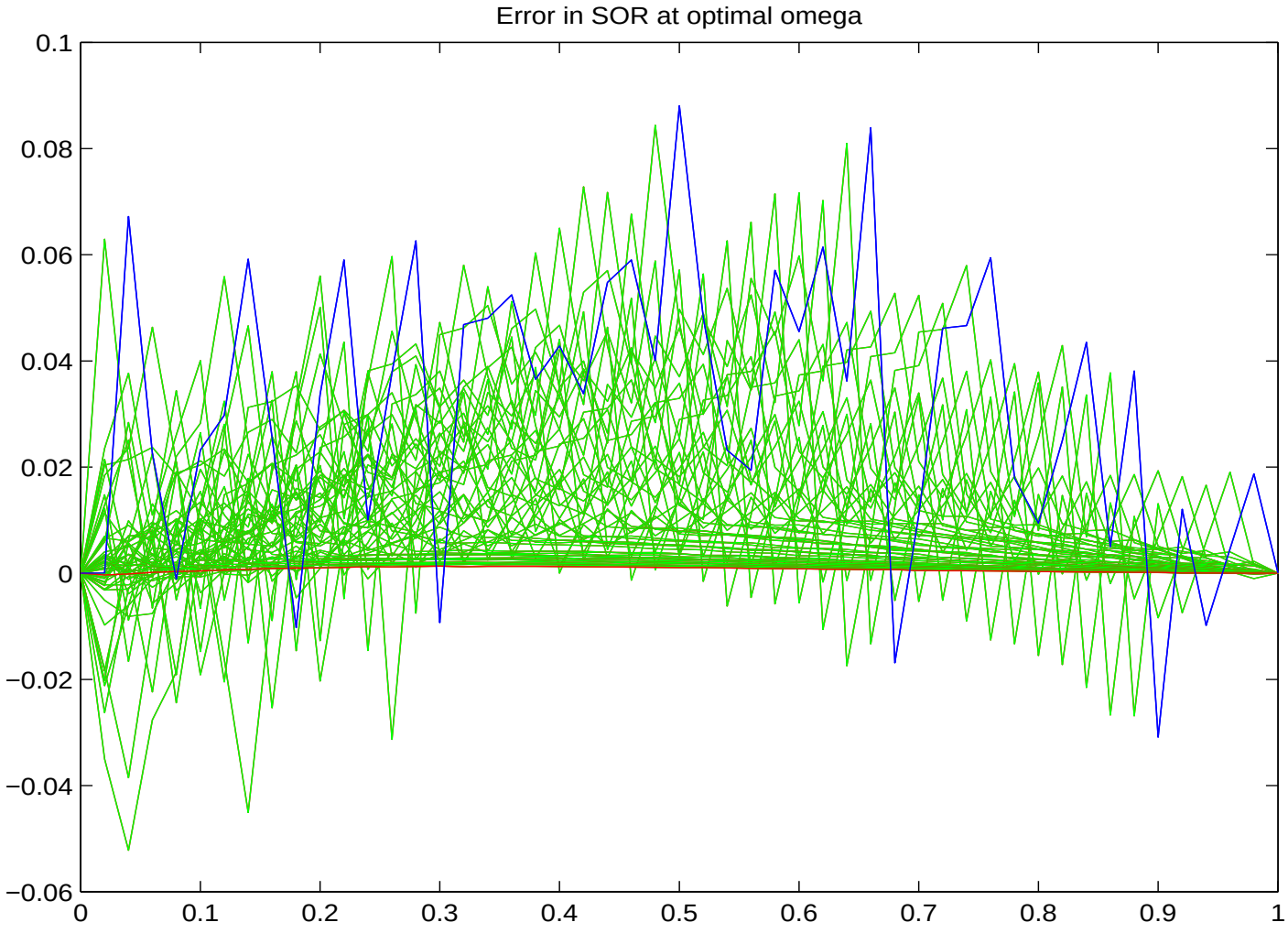
$$\hat{\omega} \approx 2 - 2\pi \Delta x$$

At this value the spectral radius of the SOR matrix is

$$\rho[P_{SOR}(\hat{\omega})] = \hat{\omega} - 1 \approx 1 - 2\pi \Delta x$$

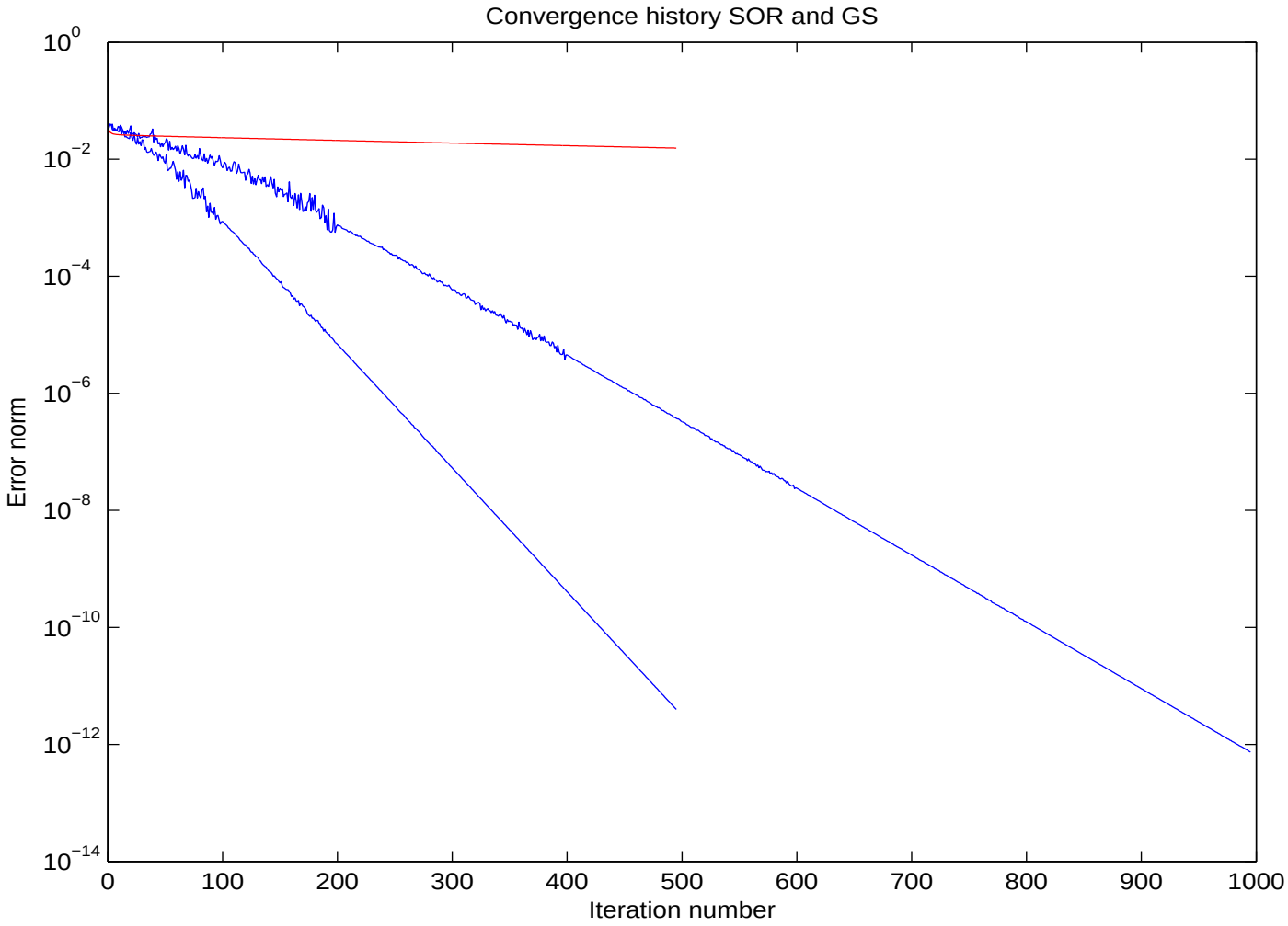
Much improved over Jacobi and Gauss–Seidel

SOR error sequence simulation



Note Error “transportation” in live demo!

