Untersuchung iterativer Lösungsverfahren am Beispiel diskretisierter Konvektions–Diffusions–Reaktions–Gleichungen

Diplomarbeit

vorgelegt von
Andreas P. Priesnitz
aus
Göttingen

angefertigt am
Institut für
Numerische und Angewandte Mathematik
der
Georg–August–Universität zu Göttingen
1996

Inhaltsverzeichnis

Einleitung

Konvektions–Diffusions–Reaktions–Gleichungen der Form

$$\begin{aligned} -\varepsilon\Delta u + (b, \nabla u) + cu &= f \quad \text{auf } \Omega, \\ u &= 0 \quad \text{auf } \partial\Omega \end{aligned}$$

entstehen aus vielen Anwendungen in Naturwissenschaften und Technik als Problembeschreibung oder als Teilproblem bei der Lösung komplexerer Aufgaben, wie zum Beispiel der Navier–Stokes–Gleichungen der Strömungsmechanik. Ihre Lösung bereitet besonders im *singulär gestörten Fall* $\varepsilon \to 0$ Schwierigkeiten.
In Kapitel 1 werden Eigenschaften der Lösung in diesem Fall beschrieben, und es wird eine *diskretisierte Gebietszerlegungsmethode* zur Bestimmung der Lösung eingeführt, bei der diese durch die endlichdimensionale Lösung eines linearen Gleichungssystems approximiert wird. Dessen Matrix hat die besondere Eigenschaft, schwach besetzt zu sein, also nur wenige Einträge zu besitzen, die nicht Null sind.
In Kapitel 2 wird die Theorie der *iterativen Lösungsverfahren* dargestellt, die zum Zweck der Lösung schwachbesetzter Probleme entwickelt worden ist. Die Leistungsfähigkeit dieser Verfahren hängt von der Kondition der Matrix ab. Diese kann bei Definition der Matrix durch ein Diskretisierungsverfahren sehr groß sein. Daher werden in Kapitel 3 *Vorkonditionierungsverfahren* beschrieben, die die Kondition des Systemes verbessern sollen.
Um Lösungsverfahren und Vorkonditionierer auszuwählen, die zur Lösung der durch die Methoden aus Kapitel 1 erzeugten Systeme besonders geeignet sind, werden in Kapitel 4 anhand von Modellproblemen die Verfahren und Vorkonditionierer getestet und verglichen. Aufgrund der durch die Experimente gewonnenen Erkenntnisse werden Empfehlungen zur Wahl solcher Methoden ausgesprochen.
In Kapitel 4 wird außerdem die Bibliothek *BLANC* vorgestellt, die im Rahmen dieser Arbeit sowohl zur Realisierung der Lösungsverfahren und Vorkonditionierer entwickelt wurde, wie auch als objekt–orientierte Grundlage für die komfortable und sichere Programmierung mathematischer Algorithmen auf blockstrukturierten Matrizen und Vektoren. Die Motivation dieses Paketes ist der Einsatz in Programmen zur Behandlung vektorwertiger Probleme, wie zum Beispiel der Navier–Stokes–Gleichungen, bei deren Lösung blockstrukturierte Matrizen und Vektoren entstehen. Zur Berücksichtigung dieses mathematischen Hintergrundes werden in den Kapiteln 2 und 3 Aussagen gegebenenfalls auch für Blockmatrizen formuliert werden.

Kapitel 1

Singulär gestörte Konvektions–Diffusions–Reaktions–Gleichungen

In diesem Kapitel werden Eigenschaften der Lösung einer *Konvektions–Diffusions–Reaktions–Gleichung* sowie eine Methode zur Ermittlung dieser Lösung beschrieben. Insbesondere soll auf den *singulär gestörten Fall* eingegangen werden, daß der Einfluß durch Diffusion gering gegenüber dem durch Konvektion ist, bzw. beide gering gegenüber dem durch Reaktion sind. Nach der Beschreibung der zu erwartenden Eigenschaften der Lösung in Abschnitt 1.1 folgt in Abschnitt 1.2 auf Basis einer Finite–Elemente–Diskretisierung die Definition des *Galerkin–Least–Squares–Verfahrens* zur Lösung des Problems. Wie sich herausstellt, hängt die Güte der dadurch bestimmten Lösung von der Zerlegungsfeinheit ab. Damit steigt jedoch auch die Komplexität des zu lösenden linearen Gleichungssystems. Als Konsequenz werden in Abschnitt 1.3 *Gebietszerlegungsverfahren* vorgestellt, die das Problem in kleinere und daher einfachere, miteinander verknüpfte Teilprobleme umformen.
Da der Schwerpunkt dieser Arbeit die Lösung der entstehenden linearen Gleichungssysteme ist, bleibt die Darstellung der Methoden in diesem Kapitel relativ kurz und soll nur dem Überblick dienen. Für detailliertere Beschreibungen der Theorie sei der Leser auf [43] oder [45] verwiesen. Dort findet man auch die Definitionen von Begriffen und Symbolen, die hier nicht erläutert sind.

1.1 Analytische Eigenschaften der Lösung

1.1.1 Problemstellung und Lösbarkeit

Gegeben sei ein beschränktes Gebiet $\Omega \in \mathbb{R}^d$, $d \in \mathbb{N}$ (in der Praxis $d \in \{1, 2, 3\}$) mit Lipschitz–stetigem Rand $\partial\Omega$. Gesucht ist eine Lösung u des *stationären* (zeitun-

abhängigen) *Konvektions–Diffusions–Reaktions–Problems*

$$Lu := -\varepsilon\Delta u + (b, \nabla u) + cu = f \quad \text{auf } \Omega, \tag{1.1}$$

($(\cdot,\cdot)$ bezeichnet stets das euklidische Skalarprodukt) mit homogener Dirichlet–Randbedingung

$$u = 0 \quad \text{auf } \partial\Omega. \tag{1.2}$$

Als Daten sind der *Diffusionskoeffizient* $\varepsilon > 0$, das *Geschwindigkeitsfeld* b, der *Absorptionskoeffizient* c und der *Quellterm* f gegeben. $-\varepsilon\Delta u$ heißt der *Diffusions-*, $(b, \nabla u)$ der *Konvektions-* und $cu - f$ der *Reaktionsanteil* des Problems.
Neben Aufgaben, bei denen solche Gleichungen direkt entstehen (z.B. bei der Untersuchung von Temperatur- oder Konzentrationsverteilungen einfacher chemischer Reaktionen in Strömungsfeldern), führen auch Methoden zur Lösung der Navier–Stokes–Gleichungen auf Probleme dieser Art (siehe z.B. in [43]).
Insbesondere interessiert man sich für die *singulär gestörten Fälle* der *Konvektionsdominanz* ($0 < \varepsilon \ll \|b\|_{L^\infty}$) und der *Reaktionsdominanz* ($0 < \varepsilon, \|b\|_{L^\infty} \ll \|c\|_{L^\infty}$).
Eine *klassische Lösung* $u \in C^2(\Omega) \cap C(\overline{\Omega})$ existiert dabei nur unter starken Glätteforderungen (Hölder–Stetigkeit) an den Rand, die Daten und die Randbedingungen. Gebiete mit Ecken, wie sie in der Praxis häufig auftreten, bereiten bereits Probleme (Beispiele in [45]).
Um einen allgemeineren Lösungsbegriff zu erhalten, der noch für

$$b \in W^{1,\infty}(\Omega, \mathbb{R}^d), \quad c \in L^\infty(\Omega, \mathbb{R}), \quad f \in L^2(\Omega, \mathbb{R})$$

gewisse Glätteeigenschaften besitzt, multipliziert man (1.1) mit einer *Testfunktion* $v \in H_0^1(\Omega, \mathbb{R}) = \{v \in H^1(\Omega, \mathbb{R}) \,:\, v|_{\partial\Omega} = 0\}$ und integriert beide Seiten über Ω. Man erhält daraus durch partielle Integration

$$\int_\Omega (-\varepsilon\,\Delta u + (b, \nabla u) + cu)\, v\, dx = \int_\Omega f v\, dx$$

$$\overset{\text{p.I.}}{\Leftrightarrow} \quad \int_\Omega \varepsilon\, \nabla u \nabla v\, dx + \int_{\partial\Omega} \underbrace{\varepsilon\, \nabla u v}_{=0}\, dx + \int_\Omega ((b, \nabla u) + cu)\, v\, dx = \int_\Omega f v\, dx$$

und mit

$$a(u, v) := \int_\Omega (\varepsilon\, \nabla u \nabla v\, dx + ((b, \nabla u) + cu)\, v)\, dx$$
$$f(v) := \int_\Omega f v\, dx$$

die *verallgemeinerte* oder *schwache* Formulierung von (1.1):
Finde $u \in H_0^1(\Omega, \mathbb{R})$, so daß

$$a(u, v) = f(v) \quad \text{für alle } v \in H_0^1(\Omega, \mathbb{R}). \tag{1.3}$$

Im folgenden sei stets für ein festes $\omega > 0$ die Elliptizitätsbedingung

$$c - \frac{1}{2}\nabla \cdot b \geq \omega > 0 \quad \text{f.ü. auf } \Omega. \tag{1.4}$$

gegeben. Dann ist a eine H_0^1–elliptische Bilinearform, und damit ist, da a und f stetig sind, nach dem Lemma von Lax–Milgram die eindeutige Existenz der Lösung $u \in H_0^1$ gesichert.

1.1.2 Beschreibung der Lösung im singulär gestörten Fall

Im singulär gestörten Fall erwartet man für $\varepsilon \to 0$, daß die Lösung u von (1.1) in die Lösung w des *reduzierten Problems*

$$(b, \nabla w) + cw = f \tag{1.5}$$

mit geeigneten Randbedingungen übergeht. Starke Abweichungen entstehen dabei in der Nähe von $\partial\Omega$ in *Randgrenzschichten*, wenn die Randbedingungen nicht erfüllt werden können, und in *inneren Grenzschichten* in Ω, wenn w dort nicht hinreichend glatt ist. Für eine genauere Betrachtung der Grenzschichten zerlegt man $\partial\Omega$ in die Teilmengen

$$\begin{aligned}
\partial\Omega^+ &:= \{x \in \partial\Omega \ : \ (b(x), n(x)) > 0\}, \\
\partial\Omega^0 &:= \{x \in \partial\Omega \ : \ (b(x), n(x)) = 0\}, \\
\partial\Omega^- &:= \{x \in \partial\Omega \ : \ (b(x), n(x)) < 0\}.
\end{aligned}$$

$n(x)$ bezeichnet den auswärts gerichteten Normalenvektor im Randpunkt x. Es sind dann $\partial\Omega^+$ und $\partial\Omega^-$ *Aus-* und *Einströmrand* der durch b beschriebenen Strömung. Der *charakteristische Rand* $\partial\Omega^0$ ist der Teil des Randes, zu dem die Strömung parallel läuft. Randgrenzschichten, die in Ω abklingen, können nur an $\partial\Omega^+$ und $\partial\Omega^0$ entstehen (siehe dazu Abschnitt 1.3); daher wählt man als Randbedingung für das reduzierte Problem:

$$w = 0 \quad \text{auf } \partial\Omega^- \tag{1.6}$$

Satz 1.1
Mit $b \in C^1(\overline{\Omega}, \mathbb{R}^d)$, $c \in C^1(\overline{\Omega}, \mathbb{R})$ und $f \in L^2(\Omega, \mathbb{R})$ und unter der Elliptizitätsbedingung (1.4) existiert die Lösung $w \in L^2(\Omega, \mathbb{R})$ des reduzierten Problems (1.5)&(1.6) und ist eindeutig bestimmt.

Beweis: [5]

Im allgemeinen kann man für w nur geringe Glattheit erwarten. Für glatten Rand $\partial\Omega$, abgeschlossenen $\partial\Omega^-$ und hinreichend großes c läßt sich immerhin noch $w \in C^{k,\alpha}(\Omega, \mathbb{R})$ mit $k > 0$ zeigen (Beweis in [37]).

Wie hängen nun die Lösungen des reduzierten und des singulär gestörten Problems zusammen?

Satz 1.2
Unter den Voraussetzungen von Satz 1.1 konvergiert die Lösung u von (1.1) für $\varepsilon \to 0$ in $L^2(\Omega, \mathbb{R})$ schwach gegen w.

Beweis: [31]

Die Approximation von u durch w ist also nicht unbedingt gut im punktweisen Sinn. Entsprechende Aussagen erhält man aber durch stärkere Bedingungen an die Problemstellung:

Satz 1.3
Sind $\partial\Omega^+$ und $\partial\Omega^-$ abgeschlossen, so gilt in allen Punkten $x \in \Omega$, die hinreichend weit von Grenzschichten entfernt sind:

$$|u(x) - w(x)| \leq C\varepsilon^{\frac{1}{2}}$$

Beweis: [14]

Ist $\partial\Omega$ überdies glatt, so gilt diese Abschätzung nach [45] sogar mit ε statt $\varepsilon^{\frac{1}{2}}$.

Zur Bewertung der Grenzschichten versucht man, eine dort definierte *lokale Korrektur* v zu finden, mit der u durch $w + v$ ebenfalls punktweise approximiert wird. Ziel ist eine Abschätzung gleicher Ordnung, also

$$|u(x) - (w(x) + v(x))| \leq C\varepsilon^{\frac{1}{2}}.$$

Unter den gegebenen Voraussetzungen ($\partial\Omega^+$, $\partial\Omega^-$ sind abgeschlossen) läßt sich v konkret beschreiben:

Für $b \equiv 0$ ($\Rightarrow \partial\Omega^- = \partial\Omega$) und $c \geq \omega \gg \varepsilon$ (*reaktionsdominantes Diffusions–Reaktions–Problem*) ist v in der Randgrenzschicht an $\partial\Omega$ von der Gestalt

$$v(x) = -w(\tilde{x})\, e^{-G(\tilde{x})\frac{\rho}{\sqrt{\varepsilon}}}.$$

$\tilde{x}$ ist dabei die Projektion von x auf $\partial\Omega$ in Normalenrichtung und $\rho = \|x - \tilde{x}\|_2$ der Abstand von x zu $\partial\Omega$ (d.h. $x = \tilde{x} - \rho\, n(\tilde{x})$). G ist eine auf $\partial\Omega$ definierte positive problemabhängige Funktion (siehe [22]).
Man spricht in diesem Fall entsprechend von einer *exponentiellen Grenzschicht.*

Schwieriger erweist sich die Beschreibung von v in Randgrenzschichten bei Anwesenheit von Konvektion, also $b \not\equiv 0$, da nun die Strömung die diffusiven Vorgänge in den Grenzschichten beeinflußt. An $\partial\Omega^+$ liegt, ganz ähnlich wie oben, wegen

$$v(x) = -w(\tilde{x})\, e^{-G(\tilde{x})\frac{\rho}{\varepsilon}},$$

eine exponentielle Grenzschicht vor, wobei diesmal $\tilde{x}$ die Projektion von x auf $\partial\Omega^+$ in Normalenrichtung ist und $\rho = \|x - \tilde{x}\|_2$ der Abstand von x zu $\partial\Omega^+$. G ist auf $\partial\Omega^+$

definiert und wieder positiv.
Komplizierter ist der Fall von Grenzschichten an $\partial\Omega^0$ wegen des dazu parallelen Konvektionsverlaufs. Dort läßt sich v durch eine parabolische Differentialgleichung beschreiben, so daß dann eine *parabolische Grenzschicht* vorliegt. Die wegen ihrer Problembezogenheit praktisch relevanteren parabolischen Grenzschichten besitzen kompliziertere analytische Eigenschaften als die exponentiellen und sind „dicker". Durch diese Streckung ist aber auch ihr Verlauf glatter, der Betrag ihrer Ableitung geringer. Damit sie nicht durch die wesentlich erheblichere Ableitung der exponentiellen Grenzschicht überdeckt wird, wird diese oft durch die Wahl von Neumann– statt Dirichlet–Bedingungen am Ausströmrand gedämpft. Dadurch wird die Grenzschicht von u auf $\frac{\partial u}{\partial n}$ verlagert. Nach [45] konvergiert erfahrungsgemäß so die Lösung u von (1.1), (1.2) für $\epsilon \to 0$ mit höherer Ordnung gegen die Lösung w des reduzierten Problems.

Sind die obigen Voraussetzungen nicht erfüllt, ist also $\partial\Omega^+$ oder $\partial\Omega^-$ nicht abgeschlossen, so entstehen schwierigere Fälle. Ein Beispiel sind Ecken zwischen $\partial\Omega^+$ und $\partial\Omega^0$, an denen sich exponentielle und parabolische Grenzschichten überlagern. Hier sind komplexere Korrekturen nötig.

Wie oben erwähnt, können Grenzschichten auch entfernt vom Rand als innere Grenzschichten dort entstehen, wo w nicht hinreichend glatt ist. Um das Verhalten der Lösung im Inneren von Ω zu beschreiben, insbesondere an inneren Grenzschichten, definiert man für $x \in \overline{\Omega}$ in Ω die *Charakteristik* $\xi_x(t)$ durch die Differentialgleichung

$$\frac{d\xi_x}{dt} = b(\xi_x(t)), \quad \xi_x(0) = x.$$

Sie beschreibt den Strömungsverlauf von x aus in Abhängigkeit von der Zeit t. Wichtig sind die Charakteristiken $\xi_x(t)$ mit $x \in \partial\Omega^-$, da entlang dieser die Information der Randbedingungen in Ω hineintransportiert wird. Liegt in $x \in \partial\Omega^-$ eine Unstetigkeit in den Randbedingungen vor und wird Ω von ξ_x „durchquert" (eine genauere Beschreibung dieses Sachverhalts wird in [45] gegeben), so entsteht eine parabolische innere Grenzschicht entlang ξ_x. Komplizierter ist das Verhalten der Lösung in der Nähe von Charakteristiken, die in Ω gegen eine *Grenzmenge* konvergieren. Mögliche Formen davon sind Nullstellen (*stationäre Punkte*) des Geschwindigkeitsfeldes (je nach Strömungsrichtung in der Umgebung unterscheidet man *Quellen* und *Senken*), Konvergenz der Charakteristik gegen einen *Grenzkreis* oder *geschlossene Charakteristiken*.

Eine genauere Beschreibung der hier angesprochenen analytischen Eigenschaften der Lösung in Grenzschichten und Aussagen zu a–priori–Abschätzungen des zu erwartenden Fehlers findet man in [45].

1.2 Diskretisierung des Randwertproblems

Um (1.1)&(1.2) bzw. (1.3) numerisch zu lösen, sucht man Näherungslösungen u^n in endlichdimensionalen Unterräumen X_n mit $\dim(X_n) = n$, die so beschaffen sein sollen,

daß u^n für $n \to \infty$ gegen u strebt.
Zur Durchführung einer *Finite–Elemente–Methode* zerlegt man zur Konstruktion dieser Räume das Gebiet Ω in *finite Elemente*. Darunter versteht man eine Überdeckung $\mathcal{T}_h$ von Ω durch abgeschlossene Simplices oder Hyperquader T mit paarweise disjunkten Inneren, für die gilt:

- Für alle $T \in \mathcal{T}_h$ ist der Durchmesser h_T der kleinsten das Element T umschliessenden Sphäre nach oben durch $h > 0$ beschränkt.
- Das Verhältnis der Radien von kleinster umbeschreibbarer und größter einbeschreibbarer Sphäre ist beschränkt.
- Ein Eckpunkt eines Elements ist Eckpunkt aller Elemente, zu denen er gehört.

$X_n = X_n^k$ wird dann als Schnittmenge des jeweils zulässigen Raumes für die Lösung u mit dem Raum der stückweise auf den Elementen polynomialen Funktionen (vom Grad $k \geq 1$) gewählt (dabei existieren verschiedene Methoden zur Konstruktion einer Basis von X_n^k, siehe [43]).

Beim *Galerkin–Verfahren* formuliert man (1.3) diskret:
Suche $u^n \in X_n^k \subset H_0^1(\Omega, \mathbb{R})$, so daß

$$a(u^n, v) = f(v) \quad \text{für alle } v \in X_n^k.$$

Sei $\{\xi_1, \ldots, \xi_n\}$ eine Basis von X_n^k. Dann ist die Lösung u^n dadurch eindeutig bestimmt, daß die Forderung für $v := \xi_1, \ldots, \xi_n$ erfüllt wird. Mit der Basisdarstellung $u^n = \sum_{i=1}^n u_i^n \xi_i$ erhält man daraus die Darstellung des Galerkin–Problemes durch ein System $Au^n = f$ von n linearen Gleichungen mit $u_1^n, \ldots, u_n^n$ als Unbekannten.

Das *Least–Squares–Verfahren* dagegen fordert die Minimierung des Fehlers des Ausgangsproblems (1.1):
Suche $u^n \in X_n^k \subset H_0^1(\Omega, \mathbb{R}) \cap H^2(\Omega, \mathbb{R})$, so daß

$$\|Lu^n - f\|_{L^2} = \min_{v \in X_n^k} \|Lv - f\|_{L^2},$$

oder, äquivalent umformuliert:

$$(Lu^n - f, Lv) = 0 \quad \text{für alle } v \in X_n^k.$$

Auch hier erhält man mit einer Basis von X_n^k ein lineares Gleichungssystem.

Vorteile des Galerkin–Verfahrens gegenüber dem Least–Squares–Verfahren sind die einfachere Glätteforderung an die Approximierende u^n (und damit an die Basisfunktionen) und die bessere Kondition der Matrix des entstehenden Gleichungssystems.
Das Least–Squares–Verfahren ist jedoch gerade wegen der stärkeren Glätteforderung

im konvektionsdominanten Fall stabiler, während das Galerkin–Verfahren bei Konvektion Oszillationen quer zur Grenzschicht verursachen kann ([43]). Außerdem führt Least–Squares zu einer einfacher zu lösenden positiv definiten symmetrischen Matrix. Um — insbesondere im konvektionsdominanten Fall — Vorteile der Verfahren zu verbinden und Nachteile zu vermeiden, vereinigt man sie im *Galerkin–Least–Squares–Verfahren (GLS)*:
Suche $u^n \in X_n^k \subset H_0^1(\Omega, \mathbb{R})$, so daß:

$$\underbrace{a(u^n, v) + \sum_{T \in \mathcal{T}_h} \delta_T \left(Lu^n, Lv\right)_T}_{=:a_h(u^n, v)} = \underbrace{f(v) + \sum_{T \in \mathcal{T}_h} \delta_T \left(f, Lv\right)_T}_{=:(f,v)} \quad \text{für alle } v \in X_n^k \tag{1.7}$$

mit $\delta_T > 0$ für alle T. $(\cdot, \cdot)_T$ bezeichnet das L_2–Skalarprodukt, ausgewertet auf dem Element T. Durch die Einschränkung auf T erreicht man, daß für die Durchführbarkeit des Verfahrens die Forderung $X_n^k \subset H_0^1(\Omega, \mathbb{R})$ genügt. δ_T bestimmt also, wie stark die Galerkin–Approximation durch die Least–Squares–Approximation korrigiert wird. Dabei ist das Verfahren für alle δ_T konsistent.
Mit

$$\|v\|_{GLS} := a_h(v, v)^{\frac{1}{2}} \quad \text{für alle } v \in X_n^k$$

ist eine zerlegungsabhängige Norm gegeben. Man erreicht folgende (gegenüber dem reinen Galerkin–Verfahren verbesserte) Stabilitätsaussage:

Satz 1.4
Die durch GLS definierte Bilinearform a_h ist X_n^k–elliptisch, die Linearform f_h stetig. Es gilt

$$\|u^n\|_{GLS} \leq C \|f\|_{L^2(\Omega, \mathbb{R})}$$

Beweis: [45], [43]

Satz 1.5
Sei $u \in H_0^{k+1}(\Omega, \mathbb{R})$ mit $k \geq 1$. Dann gilt für die Lösung u^n von (1.7):

$$\|u - u^n\|_{GLS} \leq C h^k \left(\sum_{T \in \mathcal{T}_h} \left(\varepsilon + \varepsilon^2 \delta_T h_T^{-2} + \delta_T + h_T^2 + \delta_T^{-1} h_T^2 \right) |u|_{H^{k+1}}^2 \right)^{\frac{1}{2}},$$

wobei $|\cdot|_{H^{k+1}}$ die H^{k+1}–Seminorm und h_T den Durchmesser von T darstellt.

Beweise: [45]

Um optimale Konvergenz zu erreichen, bietet sich die Wahl

$$\delta_T := \delta_0 \frac{h_T}{\sqrt{1 + (\varepsilon/h_T)^2}}$$

mit beliebigem $\delta_0 > 0$ für den GLS–Parameter δ_T an, wodurch die rechte Seite von (1.8) minimiert wird.

Da die durch das Least–Squares–Verfahren definierte Matrix symmetrisch und positiv definit ist, wirkt deren Anteil in der GLS–Matrix numerisch stabilisierend. Wegen des Galerkin–Anteils ist diese zwar noch positiv definit, aber im allgemeinen nicht mehr symmetrisch ([43]). Gewöhnlich besitzt sie auch keine M–Eigenschaft ([45]), doch läßt sich diese durch eine ausreichend feine Zerlegung und die Vermeidung stumpfer Winkel in den Finiten Elementen erzwingen.

Mit kleinerem h wird n und damit das zu lösende System größer. Da jedoch jede Zeile von A für alle h nur eine beschränkte und kleine Anzahl nichtnullwertiger Einträge enthält (entsprechend der Anzahl der Nachbarn der jeweiligen Stützstelle), entsteht daraus erst für sehr kleine h ein Speicherplatzproblem. Im 2. Kapitel werden Lösungsverfahren vorgestellt, die diese Struktur ausnutzen und so auch große Systeme effizient bearbeiten können.
Allerdings kann die Kondition von A mit steigender Feinheit potentiell anwachsen, was sich verheerend auf die Konvergenzeigenschaften jener Verfahren auswirkt. Um diesem Mißstand abzuhelfen, möchte man das System *vorkonditionieren*, also so umformen, daß das resultierende System bei gleicher Lösung eine (wesentlich) bessere Kondition hat und dadurch leichter lösbar ist. Man realisiert dies implizit, indem man aus einer Vereinfachung des Problems eine bereits relativ gute Näherungslösung gewinnt (mehr zum Begriff der Vorkonditionierung in Kapitel 3).

1.3 Gebietszerlegungsverfahren

Eine übliche Vorkonditionierung für Finite–Elemente–Methoden sind *Gebietszerlegungsverfahren*. Man zerlegt Ω in *überlappende* oder *nichtüberlappende* (d.h. sich schneidende oder disjunkte) Teilgebiete Ω_i, $i = 1, \ldots, m$ und betrachtet auf jedem dieser Teilgebiete die kleinere und entsprechend einfacher zu lösende Einschränkung des Ausgangsproblems. Damit dieser Lösungsbegriff auf dem gesamten Gebiet Ω seine Gültigkeit behält, verknüpft man die so entstehenden Teilprobleme durch Koppelungsbedingungen an den Rändern der Teilgebiete miteinander.
Hierbei liegt es nahe, die Zerlegung zur Parallelisierung des Problems zu nutzen, wobei die durch die Koppelungsbedingungen entstehende Abhängigkeit zwischen den Teilproblemen umgangen werden muß.
Ein weiterer Vorteil der Gebietszerlegung ist die Möglichkeit, die Teilgebiete so zu wählen, daß die Teilprobleme bestimmte Eigenschaften besitzen, die dann angepaßt

behandelt werden können. So kann man z.B. bei einem Strömungsproblem Zonen laminarer Strömung und Wirbel durch verschiedene Teilgebiete abdecken. Teilprobleme mit unterschiedlichen physikalischen Eigenschaften, wie z.B. unterschiedlicher Viskosität des Fluids, sind denkbar. Eine andere Möglichkeit ist, Grenzschichten vom einfacher zu lösenden inneren Teil von Ω zu trennen und jeweils geeignete Lösungsverfahren zu verwenden (*heterogene Methode*, siehe dazu [43]).
Es soll nun eine *nichtüberlappende* Gebietszerlegungsmethode zur Lösung des Konvektions–Diffusions–Reaktions–Problems formal dargestellt werden. Dazu wird vereinfachend eine Zerlegung von Ω in vorerst nur zwei disjunkte und Lipschitz-stetig berandete Teilgebiete Ω_1 und Ω_2 betrachtet, die den gemeinsamen Rand (*Interface*) $\Gamma := \partial\Omega_1 \cap \partial\Omega_2$ im Inneren von Ω besitzen. Sei $n_i(x)$ der auswärts gerichtete Normaleneinheitsvektor in $x \in \partial\Omega_i \cap \Gamma$ für $i = 1, 2$ und sei $n := n_1$. Zur Vereinfachung soll Γ stückweise stetig differenzierbar sein. Nun wird (1.1) auf diesen Teilgebieten äquivalent reformuliert:
Suche $u_i \in \Omega_i$, $i = 1, 2$, so daß

$$\begin{aligned} Lu_i &= f && \text{auf } \Omega_i, \\ u_i &= 0 && \text{auf } \partial\Omega_i \cap \partial\Omega, \\ u_1 &= u_2 && \text{auf } \Gamma, \qquad (1.8) \\ \frac{\partial u_1}{\partial n} &= \frac{\partial u_2}{\partial n} && \text{auf } \Gamma. \qquad (1.9) \end{aligned}$$

Die Dirichlet–Bedingung (1.8) und die Neumann–Bedingung (1.9) beschreiben die *Transmissionsbedingungen* zur Koppelung von u_1 und u_2 auf Γ, man nennt dies daher die *Dirichlet–Neumann–Formulierung* des Problems.
Gleichbedeutend zu (1.8), (1.9) sind die *Robin–Dirichlet–Formulierung*

$$\begin{aligned} \varepsilon\frac{\partial u_1}{\partial n} - (b,n)u_1 = \varepsilon\frac{\partial u_2}{\partial n} - (b,n)u_2 \quad &\text{auf } \Gamma, \\ u_1 = u_2 \quad &\text{auf } \Gamma \end{aligned}$$

und, falls Γ^0 höchstens aus einer endlichen Anzahl von Punkten besteht, die *Robin–Neumann–Formulierung*

$$\begin{aligned} \varepsilon\frac{\partial u_1}{\partial n} - (b,n)u_1 = \varepsilon\frac{\partial u_2}{\partial n} - (b,n)u_2 \quad &\text{auf } \Gamma, \\ \frac{\partial u_1}{\partial n} = \frac{\partial u_2}{\partial n} \quad &\text{auf } \Gamma. \end{aligned}$$

Einen Beweis der Äquivalenz dieser Formulierungen zu (1.1) findet man in [21]. Welchen Ansatz man wählt, hängt von den Problemdaten ab (siehe [21]).
Um die Teilprobleme unabhängig voneinander lösen zu können, formuliert man ein *Gebietsiterationsverfahren*, das zwei Sequenzen $\{u_1^k\}$ und $\{u_2^k\}$ erzeugt, die gegen u_1 bzw. u_2 konvergieren. Seien $\Gamma', \Gamma'' \subseteq \Gamma$ Interfaceabschnitte und Φ', Φ'', Ψ' und Ψ'' problemabhängige Differentialoperatoren. Damit definiert man:

Algorithmus 1 Gebietsiterationsverfahren

Wähle beliebige u_1^0, u_2^0 mit $u_i^0|_{\partial\Omega_1\cap\partial\Omega_2} = 0$.
Für $k = 0, 1, \ldots$:

Löse:

$$\begin{cases} Lu_1^{k+\frac{1}{2}} = f & \text{auf } \Omega_1 \\ u_1^{k+\frac{1}{2}} = 0 & \text{auf } \partial\Omega_1 \cap \partial\Omega \\ \Phi'(u_1^{k+\frac{1}{2}}) = \Phi'(u_2^k) & \text{auf } \Gamma' \\ \Phi''(u_1^{k+\frac{1}{2}}) = \Phi''(u_2^k) & \text{auf } \Gamma'' \end{cases}$$

Setze:

$$u_1^{k+1} := u_1^k + \vartheta'_{k+1}(u_1^{k+\frac{1}{2}} - u_1^k) \quad \text{auf } \overline{\Omega_1}$$

Löse:

$$\begin{cases} Lu_2^{k+\frac{1}{2}} = f & \text{auf } \Omega_2 \\ u_2^{k+\frac{1}{2}} = 0 & \text{auf } \partial\Omega_2 \cap \partial\Omega \\ \Psi'(u_2^{k+\frac{1}{2}}) = \Psi'(u_1^{k+1}) & \text{auf } \Gamma' \\ \Psi''(u_2^{k+\frac{1}{2}}) = \Psi''(u_1^{k+1}) & \text{auf } \Gamma'' \end{cases}$$

Setze:

$$u_2^{k+1} := u_2^k + \vartheta''_{k+1}(u_2^{k+\frac{1}{2}} - u_2^k) \quad \text{auf } \overline{\Omega_2}$$

Ende

Die Parameter ϑ' und ϑ'' dienen der Relaxation.
Eine parallelisierbare Version dieses Verfahrens erhält man unter Einbuße an Konvergenzgeschwindigkeit, wenn man das zweite Teilproblem folgendermaßen modifiziert:

Löse:

$$\begin{cases} Lu_2^{k+\frac{1}{2}} = f & \textit{auf } \Omega_2 \\ u_2^{k+\frac{1}{2}} = 0 & \textit{auf } \partial\Omega_2 \cap \partial\Omega \\ \Psi'(u_2^{k+\frac{1}{2}}) = \Psi'(u_1^k) & \textit{auf } \Gamma' \\ \Psi''(u_2^{k+\frac{1}{2}}) = \Psi''(u_1^k) & \text{auf } \Gamma'' \end{cases}$$

Mit $\Gamma' := \Gamma'' := \Gamma$ und $\Phi'' := \Psi'' := 0$ erhält man aus verschiedenen Formulierungen des Problems unter anderen folgende Methoden (siehe dazu [42]):

- die *Dirichlet–Neumann–Methode* durch

$$\begin{aligned} \Phi'(u_i^m) &:= u_i^m, \\ \Psi'(u_i^m) &:= \frac{\partial u_i^m}{\partial n}, \\ \vartheta'_m &:= \vartheta, \\ \vartheta''_m &:= 1, \end{aligned}$$

mit $\vartheta > 0$. Dadurch entsteht im ersten Teil eine Dirichlet–, im zweiten eine Neumann–Forderung.

- die *Robin–Methode* durch

$$\Phi'(u_i^m) := \Psi'(u_i^m) := \frac{\partial u_i^m}{\partial n} + p u_i^m,$$
$$\vartheta'_m := \vartheta''_m := 1,$$

mit beliebigem $p \in \mathbb{R}$. Dies führt zu Robin–Randbedingungen auf Γ.

- die *Lebedev–Agoshkov–Methode* durch

$$\Phi'(u_i^m) := q_m \frac{\partial u_i^m}{\partial n} + u_i^m,$$
$$\Psi'(u_i^m) := \frac{\partial u_i^m}{\partial n} + p_m u_i^m.$$

p_m, q_m, ϑ'_m und ϑ''_m sind frei wählbar. Aus dieser Methode entstehen die beiden vorigen und viele weitere aus der Literatur als Spezialfälle.

Nun läßt sich Γ, wie oben $\partial\Omega$, in die Teilmengen

$$\Gamma^+ := \{x \in \Gamma \ : \ (b(x), n(x)) > 0\},$$
$$\Gamma^0 := \{x \in \Gamma \ : \ (b(x), n(x)) = 0\},$$
$$\Gamma^- := \{x \in \Gamma \ : \ (b(x), n(x)) < 0\}$$

zerlegen. Γ^+ ist also der *Ausströmrand*, Γ^- der *Einströmrand* von Ω_1, umgekehrt für Ω_2.
Um künstliche Grenzschichten durch die Zerlegung zu vermeiden, formuliert man *adaptive* Versionen der Gebietsiterationsmethoden. Mit $\Gamma' := \Gamma^+$, $\Gamma'' := \Gamma^-$ und beliebigen ϑ'_m, ϑ''_m erhält man zum Beispiel (siehe dazu [21]):

- die *adaptive Dirichlet–Neumann–Methode* durch

$$\Phi''(u_i^m) := \Psi'(u_i^m) := u_i^m,$$
$$\Phi'(u_i^m) := \Psi''(u_i^m) := \frac{\partial u_i^m}{\partial n}.$$

Man stellt also am jeweiligen Ausströmrand eine Neumann–, am Einströmrand eine Dirichlet–Bedingung, wie bereits im vorigen Abschnitt für den nichtzerlegten Fall empfohlen wurde.

- die *adaptive Robin–Neumann–Methode* durch

$$\Phi''(u_i^m) := \Psi'(u_i^m) := \varepsilon \frac{\partial u_i^m}{\partial n} - (b, n) u_i^m,$$
$$\Phi'(u_i^m) := \Psi''(u_i^m) := \frac{\partial u_i^m}{\partial n}.$$

- die *adaptive Robin–Methode nach Nataf/Rogier* ([34]) durch

$$\begin{aligned}\Phi'(u_i^m) := \Psi'(u_i^m) &:= \varepsilon\frac{\partial u_i^m}{\partial n} + \frac{1}{2}\left(\sqrt{(b,n)^2 + 4\varepsilon c} - (b,n)\right) u_i^m,\\ \Phi''(u_i^m) := \Psi''(u_i^m) &:= 0,\\ \vartheta'_m := \vartheta''_m &:= 1.\end{aligned}$$

Alle vorgestellten Verfahren sind bei entsprechender Anpassung von Φ', Φ'', Ψ' und Ψ'' auch zur Lösung anderer Randwertprobleme anwendbar.
Betrachtet man nun statt einer Zerlegung von Ω in zwei Gebiete eine beliebige Zerlegung (unter gleichen Anforderungen an die Interfacekanten), so überträgt sich das Gebietsiterationsverfahren in unmittelbarer Weise. Beim Lösen der Teilprobleme werden nun alle Transmissionsbedingungen auf Interfaces zu benachbarten Teilgebieten gleichzeitig berücksichtigt. Auf welche Weise dabei die Teilprobleme gelöst werden, ist vom Gebietsiterationsalgorithmus unabhängig.

Jetzt stellt sich die Frage nach der effizienten Lösung der aus den Teilproblemen entstehenden großen und schwachbesetzten Gleichungssysteme.

Kapitel 2

Lösungsverfahren für große und schwachbesetzte lineare Gleichungssysteme

2.1 Voraussetzungen

Problem 2.1
Gesucht sei für $n \in \mathbb{N}$ die Lösung $x \in \mathbb{R}^n$ des Gleichungssystems

$$Ax = b \tag{2.1}$$

mit $A = (a_{i,j})_{1 \le i,j \le n} \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$. Sei

$$N := |\{a_{i,j} \neq 0 \,;\, 1 \le i, j \le n\}|$$

die Anzahl der nichtnullwertigen Einträge von A.
Das System habe folgende Eigenschaften:

$$\det A \neq 0, \tag{2.2}$$

$$N = \mathcal{O}(n) \tag{2.3}$$

Die Matrix bzw. das Problem ist also *nichtsingulär* (und somit eindeutig lösbar), *schwachbesetzt* (und damit auch „groß“, i.d.R. $n \ge 10^3$) und im allgemeinen *nichtsymmetrisch*. Alle Aussagen in diesem Kapitel werden aber auch für symmetrische Matrizen zutreffen; insbesondere werden einige speziell für diese formuliert.
Ferner sei vorausgesetzt, daß das Problem *sequentiell* (also konventionell) zu lösen ist. *Parallele* Verfahren müssen ganz anderen Anforderungen genügen; sie verwenden zwar auch die Ideen der hier vorgestellten Verfahren, realisieren diese aber in der Regel auf grundverschiedene (dann auch sehr von der Rechenumgebung abhängige) Weise. Dem

an parallelen Ansätzen interessierten Leser seien [35] und [44] als einführende Literatur empfohlen.
Eine weitere technische Einschränkung ist die limitierte Speicherkapazität. In jedem Fall benötigt man $N + 2n$ Speicherplätze zur Aufnahme der Problemgrößen A, x und b. Der Speicheraufwand eines Verfahrens soll dann auch auf die entsprechende Größenordnung $\mathcal{O}(n)$ begrenzt bleiben, wobei natürlich in diesem groben Rahmen eine weitere Differenzierung beim Vergleich der Verfahren nötig ist.

Nun sollen systematisch Verfahren zur Lösung von (2.1) unter den beschriebenen Voraussetzungen vorgestellt werden, ausgehend von der Unterteilung in *direkte* und *iterative* Verfahren.

Definition 2.2
Ein direktes Verfahren *bestimmt, bis auf Rundungsfehler, die* exakte Lösung x *von (2.1) durch eine endliche und geschlossene Rechnung.*

Direkte Verfahren sind Variationen des Gaußschen Eliminationsverfahrens. Da dieses durch Erzeugen neuer nichtnullwertiger Matrixeinträge einen Speicheraufwand von $\mathcal{O}(n^2)$ erreichen kann, ist es im allgemeinen Fall bei heutiger Technologie gewöhnlich bereits für Probleme der Größe $\mathcal{O}(10^4)$ (was bei der Diskretisierung partieller Differentialgleichungen ein kleiner Wert ist) unbrauchbar. Erst auf speziell strukturierten Matrizen (Bandmatrizen u.ä.) läßt sich die Anzahl zusätzlich erzeugter Einträge reduzieren. Die so formulierten direkten Verfahren können dann allerdings extrem leistungsfähig sein. Um durch Diskretisierungen solche Matrizen zu erhalten, sind regelmäßige Vernetzungen erforderlich. Die erwünschte Matrixstruktur entsteht dann durch entsprechende Numerierung der Knotenpunkte oder durch Umsortieren der Matrix. Ausführliche Darstellungen von Möglichkeiten zur Realisierung direkter Verfahren auf schwachbesetzten Matrizen findet man in [9], [10] und [36].

Definition 2.3
Ein iteratives Verfahren *liefert, ausgehend von einer gegebenen Näherungslösung x_0, sukzessive* Näherungslösungen x_k, $k = 1, 2, \ldots$, *für (2.1).*

Ob dabei die exakte Lösung x als Näherungslösung gefunden wird, ist nicht notwendigerweise gesichert. Einige Verfahren garantieren, exakte Arithmetik vorausgesetzt, daß ein $\tilde{k} \leq n$ mit $x_{\tilde{k}} = x$ existiert. Insofern kann man diese auch als direkte Verfahren bezeichnen. Anders als bei direkten Verfahren werden bei erfolgreichem Verlauf der iterativen durch ihre Näherungslösungen x_k bereits für $k < n$ und auch für $k \ll n$ „sinnvolle“ Werte für x dargestellt. Diese reichen in der Regel aus, zumal die exakte Lösung des Problems in der Praxis wegen Rundungsfehlern ohnehin nicht bestimmt werden kann. Der neue Lösungsbegriff wird durch folgende Definition beschrieben:

Definition 2.4
Seien eine Norm $\|\cdot\|$ *auf* $\mathbb{R}^n$ *und* $\delta > 0$ *gegeben. Für* $\tilde{x} \in \mathbb{R}^n$ *heißt*

$$\begin{aligned} &\|x - \tilde{x}\| && \textit{der} \text{ (echte) Fehler} \\ &\frac{\|x - \tilde{x}\|}{\|x\|} && \textit{mit } x \neq 0 \textit{ der} \text{ relative Fehler} \end{aligned}$$

von $\tilde{x}$ *bezüglich* x. $\tilde{x}$ *heißt* hinreichend gute Näherung *von* x, *wenn*

$$\|x - \tilde{x}\| < \delta. \tag{2.4}$$

Da x nicht bekannt ist, betrachtet man das *Residuum* $\tilde{r} := b - A\tilde{x}$ von $\tilde{x}$, womit man wegen

$$\|x - \tilde{x}\| = \|A^{-1}(b - A\tilde{x})\| = \|A^{-1}\tilde{r}\| \leq \|A^{-1}\|\|\tilde{r}\|$$

aus einer Schranke $\tilde{\delta}$ mit

$$\|\tilde{r}\| \leq \tilde{\delta}$$

eine Abschätzung der Form (2.4) erhält. Kennt man $\|A^{-1}\|$, so ergibt sich die Wahl $\tilde{\delta} := \delta/\|A^{-1}\|$. Anderenfalls betrachtet man:

Satz 2.5
(Störungslemma): *Seien* $\Delta A \in \mathbb{R}^{n\times n}$ *und* $\Delta b \in \mathbb{R}^n$ *so gewählt, daß* $\tilde{x}$ *das System*

$$(A + \varepsilon\Delta A)\,\tilde{x} = b + \varepsilon\Delta b$$

löst. Sei $\mathrm{cond}(A) := \|A\|\,\|A^{-1}\|$ *die* Kondition *von* A *bezüglich* $\|\cdot\|$. *Dann gilt*

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \varepsilon\,\mathrm{cond}(A)\left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|}\right) + \wr(\varepsilon). \tag{2.5}$$

Beweis: [23]

Diese Aussage unterstellt, daß im Fall einer schlechtkonditionierten Matrix A kleine Fehler im System große Fehler in der Lösung verursachen können. Andersherum kann man aus einer Abschätzung der rechten Seite von (2.5) eine des relativen Fehlers erhalten. In [1] und [6] werden Abschätzungen für $\|\tilde{r}\|$ vorgestellt, die Abschätzungen von $\|\Delta b\|/\|b\|$ und $\|\Delta A\|/\|A\|$ erreichen. Daraus lassen sich dann Abbruchkriterien für iterative Verfahren formulieren.So führt zum Beispiel bei Kenntnis von $\|A\|$ oder einer Abschätzung davon die Forderung

$$\|\tilde{r}\| \leq \gamma(\|A\| \cdot \|\tilde{x}\| + \|b\|)$$

zu

$$\|\Delta b\|/\|b\| \leq \gamma \quad \text{und} \quad \|\Delta A\|/\|A\| \leq \gamma$$

und die oft verwendete Vereinfachung

$$\|\tilde{r}\| \leq \gamma \|b\|$$

zu

$$\|\Delta b\|/\|b\| \leq \gamma \quad \text{und} \quad \|\Delta A\| = 0.$$

Die Startlösung x_0 wird zufällig gewählt (z.B. $x_0 := 0$) oder aus vorangegangenen Näherungsrechnungen übernommen. Hier knüpfen *Mehrgitterverfahren* (*Multigrid-/Multilevel-/Multiskalenverfahren*) an, die aus einer Vereinfachung des Ausgangsproblems unter somit geringerem Aufwand (rekursiv) eine bereits relativ gute Startlösung berechnen. Diese Verfahren sind sehr leistungsfähig, können jedoch im Rahmen dieser Arbeit nicht berücksichtigt werden. Zu diesem Thema siehe z.B. [7] oder [25].
Die iterativen Verfahren zerfallen nun wieder in zwei Klassen: die *stationären* und die *instationären* Verfahren, wobei die letzteren durch die Klasse der *Petrov–Galerkin–Krylov*–Verfahren repräsentiert werden.

2.2 Stationäre Verfahren

In diesem Abschnitt wird der Iterationsindex hochgeschrieben, um ihn von den Komponentenindices der Vektoren zu unterscheiden, also x^k statt x_k.
Ferner soll die Struktur der Matrix auf folgende Weise genauer beschrieben werden:

Definition 2.6
Seien $l, m \in \mathbb{N}$, so daß $lm = n$. $A \in \left(\mathbb{R}^{l\times l}\right)^{m\times m}$ (d.h. $A = (A_{i,j})_{1\leq i,j,\leq m}$ mit $A_{i,j} \in \mathbb{R}^{l\times l}$) heißt Blockmatrix, *$b \in \left(\mathbb{R}^{l}\right)^{m}$ (d.h. $b = (b_i)_{1\leq i\leq m}$ mit $b_i \in \mathbb{R}^l$)* Blockvektor. *Entsprechend heißt $A_{i,j}$* Matrixblock, *b_i* Vektorblock.

In natürlicher Weise gilt nach wie vor $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$. Insbesondere ist bei der Berechnung des Matrix–Vektor–Produktes gleich, ob man es herkömmlich ausrechnet oder im Blocksinn die einzelnen Matrixblock–Vektorblock–Produkte auswertet und addiert. Nur an wenigen Stellen wird deshalb die besondere Berücksichtigung von Matrizen als Blockmatrizen nötig werden. Für $l = 1$ erhält man gewöhnliche Matrizen und Vektoren als Spezialfälle. Jede Matrix ist also auch eine Blockmatrix.
Es soll an A die zusätzliche Bedingung

$$\det A_{i,i} \neq 0 \quad \text{für} \quad 1 \leq i \leq m \tag{2.6}$$

gestellt werden. Die Diagonalblöcke von A sollen also invertierbar bzw. für $l = 1$ die Diagonaleinträge nicht nullwertig sein. Dies ist durch (2.2) zwar noch nicht gewährleistet, kann aber, wie man induktiv nachweist, ohne weitere Einschränkung der bisher gestellten Bedingungen und ohne Änderung der Lösung x durch Umsortieren der Blockzeilen (Zeilen mit Blöcken als Einträgen) in (2.1) erreicht werden.

Definition 2.7
Ein iteratives Verfahren zur Lösung des Gleichungssystems $Ax = b$ mit $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ heißt stationär, *falls es eine konstante Zerlegung $A = M - N$ mit einer nichtsingulären Matrix M gibt, so daß die Iterationsvorschrift in der Form*

$$Mx^{k+1} := Nx^k + b \quad \text{für } k = 1, 2, \ldots$$

bei einer gegebenen Startlösung $x^{(0)}$ dargestellt werden kann. Mit der Iterationsmatrix *$B := M^{-1}N$ und $c := M^{-1}b$ schreibt man*

$$x^{k+1} := Bx^k + c \quad \text{für } k = 1, 2, \ldots$$

Insbesondere wird x^k auf dem gesamten $\mathbb{R}^n$ bestimmt, im Gegensatz zu den im nächsten Abschnitt vorgestellten *Petrov–Galerkin*–Verfahren, die sich auf einen affinen Unterraum beschränken.

Satz 2.8
Die durch ein stationäres Verfahren produzierten Iterierten x^k konvergieren für beliebiges $x^{(0)}$ gegen die exakte Lösung x genau dann, wenn

$$\rho(B) < 1, \tag{2.7}$$

wobei $\rho(B) := \max_{\lambda\in\sigma(B)} |\lambda|$ der Spektralradius *von B ist.*

Beweis: [47]

Satz 2.9
Das asymptotische Konvergenzverhalten der durch ein stationäres Verfahren bestimmten Näherungslösungen wird beschrieben durch

$$\max_{x^0\in\mathbb{R}^n} \lim_{k\to\infty} \left(\frac{\|x^k - x\|}{\|x^0 - x\|} \right)^{\frac{1}{k}} = \rho(B), \tag{2.8}$$

wobei $\|\cdot\|$ eine beliebige natürliche Norm auf $\mathbb{R}^n$ ist.

Beweis: [47]

Falls ein stationäres Verfahren konvergiert, konvergiert es bezüglich des Fehlers also linear.
Die *Richardson–Iteration* ist eine Verallgemeinerung der Idee der stationären Verfahren, aus der sich für symmetrische und positiv definite Matrizen weitere Verfahren ableiten lassen. Diese weisen jedoch schwächere Konvergenzeigenschaften als das im weiteren vorgestellte CG–Verfahren auf und sollen deshalb hier nicht behandelt werden. Eine Darstellung dieser Methoden findet man in [43].
Speziell für Matrizen, die aus Finite–Differenzen–Diskretisierungen elliptischer oder parabolischer Differentialgleichungen entstehen, wurde 1955 von Peaceman und Rachford in [40] das *Alternating–Direction–Implicit*–Verfahren (*ADI*) vorgestellt.
Auf größere Klassen von Matrizen sind dagegen die *klassischen Verfahren* anwendbar, deren Idee es ist, in jedem Iterationsschritt einen Block des Residuums zu eliminieren. Um die diese Verfahren bestimmenden Zerlegungen $A = M - N$ zu beschreiben, seien für $A = (A_{i,j})_{1 \le i,j \le m}$ durch

$$U := (A_{i,j})_{1 \le j < i \le m} \quad D := (A_{i,i})_{1 \le i \le m} \quad O := (A_{i,j})_{1 \le i < j \le m}$$

die *strikt untere Blockdreiecksmatrix*, die *Blockdiagonalmatrix* und die *strikt obere Blockdreiecksmatrix* von A definiert, so daß also $A = U + D + O$.
Auf das *Jacobi–Verfahren* ($M := D$, $N := -U - O$) und das *Gauß–Seidel–Verfahren* ($M := U + D$, $N := -O$) soll hier wegen gewöhnlich schlechterer Konvergenz (Jacobi) bzw. als Spezialfall des *SOR-Verfahrens* (Gauß–Seidel) nicht gesondert eingegangen werden.

2.2.1 SOR

SOR steht für *Successive OverRelaxation*. Zum einen bedeutet dies, daß man in jedem Iterationsschritt sukzessive die Komponenten des Residuums eliminiert, wobei die vorher im selben Schritt berechneten Einträge mitberücksichtigt werden (wie beim Gauß–Seidel–Verfahren). Zum anderen wird bei jedem Update die neu berechnete Komponente noch um einen Faktor $\omega \in \mathbb{R}$ überrelaxiert (extrapoliert).
Das Verfahren ist beschrieben durch

$$\begin{aligned} M(\omega) &:= U + \frac{1}{\omega} D \\ N(\omega) &:= \frac{1-\omega}{\omega} D - O \end{aligned}$$

oder instruktiver durch

Algorithmus 2 SOR

Für $k = 1, 2, \ldots$:
  Für $i = 1, \ldots, m$:
    $\tilde{x}_i^k := A_{i,i}^{-1} \left(b_i - \sum_{j=1}^{i-1} A_{i,j} x_j^k - \sum_{j=i+1}^{m} A_{i,j} x_j^{k-1} \right)$
    $x_i^k := \omega \tilde{x}_i^k + (1 - \omega) x_i^{k-1}$
  Ende
Ende

Die Bezeichnung „Überrelaxation“ ist eigentlich nur für $\omega > 1$ richtig, doch spricht man für alle ω von „SOR“. Für $\omega = 1$ erhält man wieder das Gauß–Seidel–Verfahren.

