High Performance and Parallel Computing (MA 6252)

Lecture 8

A Tutorial Introduction to Multigrid

Ulrich Rüde
Lehrstuhl für Simulation
Universität Erlangen-Nürnberg

visiting Professor at
Department of Mathematics
National University of Singapore

Contents of High Performance and Parallel Computing

- Introduction
  - Computational Science and Engineering
  - High Performance Computing
- Computer Architecture
  - memory hierarchy
  - pipeline
  - instruction level parallelism
  - multi-core systems
  - parallel clusters
- Efficient Programming
  - computational complexity
  - efficient coding
  - architecture aware programming
  - shared memory parallel programming (OpenMP)
  - distributed memory parallel programming (MPI)
  - parallel programming with Graphics Cards

Introduction to Multigrid
Based on a Tutorial by

Irad Yavneh

Department of Computer Science

Technion – Israel Institute of Technology

Haifa 32000, Israel

irad@cs.technion.ac.il

Further Acknowledgements

- Multigrid Lecture Notes by S. McCormick
- Slides from Multigrid Tutorial by V. Henson
- „Why Multigrid Methods are so Efficient“ by I. Yavneh
- Multigrid Tutorial by B. Briggs, S. McCormick, V. Henson
- „The Multigrid Guide“ by A. Brandt

Overview

- Multigrid Tutorial
- Multilevel Adaptive Scheme
- Parallel Multigrid

Some Relevant Books/Articles:

1. W.L. Briggs, V.E. Henson, and S.F. McCormick: “A Multigrid Tutorial”, 2nd ed., SIAM, 2000.
2. U. Trottenberg, C.W. Oosterlee, and A. Schueller: “Multigrid”, Academic Press, 2001.
3. Brandt, A., “1984 Guide with Applications to Fluid Dynamics”, SIAM Classics in Applied Math. Series, 2011.
4. Hackbusch, W., “Multigrid Methods and Applications”, Springer, Berlin, 1985.
5. W. Hackbusch and U Trottenberg eds.: “Multigrid Methods”, Springer-Verlag, Berlin, 1982.
6. Wienands, R., and Joppich, W., “Practical Fourier Analysis for Multigrid Methods”, Chapman & Hall/CRC, 2004.
7. Computing in Science and Engineering (CiSE) Special Issue on Multigrid Computing, Comput. Sci. Eng. 8, 56 (2006)
8. Steve McCormick and Ulrich Rüde, "Multigrid Computing," Computing in Science & Engineering, vol. 8, no. 6, November/December 2006, pp. 10-11

What can multigrid achieve?

- Solve elliptic PDE in *asymptotically optimal complexity*
- Poisson‘s eqn in 2D: <30 FLOPs per unknown
  - Cheaper than computing

$$u_{i,j} = \sin(x_i)\sin(y_j)$$

- Solve FE problems (linear, scalar, elliptic PDE) with more than 10^{12} tetrahedral elements
  - on a massively parallel supercomputer
  - reminder: $10^{12} \neq 2 \times 10^6$

(*CISE* EXCLUSIVE CONTENT)

GUEST EDITORS' INTRODUCTION

Multigrid Computing

Steve McCormick and Ulrich Rüde
University of Colorado, Boulder and Universität Erlangen-Nürnberg

NOV./DEC. 2006

Multigrid methods are among the most important algorithms for computational scientists because they're the most efficient solvers for a wide range of problems. The modern era of multigrid methods began more than 30 years ago with the publication of Achi Brandt's seminal papers.[1,2]

Although it was originally successful for solving elliptic partial differential equations (PDEs) and the linear systems that arise when they're discretized, the basic multigrid principle of coupling multiple scales has much wider applicability.

Why a Special Issue?

At first glance, it might seem strange that such a fundamental task as the solution of linear systems is still worth a special issue in a computational science magazine in 2006. In fact, the solution algorithms for linear systems are usually hidden from the computational scientist—say, behind Matlab's backslash operator or within state-of-the art commercial finite element packages or computational fluid dynamics tools. Ideally, computational scientists shouldn't have to worry how a linear system is being solved, so why, then, did we bring together a special issue on it?

The answer is that the linear solver is still the computational bottleneck in terms of complexity for many mesh-based numerical simulations. Although we can usually perform data pre- and postprocessing linearly in the number of mesh points and then parallelize it with standard partitioning techniques, the complexity of most linear system solvers—direct or iterative—is worse than linear. Multigrid methods are among the few exceptions. In practice, the solver's complexity might not dominate small- or moderate-sized problems, but using multigrid methods can offer tremendous gains in efficiency for large-scale computing. One more aspect to the complexity argument concerns *asymptotically optimal* solvers—their overall cost is linear or close to linear in the number of unknowns. A classical example is a fast Fourier transform- (FFT-) based algorithm with $N\log N$ complexity.

Why Multigrid Methods Are So Efficient

Originally introduced as a way to numerically solve elliptic boundary-value problems, multigrid methods, and their various multiscale descendants, have since been developed and applied to various problems in many disciplines. This introductory article provides the basic concepts and methods of analysis and outlines some of the difficulties of developing efficient multigrid algorithms.

Contents of Tutorial

- Basic Concepts: The main ideas via the soldier alignment problem (due to A. Bruckstein)
- Multigrid smoothing and coarse grid correction
- V-cycle
- Full Multigrid
- Full approximation storage
- Tau-Extrapolation
- Algebraic Multigrid

What's it about?

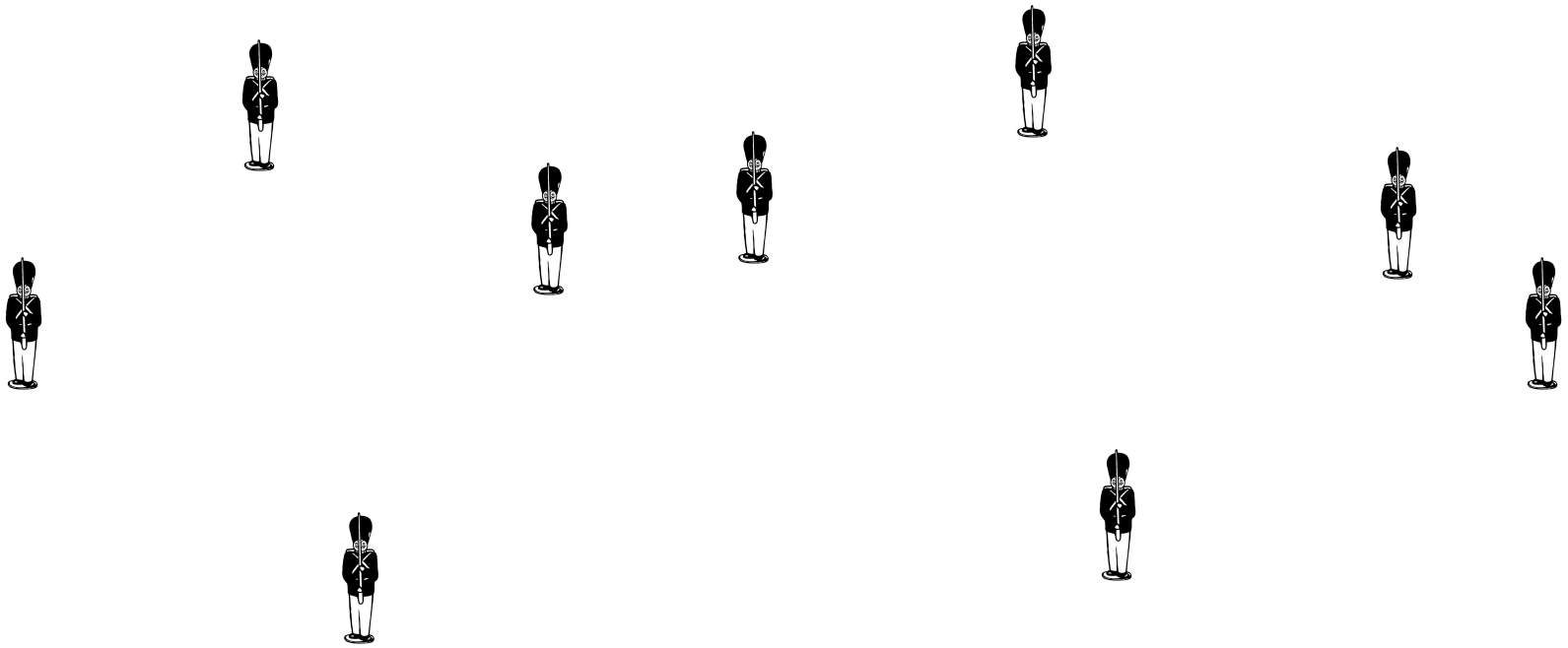
- A framework of efficient iterative methods for solving problems with many variables and many scales.

- Framework: common concept, different methods.
- Efficient: usually $O(N)$ or $O(N \log N)$ operations
  The importance of efficient methods becomes greater as computers grow stronger!
- Iterative: most nontrivial problems in our field cannot be solved directly efficiently.
- Solving: approximately, subject to appropriate convergence criteria, constraints, etc.
- Many variables: the larger the number of variables, the greater the gain of efficient methods
- Many scales: typical spatial and/or temporal sizes.

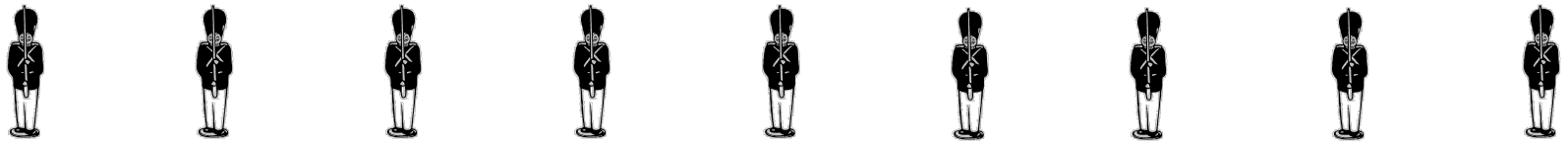
A framework of efficient iterative methods for solving problems with many variables and many scales.

Basic Concepts: Local vs. Global processing.

Imagine a large number of soldiers who need to be arranged in a straight line and at equal distances from each other.
The two soldiers at the ends of the line are fixed. Suppose we number the soldiers 0 to N , and that the length of the entire line is L.

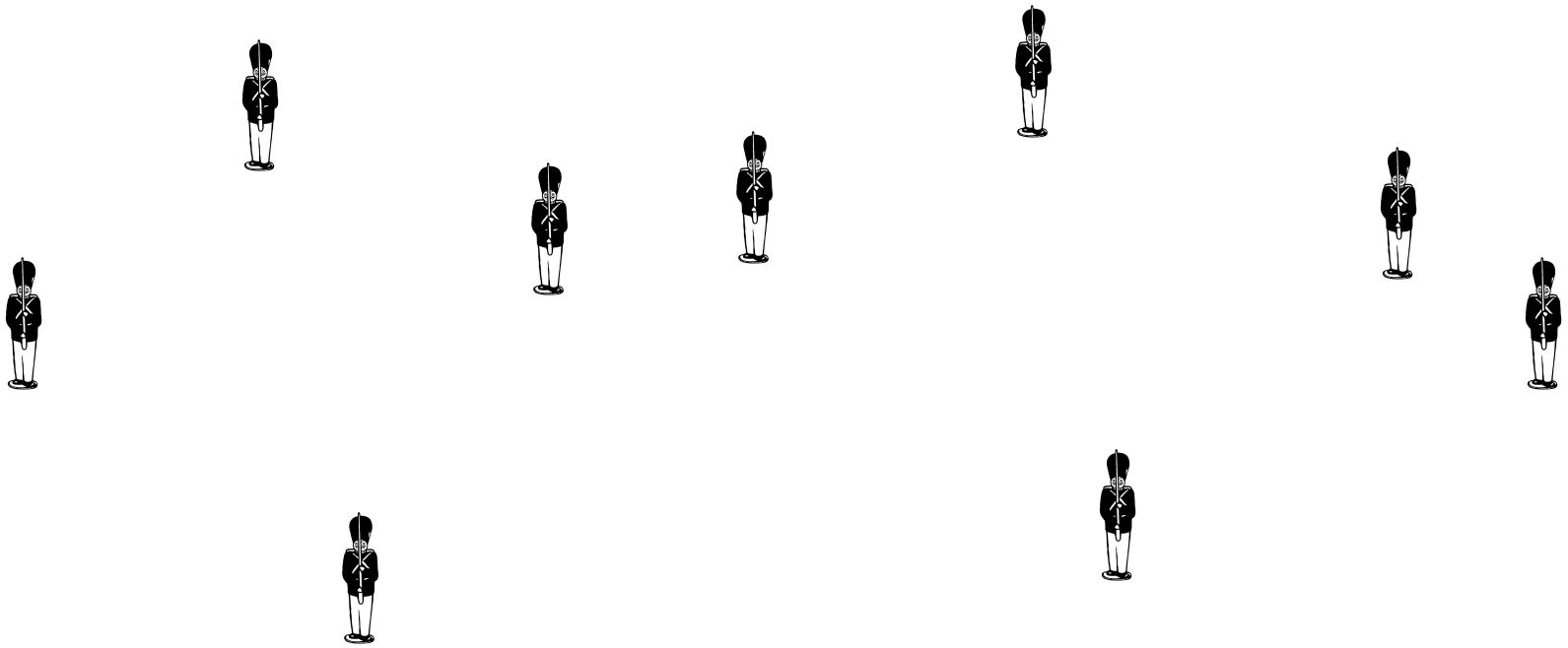


Initial Position



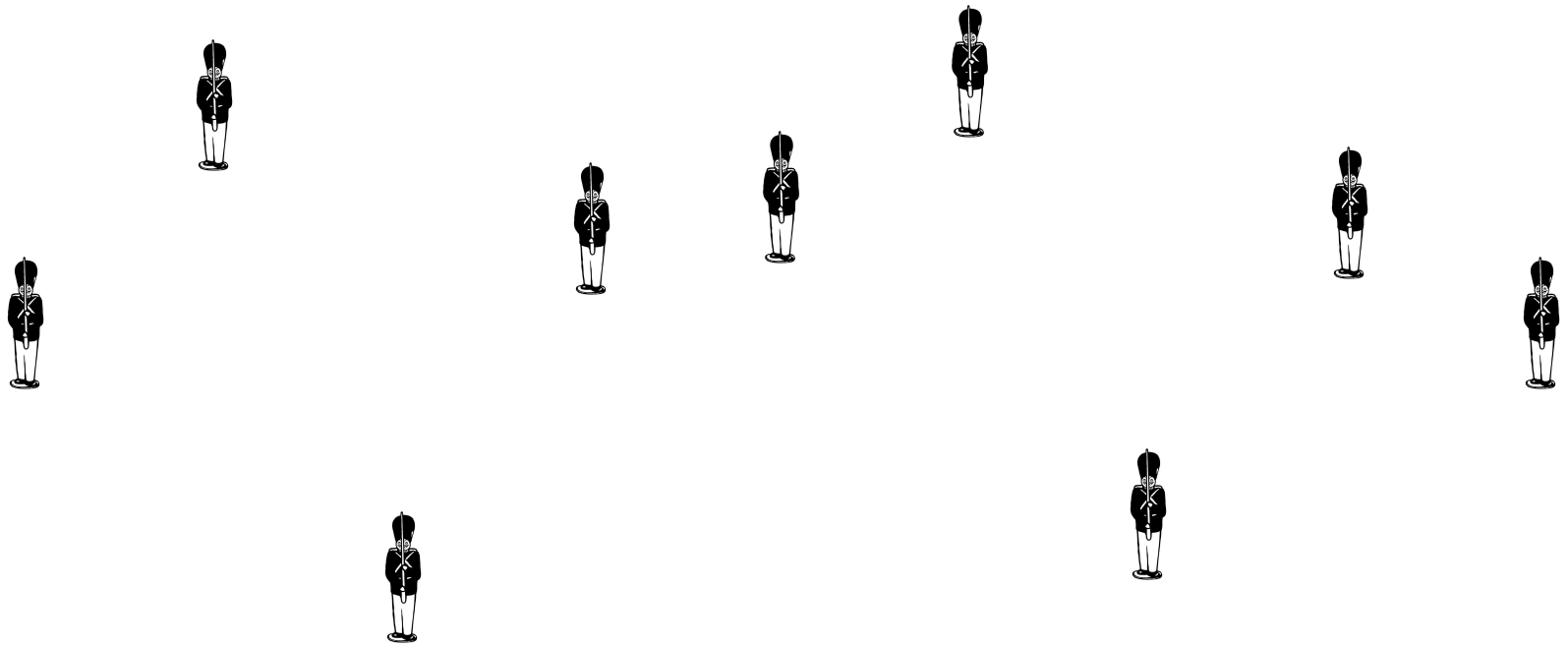
Final Position

Global processing. Let soldier number j stand on the line connecting soldier 0 to soldier N at a distance jL/N from soldier number 0.

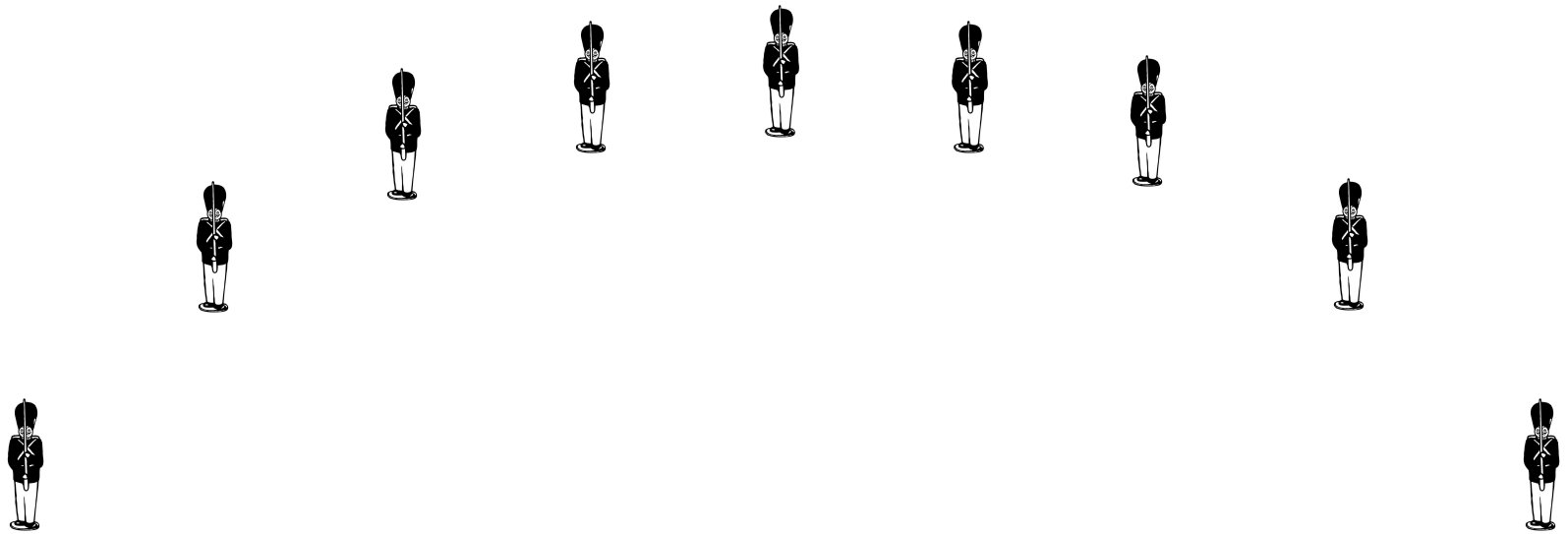


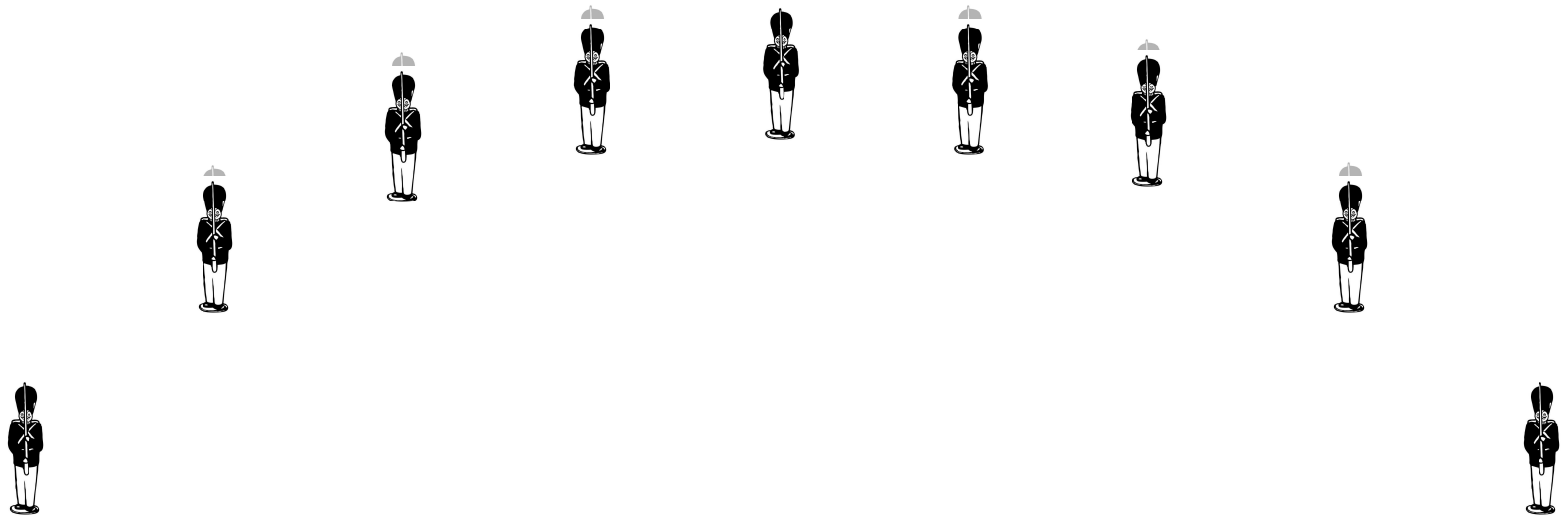
Global processing. Let soldier number j stand on the line connecting soldier 0 to soldier N at a distance jL/N from soldier number 0.

This method solves the problem directly, but it requires a high degree of sophistication: recognition of the extreme soldiers and some pretty fancy arithmetic.

