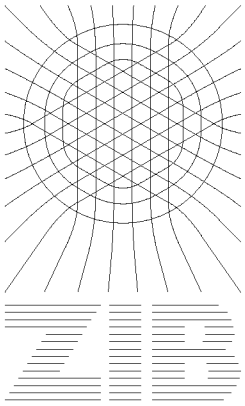
Konrad-Zuse-Zentrum für Informationstechnik Berlin



B. Erdmann R. Roitzsch F. Bornemann

KASKADE

Numerical Experiments

Technical Report TR 91–1 (December 1991)

KASKADE

Numerical Experiments

B. Erdmann R. Roitzsch F. Bornemann

Abstract. The C–implementation of KASKADE, an adaptive solver for linear elliptic differential equations in 2D, is object of a set of numerical experiments to analyze the use of resources (time and memory) with respect to numerical accuracy. We study the dependency of the reliability, robustness, and efficiency of the program from the parameters controlling the algorithm.

Contents

Chapter 1

Introduction

The finite element method KASKADE [13] handles scalar linear elliptic two–dimensional partial differential equations of the form

$$\begin{aligned} -(p_1 u_x)_x - (p_2 u_y)_y + qu &= g \quad \text{in} \quad \Omega \\ u &= \gamma \quad \text{on} \quad \Gamma_0 \subset \partial\Omega \\ \frac{\partial u}{\partial n} + \eta u &= \xi \quad \text{on} \quad \Gamma_1 \ . \end{aligned} \tag{1.1}$$

with $\Gamma_0 \cup \Gamma_1 = \partial\Omega$ and $q(x,y) \geq 0$ and $0 \leq \eta(x,y)$, $0 < C \leq p_1(x,y), p_2(x,y)$. Here Ω denotes a polygonal domain in $\mathbb{R}^2$ and Γ_0 is composed of edges of $\partial\Omega$. Furthermore $\frac{\partial}{\partial n}$ denotes the conormal derivative associated with p_1, p_2.

The first implementation in PASCAL is described in [16]. The second implementation in C [22] is studied in this paper. This version was extended to the convection–diffusion equation [14], time–dependent PDEs [3], and obstacle problems in device simulation [15]. In this context alternative refinement strategies (blue refinement) and the BPX–preconditioner of Bramble,Pasciak, Xu [11] were implemented.

Subject of this paper are results of a set of test problems applied to KASKADE and the choice of some KASKADE parameters. Reliability, robustness, and efficiency are studied. Further the results are compared with the popular adaptive FEM–code PLTMG of R. Bank [6].

1.1 The KASKADE Algorithm

The KASKADE algorithm is an adaptive multi–level method using linear finite elements. A flow chart of its general loop can be seen in Figure 1.1.

On a given triangulation $\mathcal{T}_0$ (level 0) the first finite element solution is computed by a direct solver. KASKADE supports the usage of two different versions of a sparse Cholesky decomposition: A fully sparse elimination with the nested dissection numbering and an envelope elimination with the reverse Cuthill–McKee numbering of the nodes. The usage of a sparse solver for the initial triangulation is of particular interest in applications where the problem geometry demands an initial grid with rather many nodes.

The next steps are an error estimation (ESTIMATE) and a closely coupled refinement process (REFINE). In the first step we get an approximate value ε_{est} of the global error (energy norm). This value is used to stop the adaptive iteration cycle if the user–requested global accuracy ε_{tol} is reached. The

ESTIMATE process generates additionally a local error estimate ε_Δ for each triangle (or edge) of the mesh. These values are used in the REFINE step to select triangles for local, regular refinement if $\varepsilon_\Delta > \sigma \cdot \Theta$, where Θ is some sort of average error and σ a user–supplied constant. In [13] Θ is computed as the mean value of all local errors (short form Θ_{mean}). Bornemann [4, Section 2.4.3] contributed to KASKADE the computation of Θ by extrapolation of the local errors of the different refinement levels (short form Θ_{extrap}). Essentially this strategy goes back to Babuška/Rheinboldt [1].

After each REFINE step it is checked if the maximum of nodes $N_{\max}$ is reached or if enough new nodes are found. In this case the iterative solver (ITERATE) is called. Otherwise the ESTIMATE–REFINE cycle is repeated.

The iterative solver is a preconditioned conjugate gradient method which solves the linear system resulting from the finite element approach on the mesh $\mathcal{T}_i$

$$A_i u_i = b_i \; . \tag{1.2}$$

The iteration process should stop if the solution of the system is as accurate as the discretization error which is *predicted* by $\varepsilon_{\text{est}}^{\text{new}}$. For safety reasons we introduce a factor ρ and stop the iteration process if the error of the linear system is less than $\varepsilon_{\text{req}} := \varepsilon_{\text{est}}^{\text{new}} * \rho$.

The CG–method needs only a device for matrix–vector multiplications Ax and thus no explicit representation of the matrix A. There is an option either to maintain A completely in a sparse form or to store the local stiffness matrices for each triangle. Hierarchical Basis [13] and BPX in the implementation of Bornemann [4] are the available preconditioners.

Table 1.1 shows the combination of parameters that are tested systematically and the abbreviations to denote these tests.

	Θ	s	σ	ρ	matrix
K1	mean	2.0	0.95	0.25	local
K2	mean	2.0	0.95	0.01	local
K3	extrap	1.1	1.0	0.25	local
K4	extrap	1.1	1.0	0.01	local
K5	extrap	1.1	1.0	0.01	sparse
P	Bank's PLTMG				

Table 1.1:

1.2 The PLTMG Algorithm

We use the well–known Finite Element program PLTMG [6] as a reference system (P). PLTMG resembles KASKADE, it uses the same adaptive refinement techniques but has different ESTIMATE [10], REFINE, and ITERATE steps. The ESTIMATE/REFINE loop is optimized to reach the user–supplied $N_{\max}$, normally it needs an expansion factor $s = 4$ at each level. We note that PLTMG addresses non–linear problems.

1.3 Norms and Notation

Let u be the exact solution of the problem (1.1), $\hat{u}_L$ the iterative solution of the linear FEM equations, and $||\cdot||$ the energy norm. Then we define

$$\varepsilon_L := ||u - \hat{u}_L|| \tag{1.3}$$

$$\varepsilon_{\text{est}} \quad \text{the estimated } \varepsilon_L \tag{1.4}$$

Furthermore $\varepsilon_{\text{grid}}$ is defined as the maximal error at the nodes, the centers of the triangles and the points halfway between the center and the nodes, thus approximating the real maximum norm.
We introduce some additional abbreviations:

- i_l number of iterations on refinement level l
- T total CPU–time in seconds for solving the problem
- $\%_{\text{est}}$ percentage of total time for error estimation
- $\%_{\text{ref}}$ percentage of total time for refinement
- $\%_{\text{ite}}$ percentage of total time for assembling and iterative solution
- t_l accumulated CPU–time in seconds for all iterations, up to level l
- t_p CPU–time in seconds per iteration and point

1.4 Test Set

We use the test set recommended by W. Mitchell [18] (see Appendix A). It contains problems of different complexity including peaks, boundary layers, discontinuities, singularities, and non–quadratic domains (see Table 1.2).

name	type	remark
1	Poisson	sharp peaks
2	Helmholtz	boundary layer
3	Helmholtz	mild peaks
4	Poisson	wavefront
5	Laplace	moderate peak, geometry
6	Laplace	singularity
7	Laplace	discontinuous boundary condition
8	$\nabla a \nabla u = 0$	discontinuous a

Table 1.2: Test set

1.5 Breaking Conditions

One problem area is to define a break condition for the adaptive process. The mathematical sound method is to stop if the estimated error ε_{est} is below a user supplied ε_{tol}. We compare the estimation in our experiments with the true error ε_L in order to test the reliability of the algorithm (Chapter 2). This is also important in the context of the evaluation of efficiency (see Chapter 3).