Satz 2.10
Für die Iterationsmatrix des SOR ist

$$\rho(B(\omega)) \geq |\omega - 1|.$$

Nach (2.7) ist also durch

$$\omega \in (0, 2).$$

eine notwendige Bedingung dafür gegeben, daß das SOR-Verfahren konvergiert.

Beweis: [29]

Für symmetrische positiv definite Matrizen ist die Bedingung auch hinreichend ([43]). Die Wahl des Relaxationsfaktors ω ist entscheidend für das Konvergenzverhalten des SOR. Eine einfache Methode zur direkten Berechnung des optimalen Wertes ω_{opt} ist nicht bekannt. In [47] findet man für Matrizen bestimmter Struktur eine explizite Darstellung von ω_{opt}, deren Wert sich aus den Iterierten des Verfahrens approximieren läßt. So entsteht eine sich selbst optimierende Version des SOR.
Es kann vorteilhaft sein, ω als Element des $\mathbb{R}^m$ oder $\mathbb{R}^n$ zu betrachten und jeden Block oder Blockeintrag von $\tilde{x}^k$ mit individuellem Faktor zu relaxieren. Dies ist zum Beispiel der Fall, wenn die Matrix der Diskretisierung einer vektorwertigen partiellen Differentialgleichung entspringt, und die einzelnen Komponenten des Lösungsvektors unterschiedliche analytische Eigenschaften besitzen.
SOR führt in jeder Iteration einen Informationstransport durch, da die bereits berechneten Komponenten der nächsten Näherungslösung zur Berechnung der folgenden Komponente verwendet werden. Am Beispiel einer oberen und einer unteren Blockdreiecksmatrix erkennt man, daß dabei die Menge der transportierten Information und daher auch die Konvergenzrate sehr von der Struktur der Matrix abhängt und umso besser wird, je mehr Blöcke unterhalb der Diagonalen liegen. Um diesen Zusammenhang auszunutzen, kann man die Bearbeitungsreihenfolge der Zeilen ändern. Man ersetzt

dafür i in Algorithmus 2 durch $\pi(i)$, wobei $\pi : \mathbb{N} \mapsto \mathbb{N}$ eine Permutation beschreibt. Diese ist also optimal, wenn

$$\forall\, 1 \leq i, j \leq n : \quad A_{i,j} \neq 0 \;\Rightarrow\; \pi(i) \geq \pi(j).$$

Bei der Anwendung von Algorithmus 2 auf *echte Blockmatrizen* ($l > 1$) erhebt sich die Frage nach der Berechnung des Produktes mit $A_{i,i}^{-1}$. Oft besitzen die Blöcke spezielle Struktur, so daß man, wie bei den direkten Verfahren, angepaßte Methoden einsetzen kann. Im allgemeinen Fall (beliebig besetzter Block) ist die Inversion des Blockes seiner LR–Zerlegung vorzuziehen, die man hier, verbunden mit jeweiligem Vorwärts–/Rückwärts–Einsetzen, ebenfalls verwenden könnte. Die zugunsten der numerischen Stabilität notwendige Pivotisierung kann in den Inversen unter geringerem Speicher– und Rechenaufwand realisiert werden. Deren höherer Konstruktionsaufwand ist vernachlässigbar, da dieser nur einmal, bei Start des Verfahrens, entsteht. Im weiteren Verlauf wird er durch die dann bei gleichem theoretischem Aufwand effizienter implementierbaren Berechnungen ausgeglichen.

2.2.2 SSOR

Eine Iteration des *SSOR (Symmetric SOR)* kombiniert einen SOR–Vorwärtsschritt mit einem nachfolgenden SOR–Rückwärtsschritt. Der Name steht nicht nur für die Vorgehensweise, sondern auch dafür, daß für symmetrisches A die *Iterationsmatrix*

$$B_{SSOR} := B_{SORr} B_{SORv}$$

(B_{SORr} und B_{SORv} sind die Iterationsmatrizen von „SOR rückwärts/vorwärts“) ebenfalls symmetrisch ist. Damit ist B_{SSOR} als Vorkonditionierung für symmetrische Probleme geeignet (dazu in Kapitel 3).
Alle Überlegungen zu SOR übertragen sich direkt auf SSOR, wenngleich die Frage der optimalen Abarbeitungsreihenfolge für die Zeilen nicht so einfach geklärt werden kann. Bei Bearbeitung in natürlicher Reihenfolge ($\pi = \mathrm{id}$) läßt sich jedoch der Aufwand des SSOR unter Verwendung eines Hilfsvektors fast auf die Hälfte reduzieren. Betrachtet man nämlich Algorithmus 2 und nennt die nach dem Vorwärtsschritt erreichte Näherungslösung $x^{k-\frac{1}{2}}$, so erkennt man, daß bei der Hintereinanderausführung jeweils $Ux^{k-\frac{1}{2}} = \{\sum_{j=1}^{i-1} A_{i,j} x_j^{k-\frac{1}{2}}\}_{1 \leq i \leq m}$ berechnet wird. Merkt man sich diesen Vektor bei der ersten Berechnung, so spart man im Rückwärtsschritt fast die Hälfte des Aufwands ein. Gleiches läßt sich für x^k vom Rückwärts– zum Vorwärtsschritt durchführen, wobei man denselben Vektor benutzen kann. Man erhält so

Algorithmus 3 SSOR

Für $i = 1, \ldots, m$:
  $y_i := \sum_{j=i+1}^{m} A_{i,j} x_j^0$
Ende
Für $k = 1, 2, \ldots$:
  Für $i = 1, \ldots, m$:
    $\tilde{y}_i := \sum_{j=1}^{i-1} A_{i,j} x_j^{k-\frac{1}{2}}$
    $\tilde{x}_i^{k-\frac{1}{2}} := A_{i,i}^{-1} (b_i - y_i - \tilde{y}_i)$
    $x_i^{k-\frac{1}{2}} := \omega \tilde{x}_i^{k-\frac{1}{2}} + (1-\omega) x_i^{k-1}$
  Ende
  Für $i = m, \ldots, 1$:
    $y_i := \sum_{j=i+1}^{m} A_{i,j} x_j^{k-\frac{1}{2}}$
    $\tilde{x}_i^{k} := A_{i,i}^{-1} (b_i - y_i - \tilde{y}_i)$
    $x_i^{k} := \omega \tilde{x}_i^{k} + (1-\omega) x_i^{k-1}$
  Ende
Ende

2.3 Petrov–Galerkin–Krylov–Verfahren

Statt die nächste Näherungslösung gemäß einer invarianten Iterationsvorschrift auf dem gesamten $\mathbb{R}^n$ zu bestimmen, wie es die stationären Verfahren tun, suchen die in diesem Abschnitt vorgestellten Verfahren für $k = 1, 2, \ldots$ auf einem k–dimensionalen affinen Unterraum eine in gewisser Hinsicht optimale Näherungslösung x_k. Spätestens für $k = n$ ist man dann am Ziel, weil die exakte Lösung in jedem Fall diese Optimalitätsforderung erfüllt. Durch die Optimierung möchte man jedoch schon für $k \ll n$ eine hinreichend gute Näherung x_k von x erreichen. Da man x nicht kennt, wird die Optimalitätsforderung an das Residuum gestellt. Diese Forderung läßt sich in dem k–dimensionalen Unterraum auch als Orthogonalitätsbedingung bezüglich k Vektoren des $\mathbb{R}^n$ bzw. bezüglich eines weiteren k–dimensionalen Unterraumes formulieren.

Definition 2.11
Seien für $1 \leq k \leq n$ durch $v_1, \ldots, v_k$ und $w_1, \ldots, w_k$ jeweils linear unabhängige Systeme gegeben, so daß mit

$$\begin{array}{ll} V_k := (v_1, \ldots, v_k), & W_k := (w_1, \ldots, w_k) \qquad \in \mathbb{R}^{n \times k} \\ K_k := \operatorname{span}(v_1, \ldots, v_k), & L_k := \operatorname{span}(w_1, \ldots, w_k) \end{array}$$

gilt:

$$\det\left(W_k^T A V_k\right) \neq 0. \tag{2.9}$$

Das durch K_k und L_k bestimmte Projektionsverfahren *oder* Petrov–Galerkin–Verfahren *hat dann die Form:*

Finde für $1 \leq k \leq n$ ein $x_k \in x_0 + K_k$, so daß

$$r_k \perp L_k \qquad \text{(Petrov–Galerkin–Bedingung)},$$

bzw. anders formuliert:

Finde für $1 \leq k \leq n$ ein $z_k \in K_k$, so daß

$$r_0 - Az_k \perp L_k, \tag{2.10}$$

und setze

$$x_k := x_0 + z_k.$$

Die Orthogonalitätsbeziehung „$\perp$“ ist dabei über das euklidische Skalarprodukt definiert. Wegen

$$r_0 - Az_k = b - Ax_0 - Az_k = b - Ax_k = r_k$$

sind beide Formulierungen äquivalent.
Solch ein Verfahren bestimmt also eine neue Näherungslösung in dem k–dimensionalen affinen Unterraum $x_0 + K_k$, indem die Orthogonalität des zugehörigen Residuums bezüglich dem k–dimensionalen Unterraum L_k verlangt wird. Die zweite, etwas umständlicher erscheinende Formulierung hat den Vorteil, daß man so z_k einfacher in einem echten statt einem affinen Unterraum suchen kann.
Entsprechend bezeichnet man die Basisvektoren $v_1, \ldots, v_k$ von K_k als *Suchrichtungen.*

Durch die Wahl $L_k := K_k$ erhält man *orthogonale Projektionsverfahren* (da dann $r_k \perp K_k$, was man die *Galerkin–Bedingung* nennt), für $L_k \neq K_k$ entstehen *schiefe* oder *nichtorthogonale Projektionsverfahren.*
Auch die stationären Verfahren sind Projektionsverfahren. So ist eine Gauß–Seidel–Iteration eine Sequenz von n orthogonalen Projektionsschritten bezüglich der invarianten eindimensionalen Unterräume $L^i = K^i = \text{span}\{e_i\}$, $i = 1, \ldots, n$. Die in diesem Abschnitt vorgestellten Projektionsverfahren verwenden jedoch Unterräume, die von den Iterierten abhängen, und sind somit *instationär*, also nicht darstellbar gemäß Definition 2.7. Da sie, wie sich zeigen wird, die Koeffizientenmatrix nur in Matrix–Vektor–Produkten mit A bzw. A^T ansprechen, sind sie einfacher und von der Speicherungsweise von A unabhängig implementierbar.
Unter den geforderten Voraussetzungen ist ein Projektionsverfahren wohldefiniert, da

$$y_k := \left(W_k^T A V_k\right)^{-1} W_k^T r_0$$

wegen (2.9) eindeutig bestimmt und so (2.10) wegen

$$\begin{aligned} W_k^T (r_0 - AV_k y_k) &= W_k^T r_0 - \left(W_k^T AV_k\right) y_k \\ &= 0 \\ \Leftrightarrow \qquad r_0 - AV_k y_k &\perp L_k \end{aligned}$$

mit $z_k := V_k y_k \in K_k$ eindeutig lösbar ist.
Nach spätestens n Schritten wird dann die exakte Lösung gefunden, da für $k = n$

$$r_n \perp L_n = \mathbb{R}^n \Leftrightarrow r_n = 0 \Leftrightarrow x_n = x.$$

Wichtig ist nun die Tatsache, daß (2.9) in bestimmten Fällen immer gewährleistet ist:

Satz 2.12
Für beliebige Basen $(v_1, \ldots, v_k)$ von K_k und $(w_1, \ldots, w_k)$ von L_k gilt (2.9), falls

$$A \text{ positiv definit ist und } L_k = K_k$$

oder

$$A \text{ nichtsingulär ist und } L_k = AK_k$$

Beweis: [47]

Nun zur Wahl von K_k:

Definition 2.13
Für $v \in \mathbb{R}^n$, $B \in \mathbb{R}^{n \times n}$, $k \in \mathbb{N}$ heißt

$$\mathcal{K}_k(v, B) := \operatorname{span}\{v, Bv, B^2 v, \ldots, B^{(k-1)} v\}$$

der k–te Krylov–Unterraum *zu v und B.*

Definition 2.14
Ein Krylov–Unterraum–Verfahren *ist ein Petrov–Galerkin–Verfahren, das $K_k := \mathcal{K}_k(r_0, A)$ wählt.*

Im Folgenden werden ausschließlich Krylov–Unterraum–Verfahren betrachtet. Ihre allgemeine Formulierung ist der *PGK*–Algorithmus *(Petrov–Galerkin–Krylov)*:

Algorithmus 4 PGK

$r_0 := b - Ax_0$
$v_1 := w_1 := r_0/\|r_0\|_2$
Krylov–Unterraum–Konstruktion:
    Für $k = 1, 2, \ldots$:
        Falls $K_{k+1}(r_0, A) = K_k(r_0, A) \longrightarrow$ Schleife beenden
        Bestimme $v_{k+1} \in K_{k+1}(r_0, A)$ und $w_{k+1} \in L_{k+1}(r_0, A)$, so daß
            $\det\left(W_{k+1}^T A V_{k+1}\right) \neq 0$
        Falls dies nicht möglich ist $\longrightarrow$ Stop
    Ende
Lösung des Petrov–Galerkin–Problems:
    $V_k := (v_1, \ldots, v_k)$
    $W_k := (w_1, \ldots, w_k)$
    $y_k := (W_k^T A V_k)^{-1} W_k^T r_{k-1}$
    $x_k := x_{k-1} + V_k y_k$
    $r_k := b - Ax_k$

Der Index k im Lösungsteil ist der Laufwert, für den die Konstruktion der Basis der Krylov–Unterräume wegen $K_{k+1}(r_0, A) = K_k(r_0, A)$ abgeschlossen worden ist.
Die Aufteilung des Algorithmus in Basisbestimmung und Lösungsvorgang bleibt jedoch in den Lösungsverfahren im allgemeinen nicht erhalten. Viele verbinden Konstruktions– und Lösungsprozeß. Entsprechend wird zur Bestimmung von v_{k+1} entweder Av_k oder r_k gemäß (2.9) im allgemeinen Sinn orthogonalisiert. Die Bestimmung von w_{k+1} hängt von der Definition von L_{k+1} ab.
Bei den Verfahren, die tatsächlich das Petrov–Galerkin–Problem erst nach der vollständigen Entfaltung der Krylov–Unterräume lösen, läßt sich in der Regel $\|r_k\|_2$ während deren Konstruktion berechnen, so daß auch ohne Kenntnis von x_k das Abbruchkriterium überprüft werden kann.
Je nach Wahl von L_k entstehen nun mathematisch verschiedene Verfahren, d.h. Verfahren, die unterschiedliche Folgen von Näherungslösungen erzeugen. Man unterscheidet (die Motivation zur jeweiligen Wahl von L_k wird erläutert werden):

- *Galerkin–/orthogonal–residual–Verfahren* ($L_k := K_k$)
- *least–squares–/minimal–residual–Verfahren* ($L_k := AK_k$)
- *Lanczos–Verfahren* ($L_k := \mathcal{K}_k(r_0, A^T)$)

Jedes dieser Verfahren kann dann wiederum auf verschiedene Weisen realisiert werden. Gemeinsames Ziel aller Verfahren ist, die Krylov–Basen so zu wählen, daß $W_k^T A V_k$ einfache Gestalt besitzt und somit das Petrov–Galerkin–Problem leicht lösbar ist.
Galerkin– und least–squares–Verfahren haben gemeinsam, daß sie $w_1, \ldots, w_{k+1}$

nicht explizit speichern müssen, da diese Werte durch $Av_1, \ldots, Av_{k+1}$ gegeben sind. Deren Krylov–Unterraum–Konstruktion läßt sich daher durch die *Arnoldi–Orthonormalisierung* realisieren, die im nächsten Abschnitt beschrieben wird. In den Abschnitten 2.5 und 2.6 folgen die daraus abgeleiteten *Galerkin–* und *least–squares–Verfahren*. Danach wird die *Lanczos–Biorthogonalisierung* dargestellt, die entsprechend zur Arnoldi–Orthonormalisierung die Krylov–Basis für die in 2.8 aufgeführten *Lanczos–Verfahren* bestimmt. In Abschnitt 2.9 wird sich schließlich zeigen, wie im Fall symmetrischer Matrizen nach beiden Ansätzen dieselben Basen entstehen und mit dem *CG–Verfahren* für positiv definite symmetrische Matrizen, dem gemeinsamen Kern aller vorgestellten Methoden, auch wieder nichtsymmetrische Probleme gelöst werden können.

2.4 Arnoldi–Orthonormalisierung

Die *Arnoldi–Orthonormalisierung* (vorgestellt 1951 von Arnoldi in [2]) ist eine modifizierte Gram–Schmidt–Orthonormalisierung, bei der die Ausgangsvektoren $\tilde{v}_2, \ldots, \tilde{v}_k$ nicht gegeben sind, sondern als die Bilder Av_{k-1} des jeweils vorher ermittelten Basisvektors durch das Verfahren erzeugt werden. Die Modifikation besteht darin, daß in diesem Algorithmus $\tilde{v}_k$ zugunsten numerischer Stabilität sukzessive zu $v_1, \ldots, v_{k-1}$ orthogonalisiert wird.

Algorithmus 5 Arnoldi–Orthonormalisierung

Sei $\tilde{v}_1 \in \mathbb{R}^n$ gegeben.
$v_1 := \tilde{v}_1 / \|\tilde{v}_1\|_2$
Für $k = 1, 2, \ldots$:
    $\tilde{v}_k := Av_k$
    Für $i = 1, \ldots, k$:
        $h_{i,k} := (\tilde{v}_k, v_i)$
        $\tilde{v}_k := \tilde{v}_k - h_{i,k} v_i$
    Ende
    $h_{k+1,k} := \|\tilde{v}_k\|_2$
    Wenn $h_{k+1,k} = 0 \longrightarrow$ Stop
    $v_{k+1} := \tilde{v}_k / h_{k+1,k}$
Ende

Sei $\tilde{k}$ der Wert k, mit dem der Arnoldi–Algorithmus gestoppt hat. Induktiv beweist man:

Satz 2.15
Das Arnoldi–Orthonormalisierungsverfahren erzeugt für $1 \leq k \leq \tilde{k}$ *eine Orthonormal-*

basis $\{v_1, \ldots, v_k\}$ *von* $\mathcal{K}_k(\tilde{v}_1, A)$, *also*

$$(v_i, v_j) = \delta_{i,j} \quad \textit{für} \quad 1 \leq i, j \leq k.$$

Man spricht daher von einem Verfahren der *orthogonalen Richtungen*. Die entscheidende Eigenschaft dieses Verfahrens ist, daß es *unbedingt stabil* ist, also in jedem Fall das gewünschte Resultat liefert. Es gilt nämlich

Satz 2.16
Für die durch die Arnoldi–Orthonormalisierung erzeugten Größen sind folgende Aussagen äquivalent:

$$\begin{aligned} & h_{k+1,k} = 0 \\ \Leftrightarrow \quad & K_1 \subset K_2 \subset \ldots \subset K_k = K_{k+1} = \ldots = K_n \end{aligned}$$

Beweis: [32]

Das Verfahren bricht genau dann ab, wenn eine Orthonormalbasis des Krylov–Unterraumes $\mathcal{K}_n(\tilde{v}_1, A)$ gefunden wurde. Somit ist das Verfahren mathematisch stabil. Mit der durch Algorithmus (5) produzierten oberen Hessenbergmatrix (der sogenannten *Arnoldi–Matrix*)

$$(H_k)_{i,j} := \left\{ \begin{array}{ll} h_{i,j} & i \leq j+1 \\ 0 & i > j+1 \end{array} \right\} \quad \text{mit} \quad 1 \leq i, j \leq k$$

läßt sich der k–te Schritt des Verfahrens schreiben als

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T,$$

oder mit

$$\tilde{H}_k := \left(\begin{array}{c} H_k \\ h_{k+1,k} e_k^T \end{array} \right) \in \mathbb{R}^{(k+1)\times k}$$

als

$$AV_k = V_{k+1} \tilde{H}_k. \tag{2.11}$$

Zu beachten ist, daß $\tilde{H}_k$ vollen Rang hat, da die unteren k Zeilen von $\tilde{H}_k$ eine obere Dreiecksmatrix bilden.
Durch Multiplikation von links mit V_k^T folgt

$$H_k = V_k^T A V_k. \tag{2.12}$$

Da zur Durchführung des k–ten Schrittes des Arnoldi–Verfahrens die gesamte Folge $\{v_1, \ldots, v_{k-1}\}$ benötigt wird, also gespeichert sein muß, hat es so nur theoretischen Wert. Die Lösungsverfahren, die es benutzen, modifiziert man durch folgende Verkürzungen, durch die sich allerdings die numerischen Eigenschaften der Verfahren verschlechtern können:

- *Restart:* Nach m Iterationen wird der Arnoldi–Prozeß abgebrochen und dann mit $x_0 := x_m$ das Verfahren neu gestartet. m beschreibt die maximale Dimension der Krylov–Unterräume.
- *Abschneidung (Truncation):* Im Arnoldi–Verfahren wird $\tilde{v}_k$ nur zu den letzten $t \geq 0$ Basisvektoren orthonormalisiert. Die Anzahl der oberen Nebendiagonalen von H_k ist also auf $t-1$ beschränkt.
- *Einschränkung:* In der Petrov–Galerkin–Bedingung werden nur die letzten $s > 0$ Basisvektoren berücksichtigt. Lediglich $s-1$ der oberen Nebendiagonalen von H_k gehen bei der Lösung ein.

Sind Rundungsfehler ein sehr kritischer Punkt, empfiehlt es sich, im Arnoldi–Algorithmus nicht die Gram–Schmidt–Orthogonalisierung, sondern Householder–Transformationen zur Konstruktion von H_k zu verwenden. Dadurch erreicht man höhere numerische Stabilität, allerdings unter etwa doppelt so hohem Rechenaufwand. Eine ausführliche Darstellung findet man in [47].

2.5 Galerkin–Verfahren

Galerkin–Verfahren benutzen als Petrov–Galerkin–Bedingung die *Galerkin–Bedingung*

$$r_k \perp L_k = K_k = \mathcal{K}(r_0, A),$$

wodurch sich ihr Name wie auch die englische Bezeichnung *orthogonal residual methods* erklärt.

Satz 2.17
Sei A symmetrisch und positiv definit, so daß durch $\|\cdot\|_A \equiv (A\cdot,\cdot)^{\frac{1}{2}}$ und entsprechend $\|\cdot\|_{A^{-1}}$ Normen auf $\mathbb{R}^n$ erklärt sind. Dann ist x_k genau dann eine durch ein Galerkin–Verfahren erzeugte Näherungslösung, wenn

$$\|r_k\|_{A^{-1}} (= \|b - Ax_k\|_{A^{-1}}) = \min_{\tilde{x} \in x_0 + K_k} \|b - A\tilde{x}\|_{A^{-1}}$$

und dies wiederum ist genau dann der Fall, wenn

$$\|x - x_k\|_A = \min_{\tilde{x} \in x_0 + K_k} \|x - \tilde{x}\|_A$$

Beweis: [48]

Im Fall nichtsymmetrischer Matrizen existieren keine Optimalitätsaussagen für Galerkin–Verfahren, so daß sie möglicherweise unregelmäßig oder gar nicht konvergieren. Die Orthogonalitätsbedingung (2.9) ist nicht unbedingt erfüllbar, so daß auch die eindeutige Lösbarkeit der Petrov–Galerkin–Bedingung nicht gewährleistet ist.

2.5.1 FOM

Die *Full Orthogonalization Method* (*FOM*) ist die unmittelbare Anwendung des Arnoldi–Verfahrens mit $\tilde{v_1} := r_0$ als Galerkin–Methode. Wegen der Orthonormalität von $v_1, \ldots, v_k$ ist

$$V_k^T r_0 = V_k^T \|r_0\|_2 v_1 = \|r_0\|_2 e_1,$$

und damit wegen $W_k = V_k$ und (2.12)

$$y_k := \left(W_k^T A V_k\right)^{-1} W_k^T r_0 = \left(V_k^T A V_k\right)^{-1} V_k^T r_0 = H_k^{-1} \|r_0\|_2 e_1.$$

Diese Berechnung führt man durch Anwendung von Givens–Rotationen (siehe [23]) auf das System

$$H_k y_k = \|r_0\|_2 e_1$$

aus, wodurch man ein durch Rückwärtseinsetzen einfach zu lösendes oberes Dreiecks-system

$$R_k y_k = g_k$$

erhält, falls H_k nichtsingulär ist. Diese Umformungen können auch während der Orthonormalisierung durch fortlaufende Updates von g_k und R_k durchgeführt werden (in [47] wird dieses Vorgehen erläutert). Dies ist möglich, da die Spalten $h_{*,1}, \ldots, h_{*,k-1}$ von H_k bei der Orthonormalisierung von v_k nicht mehr auftreten.

Algorithmus 6 FOM

$r_0 := b - Ax_0$
$v_1 := r_0 / \|r_0\|_2$
Für $k = 1, 2, \ldots$:
    $\tilde{v}_k := Av_k$
    Für $i = 1, \ldots, k$:
        $h_{i,k} := (v_i, \tilde{v}_k)$
        $\tilde{v}_k := \tilde{v}_k - h_{i,k} v_i$
    Ende
    $h_{k+1,k} := \|\tilde{v}_k\|_2$
    Falls $h_{k+1,k} = 0 \longrightarrow$ Schleife beenden
    $v_{k+1} := \tilde{v}_k / h_{k+1,k}$
    Falls $k > 1 \longrightarrow$ Update von R_k und g_k durch die bisherigen Givens–Rotationen und die zu $h_{k+1,k}$
Ende
$y_k := R_k^{-1} g_k$
$x_k := x_0 + V_k y_k$

Der erfolgreiche Ablauf des Algorithmus ist gewährleistet, sofern (2.9) erfüllt ist, bzw. H_k nichtsingulär ist.
Folgende Varianten vermeiden die nötige Abspeicherung aller $v_1, \ldots, v_k$:

- **FOM(m)** führt einen Restart nach m Schritten durch, d.h. die äußere Schleife wird nun mit $k = 1, \ldots, m$ durchlaufen, dann das Verfahren mit $x_0 := x_m$ neu gestartet.

- **IOM(t)** *(Incomplete Orthogonalization Method)* entsteht durch Abschneidung. Die innere Schleife des FOM (die Orthonormalisierung) wird nur für $i = \max\{1, k-t+1\}, \ldots, k$ durchlaufen. Damit werden dort nur die letzten t Vektoren v_i benötigt. Da jedoch zur Berechnung von x_k nach wie vor $v_1, \ldots, v_k$ bereitstehen müssen, ist dieses Verfahren nur theoretische Grundlage der folgenden Methode.

- **DIOM(t)** *(Direct IOM)* nutzt aus, daß die durch IOM(t) erzeugte Matrix H_k nur $t-1$ obere Nebendiagonalen besitzt, also eine Bandmatrix ist. Die LR–Zerlegung dieser Matrix kann wegen ihrer Hessenberggestalt durch einfache Updates nach jedem Durchlauf der inneren Schleife berechnet werden. Wegen der Bandform der Matrix sind dafür nur t Vektoren zu speichern. Mit diesen Updates kann auch die jeweilige Näherungslösung x_j günstig innerhalb der äußeren FOM–Schleife berechnet werden, so daß der Schwachpunkt des IOM(t) behoben ist. In [47] wird dieses Verfahren im Detail beschrieben.
  Unter Verwendung eines zusätzlichen Vektors kann dabei zugunsten der mathematischen und numerischen Stabilität bei der LR–Zerlegung eine Pivotisierung durchgeführt werden (siehe dazu [46]).

IOM(t) und DIOM(t) sind keine orthogonalen Projektionsmethoden mehr, der sie definierende Raum L_k kann jedoch leicht angegeben werden. Die Darstellung findet man in [47], wie auch den Beweis des folgenden Satzes:

Satz 2.18
Die Residuennorm der k–ten durch FOM, FOM(m), IOM(t) oder DIOM(t) bestimmten Iterierten ist gegeben durch

$$\|b - Ax_k\|_2 = h_{k+1,k}|e_k^T y_k|.$$

Dank dieser Aussage kann bei allen Verfahren schon in der äußeren Schleife, ohne Kenntnis der entsprechenden aktuellen Näherung x_j, eine Residuenkontrolle durchgeführt werden. Auch im DIOM(t) entsteht dadurch ein Vorteil, da die Berechnung des Residuums zu x_j vermieden wird.

2.6 Least–squares–Verfahren

Satz 2.19
x_k ist genau dann eine durch ein least–squares–Verfahren erzeugte Näherungslösung, wenn

$$\|r_k\|_2 \,(= \|b - Ax_k\|_2) = \min_{\tilde{x} \in x_0 + K_k} \|b - A\tilde{x}\|_2$$

Beweis: [48]

Die Residuennorm fällt monoton mit wachsender Dimension der Krylov–Unterräume. Aus dieser Eigenschaft stammt der Name *minimal–residual–Verfahren* für diese Methoden. Auch wird ihr Lösungsprozeß anhand der Minimierungsforderung statt der Petrov–Galerkin–Bedingung konstruiert. Das führt auf ein least–squares–Problem, was den Namen der least–squares–Verfahren erklärt. Man kann die Minimierung auch als Motivation für die least–squares–Verfahren deuten und daraus wegen der Äquivalenz in Satz 2.19 die Wahl $L_k := AK_k = \mathcal{K}(Ar_0, A)$ ableiten.

2.6.1 GMRES

GMRES (*General Minimized RESidual*, entwickelt in [49]) ist das least–squares–Pendant zu FOM. Die Konstruktion der Krylov–Basis ($w_1, \ldots, w_k$ sind als $Av_1, \ldots, Av_k$ gegeben) erfolgt durch das Arnoldi–Verfahren; die Lösung des Petrov–Galerkin–Problemes wird, wie oben erwähnt, aus der Minimierungsforderung abgeleitet:

$$\begin{aligned}
\|r_k\|_2 &= \|b - Ax_k\|_2 \\
&= \|b - Ax_0 - Az_k\|_2 \\
&= \|r_0 - AV_k y_k\|_2 \\
&\overset{(2.11)}{=} \left\| V_{k+1} \left(\|r_0\|_2 e_1 - \tilde{H}_k y_k \right) \right\|_2 \\
&= \left\| \|r_0\|_2 e_1 - \tilde{H}_k y_k \right\|_2 ,
\end{aligned}$$

wobei sich die letzte Gleichung aus der Orthonormalität der Spalten von V_{k+1} erklärt. Die 2–Norm des Residuums wird also minimal für $x_k := x_0 + V_k y_k$ mit

$$y_k := \arg\min_{y \in \mathbb{R}^k} \left\| \|r_0\|_2 e_1 - \tilde{H}_k y \right\|_2 .$$

Dieses least–squares–Problem ist eindeutig lösbar, da $\tilde{H}_k$ vollen Spaltenrang hat. Weil $\tilde{H}_k$ Hessenbergmatrix ist, läßt sich diese Lösung leicht realisieren. Man bestimmt dafür,

ähnlich wie im FOM, durch Givens–Rotationen eine Zerlegung von $\tilde{H}_k = Q_k \tilde{R}_k$ in eine orthogonale Matrix $Q_k \in \mathbb{R}^{(k+1)\times(k+1)}$ und eine Matrix $\tilde{R}_k \in \mathbb{R}^{(k+1)\times k}$ der Form

$$\tilde{R}_k = \begin{pmatrix} R_k \\ 0 \end{pmatrix}$$

mit einer oberen Dreiecksmatrix $R_k \in \mathbb{R}^{k\times k}$. Diese Zerlegung führt man wieder durch laufende Updates der Dreiecksmatrix R_k und die entsprechenden Umformungen auf $\|r_0\|e_1$ innerhalb der äußeren Schleife durch. Das Problem transformiert man so wegen der Orthogonalität von Q_k in die durch Rückwärtseinsetzen lösbare Form

$$\begin{aligned} \min_{y\in\mathbb{R}^k} \left\| \|r_0\|_2 e_1 - \tilde{H}_k y \right\|_2 &= \min_{y\in\mathbb{R}^k} \left\| Q_k(\|r_0\|_2 e_1 - \tilde{H}_k y) \right\|_2 \\ &= \min_{y\in\mathbb{R}^k} \Big\| \underbrace{Q_k \|r_0\|_2 e_1}_{=:\tilde{g}_k \in \mathbb{R}^{k+1}} - \tilde{R}_k y \Big\|_2 . \end{aligned}$$

Algorithmus 7 GMRES

$r_0 := b - Ax_0$
$v_1 := r_0/\|r_0\|_2$
Für $k = 1, 2, \ldots$:
    $\tilde{v}_k := Av_k$
    Für $i = 1, \ldots, k$:
        $h_{i,k} := (v_i, \tilde{v}_k)$
        $\tilde{v}_k := \tilde{v}_k - h_{i,k} v_i$
    Ende
    $h_{k+1,k} := \|\tilde{v}_k\|_2$
    Falls $h_{k+1,k} = 0 \longrightarrow$ Schleife beenden
    $v_{k+1} := \tilde{v}_k / h_{k+1,k}$
    Update von $\tilde{R}_k$ und $\tilde{g}_k$ durch die bisherigen Givens–Rotationen und die zu $h_{k+1,k}$
Ende
$y_k := R_k^{-1} g_k$
$x_k := x_0 + V_k y_k$

Höhere numerische Stabilität läßt sich wieder erreichen, indem man die Gram–Schmidt–Orthonormalisierung des Arnoldi–Verfahrens durch Householdertransformationen ersetzt. Wird dies in Algorithmus 7 naiv durchgeführt, fallen jedoch doppelt so viele zu speichernde Vektoren an. Denn neben $v_1, \ldots, v_k$ werden weitere k Householder–Vektoren benötigt. Um dies zu umgehen, ist eine Umformulierung des Verfahrens nötig, nach der x_k aus den Householder–Vektoren ermittelt werden kann. Diese Methode ist in [47] ausgeführt.
Wie bei den im letzten Abschnitt vorgestellten Verfahren ist die Kontrolle des Abbruchkriteriums nicht an die Kenntnis der aktuellen Näherungslösung gebunden:

Satz 2.20
Die Residuennorm der k-ten Iterierten des GMRES–Verfahrens ist gegeben durch

$$\|r_k\|_2 = |e_{k+1}^T \tilde{g}_k|$$

Beweis: [49]

Man kann die Konvergenz also wieder schon in der äußeren Schleife überprüfen und braucht erst dann das least–square–Problem zu lösen und x_k zu berechnen, wenn die Residuennorm klein genug ist.
Aus der Eigenschaft des Arnoldi–Algorithmus, genau dann zu terminieren, wenn ein die exakte Lösung enthaltender Raum aufgespannt wird, verbunden mit der Minimierungseigenschaft des GMRES, erhält man die folgende Stabilitätsaussage:

Satz 2.21
Das GMRES–Verfahren endet genau dann, wenn die exakte Lösung x gefunden wurde, was wiederum nach höchstens n Schritten der Fall ist.

Die Konvergenz des GMRES läßt sich theoretisch abschätzen durch

Satz 2.22
Sei A diagonalisierbar und X eine Matrix aus Eigenvektoren von A. Sei $\mathcal{P}_k$ die Menge der Polynome vom Höchstgrad k. Dann gilt:

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \varepsilon_k \operatorname{cond}_2(X)$$

mit

$$\varepsilon_k := \min_{\psi \in \mathcal{P}_k : \psi(0)=1} \left(\max_{\lambda \in \lambda(A)} |\psi(\lambda)| \right).$$

Beweis: [49]

Allerdings ist dieser Satz in der Praxis selten anwendbar bzw. meist wegen $\operatorname{cond}_2(X) \gg 1$ nicht sehr aussagekräftig.
GMRES und FOM unterscheiden sich lediglich in der Ableitung von R_k und g_k aus $\tilde{H}_k$ bzw. H_k. Dies wiederum kommt erst bei der k–ten Givens–Rotation zur Elimination von $h_{k+1,k}$ zum tragen, die allein vom GMRES durchgeführt wird. Der Unterschied liegt also nur in den Komponenten $(R_k)_{k,k}$ und $(g_k)_k$. Daher besteht folgender Zusammenhang zwischen den Residuennormen der (nicht explizit berechneten) Iterierten x_k:

Satz 2.23
Seien r_k^{FOM} und r_k^{GMRES} die Residuen der (nicht explizit bekannten) Iterierten x_k des FOM bzw. GMRES. Sei c_k der Cosinusanteil der k–ten Givens–Rotation des GMRES

zur Elimination von $h_{k+1,k}$. Dann ist

$$\|r_k^F\|_2 = \frac{1}{c_k}\|r_k^G\|_2.$$

Beweis: [47]

Bezüglich der praktischen Umsetzung gilt für GMRES dasselbe wie für FOM:

- **GMRES(m)** beschränkt die Anzahl der benötigten Vektoren durch Restart nach m Schritten. Wegen der dadurch begrenzten Dimension der Krylov–Unterräume gilt jedoch Satz 2.21 nicht mehr, die Konvergenz ist nicht gesichert. Nur für positiv definite Matrizen konvergiert das Verfahren nach [47] für alle $m \geq 1$. Andernfalls kann es stagnieren (siehe dazu [27], [54]). Je größer m gewählt wird, umso stabiler ist GMRES(m), gleichzeitig wachsen aber Speicher– und Rechenaufwand stark an. Wo der Kompromiß zu setzen ist, hängt sehr vom Problem ab. Gewöhnlich wählt man $4 \leq m \leq 20$, doch kann es auch nötig sein, den vorhandenen Speicherplatz auszuschöpfen, um Konvergenz zu erreichen.
- **QGMRES(t)** *(Quasi–GMRES)* ist das Äquivalent zu IOM(t). Durch Abschneiden der inneren Schleife wird die Orthonormalisierung auf die letzten t Basisvektoren reduziert.
- **DQGMRES(t)** *(Direct QGMRES)* ist, entsprechend zu DIOM(t), die Umformung des QGMRES(t) durch eine LR–Zerlegung, die auch den absoluten Speicherbedarf des Verfahrens auf t Vektoren beschränkt. Die Herleitung ist ähnlich zu der des DIOM(t) und kann in [47] gefunden werden.

Wegen der unvollständigen Orthogonalisierung führen die beiden letzten Verfahren keine echte Minimierung der Residuennorm durch. Abschätzungen dieser Norm sind in [49] gegeben.

2.6.2 GCR, ORTHOMIN, ORTHODIR

Während GMRES das Arnoldi–Verfahren verwendet, um die Orthonormalität der Basis $\{v_1, \ldots, v_k\}$ von K_k durchzusetzen, fordern eine Reihe von Verfahren stattdessen die A^TA–Orthogonalisierung

$$(Av_i, Av_j) = 0 \quad \text{für } i \neq j. \tag{2.13}$$

Der Vorteil dieses Ansatzes ist, daß man dann durch einfache Updates der Art

$$x_k := x_{k-1} + \frac{(r_{k-1}, Av_{k-1})}{(Av_{k-1}, Av_{k-1})} v_{k-1}$$

die Näherungslösungen $x_k \in \mathcal{K}_k$ erhält, die die Residuennorm minimieren (siehe [47], auch für eine Darstellung der Verfahren). Die jeweiligen Residuen lassen sich dabei durch entsprechende Updates berechnen.
Um die Basis gemäß (2.13) zu bestimmen, ist jedoch gegenüber GMRES doppelter Speicheraufwand nötig, da sowohl $Av_1, \ldots, Av_k$ für die Orthogonalisierung als auch $v_1, \ldots, v_k$ für die Berechnung von v_{k+1} benötigt werden. Ebenfalls erhöht sich der Rechenaufwand pro Iteration, bei mathematisch gleichen Näherungslösungen. Deshalb sollen diese Verfahren hier nur eingeordnet werden:

- **GCR** (*Generalized Conjugate Residual*, vorgestellt in [11], ausgearbeitet in [12]) bestimmt den nächsten Basisvektor v_{k+1}, indem r_k nach (2.13) A^TA–*orthogonalisiert* wird. Dieses Verfahren ist aus Speicherplatzgründen wieder so nicht durchführbar.
- **GCR(m)** ist die Restart–Variante des GCR.
- **ORTHOMIN(t)** (*ORTHOgonal MINimum*, entwickelt in [55]) entsteht durch Truncation des GCR. Anders als bei IOM(t) und QGMRES(t) wird hierdurch eine echte Reduzierung des Speicheraufwandes erreicht, da die Vektoren Av_k lediglich in der Orthogonalisierung auftreten und so tatsächlich nur eine beschränkte Anzahl davon benötigt wird.
- **ORTHODIR** (*ORTHOgonal DIRection*, entwickelt in [28]) A^TA–orthogonalisiert Av_k. Auch hierzu gibt es die Restart– und Truncation–Version.

2.7 Lanczos–Biorthogonalisierung

Das Problem der bisher vorgestellten Verfahren ist die nötige Speicherung von $v_1, \ldots, v_k$ und die entsprechend lange Rekursion zur Berechnung der nächsten Näherungslösung aufgrund der Hessenberg–Gestalt von H_k. Statt die Orthogonalisierung durch ein allgemeines Hessenbergverfahren wie den Arnoldi–Algorithmus durchzuführen, verwenden deshalb viele Verfahren die *Lanczos–Biorthogonalisierung*. Deren Motivation ist der Wunsch, L_k und seine Basis $w_1, \ldots, w_k$ so zu bestimmen, daß $H_k = W_k^T A V_k$ eine Tridiagonalmatrix (die sogenannte *Lanczos–Matrix*) ist:

$$H_k = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \delta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \delta_k & \alpha_k \end{pmatrix}$$

Die Einträge dieser Matrix lassen sich dann, statt durch Orthonormalisierung wie beim Arnoldi–Algorithmus, direkt durch Koeffizientenvergleich berechnen (Herleitung

in [30]). Dafür und auch für die Konstruktion der Krylov–Basisvektoren v_{k+1} und w_{k+1} benötigt man jeweils nur die letzten drei Basisvektoren, so daß das Verfahren konstanten und dabei relativ geringen Speicheraufwand hat.
Das so formulierte Ziel wird durch die Wahl $L_k := \mathcal{K}_k\left(r_0, A^T\right)$ erreicht, wie im folgenden gezeigt wird.

Algorithmus 8 Lanczos–Biorthogonalisierung

Seien $\tilde{v}_1$ und $\tilde{w}_1$ mit $\|\tilde{v}_1\|_2, \|\tilde{w}_1\|_2 \neq 0$ gegeben.
$\beta_1 := \delta_1 := 0$
$w_0 := v_0 := 0$
$v_1 := \tilde{v}_1 / \|\tilde{v}_1\|_2$
$w_1 := \tilde{w}_1 / \|\tilde{w}_1\|_2$
Für $k = 1, 2, \ldots$:
$\alpha_k := (Av_k, w_k)$
$\tilde{v}_{k+1} := Av_k - \alpha_k v_k - \beta_k v_{k-1}$
$\tilde{w}_{k+1} := A^T w_k - \alpha_k w_k - \delta_k w_{k-1}$
$\gamma_{k+1} := (\tilde{v}_{k+1}, \tilde{w}_{k+1})$
Falls $\gamma_{k+1} = 0 \longrightarrow \beta_{k+1} := \delta_{k+1} := 0$, Stop
Wähle β_{k+1} und δ_{k+1}, so daß $\beta_{k+1}\delta_{k+1} = \gamma_{k+1}$
$v_{k+1} := \tilde{v}_{k+1} / \delta_{k+1}$
$w_{k+1} := \tilde{w}_{k+1} / \beta_{k+1}$
Ende

Eine übliche Wahl für $\beta_{k+1}, \delta_{k+1}$ ist

$$\begin{aligned} \beta_{k+1} &:= \sqrt{|\gamma_{k+1}|}, \\ \delta_{k+1} &:= \operatorname{sign}(\gamma_{k+1})\, \beta_{k+1}, \end{aligned}$$

wodurch man $\beta_{k+1} = \pm\delta_{k+1}$ erhält.

Satz 2.24
Das Lanczos–Biorthogonalisierungsverfahren erzeugt bei erfolgreichem Verlauf zueinander biorthogonale *Basen* $\{v_0, \ldots, v_k\}$ *von* $\mathcal{K}_k(v_0, A)$ *und* $\{v_0, \ldots, v_k\}$ *von* $\mathcal{K}_k(v_0, A^T)$, *d.h.*

$$W_k^T V_k = I.$$

Satz 2.25
Für die durch die Lanczos–Biorthogonalisierung erzeugten Matrizen gilt:

$$\begin{aligned} H_k &= W_k^T A V_k \\ AV_k &= V_k H_k + \delta_{k+1} v_{k+1} e_k^T \\ A^T W_k &= W_k H_k^T + \beta_{k+1} w_{k+1} e_k^T \end{aligned}$$

Ferner haben V_K und W_k maximalen Rang.

Beweis: [47]

Aus diesen Gleichungen kann man, wie oben angesprochen, durch Koeffizientenvergleich die Rekursionsformeln zur Berechnung von v_{k+1} und w_{k+1} aufstellen. Weiterhin folgt daraus, daß $L_k = \mathcal{K}_k\left(r_0, A^T\right)$ ist.
Das Verfahren bricht genau dann ab, wenn

$$(\tilde{v}_{k+1}, \tilde{w}_{k+1}) = 0 \tag{2.14}$$

eintritt. Da H_k Tridiagonalmatrix ist, entspricht dies genau $\det(H_k) = 0$, also dem Fall, daß (2.9) nicht erfüllt werden kann. Es gibt genau zwei mögliche Ursachen für den Abbruch:

- $\tilde{v}_{k+1} = 0$ oder $\tilde{w}_{k+1} = 0$: normaler Abbruch (*Lucky Breakdown*), sobald $\mathcal{K}_k(r_0, A)$ durch $v_1, \ldots, v_k$ oder $\mathcal{K}_k\left(r_0, A^T\right)$ durch $w_1, \ldots, w_k$ aufgespannt wird, d.h. wenn ein A– bzw. A^T–invarianter Unterraum von $\mathbb{R}^n$ gefunden wurde. Ob dann die durch ein Lanczos–Lösungsverfahren bestimmte Näherungslösung die exakte Lösung von (2.1) ist, hängt davon ab, ob $\tilde{v}_{k+1} = 0$ ist oder nicht.

- $\tilde{v}_{k+1} \perp \tilde{w}_{k+1}$: Versagen des Verfahrens (*Serious Breakdown*), das auch für gutkonditionierte Matrizen eintreten kann. Dieses Ereignis läßt sich durch sogenannte *look–ahead–Strategien* vermeiden. Es gibt dafür zwei Ansätze: In [39] wird nicht direkt auf der Lanczos–Biorthogonalisierung aufgebaut, was generell höheren Aufwand pro Iterationsschritt zur Folge hat. In [18] wird dagegen eine natürliche Weiterentwicklung vorgestellt, deren Idee es ist, im Fall (2.14) nicht nur $\tilde{v}_{k+1}$ und $\tilde{w}_{k+1}$ zu biorthogonalisieren, sondern jeweils die nächsten Basisvektoren miteinzubeziehen. Die Biorthogonalitätsforderung wird dann an die durch diese Vektoren aufgespannten Räume zueinander und bezüglich K_k bzw. L_k gestellt. Es werden soviele weitere Basisvektoren berücksichtigt, wie nötig sind, um den Breakdown zu vermeiden. Man geht nur bei drohendem Breakdown anders vor, ohne daß dafür erhöhter Aufwand betrieben werden müßte.
  Der Serious Breakdown tritt allerdings selten auf. Eine stärkere Rechtfertigung finden look–ahead–Varianten bei der Vermeidung von „Beinahe–Breakdowns“, wenn $(\tilde{v}_{k+1}, \tilde{w}_{k+1})$ klein ist und starke Rundungsfehler verursachen kann.

2.8 Lanczos–Verfahren

Wie der letzte Abschnitt gezeigt hat, besitzen die aus der Lanczos–Biorthogonalisierung mit $\tilde{v_1} := r_0$ abgeleiteten *Lanczos–Verfahren* im Vergleich zu den Galerkin– und least–squares–Verfahren geringen und konstanten Speicher– und Rechenaufwand. Doch gehen gleichzeitig die Eigenschaften der Residuen verloren, die jene Verfahren erzwingen.

Satz 2.26
(Faber–Manteuffel–Theorem): *Bis auf spezielle Ausnahmen existieren PGK–Verfahren, die sowohl durch kurze Rekursionen als auch durch orthogonale oder minimale Residuen gekennzeichnet sind, nur für Matrizen $B \in \mathbb{C}^{n\times n}$ der Art*

$$B = e^{i\vartheta}(T + \sigma I) \quad \textit{mit } T = T^H \in \mathbb{C}^{n\times n},\ \vartheta \in \mathbb{R},\ \sigma \in \mathbb{C},$$

insbesondere also im reellen Fall nur für symmetrische Matrizen.

Beweis: [13]

Im allgemeinen Fall ist also mit unregelmäßigem Konvergenzverhalten zu rechnen, oder sogar damit, daß die Verfahren gar nicht konvergieren. Mit

$$W_k^T r_0 = \|r_0\|_2 W_k^T v_1 = \|r_0\|_2 e_1 \tag{2.15}$$

erhält man aus der allgemeinen Formulierung der PGK–Verfahren für Lanczos–Verfahren die Lösungsdarstellung

$$\begin{aligned} x_k :&= x_0 + V_k \left(W_k^T A V_k\right)^{-1} W_k^T r_0 \\ &= x_0 + \|r_0\|_2 V_k H_k^{-1} e_1. \end{aligned}$$

x_k ist dann die eindeutige Lösung der Petrov–Galerkin–Bedingung

$$r_k \perp L_k := \mathcal{K}_k\left(r_0, A^T\right).$$

2.8.1 Das Lanczos–Lösungsverfahren

Entsprechend zur Herleitung von FOM und GMRES aus der Arnoldi–Orthonormalisierung läßt sich aus der Lanczos–Biorthogonalisierung direkt ein Lanczos–Verfahren konstruieren.

Algorithmus 9 Lanczos–Lösungsverfahren

$r_0 := b - Ax_0$
$\tilde{v_1} := \tilde{w_1} := r_0$
Führe die Lanczos–Biorthogonalisierung durch
$y_k := H_k^{-1}\|r_0\|_2 e_1$
$x_k := x_0 + V_k y_k$

Wenngleich schon während der Biorthogonalisierung eine Konvergenzkontrolle durchgeführt werden kann, da nach [47]

$$\|r_k\|_2 = |\delta_{k+1} e_k^T y_k|\ \|v_{k+1}\|_2,$$

müssen doch die Vektoren $v_1, \ldots, v_k$ zur Berechnung von x_k explizit bereitstehen. Um den Vorteil des geringen und konstanten Speicherbedarfs der Lanczos–Biorthogonalisierung zu erhalten, wird das Verfahren auf die im folgenden Abschnitt beschriebene Weise modifiziert.

2.8.2 Bi-CG

Das *Bi–Conjugate Gradient*–Verfahren (*Bi-CG*, vorgeschlagen 1952 von Lanczos in [30], und ausgearbeitet durch Fletcher 1976 in [15]) erzeugt direkt die LR–Zerlegung des Tridiagonalsystems H_k des Lanczos–Verfahrens. Damit können bereits in der Lanczos–Iteration die Näherungslösungen x_k bestimmt werden. Statt v_{k+1} und w_{k+1} durch Dreitermrekursionen wie im Lanczos–Lösungsverfahren zu konstruieren, erhält man sie nun durch je zwei miteinander gekoppelte Zweitermrekursionen. Es müssen nunmehr für das ganze Verfahren lediglich die beiden jeweils letzten Basisvektoren gespeichert werden. Sei $L_k R_k = H_k$ die LR–Zerlegung von H_k. Mit

$$P_k = (p_1, \ldots, p_k) := V_k R_k^{-1}$$

und

$$z_k := L_k^{-1} \|r_0\| e_1$$

schreibt sich (2.8) als

$$x_k = x_0 + P_k z_k. \tag{2.16}$$

Aus der Dreiecksgestalt der Zerlegungsmatrizen folgt, daß sich x_k und r_k durch Updates

$$\begin{aligned} x_k &:= x_{k-1} + \alpha_{k-1} p_{k-1}, \\ r_k &:= r_{k-1} - \alpha_{k-1} A p_{k-1} \end{aligned}$$

berechnen lassen. Ferner läßt sich zeigen, daß r_k ein skalares Vielfaches von v_{k+1} ist, woraus auch für p_k eine rekursive Berechnungsvorschrift folgt:

$$p_k := r_k + \beta_{k-1} p_{k-1}.$$

Parallel betrachtet man das *duale System*

$$A^T \tilde{x}_k = b$$

zur Erzeugung der zweiten Rekursionsfolge. Man erhält dafür mit der Wahl von

$$\tilde{P}_k = (\tilde{p}_1, \ldots, \tilde{p}_k) := W_k L_k^{-T}$$

entsprechende Darstellungen für $\tilde{x}_k$, $\tilde{r}_k$ und $\tilde{p}_k$. Die skalaren Größen α_{k-1} und β_{k-1} in den Updates ermittelt man dann aus der Gleichung

$$\tilde{P}_k^T A P_k = L_k^{-1} W_k^T A V_k R_k^{-1} = L_k^{-1} H_k R_k^{-1} = I$$

und unter Berücksichtigung der Biorthogonalität der Residuen als skalare Vielfache der Basisvektoren.