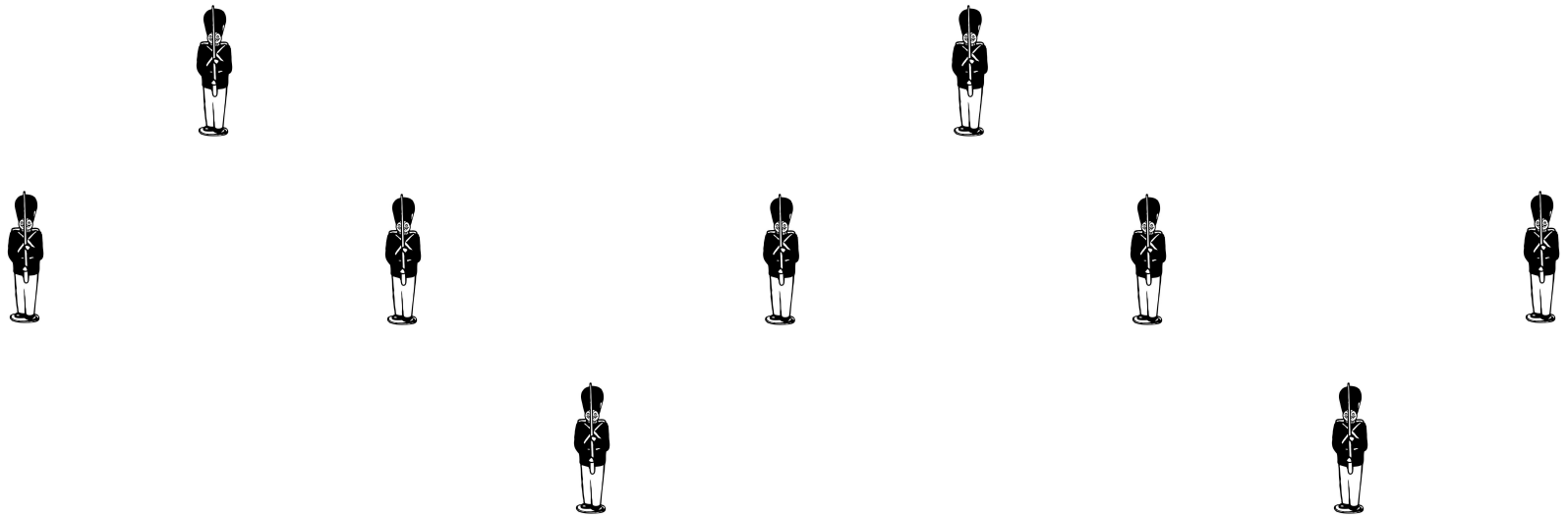


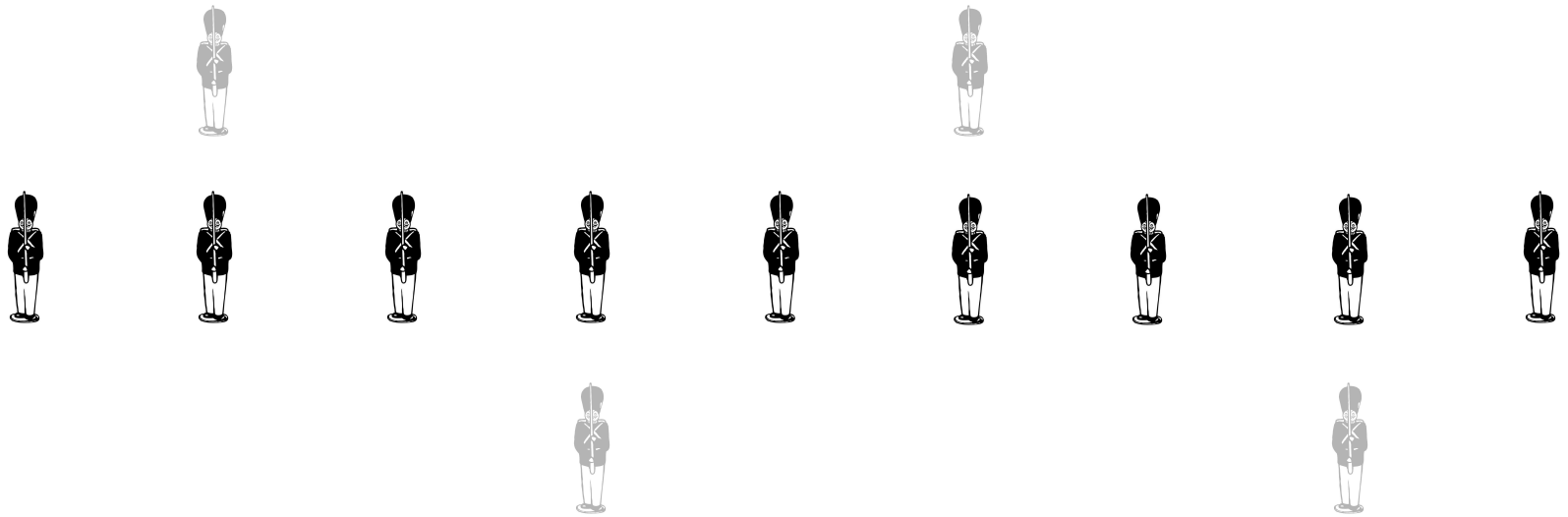
A step in the right direction



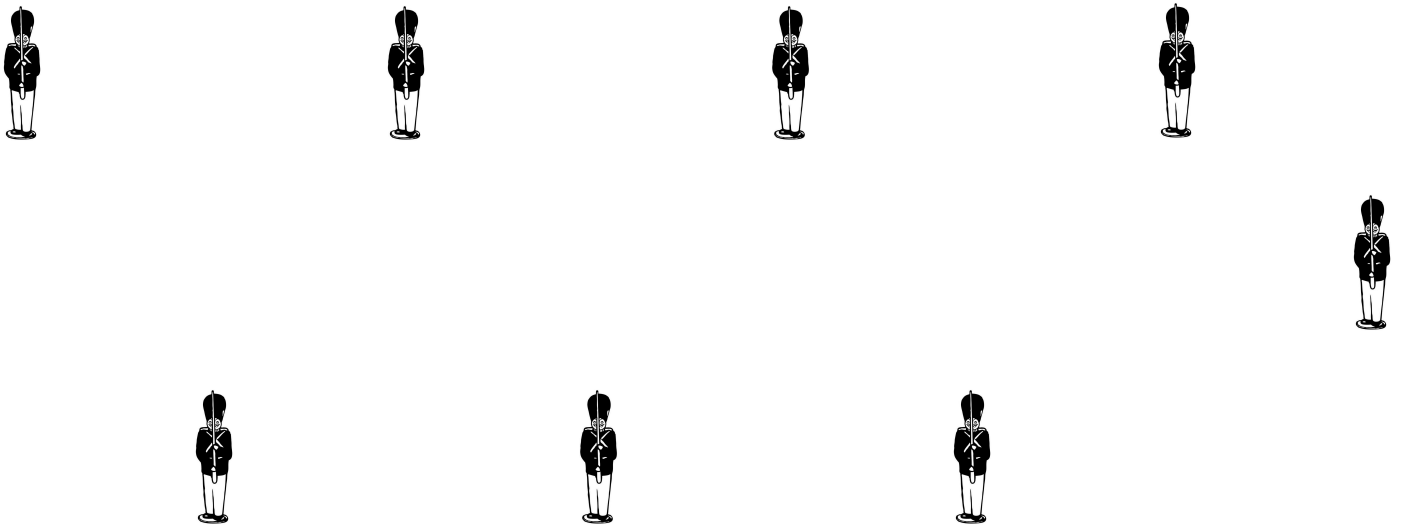


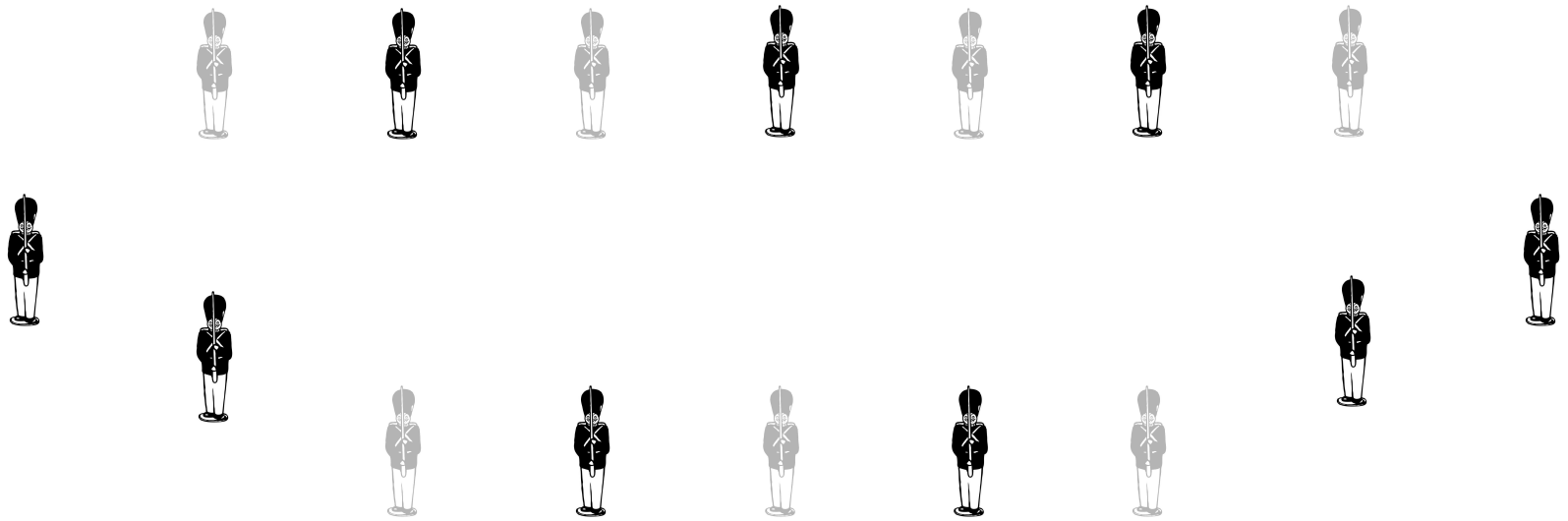
Slow convergence



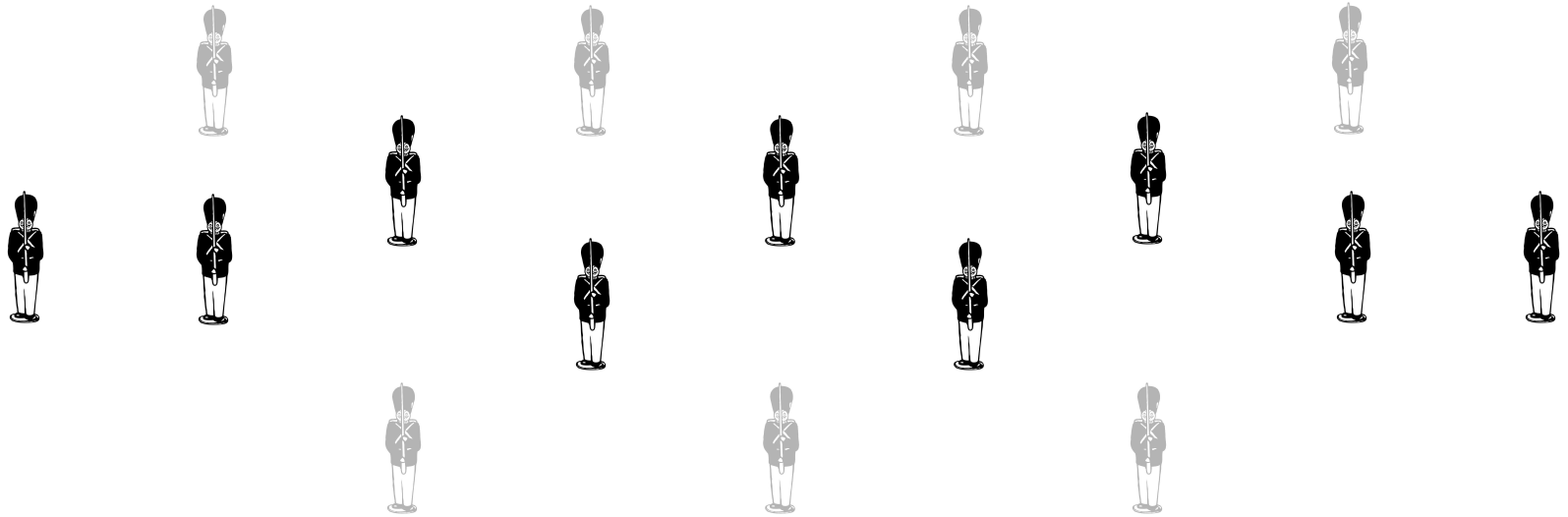


Fast convergence

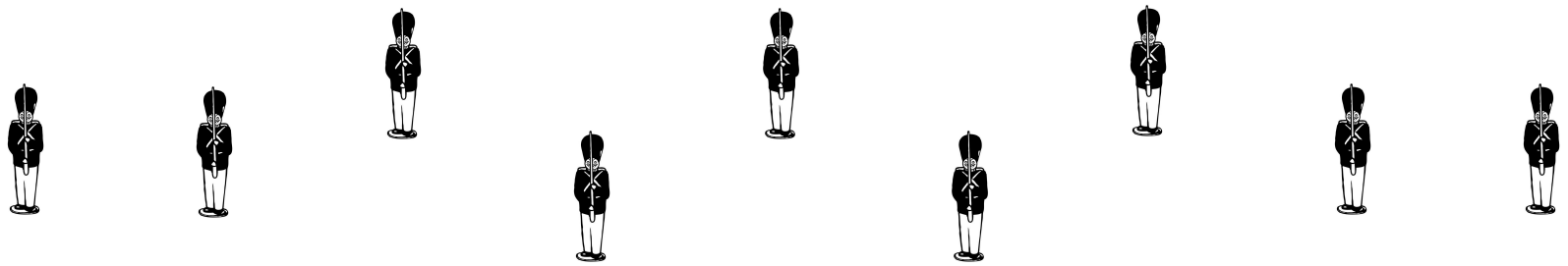




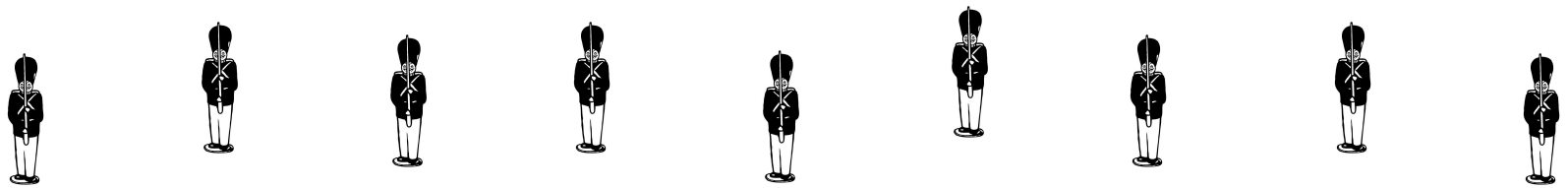
Slow convergence



Local solution: damping



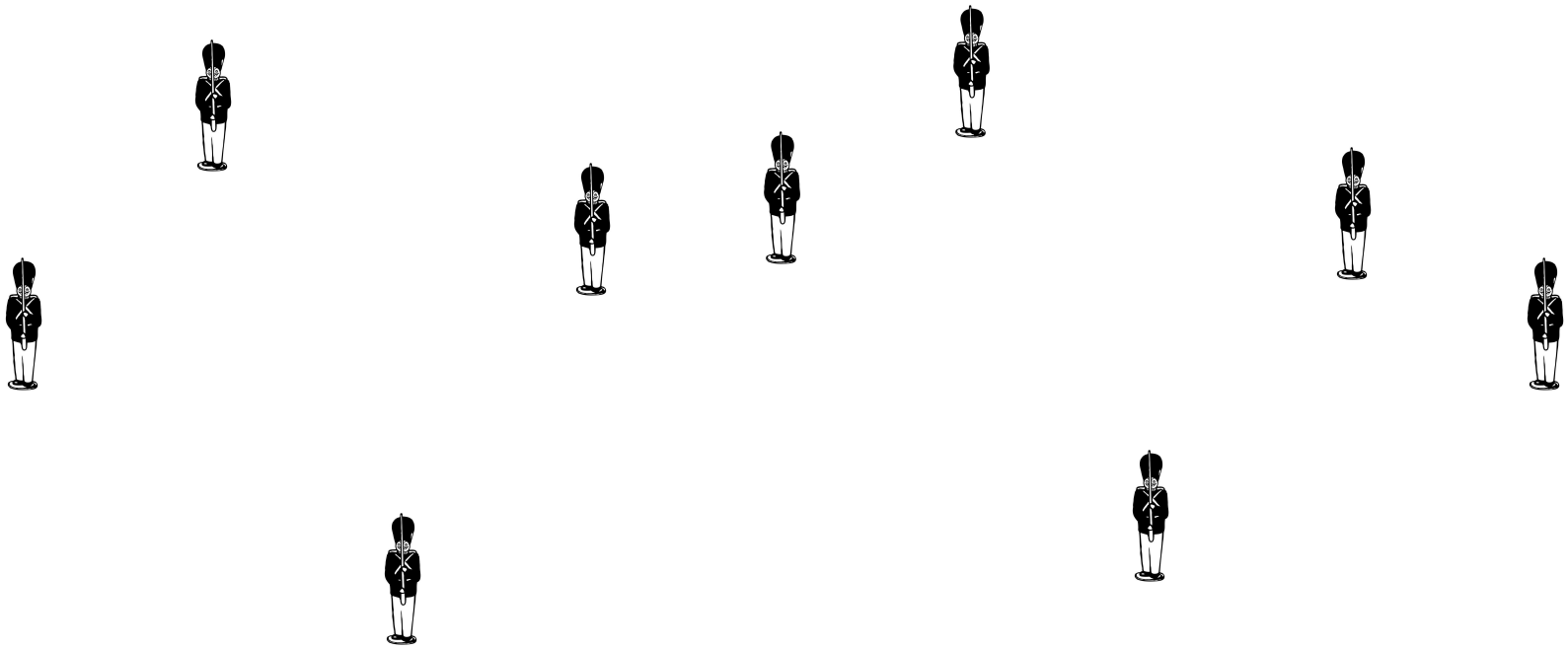
Local solution: damping

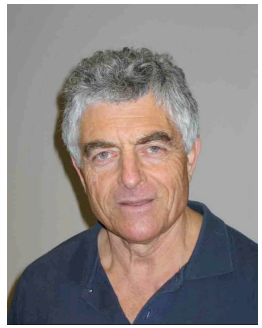


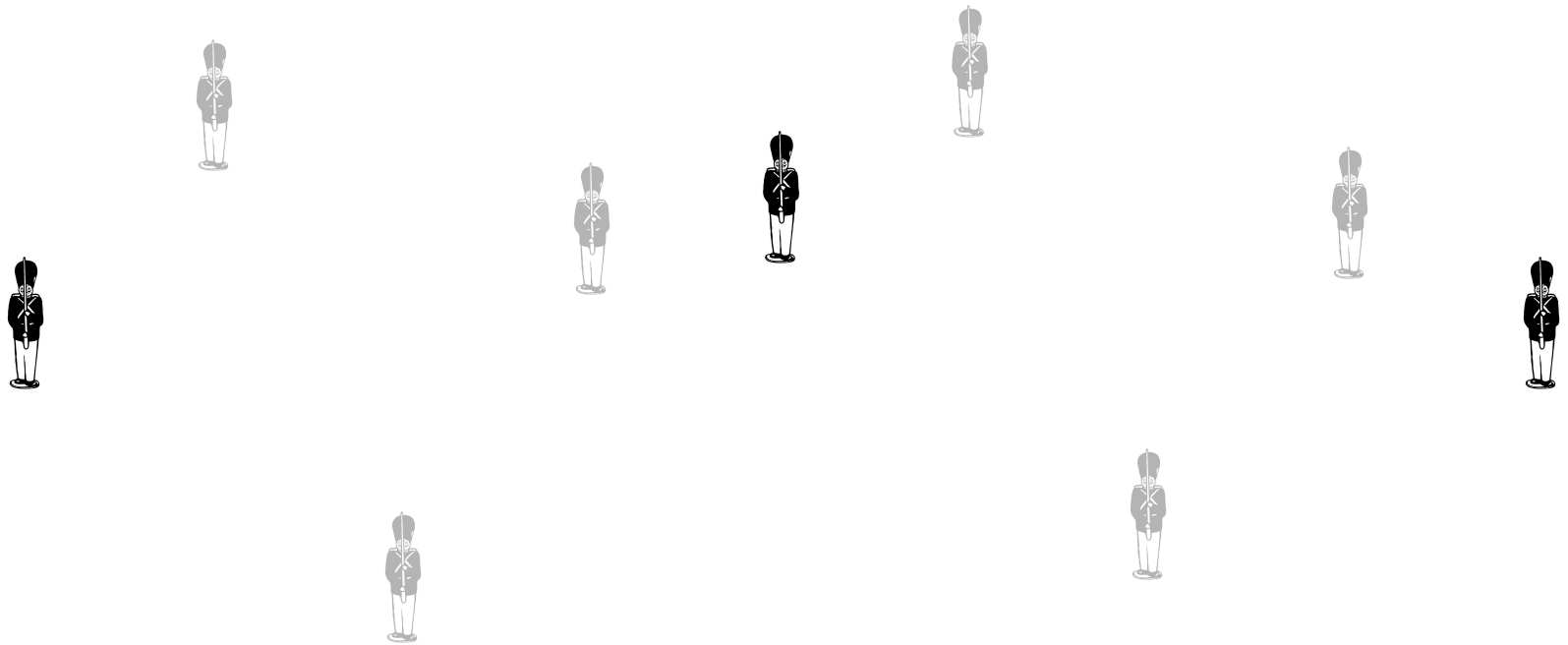
Local solution: damping

Local solution: damping

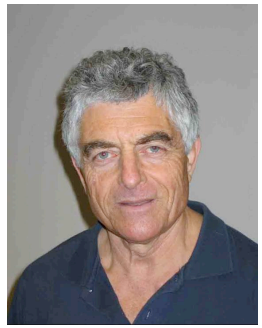
The multiscale idea: Employ the local processing with simple arithmetic. But do this on all the different scales.