SOR error sequence simulation, $N = 99, 199$



Note Unit error reduction requires only $\mathrm{O}(N)$ iterations

2. Symmetric SOR (SSOR)

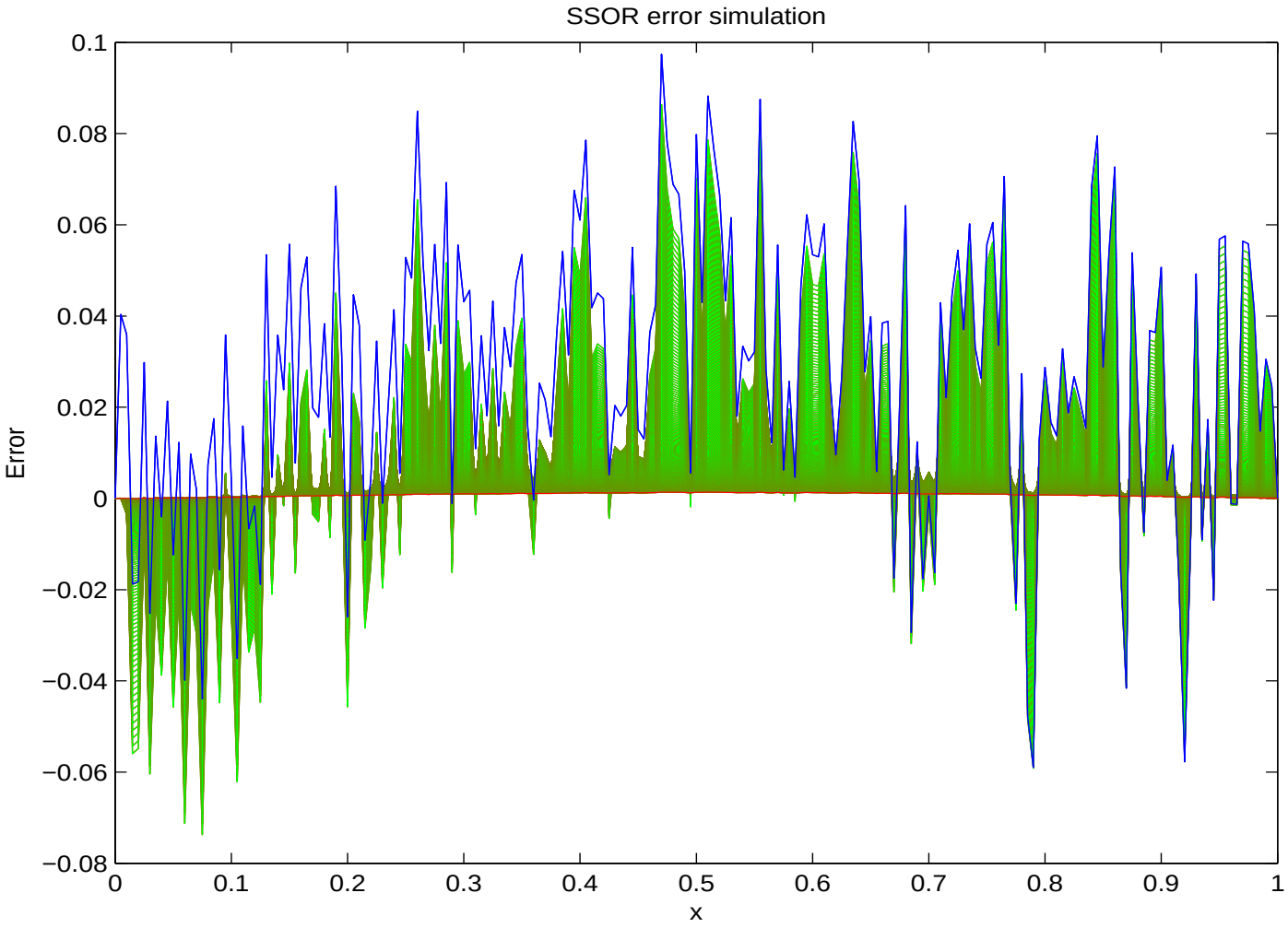
Neutralize “error transportation” to the left by alternating between forward and backward sweeps

$$(D-\omega L)u^{m+1} = \big(\omega U + (1-\omega)D\big)u^m + \omega f$$
$$(D-\omega U)u^{m+2} = \big(\omega L + (1-\omega)D\big)u^{m+1} + \omega f$$

Iteration matrix $P_{SORfwd} = (D-\omega L)^{-1}\big(\omega U + (1-\omega)D\big)$ and $P_{SORbwd} = (D-\omega U)^{-1}\big(\omega L + (1-\omega)D\big)$