Algorithmus 10 Bi-CG

$p_0 := r_0 := b - Ax_0$
$\tilde{p}_0 := \tilde{r}_0 := r_0$
$\rho_0 := (r_0, \tilde{r}_0)$
Für $k = 0, 1, 2, \ldots$:
  Falls $(Ap_k, \tilde{p}_k) = 0 \longrightarrow$ Stop
  $\alpha_k := \rho_k / (Ap_k, \tilde{p}_k)$
  $x_{k+1} := x_k + \alpha_k p_k$
  $r_{k+1} := r_k - \alpha_k A p_k$
  $\tilde{r}_{k+1} := \tilde{r}_k - \alpha_k A^T \tilde{p}_k$
  $\rho_{k+1} := (r_{k+1}, \tilde{r}_{k+1})$
  Falls $\rho_{k+1} = 0 \longrightarrow$ Stop
  $\beta_k := \rho_{k+1} / \rho_k$
  $p_{k+1} := r_{k+1} + \beta_k p_k$
  $\tilde{p}_{k+1} := \tilde{r}_{k+1} + \beta_k \tilde{p}_k$
Ende

Die Wahl $\tilde{r}_0 := r_0$ ist üblich, doch ist jeder beliebige Vektor $\tilde{r}_0$ mit $(r_0, \tilde{r}_0) \neq 0$ zulässig. Wählt man $\tilde{r}_0 := b - A^T x_0$, so ist $\tilde{r}_{k+1}$ das Residuum der $(k+1)$–ten Näherungslösung $\tilde{x}_{k+1}$ des dualen Problems, die durch die Rekursion

$$\tilde{x}_{k+1} := \tilde{x}_k + \alpha_k \tilde{p}_k$$

gleichzeitig berechnet werden kann. Die im Algorithmus erzeugten Sequenzen von Such– und Lösungsvektoren sind dann völlig symmetrisch zueinander.

Satz 2.27
Für die durch Bi-CG berechneten Größen gilt für $i \neq j$:

$$(Ap_i, \tilde{p}_j) = 0$$
$$(r_i, \tilde{r}_j) = 0$$

Beweis: [15]

Wie zu Beginn über die Lanczos–Verfahren erwähnt wurde, kann im allgemeinen Fall keine Konvergenzaussage getroffen werden; entsprechend weist Bi-CG in der Regel starke Oszillationen im Konvergenzverlauf der Residuennorm auf und bisweilen konvergiert das Verfahren überhaupt nicht. Die Oszillationen führen wegen der Größenordnungsunterschiede zu Rundungsfehlern, so daß auch bei Konvergenz der Norm des berechneten Residuums der echte Fehler noch groß sein kann.
Außerdem kann der Algorithmus vorzeitig zusammenbrechen, wenn

- $(Ap_k, \tilde{p}_k) = 0$: Dieser Fall tritt genau dann ein, wenn die Lanczos–Matrix singulär ist und daher die Galerkin–Bedingung nicht erfüllt werden kann. Man sieht am Algorithmus, daß dann tatsächlich x_{k+1} nicht bestimmt werden kann. Dieser Fall läßt sich durch eine quasi–Minimierung des Residuums vermeiden (dazu später), was gleichzeitig das Konvergenzverhalten glättet.

- $(r_{k+1}, \tilde{r}_{k+1}) = 0$: Dies entspricht dem Breakdown in der Lanczos–Biorthogonalisierung, da r_{k+1} und $\tilde{r}_{k+1}$ skalare Vielfache von v_{k+1} bzw. w_{k+1} in der Lanczos–Biorthogonalisierung sind. Im Fall des Lucky Breakdown ($r_{k+1} = 0$ oder $\tilde{r}_{k+1} = 0$) ist der erfolgreiche Abschluß des Algorithmus offensichtlich.
  Im anderen Fall helfen die beschriebenen look–ahead–Methoden, die sinnvollerweise gemeinsam mit einer quasi–Minimierung eingesetzt werden, um generelle Stabilität des Verfahrens zu erreichen.

Ein weiterer Nachteil kann sein, daß Bi-CG explizit A^T verwendet, was nicht realisierbar ist, wenn A nicht explizit vorliegt. Dies ist bei den in dieser Arbeit betrachteten Problemen nicht der Fall, doch sind die zur Vermeidung dieser Einschränkung entwickelten Verfahren auch allgemein interessant.
Im folgenden werden nun Verfahren vorgestellt, die die Vorteile des Bi-CG (geringer Rechen– und Speicheraufwand pro Iteration) besitzen, dabei aber die genannten Nachteile umgehen.

2.8.3 CGS

Das *Conjugate Gradient Stabilized*–Verfahren (*CGS*, vorgestellt von Sonneveld 1989 in [51]) faßt zwei Bi-CG–Schritte zu einem zusammen, welches sich ohne Erhöhung des Aufwandes, aber dafür ohne die Verwendung der Transponierten A^T realisieren läßt. Um das Vorgehen beschreiben zu können, stellt man die im Bi-CG konstruierten Vektoren als Polynome dar. Wie man induktiv aus den Rekursionen schließt, gilt

$$r_k = \varphi_k(A)r_0 \qquad \tilde{r}_k = \varphi_k(A^T)r_0, \tag{2.17}$$

$$p_k = \psi_k(A)r_0 \qquad \tilde{p}_k = \psi_k(A^T)r_0, \tag{2.18}$$

mit Polynomen φ_k und ψ_k vom Grad $\leq k$. Setzt man nun

$$r_k^* := \varphi_k^2(A)r_0 \quad \text{und} \quad p_k^* := \psi_k^2(A)r_0,$$

so folgt für die skalaren Größen des Bi-CG–Algorithmus:

$$\begin{aligned}\alpha_k &= \frac{(r_k, \tilde{r}_k)}{(Ap_k, \tilde{p}_k)} = \frac{\left(\varphi_k(A)r_0, \varphi_k(A^T)r_0\right)}{(A\psi_k(A)r_0, \psi_k(A^T)r_0)} \\ &= \frac{(\varphi_k^2(A)r_0, r_0)}{(A\psi_k^2(A)r_0, r_0)} = \frac{(r_k^*, r_0)}{(Ap_k^*, r_0)}, \\ \beta_k &= \frac{(r_{k+1}, \tilde{r}_{k+1})}{(r_k, \tilde{r}_k)} = \frac{\left(\varphi_{k+1}(A)r_0, \varphi_{k+1}(A^T)r_0\right)}{(\varphi_k(A)r_0, \varphi_k(A^T)r_0)} \\ &= \frac{\left(\varphi_{k+1}^2(A)r_0, r_0\right)}{(\varphi_k(A)^2 r_0, r_0)} = \frac{\left(r_{k+1}^*, r_0\right)}{(r_k^*, r_0)}.\end{aligned} \tag{2.19}$$

Die Transponierte ist für deren Berechnung also nicht mehr erforderlich. Mit

$$q_k^* := \varphi_{k+1}(A)\psi_k(A)r_0 \quad \text{und} \quad u_k^* := \varphi_k(A)\psi_k(A)r_0$$

erhält man über Ausmultiplizieren durch φ_k^2 und ψ_k^2 Rekursionsformeln zur Berechnung von p_{k+1}^*, r_{k+1}^* und x_{k+1}^* (siehe [51]). Unter Vernachlässigung des hochgeschriebenen „$*$“ erhält man folgenden Algorithmus:

Algorithmus 11 CGS

$u_0 := p_0 := r_0 := b - Ax_0$
$\rho_0 := (r_0, r_0)$
Für $k = 0, 1, 2, \ldots$:
  Falls $(Ap_k, r_0) = 0 \longrightarrow$ Stop
  $\alpha_k := \rho_k / (Ap_k, r_0)$
  $q_k := u_k - \alpha_k A p_k$
  $x_{k+1} := x_k + \alpha_k (u_k + q_k)$
  $r_{k+1} := r_k - \alpha_k A (u_k + q_k)$
  Falls $\rho_k = 0 \longrightarrow$ Stop
  $\rho_{k+1} := (r_{k+1}, r_0)$
  $\beta_k := \rho_{k+1}/\rho_k$
  $u_{k+1} := r_{k+1} + \beta_k q_k$
  $p_{k+1} := u_{k+1} + \beta_k (q_k + \beta_k p_k)$
Ende

CGS konvergiert schneller als Bi-CG, da eine CGS–Iteration zwei Bi-CG–Iterationen entspricht. Wenn CGS aber auch ohne die Transponierte auskommt, so bleiben dennoch die anderen Nachteile des Bi-CG bestehen, nämlich die Rundungsfehler durch das unregelmäßige Konvergenzverhalten, das durch die Quadrierung der Polynome noch verstärkt wird, und die Gefahr des vorzeitigen Breakdown.

2.8.4 Bi-CGSTAB

Motiviert durch die Grundidee des CGS und um dessen Konvergenzverhalten zu glätten, entwickelte van der Vorst 1992 in [52] (nach Vorarbeiten mit Sonneveld 1990 in [53]) *Bi-CGSTAB* (*Bi-CG STABilized*). Er nutzte aus, daß zur Vermeidung der Transponierten die Polynome nicht unbedingt quadriert werden müssen. Bi-CG nimmt eine Skalierung vor, um dasselbe Polynom zur Konstruktion von r_k und $\tilde{r}_k$ zu erhalten (vgl. (2.17),(2.18)). Während CGS diese Wahl übernimmt und die Polynome mit sich selbst multipliziert, verallgemeinert Bi-CGSTAB die Vorgehensweise und baut in jedem Schritt durch

$$r_k^* := \chi_k(A)\varphi_k(A)r_0 \quad \text{und} \quad p_k^* := \chi_k(A)\psi_k(A)r_0$$

mit

$$\begin{aligned}\chi_0(A) &:= 1\\ \chi_k(A) &:= (I - \omega_{k-1}A)\chi_{k-1}(A)\end{aligned}$$

einen zusätzlichen Freiheitsgrad ω_k ein, der der Minimierung des Residuums dient. Ist ein CGS–Schritt die Verknüpfung zweier Bi-CG–Schritte, so kann man dagegen eine Bi-CGSTAB–Iteration als Verknüpfung eines Bi-CG–Schrittes mit einem GMRES–Schritt (der durch $\chi_k(A)$ dargestellt wird) betrachten.
Ähnlich wie in (2.19) erhält man unter Ausnutzung der Galerkin–Bedingung

$$\begin{aligned}\alpha_k &:= \frac{(r_k^*, r_0)}{(Ap_k^*, r_0)},\\ \beta_k &:= \frac{\alpha_k}{\omega_k}\frac{\left(r_{k+1}^*, r_0\right)}{(r_k^*, r_0)}.\end{aligned}$$

Ebenfalls entsprechend zum CGS liefert Ausmultiplizieren der Polynome und die Definition von

$$s_k^* := r_k^* - \alpha_k A p_k^*$$

rekursive Darstellungen von p_{k+1}^*, x_{k+1}^* und insbesondere

$$r_{k+1}^* := (I - \omega_k A)\, s_k^*,$$

woraus man ω_k so bestimmt, daß die Residuennorm minimal wird. Man erhält aus der Forderung

$$\omega_k = \arg\min_{\omega\in\mathbb{R}} \|(I - \omega A)s_k^*\|_2^2$$

durch Differentiation das lokale Minimum

$$\omega_k := \frac{(As_k^*, s_k^*)}{(As_k^*, As_k^*)},$$

und damit unter Weglassen des „$*$“:

Algorithmus 12 Bi-CGSTAB

$p_0 := r_0 := b - Ax_0$
für $k = 0, 1, 2, \ldots, n$:
$\alpha_k := \frac{(r_k, r_0)}{(Ap_k, r_0)}$
$s_k := r_k - \alpha_k A p_k$
$\omega_k := \frac{(As_k, s_k)}{\left(As_k^*, As_k^*\right)}$
$x_{k+1} := x_k + \alpha_k p_k + \omega_k s_k$
$r_{k+1} := s_k - \omega_k A s_k$
$\beta_k := \frac{\alpha_k \left(r_{k+1}^*, r_0\right)}{\omega_k \left(r_k^*, r_0\right)}$
$p_{k+1} := r_{k+1} + \beta_k \left(p_k - \omega_k A p_k\right)$
Ende

Aus der Idee des Bi-CGSTAB, GMRES(1) mit Bi-CG zu verknüpfen, wurde von Sleijpen und Fokkema in [50] *Bi-CGSTAB(m)* durch Erweiterung des Prinzips mit GMRES(m) entwickelt.

2.8.5 QMR

QMR (*Quasi Minimized Residual*, entwickelt 1991 von Freund und Nachtigal in [19]) stellt eine Verschmelzung von Bi-CG und GMRES(1) dar (zu unterscheiden von der Hintereinanderausführung, die Bi-CGSTAB vornimmt). Es berechnet auf Basis des Lanczos–Algorithmus Näherungslösungen, die

$$\begin{aligned} \|r_k\|_2 &= \|r_0 - AV_k y_k\|_2 \\ &= \|V_{k+1} \left(\|r_0\|_2 e_1 - \tilde{H}_k y_k \right) \|_2 \end{aligned}$$

mit

$$\tilde{H}_k = \begin{pmatrix} H_k \\ \delta_{k+1} e_k^T \end{pmatrix} \in \mathbb{R}^{(k+1)\times k}$$

minimieren sollen, um das Konvergenzverhalten des Bi-CG zu glätten. Da die durch die Lanczos–Biorthogonalisierung erzeugte Matrix V_{k+1} nicht notwendig orthogonal ist, ist diese Minimierung, anders als bei den least–squares–Verfahren, nicht immer möglich. Dennoch wird, wie wenn V_{k+1} orthogonal wäre, wie dort $x_k := x_0 + V_k y_k$ bestimmt mit

$$y_k := \arg\min_{y \in \mathbb{R}^k} \left\| \|r_0\|_2 e_1 - \tilde{H}_k y \right\|_2 . \tag{2.20}$$

Nach [17] ist dies äquivalent zur Minimierung des Residuums bezüglich einer von k abhängigen Norm, man spricht von einer *quasi–Minimierung*. Deshalb liegt hier kein Widerspruch zu Satz 2.26 vor, da dort lediglich die Minimierung bezüglich einer bestimmten Norm ausgeschlossen wird.
(2.20) wird wieder über eine QR–Zerlegung von $\tilde{H}_k$ durch Givens–Rotationen gelöst. Wegen der Tridiagonalgestalt der Matrix ist QMR eine Variante des DQGMRES(2), bei der die Arnoldi–Orthogonalisierung durch eine Lanczos–Biorthogonalisierung ersetzt ist.
Seien $\tilde{R}_k = (R_k^T, 0)^T$ und $\tilde{g}_k = (g_{k-1}^T, \gamma_k)^T$ der Vektor, die durch die Givens–Rotationen aus $\tilde{H}_k$ bzw. $\|r_0\|_2 e_1$ entstehen. R_k ist also eine obere Dreiecksmatrix mit zwei Nebendiagonalen, deren oberer Einträge durch $\xi_3, \ldots, \xi_k$ bezeichnet seien.
Damit lautet der Algorithmus des QMR wie folgt:

Algorithmus 13 QMR

$r_0 := b - Ax_0$
$w_1 := v_1 := r_0 / \|r_0\|_2$
Für $k = 1, \ldots, n$:
  Führe den k–ten Schritt der (look–ahead–)Lanczos–Biorthogonalisierung durch
  Update von $\tilde{R}_k$ und $\tilde{g}_k$ durch die bisherigen Givens–Rotationen:
    $(\tilde{R}_k)_k := \Omega_{k-1}\Omega_{k-2}(\tilde{H}_k)_k$
  und die zu δ_{k+1} :
    $s_k := \delta_{k+1} / \sqrt{\alpha_k^2 + \delta_{k+1}^2}$
    $c_k := \alpha_k / \sqrt{\alpha_k^2 + \delta_{k+1}^2}$
    $\gamma_{k+1} := -s_k \gamma_k$
    $\gamma_k := c_k \gamma_k$
    $\alpha_k := c_k \alpha_k + s_k \delta_{k+1} = \sqrt{\delta_{k+1}^2 + \alpha_k^2}$
  Update von $x_k := x_0 + V_k R_k^{-1} g_k$
    $p_k := (v_k - \beta_k p_{k-1} - \xi_k p_{k-2}) / \alpha_k$
    $x_k := x_{k-1} + \gamma_k p_k$
Ende

Wie beim Bi-CG ist allgemeiner die Wahl eines beliebigen Vektors w_1 mit $(v_1, w_1) \neq 0$ und $\|w_1\|_2 = 1$ möglich.
Da $\tilde{H}_k$ nichtsingulär ist, und die Galerkin–Bedingung daher immer erfüllt werden kann, ist die Ursache des Bi-CG–Breakdowns beseitigt. Durch eine look–ahead–Variante kann auch der Serious Breakdown vermieden werden, so daß man ein stabiles Verfahren erhält. Darüberhinaus ist die Quasi–Minimierungseigenschaft des QMR, wenn auch nicht so stark wie die der least–squares–Verfahren, doch ausreichend, um Konvergenzaussagen formulieren zu können:

Satz 2.28
Sei $H_k \in \mathbb{R}^{(k\times k)}$ die Tridiagonalmatrix, die aus der durch QMR bestimmten Matrix $\tilde{H}_k$ durch Streichen der letzten Zeile entsteht. Sie sei diagonalisierbar und $X \in \mathbb{R}^{(k\times k)}$ eine Matrix, die aus Eigenvektoren von H_k besteht. Dann gilt

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \|V_{k+1}\|_2 \,\mathrm{cond}_2(X)\varepsilon_k$$

mit ε_k wie in Satz 2.22.

Beweis: [19]

Satz 2.29
Seien r_k^{QMR} und r_k^{GMRES} die Residuen zu den durch QMR bzw. GMRES berechneten Näherungslösungen x_k. Dann gilt

$$\|r^{QMR}{}_k\|_2 \leq \mathrm{cond}_2\,(V_{k+1})\,\|r_k^{GMRES}\|_2$$

Beweis: [33]

Da die Vektoren $v_1, \ldots, v_k$ Einheitsvektoren sind, läßt sich diese Abschätzung durch

$$\|V_{k+1}\|_2 \leq \sqrt{k+1}$$

konkreter formulieren. In [19] wird gezeigt, daß die Bi-CG–Iterierten leicht aus den QMR–Iterierten konstruiert werden können, so daß man QMR als stabilisiertes Bi-CG betrachten kann.
Entsprechend der Herleitung des CGS aus Bi-CG wurde von Freund und Szeto 1991 in [20] das *QMR Squared*–Verfahren (*QMRS*) vorgestellt.

2.8.6 TFQMR

Mit *TFQMR* (*Transposed–Free QMR*) übertrug Freund 1993 in [16] die Idee der quasi–Minimierung auf CGS und erhielt so ein (bei Verwendung von look–ahead) stabiles Verfahren, welches ohne die Transponierte auskommt. Damit ist dies ein Lanczos–Verfahren ohne die arttypischen Nachteile, dabei aber mit weiterhin geringem und konstantem Speicher- und Rechenaufwand.
Im folgenden sei

$$\begin{aligned} \lfloor\cdot\rfloor : \mathbb{R} &\longrightarrow \mathbb{Z} \\ x &\longmapsto \lfloor x\rfloor \in]x-1, x] \end{aligned}$$

die Abbildung einer reellen Zahl auf die nächstkleinere ganze Zahl.
Um ein Minimierungsproblem stellen zu können, benötigt man eine Basis für den Lösungsraum K_k. Eine solche tritt im CGS–Verfahren zwar nicht mehr explizit auf, läßt sich aber leicht aus den dort berechneten Grössen zusammensetzen.

Satz 2.30
Mit den Bezeichnungen des CGS–Algorithmus und unter der Annahme, daß er nicht vor der j–ten Iteration abgebrochen hat, bilden $\tilde{v}_1, \ldots, \tilde{v}_j$ mit

$$\tilde{v}_i := \left\{ \begin{array}{ll} u_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ ungerade} \\ q_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ gerade} \end{array} \right\} \quad \textit{für } 1 \leq i \leq j$$

eine Basis von K_j.

Satz 2.31
Mit

$$\tilde{w}_i := \left\{ \begin{array}{ll} \varphi^2_{\lfloor \frac{i-1}{2} \rfloor}(A)\, r_0, & \textit{falls } i \textit{ ungerade} \\ \varphi_{\lfloor \frac{i-1}{2} \rfloor}(A)\, \varphi_{\lfloor \frac{i}{2} \rfloor}(A)\, r_0, & \textit{falls } i \textit{ gerade} \end{array} \right\} \quad \textit{für } 1 \leq i \leq j+1$$

seien

$$\begin{aligned} \tilde{V}_j &:= (\tilde{v}_1 \ldots \tilde{v}_j)\,, \\ \tilde{W}_{j+1} &:= (\tilde{w}_1 \ldots \tilde{w}_{j+1}) \end{aligned}$$

und ferner

$$B_j := \begin{pmatrix} \alpha_0^{-1} & & & & \\ -\alpha_0^{-1} & \alpha_0^{-1} & & & \\ & \ddots & \ddots & & \\ & & -\alpha^{-1}_{\lfloor \frac{j-2}{2} \rfloor} & \alpha^{-1}_{\lfloor \frac{j-1}{2} \rfloor} \\ & & & -\alpha^{-1}_{\lfloor \frac{j-1}{2} \rfloor} \end{pmatrix} \in \mathbb{R}^{(j+1)\times j}$$

definiert. Dann gilt

$$A\tilde{V}_j = \tilde{W}_{j+1} B_j.$$

Beweise: [32]

Über die Darstellung der Näherungslösung als

$$x_j := x_0 + \tilde{V}_j y_j \in x_0 + K_j \tag{2.21}$$

mit $y_j \in \mathbb{R}^j$, die Feststellung, daß nach Definition $\tilde{w}_1 = r_0$, und die Definition der *Skalierungsmatrix*,

$$S_{j+1} := \begin{pmatrix} \|\tilde{w}_1\|_2 & & \\ & \ddots & \\ & & \|\tilde{w}_{j+1}\|_2 \end{pmatrix} \in \mathbb{R}^{(j+1)\times(j+1)},$$

erhält man aus dem letzten Satz:

$$\begin{aligned}\|r_j\|_2 &= \|b - Ax_j\|_2 \\ &= \left\|b - Ax_0 - A\tilde{V}_j y_j\right\|_2 \\ &= \left\|r_0 - \tilde{W}_{j+1} B_j y_j\right\|_2 \\ &= \left\|\tilde{W}_{j+1} S_{j+1}^{-1} S_{j+1} \left(e_1 - B_j y_j\right)\right\|_2 \\ &\leq \left\|\tilde{W}_{j+1} S_{j+1}^{-1}\right\|_2 \underbrace{\left\|\|w_1\|_2 e_1 - S_{j+1} B_j y_j\right\|_2}_{=:\tau_j}. \end{aligned} \tag{2.22}$$

Für die euklidische Norm einer Matrix $M \in \mathbb{R}^{m\times n}$ gilt:

$$\|M\|_2 \leq \left(\sum_{i=1}^{m}\sum_{i=1}^{n} a_{i,j}^2\right)^{\frac{1}{2}}.$$

Daraus erhält man

$$\left\|\tilde{W}_{j+1} S_{j+1}^{-1}\right\|_2 \leq \left(\sum_{i=1}^{j+1} \left\|\frac{\tilde{w}_i}{\|\tilde{w}_i\|_2}\right\|_2\right)^{\frac{1}{2}} = \sqrt{j+1}$$

und somit:

Satz 2.32
Mit τ_j wie in (2.22) definiert, ist

$$\|r_j\|_2 \leq \sqrt{j+1}\,\tau_j.$$

Um $\|r_j\|_2$ zu quasi–minimieren, wird nun y_j so bestimmt, daß τ_j minimal wird. Dazu führt man eine QR–Zerlegung von $S_{j+1}B_j \in \mathbb{R}^{(j+1)\times j}$ über Givens–Rotationen durch, die sich als Rekursion ausdrücken läßt, da die Matrix eine obere Bandmatrix der Breite 2 ist. Es ergibt sich:

Satz 2.33
Der Vektor $y_j \in \mathbb{R}^j$, der τ_j minimiert, ist für $j > 0$ mit

$$\begin{aligned}\vartheta_j &:= \frac{\|\tilde{w}_{j+1}\|_2}{\tau_{j-1}}, \\ c_j &:= \frac{1}{\sqrt{1+\vartheta_j^2}}\end{aligned}$$

und

$$\tilde{y}_j := \left(\alpha_0, \alpha_0, \alpha_1, \ldots, \alpha_{\lfloor\frac{j-1}{2}\rfloor}\right)^T \in \mathbb{R}^j,$$

wobei $\{\alpha_i\}_{1\leq i\leq \lfloor \frac{j-1}{2} \rfloor}$ *durch den CGS–Algorithmus bestimmt sind, gegeben durch*

$$y_j := \left(1 - c_j^2\right) \begin{pmatrix} y_{j-1} \\ 0 \end{pmatrix} + c_j^2 \tilde{y}_j.$$

Das Minimum hat dann den Wert

$$\tau_j := \tau_{j-1} \vartheta_j c_j \quad mit \quad \tau_0 := \|r_0\|_2.$$

Beweis: [32]

Durch Einsetzen dieser Resultate in die Darstellung (2.21) und einige Rechnung findet man schließlich:

Satz 2.34
Für $j > 0$ *besitzt mit*

$$\begin{aligned} \eta_j &:= \alpha_{\lfloor \frac{j-1}{2} \rfloor} c_j^2, \\ d_0 &:= 0 \\ d_j &:= \tilde{v}_j + \frac{\vartheta_{j-1}^2 \eta_{j-1}}{\alpha_{\lfloor \frac{j-1}{2} \rfloor}} d_{j-1} \end{aligned}$$

die j*–te Näherungslösung die rekursive Darstellung*

$$x_j := x_{j-1} + \eta_j d_j.$$

Beweis: [32]

Die verbleibenden Vektoren q_k, u_{k+1} und p_{k+1} des CGS–Algorithmus lassen sich über die Einführung von

$$t_k := A p_k$$

einsparen. Es gilt dann

$$\begin{aligned} \tilde{v}_{2k} &= q_{k-1} = u_{k-1} - \alpha_{k-1} t_{k-1} = \tilde{v}_{2k-1} - \alpha_{k-1} t_{k-1}, \\ \tilde{v}_{2k+1} &= u_k = r_k + \beta_{k-1} q_{k-1} = \tilde{w}_{2k+1} + \beta_{k-1} \tilde{v}_{2k}, \end{aligned}$$

wobei sich wegen

$$p_k = u_k + \beta_{k-1} \left(q_{k-1} + \beta_{k-1} p_{k-1}\right) = \tilde{v}_{2k+1} + \beta_{k-1} \left(\tilde{v}_{2k} + \beta_{k-1} p_{k-1}\right)$$

auch t_k unabhängig von diesen Größen rekursiv darstellen läßt als

$$t_k := A\tilde{v}_{2k-1} + \beta_{k-1} \left(A\tilde{v}_{2k} + \beta_{k-1} t_{k-1}\right).$$

Damit läßt sich der TFQMR–Algorithmus folgendermaßen formulieren:

Algorithmus 14 TFQMR

$\tilde{w}_1 := \tilde{v}_1 := r_0 := b - Ax_0$
$\tau_0 := \|r_0\|_2$
$t_0 := A\tilde{v}_1$
$d_0 := 0, \qquad \eta_0 := 0, \qquad \vartheta_0 := 0$
Für $k = 1, 2, \ldots :$
  $\alpha_{k-1} := \frac{(\tilde{w}_{2k-1}, r_0)}{(t_{k-1}, r_0)}$
  $\tilde{v}_{2k} := \tilde{v}_{2k-1} - \alpha_{k-1} t_{k-1}$
  Für $j = 2k-1, 2k :$
    $\tilde{w}_{j+1} := w_j - \alpha_{k-1} A\tilde{v}_j$
    $\vartheta_j := \frac{\|\tilde{w}_{j+1}\|_2}{\tau_{j-1}}$
    $c_j := \frac{1}{\sqrt{1+\vartheta_j^2}}$
    $d_j := \tilde{v}_j + \frac{\vartheta_{j-1}^2 \eta_{j-1}}{\alpha_{k-1}} d_{j-1}$
    $\eta_j := c_j^2 \alpha_{k-1}$
    $x_j := x_{j-1} + \eta_j d_j$
    $\tau_j := \tau_{j-1} \vartheta_j c_j \;\rightarrow\; \|r_j\|_2 \leq \sqrt{j+1}\, \tau_j$
  Ende
  $\beta_{k-1} := \frac{(\tilde{w}_{2k+1}, r_0)}{(\tilde{w}_{2k-1}, r_0)}$
  $\tilde{v}_{2k+1} := \tilde{w}_{2k+1} + \beta_{k-1} \tilde{v}_{2k}$
  $t_k := A\tilde{v}_{2k+1} + \beta_{k-1} (A\tilde{v}_{2k} + \beta_{k-1} t_{k-1})$
Ende

2.8.7 QMRCGSTAB

Chan et al. übertrugen 1994 in [8] nach dem Muster des TFQMR das Prinzip der Quasi–Minimierung auf Bi-CGSTAB. Entsprechend wie bei der Herleitung des TFQMR zeigt man die folgenden Aussagen:

Satz 2.35
Mit den Bezeichnungen des Bi-CGSTAB–Algorithmus bilden $\tilde{v}_1, \ldots, \tilde{v}_j$ mit

$$\tilde{v}_i := \left\{ \begin{array}{ll} p_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ ungerade} \\ s_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ gerade} \end{array} \right\} \quad \textit{für } 1 \leq i \leq j$$

eine Basis von K_j.

Satz 2.36
Mit

$$\tilde{w}_i := \left\{ \begin{array}{ll} r_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ ungerade} \\ s_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ gerade} \end{array} \right\} \quad \textit{für } 1 \leq i \leq j+1$$

und

$$\delta_i := \left\{ \begin{array}{ll} \alpha_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ ungerade} \\ \omega_{\lfloor \frac{i-1}{2} \rfloor}, & \textit{falls } i \textit{ gerade} \end{array} \right\} \quad \textit{für } 1 \leq i \leq j$$

definiere man

$$\begin{aligned} \tilde{V}_j &:= (\tilde{v}_1 \dots \tilde{v}_j), \\ \tilde{W}_{j+1} &:= (\tilde{w}_1 \dots \tilde{w}_{j+1}), \end{aligned}$$

und, anders als in Satz 2.31,

$$B_j := \begin{pmatrix} \delta_1^{-1} & & & \\ -\delta_1^{-1} & \delta_2^{-1} & & \\ & \ddots & \ddots & \\ & & -\delta_{j-1}^{-1} & \delta_j^{-1} \\ & & & -\delta_j^{-1} \end{pmatrix} \in \mathbb{R}^{(j+1)\times j}.$$

Dann gilt

$$A\tilde{V}_j = \tilde{W}_{j+1} B_j.$$

Satz 2.37
Mit

$$\tau_j := \left\| \|w_1\|_2 e_1 - S_{j+1} B_j y_j \right\|_2$$

ist

$$\|r_j\|_2 \leq \sqrt{j+1}\, \tau_j.$$

Beweise: [32]

Mit den im Vergleich zum TFQMR anderen Einträgen der Matrix B_j, die dabei aber die gleiche Struktur besitzt, folgt entsprechend:

Satz 2.38
Der Vektor $y_j \in \mathbb{R}^j$, der τ_j minimiert, ist für $j > 0$ mit

$$\begin{aligned} \vartheta_j &:= \frac{\|\tilde{w}_{j+1}\|_2}{\tau_{j-1}}, \\ c_j &:= \frac{1}{\sqrt{1+\vartheta_j^2}} \end{aligned}$$

und

$$\tilde{y}_j := (\delta_1, \delta_2, \ldots, \delta_j)^T \in \mathbb{R}^j$$

gegeben durch

$$y_j := \left(1 - c_j^2\right) \begin{pmatrix} y_{j-1} \\ 0 \end{pmatrix} + c_j^2 \tilde{y}_j.$$

Das Minimum hat dann den Wert

$$\tau_j := \tau_{j-1} \vartheta_j c_j \quad mit \quad \tau_0 := \|r_0\|_2.$$

Satz 2.39
Für $j > 0$ besitzt mit

$$\begin{aligned} \eta_j &:= \delta_j c_j^2, \\ d_j &:= \tilde{v}_j + \frac{\vartheta_{j-1}^2 \eta_{j-1}}{\delta_j} d_{j-1} \quad mit \quad d_0 := 0 \end{aligned}$$

die j–te Näherungslösung die rekursive Darstellung

$$x_j := x_{j-1} + \eta_j d_j.$$

Beweise: [32]

Anders als bei TFQMR lassen sich die weiteren Vektoren des Bi-CGSTAB nicht durch die QMRCGSTAB–Vektoren darstellen, sondern werden gemäß der Definition von $\tilde{v}_i$ und $\tilde{w}_i$ für diese in die obigen Formeln eingesetzt. Es wird jedoch wieder

$$t_k := Ap_k$$

eingeführt, da dieses Produkt mehrfach berechnet wird.

Algorithmus 15 QMRCGSTAB

$p_0 := r_0 := b - Ax_0$
$\tau_0 := \|r_0\|_2$
$d_0 := 0, \quad \eta_0 := 0, \quad \delta_0 := 0$
Für $k = 1, \ldots, \lfloor \frac{n}{2} \rfloor$:
$\alpha_{k-1} := \frac{(r_{k-1}, r_0)}{(t_{k-1}, r_0)}$
$s_{k-1} := r_{k-1} - \alpha_{k-1} A p_{k-1}$
$\tilde{\vartheta}_k := \frac{\|s_{k-1}\|_2}{\tau_{k-1}}$
$\tilde{c}_k := \left(1 + \tilde{\vartheta}_k^2\right)^{-\frac{1}{2}}$
$\tilde{d}_k := p_{k-1} + \frac{\vartheta_{k-1}^2 \eta_{k-1}}{\alpha_{k-1}} d_{k-1}$
$\tilde{\eta}_k := \tilde{c}_k^2 \alpha_{k-1}$
$\tilde{x}_k := x_{k-1} + \tilde{\eta}_k \tilde{d}_k$
$\tilde{\tau}_k := \tau_{k-1} \tilde{\vartheta}_k \tilde{c}_k$
$\omega_{k-1} := \frac{(As_{k-1}, s_{k-1})}{(As_{k-1}, As_{k-1})}$
$r_k := s_{k-1} - \omega_{k-1} A s_{k-1}$
$\vartheta_k := \frac{\|r_k\|_2}{\tilde{\tau}_k}$
$c_k := \sqrt{\frac{1}{1+\vartheta_k^2}}$
$d_k := s_{k-1} + \frac{\tilde{\vartheta}_k^2 \tilde{\eta}_k}{\omega_{k-1}} \tilde{d}_k$
$\eta_k := c_k^2 \omega_{k-1}$
$x_k := \tilde{x}_k + \eta_k d_k$
$\tau := \tilde{\tau}_k \vartheta_k c_k$
$\Rightarrow \|r_j\|_2 \leq \sqrt{2k+1}\, \tau_j$
$\beta_{k-1} := \frac{\alpha_{k-1}}{\omega_{k-1}} \frac{(r_k, r_0)}{(r_{k-1}, r_0)}$
$p_k := r_k + \beta_{k-1} (p_{k-1} - \omega_{k-1} A p_{k-1})$
Ende

2.9 Der symmetrische Fall

Ist A symmetrisch und wird das Arnoldi–Verfahren durchgeführt, so ist die entstehende Hessenbergmatrix $H_k = V_k^T A V_k$ ebenfalls symmetrisch und damit eine Tridiagonalmatrix. Seien ihre Einträge bezeichnet durch

$$H_k = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \end{pmatrix}.$$

Dann vereinfacht sich das Arnoldi–Verfahren zu demselben Algorithmus, den man aus der Lanczos–Biorthogonalisierung erhält, wenn man die dort wegen $K_k = L_k$ doppelt

berechneten Größen vernachlässigt:

Algorithmus 16 Lanczos–Orthonormalisierung

Sei $\tilde{v}_1 \in \mathbb{R}^n$ gegeben.
$v_1 := \tilde{v}_1 / \|\tilde{v}_1\|_2$
$v_0 := 0$
$\beta_1 := 0$
Für $k = 1, 2, \ldots$:
  $\alpha_k := (\tilde{v}_k, v_k)$
  $\tilde{v}_{k+1} := Av_k - \alpha_k - \beta_k v_{k-1}$
  $\beta_{k+1} := \|\tilde{v}_{k+1}\|_2$
  Wenn $\beta_{k+1} = 0 \longrightarrow$ Stop
  $v_{k+1} := \tilde{v}_{k+1} / b_{k+1}$
Ende

Gegenüber der Lanczos–Biorthogonalisierung ist nun die Möglichkeit des Serious Breakdown ausgeschlossen. Man bezeichnet diese Methode auch als *symmetrisches Lanczos–Verfahren*, was jedoch zu Verwechslungen mit dem entsprechenden Lösungsverfahren führen kann. Zu den aus dem Arnoldi–Algorithmus entstehenden Lösungsverfahren gibt es folgende symmetrische Äquivalente:

- Das **symmetrische Lanczos–Lösungsverfahren** entspricht FOM (bzw. IOM(2)) unter Verwendung der Lanczos– statt der Arnoldi–Orthonormalisierung. Nur führt man hier wegen der Tridiagonalgestalt von H_k zur Berechnung von $y_k := H_k^{-1}\|r_0\|_2 e_1$ eine LR–Zerlegung von H_k statt Givens–Rotationen durch. Weiterhin ist $\|b - Ax_k\|_2 = h_{k+1,k}|e_k^T y_k|$.

- **D-Lanczos** *(Direct Lanczos)* ist die Modifikation dieses Verfahrens, bei der statt H_k direkt die LR–Zerlegung konstruiert wird, entsprechend zu DIOM(2).

- **CG** *(Conjugate Gradient)* ist eine Umformulierung des D–Lanczos–Verfahrens für positiv definite Matrizen unter Ausnutzung von Orthogonalitätsbeziehungen zwischen den auftretenden Vektoren (siehe [38]). Die Verwandschaft zum symmetrischen Lanczos–Lösungsverfahren ist dieselbe wie die des Bi-CG zum unsymmetrischen. Dieses Verfahren wird im folgenden Abschnitt dargestellt.

- **CR** *(Conjugate Residual)* entsteht durch Verwendung der Lanczos– statt Arnoldi–Orthonormalisierung in GMRES (siehe [47]). Gegenüber CG hat dieses Verfahren bei ähnlichem Konvergenzverhalten jedoch höheren Speicher– und Rechenaufwand.

- **SYMMLQ** *(SYMMetric LQ)* führt eine LQ–Zerlegung von H_k durch. Dieses Verfahren wurde 1975 von Paige und Saunders in [38] beschrieben.

Bei den ersten beiden Verfahren sollte jedoch die LR–Zerlegung mit einer Pivotisierung durchgeführt werden (dies ist wie bei DIOM(2) unter etwas höherem Aufwand auch für das D–Lanczos–Verfahren möglich), um Breakdowns und numerische Instabilität zu vermeiden.

2.9.1 CG

CG (*Conjugate Gradient*) ist das bekannteste und bedeutendste PGK–Verfahren. Es wurde 1952 von Hestenes und Stiefel in [26] vorgestellt, zeitgleich zur Präsentation von Lanczos' Biorthogonalisierungsverfahren in [30] (in derselben Zeitschrift). Ausgehend von dieser Methode entstand nach und nach die gesamte in diesem Kapitel dargestellte Theorie. Umgekehrt ist CG der gemeinsame Kern der verschiedenen Klassen von Lösungsverfahren. Es läßt sich, wie bereits erwähnt, als Galerkin– und Lanczos–Verfahren herleiten, aber auch als least–squares–Verfahren. Als Galerkin–Verfahren erfüllt es die Voraussetzungen von Satz 2.17, nach dem CG bei jeder Iteration über $x_0 + K_k$ sowohl den Fehler als auch das Residuum bezüglich der durch A bzw. A^{-1} definierten Normen minimiert. Dementsprechend kann CG auch aus dieser Eigenschaft heraus definiert werden, nämlich als Minimierungsverfahren für das Funktional

$$\Phi(x) := \frac{1}{2}(x,x)_A - (x,b),$$

dessen Minimum gerade bei $x = A^{-1}b$ angenommen wird (diese verbreitete Herleitung kann zum Beispiel in [23] oder formaler in [24] gefunden werden).
Die nach Satz 2.17 gewährleistete Konvergenz des Fehlers kann abgeschätzt werden:

Satz 2.40
Für die durch das CG–Verfahren berechnete k–te Näherungslösung x^k ist

$$\|x_k - x\|_A \leq 2 \left(\frac{\sqrt{\mathrm{cond}_2(A)} - 1}{\sqrt{\mathrm{cond}_2(A)} + 1} \right)^k \|x_0 - x\|_A \quad \textit{für } k = 1, 2, \ldots \tag{2.23}$$

Beweis: [43]

Der CG–Algorithmus entsteht aus dem des Bi-CG unter Vernachlässigung aller mit einer Tilde bezeichneten Größen des dualen Problems, die wegen $A^T = A$ mit denen des Ausgangsproblems zusammenfallen.

Algorithmus 17 CG

$p_0 := r_0 := b - Ax_0$
$\rho_0 := (r_0, r_0)$
Für $k = 1, 2, \ldots :$
$q_{k-1} := Ap_{k-1}$
$\alpha_k := \frac{\rho_{k-1}}{(p_{k-1}, q_{k-1})}$
$x_k := x_{k-1} + \alpha_{k-1} p_{k-1}$
$r_k := r_{k-1} - \alpha_{k-1} q_{k-1}$
$\rho_k := (r_k, r_k)$
$\beta_k := \rho_k / \rho_{k-1}$
$p_k := r_k + \beta_k p_{k-1}$
Ende

Die Variablen α_k und β_k sind dabei von den gleichnamigen der Lanczos–Orthonormalisierung zu unterscheiden.

2.9.2 CGNR und CGNE

Das CG–Verfahren führt auf unmittelbare Weise auch zur Lösung nichtsymmetrischer Probleme, indem man es auf die *Normalengleichungen*

$$A^T A x = A^T b \tag{2.24}$$

oder

$$A \underbrace{A^T y}_{=:x} = b \tag{2.25}$$

anwendet, deren Koeffizientenmatrizen symmetrisch und positiv definit sind. Ansatz (2.24) führt zum CGNR–Verfahren (*Conjugate Gradient Normal Residual*), (2.25) zum CGNE–Verfahren (*Conjugate Gradient Normal Error*). Die Namen stammen aus der Tatsache, daß die Lösung der jeweiligen Normalengleichung gerade der Minimierung der 2–Norm des Residuums bzw. des Fehlers bezüglich des Ausgangsproblems (2.1) entspricht. Die von beiden Verfahren bei der Lösungssuche berücksichtigten Krylovräume sind identisch.
Zwar gelten nun die Konvergenzaussagen des CG, doch wegen

$$\mathrm{cond}_2\left(A^T A\right) = \mathrm{cond}_2\left(A A^T\right) = \|A^T A\|_2 \, \|(A^T A)^{-1}\|_2 = \|A\|_2^2 \, \|A^{-1}\|_2^2 = \mathrm{cond}_2^2(A)$$

ist für diese Verfahren im allgemeinen nur schlechte Konvergenz zu erwarten. Andererseits sind für das Beispiel einer orthogonalen Matrix A diese Verfahren optimal, da sie wegen $A^T A = A A^T = I$ die Lösung $x := A^T b$ sofort liefern.
Die Algorithmen entstehen durch Umordnung des CG–Verfahrens:

Algorithmus 18 CGNR

$r_0 := b - Ax_0$
$p_0 := z_0 := A^T r_0$
$\rho_0 := (z_0, z_0)$
Für $k = 0, 1, 2, \ldots$:
$w_k := Ap_k$
$\alpha_k := \frac{\rho_k}{(w_k, w_k)}$
$x_{k+1} := x_k + \alpha_k p_k$
$r_{k+1} := r_k - \alpha_k w_k$
$z_{k+1} := A^T r_{k+1}$
$\rho_{k+1} := (z_{k+1}, z_{k+1})$
$\beta_k := \frac{\rho_{k+1}}{\rho_k}$
$p_{k+1} := z_{k+1} + \beta_k p_k$
Ende

CGNR wird in der Regel als Lösungsverfahren vorgezogen, da die Konvergenzkontrolle anhand der Residuennorm durchgeführt wird, und daher deren Minimierung sinnvoller ist. Den Algorithmus des CGNE und weitere Verfahren zur Lösung der Normalengleichungen findet man in [47].

Kapitel 3

Vorkonditionierung linearer Systeme

In den bisherigen Darstellungen wurden wiederholt Konvergenzeigenschaften iterativer Verfahren zur Lösung eines linearen Gleichungssystems in Abhängigkeit von der Kondition der Matrix formuliert. Auch für Verfahren, zu denen keine solchen Aussagen existieren, deuten experimentelle Erfahrungen auf denselben Zusammenhang hin.
Um die Konvergenzeigenschaften der Lösungsverfahren zu verbessern, führt man deshalb eine *Vorkonditionierung* durch. Darunter versteht man eine invertierbare Abbildung $M : \mathbb{R}^n \to \mathbb{R}^n$, durch deren Anwendung das System in eines mit gleicher Lösung, aber besserer Kondition umgewandelt wird, bzw. allgemeiner in eines, das durch das Verfahren besser gelöst werden kann. Präkonditionierer können dabei an Schwächen des Verfahrens angepaßt werden. Auch entsprechende Modifikationen des Systems, die der Verbesserung der numerischen Stabilität des Verfahrens dienen, können als Vorkonditionierung bezeichnet werden. Man hat die als *links–*, *rechts–* oder *beidseitige Vorkonditionierung* bezeichneten Alternativen

$$M^{-1}Ax = M^{-1}b, \tag{3.1}$$

$$AM^{-1}u = b, \qquad u = Mx, \tag{3.2}$$

$$M_L^{-1}AM_R^{-1}u = M_L^{-1}b, \qquad u = M_R x. \tag{3.3}$$

Dabei soll $M_L M_R = M \approx A$ eine Näherung der Ausgangsmatrix sein, so daß die Matrix des vorkonditionierten Systems ähnlich zur Identität und damit gutkonditioniert ist.
Die explizite Berechnung der vorkonditionierten Matrix ist wegen des Rechen– und Speicheraufwandes (da sie in der Regel nicht schwachbesetzt ist) nicht durchführbar. Die in dieser Arbeit vorgestellten stationären Verfahren sind deshalb nicht vorkonditionierbar, weil sie direkt auf die Einträge der Matrix des zu lösenden Systems zugreifen. Es wird deshalb zuerst in Abschnitt 3.1 untersucht, wie sich Vorkonditionierung in den Algorithmen der PGK–Verfahren niederschlägt. Motiviert durch die daraus formulierten Anforderungen an die Vorkonditionierung werden im Anschluß verbreitete

Vorkonditionierungsmethoden dargestellt.
In diesem Rahmen kann jedoch kein umfassender Überblick der Theorie der Vorkonditionierung gegeben werden. Dafür sei zum Beispiel auf [47], [3] oder [24] verwiesen.

3.1 Vorkonditionierung von PGK–Verfahren

Um die vorkonditionierte Version eines Petrov–Galerkin–Krylov–Verfahrens zu erhalten, ersetzt man im entsprechenden Algorithmus jedes Auftreten von A, x und b durch die entsprechende Matrix bzw. den entsprechenden Vektor des vorkonditionierten Systems. Dadurch entstehende komplexere Berechnungen lassen sich durch Umstellungen zusammenfassen, so daß die Algorithmen letzlich nur geringen Änderungen unterliegen. Zu beachten ist, wie sich die Vorkonditionierung auf die durch den Algorithmus bestimmten Größen auswirkt. So bezeichnet zum Beispiel ein in einem Lösungsverfahren mit Links–Vorkonditionierung rekursiv berechnetes Residuum $r_k = M^{-1}(b - Ax_k)$ nunmehr das Residuum bezüglich des vorkonditionierten Systems statt zu (2.1).

3.1.1 Vorkonditionierung von GMRES

Da für GMRES mit (2.22) eine von $\text{cond}_2(A)$ abhängige Konvergenzabschätzung existiert, ist der Erfolg einer Konditionsverbessserung gewährleistet.
Die Links–Vorkonditionierung erweist sich als sehr einfache Modifikation; es ändern sich gegenüber Algorithmus 7 lediglich die beiden folgenden Berechnungen:

$$r_0 := M^{-1}(b - Ax_0)$$
$$\tilde{v}_k := M^{-1}Av_k$$

Hier tritt der erwähnte Fall ein, daß sich die gemäß 2.13 ermittelte Residuennorm auf das Ausgangssystem bezieht. Die Konvergenz kann also nur nach der Arnoldi–Orthonormalisierung überprüft werden. Da dann im Fall des GMRES(m) das tatsächliche Residuum $b - Ax$ zur Bestimmung des neuen r_0 ohnehin berechnet werden muß, entsteht dadurch jedoch kein höherer Aufwand.

Durch Rechts–Vorkonditionierung ändern sich in Algorithmus 7 die Zeilen

$$\tilde{v}_k := AM^{-1}v_k \tag{3.4}$$
$$x_k := x_0 + M^{-1}(V_k y_k) \tag{3.5}$$

Das hier berechnete Residuum ist gleich dem Residuum des Ausgangsproblems, da die Krylov–Basis auf dem tatsächlichen Residuum $r_0 := b - Ax_0$ errichtet wird. Diese Version erlaubt also gegenüber der Links–Version Konvergenzkontrolle während der Orthonormalisierung.

Die ausschließliche Anwendung von M^{-1} auf die Vektoren v_k ermöglicht eine flexible Vorkonditionierung mit von Schritt zu Schritt wählbaren M_k. Dafür speichert man die in (3.4) berechneten Vektoren $z_k := M^{-1}v_k$ und ersetzt mit $Z_k := (z_1, \ldots, z_k)$ die Berechnung (3.5) durch

$$x_k := x_0 + Z_k y_k.$$

Das dadurch definierte Verfahren heißt *FGMRES* (*Flexible GMRES*) und bietet unter erhöhtem Speicheraufwand mannigfaltige Möglichkeiten, Stagnation des GMRES(m) zu verhindern oder die Eigenschaften verschiedener Vorkonditionierer bei der Lösung eines einzelnen Problemes zu verbinden. Durch die Variation der M_k ist jedoch K_k im allgemeinen kein Krylov–Unterraum mehr, da die Suchrichtungen durch unterschiedliche Abbildungen definiert werden. Die Konvergenzeigenschaften des GMRES gelten also nicht mehr uneingeschränkt. In [47] werden einige entsprechend abgeschwächte Aussagen für FGMRES getroffen. Dort wird auch eine flexible Version des DQGMRES hergeleitet, die wegen der durch Updates formulierten Berechnung der Näherungslösung ohne die Speicherung zusätzlicher Vektoren auskommt.

Die beidseitige Vorkonditionierung des GMRES ist nur dann von Vorteil gegenüber den anderen Möglichkeiten, wenn A nahezu symmetrisch ist und diese Eigenschaft durch eine geeignete Zerlegung von M auf die vorkonditionierte Matrix übertragen werden kann. Man ersetzt dann in den wie oben geänderten Zeilen M durch M_L bzw. M_R.

3.1.2 Vorkonditionierung von CG

Der im vorangegangenen Abschnitt zuletzt angesprochene Punkt ist für PCG (*Preconditioned CG*) wesentlich. Da CG nur auf symmetrische und positiv definite Matrizen angewendet werden kann, muß die Vorkonditionierung diese Eigenschaften von A erhalten. Eine beidseitige Vorkonditionierung mit einer symmetrischen Zerlegung $M = M_L M_L^T$ erfüllt diesen Zweck. Man kann jedoch den Algorithmus, der durch Einsetzen dieser Matrizen und der entsprechenden Vektoren entsteht, so umstellen, daß lediglich M auftritt.
Auch linke und rechte Vorkonditionierung des CG lassen sich realisieren, indem man im CG–Algorithmus das euklidische Skalarprodukt durch $(\cdot,\cdot)_M$ und $(\cdot,\cdot)_{M^{-1}}$ ersetzt. Es stellt sich heraus, daß alle drei Methoden mathematisch identisch sind und zu folgendem Algorithmus führen:

Algorithmus 19 PCG

$r_0 := b - Ax_0$
$p_0 := z_0 := M^{-1}r_0$
Für $k = 0, 1, 2, \ldots$:
$\alpha_k := (r_k, z_k)/(Ap_k, p_k)$
$x_{k+1} := x_k + \alpha_k p_k$
$r_{k+1} := r_k - \alpha_k Ap_k$
$z_{k+1} := M^{-1}r_{k+1}$
$\beta_k := (r_{k+1}, z_{k+1})/(r_k, z_k)$
$p_{k+1} := z_{k+1} + \beta_k p_k$
Ende

Nach Satz 2.23 kann man für PCG gegenüber CG verbesserte Konvergenz erwarten.