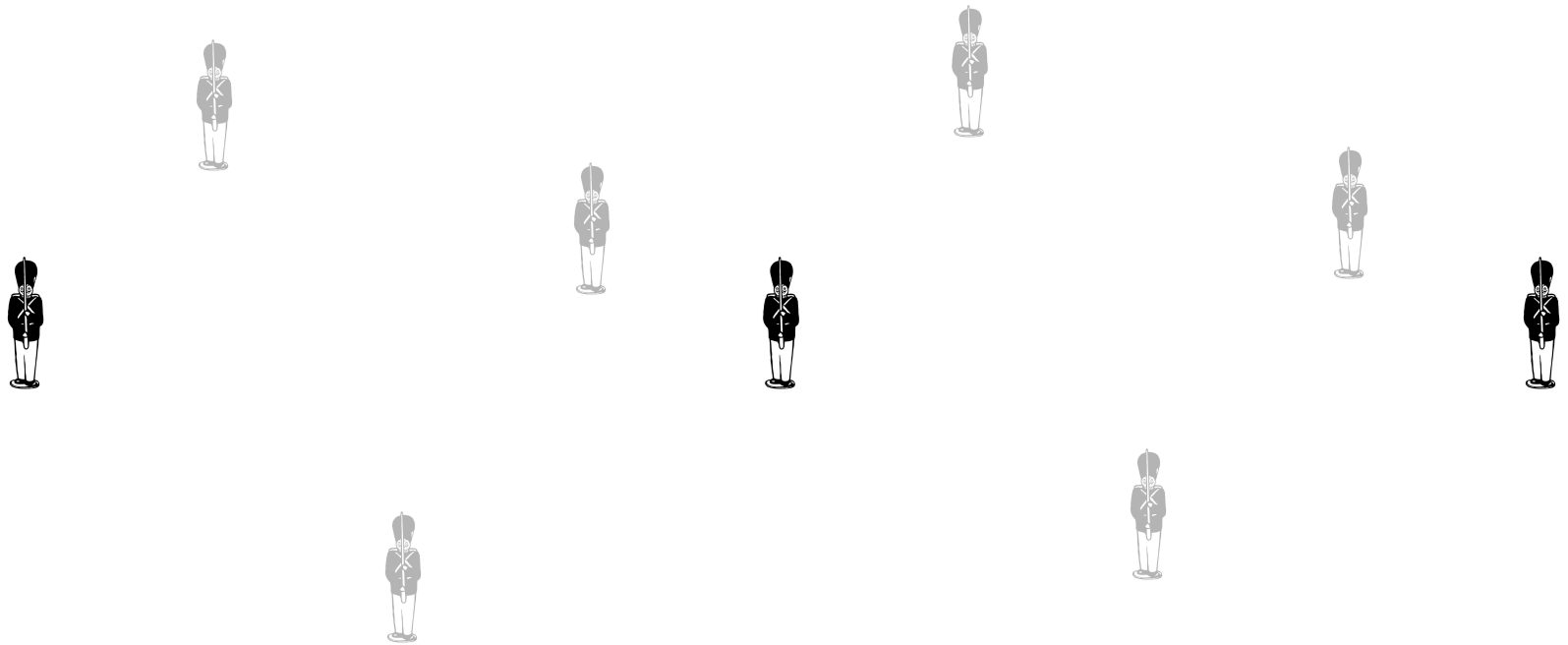




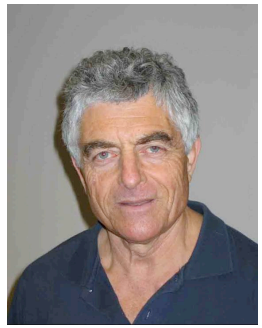


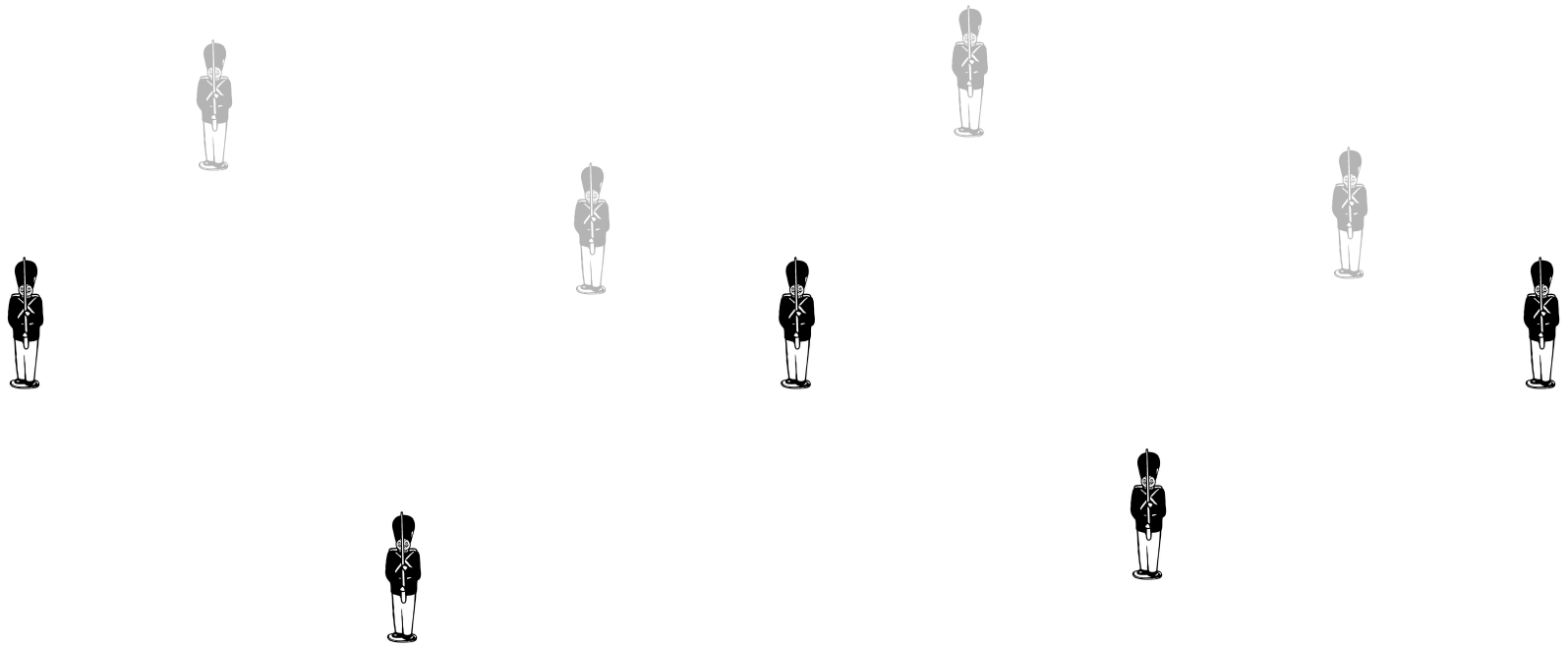
Large scale



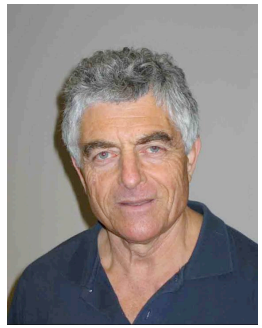


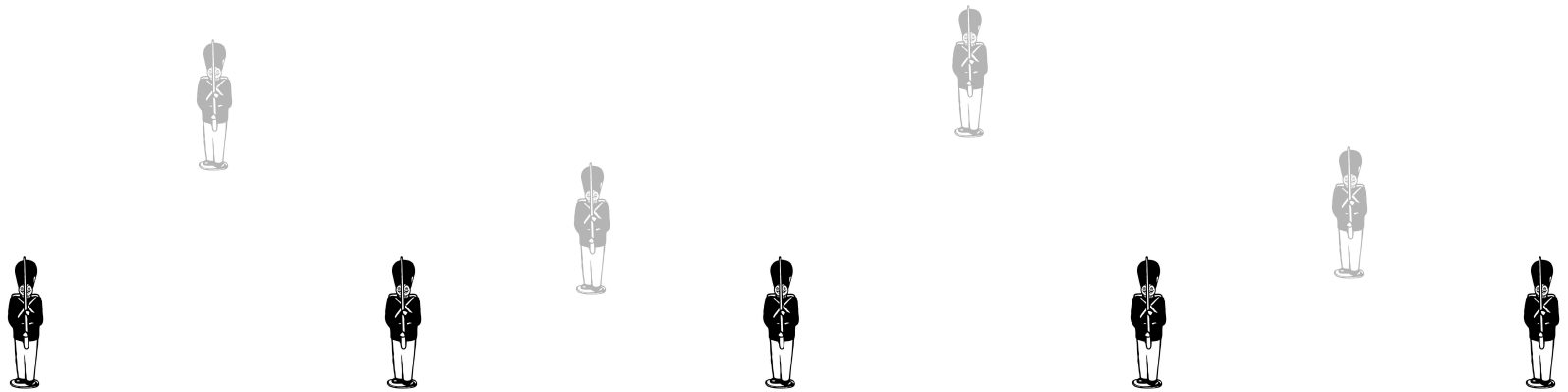
Large scale



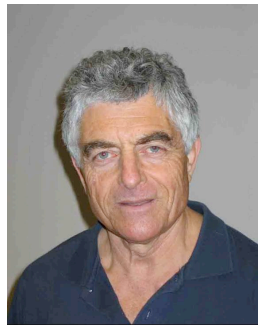


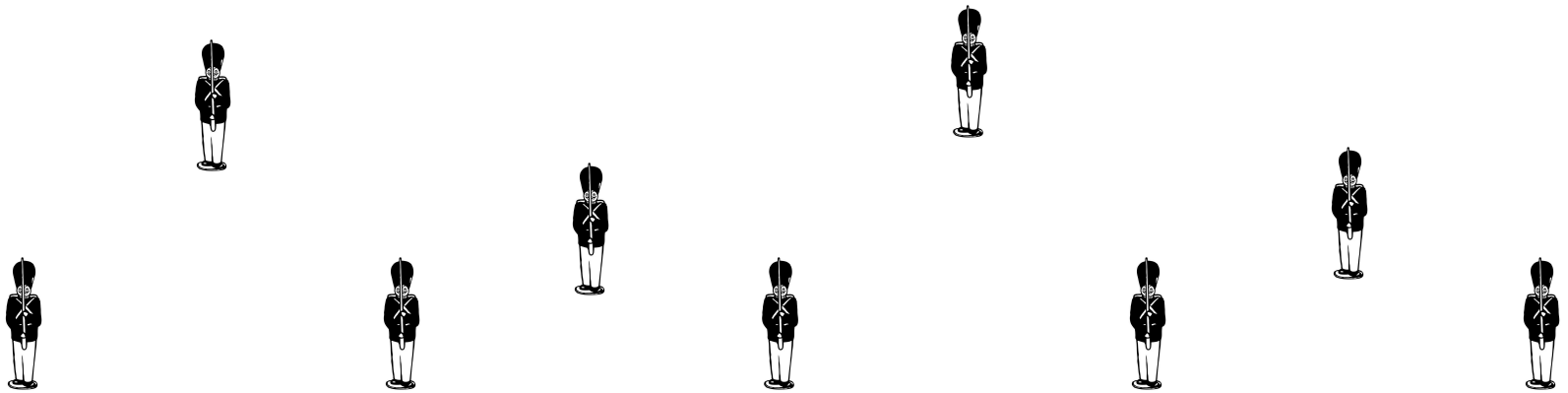
Intermediate scale



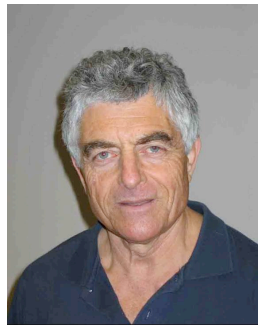


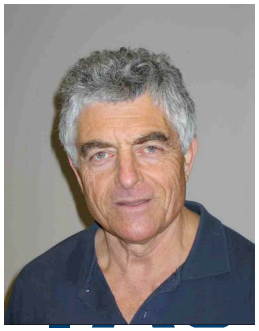
Intermediate scale





Small scale





How much work do we save?

Jacobi's method requires about N^2 iterations and $N^2 * N = N^3$ operations to improve the accuracy by an order of magnitude.

The multiscale approach solves the problem in about $\text{Log}_2(N)$ iterations (whistle blows) and only about N operations.

Example: for $N = 1000$ we require about:

10 iterations and 1000 operations

instead of about

1,000,000 iterations and 1,000,000,000 operations

How important is computational efficiency?
Suppose that we have three different algorithms for a given problem, with different computational complexities for input size N :

Algorithm 1: $10^6\ N$ operations
Algorithm 2: $10^3\ N^2$ operations
Algorithm 3: N^3 operations

Suppose that the problem size, N, is such that Algorithm 1 requires one second.
How long do the others require?

Algorithm 3 $O(N^3)$	Algorithm 2 $O(N^2)$	Algorithm 1 $O(N)$	N	Computer Speed (ops/sec)
0.000001 sec	0.001 sec	1 sec	1	1M (~10^6) (1980's)
1 sec	1 sec	1 sec	1K	1G (~10^9) (1990's)
12 days	17 min	1 sec	1M	1T (~10^{12}) (2000's)
31,710 years	12 days	1 sec	1G	1P (~10^{15}) (2010's)

Stronger Computers $\Rightarrow$

Greater Advantage of Efficient Algorithms!

Multigrid is not the answer to everything!

+ Sparse, low dimension, large, stiff, elliptic PDE, geometric, smooth long-range effects, structured, isotropic, smoothly varying coefficients, symmetric positive definite.

~ Nonlinear, disordered, anisotropic, discontinuous coefficients, singular-perturbation and non-elliptic PDE, PDE systems, non-symmetric, indefinite, non-deterministic.

- Dense, high-dimensional, small, single-scale.

The catch: in less trivial problems, we cannot construct appropriate equations on the large scales without first propagating information from the small scales.

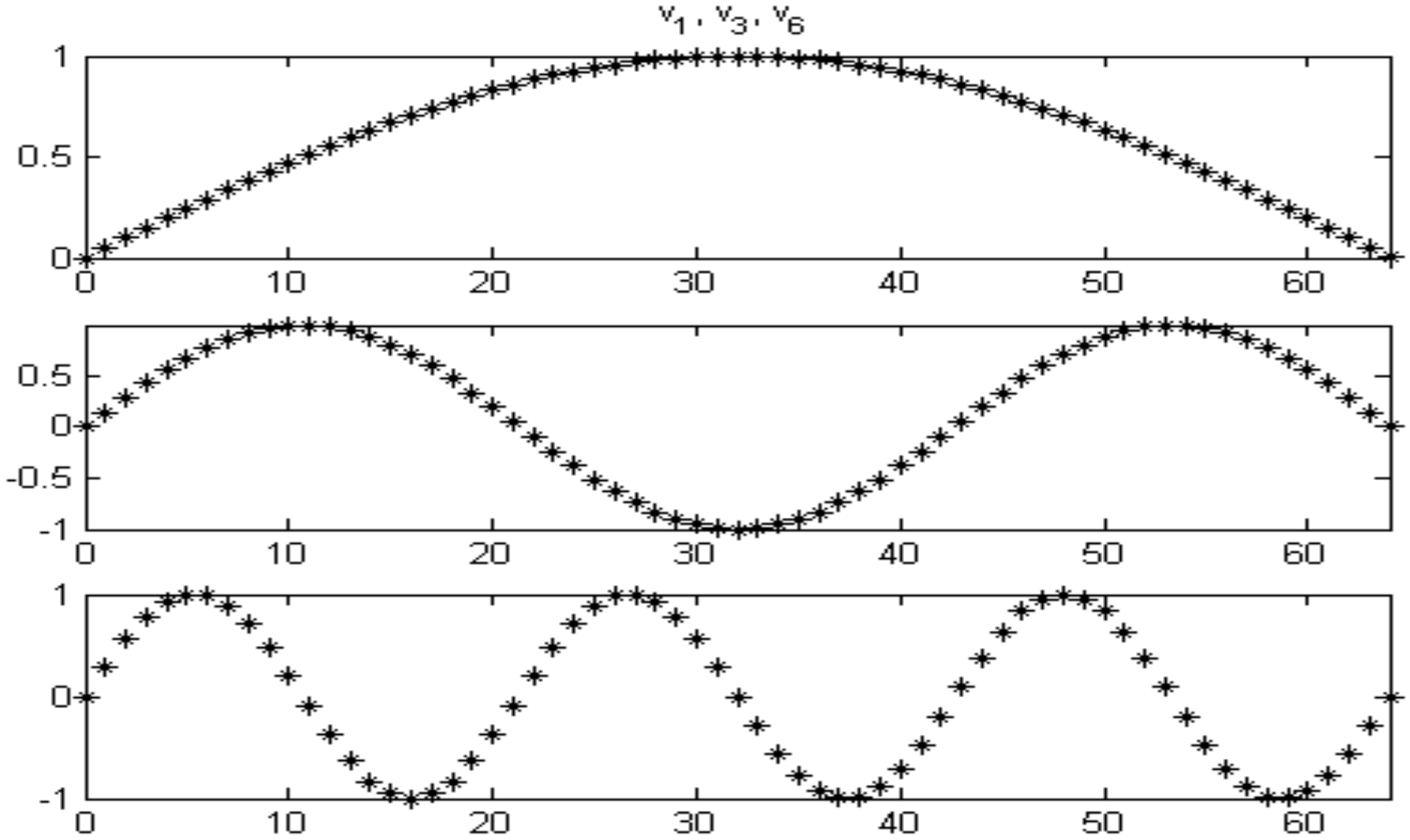
Skill in developing efficient multilevel algorithms is required for:

1. Choosing a good local iteration.
2. Choosing appropriate coarse-scale variables.
3. Choosing inter-scale transfer operators.
4. Constructing coarse-scale approximations to the fine-scale problem.

Analysis of the local iterative process

We can show that the eigenfunctions of the iterative process are sine-shaped functions.

This means that we can analyze the processes by examining sine functions of different frequencies independently.


v_1, v_3, v_6
1
0.5
0
0
10
20
30
40
50
60
0.5
0
-0.5
-1
0
10
20
30
40
50
60
1
0
-1
0
10
20
30
40
50
60

Analysis of the iterative process

Matrix representation:

$$\mathbf{x}^{(i)} = \mathbf{S}\mathbf{x}^{(i-1)}$$

where

$$\mathbf{S} = \frac{1}{2}\begin{bmatrix} 0 & 1 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & \cdots & \cdots & \cdots & & \\ & & & 1 & 0 & 1 & \\ & & & & 1 & 0 & 1 \\ & & & & & 1 & 0 \end{bmatrix}$$

This matrix S has $N-1$ linearly independent eigenvectors, λ_k :and corresponding real eigenvalues, $\mathbf{v}^k$

$$\mathbf{S} \quad \mathbf{v}^k = \lambda_k \mathbf{v}^k .$$

Since $\mathbf{v}^k$ span the space $\Re^{N-1}$, any initial configuration of the soldiers can be written as a linear combination:

$$\mathbf{x}^{(0)} = \sum_{k=1}^{N-1} c_k \mathbf{v}^k$$

with some coefficients, c_k.

Hence, we obtain after *m* iterations:

$$\mathbf{x}^{(m)} = \mathbf{S}\mathbf{x}^{(m-1)} = \mathbf{S}^2\mathbf{x}^{(m-2)} =$$

$$\ldots = \mathbf{S}^m\mathbf{x}^{(0)} = \mathbf{S}^m \sum_k c_k \mathbf{v}^k = \sum_k c_k \lambda_k^m \mathbf{v}^k$$

Conclusion:

$$\lim_{m\to\infty} \mathbf{x}^{(m)} \to 0 \quad if \quad |\lambda_k| < 1, \quad k = 1, \ldots, N-1$$

That is, the iteration converges if the spectral radius, ρ, of the iteration matrix, $\mathbf{S}$, is smaller than 1.

Note: since $|\lambda_k| < 1$, the method converges. But, for some eigenvectors, $|\lambda_k|$ is close to 1, so convergence is slow. In particular, for $k\pi / N << 1$, we have,

$$\cos\left(\frac{k\pi}{N}\right) \approx 1 - \frac{1}{2}\left(\frac{k\pi}{N}\right)^2$$

For $k = 1$ we obtain

$$\lambda_1^m \approx \left[1 - \frac{1}{2}\left(\frac{\pi}{N}\right)\right]^m \approx e^{-\frac{1}{2}m\left(\frac{\pi}{N}\right)^2}$$

Hence, $O(N^2)$ iterations are required to reduce such an error by an order of magnitude.

Observation: the eigenvectors and eigenvalues of the matrix $\mathbf{S}$ are given by

$$\mathbf{v}^k = \{v_j^k\} = \sin\left(\frac{jk\pi}{N}\right), \quad j = 1, \ldots, N-1,$$

$$\lambda_k = \cos\left(\frac{k\pi}{N}\right),$$

with $k = 1, \ldots, N-1$.

Proof: Using the trigonometric identity,

$$\frac{1}{2}\left[\sin\frac{(j-1)k\pi}{N} + \sin\frac{(j+1)k\pi}{N}\right] = \cos\frac{k\pi}{N}\sin\frac{jk\pi}{N},$$

and the fact that $\sin 0 = \sin\pi = 0$, we obtain by substitution, $\mathbf{S}\,\mathbf{v}^k = \lambda_k \mathbf{v}^k$

Note that convergence is also slow for $k/N \approx 1$.

This can be overcome by damping:

$$x_j^{(i)} = (1-\omega)x_j^{(i-1)} + \omega\frac{1}{2}\left(x_{j-1}^{(i-1)} + x_{j+1}^{(i-1)}\right),$$

where ω is a parameter. Then,

$$\mathbf{x}^{(i)} = \mathbf{S}_\omega \mathbf{x}^{(i-1)},$$

where

$$\mathbf{S}_\omega = (1-\omega)\mathbf{I} + \omega\mathbf{S}.$$

Note: $\mathbf{v}^k$ are eigenvectors of S_ω . The corresponding eigenvalues are now

$$\lambda_k^{(\omega)} = 1 - \omega + \omega\lambda_k = 1 - \omega(1-\lambda_k).$$

For $0 < \omega \le 1$ we have convergence, $\left|\lambda_k^{(\omega)}\right| < 1$.

Definition: Eigenvectors $\mathbf{v}^k$ with $1 \le k < N/2$

are called smooth (low-frequency). Those for k large are called rough or oscillatory (high-frequency).

$$N/2 \le k < N$$

Note that for rough eigenvectors, $\lambda_k \le 0$.

Problem: Find $0 < \omega < 1$ which yields optimal convergence for the set of rough modes for arbitrary N, i.e.,

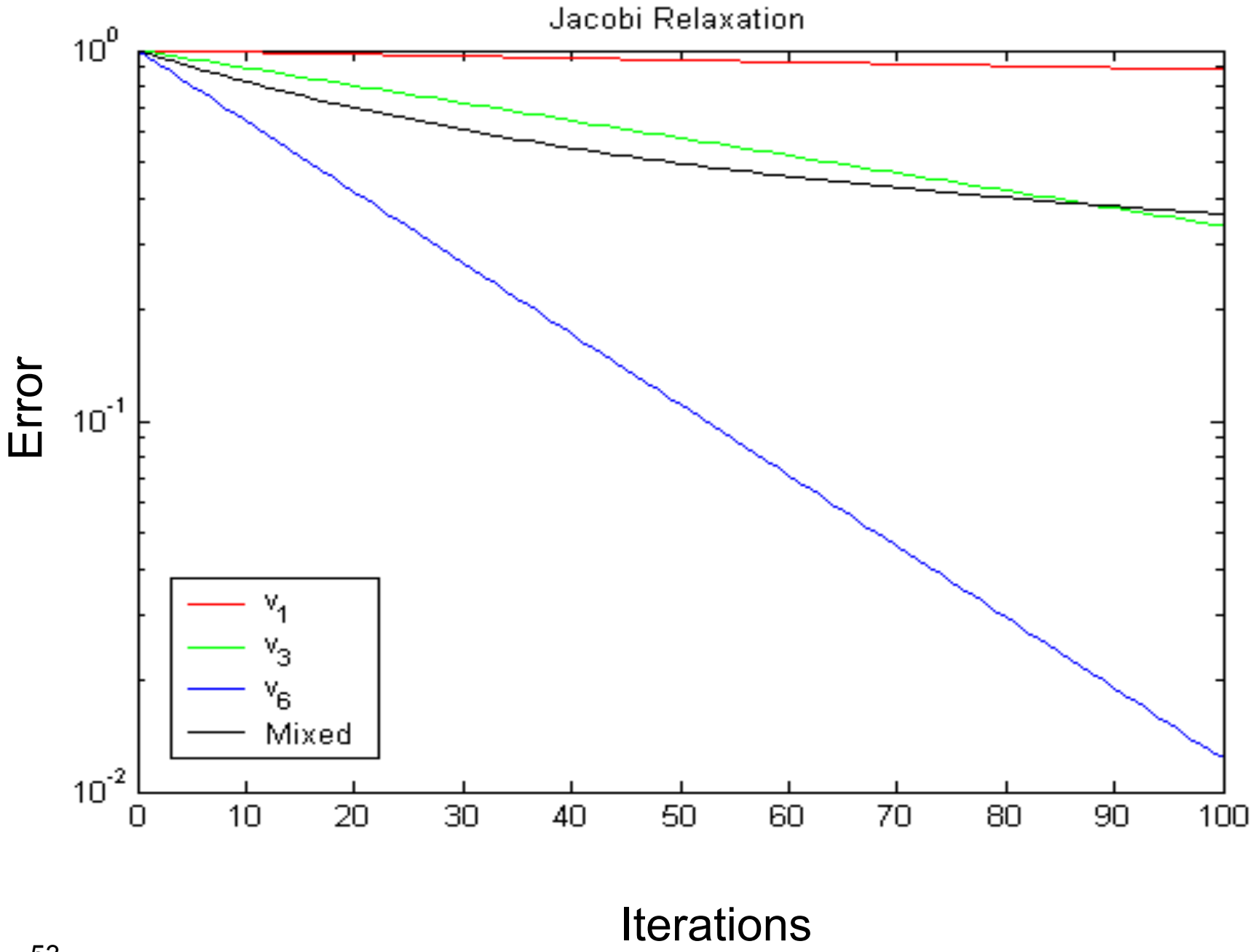
$$\omega: \quad \sup_{N} \max_{\frac{N}{2} \le k < N} \left|\lambda_k^{(\omega)}\right| = \min!$$

Note that λ_k can be any number in (-1, 0].

What is then the bound on the convergence factor, $\left|\lambda_k^{(\omega)}\right|$, of the rough modes?

For the smoothest eigenfunction, ν_1 , we have shown that $O(N^2)$ iterations are required to reduce the error by an order of magnitude.

On the other hand, highly-oscillatory eigenfunctions require only $O(1)$ iterations (when damping is used). These are eigenfunctions whose wave-lengths are on a scale of just a few variables (“soldiers” in our example).


Jacobi Relaxation
Error
Iterations
10^0
10^{-1}
10^{-2}
0
10
20
30
40
50
60
70
80
90
100
v_1
v_3
v_6
Mixed

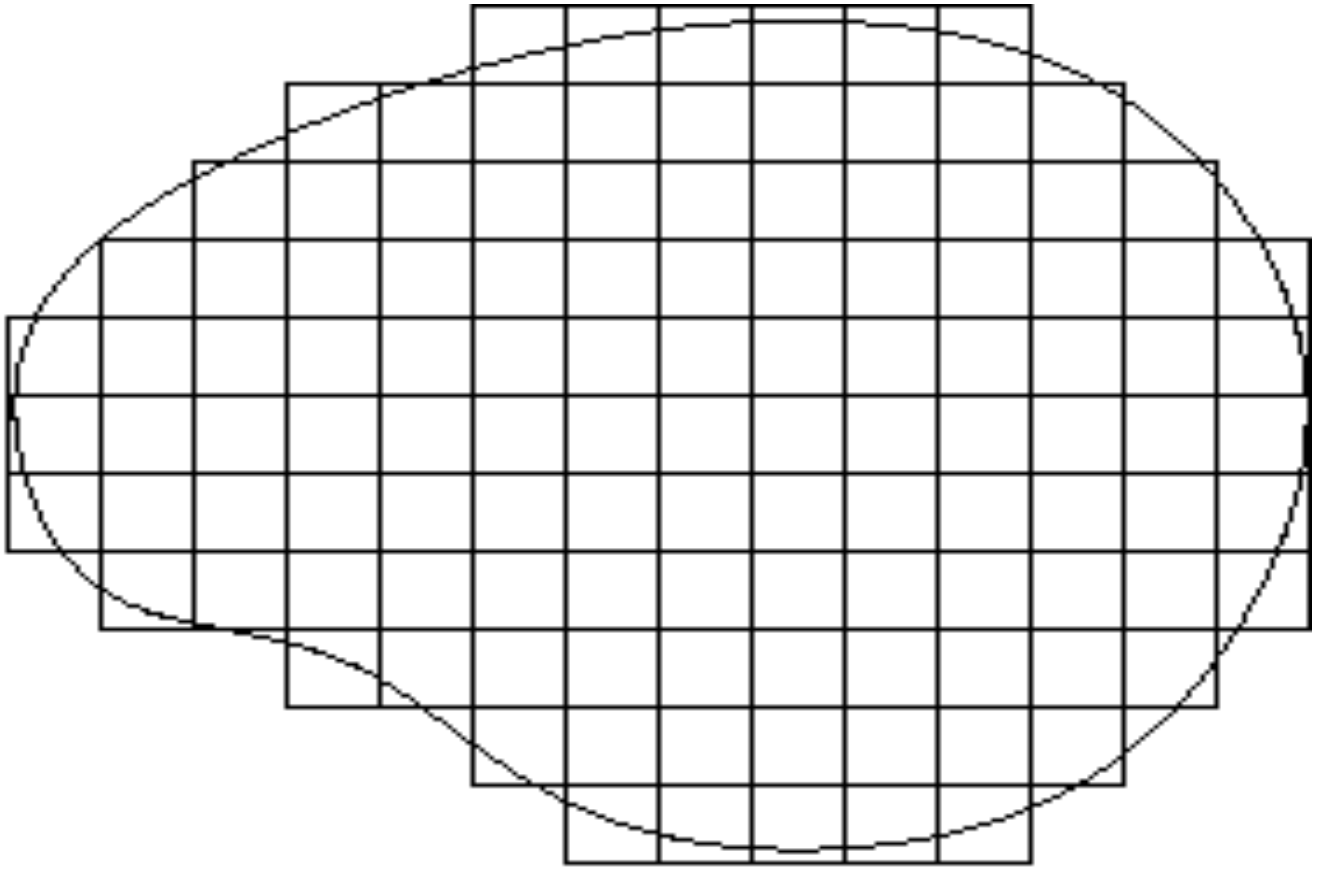
2D Model Problem

Find *u* which satisfies:

$$Lu = u_{xx} + u_{yy} = f(x,y), \quad (x,y) \in \Omega, \tag{4}$$

$$u = g(x,y), \quad (x,y) \in \partial\Omega.$$

This is the 2D Poisson equation, with Dirichlet boundary conditions. It is an elliptic partial differential equation which appears is many models.