In PLTMG the error is computed in the H^1–norm, in KASKADE in the energy norm. Thus the handling of the break condition makes the comparison of the two solvers difficult. Hence in tables where both the methods are mentioned, we didn't consider the Tests 2 and 3 and changed the other ones a little by adding the solution u on both sides of the equations. On the right side of (1.1) we replaced u by the known expression of the solution getting test equations for which the H^1–norm agrees with the energy norm.

In some real life problems it might be useful to take the number of nodes $N_{\max}$ as break condition due to limits of memory.

1.6 Computing Environment

All computations are done on a Sun Sparc 1+ station under OS, Release 4.1 using the `f77` and `cc` compiler with their highest optimization level. This configuration should yield about 1.4 MFlops for the Fortran double precision LINPACK benchmark.

32–bit real data types (real respectively float) are used. For better accuracy double precision computation (64–bit arithmetic) is recommended.

1.7 Delicate Testing Questions

The comparison of different algorithms or implementations is extremely delicate, e.g., for the following reasons:

- Small changes in parameters of the algorithms might totally change the adaptive development, e.g., produce a different sequence of refined triangles.
- The solution process might depend very sensitive on the initial triangulation. An improper choice of the initial triangulation could easily spoil the solution.
- An improper choice of the break condition could dominate the overall time because each new step of the adaptive process is much more expensive than the ones before.
- The algorithms may be optimized for a certain problem class, thus a comparison of a code for linear problems with a code for non–linear problems might be inevitably biased.
- Memory can be allocated for purposes other than the here tested, e.g., graphical processing.
- Runtime results can be dominated by hardware structure and compiler quality, e.g., a wrong model of the optimization process of the compiler can lead to bad implementations.
- Particular attention should be paid the fact that we compare implementations in different programming languages, i.e, FORTRAN and C.

These problems in mind we want to compare the behavior of KASKADE and PLTMG as carefully as possible and to prove that the KASKADE algorithm runs with reasonable efficiency, reliability, and robustness.

N	Number of nodes
$N_{\max}$	Maximal number of nodes (user supplied)
ε_{tol}	Requested global error (user supplied)
ε_{est}	Estimated global error
$\varepsilon_{\text{est}}^{\text{new}}$	Predicted global error on the new mesh
ε_{req}	Required accuracy for the PCG iteration
s	Factor for the number of new points (parameter)
ρ	Iteration safety factor (parameter)

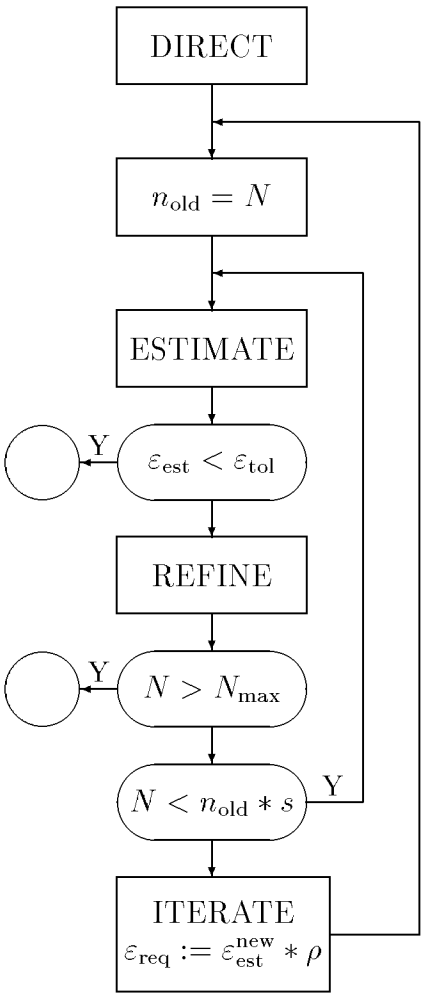


Figure 1.1: Main iteration loop of KASKADE

Chapter 2

Accuracy

In this chapter we investigate the convergence behavior and the error estimation device of the adaptive KASKADE algorithm which are essential for reliability, robustness, and efficiency.

We stress the importance of the interplay between the error estimator and the refinement strategy. The error estimator should produce local and global estimates for a given triangulation. Local estimations provide information for the refinement strategy, deciding which and how many triangles to refine. A method achieves high accuracy with a minimal number of nodes only if the error estimator represents the local error qualitatively well and the refinement strategy defines efficient rules for selecting triangles. A quantitative good estimate of the global error is needed in order to obtain a reliable break condition. An underestimation might lead to stop the solution without getting the requested accuracy and an overestimation may lead to more refinement levels (and much heavier use of computer resources).

2.1 Convergence Behavior

The adaptive refinement should reduce the size of the linear system and retain a small discretization error. Thus it seems reasonable (in view of optimal approximation properties with linear elements) to evaluate the ratio of $\log(N)$ and $|\log(\varepsilon_L)|$ with respect to the different refinement strategies relying on the mean value or on extrapolation. However, the number of nodes used to reach the requested accuracy gives only a hint of the efficiency of the method. One reason is the time spent on assembling stiffness matrices, in particular for the error estimator. Thus a method selecting only few new points at each level might reach the accuracy with less nodes but spend too much time by estimations. We'll look into this question in the next chapter.

We start with the evaluation of the convergence history for our test problems. Some typical results are noted in Appendix C. As the break condition

$$\varepsilon_{\text{est}}/||\hat{u}_L|| < \varepsilon_{\text{tol}} = 0.05 \tag{2.1}$$

was selected. Each of the solvers achieved this accuracy.

A quantitative comparison of the accuracy is practically impossible because different error estimators and refinement strategies produce different meshes. However, the algorithms show qualitatively an equivalent behavior. On

meshes with roughly the same number of nodes most tests reach the same accuracy.

A significant discrepancy of the accuracy was seldom observed. In Example 6c the refinement strategy (K2) is inferior to the other methods. Strikingly PLTMG achieved in all examples smaller values for $\varepsilon_{\text{grid}}$ on the mesh nodes, due to a more accurate solution of the linear system.

We will not analyze these results further in a fashion which might pretend a non–existing continuity. The success of a method just depends strongly on the discrete selection of triangles to refine, which is best demonstrated by the different results of the mean value refinement strategy and and the extrapolation strategy, which use nevertheless the same error estimator, see Tables and Figures in the Appendix C.

The results show only in the Examples 6a, 6b and 6c a significant drawback of the mean value strategy. All tests show comparable fast convergence for the extrapolation strategy with parameter choice (K4) and PLTMG.

2.2 The Global Estimated Error

A method should give a good estimation of the global error for a reliable stopping criterion. The error estimator can be characterized by the effectivity index [1] defined by

$$\mathrm{f}(\varepsilon_{\text{est}}) = \varepsilon_{\text{est}}/\varepsilon_L \ . \tag{2.2}$$

We follow [10] and choose as a measure of the relative error

$$\zeta(\varepsilon_{\text{est}}) = \mathrm{f}(\varepsilon_{\text{est}}) - 1 \ . \tag{2.3}$$

Having ζ near or converging to zero is clearly the most desirable situation. Positive values of ζ indicate an overestimate of the true error and are acceptable as long as ζ is not too much larger than one. Negative values of ζ mean, the error indicator has given an optimistic estimate of the error. Here the value of ζ should not be below $-1/2$.

We measured the effectivity index ζ for the methods (K2), (K4), and (P). Some results are collected in Appendix C.