3.1.3 Vorkonditionierung von PGK–Verfahren im allgemeinen

Nach demselben Schema lassen sich alle PGK–Verfahren vorkonditionieren. Links–vorkonditionierte Versionen der meisten der in dieser Arbeit vorgestellten Lösungsverfahren findet man in [6] und von CGNR und CGNE in [47]. Dort werden auch einige vorkonditionierte Verfahren für spezielle Fälle formuliert.
Schließlich bleiben in den vorkonditionierten Algorithmen an einigen Stellen Produkte der Form

$$s := \tilde{M}^{-1}t \tag{3.6}$$

zu berechnen, wobei $\tilde{M}$ für M, M_L, M_R oder deren Transponierte und s und t für die entsprechenden Vektoren stehen. Ist $\tilde{M}^{-1}$ nicht bekannt (das ist die Regel), wird s als Lösung des Systems

$$\tilde{M}s = t. \tag{3.7}$$

bestimmt. Der Vorkonditionierer $\tilde{M}$ sollte also so gewählt werden, daß (3.7) unter geringem Aufwand gelöst werden kann. Dieser Mehraufwand ist gegen den Aufwand der durch die Konvergenzbeschleunigung eingesparten Iterationen abzuwägen. Ein Vorkonditionierer darf also teuer sein, wenn er entsprechend leistungsfähig ist.

3.2 Vorkonditionierer

Nun sind die Eigenschaften bekannt, die ein Vorkonditionierer erfüllen muß. Als Vorkonditionierer M ist jede invertierbare Abbildung $M : \mathbb{R}^n \to \mathbb{R}^n$ verwendbar, die

ähnlich zu A ist, mit der aber gleichzeitig (3.7) einfach lösbar oder (3.6) einfach berechenbar ist, insbesondere wesentlich einfacher als das Ausgangsproblem.
Zur Beurteilung eines Vorkonditionierers ist eine Aufwandsabschätzung erforderlich. Auf der einen Seite steht die Ersparnis, die durch die schnellere Konvergenz erreicht wird, auf der anderen Seite stehen die zusätzlichen Kosten, die einmal bei der Konstruktion von M und bei jeder Iteration beim Lösen von (3.7) anfallen. Erst diese Bilanz zeigt, ob ein Vorkonditionierer nutzbringend ist.
Auch der Speicheraufwand kann eine Rolle bei der Wahl spielen, da einige Vorkonditionierer M explizit konstruieren, andere (3.7) allein durch die Kenntnis von A effizient lösen können. Beispiele für Vorkonditionierer sind Matrizen, die explizit oder implizit durch A gegeben sind, ein oder mehrere Schritte eines iterativen Lösungsverfahrens oder eines Mehrgitterverfahrens oder aber auch die aus Kapitel 1 bekannten Gebietszerlegungsverfahren.

3.2.1 Jacobi, SOR und SSOR

Wie in Abschnitt 2.2 beschrieben wurde, sind die stationären Verfahren durch eine Zerlegung $A = M - N$ mit einer nichtsingulären Matrix M definiert. Die Iterationsvorschrift

$$\begin{aligned} x_{k+1} &:= M^{-1}Nx_k + M^{-1}b \\ &= M^{-1}(M - A)x_k + M^{-1}b \\ &= (I - M^{-1}A)x_k + M^{-1}b \end{aligned}$$

sucht die Lösung von

$$\begin{aligned} & (I - (I - M^{-1}A))x = M^{-1}b \\ \Leftrightarrow \qquad & M^{-1}Ax = M^{-1}b. \end{aligned}$$

Die stationären Verfahren definieren also linksseitige Vorkonditionierer, deren Qualität davon abhängt, wie gut A durch M approximiert wird. Die Vorkonditionierungsmatrizen sind wie in Abschnitt 2.2 gegeben durch

$$\begin{aligned} M_{Jacobi} &:= D, \\ M_{SOR}(\omega) &:= U + \frac{1}{\omega}D, \\ M_{SSOR}(\omega) &:= (D + \omega U)D^{-1}(D + \omega O), \end{aligned}$$

und die Lösung von (3.7) entspricht dann einer Iteration des jeweiligen stationären Verfahrens mit dem Startvektor $x_0 = 0$.
Für SOR und den Vorwärtsdurchlauf des SSOR entfällt damit die Hälfte des Rechenaufwandes. Da sich beim Rückwärtsdurchlauf des SSOR, wie beim SSOR–Lösungsverfahren beschrieben, ebenfalls die Hälfte einsparen läßt, entsprechen die

Kosten eines SSOR–Vorkonditionierungsschrittes denen zweier SOR–Schritte und damit wiederum im wesentlichen einer Matrix–Vektor–Multiplikation.
Der Vorteil dieser Vorkonditionierer ist, daß M durch A implizit gegeben ist und nicht explizit gespeichert werden muß. Allerdings speichert man die Inverse der Diagonalen von A, insbesondere wenn A eine echte Blockmatrix ist. Entsprechend gering und vernachlässigbar ist der Aufwand zur Konstruktion des Vorkonditionierers.
M_{Jacobi} und M_{SSOR} sind symmetrisch, daher zur Vorkonditionierung in Lösungsverfahren für symmetrische Matrizen geeignet. SOR dagegen darf z.B. in CG nicht angewandt werden. Da sich Symmetrie oder Fast–Symmetrie von Matrizen häufig vorteilhaft auf das Konvergenzverhalten der Verfahren (die ja im gewissen Sinne Erweiterungen des CG sind) auswirkt, ist SOR als Vorkonditionierer meist weniger geeignet, zumal seine Leistung stark von der Matrixstruktur abhängt.
Der Jacobi–Vorkonditionierer stellt im allgemeinen keine gute Näherung von A dar und bringt daher, wenn überhaupt, nur mäßige Konvergenzverbesserung. Die Qualität von SOR und SSOR als Vorkonditionierer ist dagegen, wie anhand der Lösungsverfahren beschrieben, stark von der Struktur der Matrix und dem gewählten Relaxationsparameter abhängig.

3.2.2 ILU

Die *ILU*–Vorkonditionierung (*Incomplete LU*) verwendet eine unvollständige LR–Zerlegung $M = M_L M_R \approx$ von A, um durch Vorwärts– und Rückwärtseinsetzen in L und R die Lösung von (3.7) zu bestimmen. Die Zerlegung ist insofern unvollständig, als von den während des Gaußschen Eliminationsverfahrens berechneten Einträgen von M_L und M_R bestimmte nicht gespeichert werden. Anderenfalls wäre die Zerlegung zwar exakt, doch bestünde die Gefahr, daß durch zu viele neue Einträge die Anzahl der Nichtnullelemente in den Bereich $\mathcal{O}(n^2)$ steigt. Ist R die Matrix, die alle vernachlässigten Einträge enthält, so ist

$$A = M + R$$

Entscheidend ist die Frage, wie viele und welche der neu entstehenden Einträge vernachlässigt werden sollen. Werden nur an den Positionen in M_L und M_R Einträge zugelassen, wo A Einträge besitzt, und alle weiteren ignoriert, so erhält man *ILU(0)*.
Bei der Anwendung des ILU(0) auf zeilenweise abgespeicherte Sparse–Matrizen ist die direkte Übertragung des gewöhnlichen Gauß–Verfahrens wegen der darin auftretenden Spaltendurchläufe ungünstig. Durch Umsortieren der Rechenschritte (Herleitung in [47]) und unter Verwendung der in Abschnitt 2.2 eingeführten Terminologie der Blockmatrizen erhält man folgenden, geeigneteren Algorithmus, der die Matrix A wie im Gaußschen Eliminationsverfahren mit M_L und M_R überschreibt:

Algorithmus 20 ILU(0)

Für $i = 2, \ldots, n$:
  Für $j = 1, \ldots, i-1$ und $A_{i,j} \not\equiv 0$:
    $A_{i,j} := (A_{j,j})^{-1} A_{i,j}$
    Für $k = j+1, \ldots, n$ und $A_{i,k} \not\equiv 0$:
      $A_{i,k} := A_{i,k} - A_{i,j} A_{j,k}$
    Ende
  Ende
Ende

Die Bezeichnung $A_{i,j} \not\equiv 0$ bedeutet, daß der Block $A_{i,j}$ mindestens einen von Null verschiedenen Eintrag besitzt. Das Verfahren kann fehlschlagen, wenn einer der entstehenden Diagonalblöcke $A_{j,j}$ nicht invertierbar ist, bzw. für die Blockdimension $l = 1$, falls $A_{j,j} = 0$. Dies kann trotz der Nichtsingularität von A eintreten, da keine Pivotisierung durchgeführt wird und die Zerlegung nicht exakt ist.

Die durch ILU(0) erzeugte Zerlegung ist, falls Einträge vernachlässigt wurden, nicht eindeutig. Durch Änderungen in den nichtnullwertigen Einträgen von M_L und M_R entstehen verschiedene Auslassungsmatrizen R und damit verschieden gute Approximationen an A.
Ein Ansatz in dieser Richtung ist *MILU* (*Modified ILU*), das die Werte der zu vernachlässigenden Blöcke der Zerlegung ebenfalls berechnet und sie vom Diagonalblock derselben Zeile subtrahiert. Dadurch wird erreicht, daß die Zeilensummen von R nullwertig sind. Andere Varianten subtrahieren ein skalares Vielfaches der Blöcke mit einem Faktor $0 < \alpha < 1$ (*relaxierter ILU*). In [4] findet man einen Überblick über die Literatur zu diesen Varianten des ILU.

Ein anderer Weg, A durch M genauer zu approximieren, ist die Zulassung von zusätzlichen Einträgen in M_L und M_R. *ILU(p)* bestimmt *Füllgrade* (*levels of fill*) der Einträge. Ein Eintrag wird bei der Zerlegung vernachlässigt, wenn sein Füllgrad größer als p ist. Die Füllgrade werden dadurch bestimmt, daß anfangs alle nichtnullwertigen Einträge den Grad 0 und die übrigen den Grad ∞ besitzen, und daß bei der Berechnung des Eintrages $A_{i,k}$ dieser den Grad $\min\{Grad(A_{i,k}), Grad(A_{i,j}) + Grad(A_{j,k}) + 1\}$ erhält. Dadurch werden nur Einträge bis zur p–ten „Generation“ zugelassen.
Bei diesem Vorgehen kann im allgemeinen jedoch schon für $p = 1$ eine vollbesetzte Matrix entstehen. Erst für Matrizen bestimmter Struktur ist gewährleistet, daß die Anzahl der neuen Einträge im Bereich $\mathcal{O}(n)$ liegt. Auf diese kann man ILU(p) angewenden, wobei sich auch der ILU–Algorithmus an die Struktur der Matrix anpassen läßt (siehe [47]).

Wesentlich leistungsfähiger als die bisher vorgestellten Versionen des ILU sind Ansätze, die sich bei der Zulassung neuer Einträge nicht an der Struktur der Matrix, sondern an der numerischen Bedeutung der Einträge, also an der Größe ihres Betrages orientieren.

Dies erfordert zur Implementation jedoch komplexe Datenstrukturen, da die Struktur von M_L und M_R erst bei der Zerlegung bestimmt wird.
Durch *ILUT(p,τ)* werden in der i–ten Zeile $A_{i,*}$ der Zerlegung außer dem Diagonalelement alle Elemente vom Betrag kleiner als $\tau\|A_{i,*}\|$ ignoriert, wobei $\|\cdot\|$ eine geeignete Norm ist. Danach werden in der i–ten Zeile von M_L und M_R nur das Diagonalelement und jeweils die p betragsgrößten Einträge behalten und die übrigen vernachlässigt.
Um ILUT zu verwenden, ist eine (z.B. die in [47] beschriebene) effiziente Implementation wichtig, da sonst die wegen der Komplexität des Verfahrens hohen Kosten der Zerlegung den beim Lösen erreichten Nutzen übersteigen.
Durch Pivotisierung des ILUT zur Verbesserung der numerischen Stabilität entsteht ILUTP. Wegen der, wie angenommen, zeilenweisen Speicherung der Matrix verwendet man dafür eine Spaltenpivotsuche. Eine Beschreibung der Implementation dieses Verfahrens mit im wesentlichen gleichen Aufwand wie ILUT findet man in [47].

Kapitel 4

Experimentelle Untersuchungen

In diesem Kapitel werden die in den Kapiteln 2 und 3 vorgestellten Lösungs– und Vorkonditionierungsverfahren auf Systeme angewendet, die durch die in Kapitel 1 erläuterten Methoden aus repräsentativen Beispielen von Konvektions–Diffusions–Reaktions–Problemen entstehen.
Nach der Definition der Modellprobleme werden die Versuchsdurchführung und die verwendete Rechenumgebung beschrieben. Insbesondere wird das Softwarepaket *BLANC* charakterisiert, das parallel zu dieser Arbeit entwickelt wurde, um einerseits die nachfolgenden Experimente durchzuführen, andererseits, um eine universelle Basis für weitere Projekte bereitzustellen, die sich auf schwachbesetzte und/oder blockstrukturierte Matrizen beziehen.
Es folgt die Untersuchung der stationären Verfahren SOR und SSOR in Hinblick auf optimale Relaxation und ihren Einsatz als Vorkonditionierung, des GMRES(m) bei unterschiedlichen Restartintervallen und der in dieser Arbeit vorgestellten PGK–Verfahren (inklusive des GMRES(m) mit optimalem Restart). Abschließend werden die Ergebnisse bewertet und Schlußfolgerungen gezogen.

4.1 Darstellung der Modellprobleme

Die Modellprobleme sind jeweils auf $\Omega :=]0,1[^2 \subset \mathbb{R}^2$ gestellt. Es soll untersucht werden, wie sich singuläre Störungen durch Konvektionsdominanz auf mathematischer Seite und unterschiedlich genaue Zerlegungen auf numerischer Seite in der Komplexität des aus der Galerkin–Least–Squares–Diskretisierung entstehenden Gleichungssystemes niederschlagen.
Für den ersten Punkt wird der Diffusionskoeffizient jeweils über $\{1, 10^{-2}, 10^{-4}, 10^{-6}\}$ variiert. Die Zerlegung wird auf dem Einheitsquadrat durch ein regelmäßiges 32×32–, 64×64– oder 128×128–Schachbrettgitter über Ω realisiert, bei dem jedes Feld diagonal halbiert ist. Das entstehende Gleichungssystem besitzt entsprechend die Größe

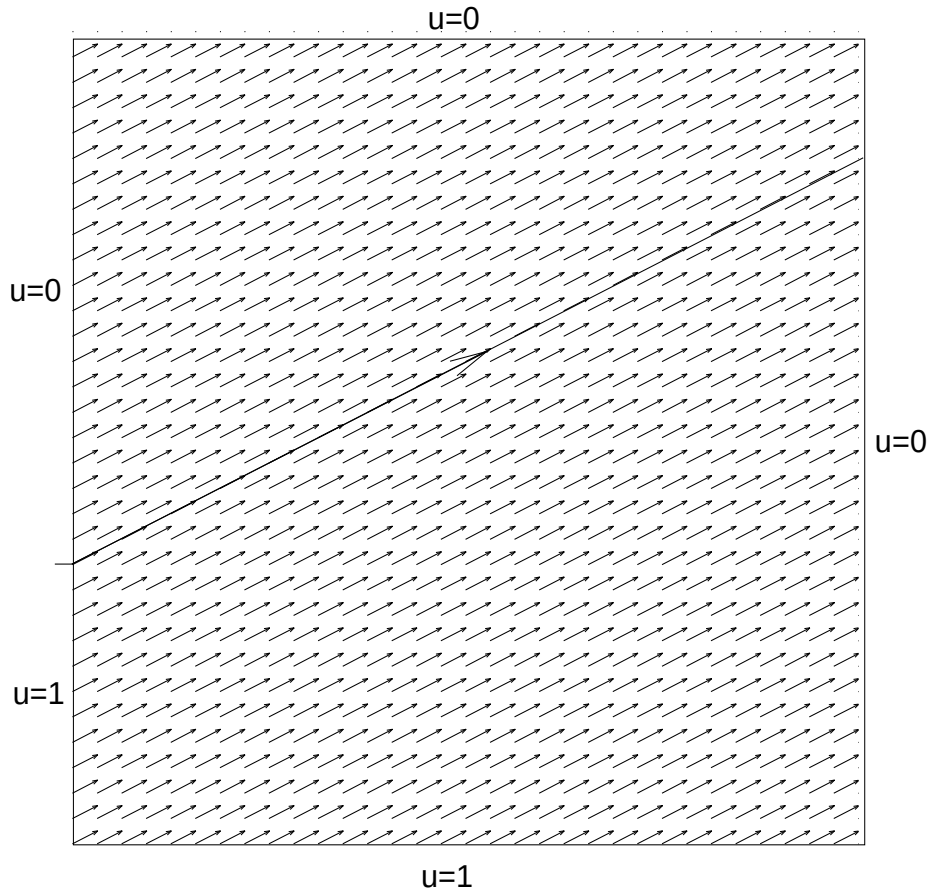


Abbildung 4.1: Schrägströmung

$n = 33 \cdot 33 = 1089$, $n = 65 \cdot 65 = 4225$ bzw. $n = 129 \cdot 129 = 16641$. Die Knoten seien lexikographisch, d.h. zeilenweise von links nach rechts und von unten nach oben numeriert.
Effekte durch Reaktionsdominanz oder die Verwendung einer Gebietszerlegung werden nur an jeweils einem Beispiel betrachtet; die Problemstellung wird dort beschrieben.

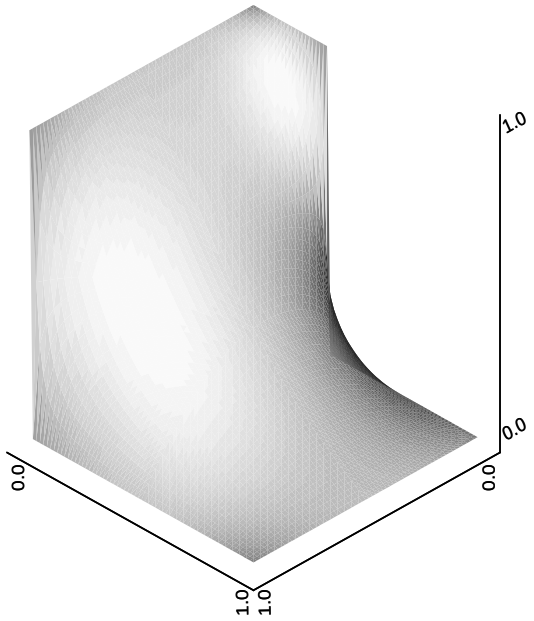


Abbildung 4.2: $\varepsilon = 10^{0}$

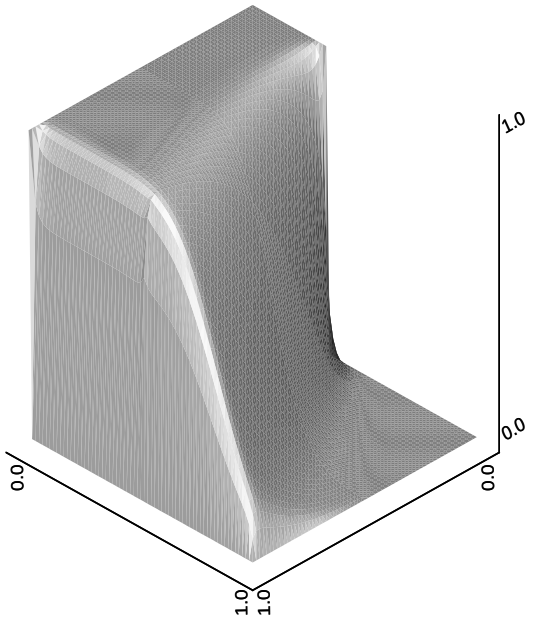


Abbildung 4.3: $\varepsilon = 10^{-2}$

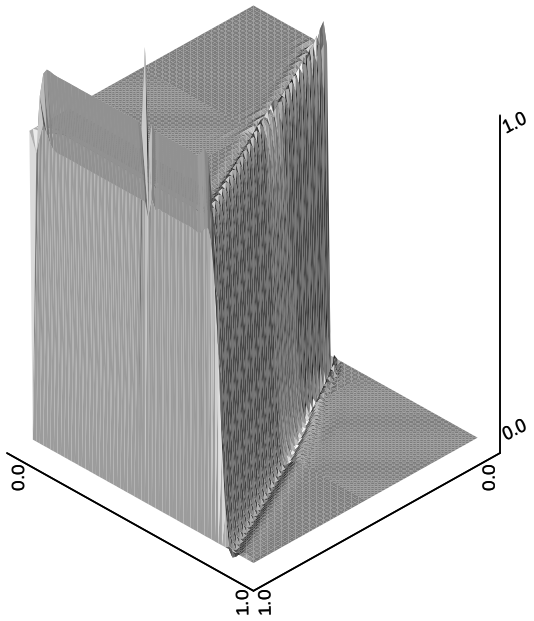


Abbildung 4.4: $\varepsilon = 10^{-4}$

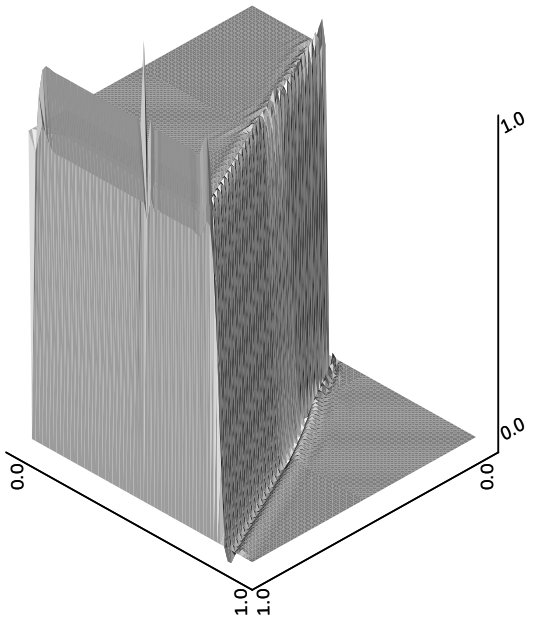


Abbildung 4.5: $\varepsilon = 10^{-6}$

4.1.1 Schrägströmung

Das Geschwindigkeitsfeld b sei mit $b \equiv \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ konstant gewählt (damit ist $\|b\|_2 = 1$). $\partial\Omega$ zerfällt damit in

$$\begin{aligned} \partial\Omega^+ &= \{(\, x \quad 1\,)^T, (\, 1 \quad y\,)^T \;:\; 0 \le x, y \le 1\}, \\ \partial\Omega^0 &= \emptyset, \\ \partial\Omega^- &= \{(\, x \quad 0\,)^T, (\, 0 \quad y\,)^T \;:\; 0 \le x, y \le 1\}. \end{aligned}$$

Auf dem Rand seien folgende Dirichlet–Bedingungen gegeben:

$$\begin{aligned} u = 1 \quad &\text{auf} \quad \left\{(\, x \quad 0\,)^T,\, (\, 0 \quad y\,)^T \;:\; 0 \le x \le 1,\; 0 \le y \le \frac{1}{3}\right\}, \\ u = 0 \quad &\text{sonst.} \end{aligned}$$

Auf dem Einströmrand befindet sich in $x^* := \left(\, 0 \quad \frac{1}{3}\,\right)^T$ eine Unstetigkeit in den Randbedingungen. Entlang ξ_{x^*} entsteht also eine parabolische innere Grenzschicht. Weiterhin sind wegen der Dirichlet–Randbedingungen Grenzschichten am Ausströmrand zu erwarten.
Das Problem sei reaktionsfrei, also $c, f \equiv 0$.
Durch diese Daten simuliert man z.B. den Wärmetransport über eine quadratische Fläche durch eine Strömung in Richtung $(\, 2 \quad 1\,)^T$ mit unstetigem Einströmrand. Dabei laufen im Fluid keine Reaktionen ab, es nimmt keine Wärmeenergie auf und setzt auch keine frei.
In Abbildung 4.1 ist das Geschwindigkeitsfeld b mit den Randbedingungen und der inneren Grenzschicht dargestellt, in den Abbildungen 4.2–4.5 die Entwicklung der Lösung für $\varepsilon \to 0$, an der man die Entstehung der erwarteten Grenzschichten verfolgen kann. Im singulär gestörten Fall ist die im ersten Kapitel angesprochene Entstehung von Oszillationen quer zur inneren Grenzschicht beobachtbar, die der Galerkin–Einfluß hervorruft. Dabei liegt Ω in der x–y–Ebene, die Werte von u werden in z–Richtung aufgetragen.
Es werden zwei Variationen dieses Problems behandelt:

- Da in diesem Problem die Strömung einheitlich orientiert ist, wird sich dies auch in der Struktur der Matrix niederschlagen. Lösungsverfahren und Vorkonditionierer, die dadurch beeinflußt werden, von den in dieser Arbeit vorgestellten also SOR und SSOR, sind theoretisch in ihrer Leistungsfähigkeit von der Numerierungsreihenfolge der Knoten abhängig. Um diesen Effekt zu untersuchen, werden diese Verfahren auch auf die Matrix angewendet, die aus der Diskretisierung des Problems unter geänderter Numerierung quer zur Strömungsrichtung entsteht. Die Knoten sind dann spaltenweise von oben nach unten und von rechts nach links durchnumeriert.

- Es soll untersucht werden, wie sich die Art der Randbedingungen auf die Komplexität der entstehenden Matrix auswirkt. Dafür wird Ω als Teilgebiet unter einer adaptiven Gebietszerlegung vom Nataf/Rogier–Typ gewählt. Die Randstücke

$$\Gamma_1 := \{(\, 0 \quad y \,)^T \,:\, 0 \le y \le 1\},$$
$$\Gamma_2 := \{(\, 1 \quad y \,)^T \,:\, 0 \le y \le 1\}$$

  werden als Interfaces zu benachbarten Teilgebieten betrachtet, so daß auf diesen die in Abschnitt 1.3 formulierten Robin–Randbedingungen gestellt sind. Das Problem wird dabei auf dem gesamten Gebiet so definiert, daß dessen reduzierte Formulierung auf Ω mit der reduzierten Formulierung des Ausgangsproblems übereinstimmt.

4.1.2 Rotationsströmung

In diesem Beispiel sei das Geschwindigkeitsfeld b bestimmt durch

$$b(x,y) := \begin{pmatrix} (2y-1)(1-(2x-1)^2) \\ 4y(2x-1)(y-1) \end{pmatrix}.$$

Dadurch ist eine Rotationsströmung um den Mittelpunkt von Ω gegeben, die am Rand $\partial\Omega$ parallel zu diesem verläuft. Damit ist $\partial\Omega = \partial\Omega^0$, und alle Charakteristiken sind geschlossen.
Auf dem Rand seien folgende Dirichlet–Bedingungen gegeben:

$$\begin{aligned} u &= -0.5 \quad \text{auf} \quad \left\{(\, 0 \quad y \,)^T \,:\, 0 \le y \le 1\right\} \\ u &= \quad 0.5 \quad \text{auf} \quad \left\{(\, 1 \quad y \,)^T \,:\, 0 \le y \le 1\right\} \\ u &= \quad\; 0 \quad \text{sonst} \end{aligned}$$

Abbildung 4.6 illustriert diese Daten. Sei wieder $f \equiv 0$.
An diesem Beispiel soll der Einfluß des Reaktionstermes untersucht werden. Dafür wird der Reaktionskoeffizient c jeweils konstant als $c \equiv \tilde{c} \in \{0, 1, 10, 100\}$ gewählt, damit die Entwicklung über das Spektrum vom reaktionsfreien bis zum reaktionsdominanten Fall erkennbar wird.
Man kann sich dieses Problem als Querschnitt durch einen Raum vorstellen, der auf

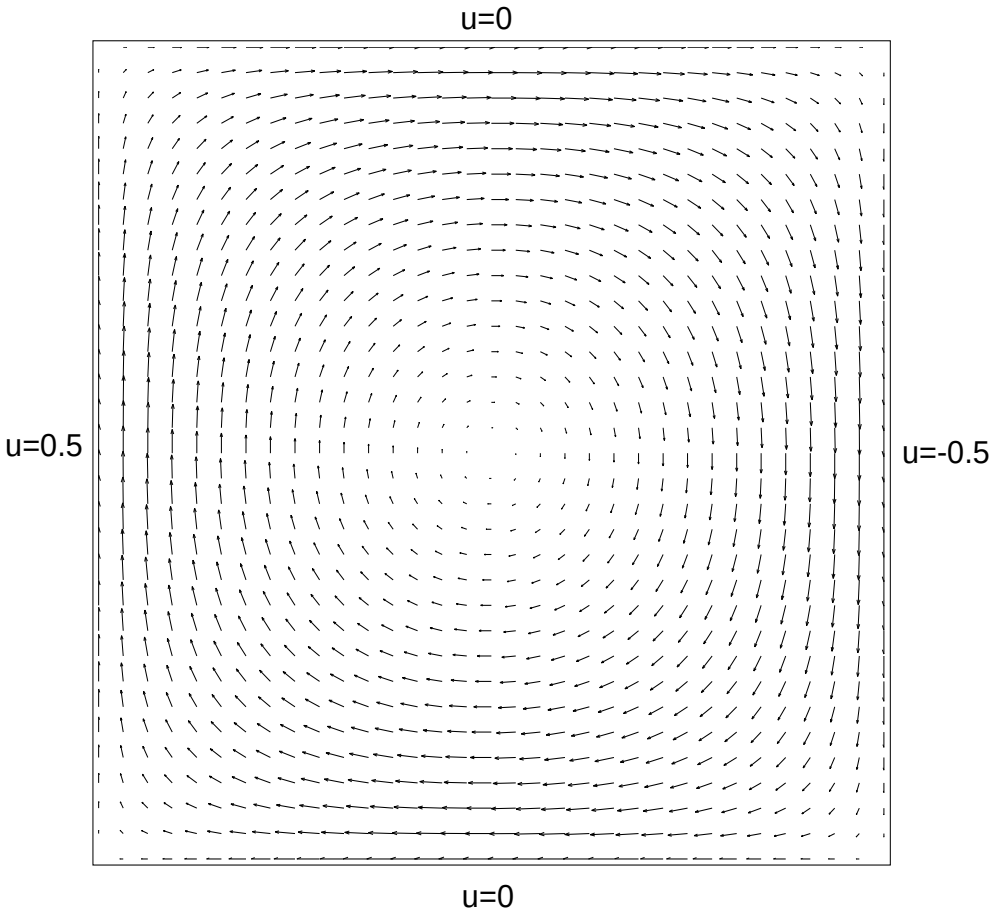


Abbildung 4.6: Rotationsströmung

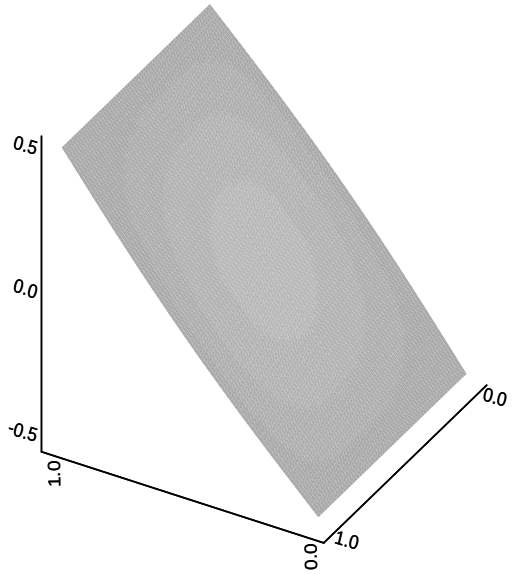


Abbildung 4.7: $\varepsilon = 10^{0}$, $c = 0$

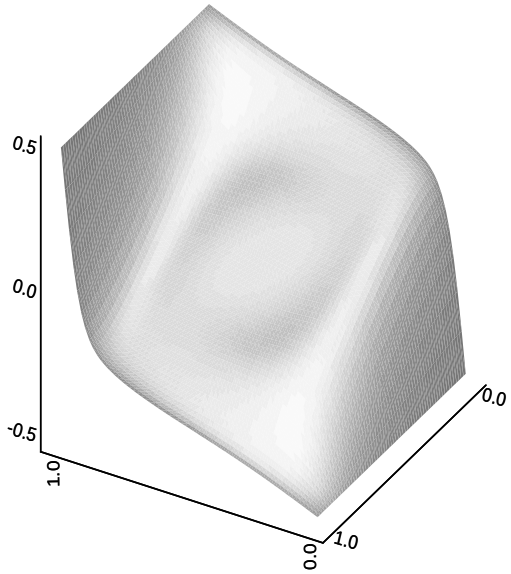


Abbildung 4.8: $\varepsilon = 10^{-2}$, $c = 0$

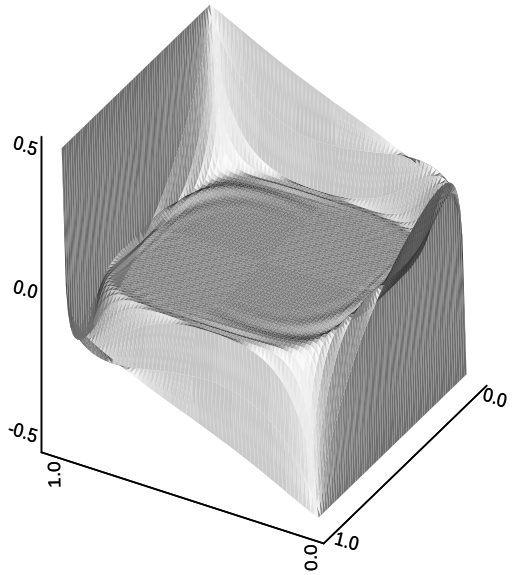


Abbildung 4.9: $\varepsilon = 10^{-4}$, $c = 0$

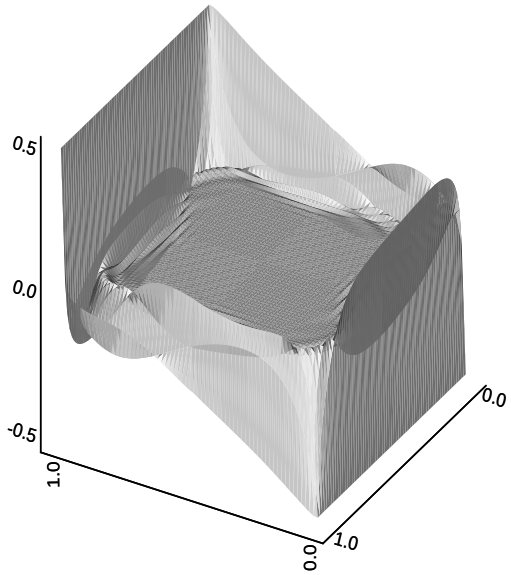


Abbildung 4.10: $\varepsilon = 10^{-6}$, $c = 0$

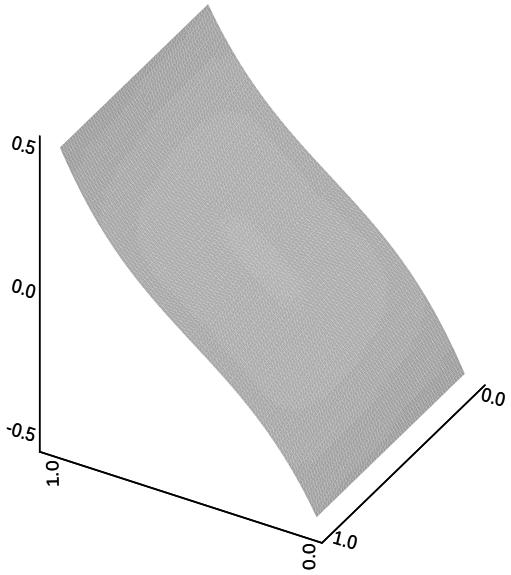


Abbildung 4.11: $\varepsilon = 10^{0}$, $c = 10$

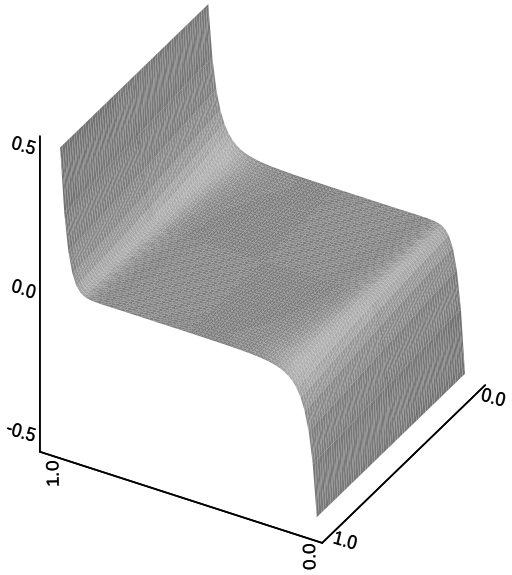


Abbildung 4.12: $\varepsilon = 10^{-2}$, $c = 10$

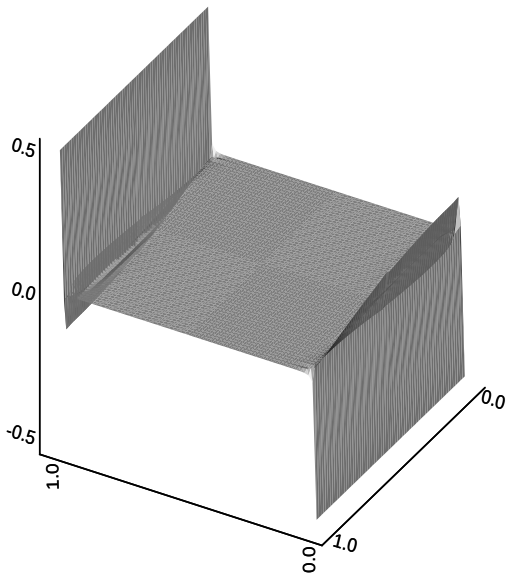


Abbildung 4.13: $\varepsilon = 10^{-4}$, $c = 10$

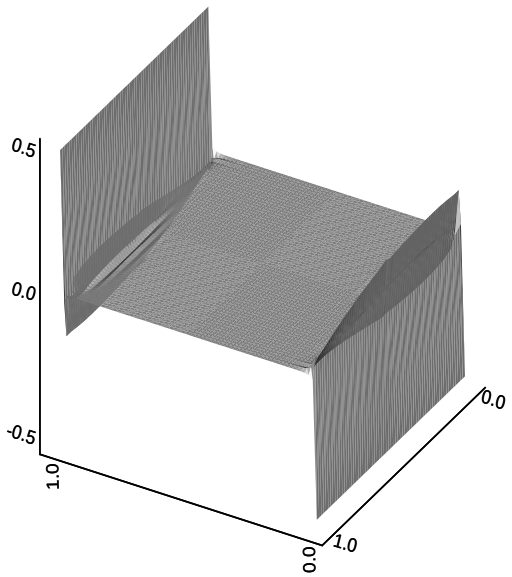


Abbildung 4.14: $\varepsilon = 10^{-6}$, $c = 10$

einer Seite beheizt und auf der anderen gekühlt wird. Durch das Geschwindigkeitsfeld wird entsprechende physikalische Konvektion durch Temperaturgefälle simuliert. Für $\varepsilon \to 0$ wird die Wärmeleitung im Fluid bedeutungslos, der Temperaturtransport erfolgt dann fast nur noch durch die Strömung, also entlang des Randes.
Dieses Verhalten ist in den Abbildungen 4.7–4.10 im reaktionsfreien Fall ($c \equiv 0$) graphisch dargestellt. Man erkennt, daß im singulär gestörten Fall kaum Transport quer zu den Charakteristiken ins Innere von Ω, aber starker Transport am Rand entlang der Charakteristiken stattfindet. Den Reaktionseinfluß kann man sich als endotherme chemische Reaktion vorstellen, die im reaktionsdominanten Fall die durch Konvektion und Diffusion verteilte Wärmeenergie aufnimmt.

4.2 Durchführung der Versuche

Um die Qualität der Lösungsverfahren miteinander zu vergleichen, wurde bei den numerischen Experimenten der zeitliche Konvergenzverlauf betrachtet, also die Entwicklung der Konvergenz $\|r_k\|_2/\|r_0\|_2$ des Residuums in der Euklidischen Norm bezüglich der Rechenzeit. In vielen Darstellungen, die Konvergenzresultate für iterative Verfahren formulieren, werden die Iterationen des Verfahrens gegen die Konvergenz aufgetragen. Dies läßt jedoch keinen sinnvollen Vergleich verschiedener Verfahren zu, da sich diese in dem pro Iteration erforderlichen Rechenaufwand eventuell signifikant unterscheiden. Bei der Bewertung der PGK–Verfahren wird zur Illustration eine entsprechende Übersicht der Kosten pro Iteration gegeben.
Es wurden jeweils die rechte Seite $b := 0$ und die Startlösung $x_0 := \frac{1}{\sqrt{n}}(1, \ldots, 1)^T$ gesetzt, so daß wegen $\|x_0\|_2 = 1$ der relative Fehler einer Näherungslösung x_k durch $\|x_k\|$ gegeben war. Dadurch konnte bei den Experimenten neben der Konvergenz bezüglich des Residuums auch die Entwicklung des relativen Fehlers beobachtet werden. Auf entsprechende Darstellung wird aus Platzgründen verzichtet; die daraus gewonnenen Erkenntnisse werden später erwähnt werden.
Verglichen wurde jeweils die Rechenzeit, die von dem Verfahren benötigt wurde, bis Konvergenz der Residuennorm auf $\|r_k\|_2/\|r_0\|_2 < 10^{-2}$ bzw. auf $\|r_k\|_2/\|r_0\|_2 < 10^{-6}$ erreicht wurde. Während die zweite Schranke beim Einsatz von Lösungsverfahren als solche als Abbruchkriterium üblich ist, tritt die erste eventuell beim Einsatz der Verfahren in Mehrgittermethoden auf. Wenngleich diese hier nicht behandelt werden, sollen doch Darstellungen für entsprechende Untersuchungen bereitgestellt werden. Die Konvergenzverläufe aller im folgenden betrachteten Kombinationen von Lösern und Vorkonditionierern sind für die Ausgangsprobleme der Schrägströmung und der Rotationsströmung im Anhang zu finden.

Als Rechenumgebung für die numerischen Experimente diente eine DEC Alpha 3000/600.
Die Diskretisierungen der Probleme wurden mit dem von Andreas Auge entwickelten Programm *PNS* (*Parallelized solution of Navier–Stokes–equations*) durchgeführt.

Es ist ein Finite–Elemente–Code zur Behandlung von Navier–Stokes–Gleichungen, der auf der adaptiven nichtüberlappenden Gebietszerlegungsmethode nach Nataf/Rogier und der Galerkin–Least–Squares–Diskretisierung aufbaut.
Zur Untersuchung der Löser und Vorkonditionierer wurde vom Verfasser im Rahmen dieser Arbeit die C–Bibliothek *BLANC* (*Blockwise Linear Algebra and Numerical Computations in C*) erstellt, die auch in PNS verwendet wird. Es soll nun eine kurze Übersicht über Aufbau und Eigenschaften des Paketes gegeben werden.

4.3 Die Funktionsbibliothek BLANC

BLANC ist eine Sammlung von Datentypen und Funktionen, die zum einen eine Oberfläche für objektorientierten Umgang mit (schwachbesetzten) Blockmatrizen und Blockvektoren bietet, zum anderen auf effiziente Weise Grundoperationen zwischen diesen und einen großen Teil der vorgestellten Lösungsverfahren für lineare Gleichungssysteme realisiert. Gewöhnliche Matrizen und Vektoren sind als Spezialfälle der blockstrukturierten in der Darstellung enthalten.
Die konsequente Berücksichtigung der Blockstruktur ist dabei das wesentliche Merkmal von BLANC, durch das es sich von den verfügbaren Bibliotheken zur Behandlung linearer Systeme unterscheidet. Vorteil dieses Ansatzes ist eine erhebliche Reduzierung im Speicherbedarf sowie die natürliche Realisierbarkeit von Blockalgorithmen (z.B. SOR/SSOR, ILU).
Außerdem wurde nach einschlägigen Erfahrungen mit bereits existierenden Bibliotheken großer Wert auf Robustheit des Codes gelegt und eine — bis auf unvermeidbare Risiken wie Overflow — umfassende Fehlerdiagnose implementiert. Diese ist jedoch auf Wunsch stufenweise abschaltbar, so daß dennoch optimale Performance erreichbar bleibt.
Durch die vollständige Einkapselung der benutzten Strukturen, verbunden mit der Möglichkeit der Rückkopplung an den Benutzer durch die Fehlerkontrolle, steht mit diesem Paket ein black–box–artiges Produkt zur Verfügung, das auch dem unerfahrenen Benutzer einfaches und übersichtliches Bearbeiten blockstrukturierter Probleme und leistungsfähige Rechnungen ermöglicht.
BLANC und eine ausführliche Dokumentation werden in Kürze über den FTP–Server des Instituts für Numerische und Angewandte Mathematik an der Internet–Adresse „http://www.num.math.uni-goettingen.de“ verfügbar gemacht werden.
Die Bibliothek gliedert sich in die Module

- ***blanc_info***, das die Stringkonstanten ***blanc_VERSION*** (Versionsnummer), ***blanc_DATE*** (Datum der Übersetzung) und ***blanc_INFO*** (gesamte Beschreibung des vorliegenden Exemplares) enthält.

- ***blanc_error***, das die Fehlerbehandlung verwaltet. Dafür dient der Typ ***blanc_Error_status***, der jeden möglichen Fehler beschreibt und anhand dessen

Wert also auch Korrekturen bei Laufzeit möglich sind. Die Ausgabe der Fehlermeldungen kann leicht auf andere Ausgabeströme umgelenkt werden.

- ***blanc_basics***, das die primitiven Datentypen ***blanc_Real*** (reeller Wert) und ***blanc_Boolean*** (Wahrheitswert) enthält. Insbesondere sind hier die in BLANC verwendeten nichttrivialen Operationen auf *blanc_Real* definiert, so daß in Abhängigkeit von Architektur und erwünschter Rechengenauigkeit leicht die gesamte BLANC–Arithmetik auf den jeweils erwünschten float–Grundtyp umgestellt werden kann.

- ***blanc_vector***, das die Datentypen ***blanc_Vector*** und ***blanc_Velt*** (*vector element*) verwaltet, die einen Blockvektor bzw. einen Vektorblock beschreiben. Außerdem steht der Typ ***blanc_Vcont*** (*vector container*) zur Verfügung, der als Verallgemeinerung des Vektorbegriffs einen *blanc_Vector*, ein *blanc_Velt* oder einen *blanc_Real* enthält. Diese vektoriellen Typen sind Objekte vom Typ ***blanc_Vtype***.

- ***blanc_voperats***, das Grundoperationen der linearen Algebra zwischen Vektoren umfaßt. Darunter fallen neben Multiplikation mit einem Skalar, Addition, Subtraktion, Skalarprodukt, komponentenweiser Multiplikation und Division und komponentenweisem Betrag auch eine Reihe von Normauswertungen: $\|\cdot\|_1$, $\|\cdot\|_2$, $\|\cdot\|_\infty$, $\|\cdot\|_{L^1}$, $\|\cdot\|_{L^2}$, $\|\cdot\|_{L^\infty}$, sowie für echte Blockvektoren die entsprechenden Blocknormen, bei denen die Norm über alle Blöcke bezüglich desselben Blockeintrages berechnet wird.

- ***blanc_matrix***, das, analog zu *blanc_vector*, die Typen ***blanc_Matrix*** und ***blanc_Melt*** (*matrix element*) bereitstellt. Eine *blanc_matrix* trägt jedoch nur die Werte der Matrix. Ihre Struktur, also ihr Besetzungsmuster, wird separat von dem Typ ***blanc_Mpat*** (*matrix pattern*) verwaltet. Da sich mehrere Matrizen auf dasselbe *blanc_Mpat* beziehen dürfen, kann so mehrfach gespeicherte Information vermieden werden. Dies läßt sich zum Beispiel dann ausnutzen, wenn auf ein und derselben Vernetzung verschiedene Probleme formuliert werden (wie bei der Behandlung von Navier–Stokes–Gleichungen durch Projektionsmethoden).
  Es besteht die Möglichkeit, diagonale Blockmatrizen günstiger zu speichern, sowie ohne wesentlichen Speicherbedarf die Transponierte einer Matrix zu bilden, die sich auf die Daten der Ausgangsmatrix bezieht.

- ***blanc_moperats***, das Operationen auf Matrizen (Multiplikation mit einem Skalar, Addition, Subtraktion, lineare Erweiterung) und zwischen Matrizen und Vektoren (Matrix–Vektor–Produkt, dasselbe mit der transponierten Matrix, Residuenberechnung) enthält. Außerdem kann die (sparsam als Diagonalmatrix gespeicherte) Inverse der Diagonale einer Matrix sowie die ILU–Zerlegung einer Matrix berechnet werden.

- ***blanc_precond***, in dem der Typ ***blanc_Precond*** (Vorkonditionierer) zur Verfügung steht. Realisiert sind Jacobi, SOR, SSOR und ILU(0). Sie werden durch den Typ ***blanc_Precond_id*** beschrieben, ihr Name kann aus dem Stringarray ***blanc_precond_name***[<*blanc_Precond_id*>] entnommen werden. Angesprochen werden sie über das *blanc_Precond*–Array ***blanc_precond***[<*blanc_Precond_id*>].
  Die Vorkonditionierungsmatrizen werden intern verwaltet. Der Benutzer kann aber auch selbst über Konstruktoren und Destruktoren bestimmen, wann sie erzeugt werden und wie lange sie existieren sollen. Bei Lösen von Systemen mit Unterschieden nur in der rechten Seite läßt sich so deren wiederholte Konstruktion vermeiden.
  In SOR und SSOR kann auf Wunsch jede Komponente des Lösungsvektors für sich relaxiert werden, als *blanc_Vcont* ist die Beschreibung des Relaxationsparameters völlig flexibel. Auch die Bearbeitungsreihenfolge der Zeilen kann vorgeschrieben werden.

- ***blanc_solver***, das iterative Lösungsverfahren als ***blanc_Solver*** anbietet. Zur Verfügung stehen SOR, SSOR, GMRES(m), CG, CGNR, Bi–CG, CGS, Bi–CGSTAB, QMR und TFQMR. ***blanc_Solver_id***, ***blanc_solver_name*** und ***blanc_solver*** erfüllen entsprechende Aufgaben wie ihre Pendants im Modul *blanc_precond*. Auch die Durchführung von SOR und SSOR kann wie dort variiert werden. Das Abbruchkriterium wird durch Angabe der maximalen Zahl von Iterationen und durch Konvergenzkontrolle gemäß einem *blanc_Cc* (siehe dazu den nächsten Punkt) definiert. Die Anzahl der tatsächlich benötigten Iterationen wird in den Speicherplatz der vorgegebenen Höchstzahl geschrieben.
  Weiterhin ist es möglich zu bestimmen, ob die Konvergenzkontrolle nur auf die Residuennorm oder auf den gesamten Residuenvektor angewendet wird, und ferner, ob die in manchen Verfahren gültigen Abschätzungen der Norm ausgenutzt werden sollen oder nicht.

- ***blanc_cc*** (*convergence control*), das mit dem Typ ***blanc_Cc*** eine flexible Konvergenzkontrolle für iterative Verfahren anbietet. Er enthält eine *blanc_Vtype*–Information, die bestimmt, ob das Residuum als *blanc_Real* (Norm des Vektors), *blanc_Velt* (eintragsweise Norm über die Blöcke) oder als *blanc_Vector* (komponentenweiser Betrag des Vektors) bewertet werden soll. Als Normen stehen die in *blanc_voperats* realisierten zur Verfügung. In Übereinstimmung mit dem vorgeschriebenen Typ können (ggf. komponentenweise) obere Schranken für die zu erzielende Genauigkeit ($\|r_k\|$) und für die erwünschte Konvergenz ($\frac{\|r_k\|}{\|r_0\|+tol}$) angegeben werden. *tol* ist ein optionaler Toleranzwert, durch den man Division durch 0 vorbeugen kann. Die zur Evaluierung aufgerufene Funktion ***blanc_cc_check*** gibt den *blanc_Boolean*–Wert ***blanc_B_FALSE*** (*falsch*) zurück, wenn eine der Schranken unterschritten wird. Andernfalls wird ***blanc_B_TRUE*** (*wahr*) zurückgegeben. Auch beliebige andere Abbruchkriterien sind durch den Benutzer definierbar, indem er dem *blanc_Cc* eine Funktion

vom Typ ***blanc_Cc_function*** bereitstellt. Diese wird in *blanc_cc_check* aufgerufen und führt durch Rückgabe von *blanc_B_FALSE* ebenfalls zu einem Abbruch. In der *blanc_Cc_function* werden dem Benutzer unter anderem die Iterationszahl k, die aktuelle Näherungslösung x_k und das aktuelle Residuum r_k übergeben.

- ***blanc_io***, das lesbare ASCII– oder speicherplatzsparende binäre Ein– und Ausgabe der Typen *blanc_Velt*, *blanc_Vector*, *blanc_Melt*, *blanc_Matrix* und *blanc_Mpat* in einem festen Format über bereitzustellende Datenströme ermöglicht. Bei ASCII–Ausgabe ist jedoch die Darstellung der einzelnen Werte im Exponential– oder Gleitkommaformat sowie die Anzahl der Stellen und Nachkommastellen variabel. Für eine *blanc_Matrix* kann bei ASCII–Ausgabe zwischen dem sparse–Format unter Angabe der jeweiligen Position und dem dense–Format unter Auffüllung mit Nullen gewählt werden.

- ***blanc_mespas***, in dem ein Interface für Datenübertragung (*message passing*) bei Benutzung von BLANC in parallelisierten Umgebungen (wie z.B. PVM) bereitgestellt wird. Objekte vom Typ *blanc_Velt* oder *blanc_Vector* können unter Angabe der Sende– und Empfangsfunktionen der Umgebung verschickt und erhalten werden. Bei Existenz selektierter Datenübertragung ist auch die Versendung jeweils desselben Eintrags aus allen Blöcken eines *blanc_Vector*'s möglich.