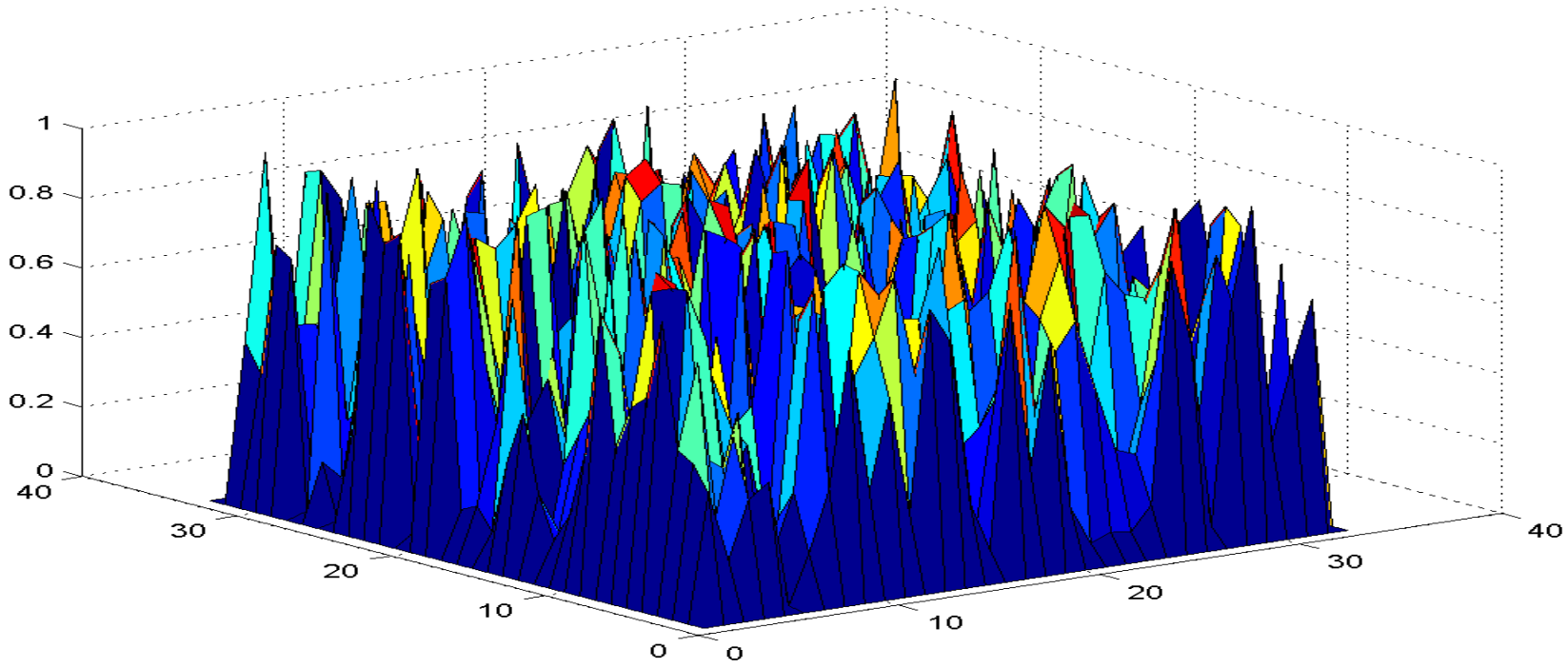
Ω^h

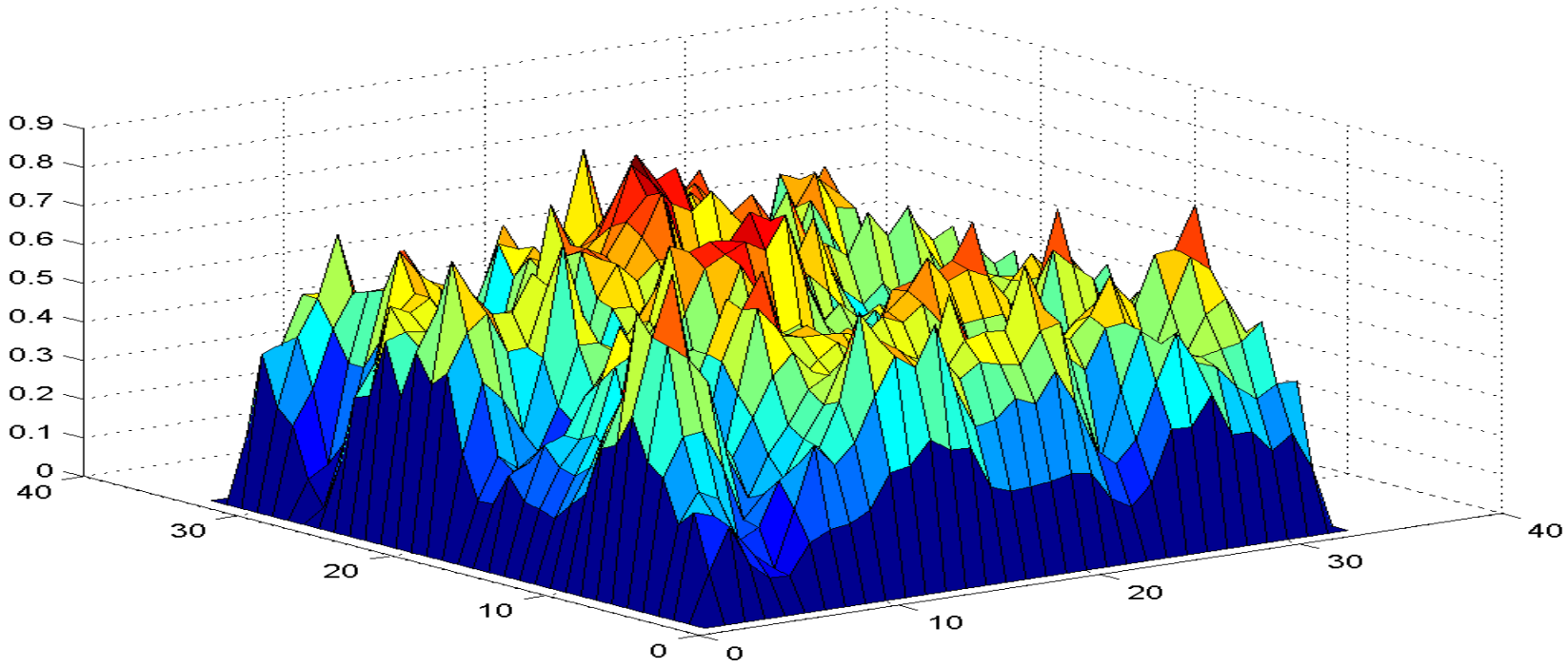
Discrete approximation

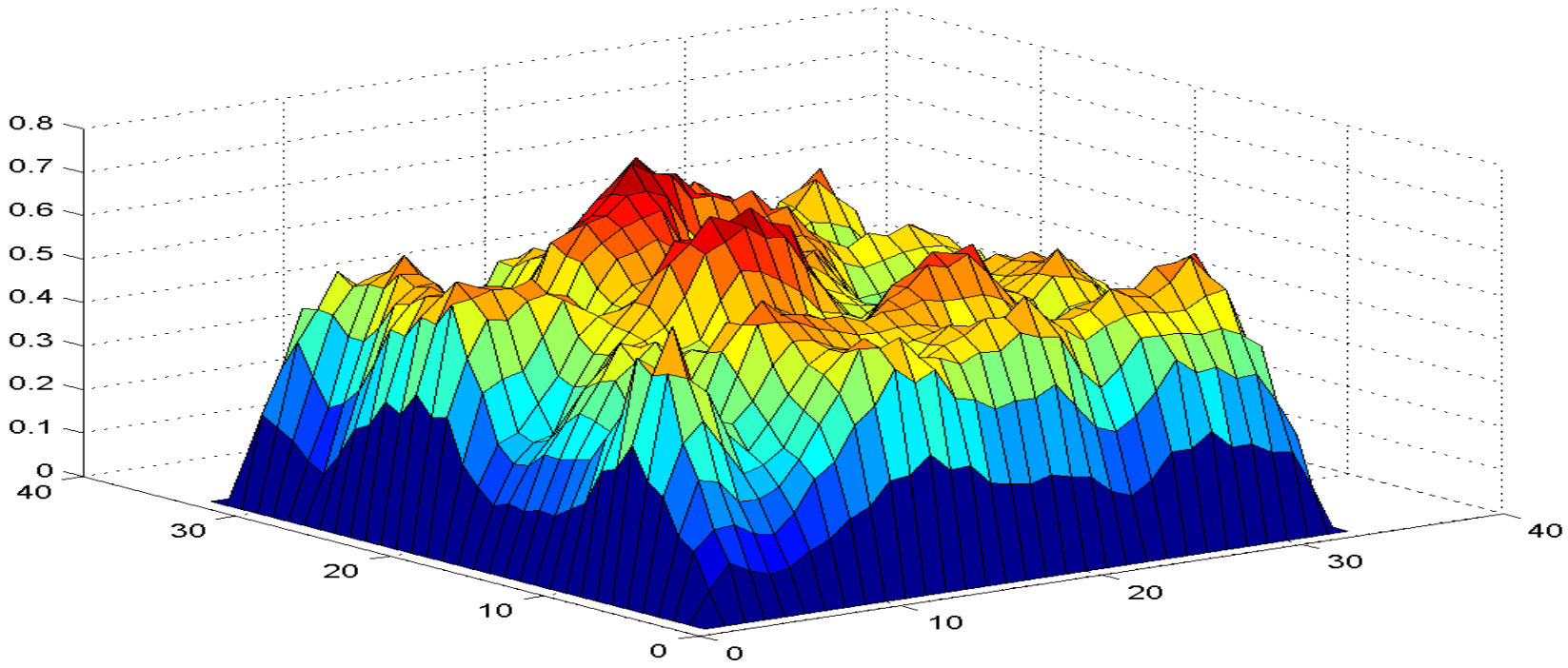
Define a grid: $\Omega^h \subset \Omega$ (assumed to be uniform for simplicity, with mesh interval h).

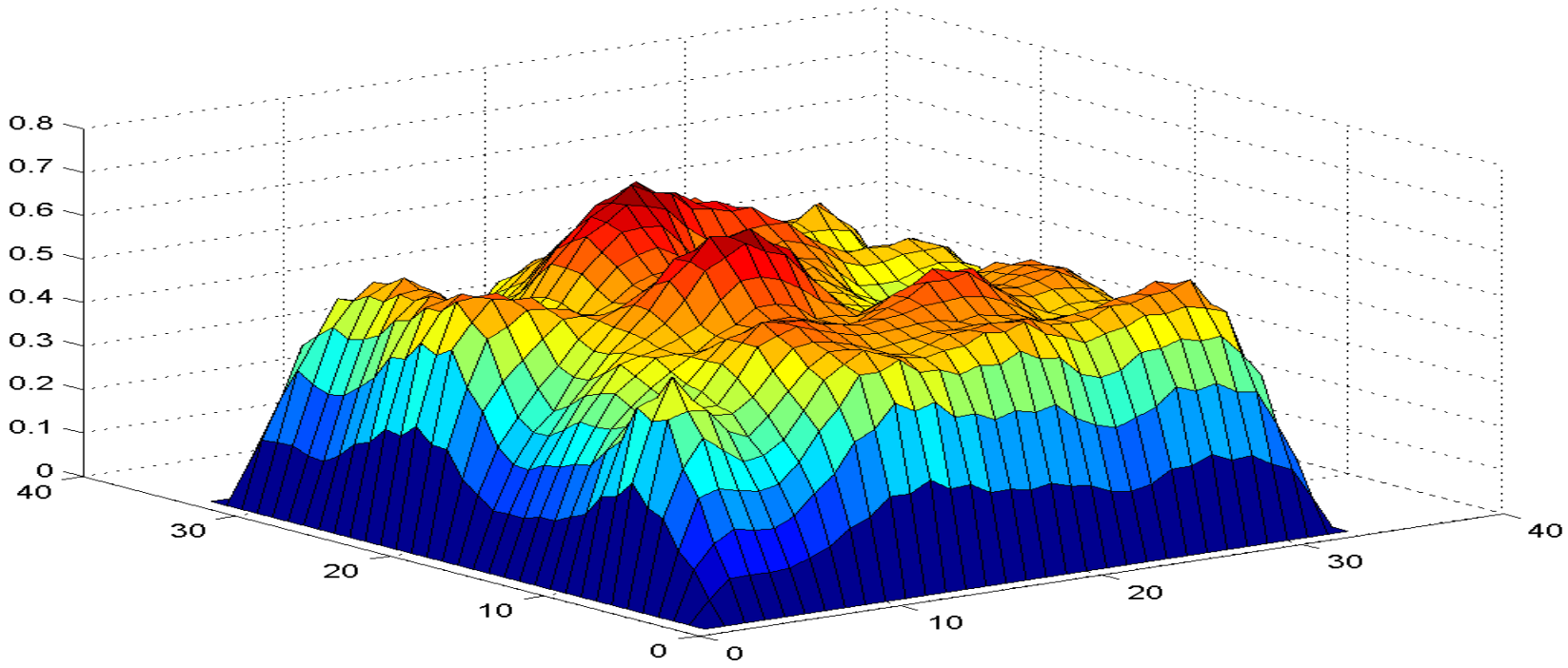
Let u^h, g^h and f^h denote discrete approximations to u, g and f defined at the nodes of the grid.

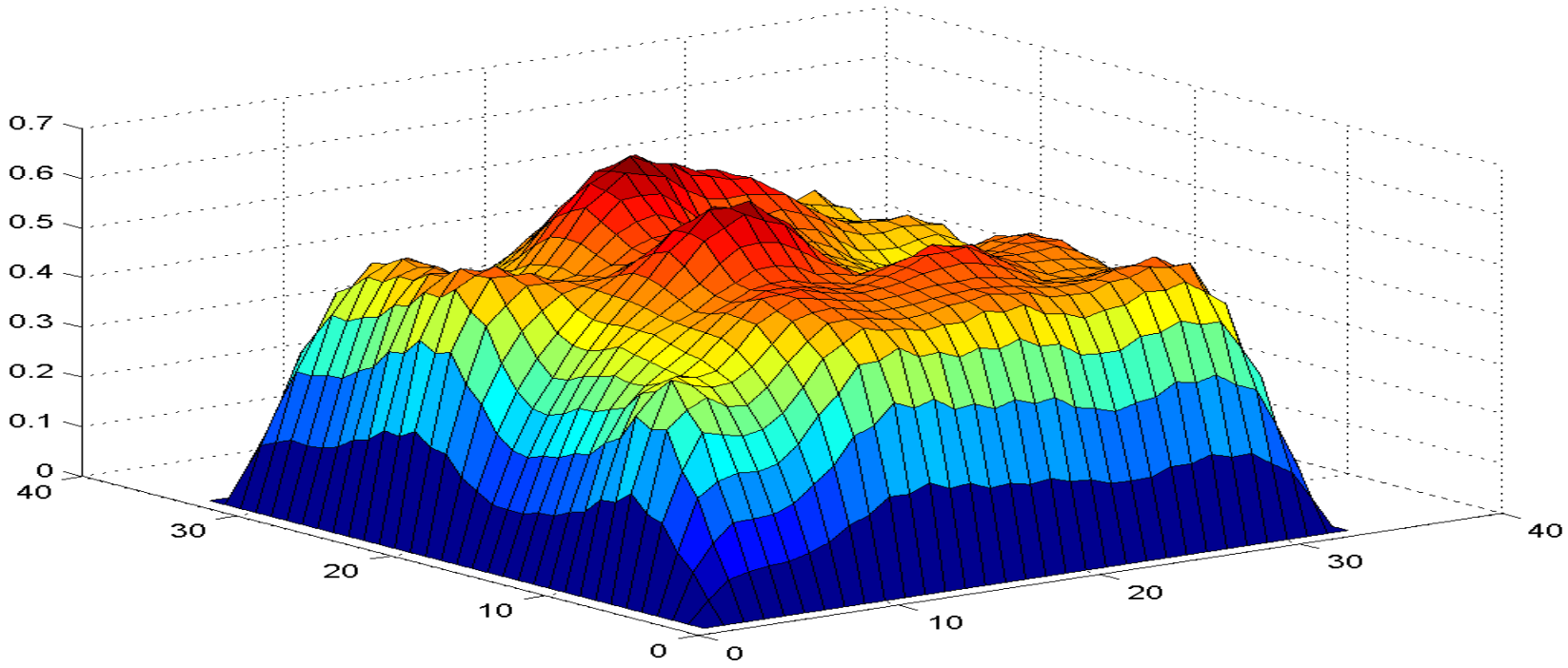
We plug in discrete approximations for the derivatives, obtaining:

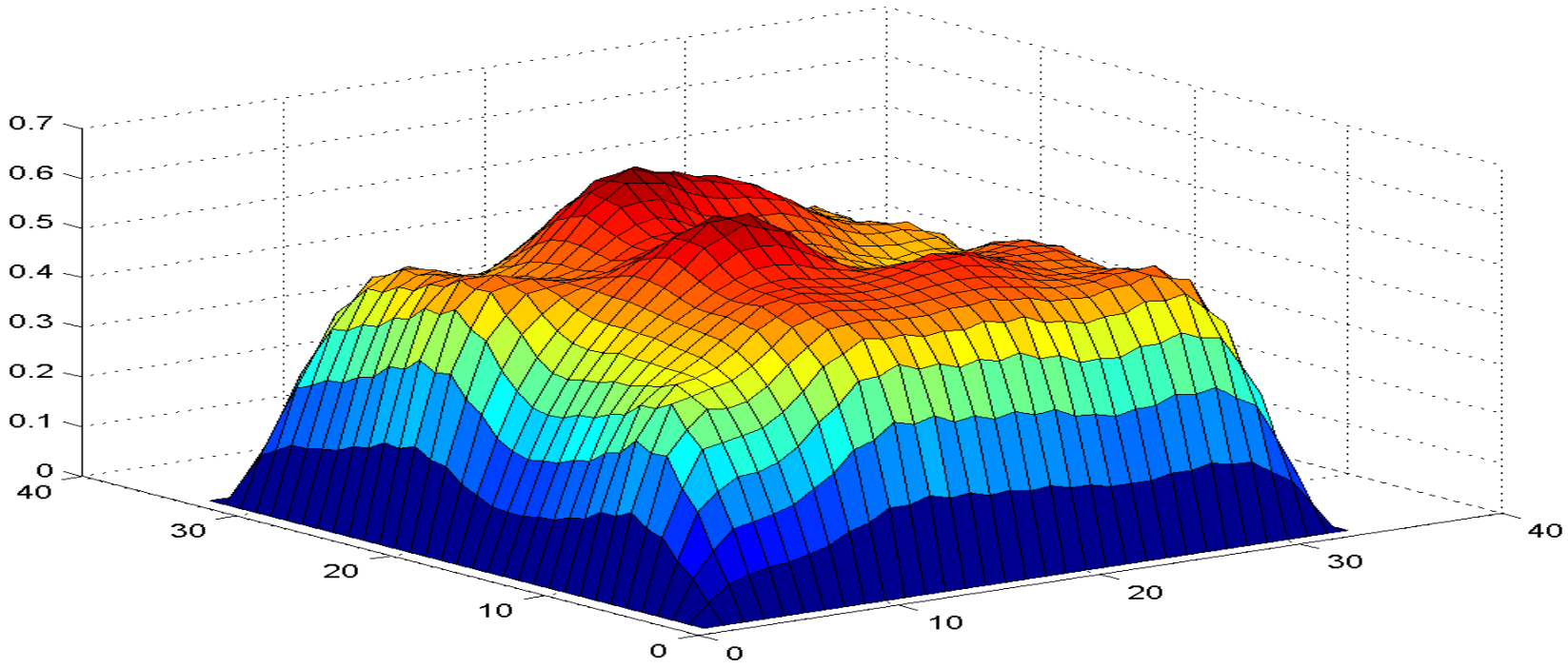

1
0.8
0.6
0.4
0.2
0
40
30
20
10
0
0
10
20
30
40

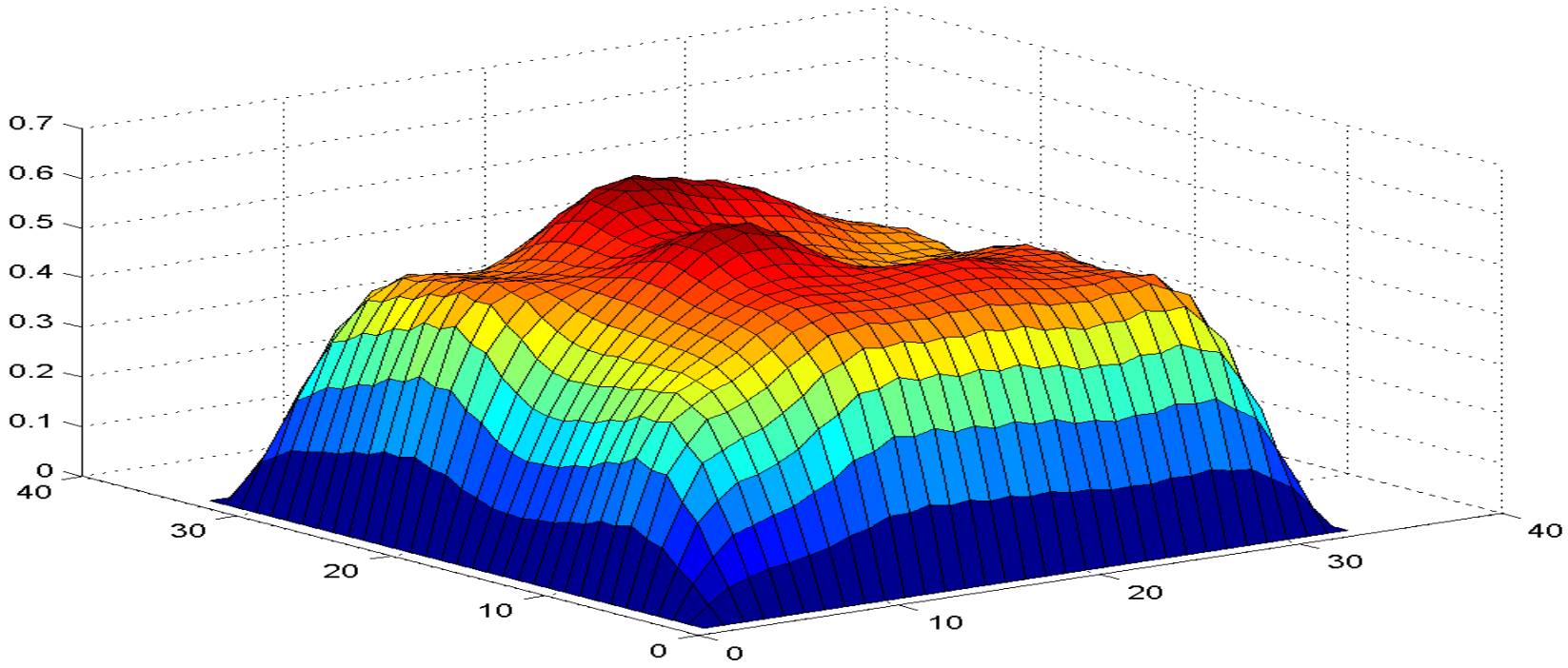

0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

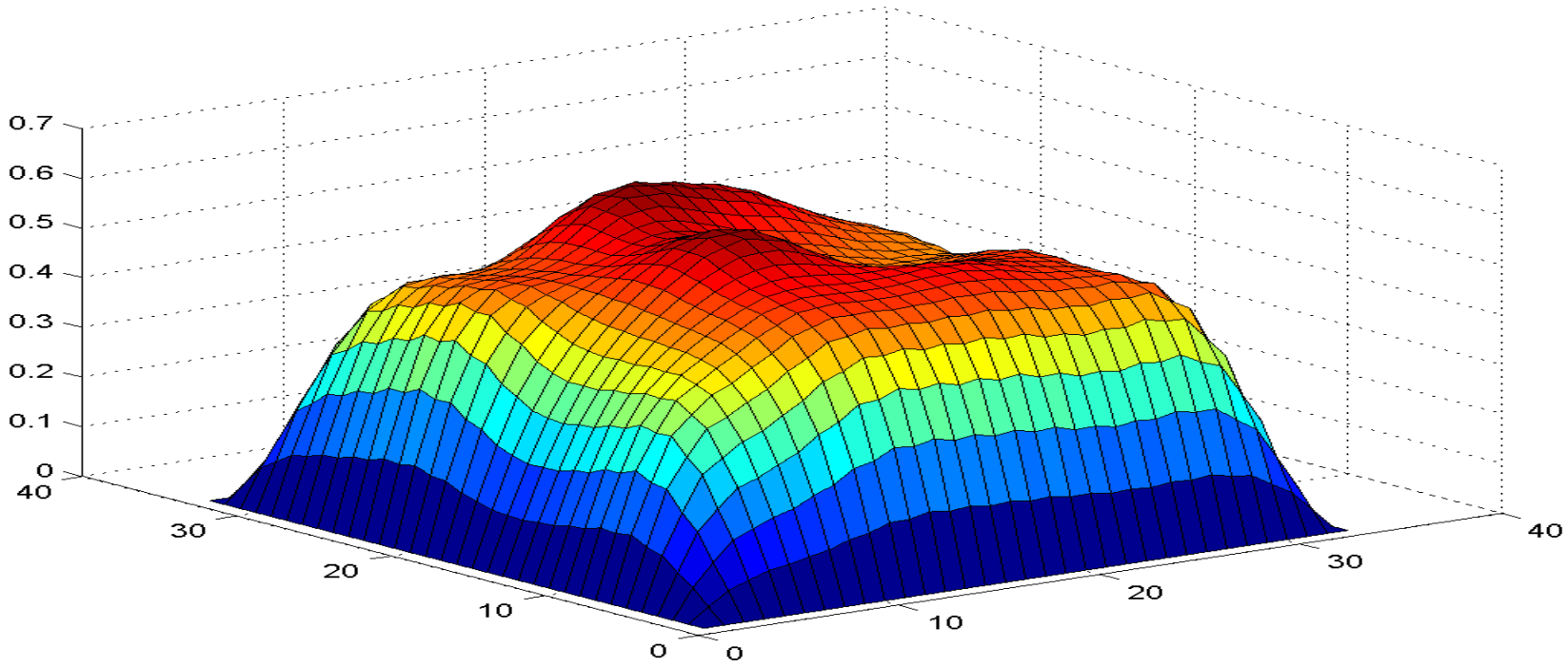

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

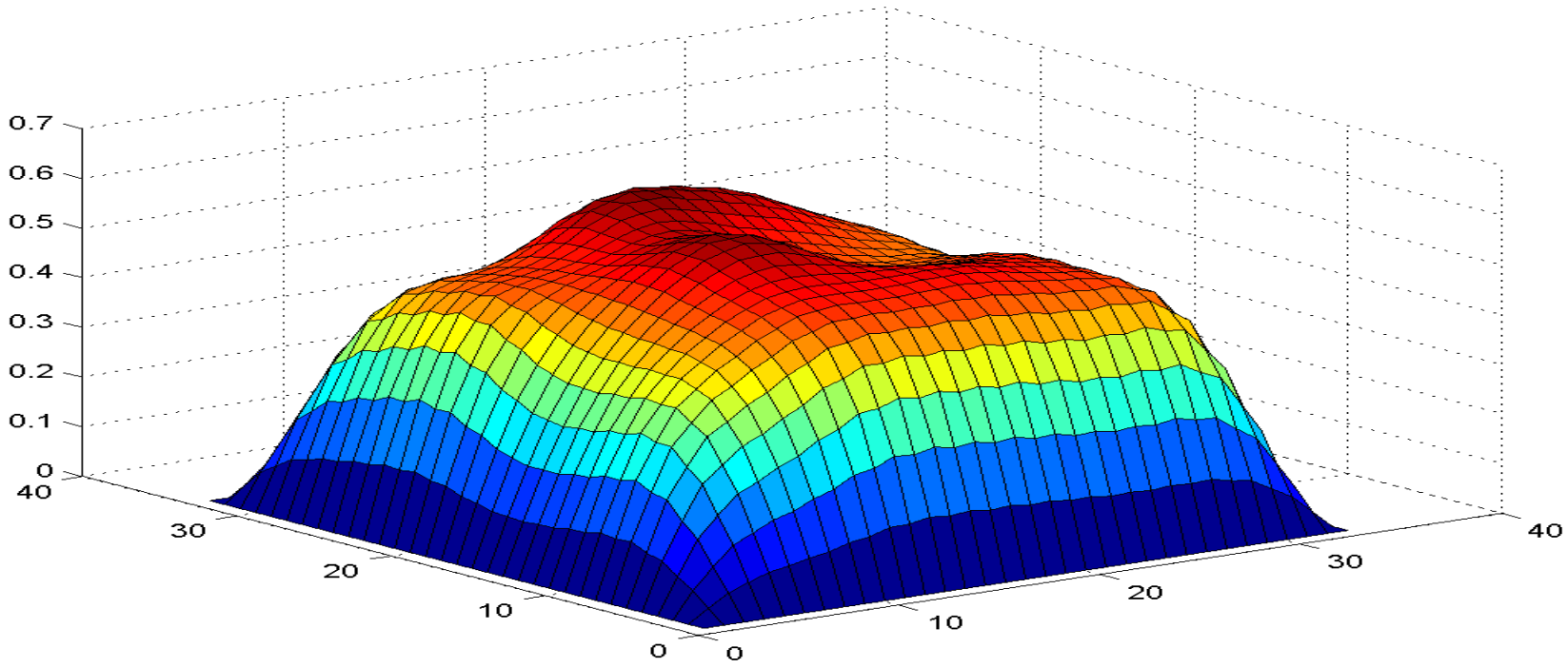

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

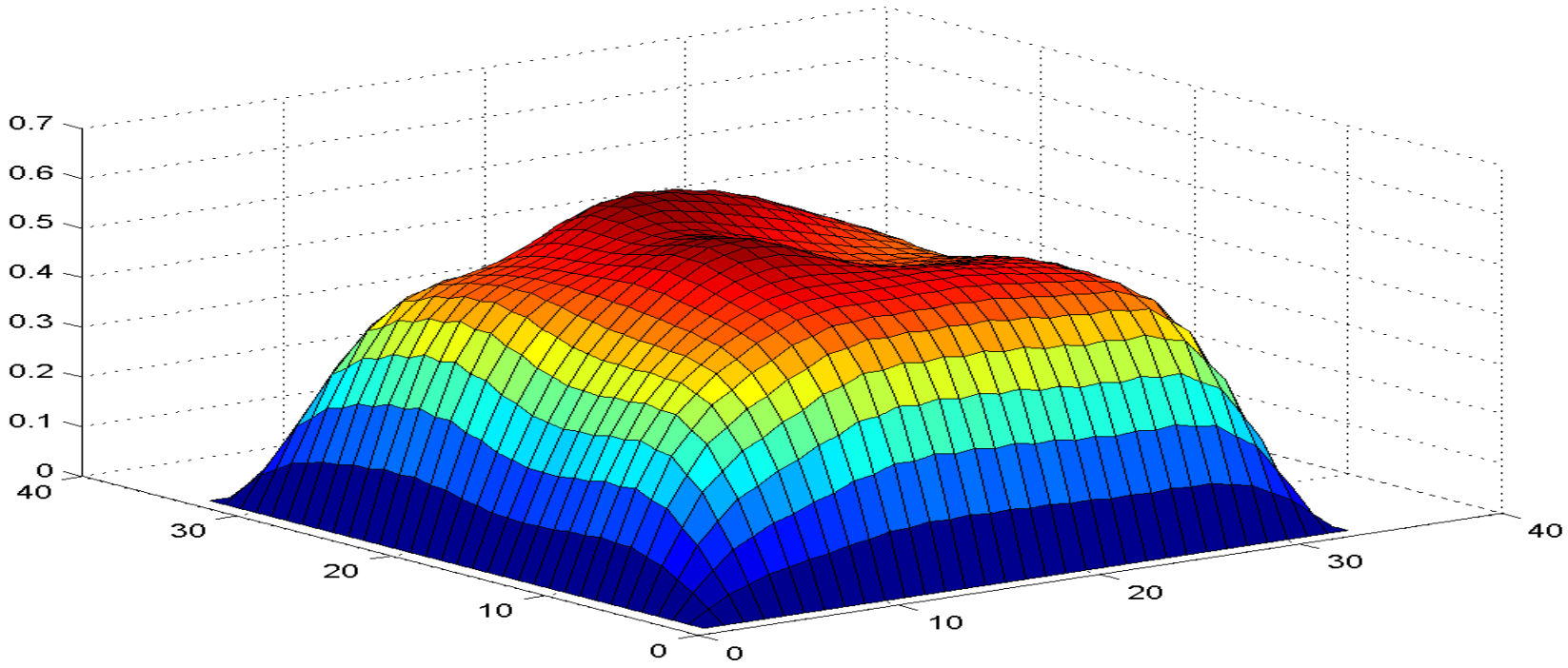

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

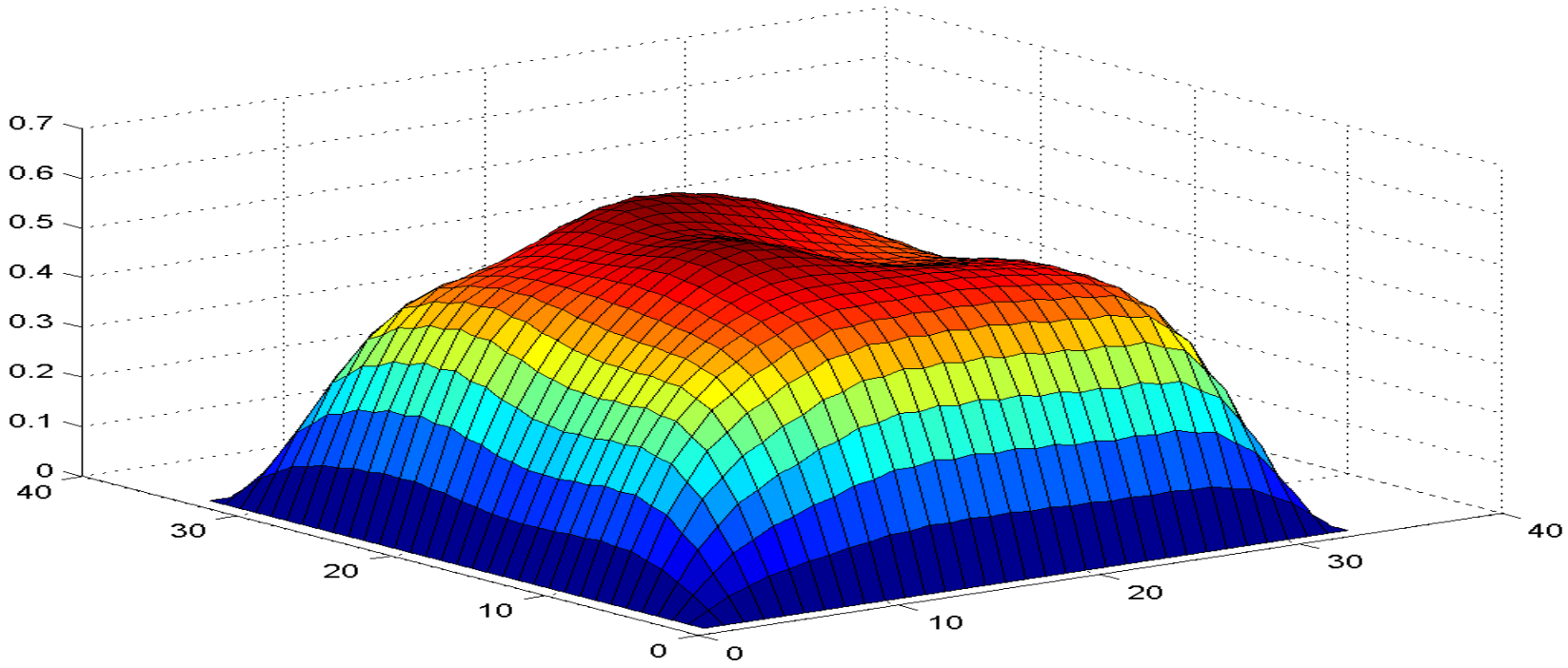

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

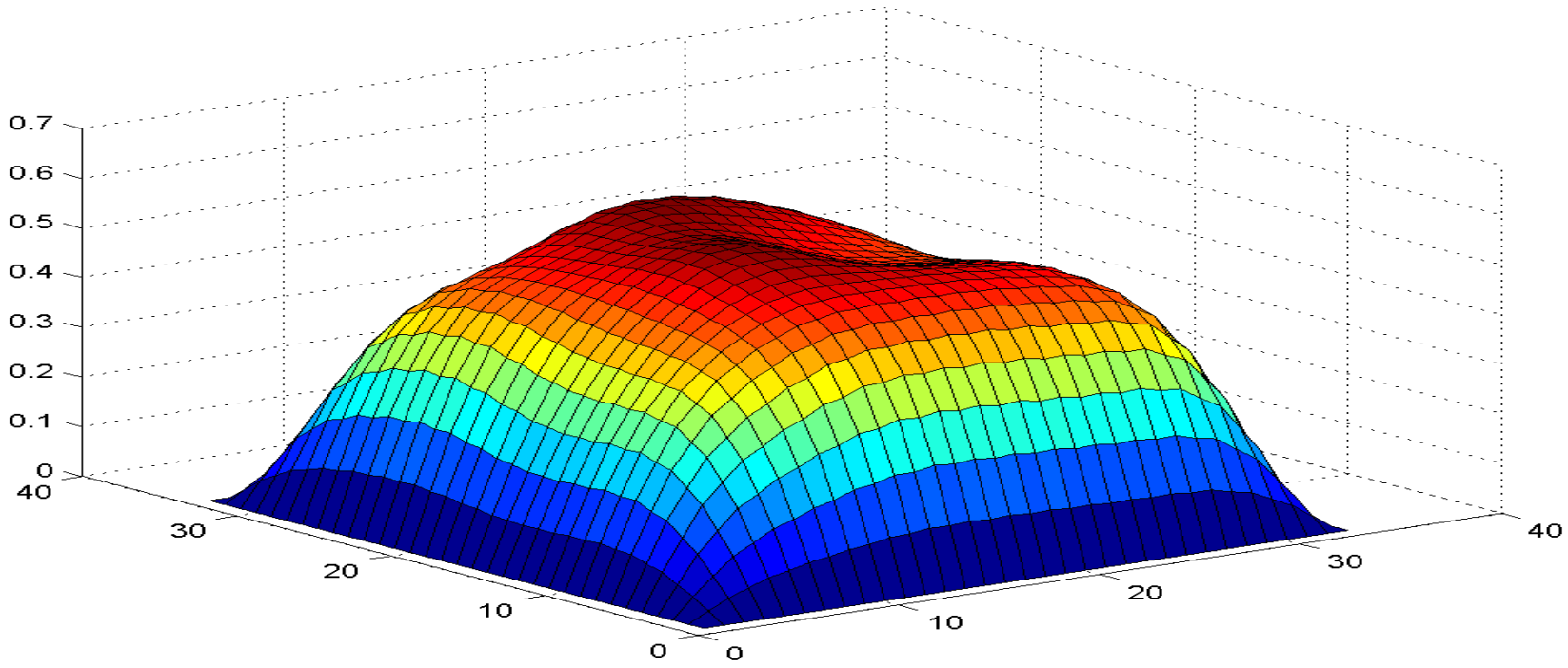

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

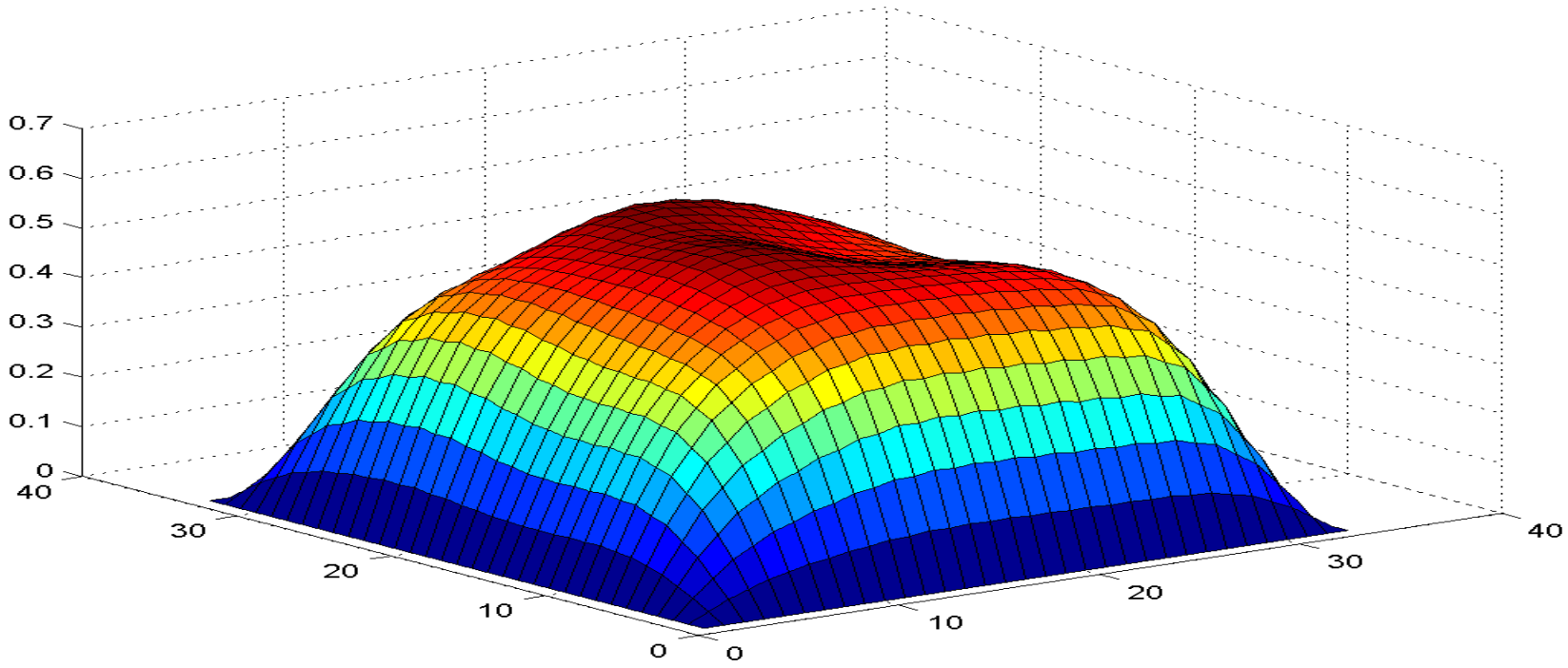

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

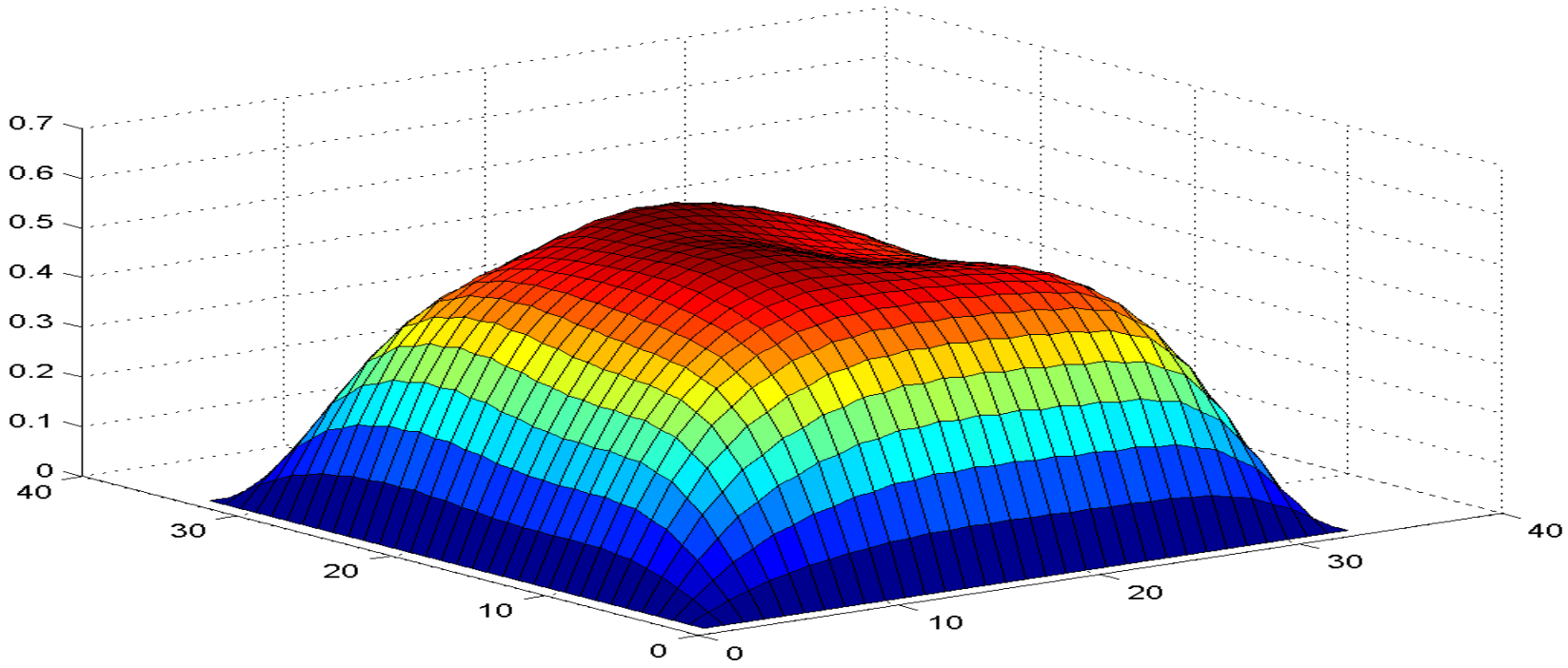

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

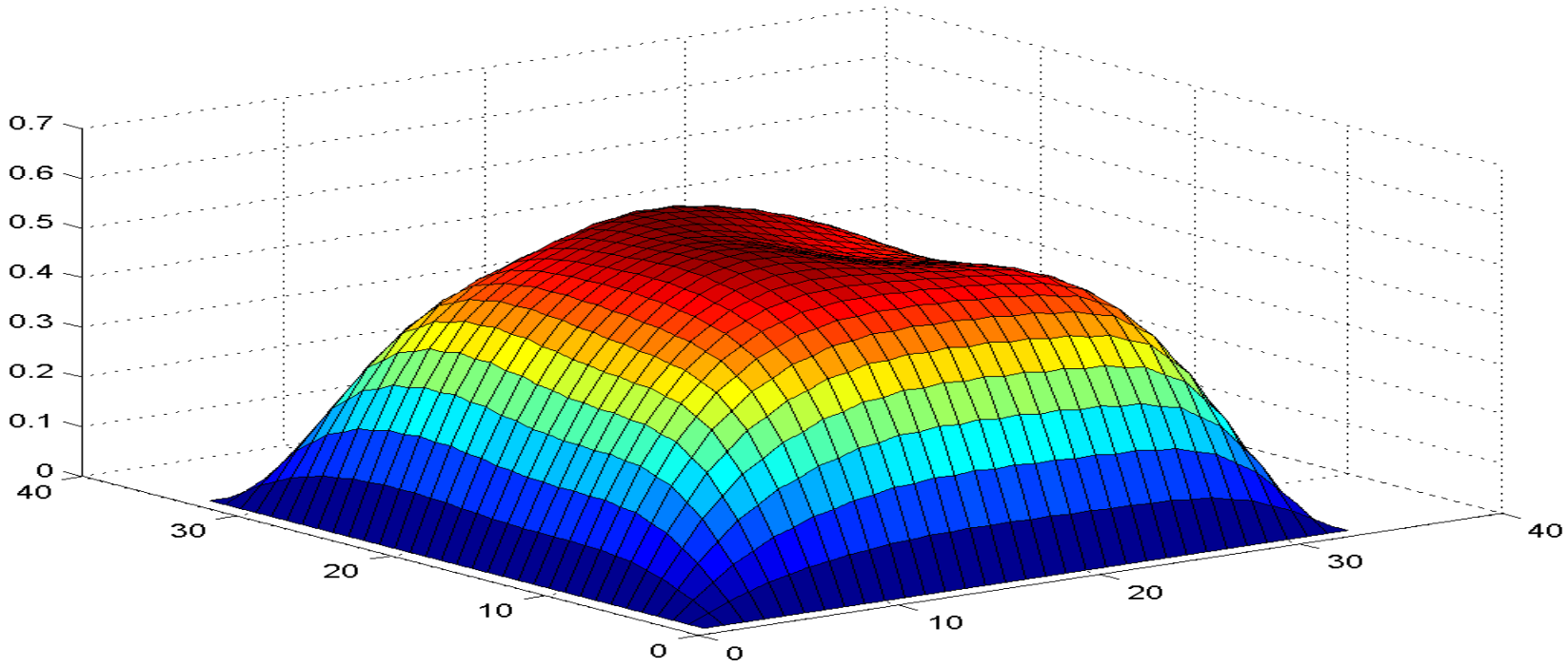

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

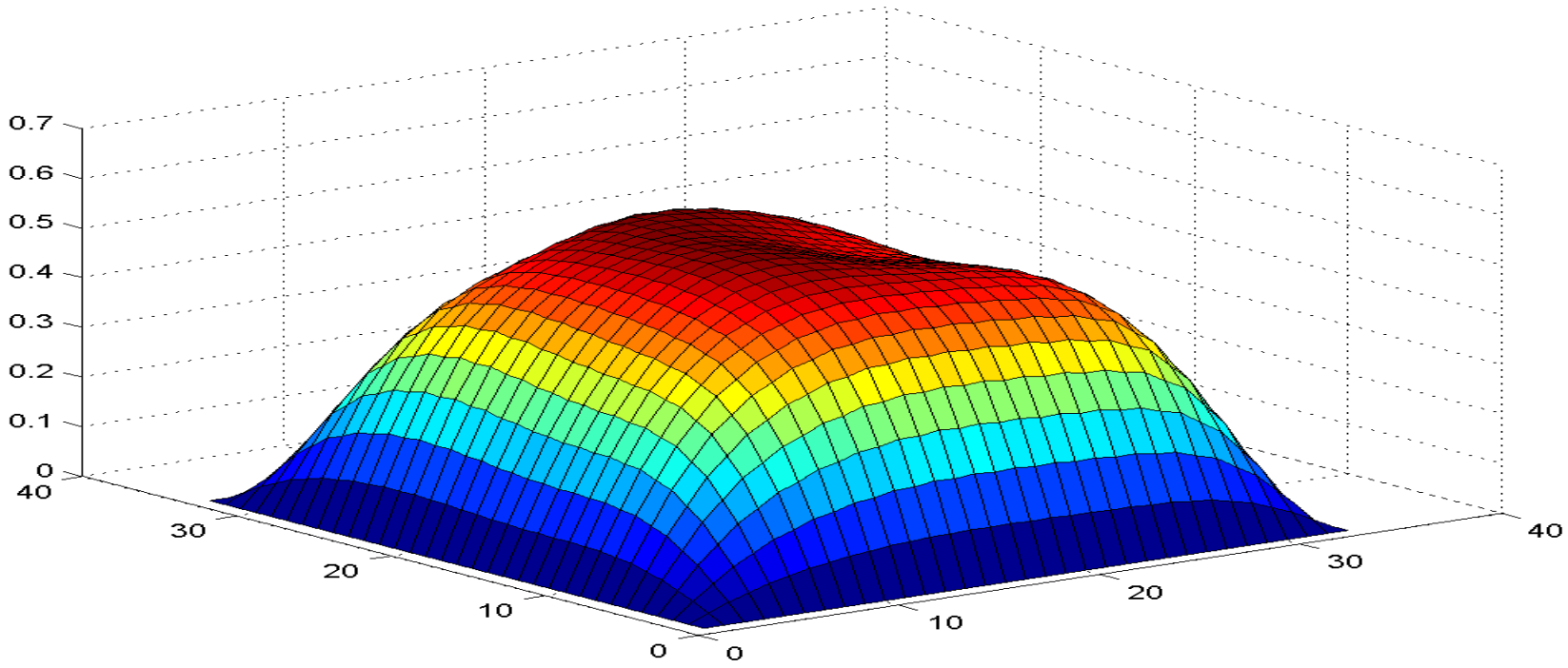

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

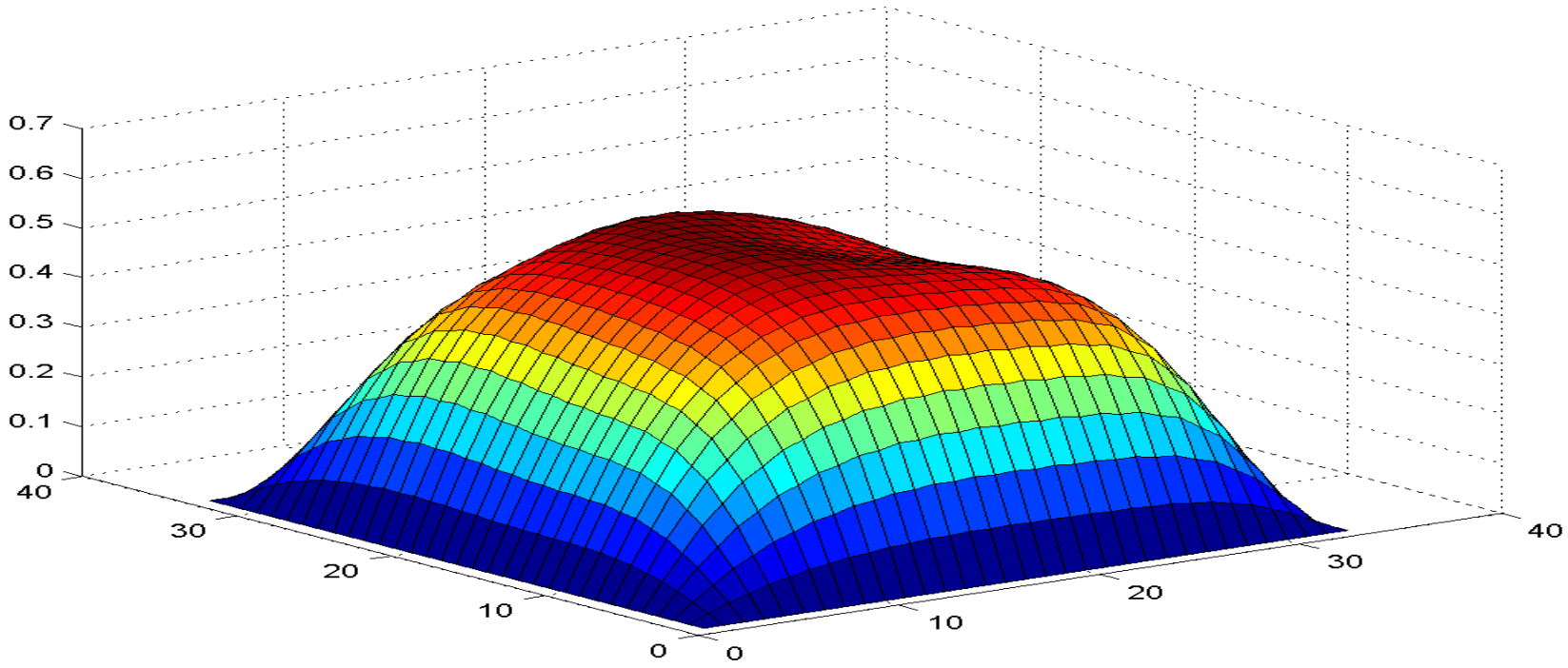

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