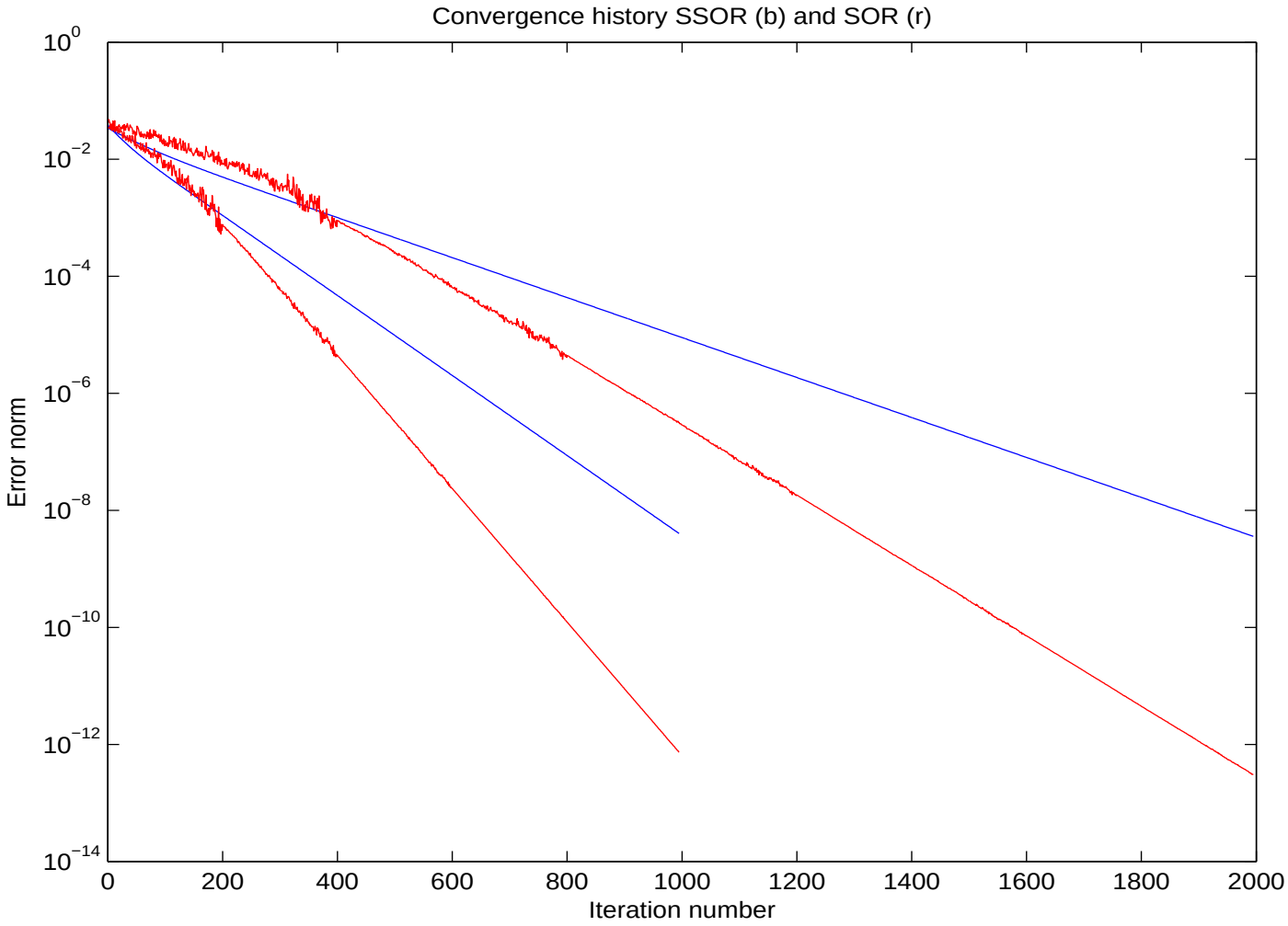
$$P_{SSOR} = P_{SORbwd}P_{SORfwd}$$

SSOR error sequence simulation, $N = 199$



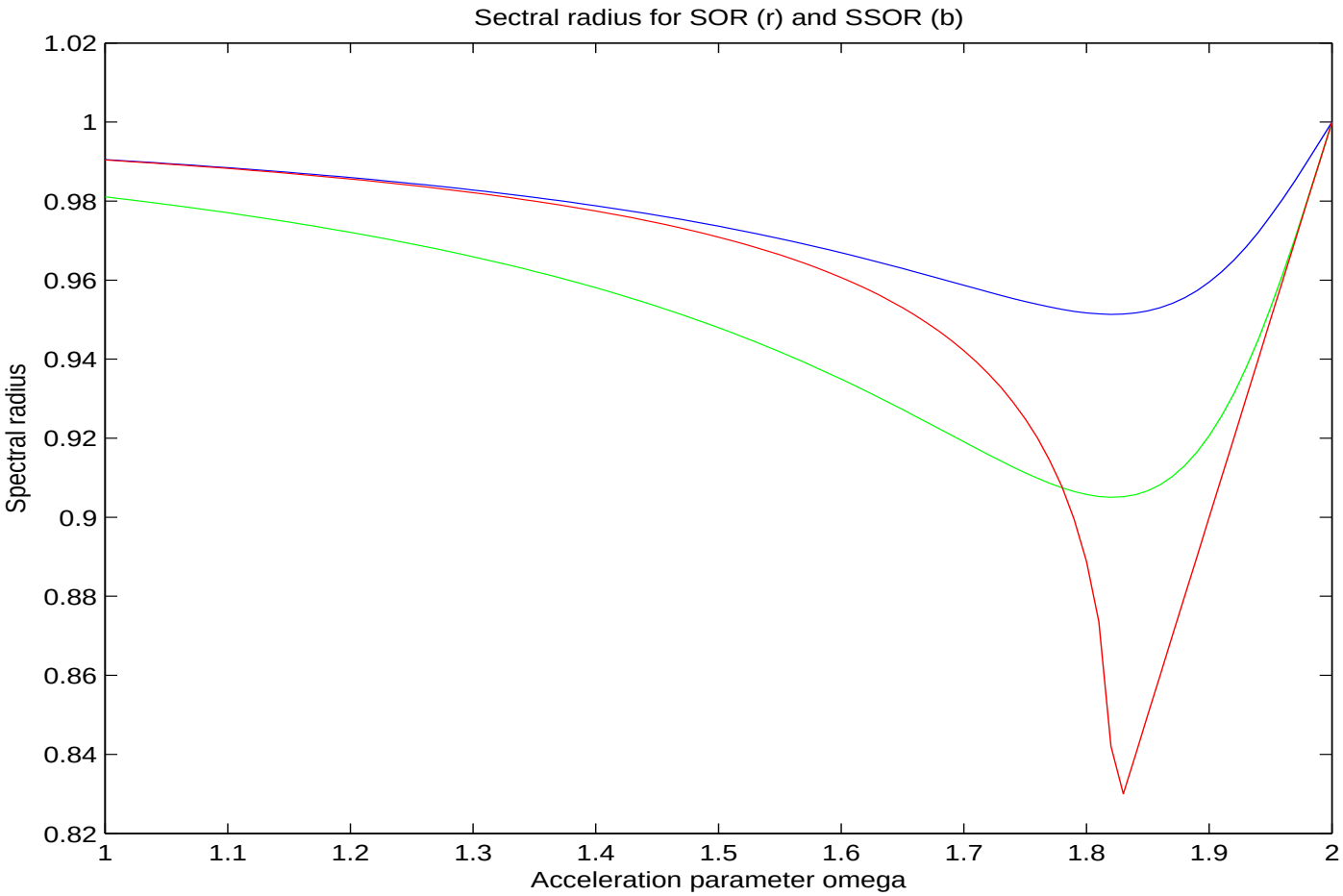
Note Error “transportation” eliminated

SSOR error sequence simulation, $N = 199,399$



Note Unit error reduction in $\mathrm{O}(N)$ iterations, but SOR faster

SOR and SSOR spectral radii vs ω



$\rho[P_{SOR}]$ $\sqrt{\rho[P_{SORbwd}P_{SORfwd}]}$ $\rho[P_{SORbwd}P_{SORfwd}]$

Note *Optimal acceleration parameter $\hat{\omega}$ does not move*

3. Error diffusion and the Jacobi method

For the damped Jacobi method the error recursion is

$$e^{m+1} = P_J(\omega) e^m$$

$$P_J(\omega) = (1-\omega) I + \omega P_J$$

Consider the *diffusion equation* $y_t = y_{xx}$ with homogeneous boundary conditions and 2nd order MOL semidiscretization

$$\dot{y} = T_{\Delta x} y$$

Use explicit Euler time–stepping to get

$$y^{m+1} = y^m + \Delta t \cdot T_{\Delta x} y^m$$

Jacobi and diffusion

Introduce the *Courant number* $\mu = \Delta t / \Delta x^2$ and the Toeplitz matrix

$$T = \begin{pmatrix} -2 & 1 & 0 & \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \\ & & 0 & 1 & -2 \end{pmatrix}$$

The recursion becomes

$$y^{m+1} = y^m + \mu \cdot T y^m$$

Jacobi and diffusion. . .

Split the matrix $T = D - L - U = -2I + S$ and note that

$$S = \begin{pmatrix} 0 & 1 & 0 & & \\ 1 & 0 & 1 & & \\ & & \ddots & & \\ & & 0 & 1 & 0 \end{pmatrix} = 2P_J$$

So the recursion can be written

$$y^{m+1} = y^m + \mu \cdot Ty^m = \left[(1-2\mu)I + 2\mu P_J\right] y^m = P_J(2\mu)y^m$$

Error diffusion in the Jacobi method

Solving diffusion equation $y_t = y_{xx}$ using Explcit Euler time stepping with finite difference MOL produces

$$y^{m+1} = P_J(2\mu)y^m$$

Solving 2pBVP $T_{\Delta x}u = f$ using damped Jacobi iterations produces the error recursion

$$e^{m+1} = P_J(\omega)e^m$$

The recursions are identical, with the interpretation twice Courant number 2μ = acceleration parameter ω

Errors in damped Jacobi "diffuse" according to $e_t = e_{xx}/2$

Discrete diffusion

Explicit time-stepping for diffusion is extremely slow

CFL condition $0 < \mu \leq 1/2$

The same thing will happen to Jacobi iterations

Convergence if $0 < \omega \leq 1$

After many steps/iterations *not on the CLF limit*, the solution will approach the low frequency mode $\sin \pi x$, while *high frequencies are efficiently damped*

Recall What is a high frequency is a grid property

Side remark: The dynamic/static connection

If we want to solve the (dynamic) diffusion problem

$$y_t = y_{xx} - f(x)$$

its (static) stationary solution solves the 2pBVP

$$y'' = f$$

When we try to solve the latter problem, *every iterative method introduces (discrete) pseudo–dynamics*

$$y^{m+1} = y^m + \omega M r^m$$

Because r depends on y, this is the Explicit Euler method for $\dot{y} = Mr(y)$ with step size ω

Summary of SOR methods

- Jacobi is *diffusive* in character, and Gauss–Seidel is *hyperbolic/diffusive*. Both require $\mathrm{O}(N^2)$ iterations for a unit error reduction
- SOR at optimal acceleration is largely *hyperbolic* in character and only requires $\mathrm{O}(N)$ iterations for a unit error reduction
- Compare $\Delta t/\Delta x^2$ and $\Delta t/\Delta x$ CFL conditions
- SOR "transports" errors in space "accross boundary" with only moderate damping along the way
- SSOR eliminates error "transportation" but is slower

4. The Multigrid idea. Basic tools

To introduce multigrid, we return to damped Jacobi

$$r^m \leftarrow T_{\Delta x} u^m - f$$

$$u^{m+1} \leftarrow u^m - \omega D^{-1} r^m$$

Theorem *Let the Jacobi matrix have eigenvalues and eigenvectors given by* $P_J v_k = \lambda_k v_k$. *Then the eigenvectors of* $P_J(\omega) = (1-\omega)I + \omega P_J$ *are* v_k *and the eigenvalues are*

$$\lambda_k(\omega) = 1 + \omega(\lambda_k - 1)$$

Note The eigenvectors coincide with those of $T_{\Delta x}$

Error expansion

For the error recursion (power iteration)

$$e^m = P_J^m(\omega)e^0$$

let $e^0 = \sum_1^N \alpha_k v_k$. Then

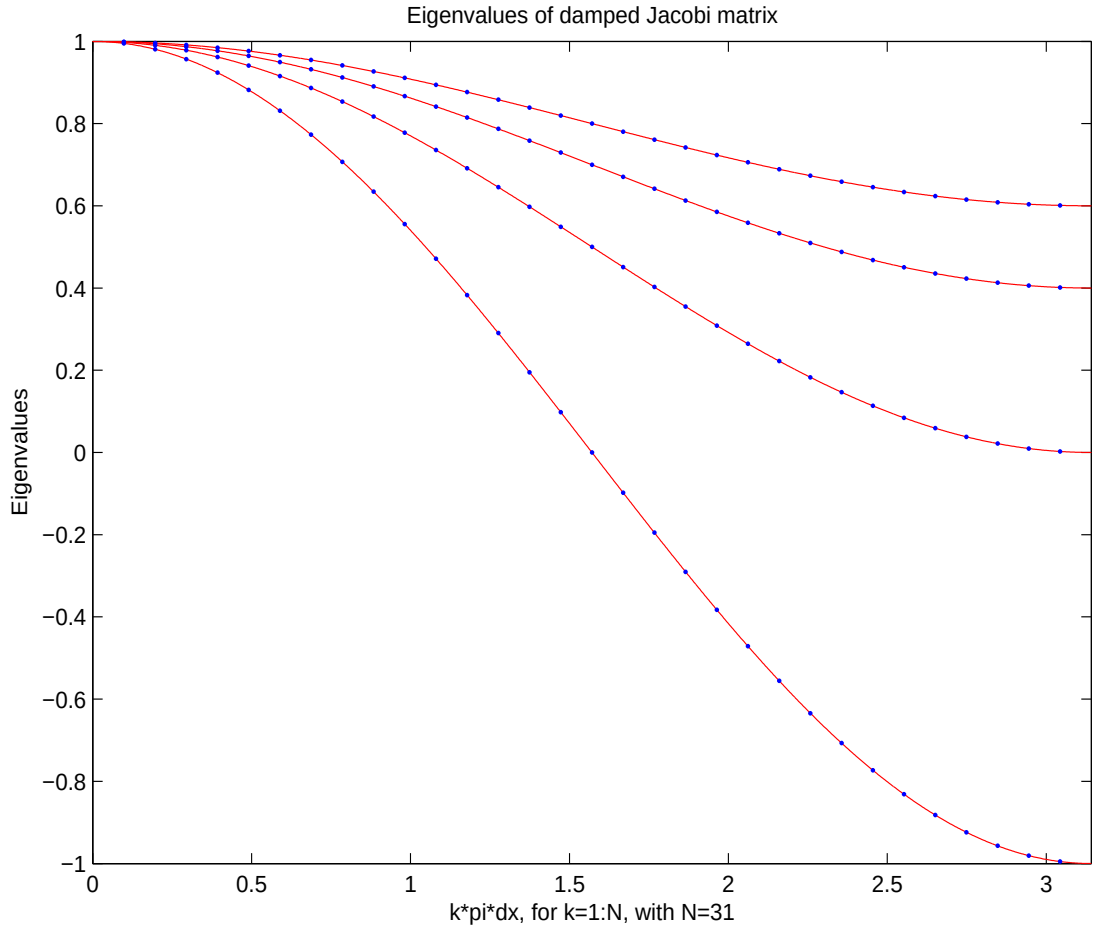
$$e^m = \sum_{k=1}^{N} \alpha_k \lambda_k^m(\omega) v_k$$

So the reduction of the k^{th} error mode is governed by the eigenvalue $\lambda_k(\omega)$ and can in part be controlled by ω

We can *make the error smooth* by choosing $\omega \in [2/3, 1/2]$

Eigenvalue locations of $P_J(\omega)$

Eigenvalues for $N = 31$ and $\omega = 1, 0.5, 0.3, 0.2$



High frequency damping can be controlled by ω

Solution, Error and Residual

Given a linear system $T_{\Delta x} u = f$ construct sequence $u^m \to u$

Now recall the following definitions

The Good *The solution is defined by* $T_{\Delta x} u = f$

The Bad *The error is defined by* $e^m = u^m - u$

The Ugly *The residual is defined by* $r^m = T_{\Delta x} u^m - f$

$$T_{\Delta x} e^m = r^m$$

Error (defect) correction

Note that $u = u^m - e^m$. If we can estimate the error, $\tilde{e} \approx e^m$, then

$$\tilde{u} \leftarrow u^m - \tilde{e}$$

is an improved solution, so we can put $u^{m+1} \leftarrow u^m - \tilde{e}$

Question *Is it simpler to solve* $T_{\Delta x} e^m = r^m$ *than* $T_{\Delta x} u = f$*?*

Answer ***YES!*** *Because we only need moderate accuracy, solve the residual–error equation on coarse grid*

This will still recover the persistent low frequency modes!