4.4 Numerische Resultate

4.4.1 Stationäre Verfahren

In dieser Versuchsserie sollen zuerst die im folgenden als Vorkonditionierer einsetzbaren stationären Verfahren SOR und SSOR miteinander verglichen werden. Da ihr Unterschied in der Bearbeitungsreihenfolge der Matrixzeilen liegt, wird dafür das Beispiel der Schrägströmung mit lexikographischer und Quernumerierung betrachtet, wie in Abschnitt 4.1 beschrieben.
Nach der Entscheidung für eines der beiden wird dieses Verfahren auf alle weiteren Probleme angewendet. Der Relaxationsfaktor ω wird jeweils unter besonderer Berücksichtigung der Grenzfälle $\omega \to 0$ und $\omega \to 2$ über $\{0.00625, 0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 1.9, 1.95, 1.975, 1.9875, 1.99375\}$ variiert, um Erkenntnisse über dessen optimale Wahl und Auswirkungen einer nichtoptimalen Wahl zu gewinnen. Der jeweils als optimal erkannte Relaxationsparameter wird im Anschluß bei der Verwendung des Verfahrens als Vorkonditionierung für PGK–Verfahren benutzt werden.
Die Rechtfertigung der Annahme, daß das wirkungsvollere stationäre Lösungsverfahren auch der effizientere Vorkonditionierer ist, beruht auf der linearen Konvergenz dieser Verfahren bezüglich des Fehlers laut (2.8). Diese wurde, um es vorwegzunehmen, durch

die Experimente durchweg bestätigt. Da Vorkonditionierung durch stationäre Verfahren einem Iterationsschritt entspricht, überträgt sich die Konvergenzeigenschaft des Verfahrens auf die Güte des Vorkonditionierers.
Die Annahme, bei der Vorkonditionierung den optimalen Relaxationsfaktor verwenden zu dürfen, wird später begründet werden.

Auf den folgenden Seiten sind pro Doppelseite die Diagramme dargestellt, die die Ergebnisse für einen mit jeweils einem Löser bearbeiteten Versuch zusammenfassen. Jede Zeile beschreibt von links nach rechts zunehmende Konvektionsdominanz, jede Spalte von oben nach unten zunehmende Diskretisierungsfeinheit bzw. Matrixdimension. In den Diagrammen ist jeweils auf der x–Achse die Rechenzeit in CPU–Sekunden, auf der y–Achse der Relaxationsparameter aufgetragen. Für jede Relaxation ist ein Balken dargestellt,

- dessen Gesamtlänge die Zeit beschreibt, die das Verfahren benötigte, bis die euklidische Norm des Residuum kleiner als $10^{-6}\|r_0\|_2$ war,
- dessen dunkler abgesetzter Teil die benötigte Zeit zur Konvergenz auf $10^{-2}\|r_0\|_2$ angibt und
- dessen schwarzer Anteil die Aufbauphase repräsentiert, d.h. die zur Inversion der Diagonalen und der Konstruktion der Hilfsvektoren verwendete Zeitspanne, bevor der eigentliche Lösungsprozeß begann.

Das Beispiel der lexikographisch numerierten Schrägströmung lösen SOR und SSOR im optimal relaxierten Fall etwa gleichschnell, wobei jedoch auffällt, daß sich die Konvergenz des SOR bei Abweichung von ω_{opt} stärker verschlechtert als die des SSOR. Dort sind also gröbere Abschätzungen des Parameters zulässig.
Der Wert des optimalen Parameters ist bei beiden Verfahren etwa gleich. Dies gilt auch bei der Lösung des quernumerierten Problems. Dort ist, wie erwartet, die Konvergenz des SOR wesentlich schlechter als im lexikographisch numerierten, die des SSOR aber bemerkenswertereise unverändert.
Da SSOR außerdem, wie beschrieben, Symmetrieeigenschaften erhält, ist es dem SOR unbedingt als Löser, und damit auch als Vorkonditionierer, vorzuziehen.

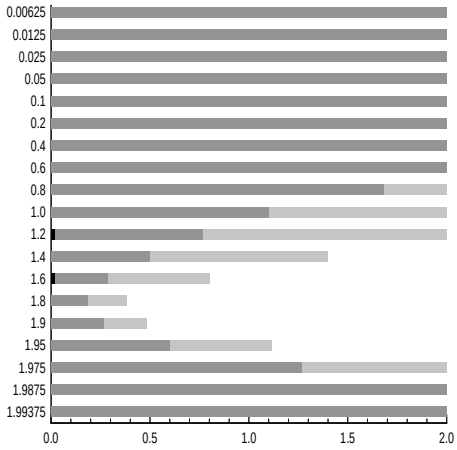
Es wird bei allen Versuchen offensichtlich, daß die Konvergenzrate des SSOR stetig von ω abhängt. Schätzt man $\omega \approx \omega_{opt}$ ab, so läßt sich daher fast–optimale Konvergenz erwarten. Bei Unterschätzung von ω_{opt} büßt man weniger an Konvergenzgeschwindigkeit ein als bei Überschätzung. Insbesondere ist bei Überschätzung die Gefahr gegeben, daß das Verfahren divergiert, bei Unterschätzung jedoch nur langsamere Konvergenz.
Der Wert des optimalen Parameters fällt mit zunehmender Konvektionsdominanz von starker Überrelaxation ($\omega \approx 1.9$) auf Unterrelaxation im Bereich $\omega \approx 0.6$ und steigt gleichzeitig mit wachsender Zerlegungsfeinheit an, jedoch so schwach, daß dieser Effekt nicht besonders berücksichtigt werden muß.

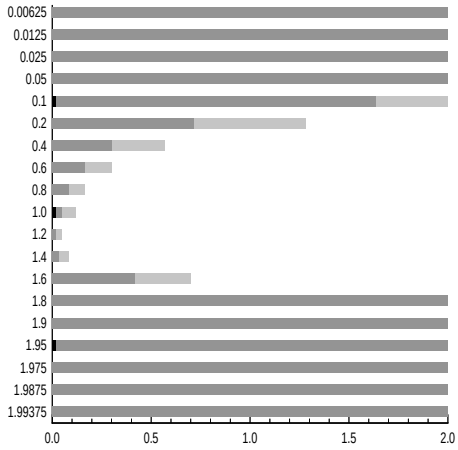
Als naive Wahl des Relaxationsparameters wird $\omega := 0.8$ empfohlen, wodurch man in den hier betrachteten Fällen immer Konvergenz und oft fast–optimale Konvergenz erzielt.
Manchmal führt diese Wahl aber auch zu schlechter Konvergenz, so daß im allgemeinen zur effektive Anwendung des SSOR eine Abschätzung des optimalen Relaxationsparameters nötig ist.
Diese läßt sich durch einige SSOR–Schritte in der Startphase des Verfahrens (bzw. des den SSOR als Vorkonditionierer benutzenden Verfahrens) realisieren. Wegen der linearen Konvergenz läßt sich durch einen Schritt bereits die Konvergenzrate abschätzen. Dafür muß jedoch die exakte Lösung des Systems bekannt sein, denn die Aussage über lineare Konvergenz bezieht sich auf den Fehler und nicht auf das Residuum (wie die Konvergenzverläufe im Anhang zeigen, ist die Konvergenz bezüglich der Residuennorm gerade in den ersten Schritten nicht linear). Man behilft sich durch das homogenisierte Problem, dessen exakte Lösung $x = 0$ bekannt ist. Mit einem beliebigen Vektor $\tilde{x} \neq 0$ läßt sich für verschiedene ω die Konvergenzrate bezüglich des Fehlers ermitteln. Durch sukzessive Teilung des Intervalls $(0, 2)$ und Tests an Stützstellen kann man den optimalen Relaxationsparameter eingrenzen.

Als weitere Modifikation des SSOR–Lösungsverfahrens wird vorgeschlagen, nur alle $t > 0$ Iterationen das Residuum zu ermitteln und die Konvergenz zu kontrollieren.
Wegen der linearen Konvergenz des Fehlers ist sogar eine a–priori Abschätzung der Iteratonszahl zum Erreichen des Abbruchkriteriums denkbar, so daß gänzlich auf Berechnung des Residuums verzichtet werden kann.
Die Ursache diese Vorschläge ist, daß im Vergleich mit den Konvergenzraten der PGK–Verfahren der optimal relaxierte SSOR konkurrenzfähig ist. Bei den zugrundegelegten Untersuchungen wurde jedoch noch in jedem SSOR–Schritt unter etwa einem Drittel zusätzlichem Aufwand das Residuum bestimmt. SSOR verspricht daher mit der vorgeschlagenen Korrektur und der oben formulierten Approximation von ω_{opt}, eine ernstzunehmende Alternative darzustellen, zumal es linear konvergiert.

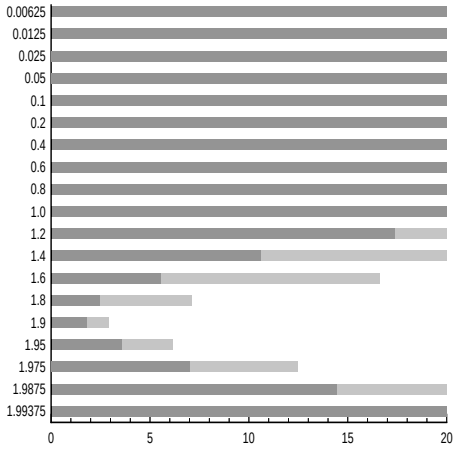
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

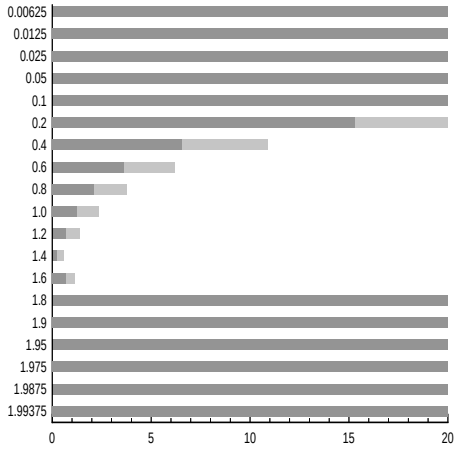
$n = 33 \times 33$



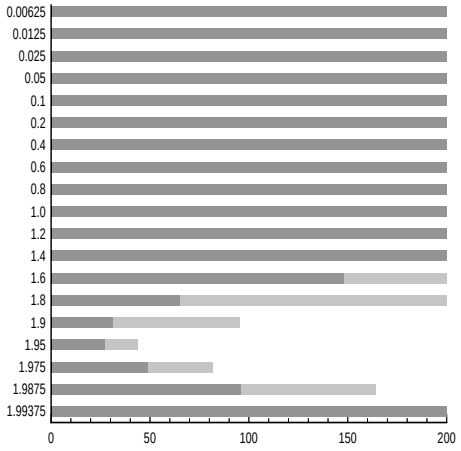


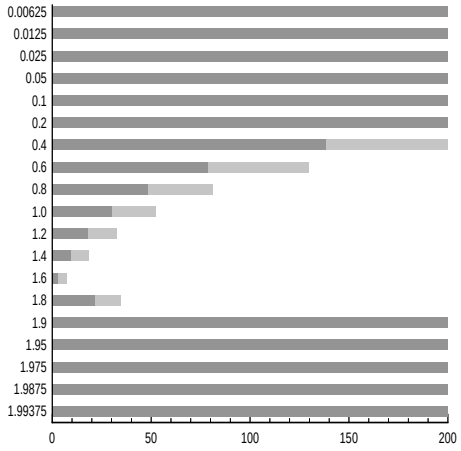
$n = 65 \times 65$



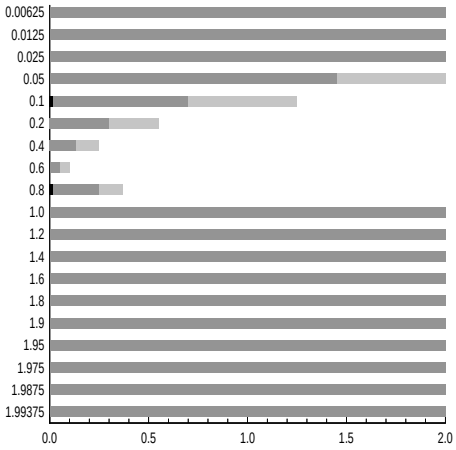


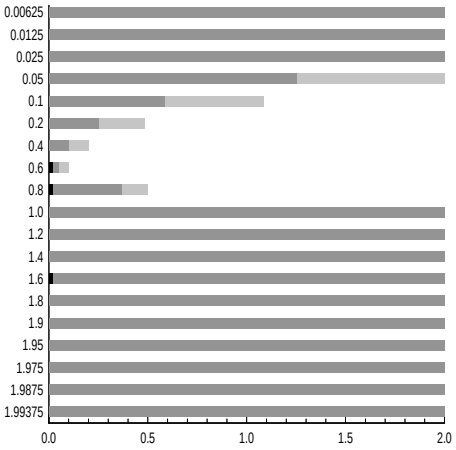
$n = 129 \times 129$



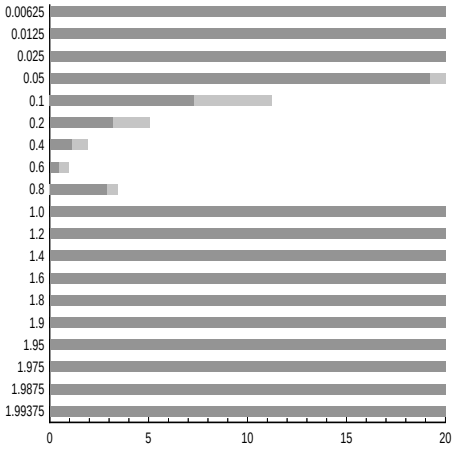


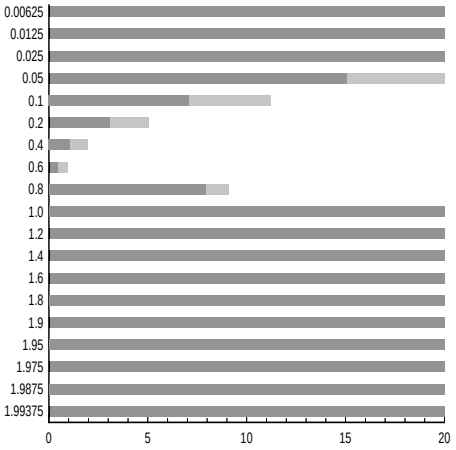
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



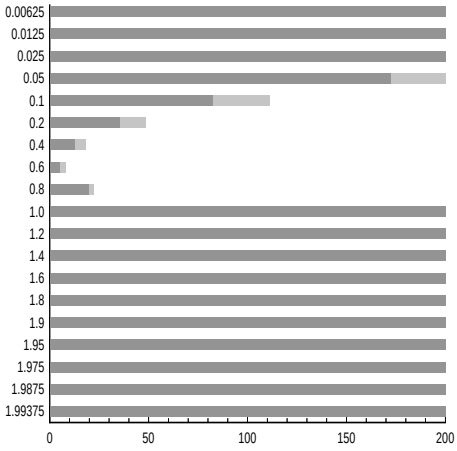


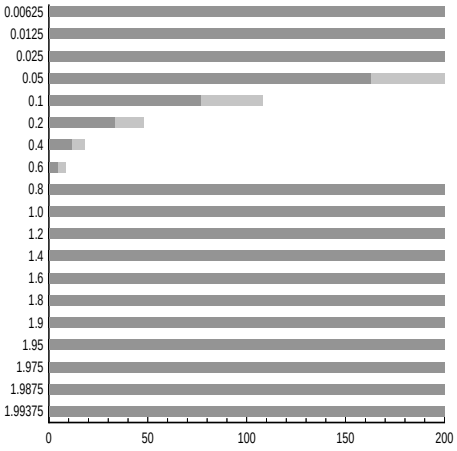
$n = 33 \times 33$





$n = 65 \times 65$

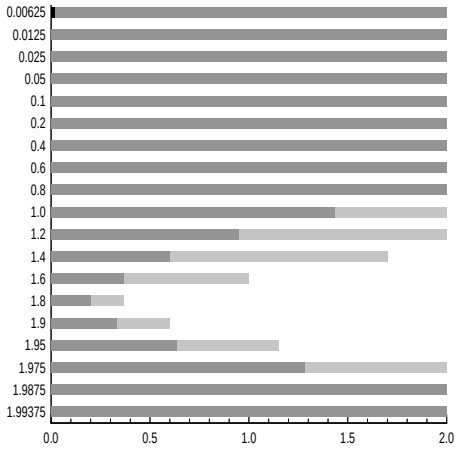


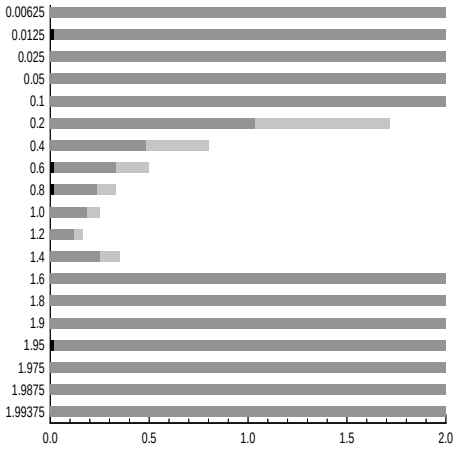


$n = 129 \times 129$

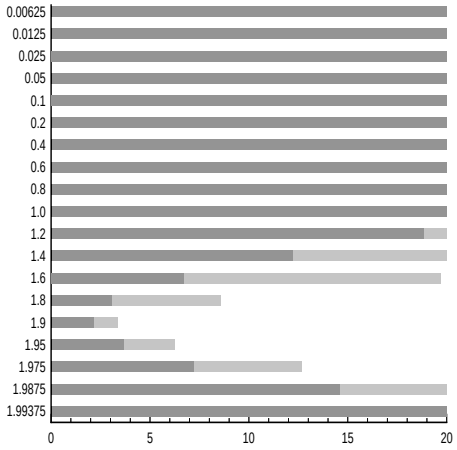
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

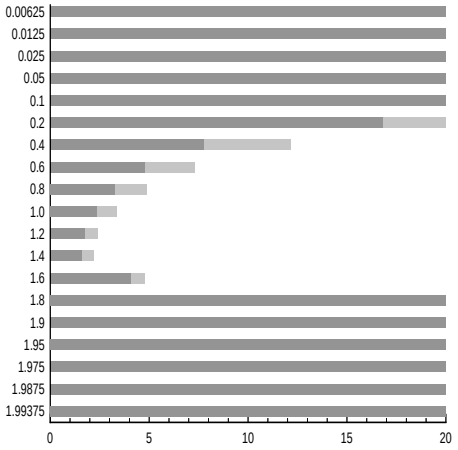
$n = 33 \times 33$



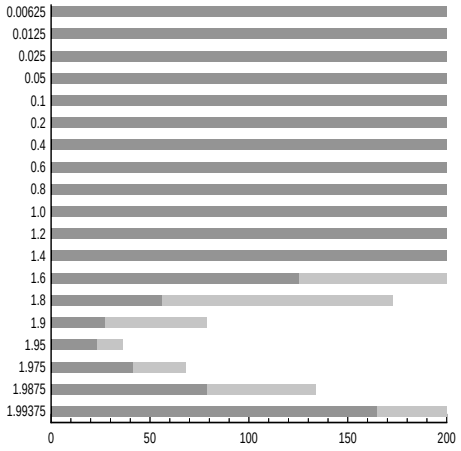


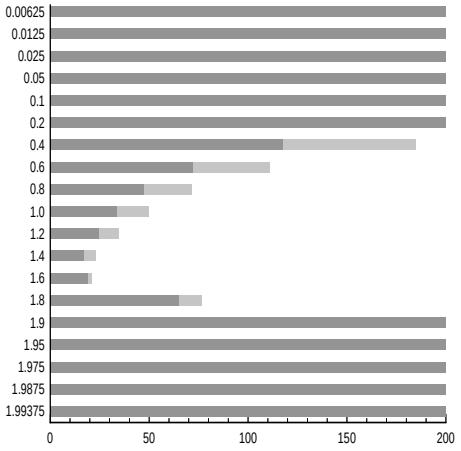
$n = 65 \times 65$



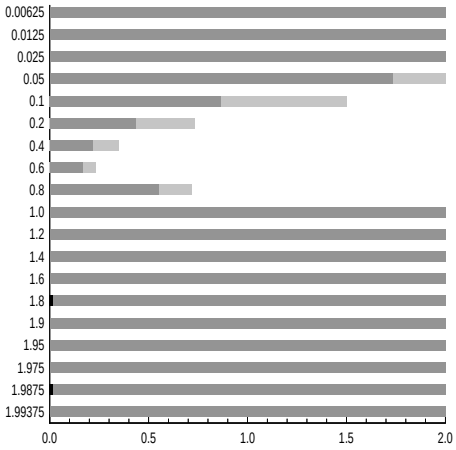


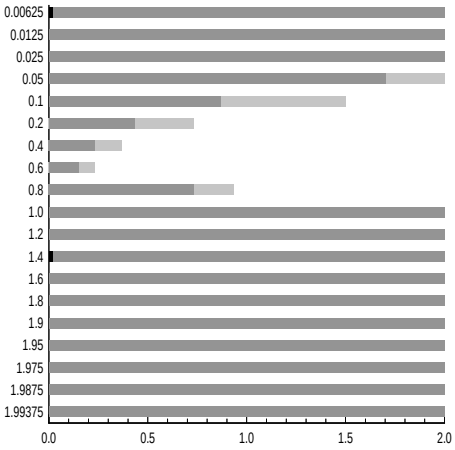
$n = 129 \times 129$



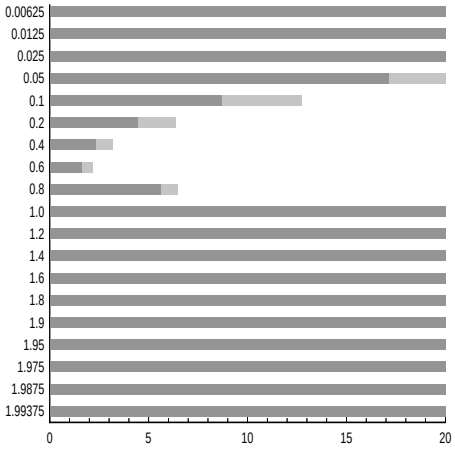


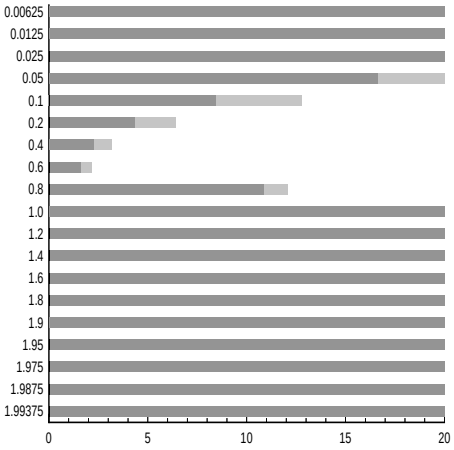
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



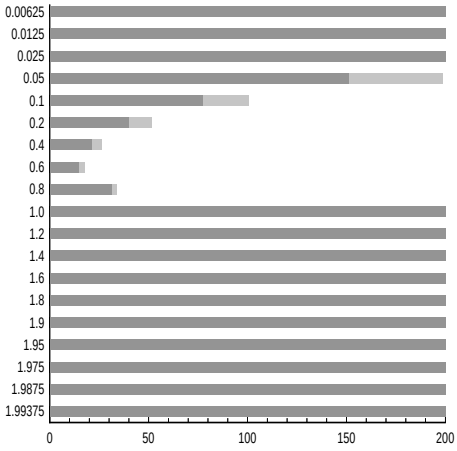


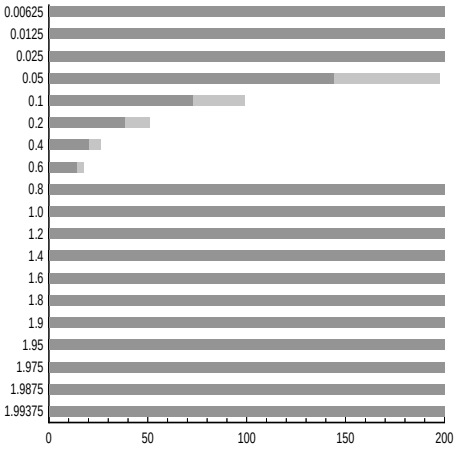
$n = 33 \times 33$





$n = 65 \times 65$

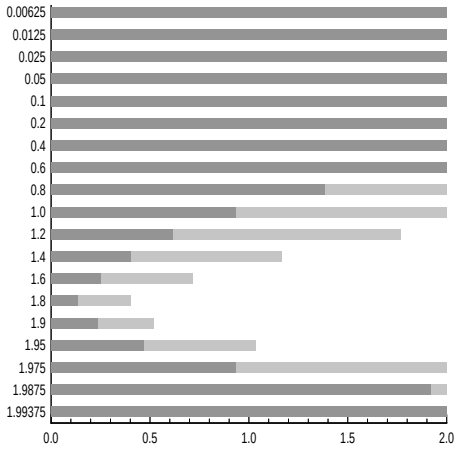


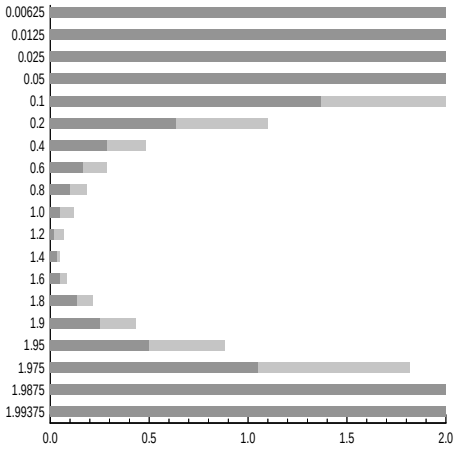


$n = 129 \times 129$

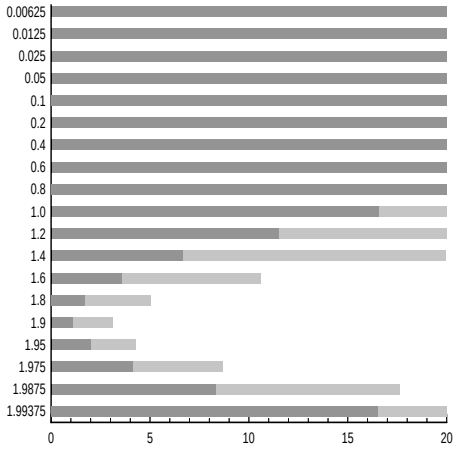
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

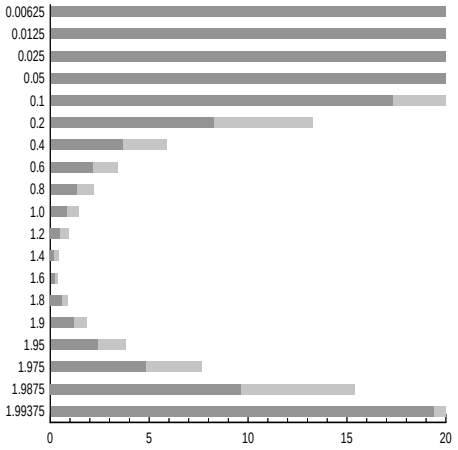
$n = 33 \times 33$



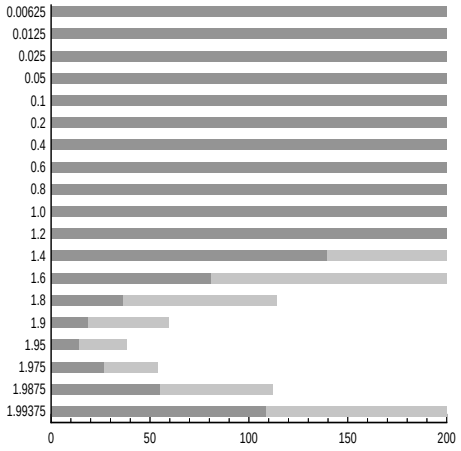


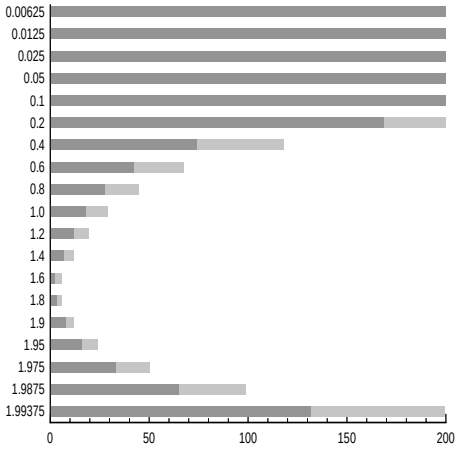
$n = 65 \times 65$



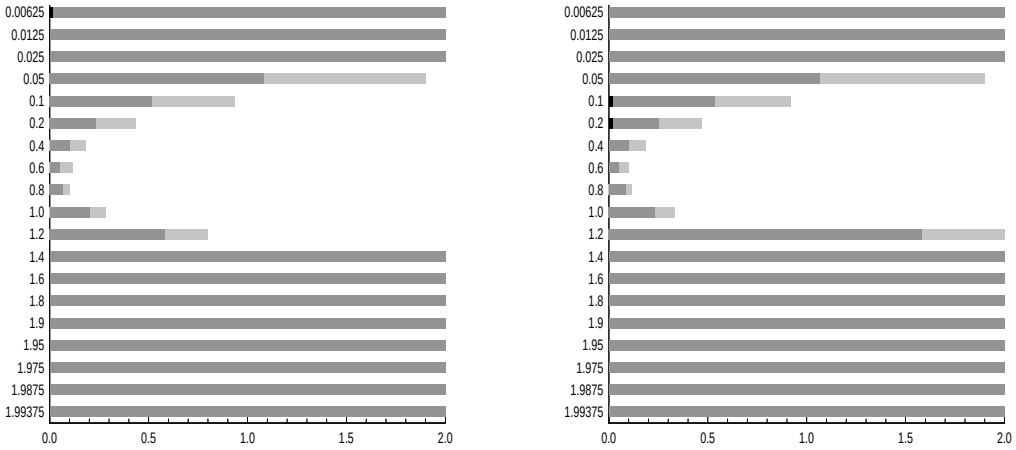


$n = 129 \times 129$

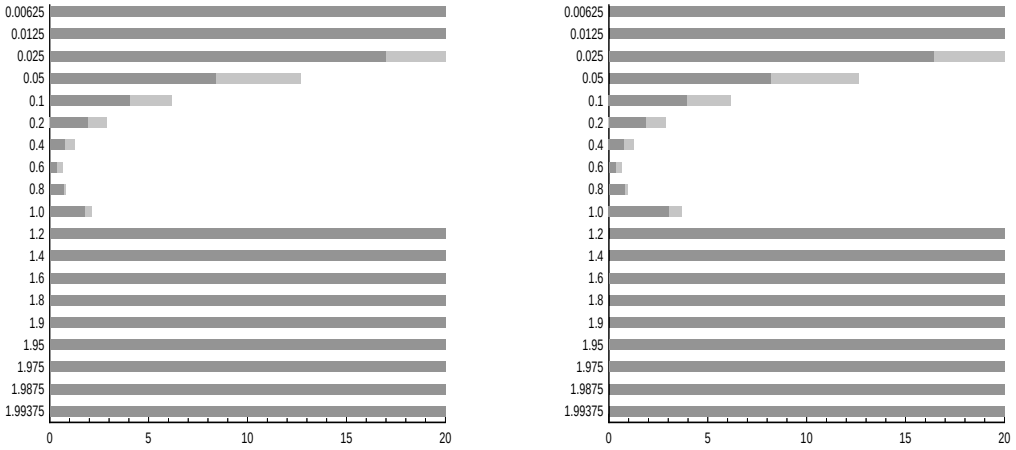




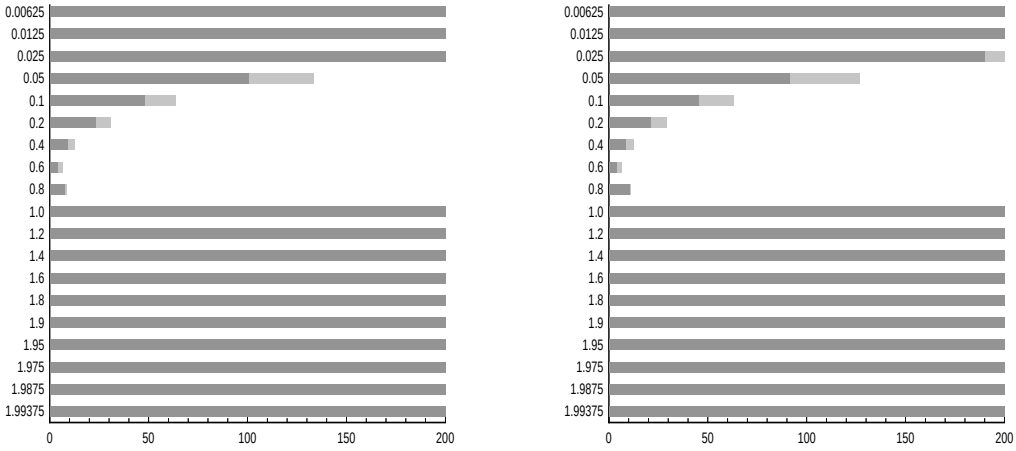
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



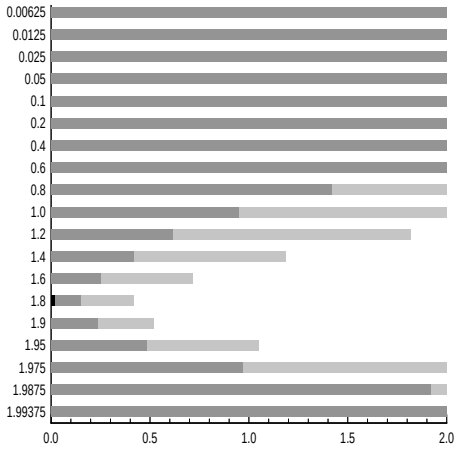
$n = 65 \times 65$

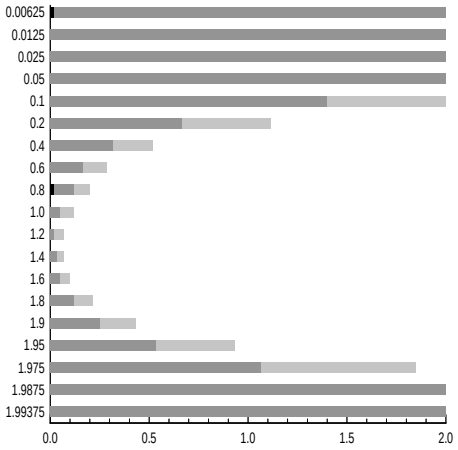


$n = 129 \times 129$

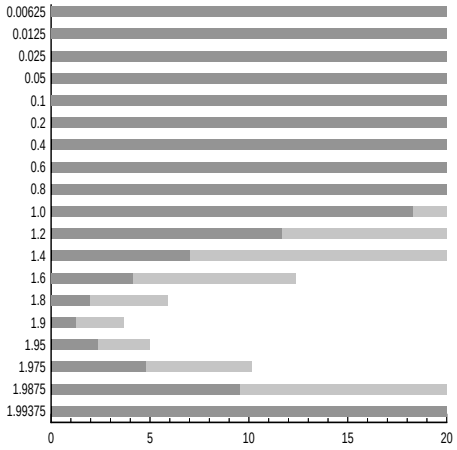
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

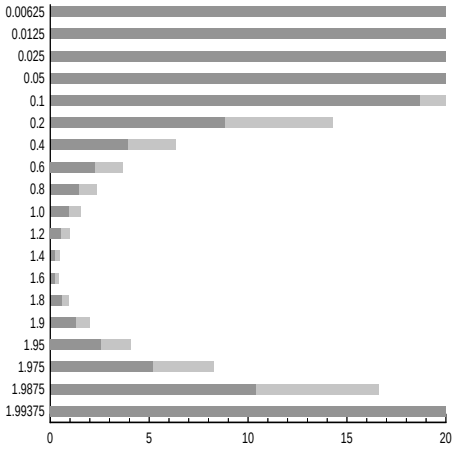
$n = 33 \times 33$



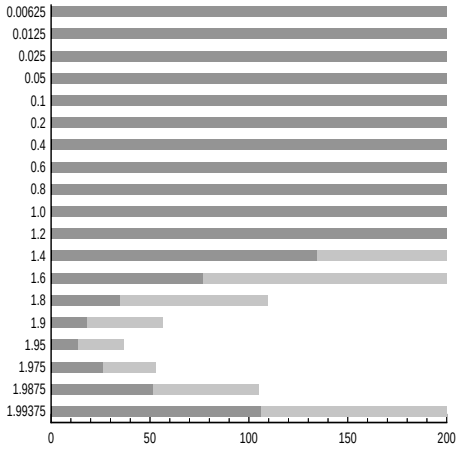


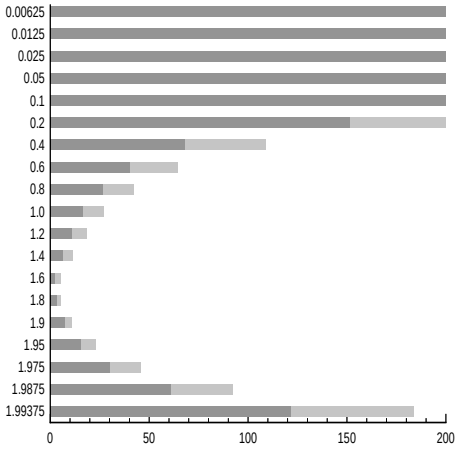
$n = 65 \times 65$



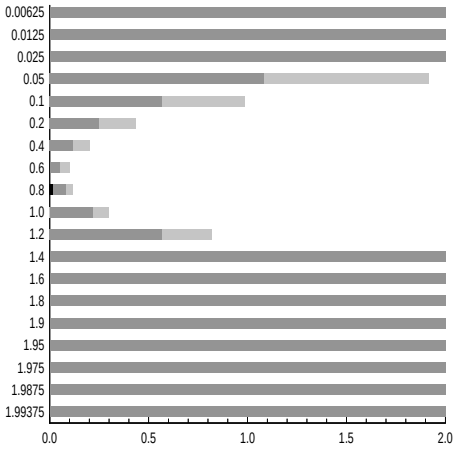


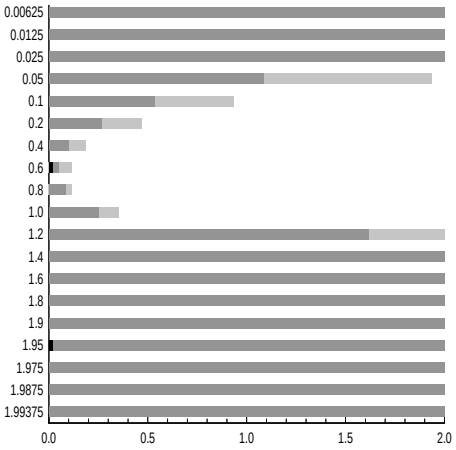
$n = 129 \times 129$



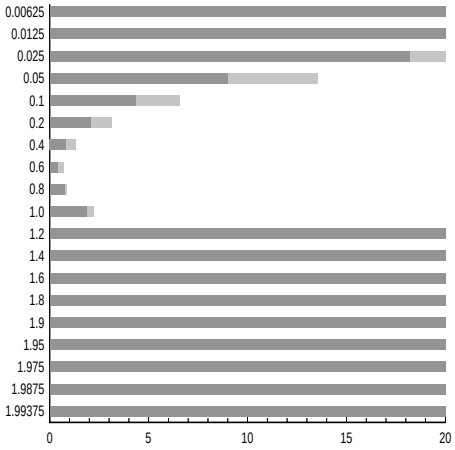


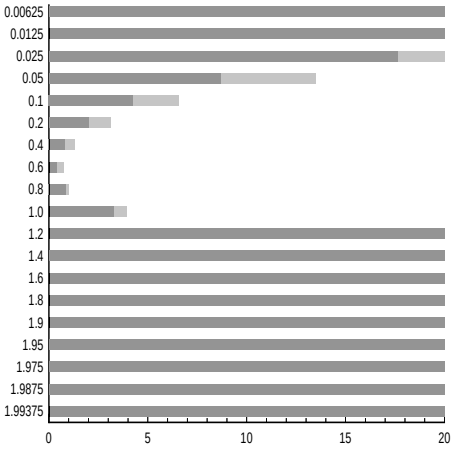
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



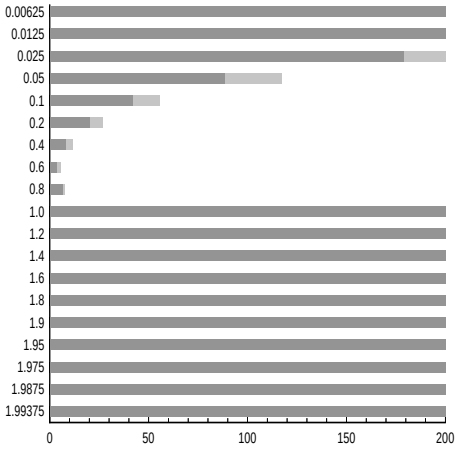


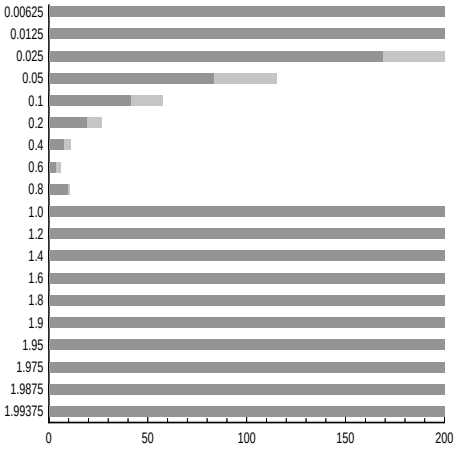
$n = 33 \times 33$





$n = 65 \times 65$

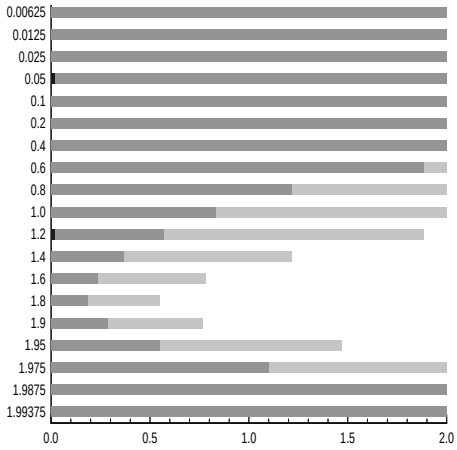


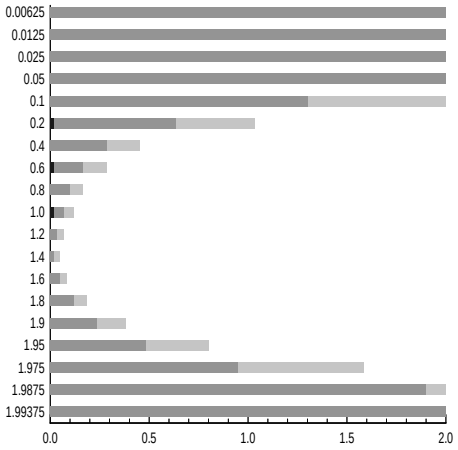


$n = 129 \times 129$

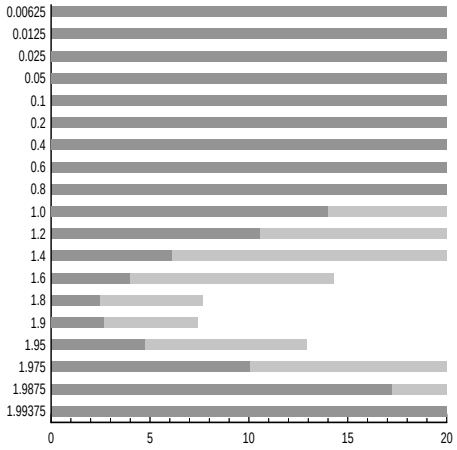
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

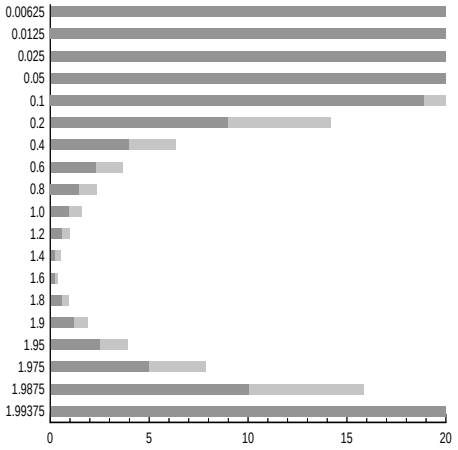
$n = 33 \times 33$



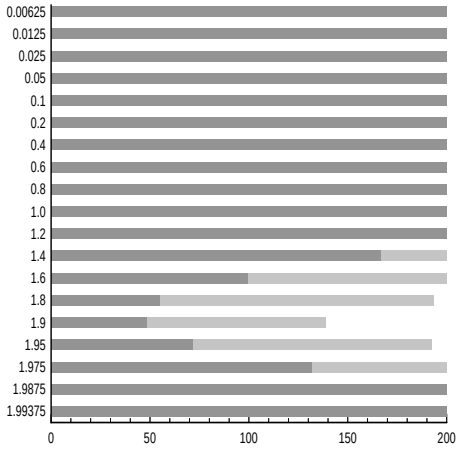


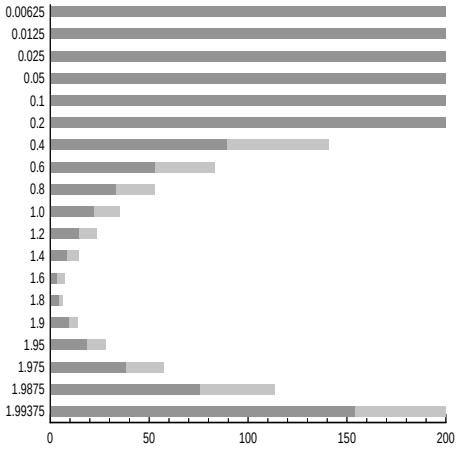
$n = 65 \times 65$



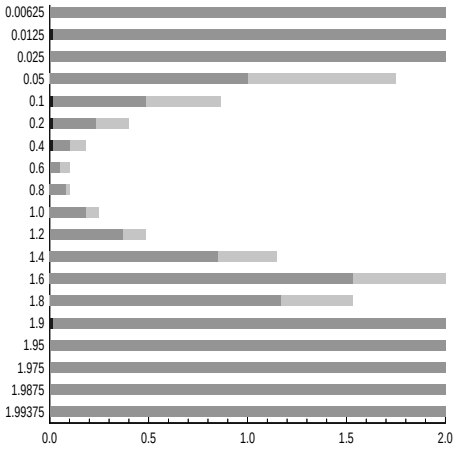


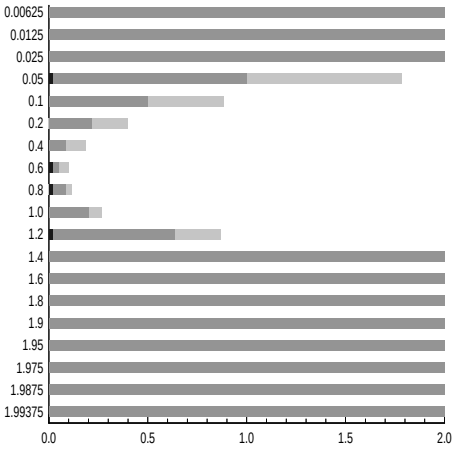
$n = 129 \times 129$



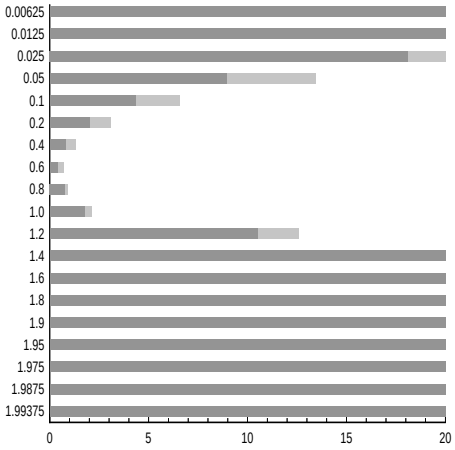


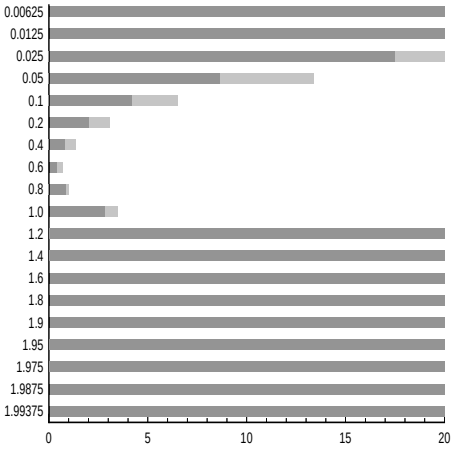
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



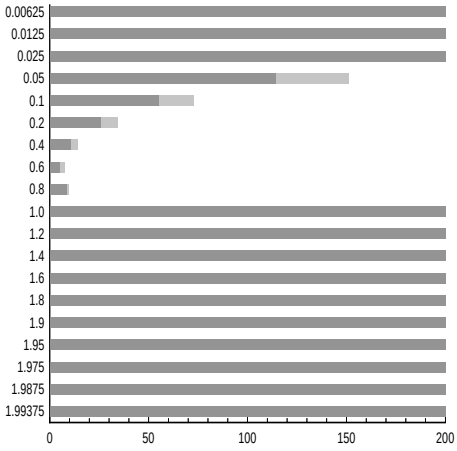


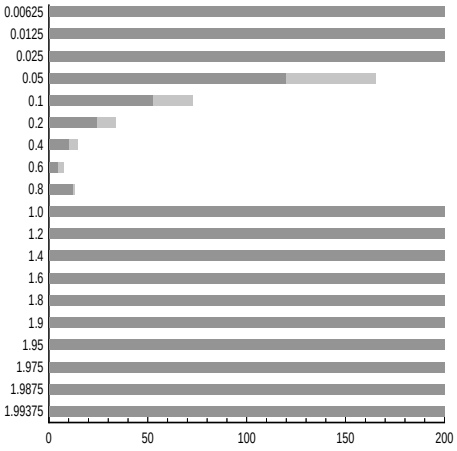
$n = 33 \times 33$





$n = 65 \times 65$

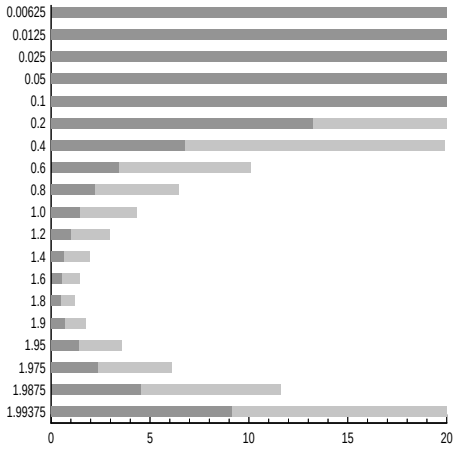


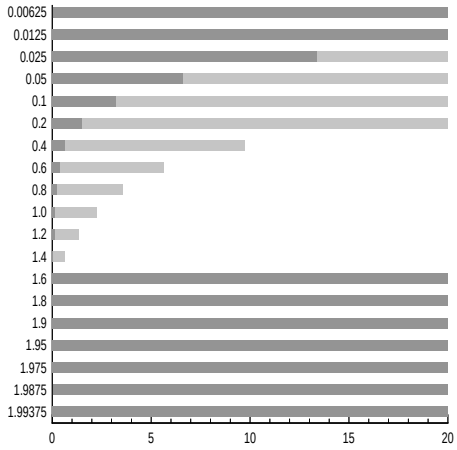


$n = 129 \times 129$

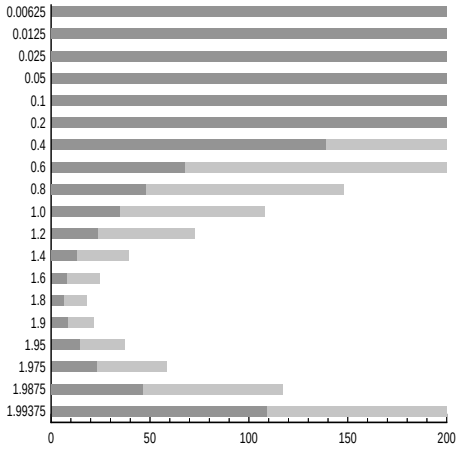
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