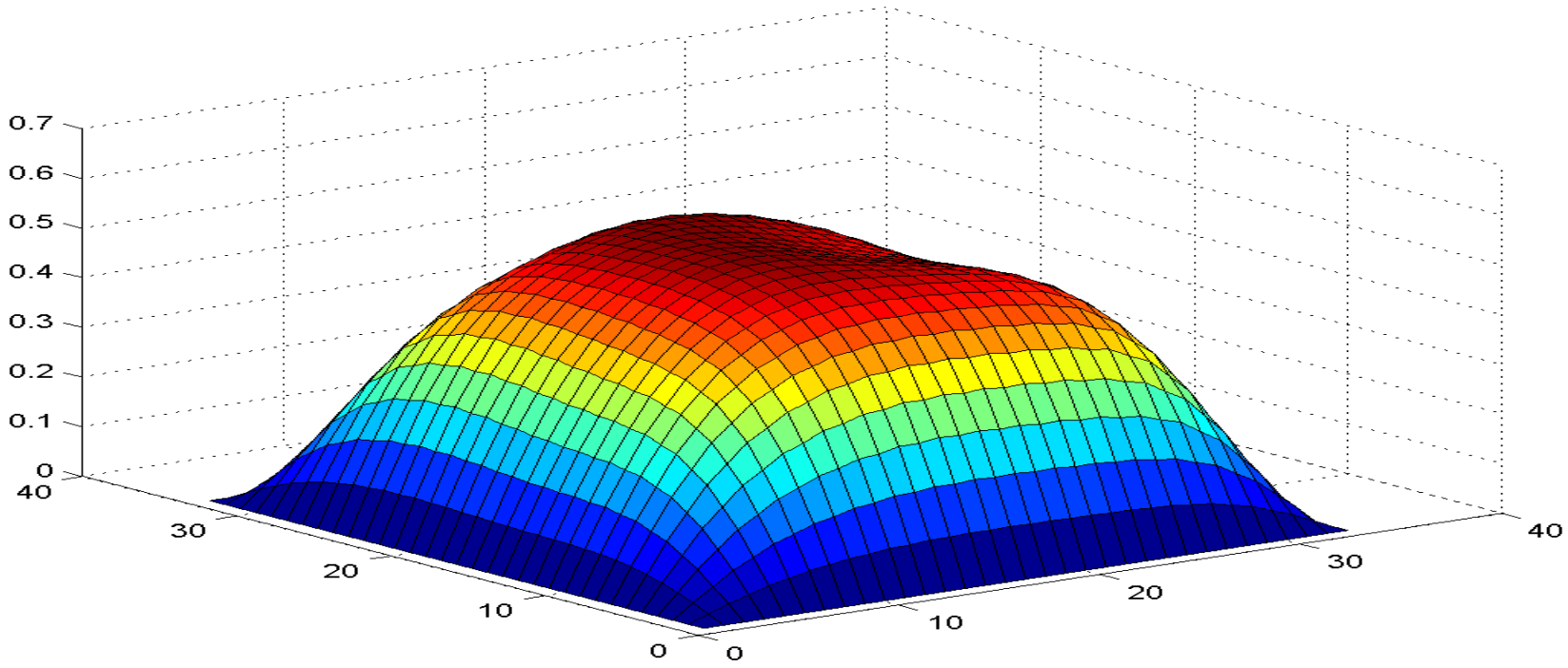

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

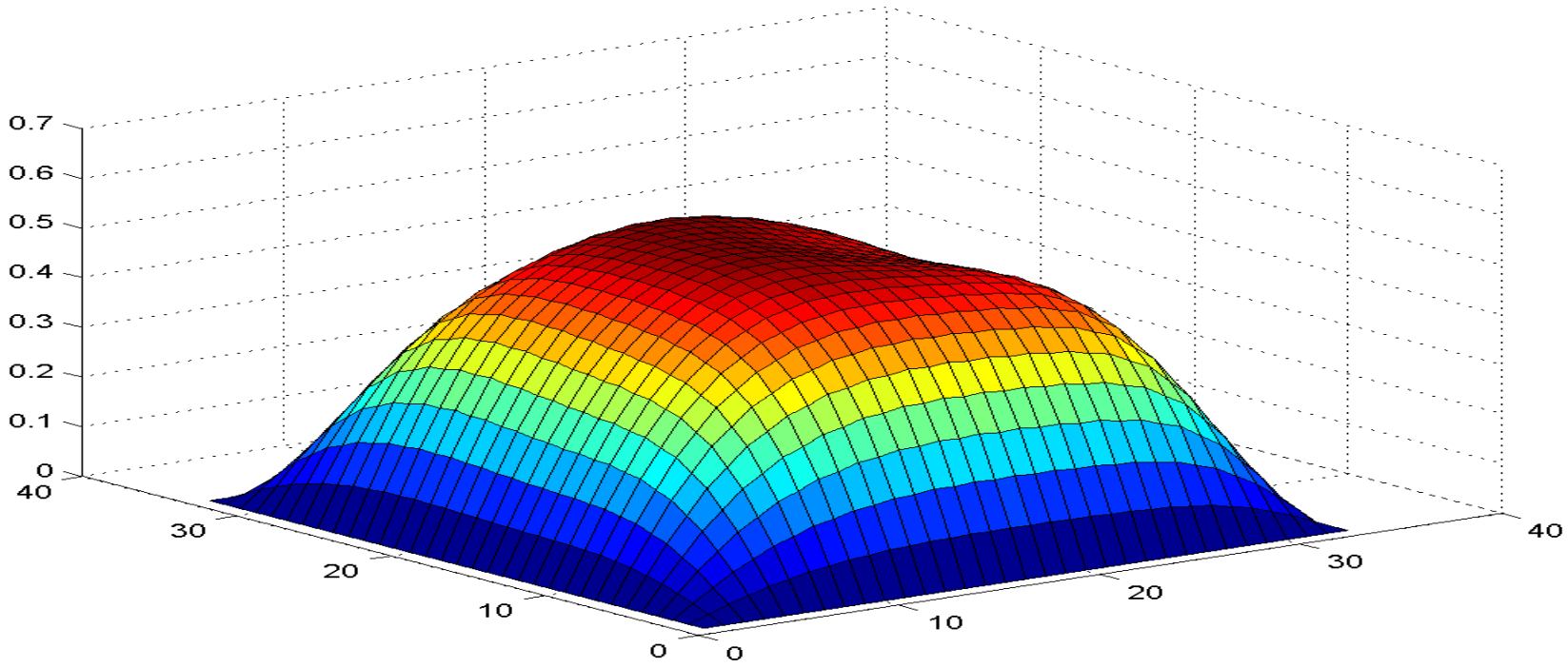

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

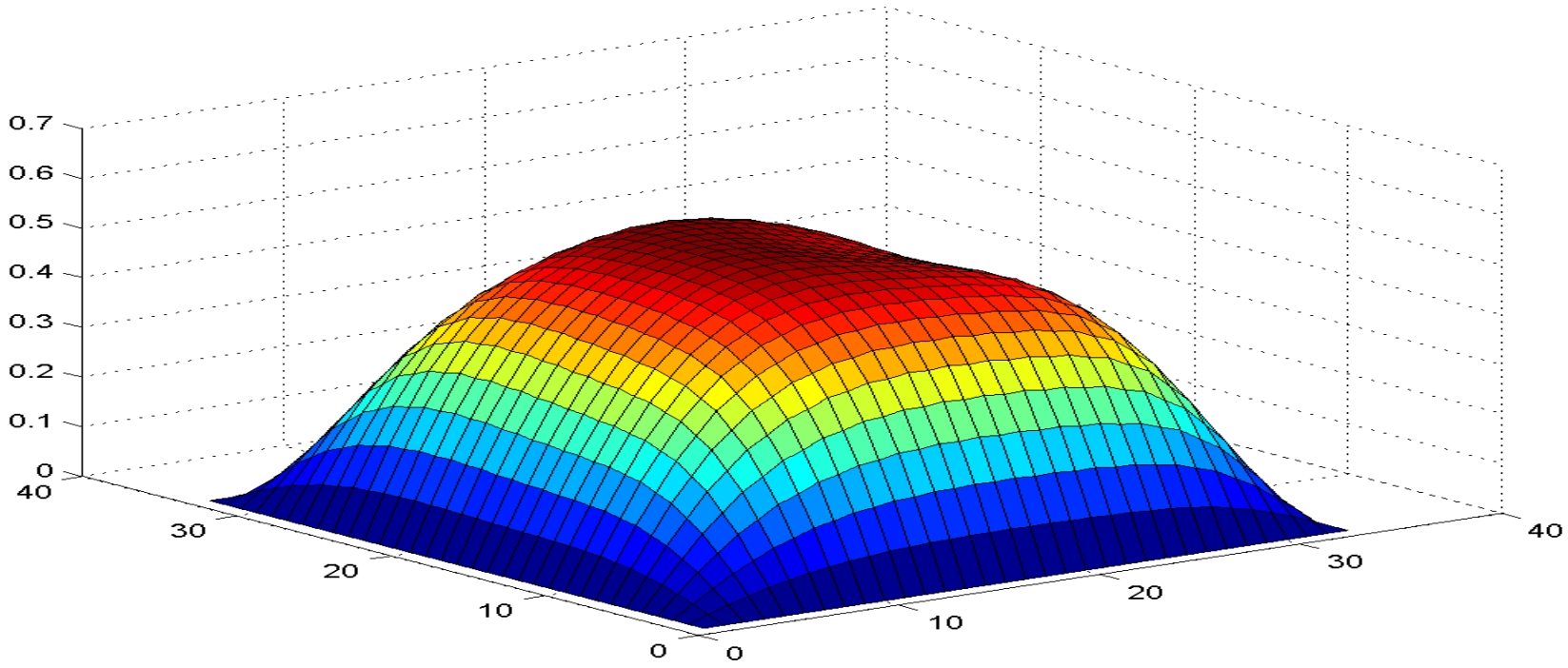

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

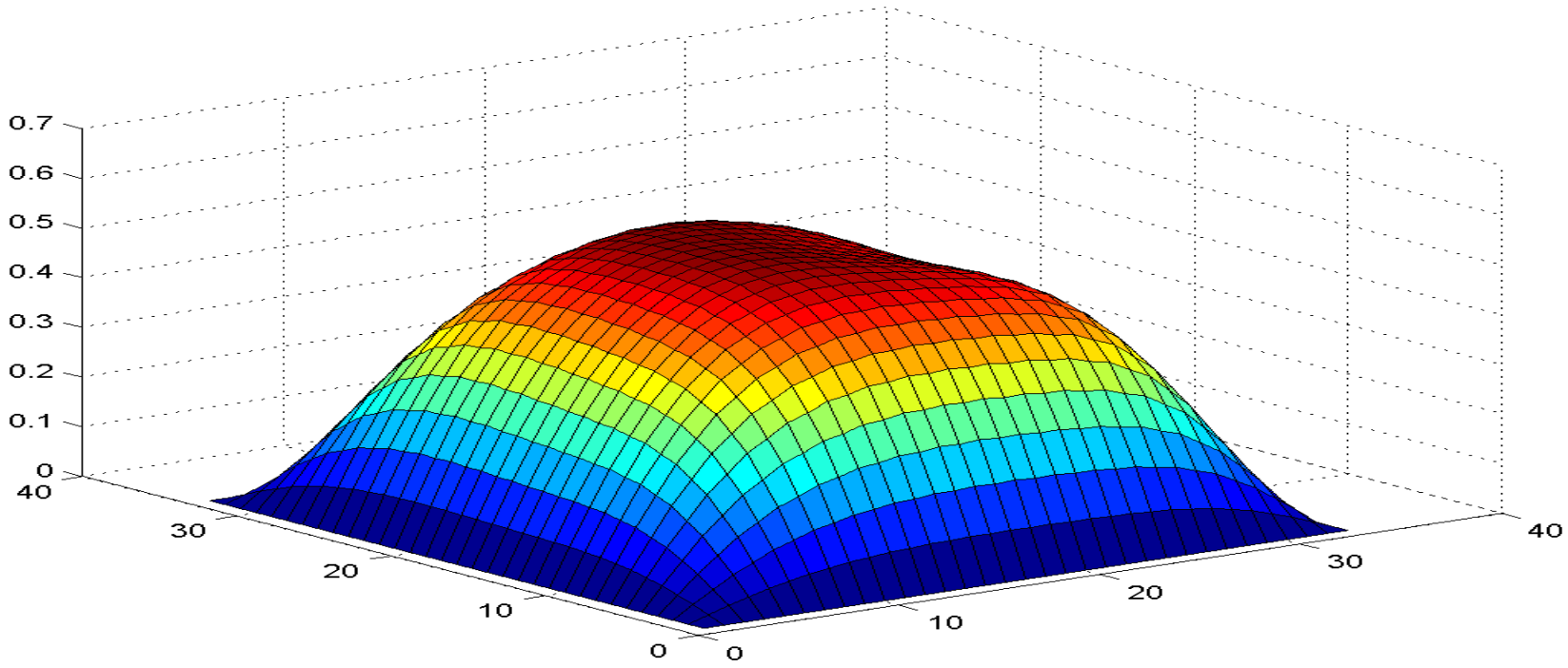

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

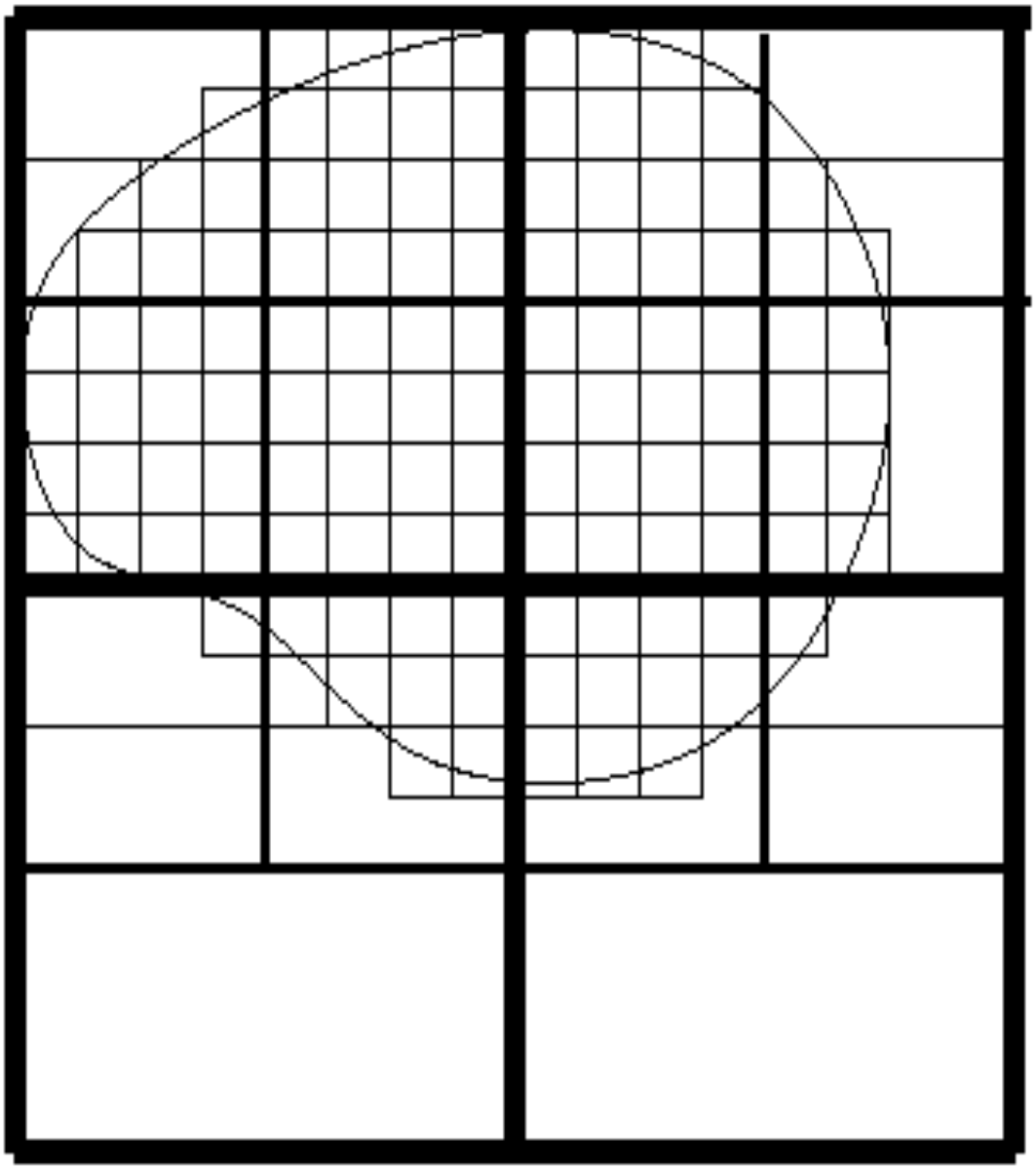

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
40
30
20
10
0
0
10
20
30
40

Key Observation re-worded: Relaxation cannot be generally efficient for reducing the error (i.e., the difference field $\tilde{u}^h - u^h$). But relaxation may be extremely efficient for smoothing the error relative to the grid.

Practical conclusion:

1. A smooth error can be approximated well on a coarser grid.
2. A coarser grid implies less variables, hence less computation.
3. On the coarser grid the error is no longer as smooth relative to the grid, so relaxation may once again be efficient.



Hierarchical Grids

Aliasing

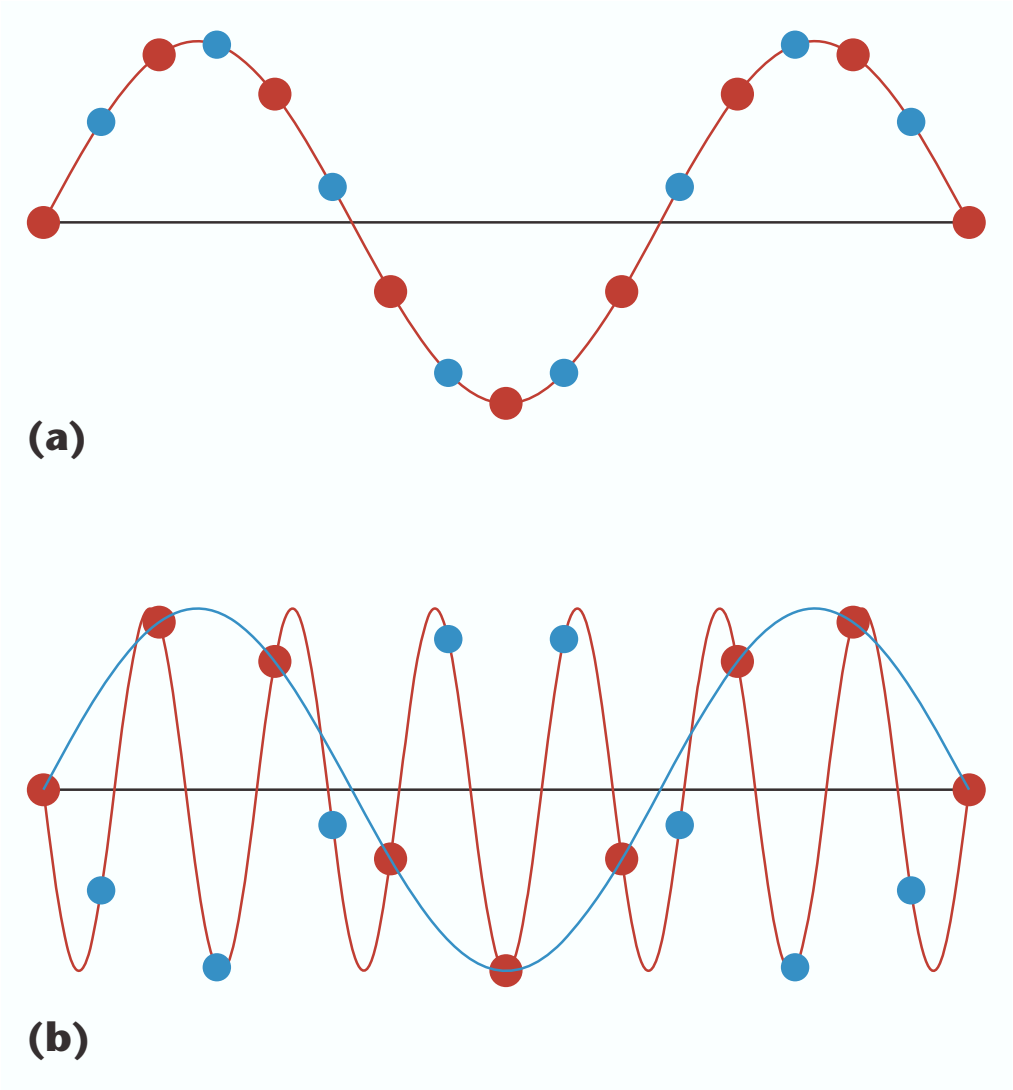


Figure 5. Aliasing. Red disks correspond to the coarse grid: (a) $v^{(3)}$ and (b) $-v^{(13)}$.