Error and residual

Recall

$$e^m = \sum_{k=1}^{N} \alpha_k \lambda_k^m(\omega) v_k$$

with residual

$$r^m = T_{\Delta x} e^m = \sum_{k=1}^{N} \alpha_k \lambda_k^m(\omega) \kappa_k v_k$$

where κ_k is an eigenvalue of $T_{\Delta x}$

Note As $\kappa_k = -\frac{4}{\Delta x^2} \sin^2 \frac{k\pi\Delta x}{2}$ the residual is *huge*

Error and residual...

Taking $\omega \in [2/3, 1/2]$ means that the error

$$e^m = \sum_{k=1}^{N} \alpha_k \lambda_k^m(\omega) v_k$$

is *smooth*, as only low frequency modes ($\lambda_k \approx 1$) present

The residual is usually much *less smooth* because *large* κ_k offset the benefit of small coefficients α_k

Solve $T_{\Delta x} e^m = r^m$ on a *coarse grid*

This still recovers low modes and converges faster

Example. Fine grid, coarse grid

Fine grid with an *odd number N of internal grid points*

$$\bar{\Omega}_h = \texttt{linspace(0,1,N+2)}$$
$$h = 1/(N+1)$$

Coarse grid with $(N-1)/2$ *internal grid points*

$$\bar{\Omega}_H = \texttt{linspace(0,1,(N+3)/2)}$$
$$H = 2/(N+1)$$

Mesh widths (step size) h and H, respectively

Grid functions (vectors)

$v_h : \Omega_h \to \mathbb{R}$ is a *grid function* with N components

It is an N–vector (excluding boundary points and data)

Compare a continuous function $u : [0,1] \to \mathbb{R}$

Note $u(\Omega_h)$ *is the restriction of* u *to the grid* Ω_h

Let $f_h \equiv f(\Omega_h)$. If $u'' = f$ and $T_h v_h = f_h$, the *error* is

$$e_h = u(\Omega_h) - v_h$$

Both the solution and the error are grid functions

Downsampling. From fine grid to coarse

Example $N_h = 7$ internal points on fine grid Ω_h map to $N_H = 3$ internal points on coarse grid via *restriction map*

$$v_H = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} v_h = I_h^H v_h$$

The restriction drops every second element. An alternative is a *smoothing map*, also known as a *weighted restriction*

$$v_H = \begin{pmatrix} 1/4 & 1/2 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/4 & 1/2 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1/4 \end{pmatrix} v_h$$

Smoothing map = plain restriction $\circ$ filter = $I_h^H \cdot F_\pi$

$$\begin{pmatrix} 1/4 & 1/2 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/4 & 1/2 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1/4 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \frac{1}{4} \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

Mapping a fine grid function to a coarse

Example Sometimes one *includes the boundary points* $N_h = 3$ internal points on fine grid $\bar{\Omega}_h$ map to $N_H = 1$ internal point on coarse grid $\bar{\Omega}_H$ via *restriction map*

$$\bar{v}_H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \bar{v}_h = \bar{I}_h^H \bar{v}_h$$

A *smoothing* counterpart is (with unchanged boundaries)

$$v_H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 1/2 & 1/4 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} v_h$$

What is a digital filter?

Example To filter out HF, *repeated averaging* can be used

$$u = \frac{1}{4}\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} v = F_\pi \cdot v$$

Then u is *smoother* than v, F_π is a 2^{nd} order lowpass filter

Note The filter F_π *is Toeplitz or a discrete convolution*, and

$$F_\pi = (1 - \frac{1}{2})I + \frac{1}{2}P_J = P_J(1/2)$$

Same as $\omega = 1/2$ damped Jacobi iteration matrix for $T_{\Delta x}$

Filters. Analogue and Digital

An *analogue* linear filter is an integral operator $F : v \mapsto u$

$$u = Fv \quad \Leftrightarrow \quad u(x) = \int_0^1 f(x,y)v(y)\,\mathrm{d}y$$

Its discrete counterpart is a *digital* filter

$$u = Fv \quad \Leftrightarrow \quad u_i = \sum_{j=1}^{N} f_{i,j}v_j$$

The operator F (as defined by the kernel function $f(x,y)$ and the matrix $\{f_{i,j}\}$) determine the filter characteristics

Convolutions and Toeplitz filters

A *convolution filter* is an integral operator $F : v \mapsto u$

$$u = Fv \quad \Leftrightarrow \quad u(x) = \int_0^1 f(x-y)\, v(y)\, \mathrm{d}y$$

A *discrete convolution* is of the form

$$u = Fv \quad \Leftrightarrow \quad u_i = \sum_{j=1}^{N} f_{i-j} v_j$$

Note F is a *Toeplitz matrix*, because if $i - j = \mathrm{const}$ then $f_{i,j} = f_{i-j} = \mathrm{const.}$ Elements are diagonalwise constant

Theorem *A Toeplitz filter is a discrete convolution*

Oversampling. From coarse grid to fine

Note The maps I_h^H do not have inverses (information loss)

I_H^h *is an interpolation operator generating new elements*

Left and right inverses

$$I_H^h \cdot I_h^H \neq I$$

$$I_h^H \cdot I_H^h = I$$

The second identity holds if I_h^H is a plain restriction, but not if it is a smoothing operator

Prolongation through linear interpolation

Example The interpolation operator I_H^h can be defined by piecewise linear interpolation

$$v_{j+1/2} \leftarrow (v_j + v_{j+1})/2$$

This defines a *prolongation of the grid function* by inserting an interpolated value between previous coarse grid points

Other types of interpolation, such as splines, are possible but are often too expensive and of little advantage

Choose linear interpolation with linear FEM basis functions

Matrix representation of the prolongation

Example Prolongation from 3 internal points to 7

$$v_h = \begin{pmatrix} 1/2 & 0 & 0 \\ 1 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 0 & 0 & 1/2 \end{pmatrix} v_H = I_H^h v_H$$

Compare using smoothing restriction: $I_H^h = 2F_\pi (I_h^H)^{\mathrm{T}}$

Tools for restriction and prolongation

Two tools needed, the *Toeplitz filter* $F_\pi = \frac{1}{4}\,\mathrm{tridiag}(1 \quad 2 \quad 1)$
and the *plain restriction* ("injection")

$$I_h^H = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

General restriction map $R_h^H = I_h^H F_\pi$ (filtering optional)

Linear interpolation prolongation map $P_H^h = 2F_\pi (I_h^H)^{\mathrm{T}}$

5. A simple two-grid iteration

Solve $T_h u_h = f_h$ iteratively using the following algorithm

1. Run one damped $\omega = 2/3$ Jacobi iteration on fine grid

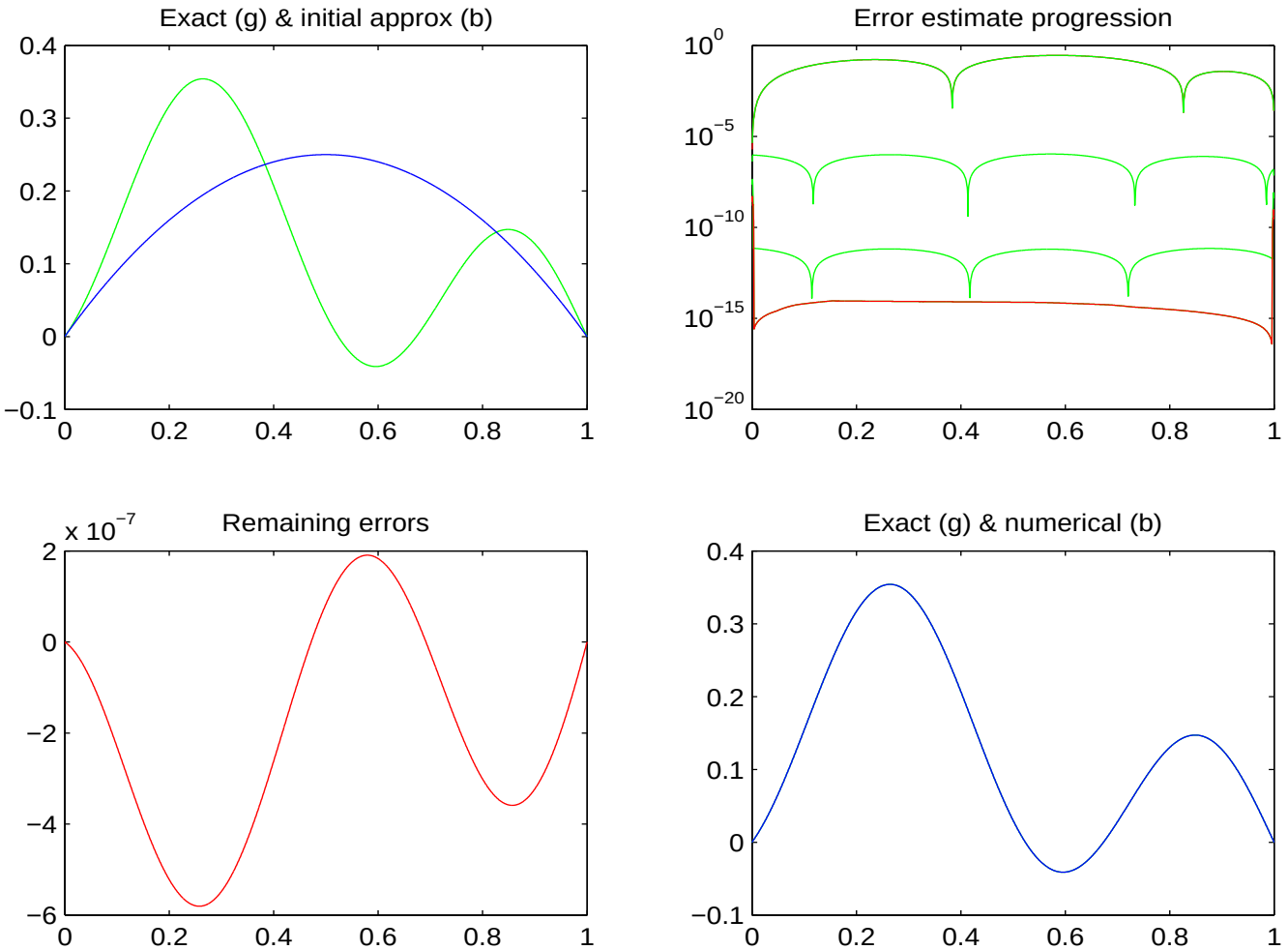
$$u_h^{m+1} \leftarrow u_h^m - \omega D_h^{-1} r_h^m \qquad m = 0 : M-1$$