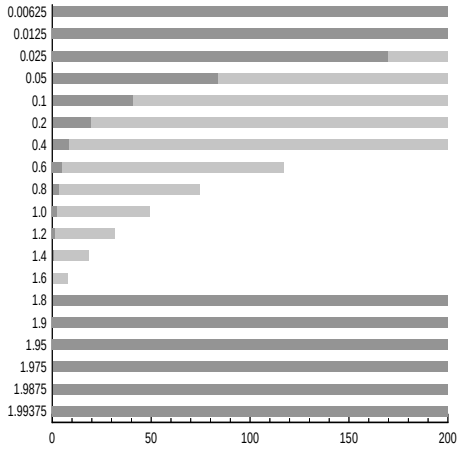
$n = 33 \times 33$



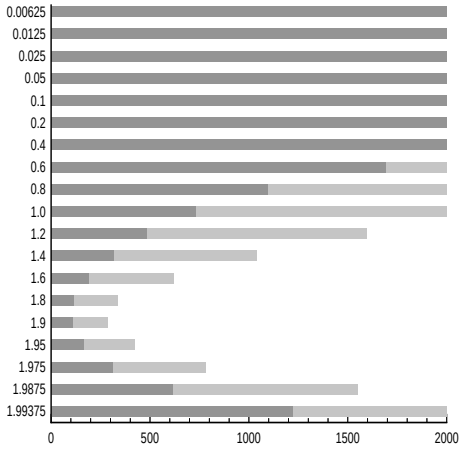


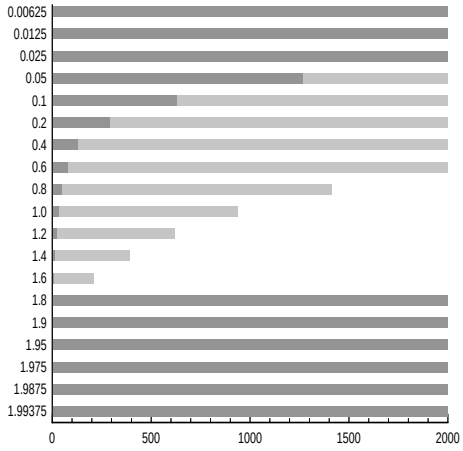
$n = 65 \times 65$



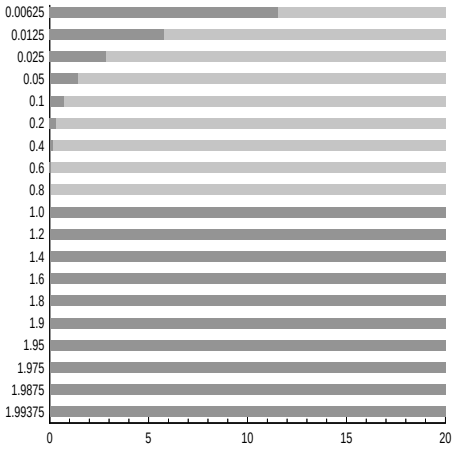


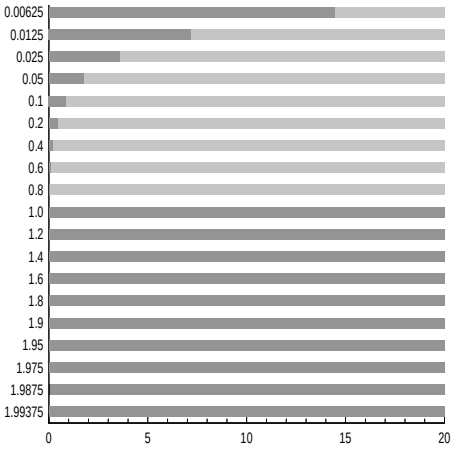
$n = 129 \times 129$



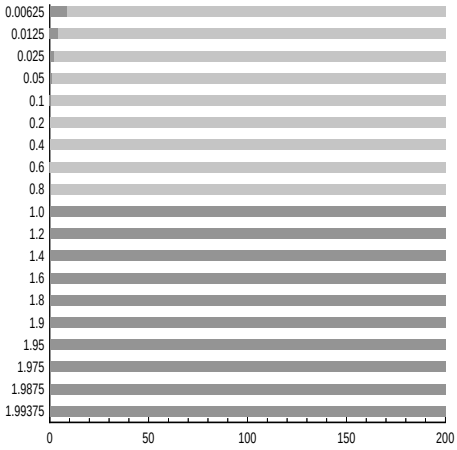


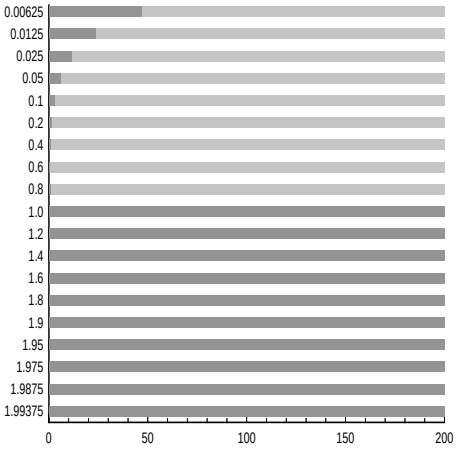
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



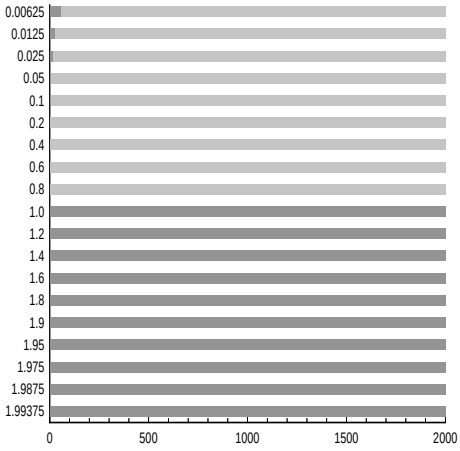


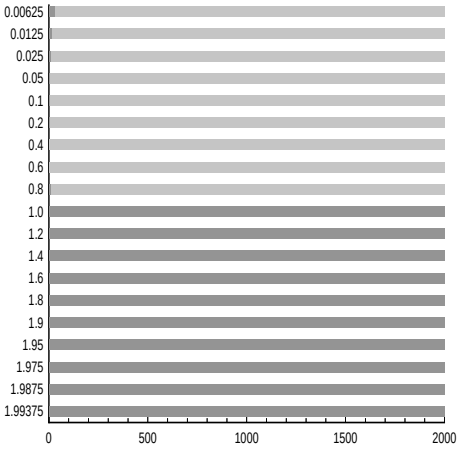
$n = 33 \times 33$





$n = 65 \times 65$

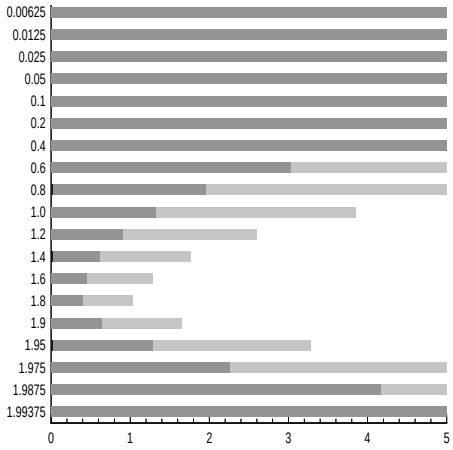


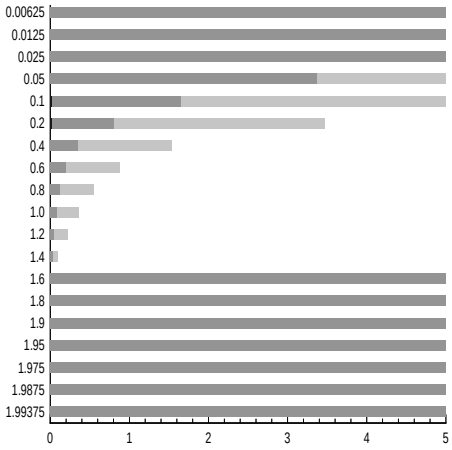


$n = 129 \times 129$

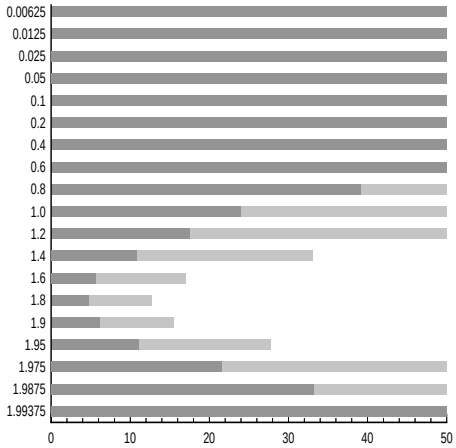
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

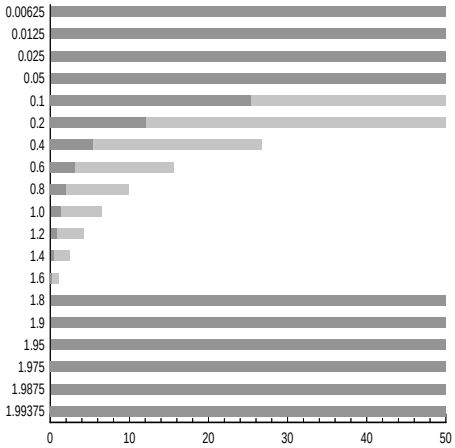
$n = 33 \times 33$



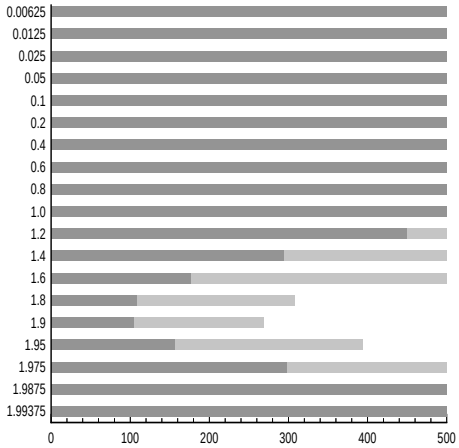


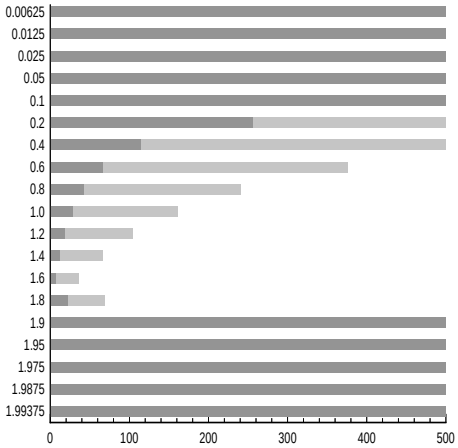
$n = 65 \times 65$



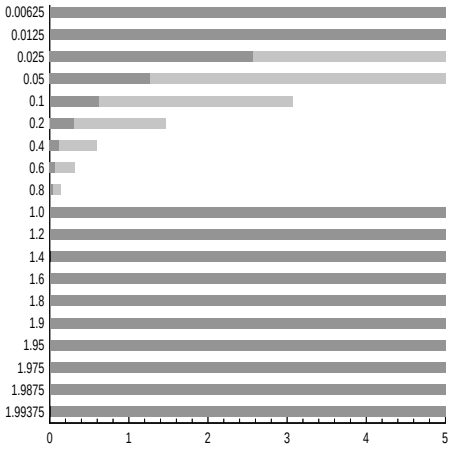


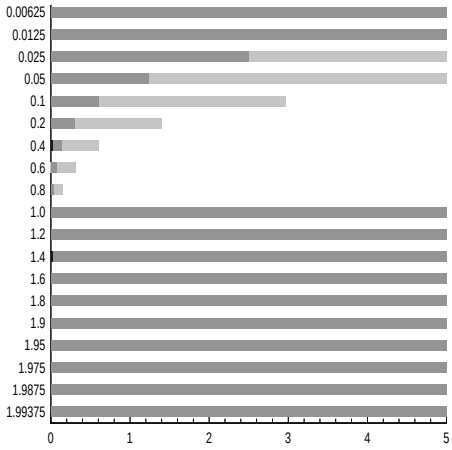
$n = 129 \times 129$



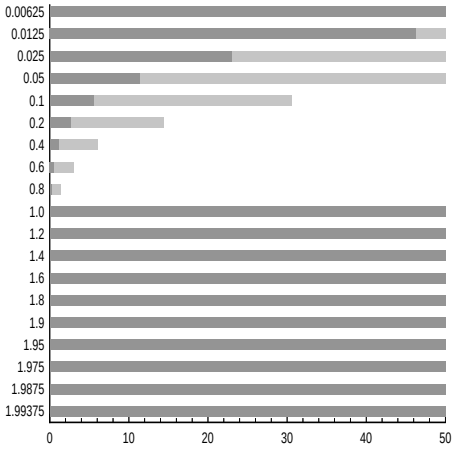


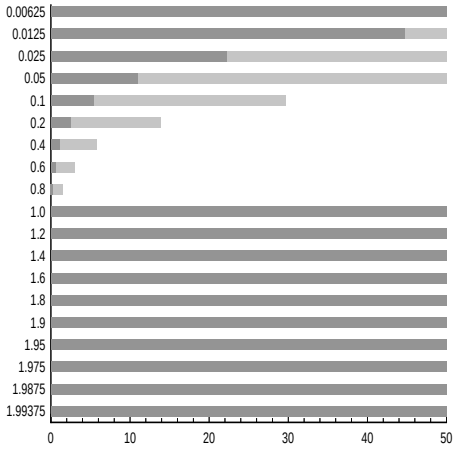
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



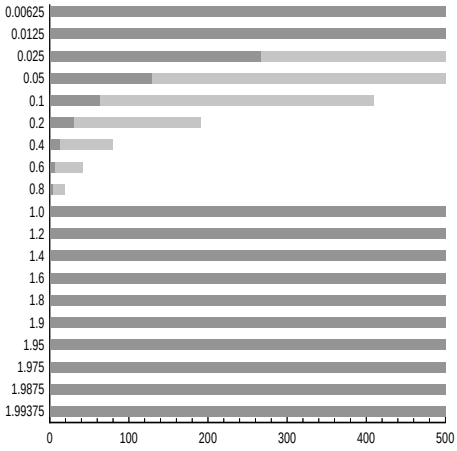


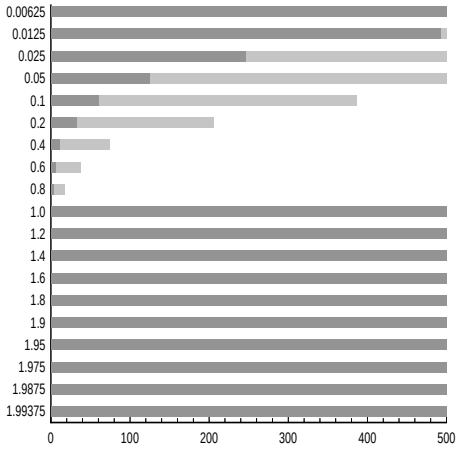
$n = 33 \times 33$





$n = 65 \times 65$

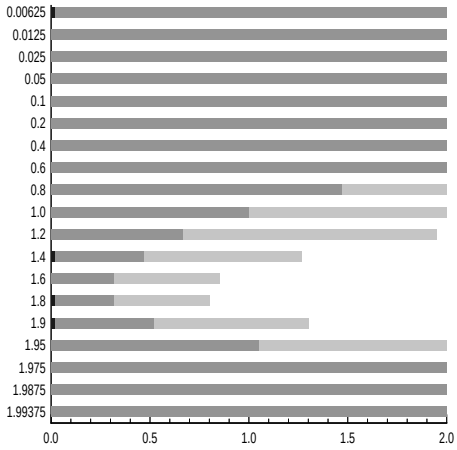


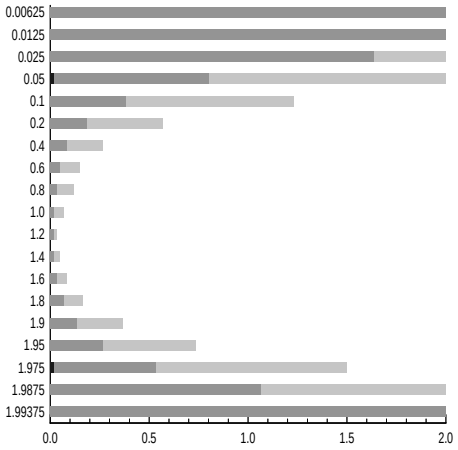


$n = 129 \times 129$

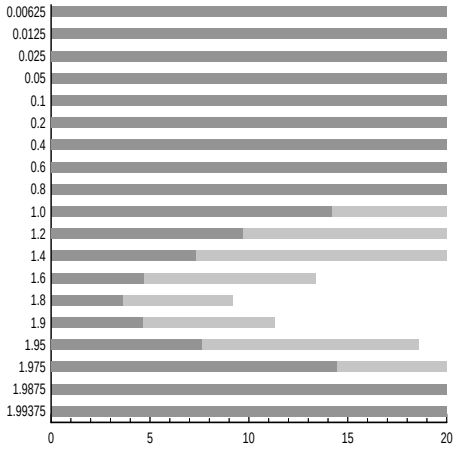
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

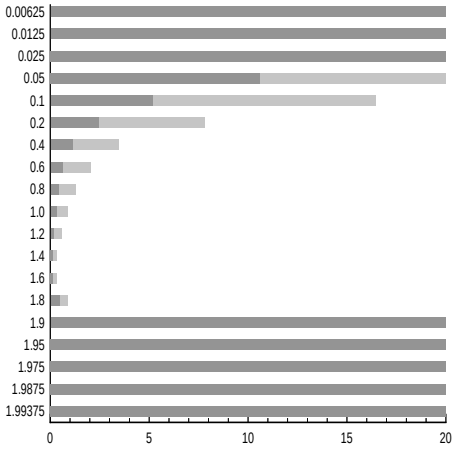
$n = 33 \times 33$



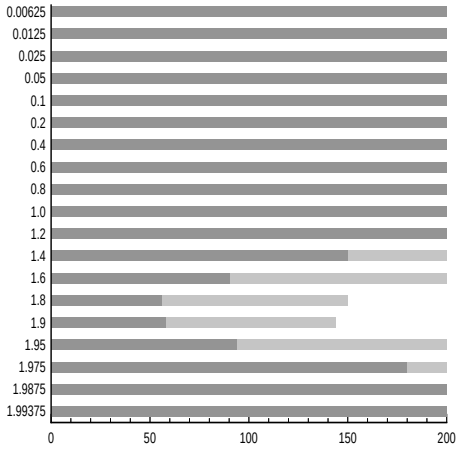


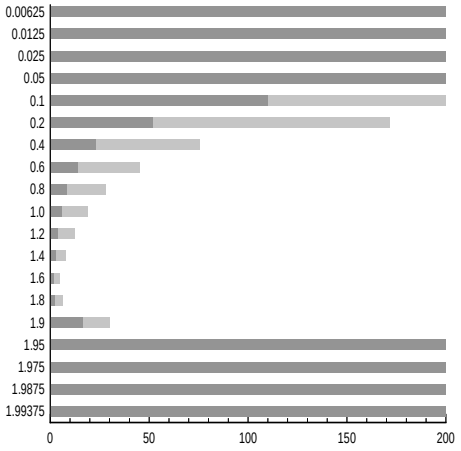
$n = 65 \times 65$



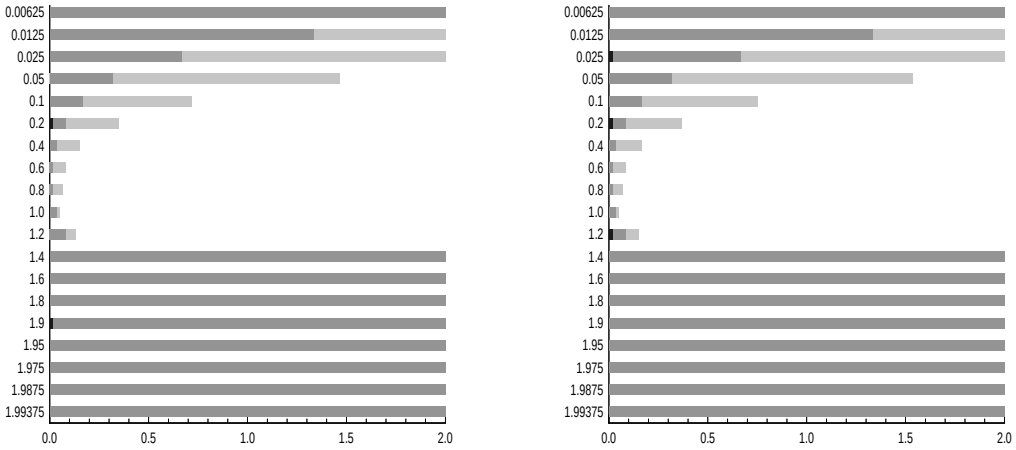


$n = 129 \times 129$

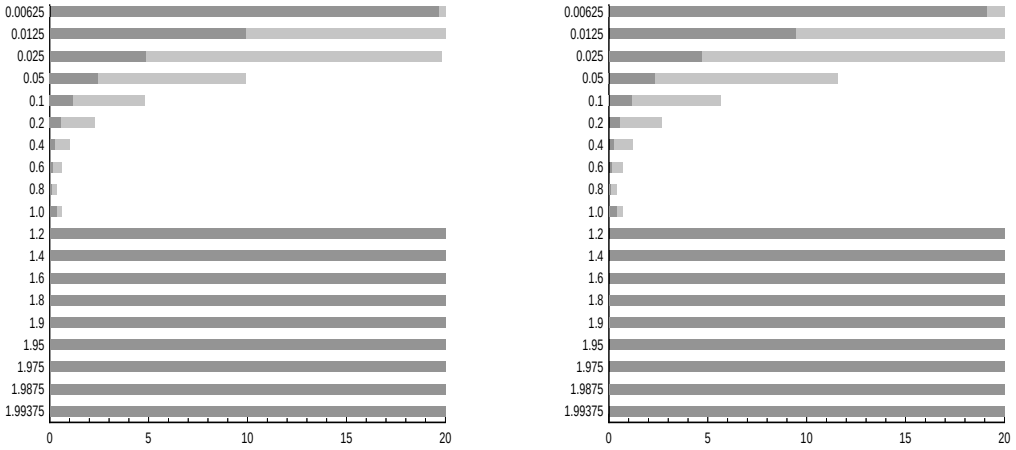




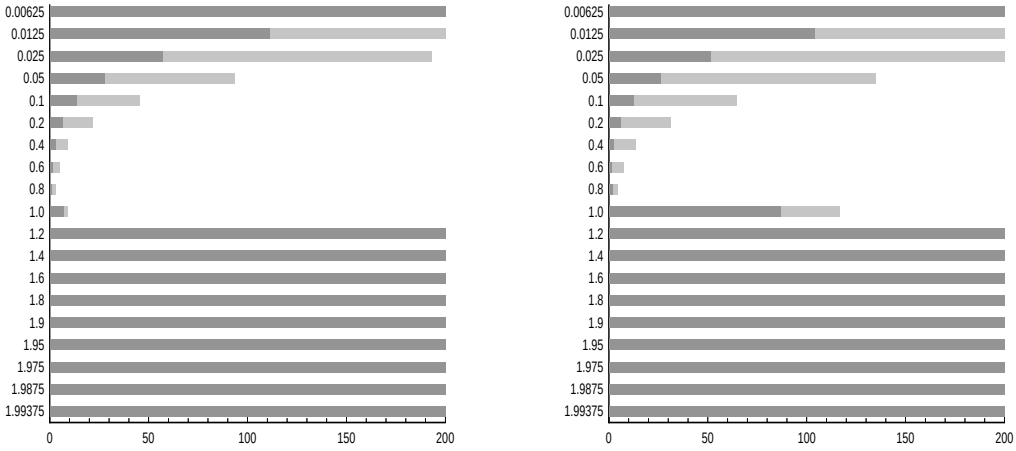
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



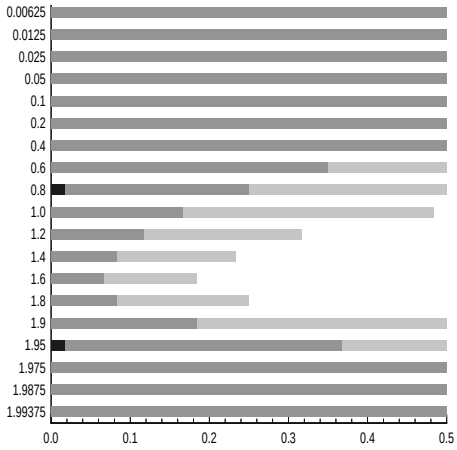
$n = 65 \times 65$

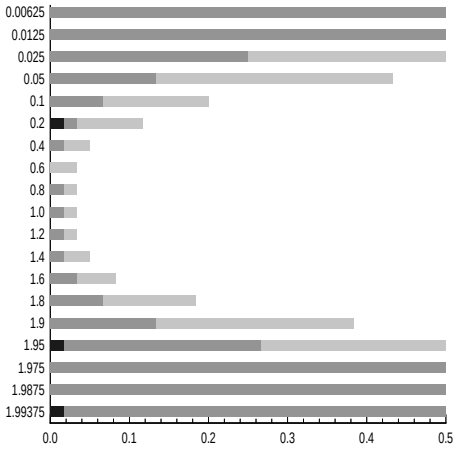


$n = 129 \times 129$

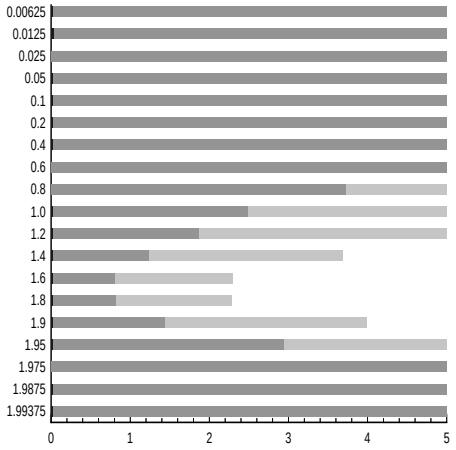
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

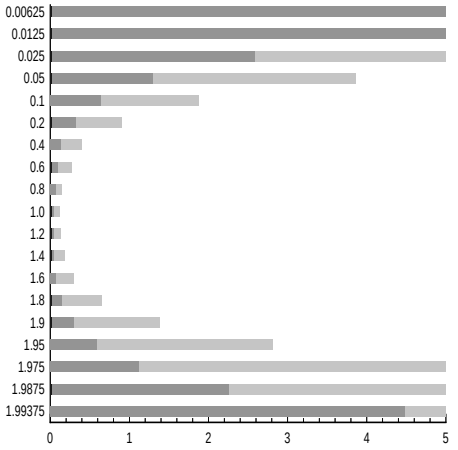
$n = 33 \times 33$



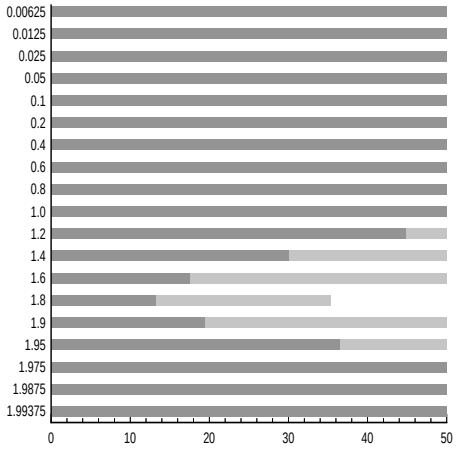


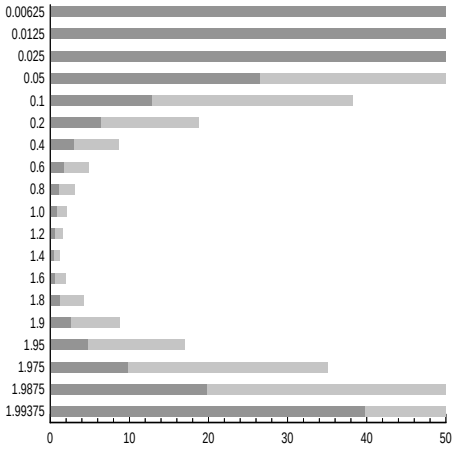
$n = 65 \times 65$



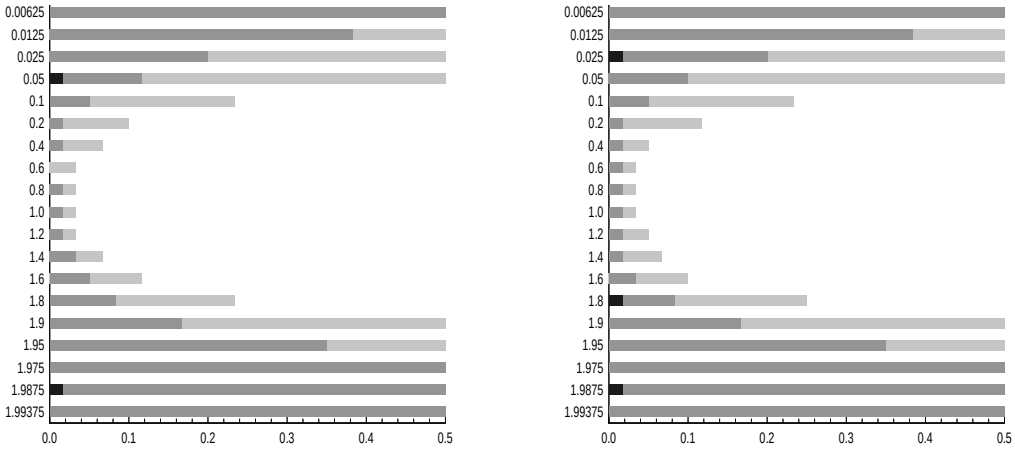


$n = 129 \times 129$

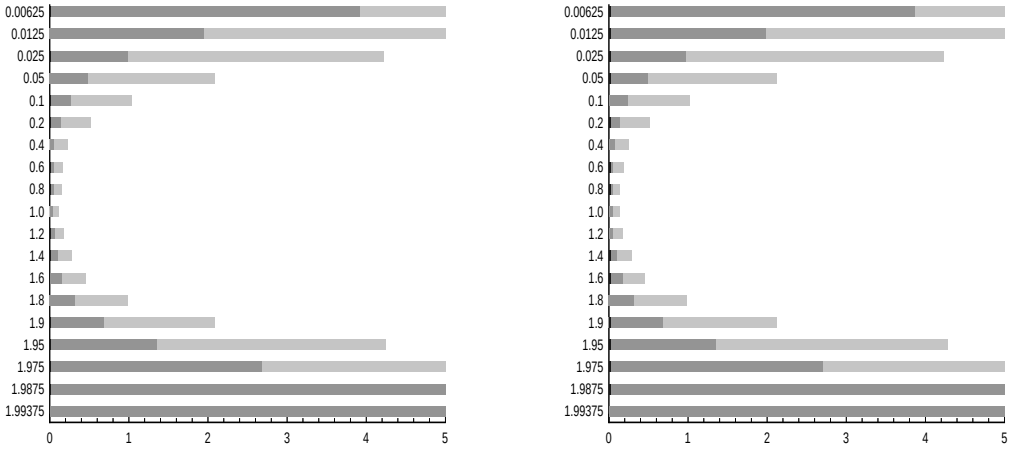




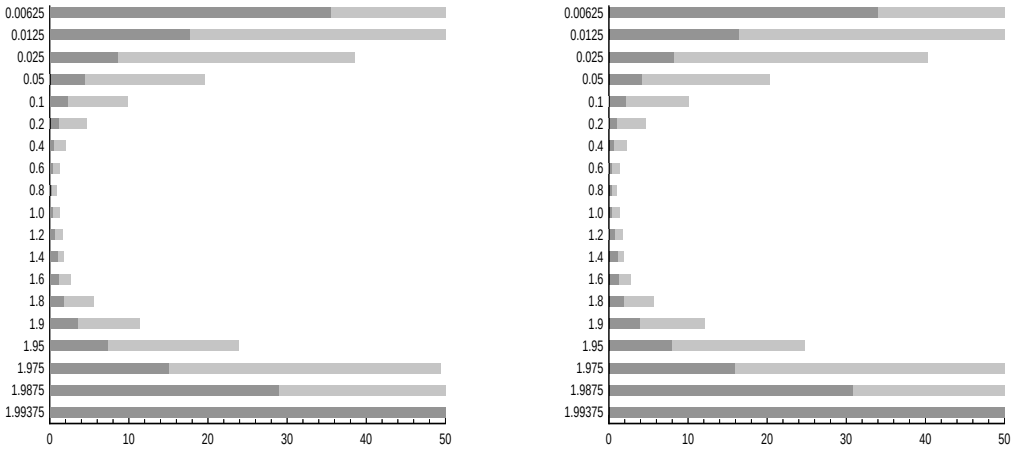
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



$n = 65 \times 65$



$n = 129 \times 129$

4.4.2 GMRES(m)

In diesem Abschnitt wird untersucht, wie sich die Konvergenzeigenschaften des GMRES(m) bei Verwendung unterschiedlicher Werte m für das Restart-Intervall und verschiedener Vorkonditionierungen ändern.
Es wird m über $\{4, 8, 12, 16, 20\}$ variiert. Die von der Speicherkapazität her nicht nötige Beschränkung von m wurde auferlegt, um im folgenden GMRES(m) bei etwa gleichem Speicheraufwand mit anderen PGK–Verfahren vergleichen zu können. Insofern liefern die beobachteten Resultate nur eine bedingte Aussage über die Qualität des GMRES(m) als Lösungsverfahren.
In den folgenden Diagrammen ist auf der x–Achse die Rechenzeit in CPU–Sekunden, auf der y–Achse m aufgetragen. Für jeden Restartwert ist von oben nach unten die Konvergenz dargestellt, die

- ohne Vorkonditionierung,
- bei Jacobi–Vorkonditionierung (Skalierung),
- bei SSOR–Vorkonditionierung mit dem bei der Untersuchung in Abschnitt 4.4.1 jeweils als optimal ermittelten Relaxationsparameter,
- bei ILU(0)–Vorkonditionierung

erreicht wird. Wieder beschreibt

- die Gesamtlänge der Balken die Dauer der Reduktion der Residuennorm um 10^{-6},
- der dunkler abgesetzte Teil die Dauer der Reduktion um 10^{-2} und
- der schwarze Anteil die Länge der Startphase (incl. Konstruktion des Vorkonditionierers).

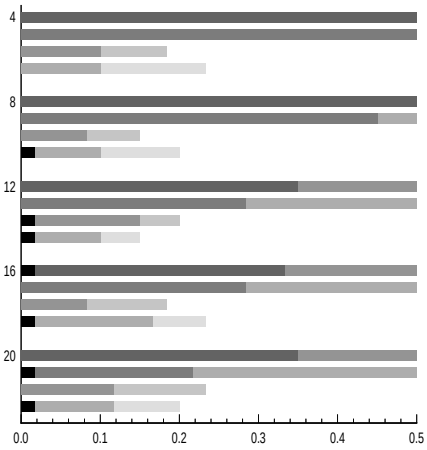
Die Versuche zeigen deutlich den erwarteten Effekt, daß GMRES von einer Vorkonditionierung profitiert. Die Frage nach der besten Vorkonditionierung ist jedoch nicht leicht zu beantworten.
Jacobi–Vorkonditionierung bringt keinen Nutzen, oft wird dadurch die Konvergenz sogar verschlechtert. Dies läßt sich dadurch erklären, daß bei nur geringer Verbesserung der Kondition der Mehraufwand durch Anwendung des Vorkonditionierers insgesamt stärker zu Buche schlägt als die eingesparten Iterationen.
Deutlich stärkere Konvergenzverbesserung wird durch SSOR und ILU(0) erreicht. Daß sie in sehr einfachen Fällen wie bei der reaktionsdominierten Rotationsströmung langsamer konvergieren, liegt daran, daß wegen der Links–Vorkonditionierung kein Abbruchkriterium in der Orthonormalisierung bereitsteht. Es werden immer mindestens m Schritte ausgeführt, die bei der in diesen Fällen schnellen Konvergenz überflüssigen Rechenaufwand erfordern.
Beim Vergleich von SSOR und ILU(0) fällt auf, daß die Leistungsfähigkeit des ILU(0)

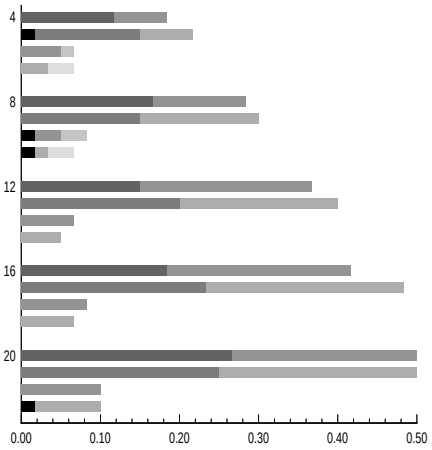
bei Konvektionsdominanz steigt. Für $\varepsilon = 1$ ist regelmäßig die SSOR–vorkonditionierte Variante schneller, im singulär gestörten Fall jedoch die ILU(0)–vorkonditionierte. Dabei bleibt in den meisten Fällen die Zeit der SSOR–Version relativ konstant bezüglich Änderung von ε, während sich die des ILU(0)–vorkonditionierten GMRES oft auf einen Bruchteil verkürzt. ILU(0) ist auch der einzige Vorkonditionierer, der im Fall der Rotationsströmung mit $\varepsilon = 10^{-4}$ zu Konvergenz führt.
Damit wird für den diffusionsdominanten Fall SSOR mit einer sinnvollen Abschätzung von ω_{opt}, bei Konvektionsdominanz ILU(0) empfohlen.

Die Wahl des Restartwertes spielt keine große Rolle, sofern GMRES(m) überhaupt konvergiert. Ein zu großer Wert von m führt zu unnötig erhöhtem Aufwand durch die Orthogonalisierung und die Lösung des Hessenbergsystems.
Wenn das Verfahren jedoch stagniert, was der einzig mögliche Grund für ein Scheitern des GMRES(m) ist, muß m vergrößert werden, sofern entsprechender Speicherplatz bereitsteht.
Die Empfehlung lautet also, den kleinsten Wert m zu wählen, für den GMRES(m) konvergiert. Allerdings deuten einige Ergebnisse darauf hin, daß bei Verwendung von ILU(0) in konvektionsdominanten Fällen eine weitere Erhöhung von m zu Konvergenzbeschleunigung führt.

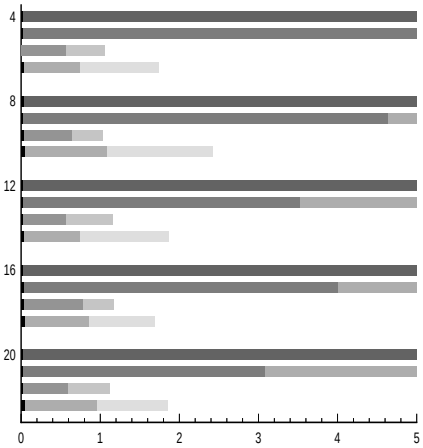
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

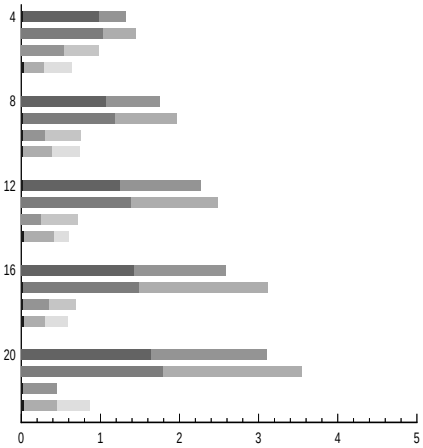
$n = 33 \times 33$



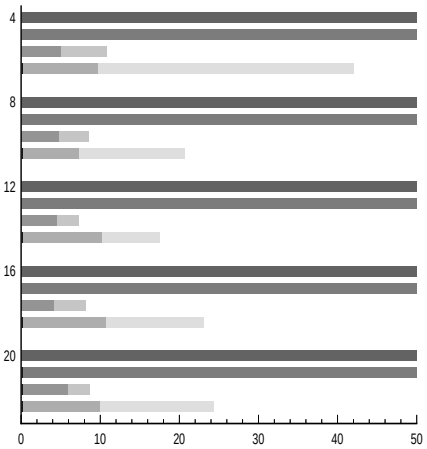


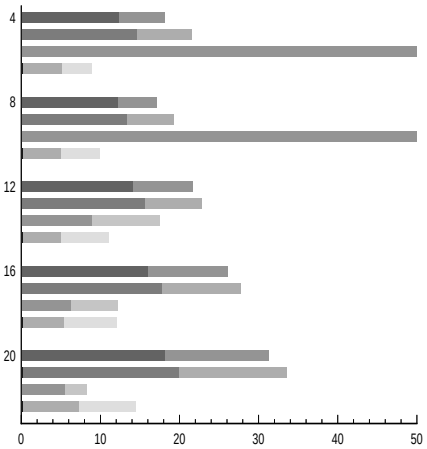
$n = 65 \times 65$



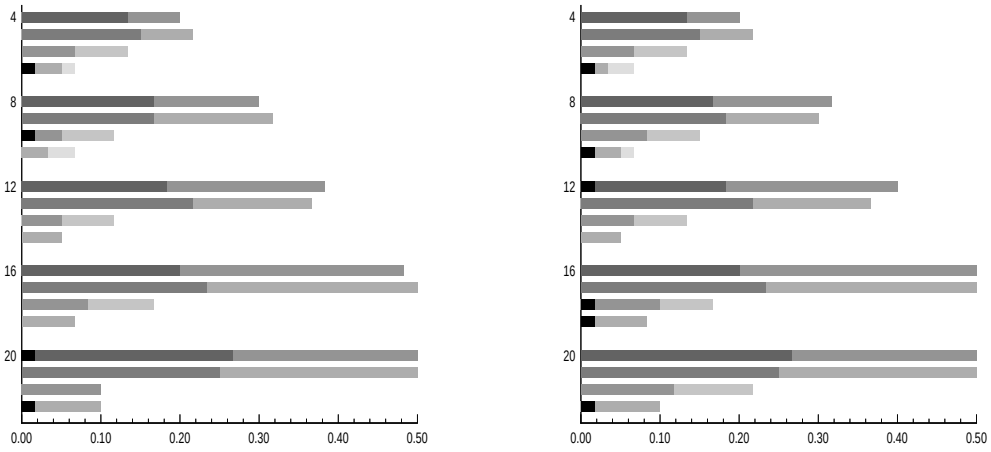


$n = 129 \times 129$

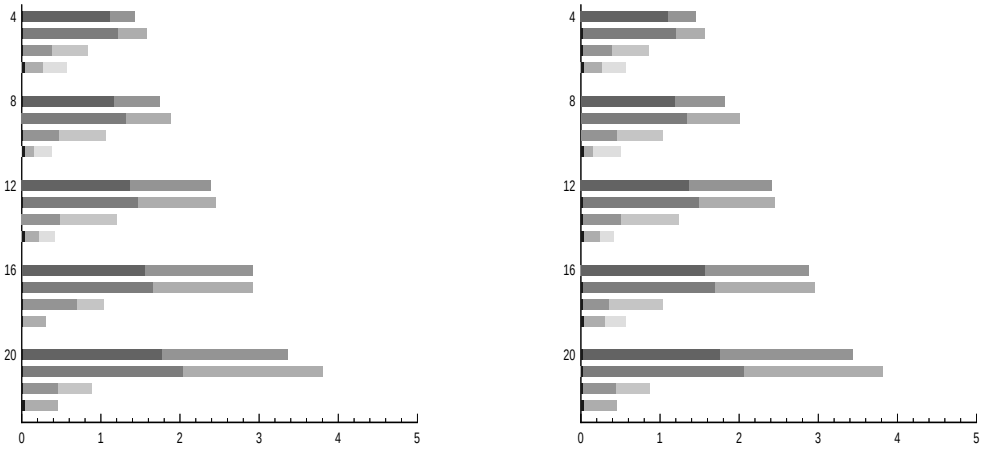




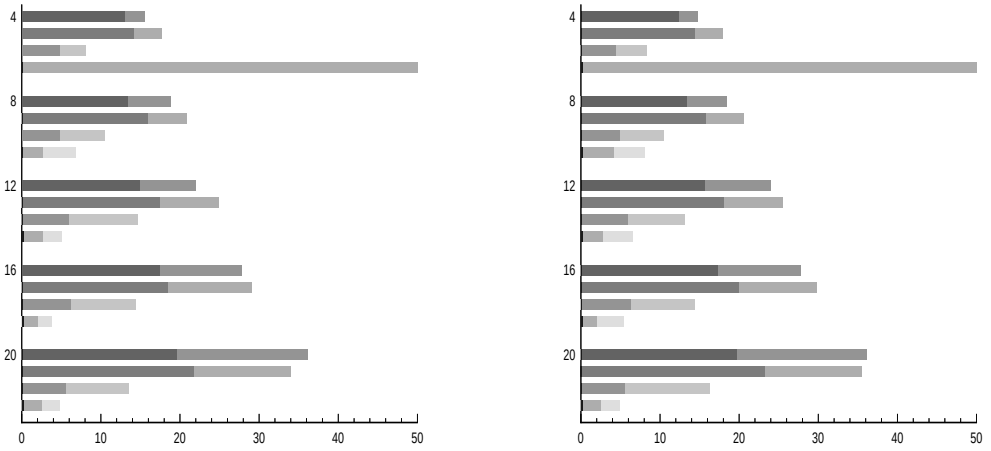
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



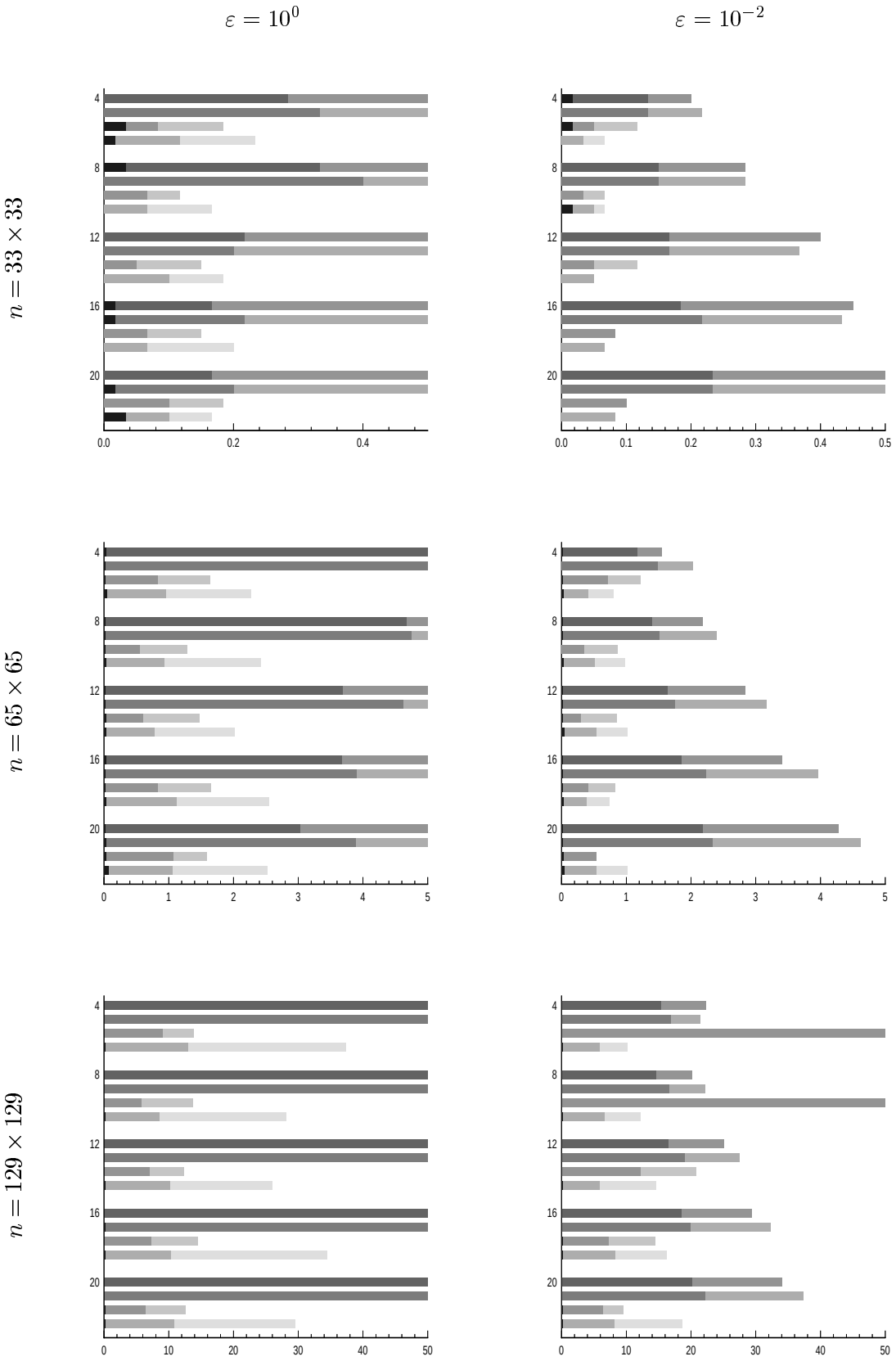
$n = 33 \times 33$



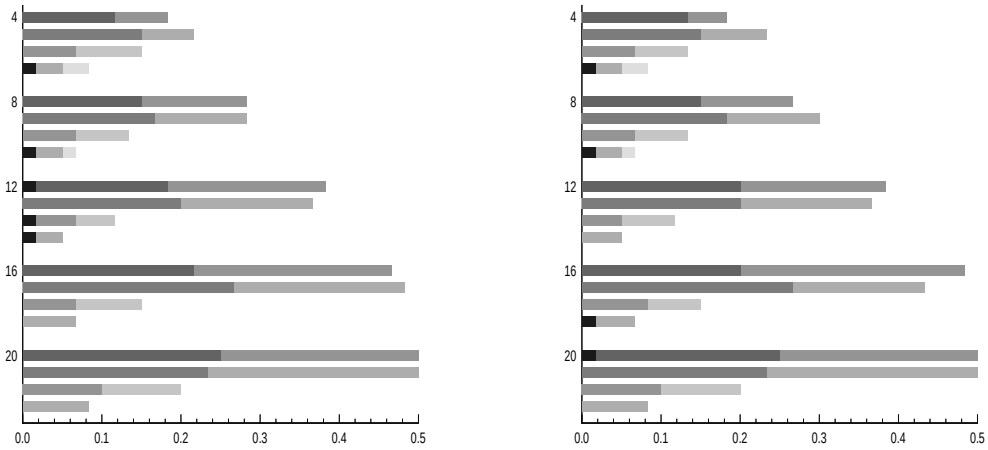
$n = 65 \times 65$



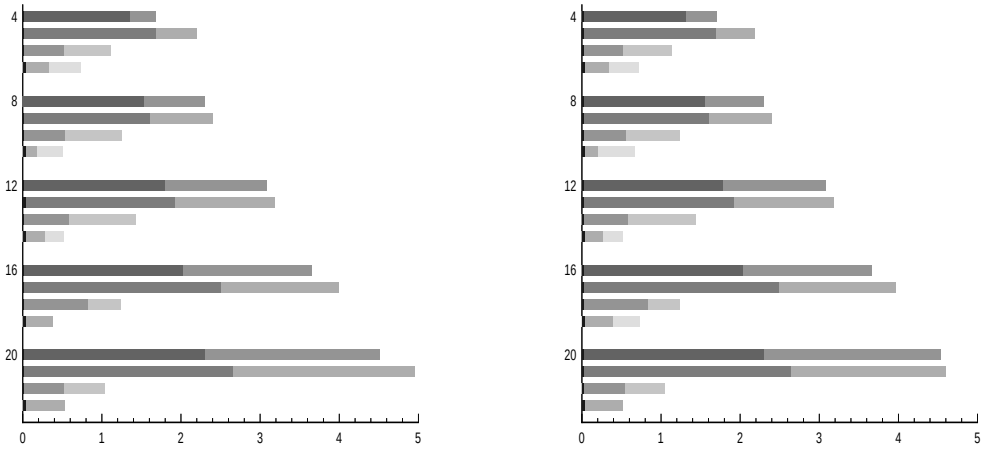
$n = 129 \times 129$


$\varepsilon = 10^{0}$
$\varepsilon = 10^{-2}$
$n = 33 \times 33$
$n = 65 \times 65$
$n = 129 \times 129$

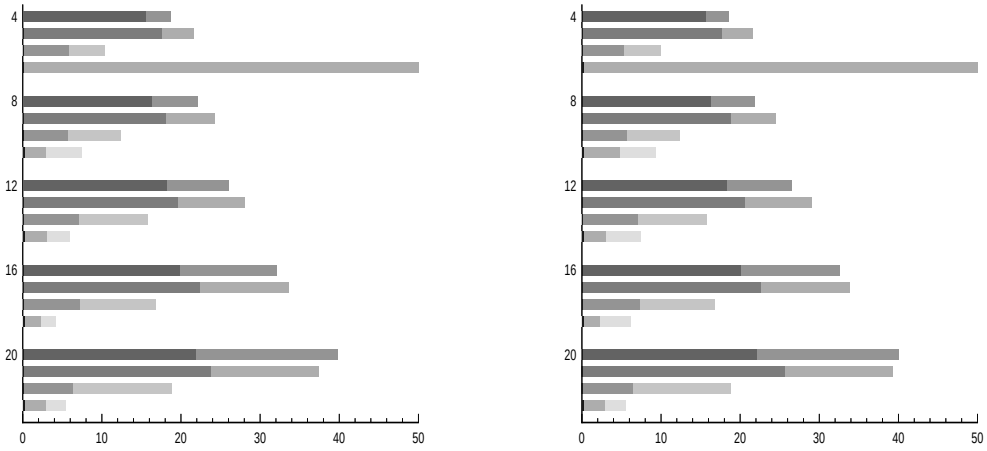
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