The solution, u^h, depends only on the equation and the data, so it is not, of course, smoothed by relaxation. Only the error is smoothed. Hence, we reformulate our problem:

Denote $v^h = u^h - \tilde{u}^h$.

Recall $L^h u^h = f^h$.

Subtract $L^h \tilde{u}^h$ from both sides, and use the linearity of L^h to obtain:

$$L^h v^h = f^h - L^h \tilde{u}^h \equiv r^h$$

It is this equation that we shall approximate on the coarse grids.

As we have seen, we need to smooth the error on the fine grid first, and only then solve the coarse-grid problem. Hence, we need two types of intergrid transfer operations:

1. A Restriction (fine-to-coarse) operator: I_h^H.
2. A Prolongation (coarse-to-fine) operator: I_H^h.

For restriction we can often use simple injection, but full-weighted (local averaging) transfers are preferable.

For prolongation, linear interpolation (bi-linear in 2D) is simple and usually effective.

Two-grid Algorithm

✧ Relax several times on grid h, obtaining $\tilde{u}^h$ with a smooth corresponding error.

✧ Calculate the residual: $r^h = f^h - L^h \tilde{u}^h .$

✧ Solve approximate error-equation on the coarse grid: $L^H v^H = f^H \equiv I_h^H r^h .$

✧ Interpolate and add correction: $\tilde{u}^h \leftarrow \tilde{u}^h + I_H^h v^H .$

✧ Relax again on grid h.

Multi-grid is obtained by recursion.

Multi-grid Cycle $V(\nu_1,\nu_2)$

Relax on $L^h u^h = f^h$ ν_1 **times**

Set $f^{2h} = I_h^{2h}\left(f^h - L^h u^h\right)$, $u^{2h} = 0$

Relax on $L^{2h} u^{2h} = f^{2h}$ ν_1 **times**

Set $f^{4h} = I_{2h}^{4h}\left(f^{2h} - L^{2h} u^{2h}\right)$, $u^{4h} = 0$

Relax on $L^{4h} u^{4h} = f^{4h}$ ν_1 **times**

Set $f^{8h} = I_{4h}^{8h}\left(f^{4h} - L^{4h} u^{4h}\right)$, $u^{8h} = 0$

...

Solve $L^{Mh} - u^{Mh} = f^{Mh}$

...

Correct $u^{4h} \leftarrow u^{4h} + I_{8h}^{4h} u^{8h}$

Relax on $L^{4h} u^{4h} = f^{4h}$ ν_2 **times**

Correct $u^{2h} \leftarrow u^{2h} + I_{4h}^{2h} u^{4h}$

Relax on $L^{2h} u^{2h} = f^{2h}$ ν_2 **times**

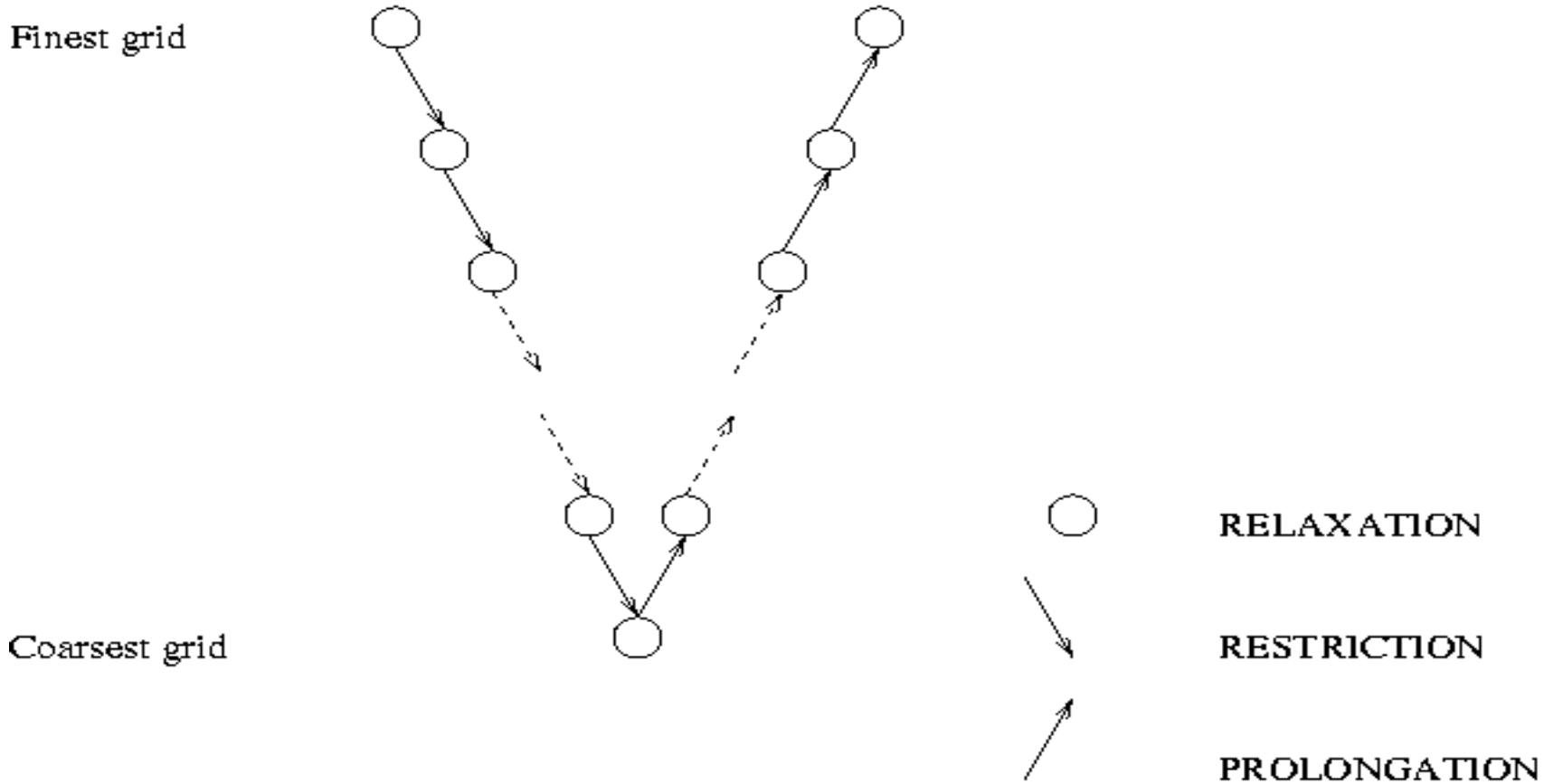
Correct $u^h \leftarrow u^h + I_{2h}^h u^{2h}$

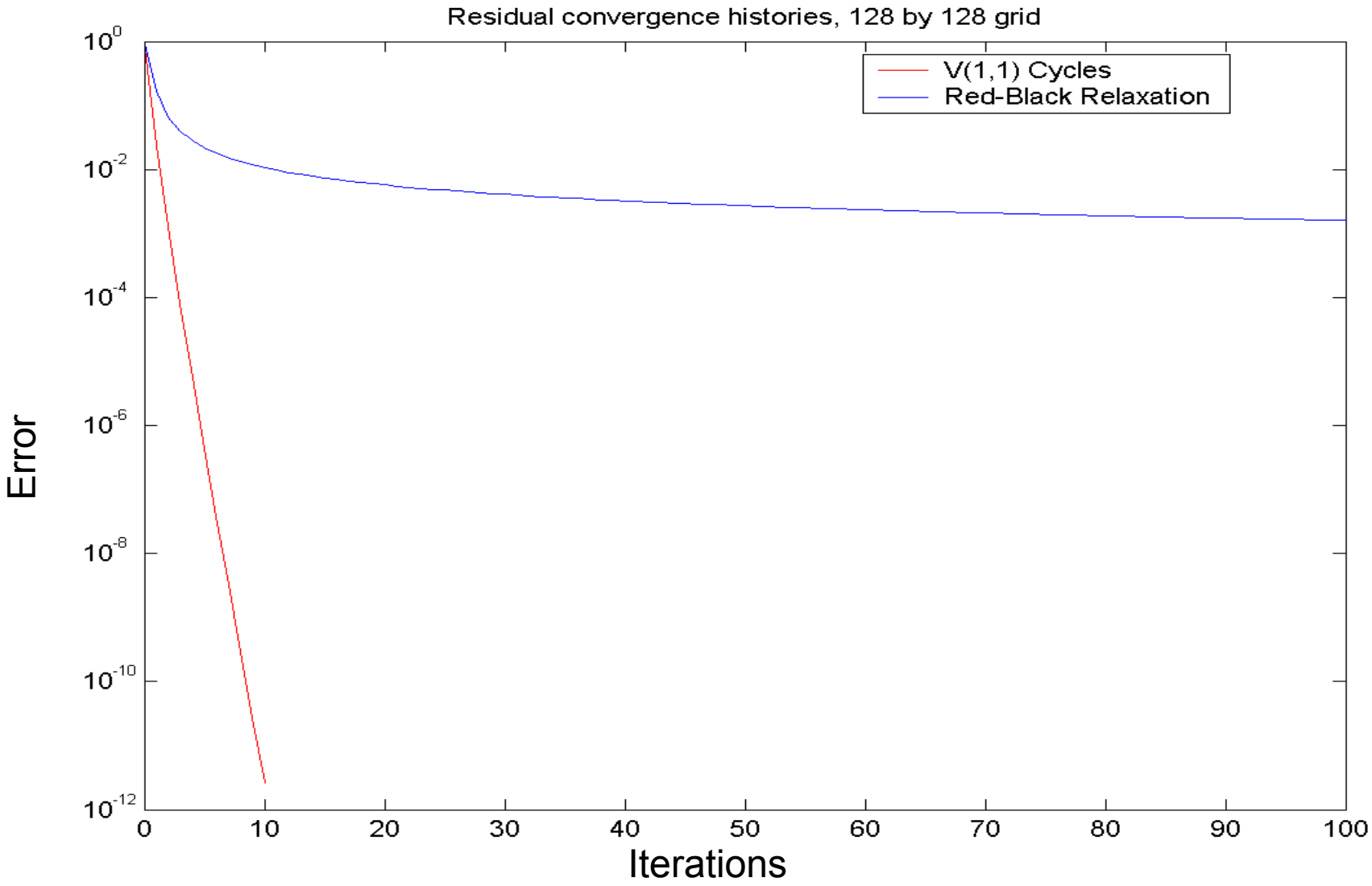
Relax on $L^h u^h = f^h$ ν_2 **times**

Remarks:

1. Simple recursion yields a *V* cycle. More generally, we can choose a cycle index $\tilde{a}$, and define a $\tilde{a}$–cycle recursively as follows: Relax; transfer to next coarser grid; perform $\tilde{a}$ cycles; interpolate and correct; Relax. (On the coarser grid define the cycle as an exact solution).
2. The best number of pre-relaxation + post-relaxation sweeps is normally 2 or 3.
3. The boundary conditions for all coarse-grid problems is zero (because the coarse-grid variable is the error). The initial guess for the coarse-grid solution must be zero.

V cycle

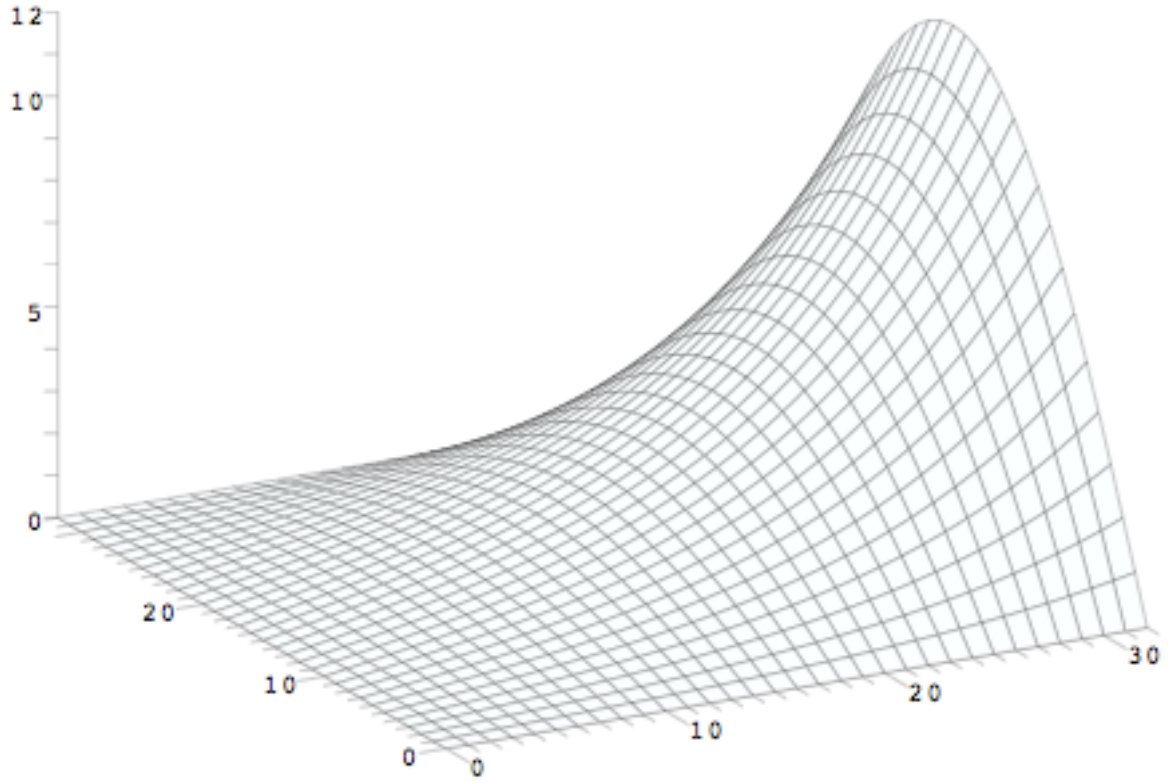

Finest grid
Coarsest grid
RELAXATION
RESTRICTION
PROLONGATION



Multigrid vs. Relaxation

Solution of Poisson‘s eqn

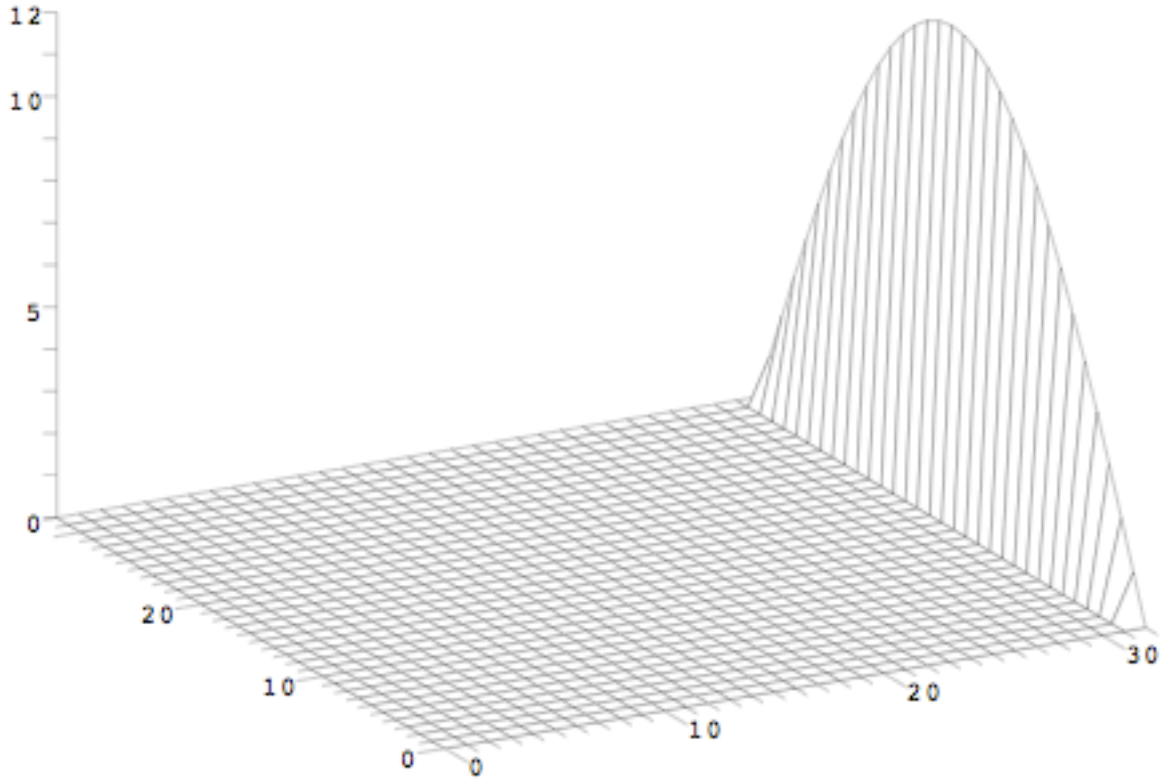
```
enorm = 9.95593313654029e-02
```




```
File: c/t
```