2. Compute residual and restrict $r_H \leftarrow I_h^H F_\pi (T_h u_h^M - f_h)$
3. Solve error equation $T_H e_H = r_H$
4. Prolong $e_h^M \leftarrow I_H^h e_H$
5. Correct $u_h^{M+1} \leftarrow u_h^M - e_h^M$
6. Run one damped Jacobi iteration on fine grid
7. Repeat from 1 until convergence

MATLAB code segment: the two-grid V step

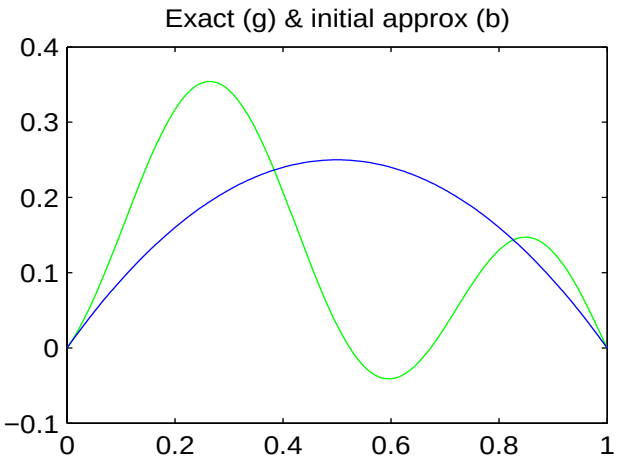
```
rf = Tdx*v - f;              % Compute residual
v = v - omega*Ddx\rf;        % Pre-smoothing Jacobi

rf = Tdx*v - f;              % Compute residual
rf = lowpass(rf);            % Optional LP filter

rc = FMGrestrict(rf);        % Restrict to coarse grid
ec = Tdxc\rc;                % Solve error equation
ef = FMGprolong(ec);         % Prolong to fine grid

v = v - ef;                  % Correct: remove error

rf = Tdx*v - f;              % Compute residual
v = v - omega*Ddx\rf;        % Post-smoothing Jacobi
```

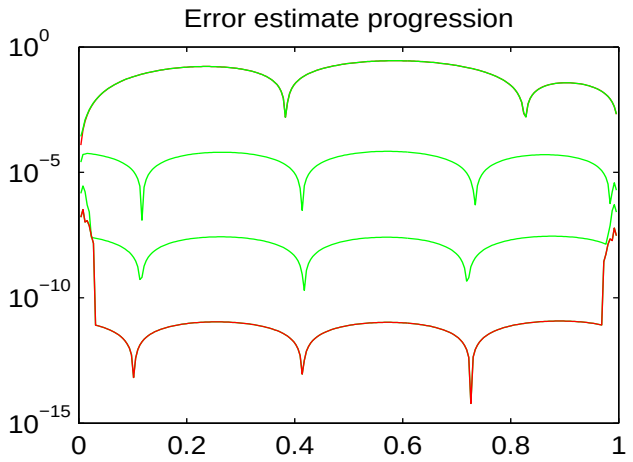
Numerical test: Two-grid method, $N = 2048$

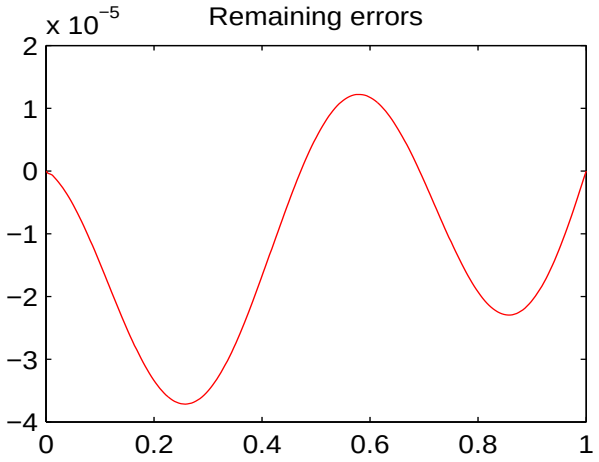


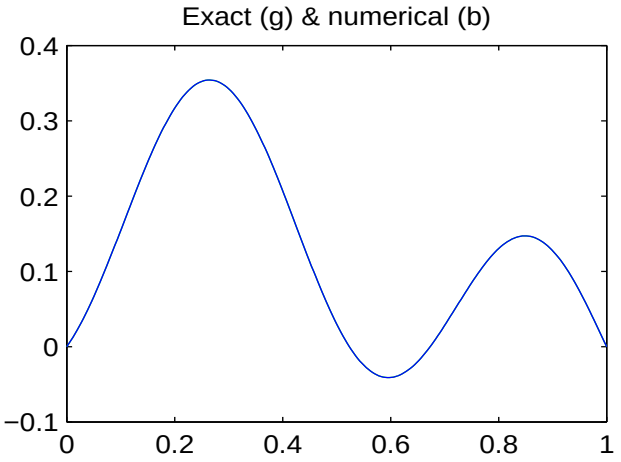
4 V-cycles on $u'' = f$ on on fine grid with 2048 points, error estimation on 1024, single $P_J(1/2)$ pre- and post-smoothing