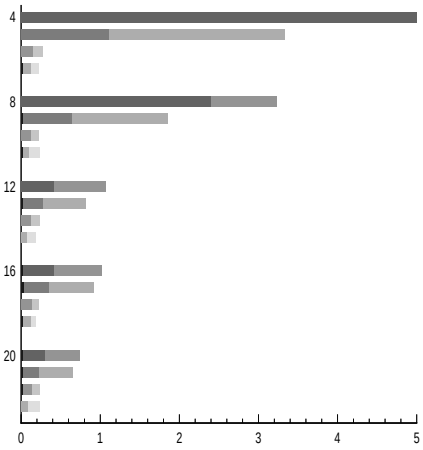
$n = 65 \times 65$

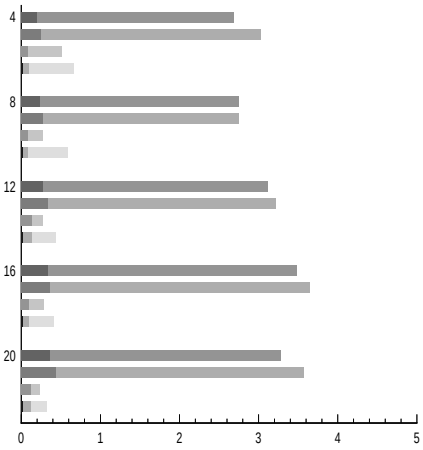


$n = 129 \times 129$

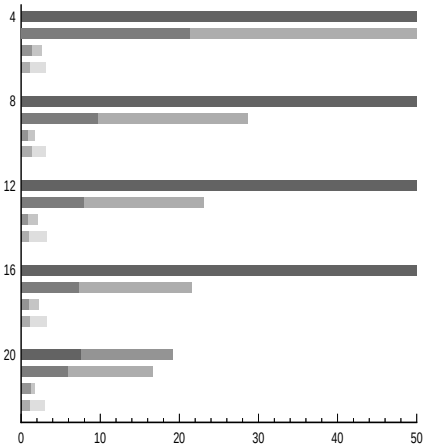
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

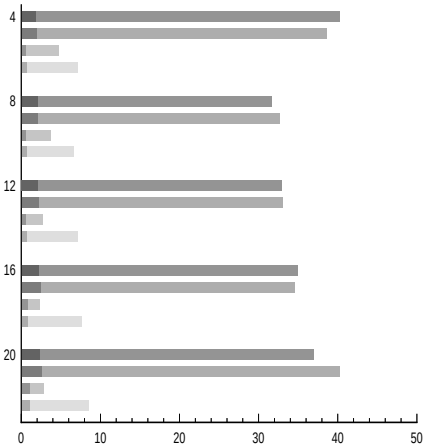
$n = 33 \times 33$



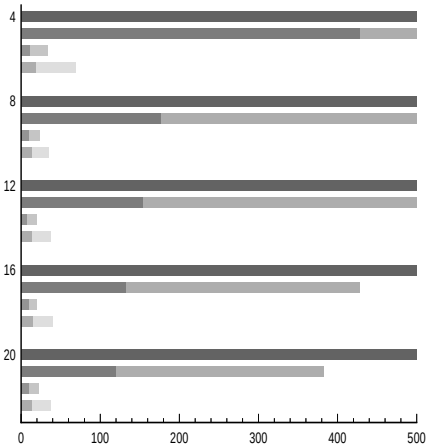


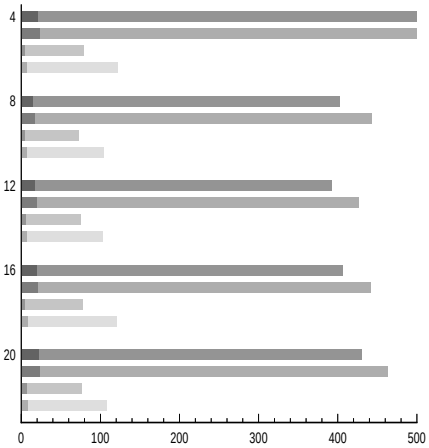
$n = 65 \times 65$



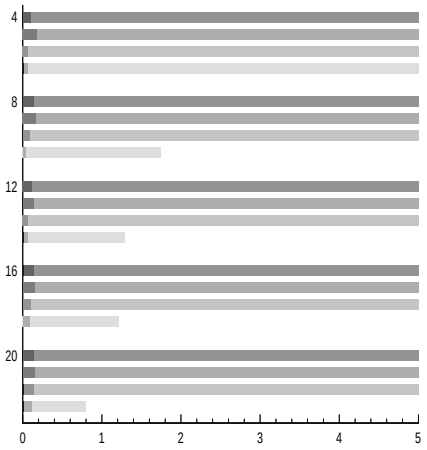


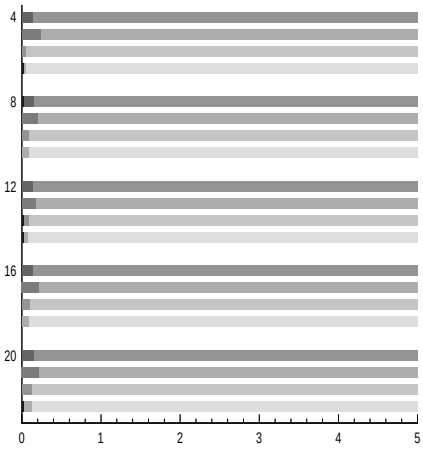
$n = 129 \times 129$



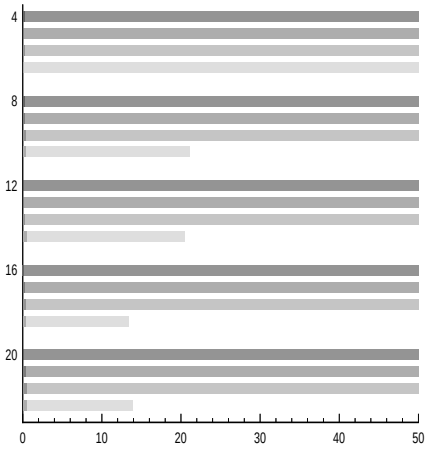


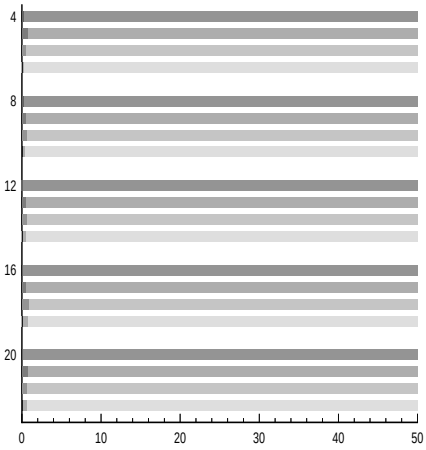
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



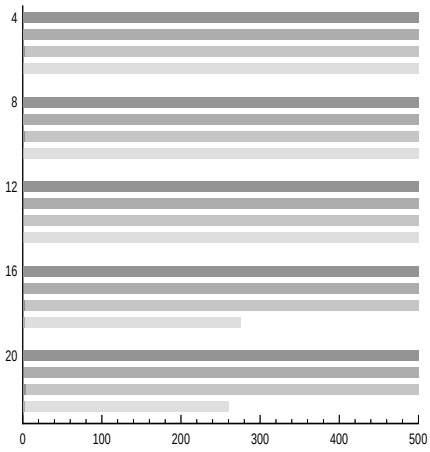


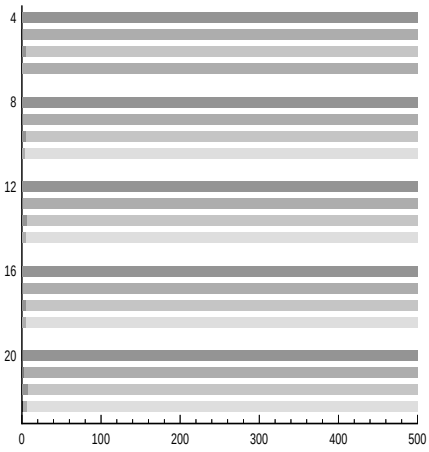
$n = 33 \times 33$





$n = 65 \times 65$

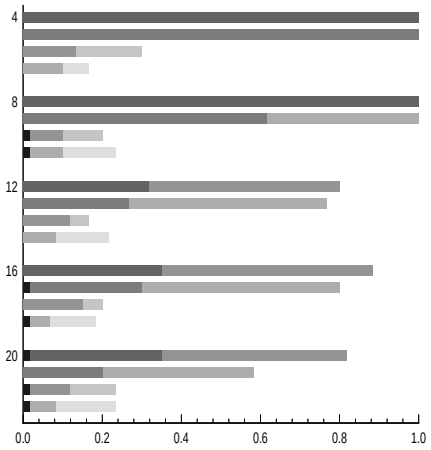


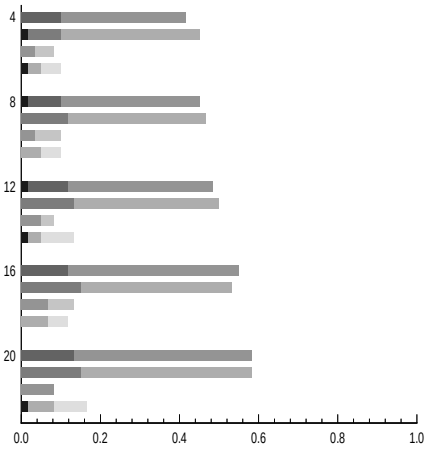


$n = 129 \times 129$

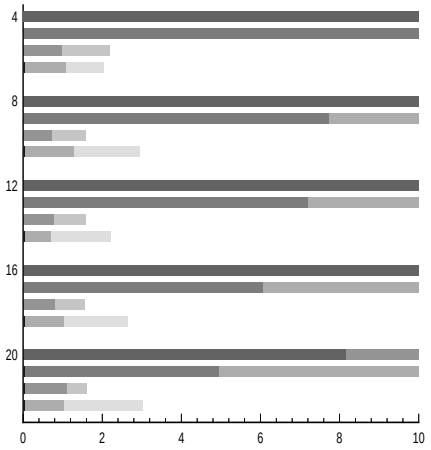
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

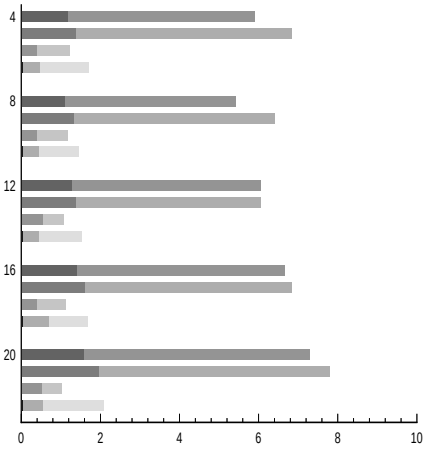
$n = 33 \times 33$



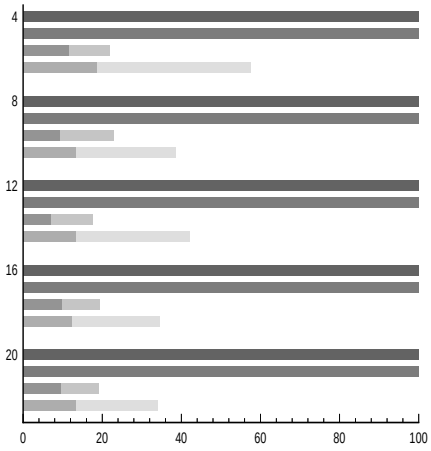


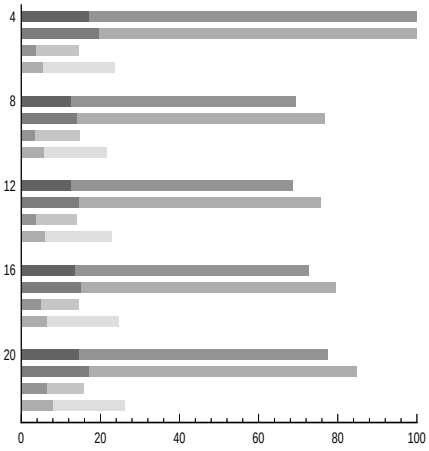
$n = 65 \times 65$



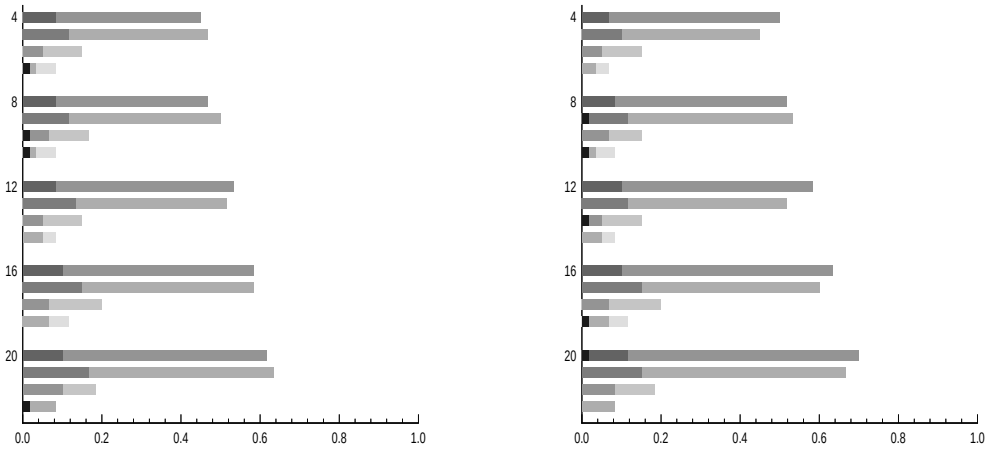


$n = 129 \times 129$

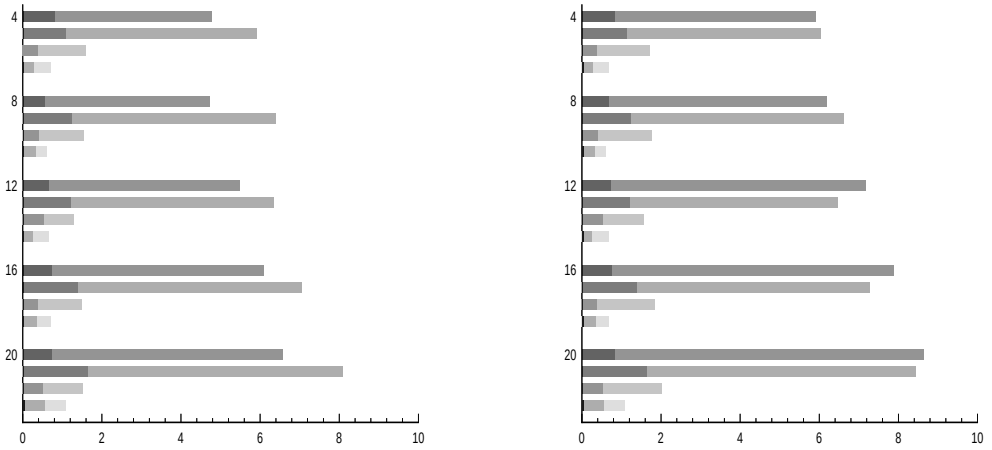




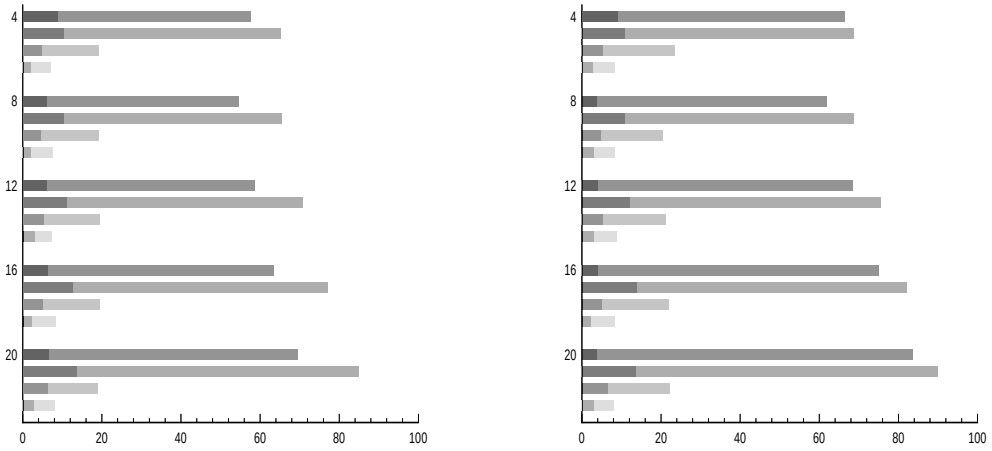
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



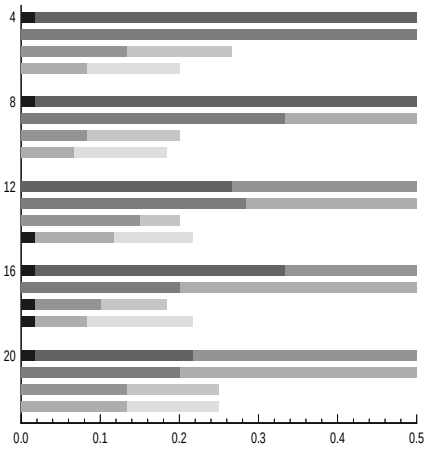
$n = 65 \times 65$

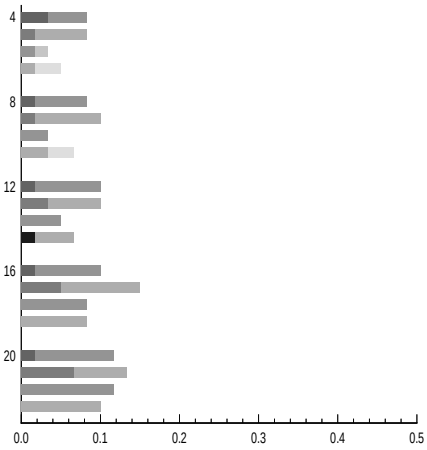


$n = 129 \times 129$

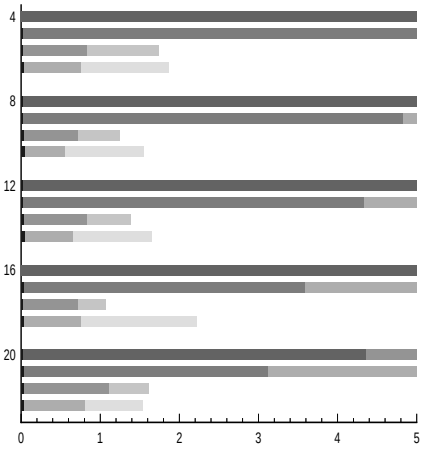
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

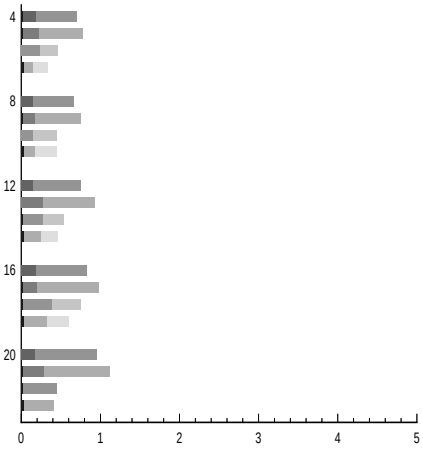
$n = 33 \times 33$



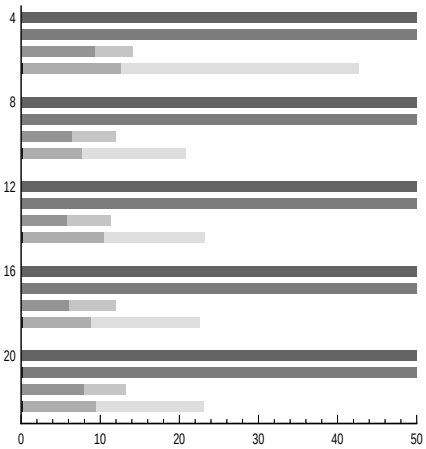


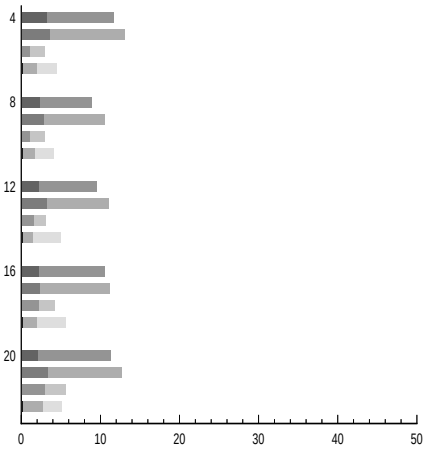
$n = 65 \times 65$



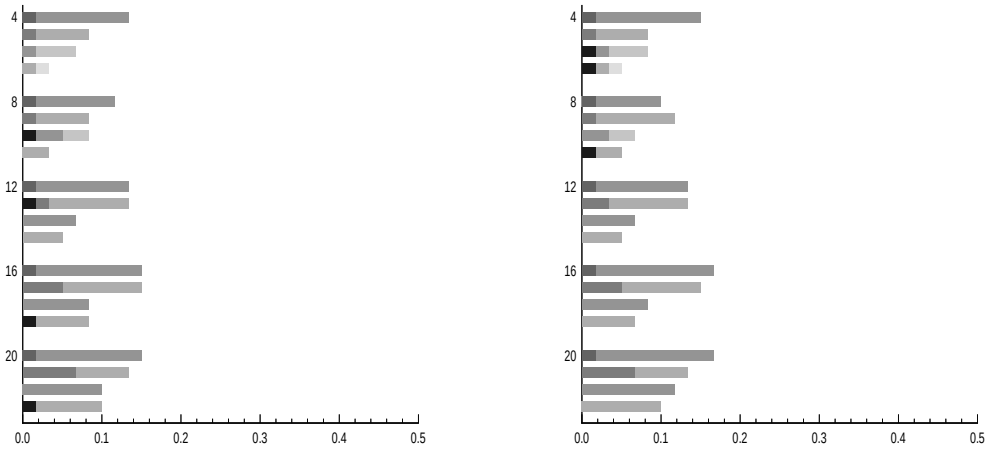


$n = 129 \times 129$

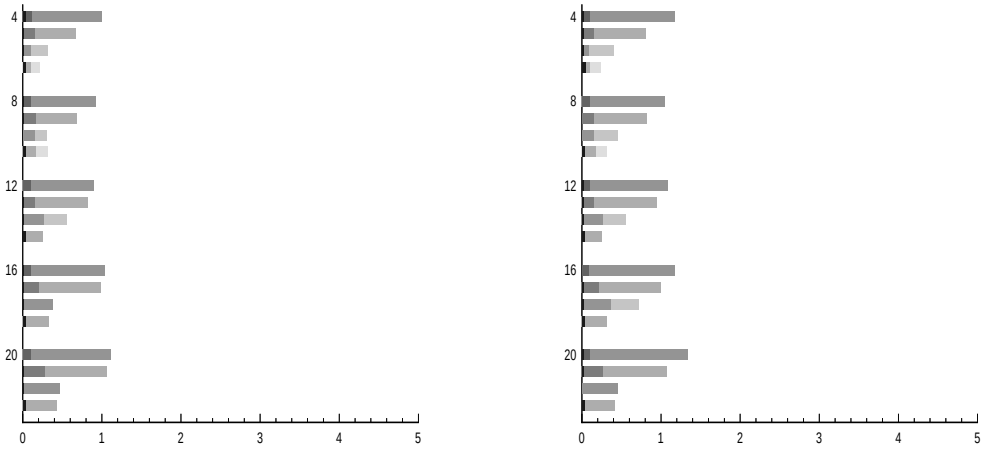




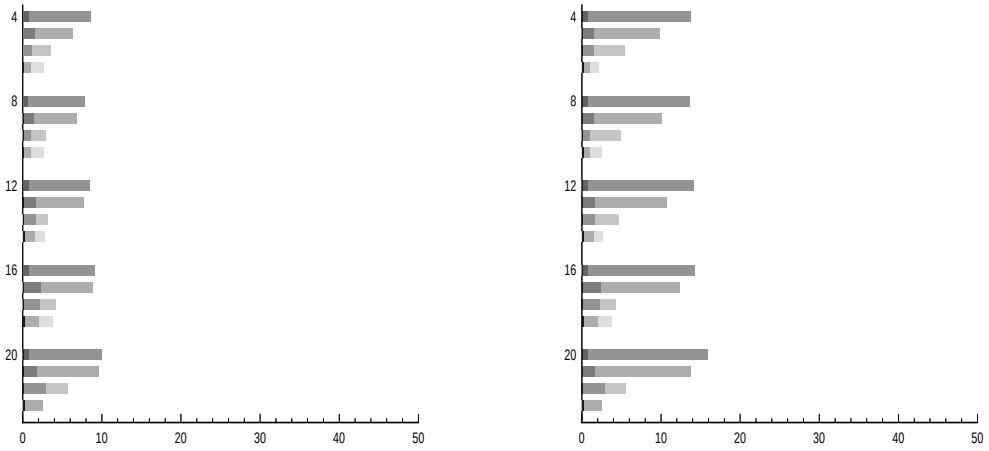
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



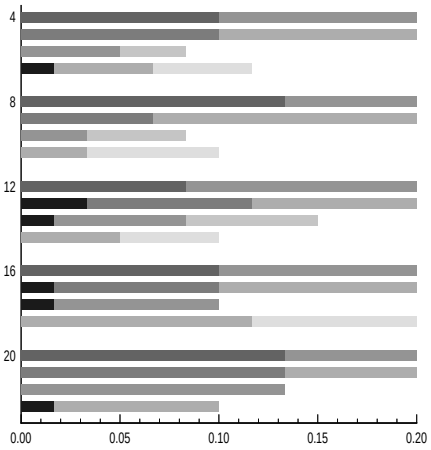
$n = 65 \times 65$

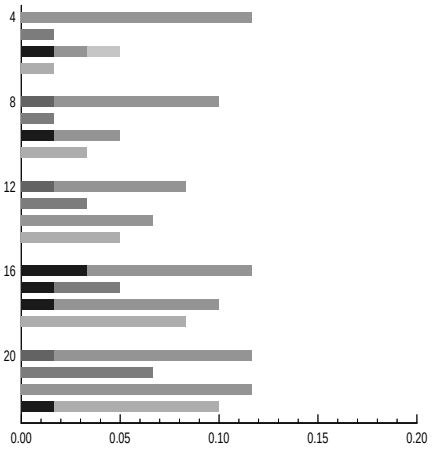


$n = 129 \times 129$

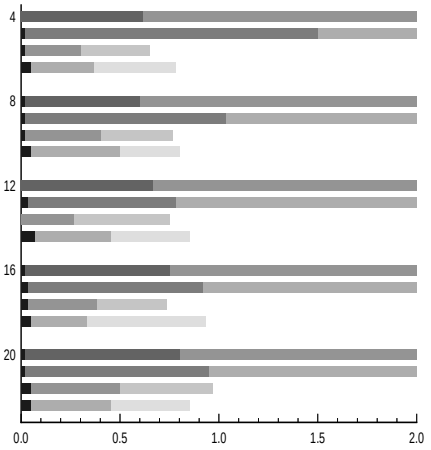
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

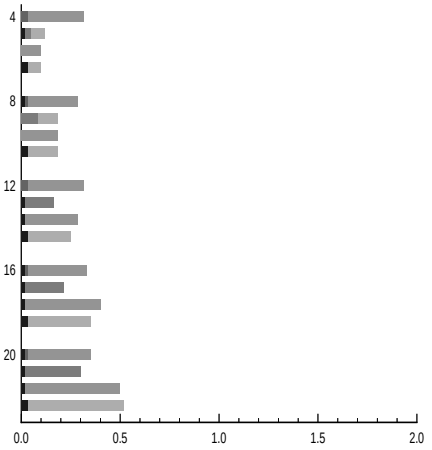
$n = 33 \times 33$



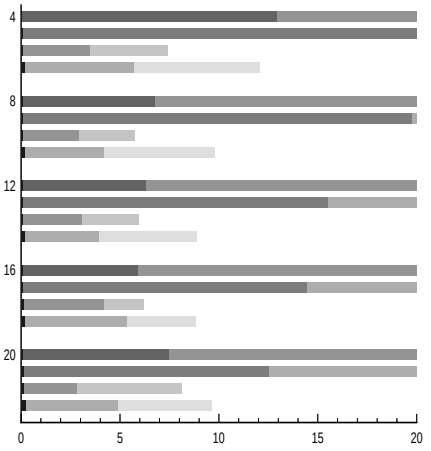


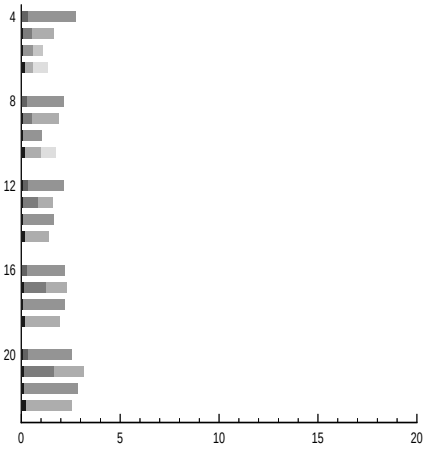
$n = 65 \times 65$



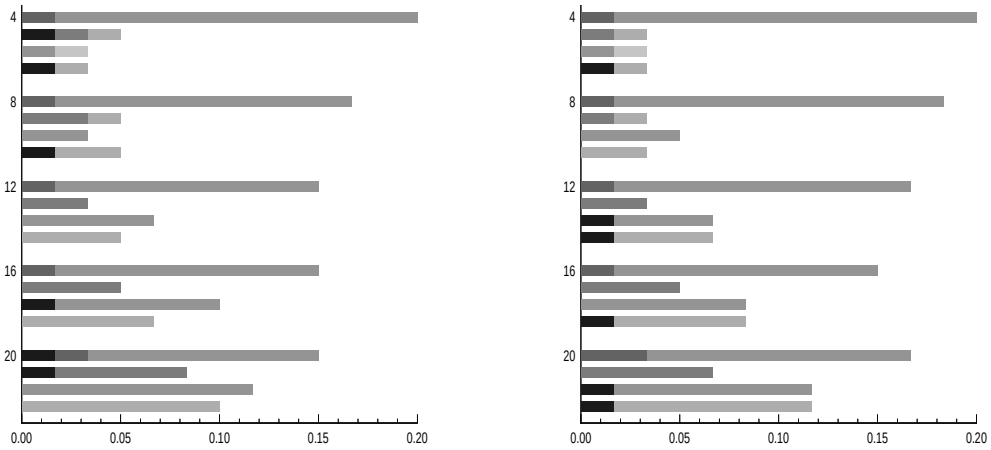


$n = 129 \times 129$

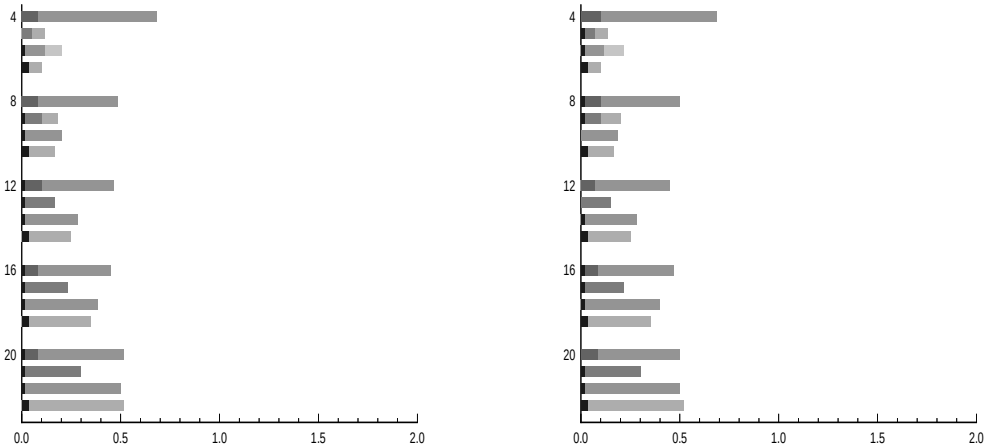




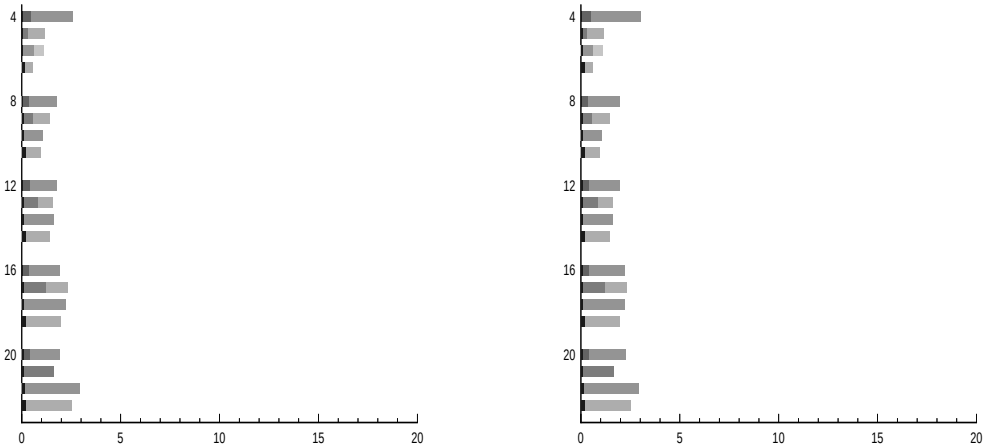
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



$n = 65 \times 65$



$n = 129 \times 129$

4.4.3 Petrov–Galerkin–Krylov–Verfahren

In diesem Abschnitt werden einige PGK–Verfahren bei unterschiedlichen Vorkonditionierungen miteinander verglichen. GMRES(m) wird jeweils mit dem Restart m einbezogen, der in der vorangegangenen Untersuchung in Abschnitt 4.4.2 bei demselben Problem mit irgendeiner Vorkonditionierung die schnellste Konvergenz um 10^{-6} erreicht hat.
In den folgenden Diagrammen ist auf der x–Achse die Rechenzeit in CPU–Sekunden aufgetragen; auf der y–Achse werden von oben nach unten die PGK–Verfahren

- GMRES(m_{opt})
- CGN (=CGNR)
- Bi-CG
- CGS
- Bi-CGSTAB
- QMR
- TFQMR

mit verschiedenen Vorkonditionierungen kombiniert:

- ohne Vorkonditionierung
- Jacobi (Skalierung)
- SSOR mit optimalem Relaxationsparameter
- ILU(0)

Wieder beschreibt

- die Gesamtlänge der Balken die Dauer der Reduktion der Residuennorm um 10^{-6},
- der dunkler abgesetzte Teil die der Reduktion um 10^{-2} und
- der schwarze Anteil die Länge der Startphase.

Die Versuche ergeben folgendes Bild:
Die Jacobi–Vorkonditionierung verbessert in vielen Fällen die Konvergenz deutlich, ohne jedoch die Leistungsfähigkeit von ILU(0) oder SSOR zu erreichen. Oft wird auch die Konvergenz gegenüber dem Verfahren ohne Vorkonditionierung verschlechtert, wobei keine einheitliche Aussage getroffen werden kann, wann Verbesserung und wann Verschlechterung eintritt.

SSOR und ILU(0) liegen in ihrer Leistungsfähigkeit ebenfalls dicht beieinander; ihre Vorteile gegenüber einander lassen sich jedoch klar beschreiben: ILU(0) erreicht, wie für GMRES, mit zunehmender Konvektionsdominanz stärkere Konvergenzbeschleunigung gegenüber SSOR. Während SSOR für $\varepsilon = 1$ meist schnellere Konvergenz erzielt, wird es etwa bei $\varepsilon = 10^{-2}$ bis $\varepsilon = 10^{-3}$ als Diffusionskoeffizienten von ILU(0) „überholt“. Dieser Effekt wird bei feinerer Zerlegung, also größerer Matrix, zugunsten des SSOR verschoben. SSOR besitzt allerdings nur in wenigen Fällen einen relevanten Vorsprung vor ILU(0), so daß man nur dann SSOR bevorzugen sollte, wenn diffusionsdominante Probleme mit relativ feinen Zerlegungen zu berechnen sind. Insbesondere ist ILU(0) naiv anwendbar, während man für sinnvolle Vorkonditionierung durch SSOR eine angemessene Abschätzung von ω benötigt.

Schwieriger ist die Bewertung der Lösungsverfahren. CGN kann ausgeklammert werden, da es generell deutlich schlechter konvergiert als die anderen Verfahren. Unter diesen sind zwar Bi-CGSTAB und CGS regelmäßig die schnellsten, doch ist ihr Vorsprung zu den anderen Verfahren nicht so groß, daß sich daraus eine eindeutige Rangfolge ergäbe. Vielmehr müssen weitere numerische Eigenschaften der Verfahren berücksichtigt werden, die mindestens ebenso wichtig sind, um Empfehlungen für bestimmte Anwendungen auszusprechen:

- Bi-CG, CGS und Bi-CGSTAB besitzen den gemeinsamen Nachteil starker Oszillationen im Konvergenzverlauf (vgl. Anhang). Diese führen wegen starker Größenordnungsunterschiede der in den Rechnungen auftretenden Werte zu Rundungsfehlern, die durch die Formulierung der Algorithmen durch Rekursionen schnell untragbar hoch werden können. x_k und r_k werden unabhängig voneinander rekursiv berechnet, so daß durch Rundungsfehler bald $r_k \neq b - Ax_k$. Konvergenz bezüglich $\|r_k\|$ ist dann eine wenig hilfreiche Aussage. Eine systematische Untersuchung dieser Effekte kann hier jedoch aus Platzgründen nicht gegeben werden. Von diesen Verfahren sollte also Bi-CGSTAB verwendet werden, da es bei glattestem Konvergenzverlauf meist am schnellsten konvergiert. Aber auch dieses sollte nur dann eingesetzt werden, wenn sich die Oszillationen in Schranken halten, bzw. nicht zu oft iteriert wird.
  Übrigens erzielte Bi-CGSTAB mit SSOR–Vorkonditionierung bei Ausdehnung des Beobachtungszeitraumes im Härtefall der Rotationsströmung mit $\varepsilon = 10^{-6}$ und $n = 129 \times 129$ als erstes Verfahren eine Reduktion der Residuennorm um den Faktor 10^{-6}, und zwar nach 800 Sekunden. Die anderen Verfahren konvergierten jedoch selbst nach 2000 Sekunden noch nicht.

- QMR und TFQMR führen eine Quasi–Minimierung durch, so daß Oszillationen unterbunden werden. Da TFQMR prinzipiell effizienter als QMR ist, was sich auch in den Experimenten (wie bei Bi-CG und CGS) zeigt, sollte dieses verwendet werden. Es konvergiert bei adäquater Vorkonditionierung in der Regel zwar langsamer als Bi-CGSTAB, aber dann nur etwa um den Faktor 1.5. Bei Problemen, die unter Bi-CGSTAB durch Rundungsfehler betroffen sind, ist TFQMR also eine gute Alternative.

Leider konnte QMRCGSTAB nicht mehr in die Untersuchungen einbezogen werden. Nach Ergebnissen in [32] liegt dessen Konvergenzgeschwindigkeit bei ILU(0)–Vorkonditionierung zwischen der des Bi-CGSTAB und der des TFQMR. Damit entspricht die Reihenfolge nach Konvergenzgeschwindigkeit der QMR–Verfahren mit QMR – TFQMR – QMRCGSTAB gerade der der zugrundeliegenden Methoden Bi-CG – CGS – Bi-CGSTAB, die jeweils etwas schneller als das QMR–Äquivalent konvergieren.

- GMRES(m) bietet unter geeigneter Vorkonditionierung bei Konvergenz konkurrenzfähige Geschwindigkeit. Die Minimierungseigenschaft für das Residuum sowie die Konstruktion der Krylov–Basis auf explizit vorliegenden Vektoren (gegenüber den Rekursionen der Lanczos–Verfahren) gewährleistet numerisch relativ stabile Berechnungen, die ggf. durch Verwendung von Householder–Transformationen statt der Gram–Schmidt–Orthonormalisierung noch robuster gemacht werden können. Es neigt aber bei gleichem Speicheraufwand leichter zum Versagen durch Stagnation als die Lanczos–Verfahren zum Versagen durch Breakdown. Bei geringer Speicherkapazität sollten daher diese vorgezogen werden.
  GMRES(m) besitzt jedoch die besondere Qualität, daß zusätzlich vorhandener Speicherplatz zur Stabilisierung des Verfahrens eingesetzt werden kann, während bei allen anderen Verfahren ein Versagen prinzipiell ist. So konnte das unter dem ersten Punkt erwähnte Problem durch GMRES(m) mit ILU(0)–Vorkonditionierung durch die bezüglich der Konvergenzgeschwindigkeit optimale Wahl von $m := 406$ in 344 Sekunden gelöst (d.h. die Residuennorm um den Faktor 10^{-6} reduziert) werden. Die Wahl eines optimalen m ist a–priori zwar nicht möglich, doch besteht durch Ausnutzung des verfügbaren Speichers mit GMRES(m) bei Vernachlässigung der erzielten Effizienz überhaupt eine Möglichkeit zur Lösung von extrem komplexen Problemen.

Die Bewertung der Lösungsverfahren anhand der Konvergenz der Residuennorm ist durch die Übertragbarkeit der Schlußfolgerungen auf die Praxis motiviert. Es soll nun aber auch aus den Ergebnissen eine empirische, am relativen Fehler orientierte Abbruchkontrolle formuliert werden. Interessanterweise stellt sich nämlich heraus, daß für jedes Verfahren zum Zeitpunkt der Reduktion der Residuennorm um den Faktor 10^{-6} der relative Fehler einen für alle Probleme ungefähr gleichen Wert hat, und zwar

$\approx 10^{-4}$ für GMRES(m), CGN und Bi-CGSTAB,

$\approx 10^{-6}$ für Bi-CG und CGS,

$\approx 10^{-7}$ für QMR und TFQMR.

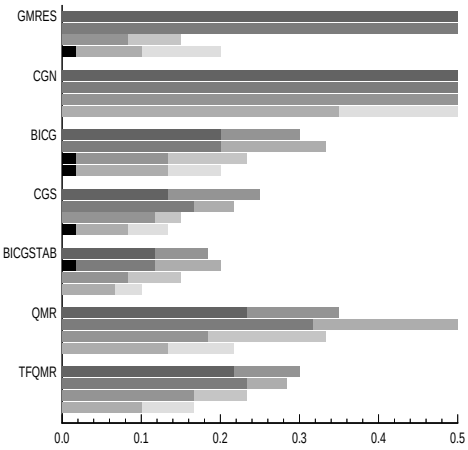
GMRES(m) zum Beispiel muß also bezüglich der Residuennorm um etwa drei Größenordnungen genauer lösen, um denselben Fehler zu erhalten wie QMR. Diese Gruppierung der Verfahren läßt sich dadurch beschreiben, daß die Verfahren unter dem ersten Punkt das Residuum minimieren, die unter dem dritten es quasi–minimieren und die unter dem zweiten keine Minimierung durchführen. Dadurch erhalten die durch die

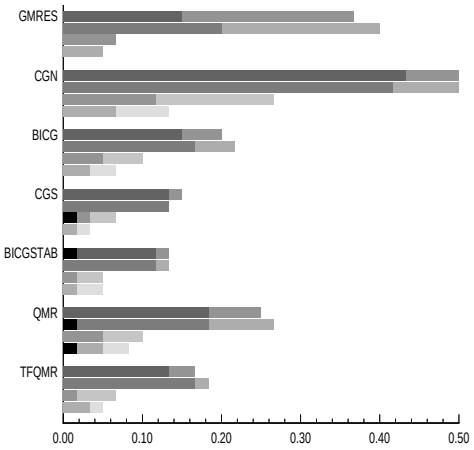
Verfahren erzielten Konvergenzeigenschaften ganz neue Bedeutung. Da jedoch keine Begründung für solche Zusammenhänge gegeben werden kann, soll diese Einordnung als empirischer Vorschlag zur Reformulierung von Konvergenzkriterien stehenbleiben. Abschließend wird die jeweilige Dauer einer Iteration der Verfahren für die Dimension $n = 65 \times 65$ in CPU–Millisekunden angegeben:

Vorkonditionierung	*Keine*	*Jacobi*	*SSOR*	*ILU(0)*
GMRES(4)	1.84	2.16	4.86	4.10
GMRES(20)	2.84	3.10	5.69	4.79
CGN	2.66	2.88	9.58	6.70
Bi-CG	2.84	3.13	7.38	6.01
Bi-CGSTAB	3.06	3.53	7.52	6.41
QMR	3.06	3.06	7.52	6.41
TFQMR	1.66	1.98	5.15	4.32

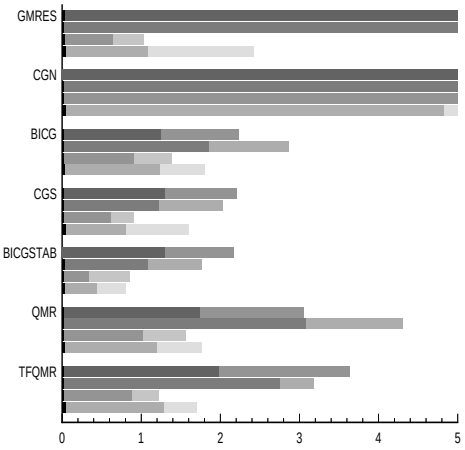
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

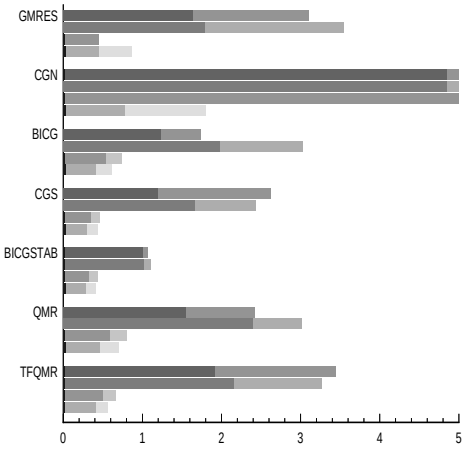
$n = 33 \times 33$



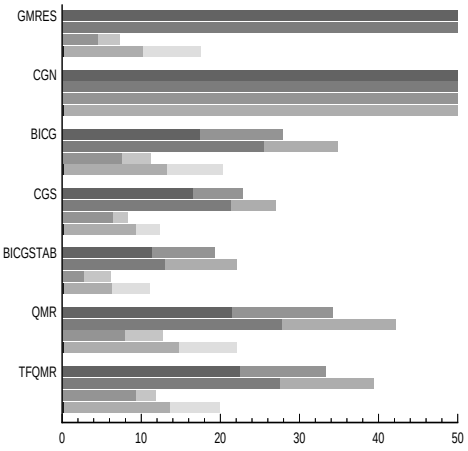


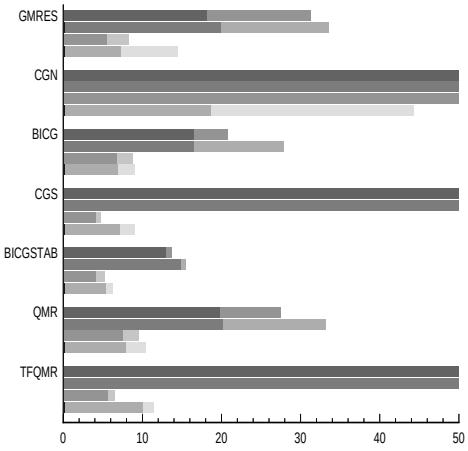
$n = 65 \times 65$



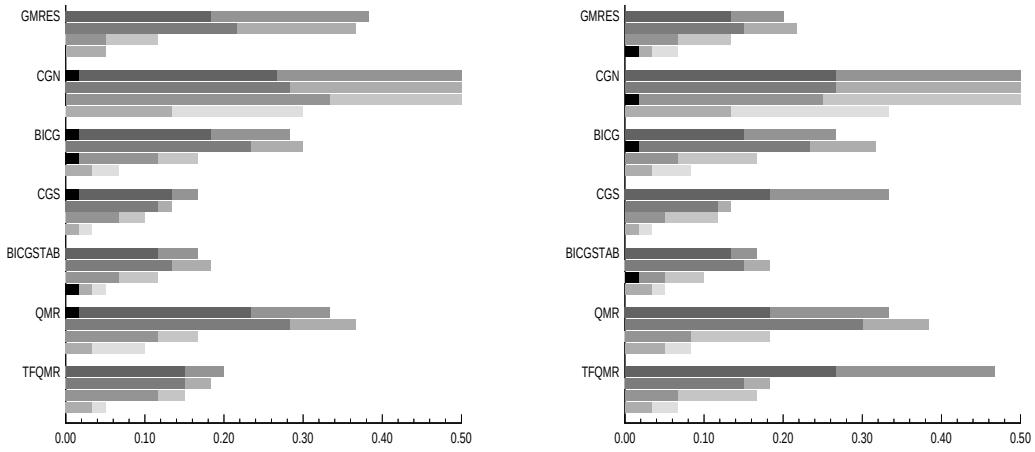


$n = 129 \times 129$

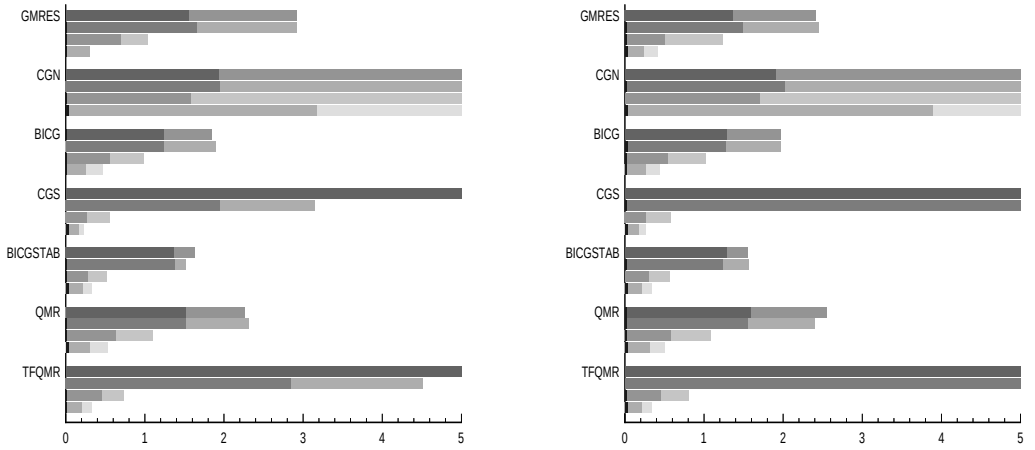




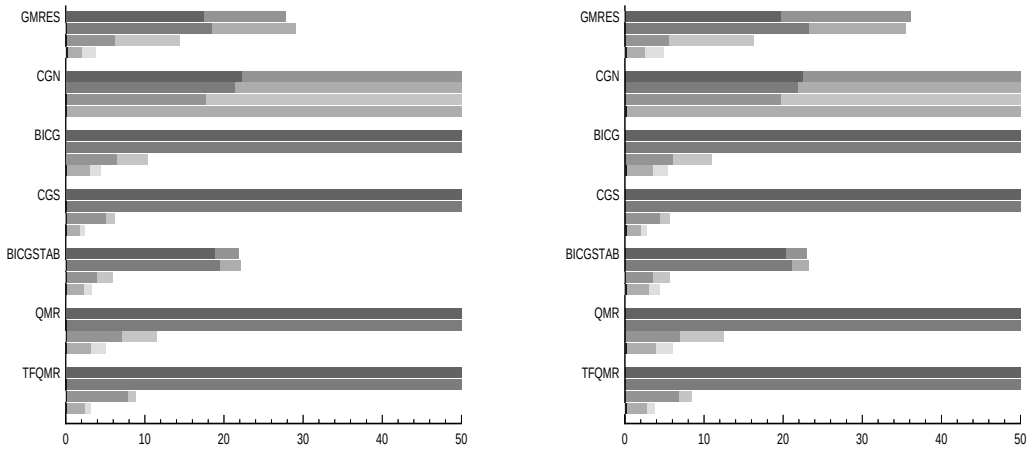
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



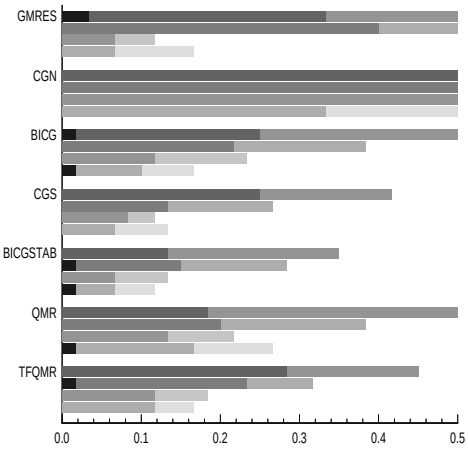
$n = 65 \times 65$

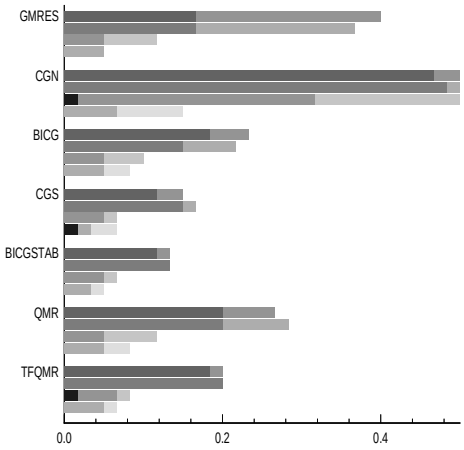


$n = 129 \times 129$

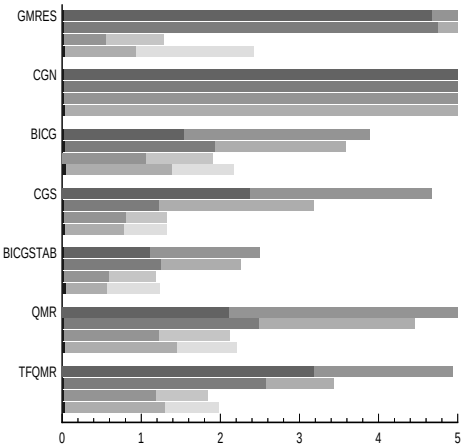
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