Dirichlet boundary conditions

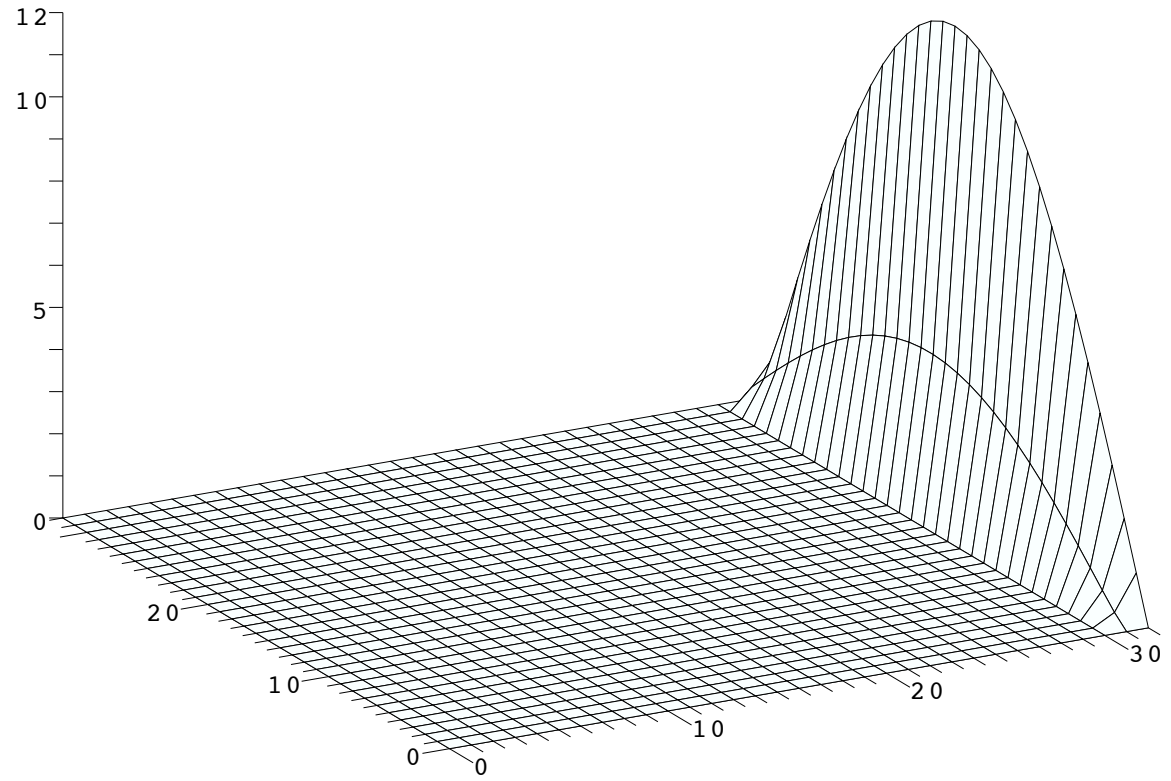
```
enorm = 4.24196119642158e-02
```



File: c/u

After one step of Gauss-Seidel

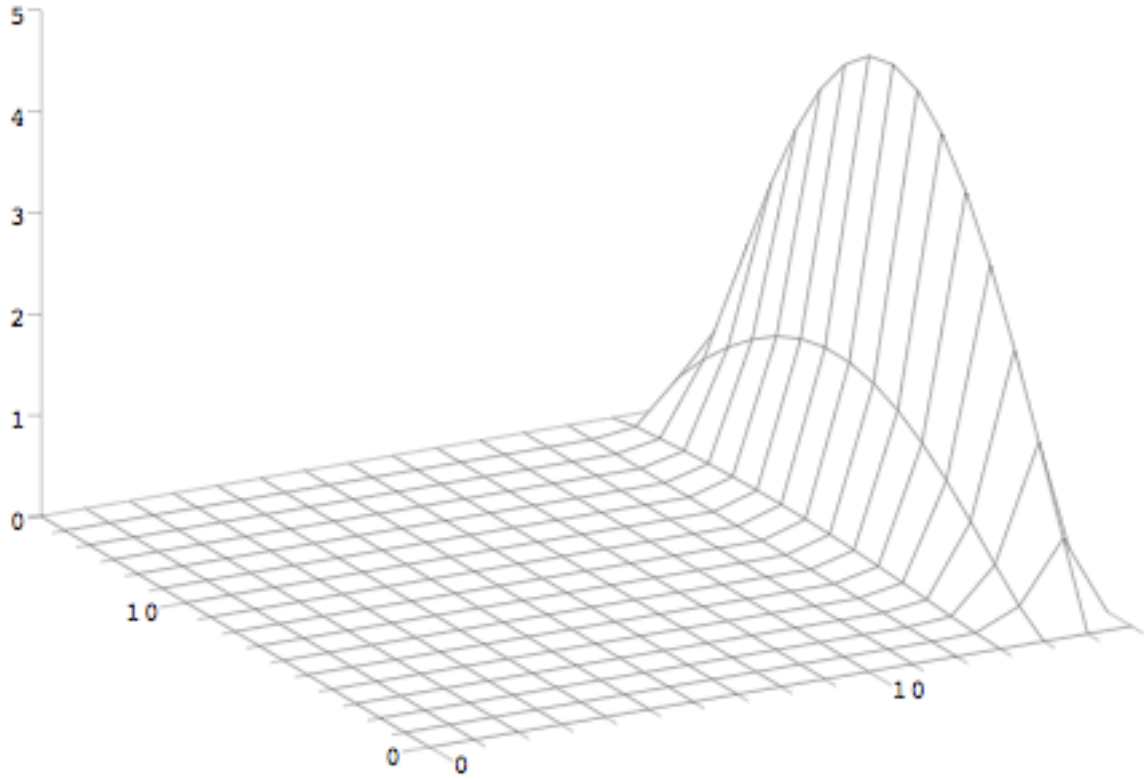
```
enorm = 4.47048252627281e-02
```



File: c/u.2

On the next coarser grid, after two Gauss-Seidel steps

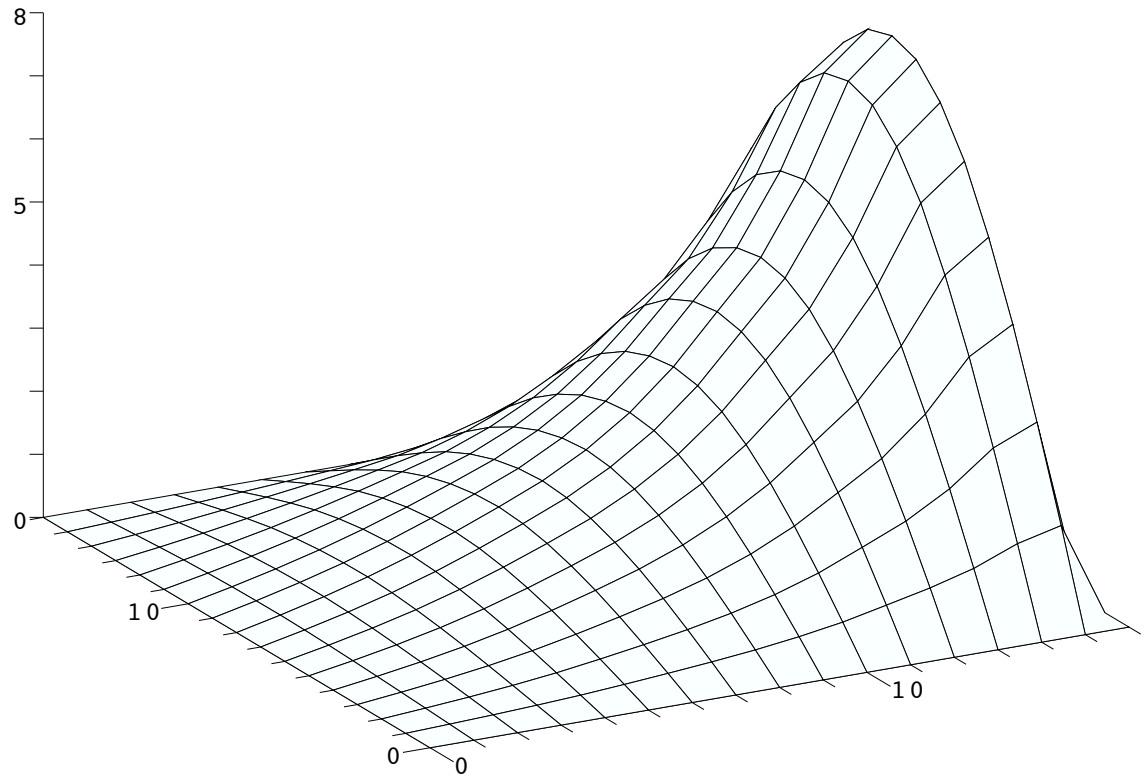
enorm = 4.67013507655193e-02



File: c/c.4/u.5

On the next coarser grid, after V-cycle reursion

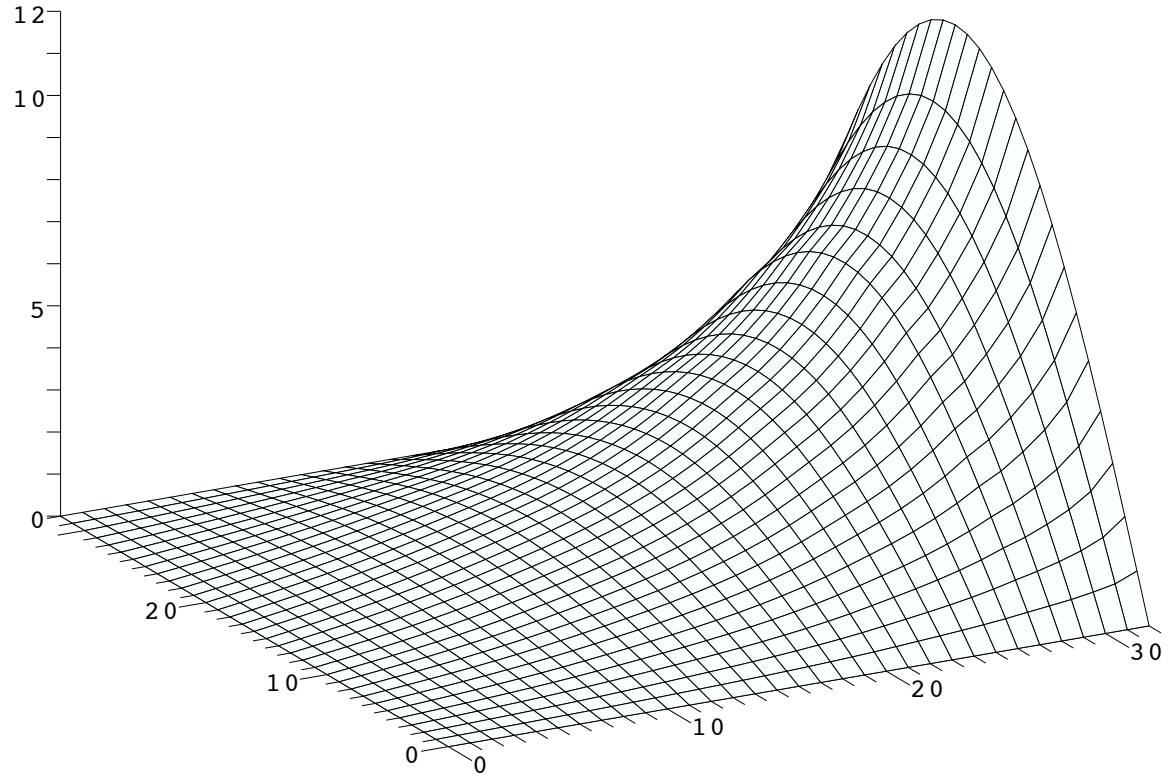
```
enorm = 1.38371516388512e-01
```



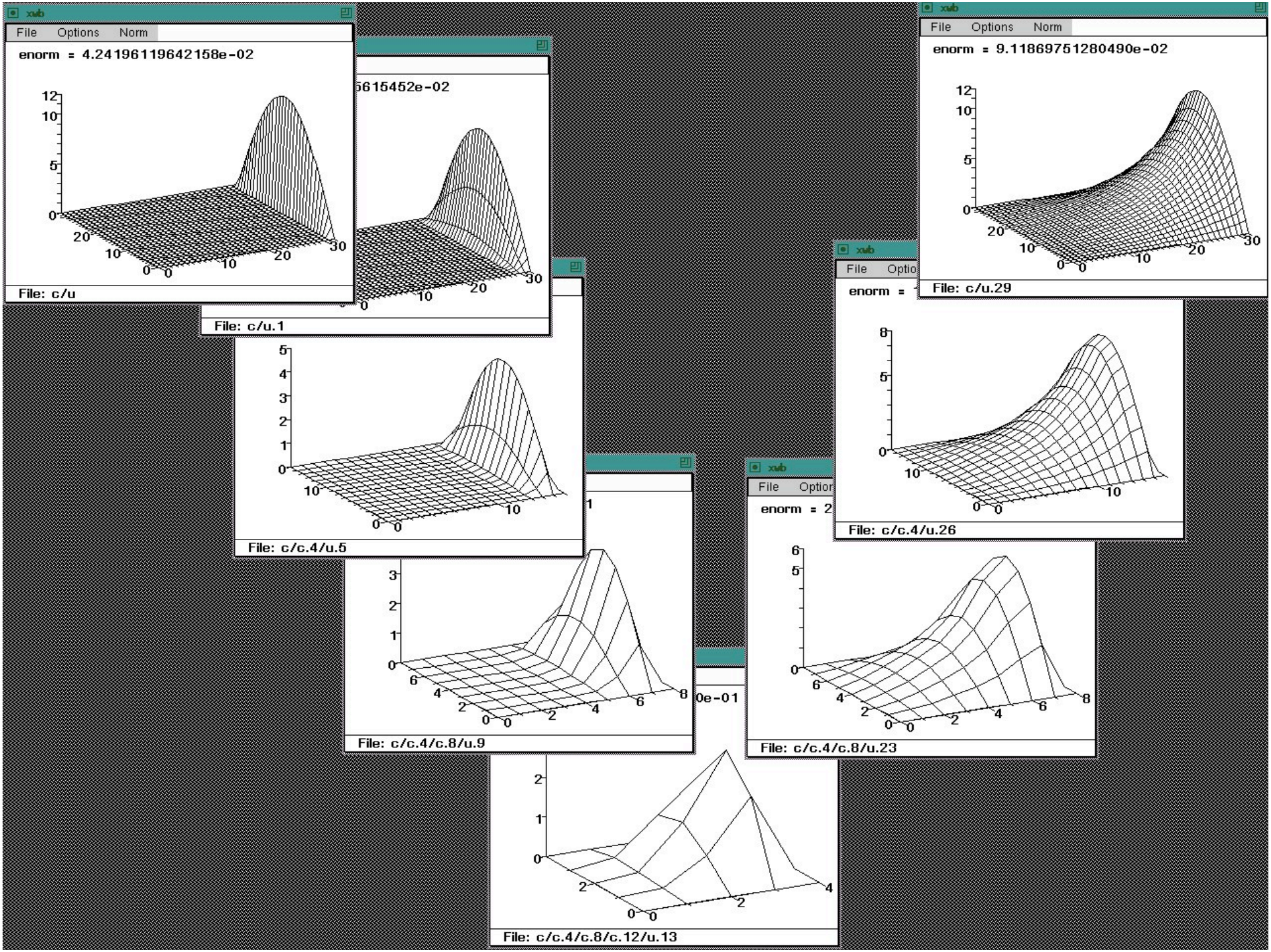
File: c/c.4/u.26

On finest grid, after coarse grid correction

```
enorm = 9.11869751280490e-02
```



File: c/u.29


xwb
File Options Norm
enorm = 4.24196119642158e-02
File: c/u
5615452e-02
File: c/u.1
File: c/c.4/u.5
File: c/c.4/c.8/u.9
0e-01
File: c/c.4/c.8/c.12/u.13
File Optio
enorm =
File: c/c.4/u.26
File Optior
enorm = 2
File: c/c.4/c.8/u.23
File Options Norm
enorm = 9.11869751280490e-02
File: c/u.29

The **cost** of the *V* cycle in terms of computation and storage is given by

$$cN \sum_{k=0}^{levels-1} 2^{-dk} < cN \frac{2^d}{2^d - 1}$$

Where *d* is the dimension and *N* is the number of variables on the finest grid. Here, *c* is some constant that depends on the discrete operators and the number of relaxation sweeps per level.

Thus, for a 2D problem, the *V*-cycle with one pre-relaxation and one post-relaxation requires approximately the same number of operations as 3-5 relaxation sweeps.

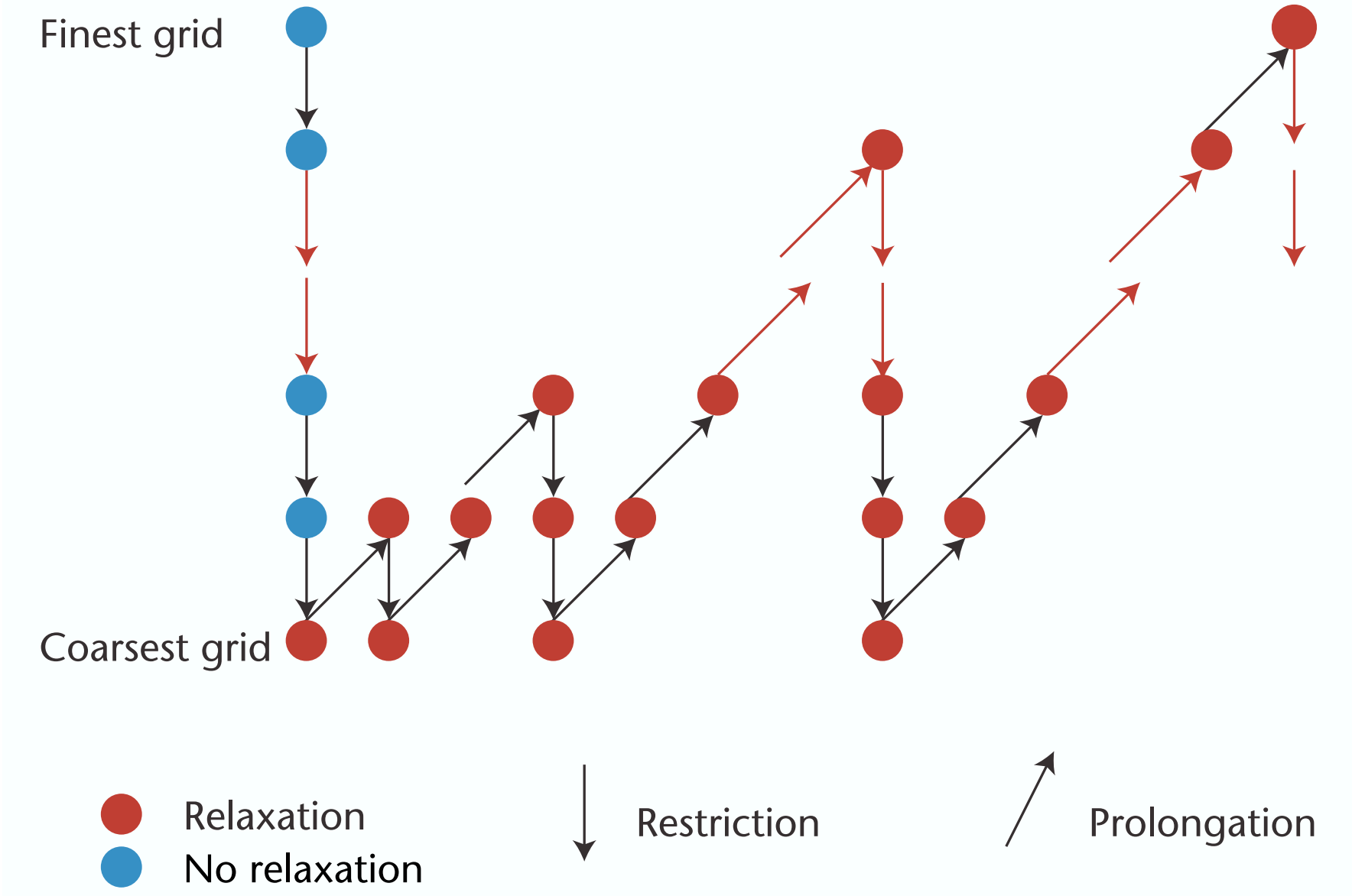
The **convergence rate** of a V-cylce is <1 and bounded away from 1 for a wide class of elliptic PDE independent of the mesh size. In practice, we try (and often succeed) to achieve a convergence rate of $\mu \approx 0.1$.

The Full Multi-Grid (FMG) Algorithm

The multigrid *V*-cycle is an iterative method, and hence it requires an initial guess for the solution. This initial approximation is obtained from a coarser grid, and so on recursively.

The FMG algorithm combines the grid-refinement approach with the *V*-cycle.

For many problems, FMG with just a single *V*-cycle per level suffices to reduce the error below truncation level. In this case, only $O(N)$ operations are required overall.


Finest grid
Coarsest grid
Relaxation
No relaxation
Restriction
Prolongation

Simplified quantitative analysis of FMG

Let u^h_{FMG} denote the solution obtained on grid h by the FMG algorithm. We want the algebraic error in this solution to be at worst comparable to the discretization error:

$$\left\| u^h - u^h_{FMG} \right\| \le \beta \left\| I^h u - u^h \right\|,$$

where u^h is the exact solution to the discrete problem, u is the solution to the differential problem, I^h is a restriction to the grid, and β is a constant. This immediately implies:

$$\left\| I^h u - u^h_{FMG} \right\| \le (1 + \beta) \left\| I^h u - u^h \right\|.$$

We see that there is no sense in working hard to reduce below ~ 1, because the effort is better spent on reducing the discretization error by using a finer grid.

Suppose that a p-order discretization is used, so that for u that is smooth on the scale of the grid,

$$I^h u - u^h \approx I^h c h^p,$$

where c is some (nearly) h-independent function.

Suppose that the FMG solution on grid $2h$ satisfies the criterion we require. When we interpolate this solution to grid h we have:

$$\begin{aligned}
&\left\| u^h - I_{2h}^h u_{FMG}^{2h} \right\| \\
&= \left\| u^h - I_{2h}^h u^{2h} + I_{2h}^h \left(u^{2h} - u_{FMG}^{2h} \right) \right\| \\
&\le \left\| u^h - I_{2h}^h u^{2h} \right\| + \left\| I_{2h}^h \left(u^{2h} - u_{FMG}^{2h} \right) \right\| \\
&\sim \left(2^p - 1 \right) I^h c h^p + \beta I^h c (2h)^p \\
&= \left[(1+\beta) 2^p - 1 \right] I^h c h^p .
\end{aligned}$$

Hence, in order to continue satisfying the β criterion as the grid is further refined, we need to reduce the error by a factor $\beta / \left[(1+\beta) 2^p - 1 \right]$

or better.

Alternatively, if we reduce the error by a factor μ_V per *V* cycle, then

$$\beta = \frac{\mu_V \left(2^p - 1 \right)}{1 - 2^p \mu_V} .$$

For example, if $\mu_V = 0.1$ and $p = 2$, then $\beta = 0.5$

The *cascadic multigrid method* does full multigrid (nested iteration) but then performs only a specific number of CG-iterations on each new level, e.g. one on the finest, 2 on the next coarser, 4 on next coarser and so on.

This has been proposed by Shaidurov, Deuflhard, Borneman, see e.g. F. Bornemann, P. Deuflhard: *The Cascadic Multigrid Method for Elliptic Problems. Numer. Math. 75, Springer International, pp. 135-152 (1996).*

Interestingly, this leads asymptotically in the H^1-norm to an optimal algorithm, but not in the L_2-norm and is therefore unacceptable in many applications.

Nonlinear Problems

There are two common approaches for handling nonlinear problems by multigrid methods. One is the classical approach of employing Newton linearization (and solving the resulting linear problems by the usual multigrid methods).

A second approach is to employ nonlinear multigrid methods, most commonly the "Full Approximation Scheme" (FAS).

Recall that, in the usual multigrid approach, we use the coarse grid to approximate the correction to the fine-grid error. That is, we approximate the fine-grid equation

$$L^h v^h = r^h,$$

by the the coarse-grid equation

$$L^H v^H = I_h^H r^h.$$

We can rewrite the fine-grid equation as

$$L^h u^h - L^h \tilde{u}^h = r^h.$$

We approximate this equation on the coarse grid by

$$L^H u^H - L^H \tilde{u}^H = r^H,$$

with $\tilde{u}^H = I_h^H \tilde{u}^h$.

The difference is that now the variable, $\tilde{u}^H$,

approximates the full solution rather than just the correction. Hence, this approach can be applied to nonlinear problems. After we solve the coarse-grid problem, we interpolate and add the correction:

$$\tilde{u}^h \leftarrow \tilde{u}^h + I_H^h \left(u^H - \tilde{u}^H \right)$$

Two-grid FAS Algorithm

- Relax several times on grid *h*, obtaining $\tilde{u}^h$ with a smooth corresponding error.
- Calculate the residual: $r^h = f^h - L^h \tilde{u}^h$.
- Solve approximate equation for the full solution on the coarse grid: $L^H u^H = f^H \equiv I_h^H r^h + L^H \hat{I}_h^H \tilde{u}^h$.
- Interpolate and add correction: $\tilde{u}^h \leftarrow \tilde{u}^h + I_H^h \left(u^H - \hat{I}_h^H \tilde{u}^h \right)$.
- Relax again on grid h.

Multi-grid is obtained by recursion.

FAS: Dual point of view

We can rewrite the FAS coarse-grid problem in the form

$$L^H u^H = I_h^H f^h + \tau_h^H ,$$

where

$$\tau_h^H = L^H \hat{I}_h^H \tilde{u}^h - I_h^H L^h \tilde{u}^h .$$

Observe that τ_h^H is the fine-to-coarse defect correction – an increment to the right-hand side designed to make its solution coincide with the fine-grid solution.

This reverses our point of view: the fine-grid "visits" can be seen as a means of obtaining a better estimate of the τ-correction.

FAS is used for adaptive refinement multigrid, when fine grids are employed only in parts of the solution domain

FAS-Multigrid is related to the Hierarchical Basis Multigrid Method by Bank and Yserentant.

If FAS is implemented as described above, it is slightly more expensive than correction multigrid.

If FAS is implemented via Hierarchical Basis transformations it can be as efficient as correction-multigrid.

In fact, Griebel has proposed in his 1989 thesis an algorithm for Poisson‘s equation in 2D, that uses

- Full multigrd
- Hierarchical Basis-FAS
- Red-Black-Gauss-Seidel
- Half-injection, cleverly exploiting zeros in the residual
- In a Full Multigrid approach using a single V-cycle on each new level

This algorithm solves the 5-pt-Stencil Poisson problem (with discretization error accuracy) in 27.5 operations per unknown.

Double-Discretization

An alternative method for obtaining higher-order accuracy is the method of double-discretization.

The residuals are computed using a discretization that is different (normally higher-order) from that used in the relaxation process.

The discretization of the residuals generally determines the accuracy of the method, while that used for relaxation determines the stability.

This method is related to the τ-extrapolation technique.

$$L^h = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & & \cdots & & & & \\ & & & 1 & -2 & 1 & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & \end{bmatrix}$$

$$I_H^h = 2\left(I_h^H\right)^T = \frac{1}{2}\begin{bmatrix} 1 & & & & & & \\ 2 & & & & & & \\ 1 & 1 & & & & & \\ & 2 & & & & & \\ & 1 & 1 & & & & \\ & & & \cdots & & & \\ & & & & 1 & 1 & \\ & & & & & 2 & \\ & & & & & 1 & 1 \\ & & & & & & 2 \\ & & & & & & 1 \end{bmatrix}$$

Given the fine-grid matrix, L^h, and the prolongation and restriction matrices, I_H^h and I_h^H, how should we define the coarse-grid matrix, L^H ?

The coarse grid should be able to correct smooth errors. We use the following (algebraic) definition of smoothness: An error v_{before}^h is smooth if it is in the range of interpolation. That is, if there exists some coarse-grid function, w^H, such that

$$v_{before}^h = I_H^h w^H .$$

The error after the coarse-grid correction is given by

$$v^h_{after} = C^h v^h_{before}$$

Where $$C^h = I^h - I^h_H \left(L^H\right)^{-1} I^H_h L^h .$$

is the coarse-grid correction iteration matrix.

Plugging in our smooth error we obtain:

$$v^h_{after} = C^h v^h_{before}$$

$$= \left[I^h - I^h_H \left(L^H \right)^{-1} I^H_h L^h \right] I^h_H w^H$$

$$= \left[I^h_H - I^h_H \left(L^H \right)^{-1} I^H_h L^h I^h_H \right] w^H$$

$$= I^h_H \left[I^H - \left(L^H \right)^{-1} I^H_h L^h I^h_H \right] w^H .$$

In order to annihilate the error we must choose the Petrov-Galerkin coarse-grid operator:

$$L^H = I^H_h L^h I^h_H .$$

For symmetric problems especially, the preferred choice for the restriction is the transpose of the prolongation. Along with the Galerkin coarse-grid operator this yields so-called variational coarsening, which arises naturally in finite-element formulations.

It remains only to define the prolongation (and, implicitly, the set of variables which defines the coarse grid). The prolongation operator should produce good approximate fine-grid values from given coarse-grid values. When used with appropriate coarse grids, such methods yield fast and robust algebraic solvers.

Algebraic Multigrid (AMG)

Introduced by Brandt et al. (1983) and developed by Ruge and Stüben.

In classical AMG, a relaxation method is chosen (usually, point Gauss-Seidel), and then the coarse-grid variables are chosen by a heuristic graph algorithm such that each fine-grid variable depends strongly on one or more coarse-grid variable (i.e., with relatively large coefficient).

AMG enables us to handle unstructured and non-PDE problems.

Abstract View of an AMG Approach

Consider the linear system

$$Au = f$$

Suppose we partition the variables, u_i , into *F* variables and *C* variables, and permute the equations and variables to produce the following partitioning of the system:

$$Au = \begin{pmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{pmatrix} \begin{pmatrix} u_F \\ u_C \end{pmatrix} = \begin{pmatrix} f_F \\ f_C \end{pmatrix}.$$

Abstract View of AMG

Given an approximate solution, $\tilde{u}$, define the error as

$$v = u - \tilde{u}.$$

Then, the partitioned equation for the error is

$$Av = \begin{pmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{pmatrix} \begin{pmatrix} v_F \\ v_C \end{pmatrix} = \begin{pmatrix} r_F \\ r_C \end{pmatrix},$$

where

$$r_F = f_F - A_{FF}\tilde{u}_F - A_{FC}\tilde{u}_C,$$
$$r_C = f_C - A_{CF}\tilde{u}_F - A_{CC}\tilde{u}_C.$$

Abstract View of AMG

The upper block yields

$$A_{FF} v_F = r_F - A_{FC} v_C ,$$
$$\Rightarrow v_F = A_{FF}^{-1} \left(r_F - A_{FC} v_C \right).$$

Plugging this into the lower block yields

$$A_{CF} A_{FF}^{-1} \left(r_F - A_{FC} v_C \right) + A_{CC} v_C = r_C ,$$
$$\Rightarrow \left(A_{CC} - A_{CF} A_{FF}^{-1} A_{FC} \right) v_C = r_C - A_{CF} A_{FF}^{-1} r_F .$$

Abstract View of AMG

Conclusion: the “ideal” prolongation and restriction are

$$P = \begin{pmatrix} -A_{FF}^{-1} A_{FC} \\ I \end{pmatrix}, \quad R = \begin{pmatrix} -A_{CF} A_{FF}^{-1} & I \end{pmatrix},$$

with the coarse-grid operator given by

$$A_C = RAP = A_{CC} - A_{CF} A_{FF}^{-1} A_{FC}.$$

Abstract View of AMG

In particular, it is straightforward to verify that the two-level solution is exact in this case, provided that either a pre-relaxation or a post-relaxation eliminates r_F.

(If this is done by post-relaxation, only u_F should be relaxed.)

The problem: A_{FF}^{-1} is not sparse, and therefore, neither are P and R. Therefore, we generally look for good sparse approximations.

One exception is tri-diagonal systems, where A_{FF} is diagonal. In this case the multigrid V-cycle with the appropriate prolongation and restriction, and with relaxation only on u_F is an exact solver, equivalent to total reduction.