Numerical test: Two-grid method, $N = 256$



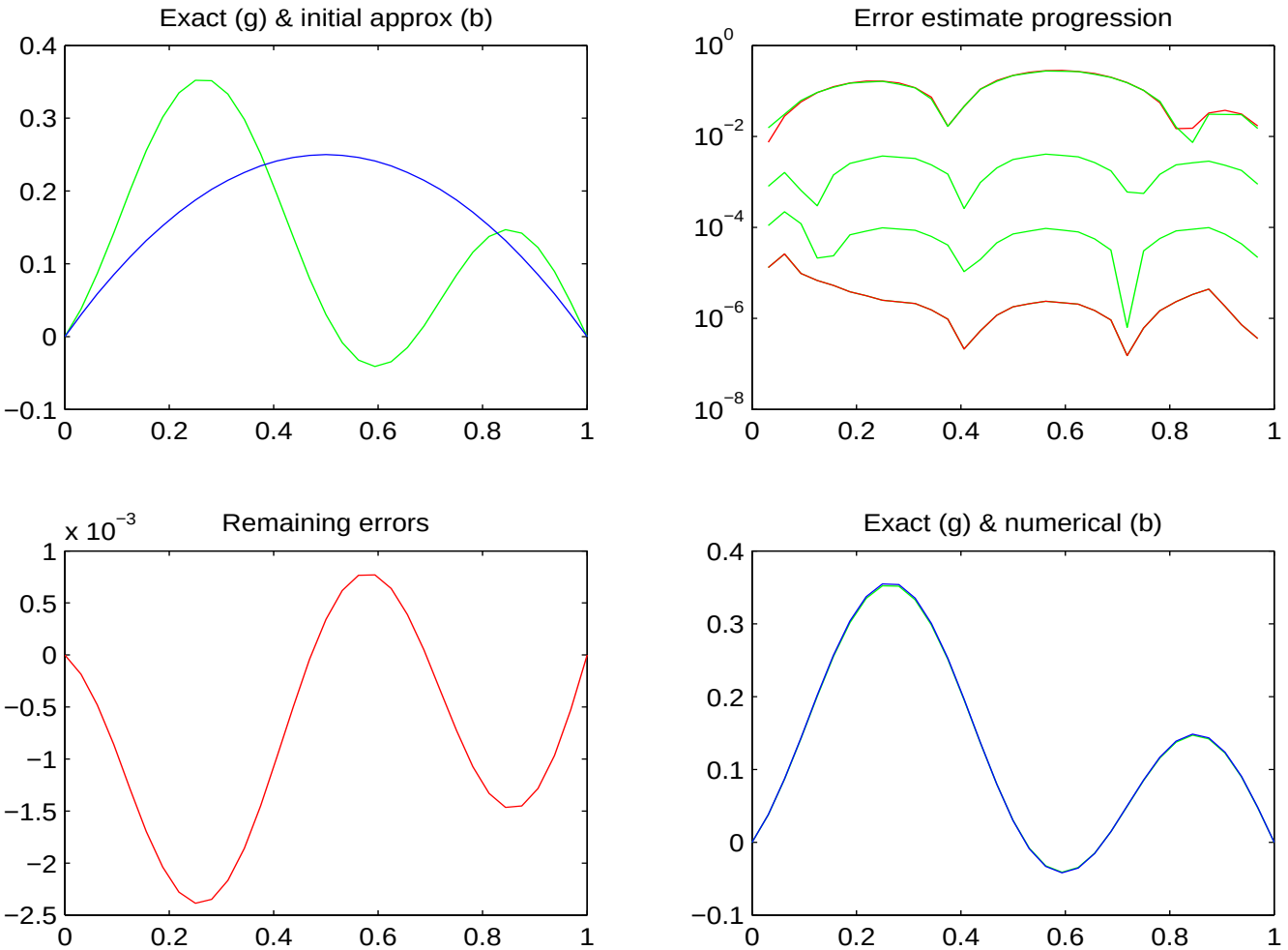






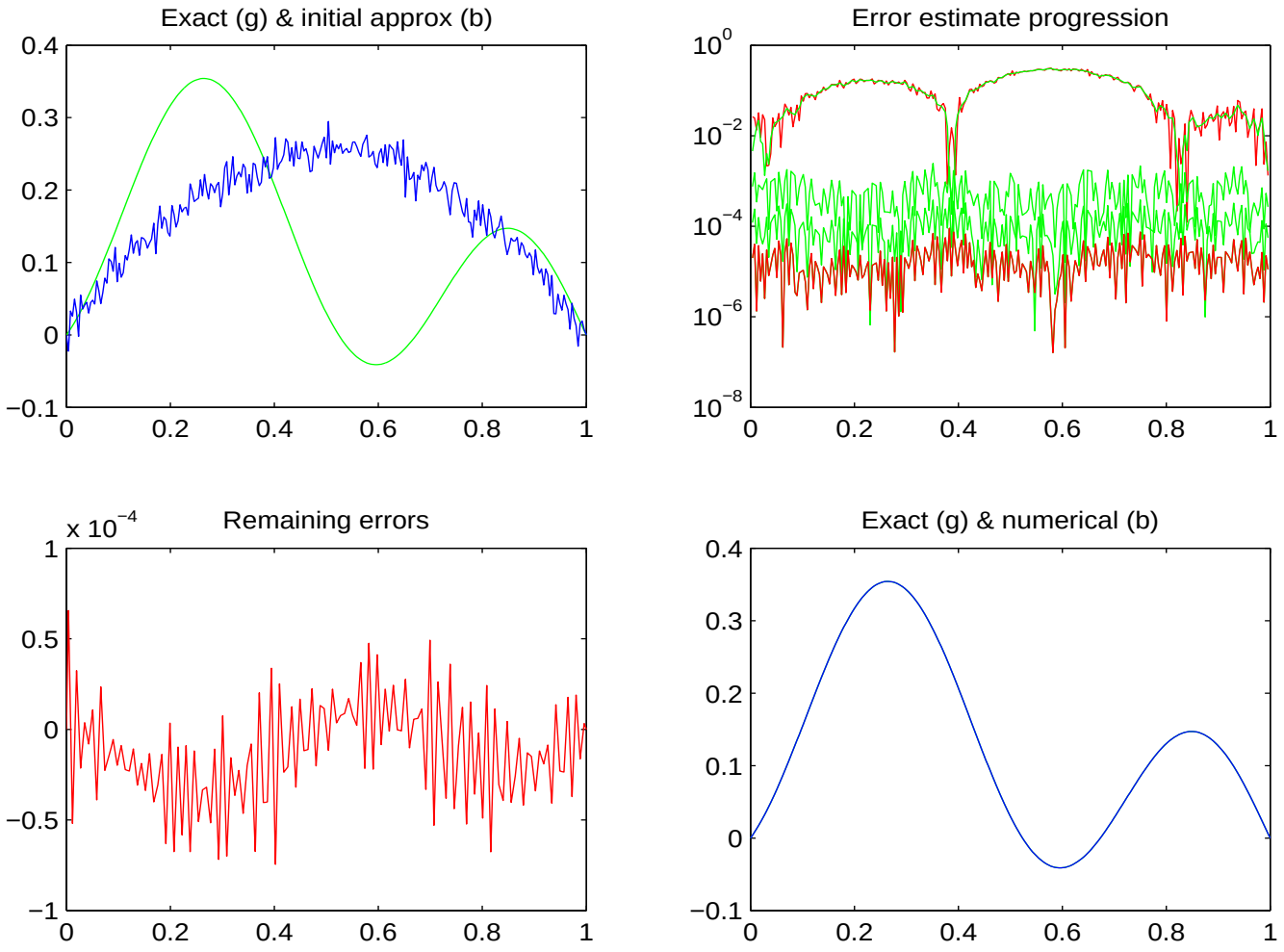
4 V-cycles on $u'' = f$ on on fine grid with 256 points, error estimation on 128, single $P_J(1/2)$ pre- and post-smoothing

Numerical test: Two-grid method, $N = 32$



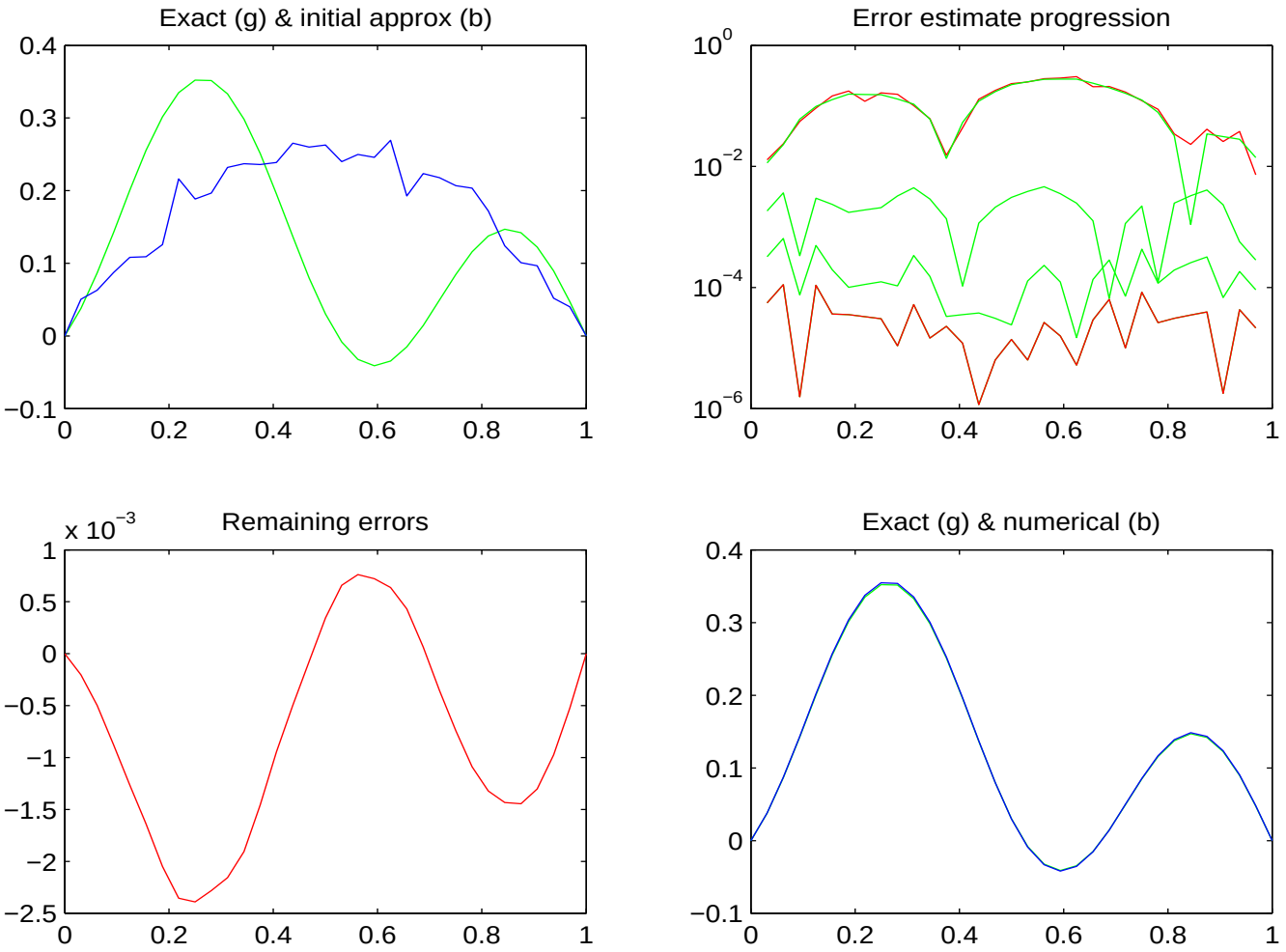
4 V-cycles on $u'' = f$ on on fine grid with 32 points, error estimation on 16, single $P_J(1/2)$ pre- and post-smoothing

Numerical test: Two-grid method, $N = 256$



4 V-cycles on $u'' = f$ on on fine grid with 256 points, error estimation on 128, with initial data contaminated by noise

Numerical test: Two-grid method, $N = 32$



4 V-cycles on $u'' = f$ on on fine grid with 32 points, error estimation on 16, with initial data contaminated by noise

Observations in the numerical tests

- *One can almost use the same number of iterations*
- *By and large, convergence is unaffected by* $\Delta x \to 0$
- *One can solve the problem to high accuracy so that only the global error remains*
- *The basic iterative method is not crucial but must be smoothing*
- *Large initial errors are not a problem*
- *The method is robust even with respect to noise*

Why is the Multigrid Method faster?

All iterative methods work with the residual

$$y^{m+1} = y^m + \omega M r^m$$

The difference is that *multigrid estimates the error*

- *The residual is large and nonsmooth*
- *The error is moderate and smooth*
- *Basic residual-reducing iterative methods typically fail to eliminate low frequency error components*
- Therefore *only error estimating methods can be fast*

6. Multigrid on nested grids. The V cycle

We want to solve $T_K u_K = f_K$ on a fine grid Ω_K with mesh width h_K

Construct an embeddeding of grids $\Omega_0 \subset \Omega_1 \subset \cdots \subset \Omega_K$ with mesh widths $h_0 > h_1 > \cdots > h_K$

On *regular grids* one typically takes $h_k = 2h_{k+1}$ to make restrictions and prolongations simple

We want to formulate multigrid methods for solving

$$T_K u_K = f_K$$

on Ω_K using all available grids

Assumptions

- *There are restriction operators* $I_k^{k-1} : \Omega_k \to \Omega_{k-1}$
- *There are prolongation operators* $I_k^{k+1} : \Omega_k \to \Omega_{k+1}$
- *The restrictions and prolongations must be very fast and run in* $\mathrm{O}(N)$ *operations*
- *There is an iterative method with the smoothing property on every grid* Ω_k*, i.e., it must attenuate high frequencies by a constant factor independently of* h_k
- $T_0 u_0 = f_0$ *can be solved fast to provide a crude estimate* u_0

The operators $\{T_k\}$

Different operators T_k are needed on the different grids
Standard approach: if possible use same discretization

It may be “difficult” to construct the T_k

Use restriction and prolongation operators!