Only for Problem 6c the values for ζ show significant differences between the methods. In all other cases the global error is estimated comparatively well. We observe a certain tendency of the KASKADE error estimator to underestimate the error and therefore to stop the solution process in some cases too early. Example 6c exemplifies this: the mean value refinement strategy leads to meshes where the KASKADE error estimator is way off the real error. However, the extrapolation refinement strategy generates meshes with far better results. We use the results from [10], to confirm their evaluation for example 6c, see Tables 2.1–2.3.

N	ε_{est}	ε_L	ζ	$\log(N)/\lvert\log(\varepsilon_L)\rvert$
10	$2.53_{10}{-}01$	$4.86_{10}{-}01$	-0.479	3.19
21	$2.45_{10}{-}01$	$4.38_{10}{-}01$	-0.441	3.69
44	$2.31_{10}{-}01$	$3.93_{10}{-}01$	-0.412	4.05
202	$1.84_{10}{-}01$	$3.03_{10}{-}01$	-0.393	4.45
588	$1.40_{10}{-}01$	$2.24_{10}{-}01$	-0.375	4.26
745	$1.24_{10}{-}01$	$1.93_{10}{-}01$	-0.358	4.02

Table 2.1: Error analysis (K2) for Problem 6c, natural boundary values

N	ε_{est}	ε_L	ζ	$\log(N)/\lvert\log(\varepsilon_L)\rvert$
10	$2.53_{10}{-}01$	$4.86_{10}{-}01$	-0.479	3.19
27	$2.40_{10}{-}01$	$4.50_{10}{-}01$	-0.467	4.13
47	$2.27_{10}{-}01$	$3.67_{10}{-}01$	-0.381	3.84
151	$1.78_{10}{-}01$	$2.27_{10}{-}01$	-0.216	3.38
207	$1.63_{10}{-}01$	$1.89_{10}{-}01$	-0.138	3.20
447	$1.18_{10}{-}01$	$1.28_{10}{-}01$	-0.078	2.97
912	$8.68_{10}{-}02$	$9.18_{10}{-}02$	-0.054	2.85

Table 2.2: Error analysis (K4) for Problem 6c, natural boundary values

N	ε_{est}	ε_L	ζ	$\log(N)/\lvert\log(\varepsilon_L)\rvert$
10	$3.04_{10}{-}01$	$4.99_{10}{-}01$	-0.390	3.31
41	$2.96_{10}{-}01$	$3.68_{10}{-}01$	-0.197	3.71
161	$2.21_{10}{-}01$	$2.08_{10}{-}01$	0.065	3.23
681	$1.23_{10}{-}01$	$1.20_{10}{-}01$	0.026	3.08

Table 2.3: Error analysis (PLTMG) for Problem 6c, natural boundary values, results from [10]

Chapter 3

Time Efficiency

In this chapter we study the CPU–time necessary to solve the problems of Appendix A.

We consider the number of grid points and the CPU–time necessary to reach the required accuracy $\varepsilon_{\text{tol}} = 0.1$ (cf. Chapter 2) using the error estimators. In those cases where the required accuracy is reached but underestimated, we continue computing until the true error ε_L fulfills

$$\varepsilon_L / \|\hat{u}_L\| < \varepsilon_{\text{tol}}.$$

We observed such an unreliable behavior only in the Examples 4 and 6c, where the KASKADE estimator underestimates the error. PLTMG always yields an overestimation.

In some examples the required accuracy is reached with quite a different number of grid points which is caused by the arbitrarily choice of the determination threshold ε_{tol}. In some cases the error on one level is estimated a little over this threshold, and on the next it is already lying considerably under it. In such unfortunate accidents we take also the preceding level into account.

Using PLTMG we set the maximal size of grid points much higher than necessary for reaching the required accuracy. Thus the number of grid points increases by the factor 4 from one level to the next.

Solver	N	ε_{est}	ε_L	$\|\hat{u}_L\|$	T	$\%_{\text{est}}$	$\%_{\text{ref}}$	$\%_{\text{ite}}$
K1	889	4.63e-03	5.68e-03	.051	5.3	59	7	34
K1	3015	2.64e-03	3.70e-03	.052	17.2	57	6	37
K2	857	3.95e-03	4.11e-03	.052	5.3	56	5	39
K3	1105	4.27e-03	5.85e-03	.051	7.6	60	6	34
K3	2498	3.04e-03	4.91e-03	.051	28.9	71	5	24
K4	899	3.91e-03	4.18e-03	.052	6.6	55	5	40
K5	899	3.91e-03	4.18e-03	.052	8.7	40	4	56
P	1051	4.52e-03	4.11e-03	.052	23.5	55	14	29

Table 3.1: Problem 1a

Solver	N	ε_{est}	ε_L	$\|\hat{u}_L\|$	T	$\%_{\text{est}}$	$\%_{\text{ref}}$	$\%_{\text{ite}}$
K1	2271	1.08e-01	1.73e-01	1.43	18.8	53	5	42
K1	5614	7.52e-02	1.30e-01	1.43	51.9	59	5	36
K2	1672	1.04e-01	1.69e-01	1.44	16.9	49	4	47
K2	4166	7.32e-02	1.19e-01	1.43	47.9	49	4	47
K3	489	1.27e-01	2.61e-01	1.43	7.0	70	4	26
K3	1767	8.54e-01	1.13e-01	1.43	45.1	76	2	22
K4	467	1.13e-01	1.35e-01	1.43	14.4	77	3	20
K5	475	1.13e-01	1.35e-01	1.43	17.7	57	2	41
P	951	1.40e-01	1.30e-01	1.43	29.8	50	21	27

Table 3.2: Problem 6c

problem	Solver	N	ε_L	T
1a	P	1051	$4.11_{10}-03$	21.2
	K	899	$4.18_{10}-03$	9.0
1b	P	1055	$1.35_{10}-01$	22.4
	KK	817	$1.55_{10}-01$	9.8
4	P	1591	$2.10_{10}-01$	32.3
	K	1075	$2.32_{10}-01$	10.1
5	P	361	$2.12_{10}-01$	5.3
	K	437	$1.68_{10}-01$	2.2
6c	P	951	$1.30_{10}-01$	29.8
	K	467	$1.35_{10}-01$	17.7
7	P	1561	$2.06_{10}-01$	31.8
	K	901	$2.70_{10}-01$	6.6
8	P	523	$<1.16_{10}-01$	8.9
	K	535	$1.16_{10}-01$	2.6

Table 3.3: KASKADE (K=K5) versus PLTMG (P)

3.1 KASKADE Internals

We start with the analysis of KASKADE to find a good parameter set. This version will be compared with the PLTMG–solver in the next section.

In KASKADE we may choose between two refinement strategies. We studied the resulting accuracy in Chapter 2. Using the same notation we analyze the influence on the time efficiency of the mean value strategy (Solver K1, K2) and the extrapolation strategy (Solver K3, K4).

In the tests we also varied the safety factor ρ (0.25 and 0.01) in order to study the influence of accuracy in the iteration process on the time requirement. Here we always used preconditioning by hierarchical bases. A comparison with the BPX–preconditioner follows in Chapter 5.

The selection of $\rho = 0.01$ (versions K2 and K4) in the iteration process seems to be better than the $\rho = 0.25$ in the versions K1 and K3. Though it needs more (up to 4) iteration steps, the approximate solution is found in most examples on a coarser grid (e.g. higher accuracy) in a shorter time. In the other problems there is only little additional work, because one iteration step takes only a small amount of time compared with other parts of the solver. Therefore we propose the smaller safety factor 0.01. In Chapter 5 we refer to some more results we made in the course of our experiments with ρ.