Definition *The operator triple* $(T_k, R_k^{k-1}, P_{k-1}^k)$ *satisfies the Galerkin condition if*

$$T_{k-1} = R_k^{k-1} T_k P_{k-1}^k$$

where R_k^{k-1} *is the restriction and* P_{k-1}^k *is the prolongation*

The V-cycle. Sweep through the grids

Obtain a *starting approximation* by solving the problem on a very coarse grid, typically Ω_0. Solving means by *direct method*, $N = 1000$ could be typical. Prolong to finest grid

A "V" sweep Iterate from finest grid on successively coarser grids down to Ω_0, then back to Ω_K on successively finer grids

Repeat V-sweeps to form a ***"V–cycle"*** and iterate until convergence

Visit enough grids (many) to suppress the entire spectrum of frequencies, coarse grids for slow, fine grids for fast

A full, recursive multigrid V-cycle

1. Procedure $v := FMGV(v_k, T_k, f_k)$
2. If $k = 0$ return $v \leftarrow T_0 \backslash f_0$ and STOP else:
3. Pre-smoothing $v_k \leftarrow v_k - \omega D_k^{-1}(T_k v_k - f_k)$
4. Restrict residual to coarse grid $r_{k-1} \leftarrow I_k^{k-1}(T_k v_k - f_k)$
5. Recursive error estimation $e_{k-1} \leftarrow FMGV(0, T_{k-1}, r_{k-1})$
6. Prolong error to fine grid $e_k \leftarrow I_{k-1}^k e_{k-1}$
7. Correct by removing error $v_k \leftarrow v_k - e_k$
8. Post-smoothing $v_k \leftarrow v_k - \omega D_k^{-1}(T_k v_k - f_k)$
9. Output $v \leftarrow v_k$

Starting procedure

If direct solution $v_0 = T_0 \backslash f_0$ is affordable, then obtain a first approximation on this grid by direct solver

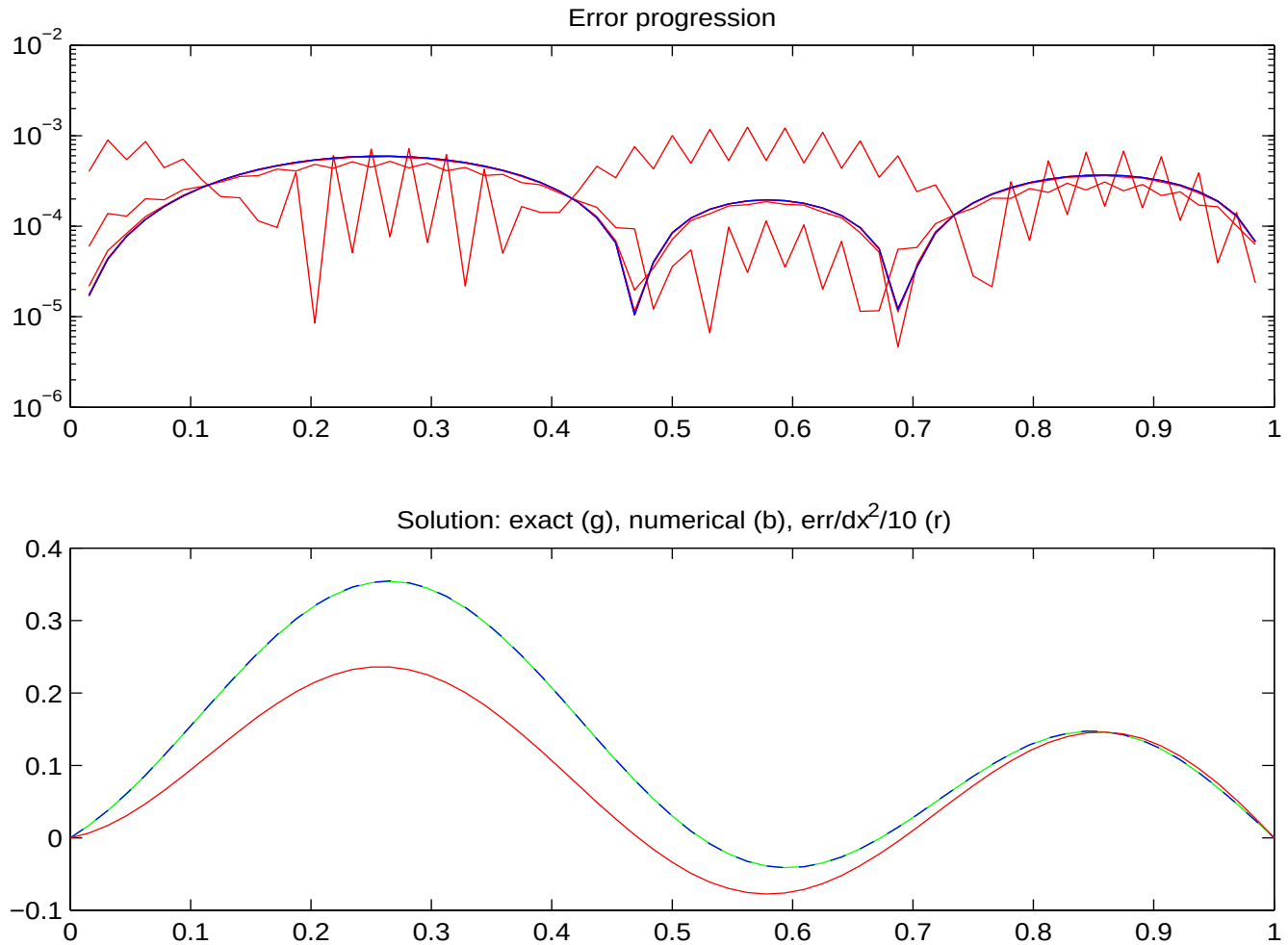
Sweep down on successively finer grids:

1. Compute $v_0 = T_0 \backslash f_0$
2. Prolong to next finer fine grid $v_k \leftarrow I_{k-1}^k v_{k-1}$
3. Compute residual $r_k = T_k v_k - f_k$
4. Smoothing iteration(s) $v_k \leftarrow v_k - \omega D_k^{-1} r_k$
5. Prolong to next grid &c. until first v_K has been obtained

MATLAB code segment. The recursive V cycle

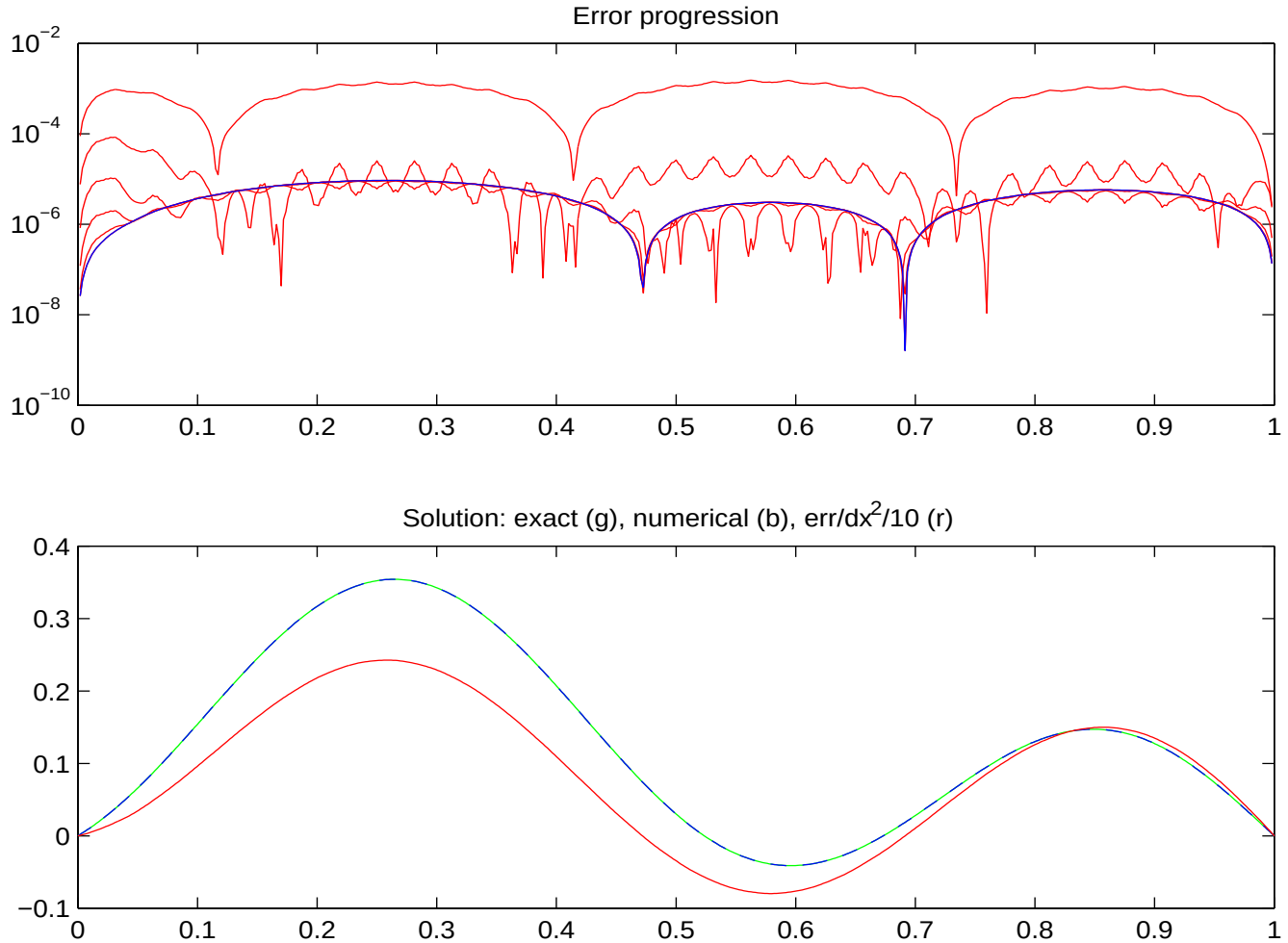
```
rf = Tdx*v - f;          % Compute residual
v = v - omega*Ddx\rf;    % Pre-smoothing Jacobi

rf = Tdx*v - f;          % Compute residual
rf = lowpass(rf);        % Optional LP filter

rc = FMGrestrict(rf);    % Restrict to coarse
ec = FMGV(0,T2dx,rc);    % Solve error equation
ef = FMGprolong(ec);     % Prolong to fine grid

v = v - ef;              % Correct: remove error

rf = Tdx*v - f;          % Compute residual
v = v - omega*Ddx\rf;    % Post-smoothing Jacobi
```