A comparison of versions K2 and K4 shows only in the Tests 1b, 6c and 7 significant differences. The extrapolation strategy, discussed in Chapter 2, has the advantage of finding the solution in a shorter time. Though the extrapolation strategy needs more refinement steps and error estimations, in most examples it generates a sufficient accurate solution on a coarser grid than the mean value strategy.iIn this context it is crucial that the extrapolation strategy does not require the doubling of grid points from level to level as the mean value strategy. The effect of the higher accuracy of the extrapolation strategy is very clear in Example 6c. In Example 1b we have to notice an exception of this rule. In this problem the mean value strategy is more accurate and yields an advantage in runtime. Both versions, K2 and K4, need the same number of refinement steps, but K2 generates less grid points.

Thus we realize that it is reasonable to include both strategies in the program. The user can select the most efficient one for his type of problems.

3.2 KASKADE versus PLTMG

We recall some results from Chapter 2:

1. Every solver reaches the required accuracy. There are no failures. In most of the testskKASKADE yields the required accuracy with fewer points than PLTMG. The reason for this behavior is not a higher accuracy of KASKADE, but the unfortunately tuned refinement strategy of PLTMG requiring an increase of points by a factor 4 for each refinement step.

2. The solvers show different accuracy only in the Examples 4 and 6c. The approximate solution of KASKADE in Problem 6c is more accurate than that of PLTMG.

In this section we study the influence of the different methods on the runtime of the solver. Solutions with coarser grids are often generated by more refinement steps, each including the solution of a linear system, an error estimation or some interpolation work. Obviously we get the following result:

> Differences in the runtime between the versions of KASKADE and PLTMG are significant in all tests. *For each problem there is a variant of KASKADE which is much faster (up to factor 5) than the solver PLTMG.*

To simplify the comparison of KASKADE and PLTMG, we only consider the version K5 of KASKADE. It uses the same refinement strategy as K4, the more robust one. K5 needs nearly as much memory as K4 due to the storing of the local stiffness matrices at each triangle. *Therefore we prefer the faster version K4.* In order to extract other effectswwe decided to use K5 here because it handles the assembling of the stiffness matrix similar to PLTMG, e.g. it recomputes all elements on a grid without using values of former grids.

The clear advantage of KASKADE is caused by the special C–implementation and the smaller problem class. We analyze some details:

1. The evaluation of the problem describing functions is in PLTMG much more expensive (up to factor 3–4) than in KASKADE. There is only one function in KASKADE which computes all values at a point, while in PLTMG different values are evaluated in different functions. In addition PLTMG uses another formulation of the problem (PLTMG is a solver for nonlinear problems too), which needs each of these functions for five different parameters. Furthermore these values are used in the integration formulas yielding superfluous arithmetic operations in the case of a linear problem.

   Such numerical integrations and function evaluations are necessary for the assembling (stiffness matrix and the right–hand side) and in the error estimation. We point out that PLTMG needs nearly the same number of evaluations of the right–hand side, but twice the number of evaluations of the coefficients per point. *These additional function calls are necessary for the error estimation process in PLTMG.* (The evaluations of a function in PLTMG with five different parameters are counted as one function call.)

   In our examples the evaluation time for these functions yields about 10% of total time. Note that we have constant coefficient problems and right–hand sides with few floating–point operations, conditional control statements and standard function calls. *In real life problems*

expensive function evaluations can dominate the total time, thus ruling the efficiency of the solver. If only the right–hand sides are expensive, both solvers will have similar runtime, but if a lot of time is spend by computing the coefficient functions, we expect KASKADE to be faster.

We found no further hints on significant numerical superiority of one of the solvers. The runtime advantage is homogeneous in all parts (error estimation, integration, linear solution) of the programs. A more detailed analysis of this question is intricate because of the very different implementations.

2. The way of handling the data structures (used for describing the dependencies of different values on the geometry) seems to have an immense influence on the costs of the analyzed solvers. Comparisons showed that the structural data types and the pointer structures in C allow a very natural programming of an adaptive algorithm. While the Fortran–coded PLTMG needs a lot of index computations to get the relation between certain values and the geometry, the C–implementation of KASKADE uses a faster access by structured types and pointers. We did not analyze these effects quantitatively. We just depict some details with considerable influence on the speed:

   - PLTMG spends much more time than KASKADE handling the grids and the refinement (both solvers use the same geometrical refinement rules) after an error estimation, even in the case of uniform refinement when both algorithms generate the same grids.
   - Though both solvers use the same integration formulas, the integration process (stiffness matrix, right–hand side) is much faster in KASKADE than in PLTMG, even in the case of uniform grids. (We took into account the more expensive evaluation of the problem describing functions in PLTMG, see above.)

   The speedup of using structured types and pointers in the C–version of KASKADE seems to correspond with higher requirements in memory, see Chapter 4.

3. The runtime of a code depends immensely on the computer architecture and the related optimization of the compilers. For example, on our computer the advantage of KASKADE will decrease when we use no optimization. We suppose that the difference in the runtime between PLTMG and KASKADE might disappear on special machines, e.g. computers with vector units (the array formulation in the Fortran–coded PLTMG might be better suited for vectorization than the structured types in the KASKADE–code).

4. In our tests we considered public domain programs, whose purpose is to solve a wide class of problems (PLTMG even nonlinear problems) in a comfortable way. Specially PLTMG is not optimized for solving our test problems. We already mentioned some details, maybe there are more.

Final remarks:

- Saving of the local stiffness matrices in KASKADE (version K4) or using the exact integration in the case of problems with constant coefficients will accelerate the version K5. Corresponding options of PLTMG are unknown.
- The variation of the safety factor ρ in the iteration process of KASKADE illustrates that even inside a solver a reasonable selection of parameters may cause considerable differences in the runtime (in the examples with the versions K3 and K4 up to 20%).
- In our test problems none of the solvers shows a significant numerical drawback neither in the error estimation nor in the solution of the linear systems. Specially both linear solvers (in PLTMG a Multigrid–Method, in KASKADE a preconditioned CG–method) need only about 10% (included in $\%_{\text{ite}}$) of the total time.

Chapter 4

Storage Efficiency

4.1 KASKADE Internals

KASKADE stores the triangulation information in data structures for points, edges, and triangles, each needing Λ_{point}, Λ_{edge}, $\Lambda_{\text{triangle}}$ bytes respectively. The data to hold the stiffness matrix and vectors is stored in associated arrays of lengths λ_{point}, λ_{edge}, $\lambda_{\text{triangle}}$ at the corresponding data structures. Table 4.1 gives a list of these values when storing the local stiffness matrices at triangles.

real	Λ_{point}	Λ_{edge}	$\Lambda_{\text{triangle}}$	λ_{point}	λ_{edge}	$\lambda_{\text{triangle}}$
float	36	52	60	36	16	36
double	44	52	60	68	32	72

Table 4.1: Local storage requirements (bytes)

To get some estimate Λ of the amount of storage needed to compute a solution at one point we use Euler's formula

$$n_{\text{point}} - n_{\text{edge}} + n_{\text{triangle}} = 1$$

where n_{point}, n_{edge}, n_{triangle} are the number of points, edges, and triangles. For larger triangulations the relations

$$n_{\text{triangle}} \approx 2n_{\text{point}} \ , \quad n_{\text{edge}} \approx 3n_{\text{point}}$$

hold approximately. Taking into account that the hierarchy of triangles is stored too, we get

$$n_{\text{triangle}} \approx 3n_{\text{point}} \ , \quad n_{\text{edge}} \approx 4n_{\text{point}} \ .$$

Thus

$$\Lambda = (\Lambda_{\text{point}} + \lambda_{\text{point}}) + 4(\Lambda_{\text{edge}} + \lambda_{\text{edge}}) + (3\Lambda_{\text{triangle}} + 2\lambda_{\text{triangle}})$$

with the values 596 for single and 772 for double precisions.

Most of the memory requests are handled dynamically: the program allocates only as much memory as used (e.g. for new points, edges, and triangles).

Such an administration of memory corresponds well to an adaptive method, in which the structure and size of mesh is not known a priori.

In order to make the allocation of memory efficient, memory is fetched in buckets big enough to hold 1.000 points. This means that on coarser grids the requirements per point are much higher than the asymptotic value. This effect and the requirements for static variables are negligible with increasing number of points.

Some values of the really allocated memory Λ_{alloc} in one example illustrate this behavior, see Table 4.2.

n_{point}	n_{edge}	n_{triangle}	allocated [Bytes]	Λ_{alloc}
209	581	373	530182	2537
642	1846	1205	770182	1200
1639	4796	3158	1606182	980
4086	12049	7964	3322182	813
9203	27325	18123	7334182	797
20787	61859	41073	16051782	772

Table 4.2: Allocated memory, using 64–Bit–floating point numbers

The data structures used in KASKADE are not free of redundancy. This allows more flexibility in the description of the problem (complex geometry) and ease of implementation for advanced methods (iterative solvers) and additional features (graphics). The redundancy sometimes also yields a faster code. An optimization for an actual application will be possible, if only a subset of KASKADE features is used. Then the user can remove not needed redundancies or can implement short cuts to get a faster and smaller code.

4.2 PLTMG

PLTMG administrates the memory requirements statically. Before compiling the program we have to fix the maximal length LENWS of an array used for the typical informations (points, edges, triangles). Corresponding to the expected grid refinement, we can compute LENWS by the formula [6]

$$\begin{aligned}\text{LENWS} &= 2(NV+NC)+4(NT+NB+12)+(50+KP+KS)*MAXV+640\\ &\cong 50*MAXV+640,\end{aligned}$$

MAXV is a number of points in the finest grid, and we have NV points, NT triangles, NB boundary edges, and NC curved boundary edges in the first coarse triangulation. KS and KP are zero in linear problems.

We see that PLTMG only requires about 50 words (200, 400 bytes in case of 32–Bit, respectively 64–Bit floating point numbers) per grid point.

There are some other arrays, but their length is negligible with an increasing number of grid points.

Chapter 5

More Numerical Experiments

In this chapter we will analyze the influence of certain special options of the KASKADE program. Some in Chapter 3 already mentioned results will be confirmed.

5.1 Preconditioners: Hierarchical Bases versus BPX

As already mentioned the arising linear systems are solved using the preconditioned CG–method.

Two preconditioners are implemented in KASKADE: First the hierarchical bases method (HB), which was theoretically investigated by Yserentant [23]. The second preconditioner (BPX) was suggested by Bramble, Pasciak, and Xu [11]. This method was further investigated for nonuniform triangulations by Yserentant [24], Bornemann [5], and Dahmen/Kunoth [12]. The implementation of BPX for the case of highly nonuniform triangulations of KASKADE was developed by Bornemann [4].

We observe their qualities in our test problems (Appendix A). The iteration is continued until the error is under a fixed threshold defined as product of the estimated global error and two further parameters. One is the quotient of actual and previous number of grid points after the last refinement, the other one is the safety factor ρ, well–known from Chapter 3. We choose $\rho = 0.01$ in Table 5.1 and $\rho = 10^{-6}$ in Table 5.2 and Table 5.3. The grids were generated by the mean value strategy.

The computation is stopped, when the energy norm of the estimated error (relating to the norm of the approximate solution) is smaller than a prescribed error tolerance ε_{tol},

$$\varepsilon_{est}/\|u_L\| < \varepsilon_{\text{tol}} \, .$$

We note the results on the final level.

Table 5.1 shows slightly less iteration steps for BPX, an effect which is even more observable for more accurate solutions of the linear systems (cf. Table 5.2 and 5.3). This corresponds well with the theory.

However we observe a clear advantage in runtime for HB, which needs less time in all our test problems without loosing accuracy in the energy norm. Obviously the higher runtime for BPX is caused by more expensive iteration steps. This is shown in details for Problem 1b, see Tables 5.2 and 5.3.

probl	prec	N	depth	i_l	t_l	T
1a	HB	2954	7	4	2.1	16.0
	BPX	2962	7	4	7.1	20.9
1b	HB	2311	7	4	1.6	13.1
	BPX	2221	7	3	4.5	15.9
2	HB	2402	7	5	2.2	11.7
	BPX	2432	7	4	6.5	16.0
4	HB	3134	7	4	2.8	15.7
	BPX	2616	7	4	6.8	17.6
5	HB	4425	6	4	3.1	18.3
	BPX	3713	6	4	7.6	20.3
6a	HB	1876	6	8	2.5	9.6
	BPX	1831	6	6	5.0	12.1
6b	HB	20305	18	7	39.3	139.0
	BPX	19720	18	6	96.9	192.3
7	HB	3984	9	5	4.2	19.6
	BPX	3568	9	5	11.3	25.3
8	HB	1597	5	2	0.4	5.9
	BPX	1627	5	2	1.7	7.2

Table 5.1: Preconditioners HB and BPX

N	depth	i_l	t_p	t_l
187	4	12	1.3e-04	0.3
787	6	12	1.3e-04	1.6
2221	7	12	1.5e-04	5.5
7543	8	12	1.5e-04	18.9

Table 5.2: HB for Problem 1b

We measured 2.75 times as much time per iteration and point as for HB. This corresponds well with the complexity analysis of the implementation [4, Section 6.5], which predicts a value between two and three.

Though BPX is not the optimal preconditioner in our context, it should be mentioned that the important advantage of BPX is its generality, e.g. it can in contrast to the HB method be extended to 3D–[11] and time dependent problems [3].

N	depth	i_l	t_p	t_l
187	4	8	4.0e-04	0.6
787	6	8	4.0e-04	3.1
2221	7	7	4.1e-04	9.4
7543	8	8	4.2e-04	34.9

Table 5.3: BPX for Problem 1b

5.2 Influence of the Iteration Error

In this section we study once more the aspect of accuracy in the iteration process and present some results in addition to those in Chapter 3.

The multi–level strategy of KASKADE relates the estimated discretization error to the error of the solution of the linear system. It is senseless and not efficient to generate a solution in the iteration process which is more accurate than the discretization. Because we do not know the exact errors of the discretization and the iteration, it is necessary to relate both by a safety factor ρ. It is well chosen if further iterations in the linear solver (preconditioned CG–method) have no or only little effect on the accuracy of the solution on the actual grid. In the first version [13] of KASKADE the authors worked with $\rho = 0.25$. However, the results from Chapter 3 and from this section suggest that a value of $\rho = 0.01$ is safer. Specially we recognize an improved efficiency index of the error estimation.

The effect on the runtime by some additional iteration steps is small compared with the total time including error estimation and integration. Specially in Chapter 3 we saw that the setting of a smaller ρ on the lower levels often improves the convergence history by reaching the required accuracy on a coarser grid.

In some examples (Tables 5.4 – 5.7) we noted the accuracy to computation with $\rho = 0.25$, afterwards we continued the iteration process by choosing smaller values for ρ and noted the improved accuracy. We counted the accumulated number of iterations i_l. The accuracy is measured in the energy norm (ε_L) and a kind of maximum norm ($\varepsilon_{\text{grid}}$, see Chapter 1).

A smaller value of ρ often decreases the maximum error.