Numerical test: Full MG V-cycles, $N = 63$



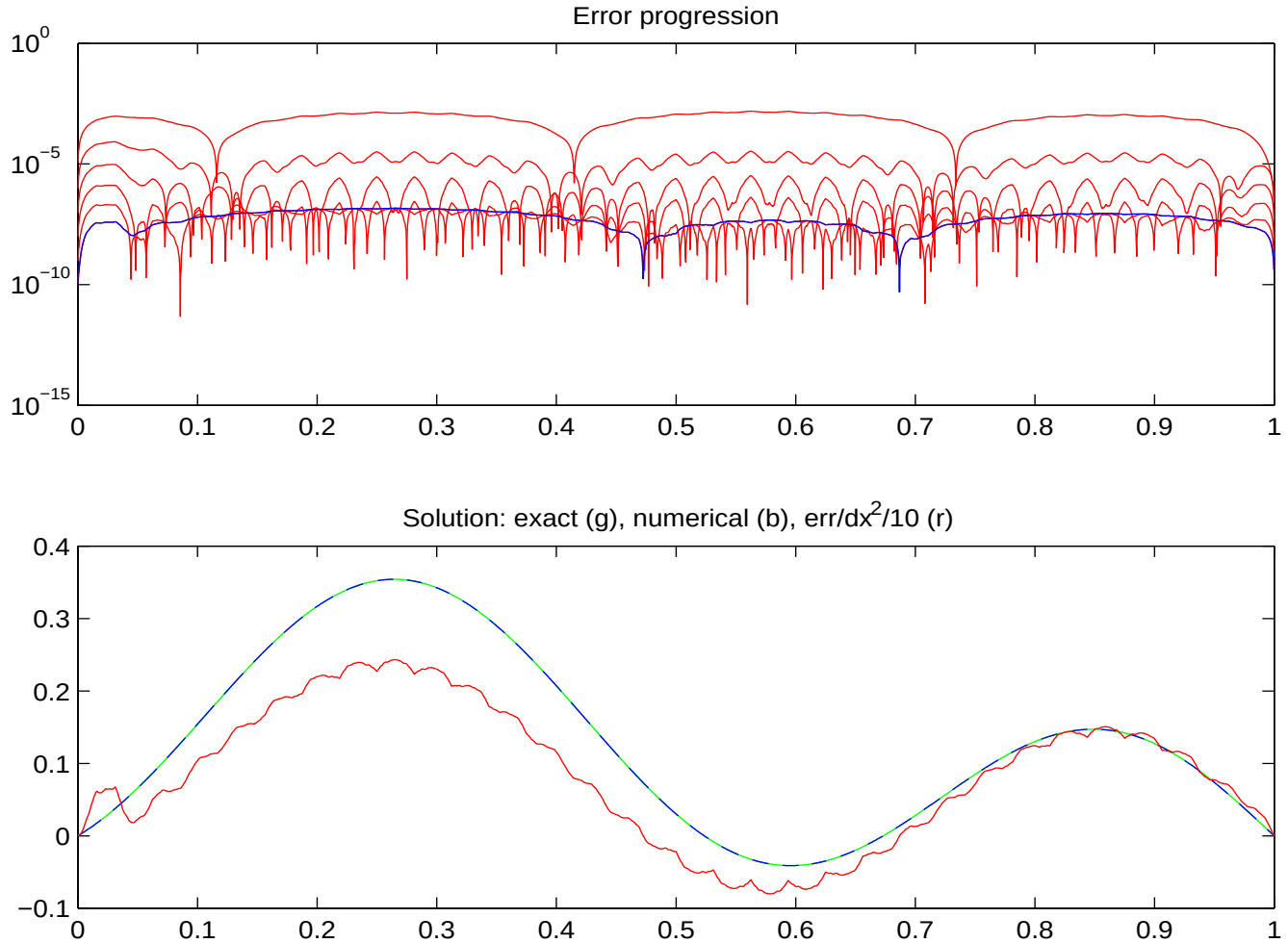
6 V-cycles on $u'' = f$ on on fine grid with 63 points, coarsest grid with 31 points and Jacobi $\omega = 2/3$

Numerical test: Full MG V-cycles, $N = 511$



6 V-cycles on $u'' = f$ on on fine grid with 511 points, coarsest grid with 31 points and Jacobi $\omega = 2/3$

Numerical test: Full MG V-cycles, $N = 4095$



6 V-cycles on $u'' = f$ on on fine grid with 4095 points, coarsest grid with 31 points and Jacobi $\omega = 2/3$

Observations in the numerical tests

- *By and large, multigrid iteration convergence is unaffected by* $\Delta x \to 0$
- *One can solve the problem to high accuracy so that only the global error remains*
- *Within 6 V-cycles the global error is reached*
- *Solution accuracy is* $\mathrm{O}(\Delta x^2)$
- *Large initial errors are not a problem*
- *The method is robust even with respect to noise, but more V-cycles may be required for large* N

7. Efficiency. Memory and execution time

Let the K^{th} grid have N internal points

Then grid $K-1$ has $(N+1)/2 - 1 < N/2$ internal points

Total storage requirement is therefore

$$S = N \cdot (1 + 2^{-1} + \cdots + 2^{-K}) < 2N$$

So, independently of the # of grids (recursion depths) *storage does not exceed that of using a single finer mesh*

Memory requirements in d dimensions

Let the K^{th} grid have N^d points (curse of dimensions)

Then grid $K-1$ has $(N+1)^d/2 - 1 < (N/2)^d$ points

Total storage requirement in d dimensions is

$$S_d < N^d \cdot (1 + 2^{-d} + \cdots + 2^{-dK}) \approx N^d$$

$$S_1 < 2N \qquad S_2 < \frac{4N^2}{3} \qquad S_3 < \frac{8N^3}{7}$$

So, for once, the *curse of dimensions works our way!*
Multigrid is always cheaper than using a single finer grid

Execution time “in theory”

Work unit U_K is the cost of one sweep on the fine K^{th} grid
$W_0 \sim 2^{-dK} N^{2d}$ is cost for direct solution on coarsest grid

Neglect restriction and prolongation costs

Cost W of V cycle with one pre- and one post-smoothing

$$W_d < 2U_K \cdot (1 + 2^{-d} + \cdots + 2^{-dK}) + 2W_0 \approx 2U_K + 2W_0$$

$$W_1 < 4U_K + 2W_0\,; \quad W_2 < \frac{8U_K}{3} + 2W_0\,; \quad W_3 < \frac{16U_K}{7} + 2W_0$$

So, *execution time does not exceed that of a single sweep on the next finer grid*

So it's cheap, what accuracy do we get?

Solving $T_{\Delta x} u_{\Delta x} = f$ we obtain $v_{\Delta x}$ with

$$\begin{aligned} \textit{Algebraic error} \quad & e_{\Delta x} = u_{\Delta x} - v_{\Delta x} \\ \textit{Global error} \quad & g_{\Delta x} = u_{\Delta x} - u(\Omega_{\Delta x}) \\ \textit{Total error} \quad & \delta_{\Delta x} = v_{\Delta x} - u(\Omega_{\Delta x}) \end{aligned}$$

Note that $\delta_{\Delta x} = g_{\Delta x} - e_{\Delta x}$, therefore

$$\begin{aligned} \|v_{\Delta x} - u(\Omega_{\Delta x})\|_{\Delta x} &\leq \|u_{\Delta x} - u(\Omega_{\Delta x})\|_{\Delta x} + \|v_{\Delta x} - u_{\Delta x}\|_{\Delta x} \\ &\leq C \cdot \Delta x^2 + \varepsilon_{\mathrm{MG}} \end{aligned}$$

Try to make $\varepsilon_{\mathrm{MG}} \approx C \cdot \Delta x^2$ then global error will dominate

Convergence analysis – a brief sketch

Assume linear convergence rate γ for each V-cycle

Run M V-cycles, reducing error to global error level if

$$\gamma^M \sim \Delta x^2 \sim N^{-2d}$$

for a 2^{nd} order method, implying number of iterations

$$M = \mathrm{O}(\log N)$$

Because the cost of each V-cycle is $\approx U_K = \mathrm{O}(N^d)$ total cost to reduce the error to the global error level is

$$\mathrm{O}(N^d \log N) = C \cdot \mathrm{cost}(\mathrm{FFT})$$