ρ	i_l	ε_{est}	ε_L	$\varepsilon_{\text{grid}}$
0.250	2	8.65e-03	9.92e-03	1.81e-03
0.100	4	7.65e-03	8.04e-03	1.07e-03
0.010	7	7.36e-03	7.67e-03	8.31e-04
0.001	10	7.34e-03	7.62e-03	8.15e-04

Table 5.4: Problem 1a, (mean) on level=5, N = 301

ρ	i_l	ε_{est}	ε_L	$\varepsilon_{\text{grid}}$
0.250	5	1.72e-01	1.87e-01	6.67e-02
0.100	7	1.46e-01	1.55e-01	6.30e-02
0.010	13	1.29e-01	1.38e-01	9.38e-03
0.001	15	1.28e-01	1.36e-01	9.33e-03

Table 5.5: Problem 2, (mean) on level=5, N = 479

ρ	i_l	ε_{est}	ε_L	$\varepsilon_{\text{grid}}$
0.250	1	1.71e-01	3.24e-01	2.43e-01
0.100	3	1.83e-01	3.00e-01	2.26e-01
0.010	6	1.87e-01	2.95e-01	2.20e-01
0.001	8	1.86e-01	2.95e-01	2.19e-01

Table 5.6: Problem 6c, (mean) on level=5, N = 307

5.3 Explicit Sparse Form versus Local Stiffness Matrices

By default the KASKADE program does not compute the values of the stiffness matrix explicitly, only the local stiffness matrix of each triangle is saved at the corresponding triangle data structure. If the stiffness matrix is used in the iteration process, these local matrices must be added up for every matrix–vector multiplication. By special selection of a parameter in the program there is the possibility to assemble and save the stiffness matrix (in sparse form) on each level. Thus we get rid of summing of local matrices in every matrix multiplication, which accelerates each iteration step. In addition the local stiffness matrices are not saved in order to reduce the memory requirements. Therefore they must be recomputed on each level,

ρ	i_l	ε_{est}	ε_L	$\varepsilon_{\text{grid}}$
0.250	3	7.57e-01	7.65e-01	8.01e-02
0.100	4	6.94e-01	7.09e-01	7.87e-02
0.010	6	6.60e-01	6.75e-01	7.87e-02
0.001	8	6.58e-01	6.72e-01	7.87e-02

Table 5.7: Problem 7, (mean) on level=6, N = 237

even for triangles which have not changed on the latest refinement level.

We studied the runtime behavior of KASKADE version (K5) which uses the explicit form of the stiffness matrix. The results are shown in Chapter 3. We realize that the time of the iteration process (multiplication with the stiffness matrix) is shortened but the loss in the integration process (integration is repeated on each level even on triangles which did not change) dominates the runtime. We need about 30% more time than in the version K4 working with local stiffness matrices. K5 should be used in cases where the number of iteration steps is much higher than in our test problems.

The complete–matrix version K5 requires less memory than the saving of all stiffness matrices in K4 (see Chapter 4), but comparedwwith the total amount of memory this advantage seems to be negligible.

Chapter 6

Summary

6.1 Analysis of KASKADE

- On the test set KASKADE has proved to be a reliable, robust and efficient algorithm.
- The refinement strategy based on local extrapolation turns out to be more robust and accurate than the refinement strategy based on the mean value. Further it generates triangulations with far fewer nodes and is superior in runtime.
- The edge oriented error estimator turns out to be efficient and accurate. In tendency it underestimates the error slightly. On the solution triangulation it agrees with the true error up to a difference of only 5%–7% (extrapolation strategy used!).
- The hierarchical bases preconditioner is as robust and accurate as the BPX preconditioner, but has runtime advantages.

6.2 Comparison of KASKADE and PLTMG

As a rule of thumb one may conclude:

> *KASKADE is 3–5 times faster than PLTMG, but uses 2–3 times as much memory as PLTMG, while they are comparably robust and reliable.*

More detailed we observed the following:

- In both programs the linear solver needs only about 10% of the total runtime.
- The triangle oriented error estimator of PLTMG needs roughly twice as many evaluations per point of the coefficient functions of the elliptic operator as the edge oriented error estimator of KASKADE. This could be a serious drawback for real applications with expensive function evaluations.
- Integration process and grid refinement are much faster in KASKADE than in PLTMG for reasons of data structure and implementation (C versus FORTRAN).

- For the same reasons PLTMG has the memory advantage. *There is a trade off between speed and memory requirement.*

Appendix A: Test Problems

We use the Dirichlet boundary condition in all examples, except in Example 6c, where we have additionally natural boundary conditions.

1. The solution of this problem has a sharp peak. Two variations of the problem run on different domains.

	source	[21]
	equation	Poisson
a)	peak at $(0.5, 0.117)$	
	domain	unit square
	initial triangles	isosceles right
	angle bounds	minimum 18.43°; maximum 116.57°
	solution	$x(x-1)\ y(y-1)e^{-100((x-0.5)^2+(y-0.117)^2)}$
b)	peak at $(0,0)$	
	domain	hexagon with corners $(1,0)$, $(\frac{1}{2}, \sqrt{3}/2)$, $(-\frac{1}{2}, \sqrt{3}/2)$, $(-1,0)$, $(-\frac{1}{2}, -\sqrt{3}/2)$, and $(\frac{1}{2}, -\sqrt{3}/2)$
	initial triangles	equilateral
	angle bounds	minimum 30°; maximum 90°
	solution	$(x+1)(x-1)(y+1)(y-1)e^{-100(x^2+y^2)}$

2. The solution of this problem has a boundary layer along the lines $x = 1$ and $y = 1$.

source	[6, 21, 19]
equation	$\nabla^2 u - 100u = f$
domain	unit square
initial triangles	isosceles right
angle bounds	minimum 18.43°; maximum 116.57°
solution	$\dfrac{\cosh(10x) + \cosh(10y)}{2\cosh 10}$

3. The solution of this problem has four mild peaks. It is fairly smooth so that a uniform grid should do nearly as well as adaptive grids. The equation has a nonconstant coefficient.

source	[21]
equation	$\nabla^2 u - (100 + \cos 2\pi x + \sin 3\pi y)u = f$
domain	unit square
initial triangles	isosceles right
angle bounds	minimum 18.43°; maximum 116.57°
solution	$-0.31(5.4 - \cos 4\pi x)(\sin \pi x)(y^2 - y)$
	$\cdot (5.4 - \cos 4\pi y)(1/(1 + \Phi^4) - 0.5)$
	$\Phi = 4(x - 0.5)^2 + 4(y - 0.5)^2$

4. The solution of this problem has a wavefront along the lines $x = 0.5$, $0 < y < 0.5$ and $y = 0.5$, $0 < x < 0.5$. The initial triangulation consists of tall isosceles triangles.

source	[21]
equation	Poisson
domain	hexagon with corners $(1, \frac{1}{2})$, $(\frac{7}{8}, 1)$, $(\frac{1}{8}, 1)$, $(0, \frac{1}{2})$, $(\frac{1}{8}, 0)$, and $(\frac{7}{8}, 0)$
initial triangles	isosceles height $\frac{1}{2}$ width $\frac{1}{4}$
angle bounds	minimum 14.04°; maximum 129.09°
solution	$\Phi(x)\Phi(y)$
	$\Phi(x) = 1$ for $x \leq 0.4$
	$\Phi(x) = 0$ for $x \geq 0.6$
	Φ is a quintic polynomial for $0.4 \leq x \leq 0.6$
	such that Φ has two continuous derivatives

5. The solution of this problem is an eighth–degree harmonic polynomial with moderate peaks at the four corners of the domain. The initial triangulation contains 3 shapes of triangles with some relatively small angles.

source	[21]
equation	Laplace
domain	$(-1, 1) \times (-1, 1)$
initial triangles	3 types of isosceles triangles
angle bounds	minimum 9.46°; maximum143.97°
solution	$1.1786 - 0.1801p + 0.006q$
	$p = x^4 - 6x^2y^2 + y^4$
	$q = x^8 - 28x^6y^2 + 70x^4y^4 - 28x^2y^6 + y^8$

6. The solution of this problem has a singularity at the origin, which is a reentrant corner of the domain. As in Problem 1, two variations of this problem are used for different initial triangles. The strength of the singularity is different for the two versions.

	source	[2, 7, 8, 19, 20]
	equation	Laplace
a)	domain	L–shaped $(-1,1)\times(-1,1)\backslash(0,1)\times(-1,0)$
	initial triangles	isosceles right
	angle bounds	minimum 18.43°; maximum 116.57°
	solution	$r^{2/3}\sin\frac{2\Theta}{3}$ (polar coordinates)
b)	domain	hexagon as in 1b with a slit along the line $(y=0, x>0)$
	initial triangles	equilateral
	angle bounds	minimum 30°; maximum 90°
	solution	$r^{1/4}\sin\frac{\Theta}{4}$
c)	domain	circle with a slit along the line $(y=0, x>0)$
	initial triangles	equilateral
	angle bounds	minimum 30°; maximum 90°
	solution	$r^{1/4}\sin\frac{\Theta}{4}$

7. The solution of this problem is harmonic, but drops very sharply near $(0.01, 0)$. If the domain were extended to $x = 0$, there would be a jump discontinuity in the boundary condition.

source	[17]
equation	Laplace
domain	$(0.01,1)\times(-1,1)$
initial triangles	right triangles with legs of length 1 and 0.495
angle bounds	minimum 12.43°; maximum135.29°
solution	$\arctan\frac{y}{x}$

8. The coefficient function in the operator of this equation is discontinuous. $a(x, y)$ is piecewise constant with the values 1 and 100 on alternate triangles of the initial triangulation. The solution is continuous, but the first derivative has a jump discontinuity where a is discontinuous.

source	[6]
equation	$\nabla a\nabla u = 0$ where a is piecewise constant as described above
domain	hexagon as in 1b
initial triangles	equilateral
angle bounds	minimum 30°; maximum 90°
solution	$\frac{y(3x^2-y^2)}{a}$

Appendix B: Grids

The initial coarse triangulation used for all our computation is depicted by slightly thicker pensize in the following drawings. Figures 6.6–6.9 show the meshes for the mean value refinement strategy and the extrapolation refinement strategy.

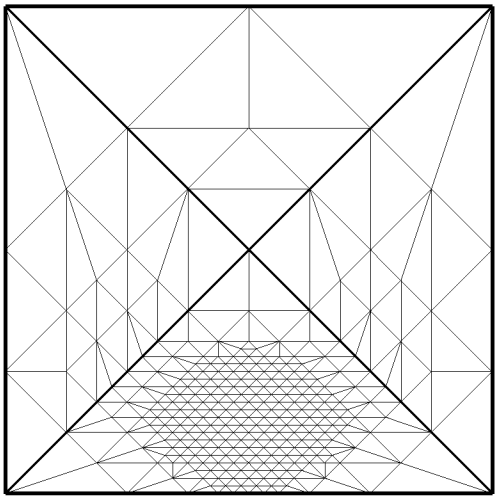


Figure B.1: Problem 1a, mean, $N_5 = 275$, $\varepsilon_{\text{grid}} = 7.73_{10} - 3$

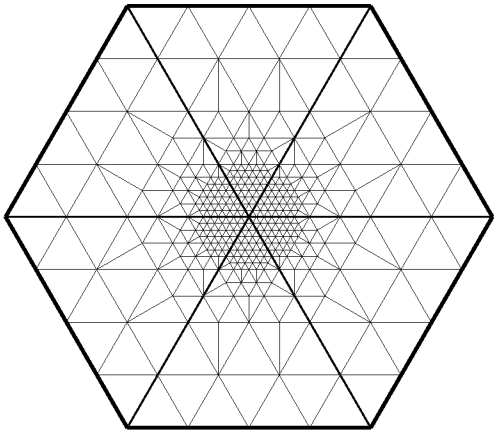


Figure B.2: Problem 1b, mean, $N_5 = 313$, $\varepsilon_{\text{grid}} = 2.83_{10} - 1$

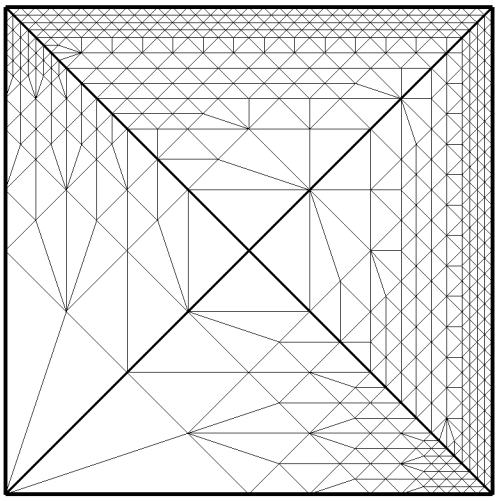


Figure B.3: Problem 2, extrap, $N_5 = 526$, $\varepsilon_{\text{grid}} = 1.09_{10} - 1$

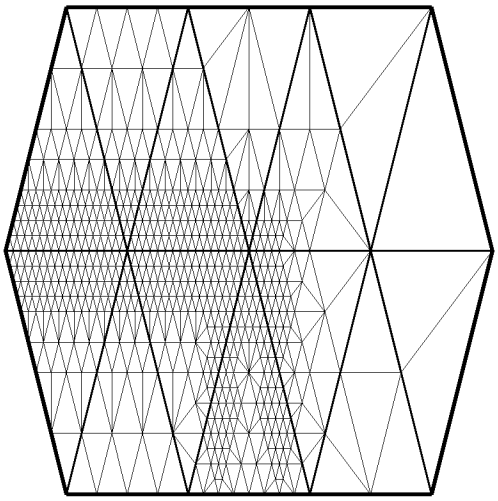


Figure B.4: Problem 4, mean, $N_4 = 529$, $\varepsilon_{\text{grid}} = 3.26_{10} - 1$

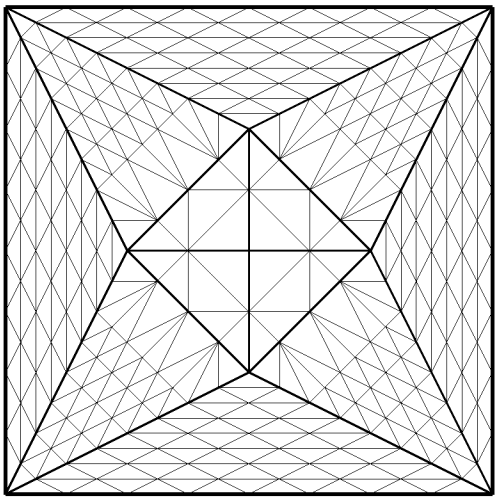


Figure B.5: Problem 5, extrap, $N_3 = 253$, $\varepsilon_{\text{grid}} = 3.37_{10} - 1$

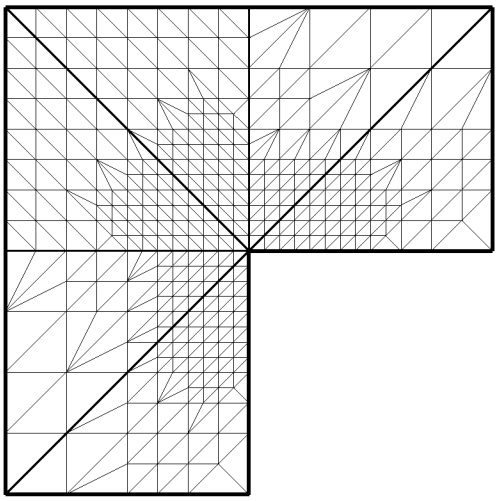


Figure B.6: Problem 6a, mean, $N_4 = 332$, $\varepsilon_{\text{grid}} = 8.17_{10} - 2$

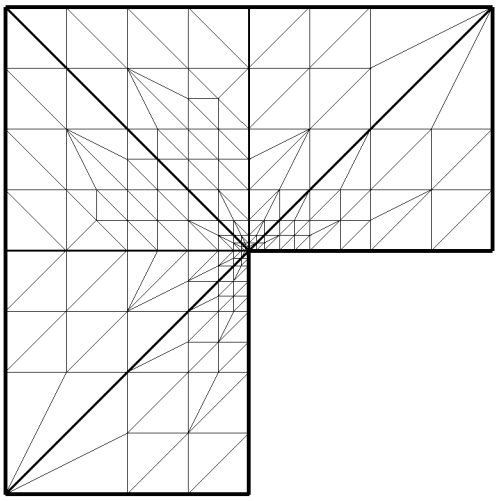


Figure B.7: Problem 6a, extrap, $N_6 = 134$, $\varepsilon_{\text{grid}} = 9.10_{10} - 2$

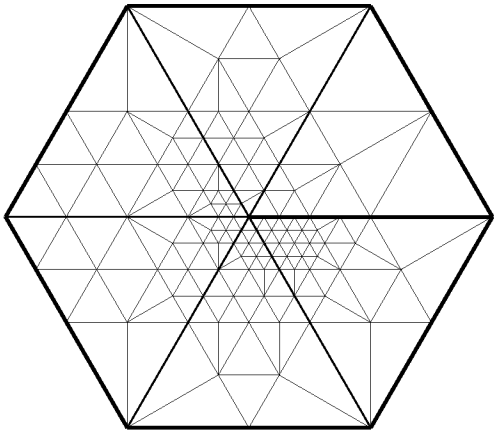


Figure B.8: Problem 6b, mean, $N_4 = 108$, $\varepsilon_{\text{grid}} = 4.17_{10} - 1$

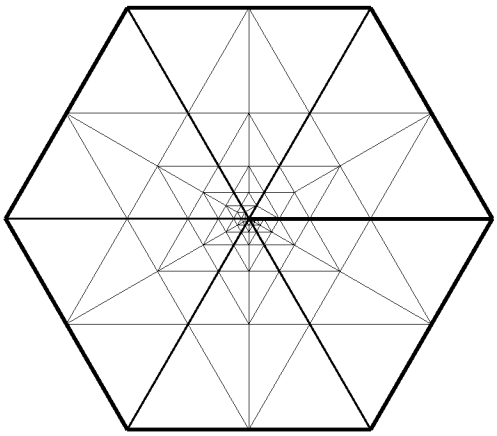


Figure B.9: Problem 6b, extrap, $N_7 = 77$, $\varepsilon_{\text{grid}} = 3.29_{10} - 1$

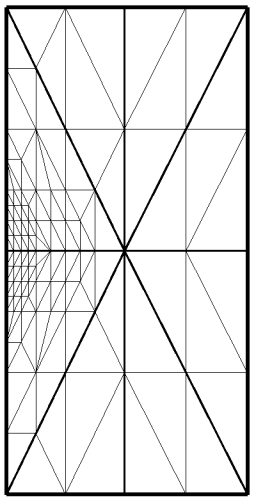


Figure B.10: Problem 7, mean, $N_4 = 90$, $\varepsilon_{\text{grid}} = 6.51_{10} - 1$

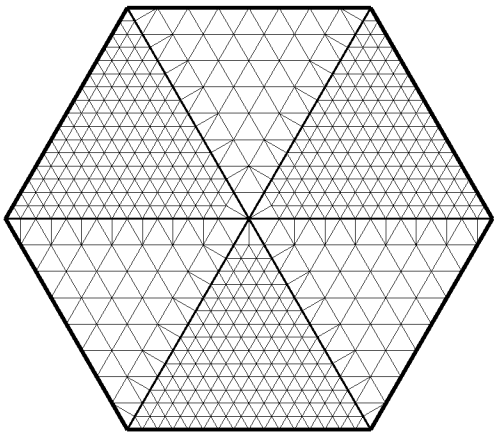


Figure B.11: Problem 8, extrap, $N_4 = 535$, $\varepsilon_{\text{grid}} = 1.14_{10} - 1$

Appendix C: Convergence History

Tables of Appendix C include the levels for which a solution was computed because for the levels resulting from the inner ESTIMATE/REFINE cycle only an interpolated solution exists.

N	lev	dep	ε_{est}	ε_L	$\|\hat{u}_L\|$	ζ
33	2	2	2.15e-02	4.57e-02	1.56e-02	-0.530
104	4	4	1.34e-02	1.45e-02	4.97e-02	-0.076
275	5	5	7.46e-03	7.80e-03	5.14e-02	-0.044
857	6	6	3.95e-03	4.11e-03	5.16e-02	-0.039
2954	7	7	2.10e-03	2.21e-03	5.17e-02	-0.050

Table C.1: Error analysis (mean) for Problem 1a

N	lev	dep	ε_{est}	ε_L	$\|\hat{u}_L\|$	ζ
37	2	2	2.15e-02	4.57e-02	1.57e-02	-0.530
61	3	3	2.09e-02	2.57e-02	4.54e-02	-0.187
103	4	4	1.39e-02	1.52e-02	4.99e-02	-0.086
145	5	5	1.05e-02	1.25e-03	5.02e-02	-0.160
405	6	6	6.27e-03	6.99e-03	5.12e-02	-0.103
899	7	7	3.91e-03	4.18e-03	5.15e-02	-0.065
1966	8	8	2.72e-03	2.99e-03	5.16e-02	-0.090
2778	10	9	2.19e-03	2.41e-03	5.16e-02	-0.091
3472	11	10	1.97e-03	2.16e-03	5.16e-02	-0.088
4829	12	11	1.74e-03	1.88e-03	5.16e-02	-0.074

Table C.2: Error analysis (extrap) for Problem 1a

N	lev	ε_{est}	ε_L	$\|\hat{u}_L\|$	ζ
64	6	2.86e-02	2.09e-02	4.73e-02	0.368
261	6	9.48e-03	8.06e-03	5.11e-02	0.176
1051	7	4.52e-03	4.11e-03	5.16e-02	0.100
3828	9	2.00e-03	1.92e-03	5.17e-02	0.042

Table C.3: Error analysis (PLTMG) for Problem 1a

N	lev	dep	ε_{est}	ε_L	$\|\hat{u}_L\|$	ζ
50	2	2	3.51e-01	5.52e-01	1.50e-00	-0.364
201	4	4	2.26e-01	3.59e-01	1.46e-00	-0.370
634	6	6	1.53e-01	2.44e-01	1.44e-00	-0.373
1672	8	8	1.07e-01	1.70e-01	1.44e-00	-0.371
4224	10	10	7.52e-02	1.19e-01	1.43e-00	-0.368

Table C.4: Error analysis (mean) for Problem 6c

N	lev	dep	ε_{est}	ε_L	$\|\hat{u}_L\|$	ζ
38	2	2	3.54e-01	5.64e-01	1.50e-00	-0.372
51	3	3	3.07e-01	4.69e-01	1.48e-00	-0.345
62	4	4	2.69e-01	4.06e-01	1.46e-00	-0.337
81	5	5	2.42e-01	3.46e-01	1.45e-00	-0.301
94	7	6	2.22e-01	3.00e-01	1.44e-00	-0.260
105	8	7	2.10e-01	2.86e-01	1.43e-00	-0.266
135	9	8	2.00e-01	2.52e-01	1.43e-00	-0.206
167	12	9	1.79e-01	2.24e-01	1.43e-00	-0.201
275	13	10	1.50e-01	1.78e-01	1.43e-00	-0.157
467	18	12	1.13e-01	1.35e-01	1.43e-00	-0.163
1453	19	13	7.34e-02	8.87e-02	1.43e-00	-0.172
2002	24	17	5.77e-02	6.46e-02	1.43e-00	-0.107
2670	27	20	4.99e-02	5.31e-02	1.43e-00	-0.060

Table C.5: Error analysis (extrap) for Problem 6c