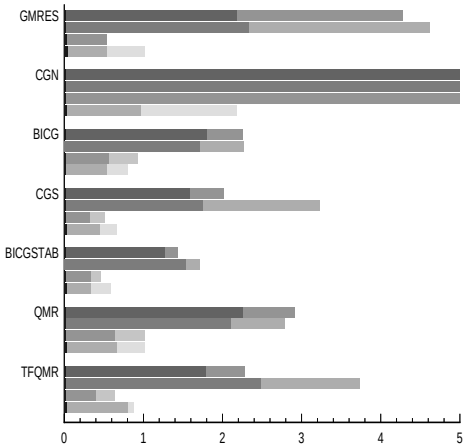
$n = 33 \times 33$



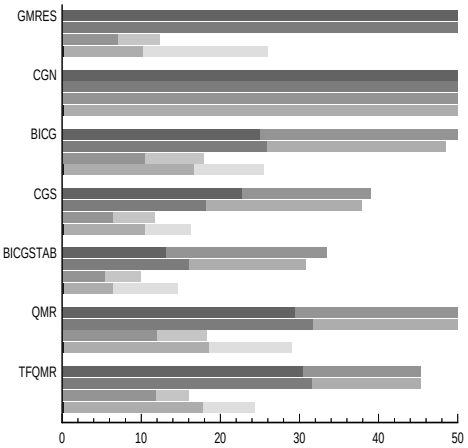


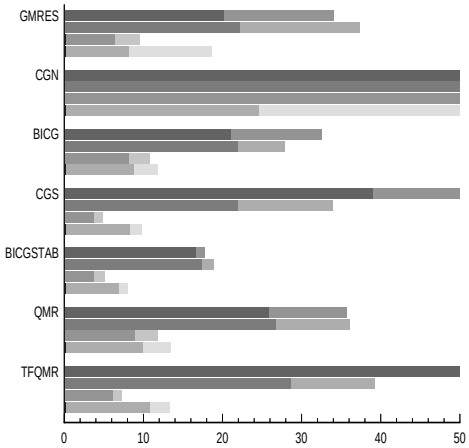
$n = 65 \times 65$



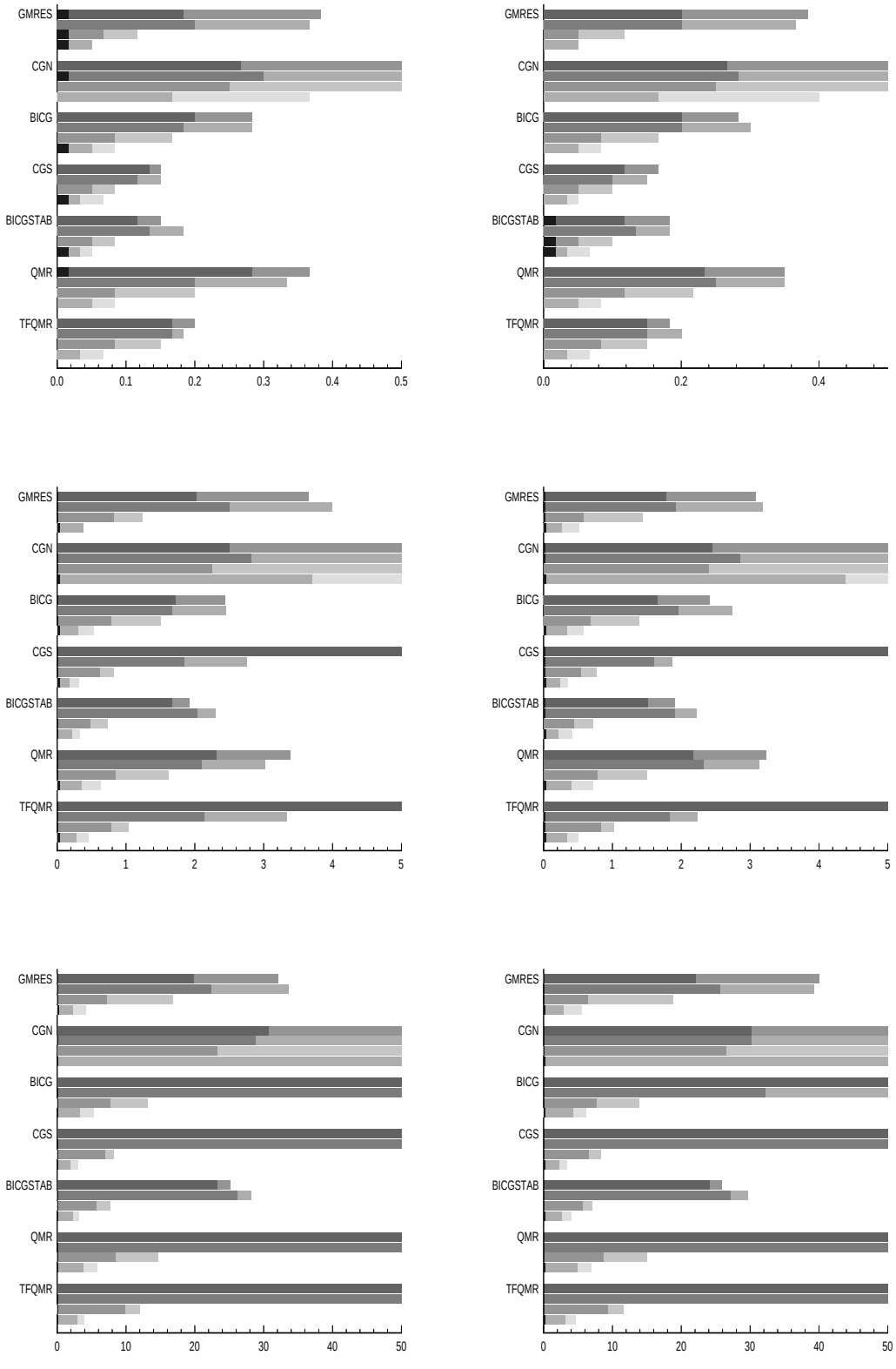


$n = 129 \times 129$





$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



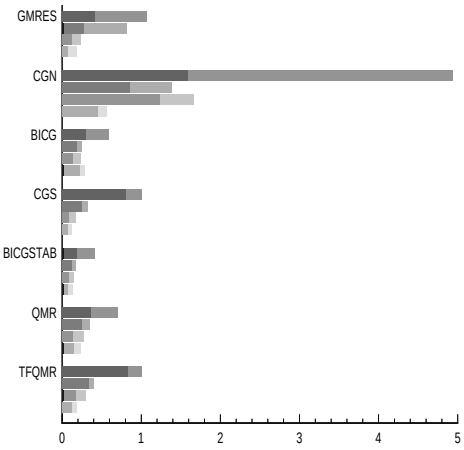
$n = 33 \times 33$

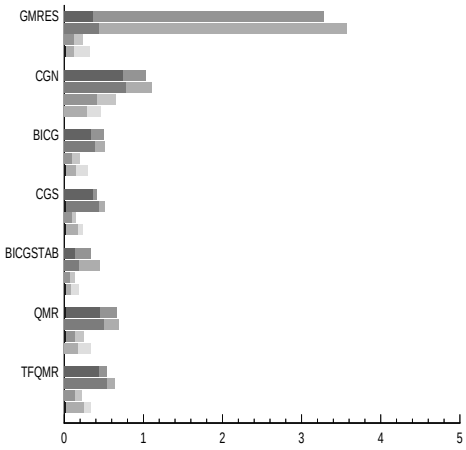
$n = 65 \times 65$

$n = 129 \times 129$

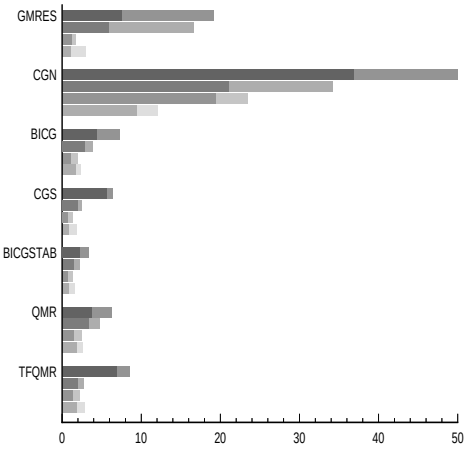
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

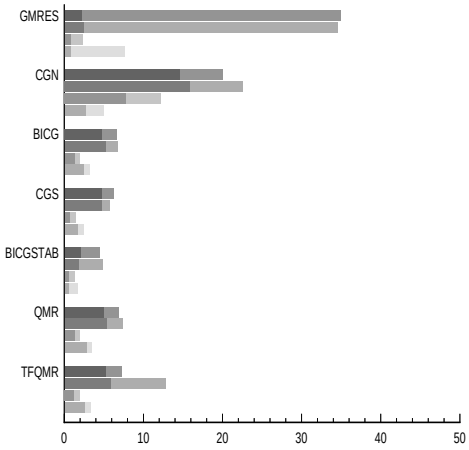
$n = 33 \times 33$



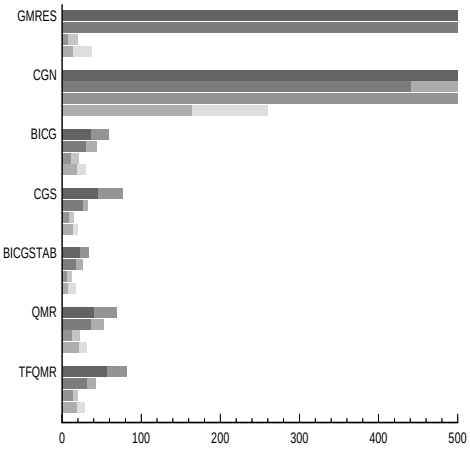


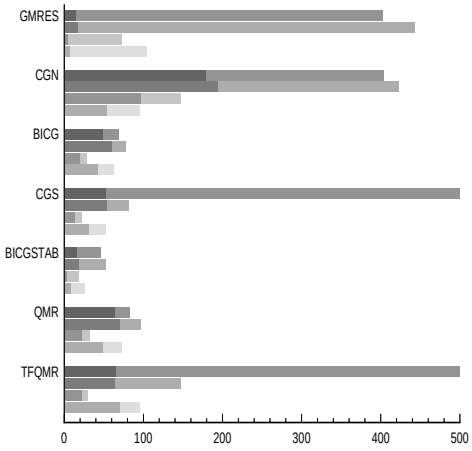
$n = 65 \times 65$



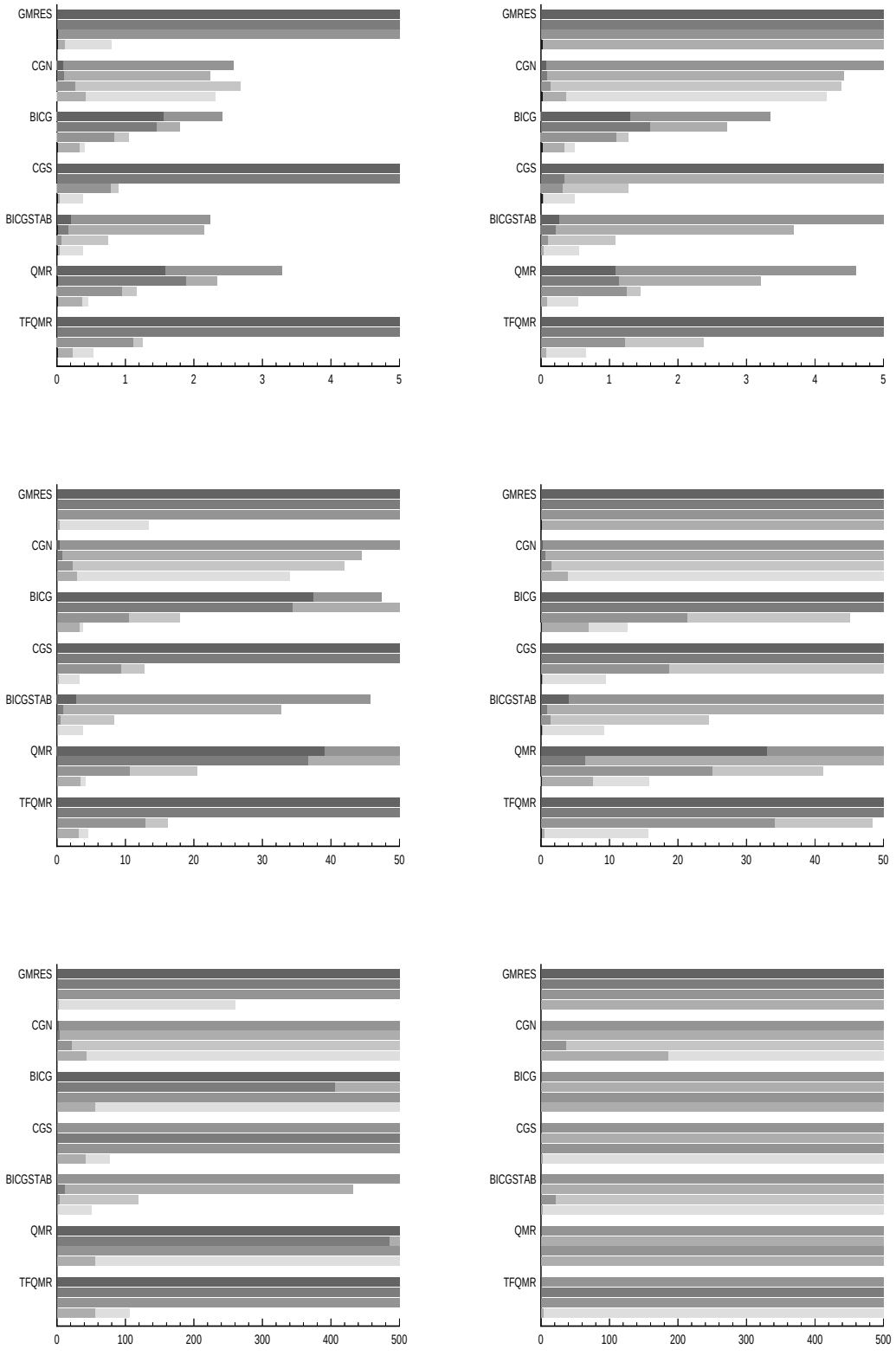


$n = 129 \times 129$





$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



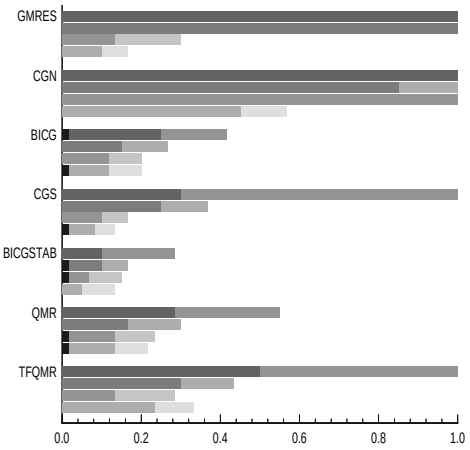
$n = 33 \times 33$

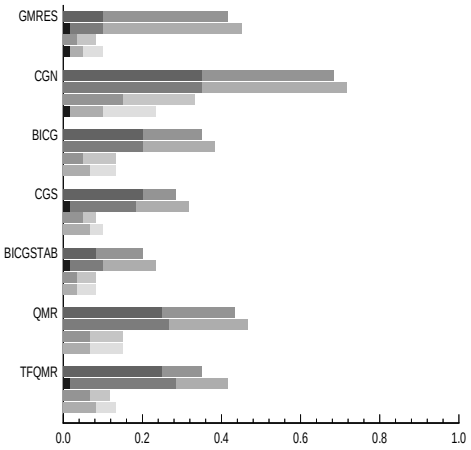
$n = 65 \times 65$

$n = 129 \times 129$

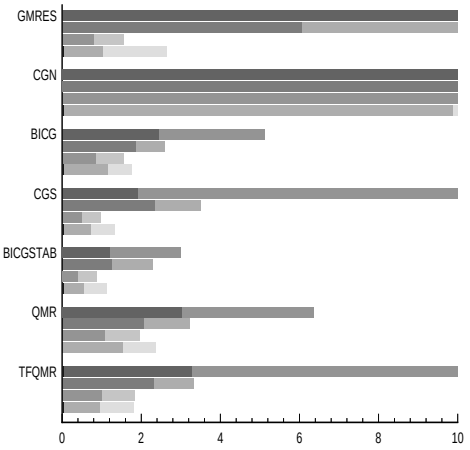
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

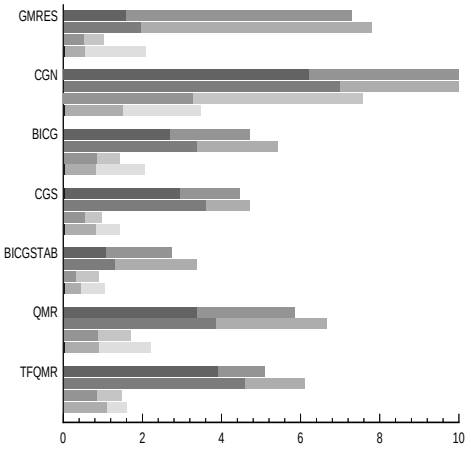
$n = 33 \times 33$



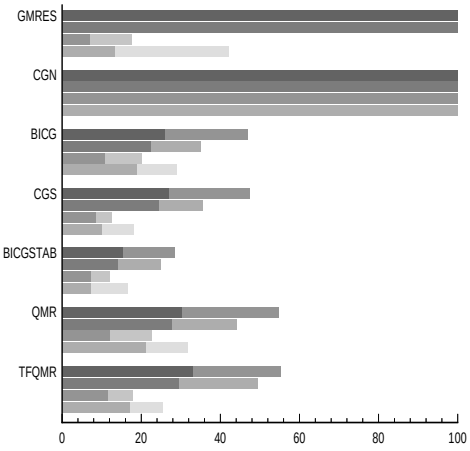


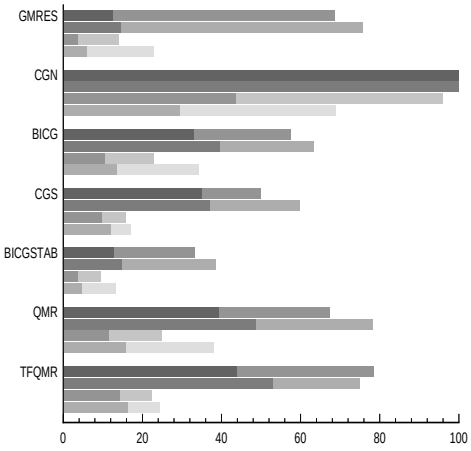
$n = 65 \times 65$



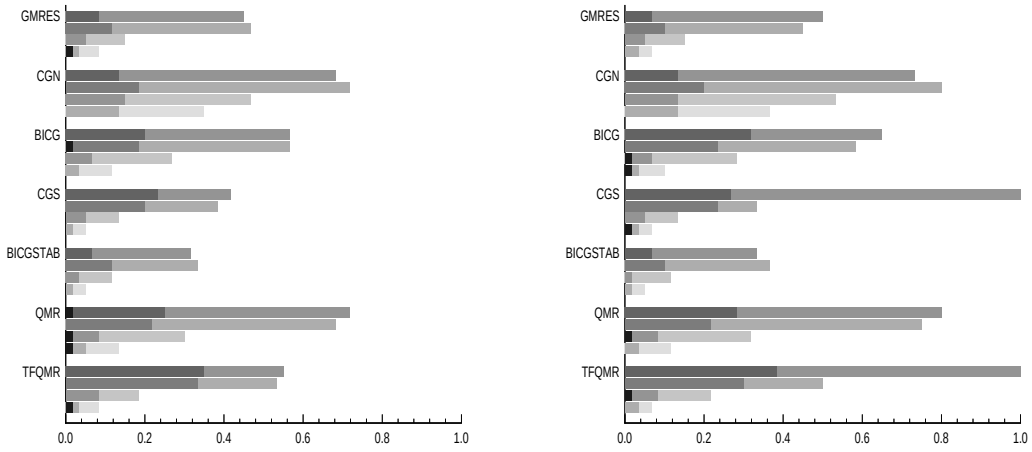


$n = 129 \times 129$

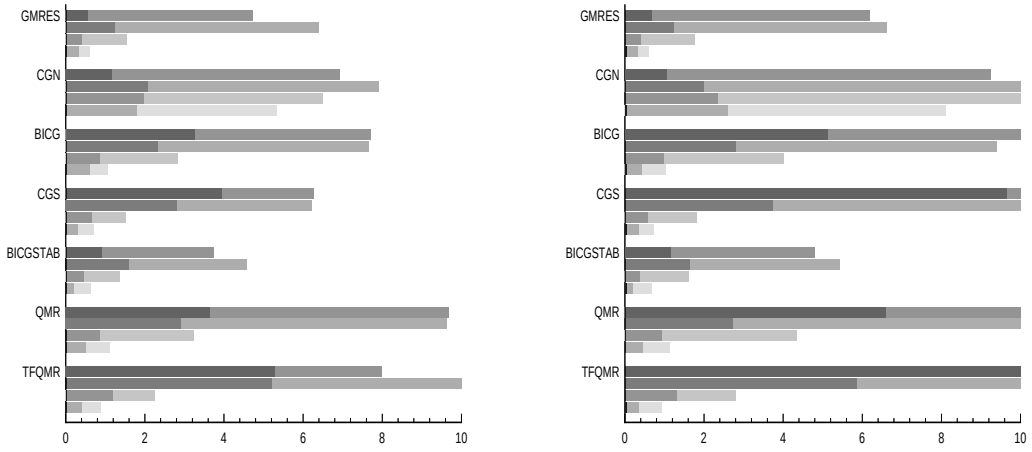




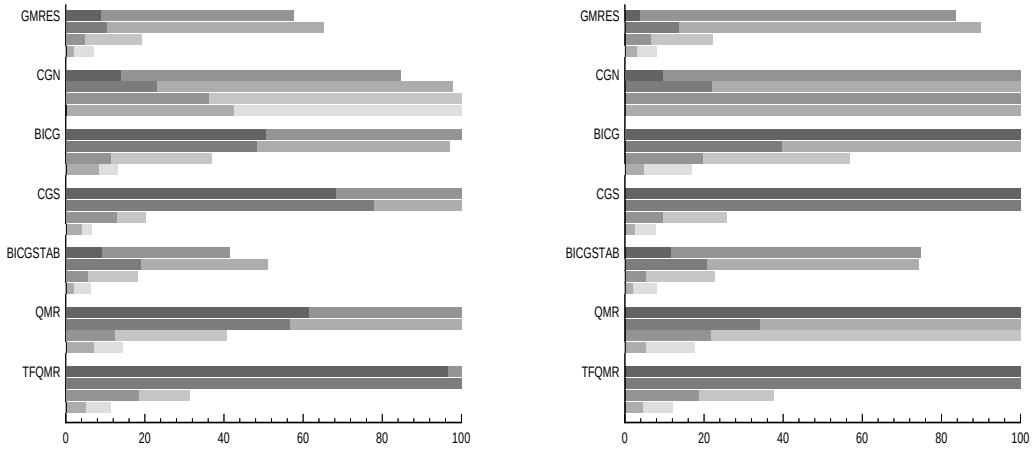
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



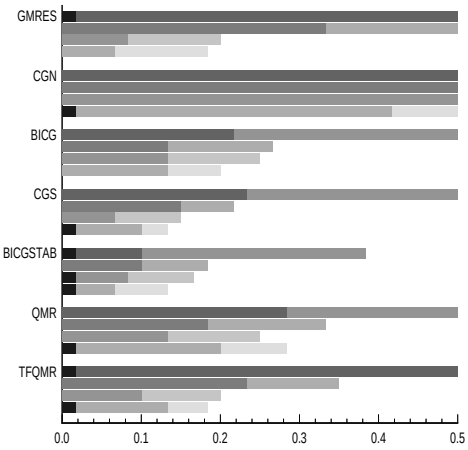
$n = 65 \times 65$

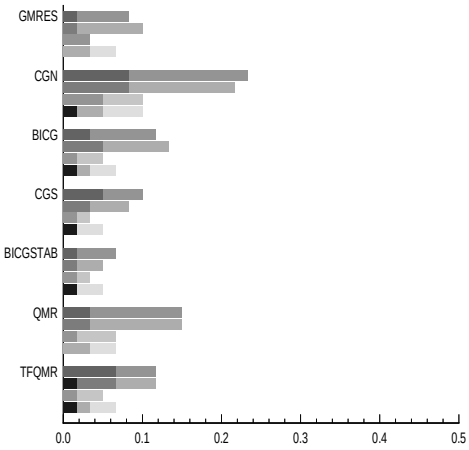


$n = 129 \times 129$

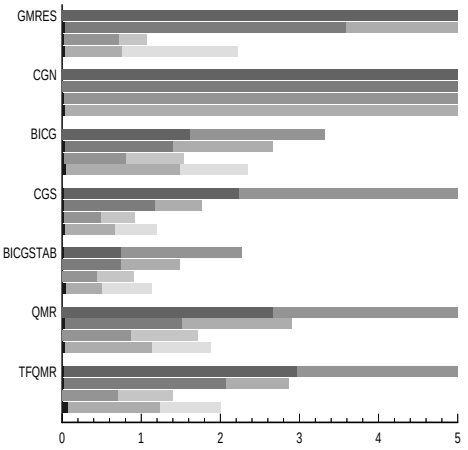
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

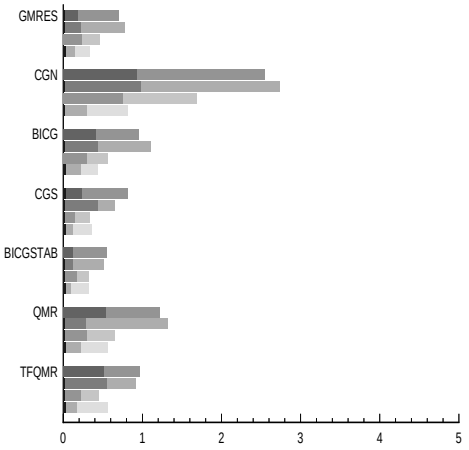
$n = 33 \times 33$



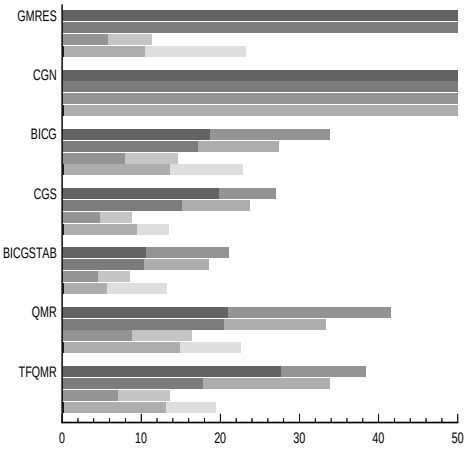


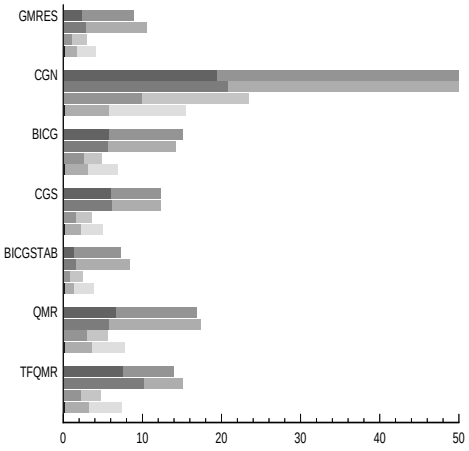
$n = 65 \times 65$



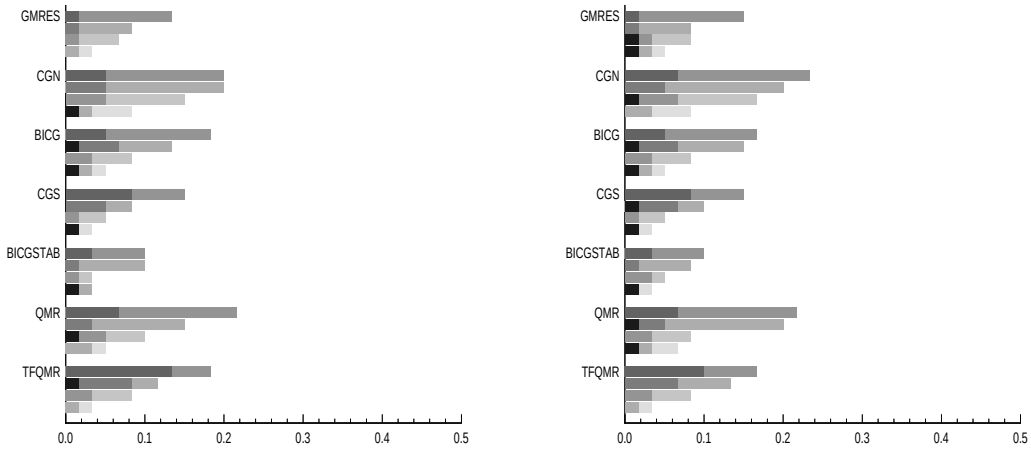


$n = 129 \times 129$

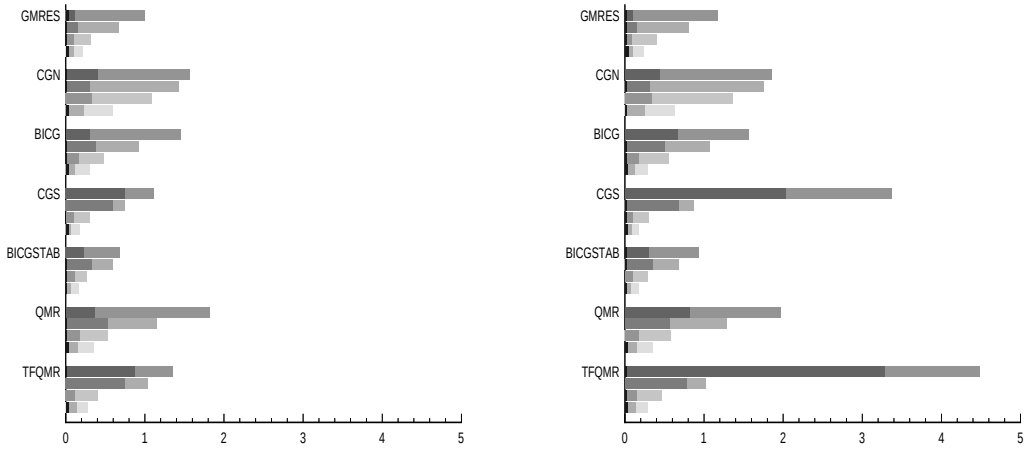




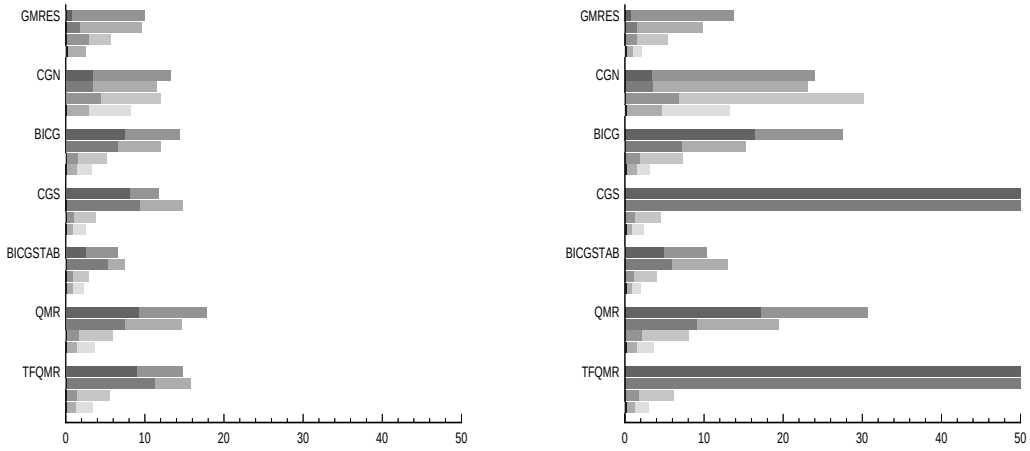
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



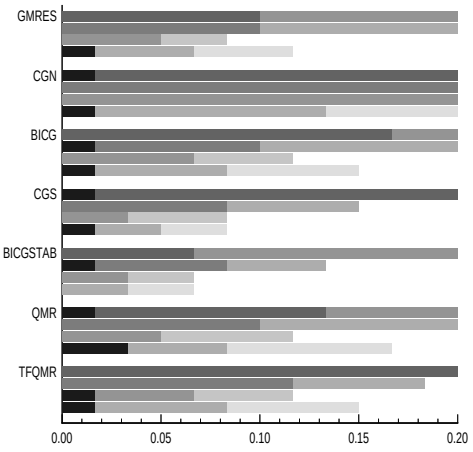
$n = 65 \times 65$

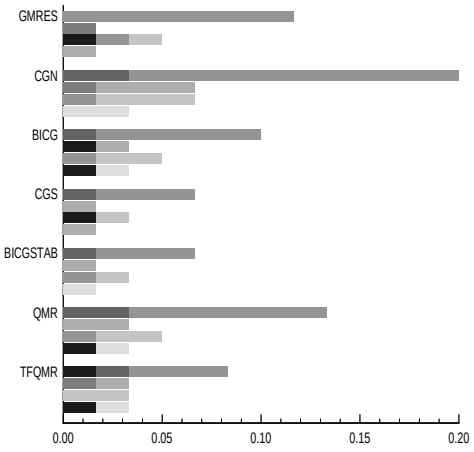


$n = 129 \times 129$

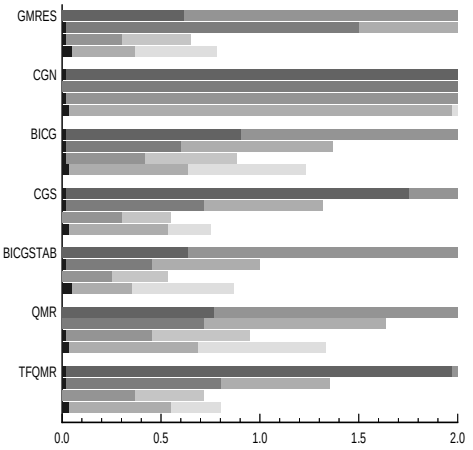
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

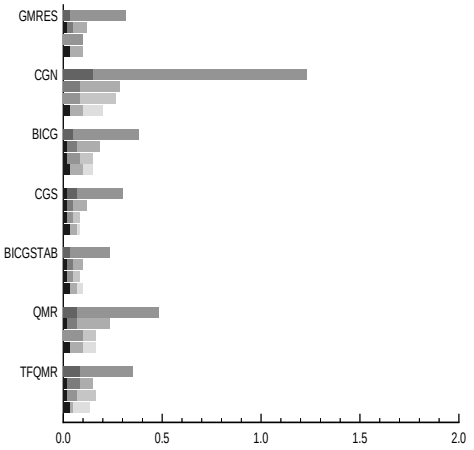
$n = 33 \times 33$



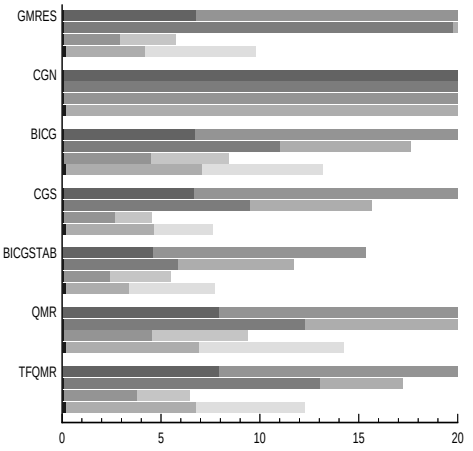


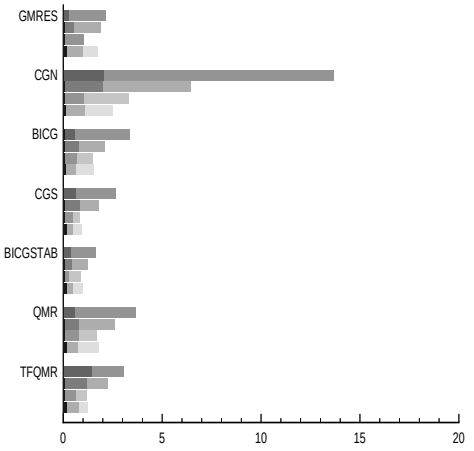
$n = 65 \times 65$



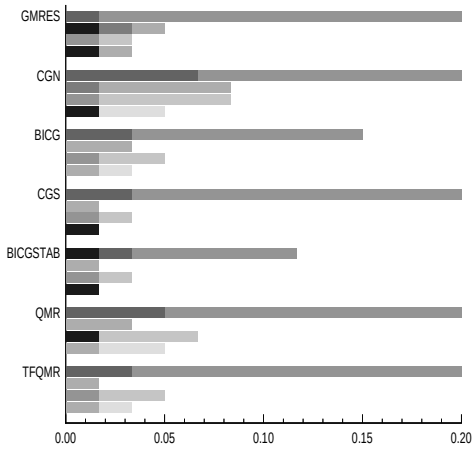


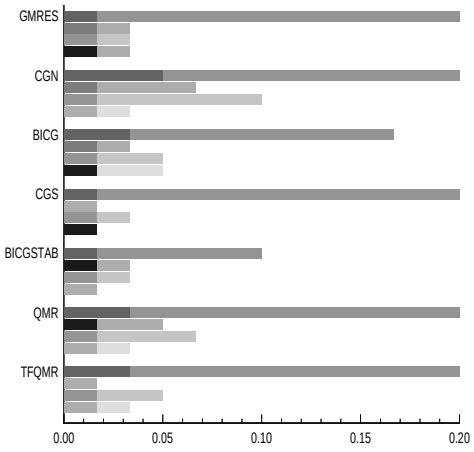
$n = 129 \times 129$



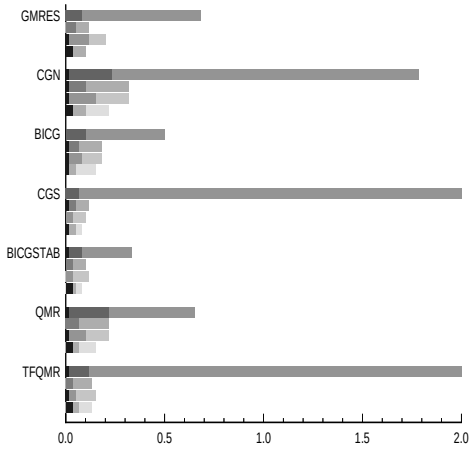


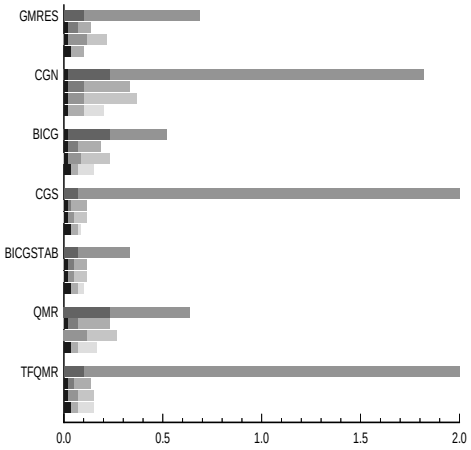
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



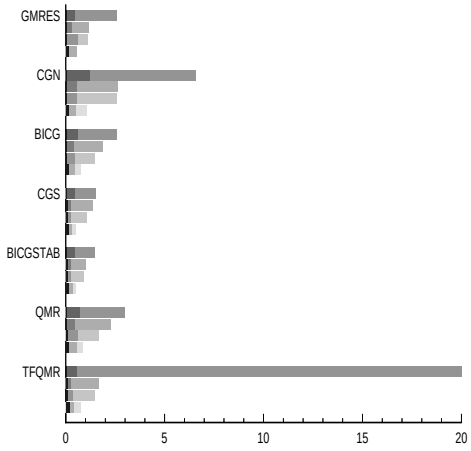


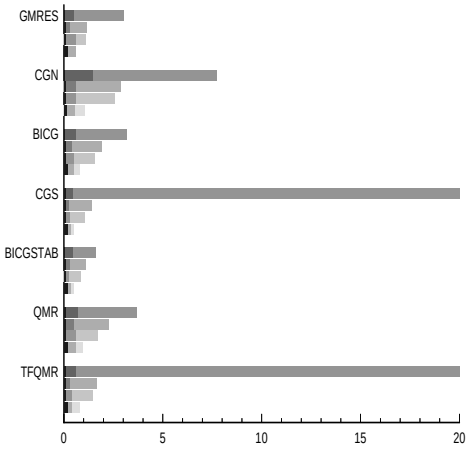
$n = 33 \times 33$





$n = 65 \times 65$





$n = 129 \times 129$

4.5 Fazit

In diesem Kapitel wurden ausführliche Versuchsreihen durchgeführt, um die Eignung von in Kapitel 2 beschriebenen Lösungsverfahren und von in Kapitel 3 beschriebenen Vorkonditionierern zur Lösung diskretisierter Konvektions–Diffusions–Reaktions–Gleichungen zu untersuchen. Die Diskretisierung wurde durch die in Kapitel 1 beschriebene Galerkin–Least–Squares–Methode durchgeführt, die repräsentativen Modellprobleme „Schrägströmung“ und „Rotationsströmung“ wurden in Abschnitt 4.1 definiert. Zuerst wurden die stationären Verfahren SOR und SSOR miteinander verglichen. Wie bei der Beschreibung dieser Verfahren in Abschnitt 2.2 beschrieben wurde, ist die Konvergenz des SOR sehr von der Struktur der Matrix und damit im Finite–Elemente–Kontext von der Numerierungsreihenfolge der Knoten abhängig. Diese Aussage wurde experimentell bestätigt. SSOR dagegen ist, obwohl es auf demselben Algorithmus beruht, wegen der symmetrischen Bearbeitung der Matrix bemerkenswert stabil gegenüber Numerierungsänderungen. Beide Verfahren konvergieren optimal bei Wahl desselben Relaxationsparameters ω, und beide reagieren in gleicher Weise auf nicht optimal gewähltes ω, nämlich durch Konvergenzeinbuße bei Unterschätzung und durch noch stärkere Einbuße oder gar Divergenz bei Überschätzung. Jedoch ist SOR empfindlicher gegenüber nichtoptimalen Relaxationen. Im Zusammenhang mit der Möglichkeit der Verwendung der stationären Verfahren als Vorkonditionierer besitzt SOR einen weiteren Nachteil, da es, anders als SSOR, eventuelle Symmetrie der vorzukonditionierenden Matrix nicht erhält.
Aus diesen Gründen erfolgt die Wahl von SSOR statt SOR als Vorkonditionierer für die im folgenden untersuchten PGK–Verfahren.
Außerdem wurden Vorschläge formuliert, wie sich SSOR optimal relaxieren läßt und damit zu einem konkurrenzfähigen Lösungsverfahren wird.

Die PGK–Verfahren wurden in Gruppen von Verfahren aufgeteilt, die sich in ihren relevanten Eigenschaften gleichen. Die jeweils leistungsfähigsten Vertreter der Gruppen wurden miteinander verglichen. Als Resultat steht die Empfehlung des TFQMR mit guter Vorkonditionierung für den allgemeinen Fall, da diese Kombination relativ schnell ist und keine zu starken Rundungsfehler verursacht. Die Beobachtungen legen dabei nahe, daß durch dieses Verfahren der Fehler etwa um eine Größenordnung schneller als das Residuum konvergiert. Als schnellste, aber gegen Rundungsfehler anfälligere Alternative kann Bi-CGSTAB verwendet werden. Der Fehler der Näherungslösung dieses Verfahren war allerdings bei gleicher Residuennorm um den Faktor 10^3 größer als der des TFQMR .
Diese Eigenschaft teilt mit ihm GMRES(m), das empfohlen wird für den Fall, wenn das Problem sehr schlecht zu lösen ist. Man verwendet es dafür mit möglichst langem Restart m. Dieser Löser verliert allerdings durch zu großes m an Effizienz.

Als Vorkonditionierer wähle man ILU(0), oder im diffusionsdominanten Fall und dann besonders bei feiner Zerlegung SSOR mit einer Abschätzung des optimalen ω.

Die Feinheit der Diskretisierung spielte im Rahmen der Experimente keine große Rolle bei der Frage nach der Lösbarkeit eines Problems. Bei Verdoppelung der Feinheit

und Vierfachung der Dimension der Matrix stieg im allgemeinen die zur Lösung des Problems benötigte Zeit etwa auf das zehnfache an. Auch die Wahl von Interface–Randbedingungen veränderte die Komplexität des Problems kaum. Die Ersetzung von Dirchlet– durch Robin–Bedingungen auf Ein– und Ausströmrand führte nur im diffusionsdominanten Fall zu verlangsamter Konvergenz.
Als besonders schwer und prädestiniert für GMRES(m) mit langem Restart erwies sich der konvektionsdominante und reaktionsfreie Fall der Rotationsströmung bei feiner Zerlegung. Zunehmende Reaktion verbesserte die Lösbarkeit sehr. Im reaktionsdominanten Fall wurde dann umgekehrt das Lösen bei Verschwinden des Diffusionsanteiles einfacher.

Anhang

Hier werden die Konvergenzverläufe aller Löser/Vorkonditionierer–Kombinationen für die beiden Ausgangsprobleme der Schrägströmung mit Dirchlet–Randbedingungen und lexikographischer Numerierung und der Rotationsströmung ohne Reaktionsanteil dargestellt. In den Diagrammen des SOR/SSOR bezeichnen die Buchstaben an den Kurven den jeweiligen Relaxationsfaktor:

$A\hat{=}0.00625$	$G\hat{=}0.4$	$O\hat{=}1.8$
$B\hat{=}0.0125$	$H\hat{=}0.6$	$P\hat{=}1.9$
$C\hat{=}0.025$	$J\hat{=}0.8$	$S\hat{=}1.95$
$D\hat{=}0.05$	$K\hat{=}1.0$	$T\hat{=}1.975$
$E\hat{=}0.1$	$L\hat{=}1.2$	$U\hat{=}1.9875$
$F\hat{=}0.2$	$M\hat{=}1.4$	$W\hat{=}1.99375$
	$N\hat{=}1.6.$	

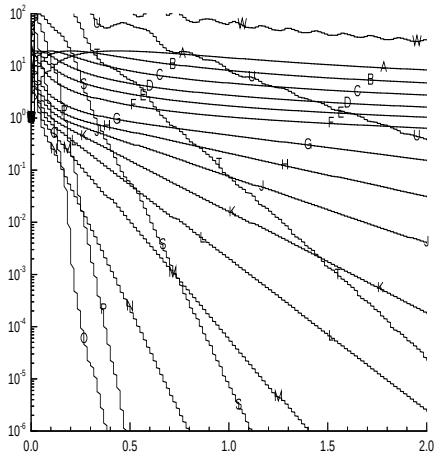
In den übrigen Diagrammen sind die Konvergenzverläufe des auf der jeweiligen Doppelseite behandelten Verfahrens unter Verwendung verschiedener Vorkonditionierungen dargestellt:

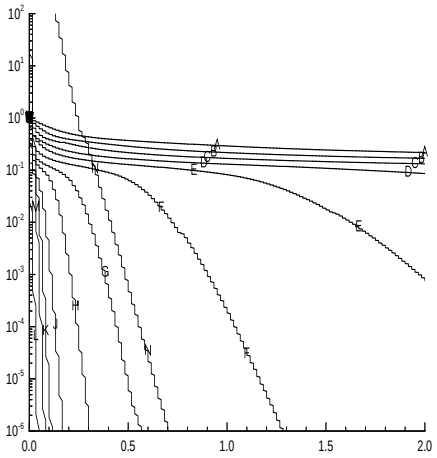
- Keine Vorkonditionierung — *durchgezogene Linie*,
- Jacobi — *Strich–Punkt–Linie*,
- SSOR(ω_{opt}) — *gestrichelte Linie*,
- ILU(0) — *gepunktete Linie*.

Auf der x–Achse ist stets die Rechenzeit in CPU–Sekunden, auf der y–Achse die Konvergenz $\|\tilde{r}\|_2/\|r_0\|_2$ des Residuums $\tilde{r}$ der aktuellen Näherungslösung aufgetragen.

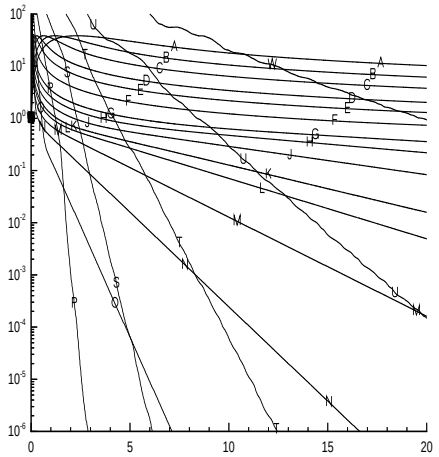
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

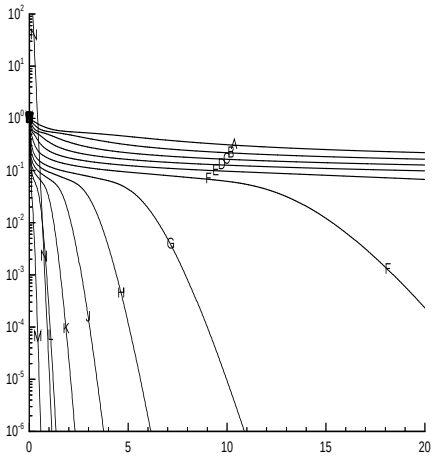
$n = 33 \times 33$



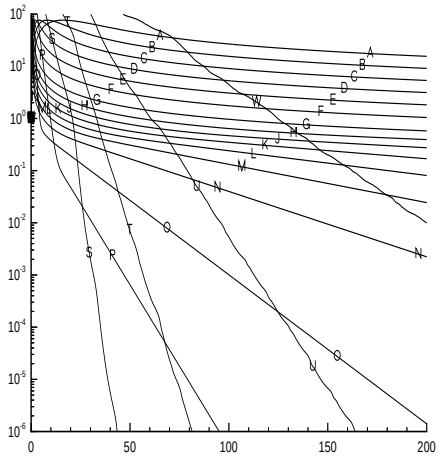


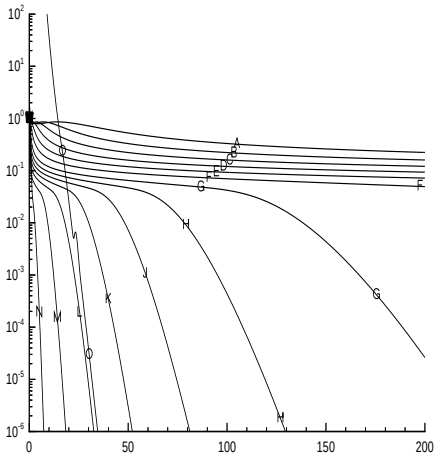
$n = 65 \times 65$



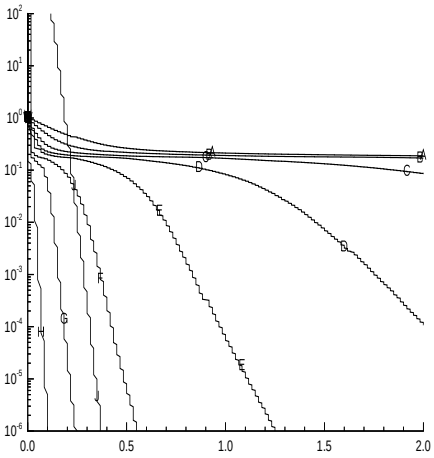
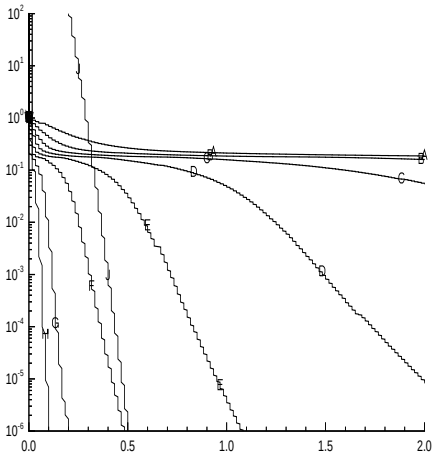


$n = 129 \times 129$

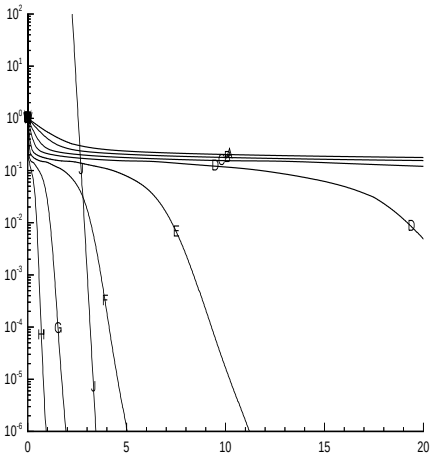
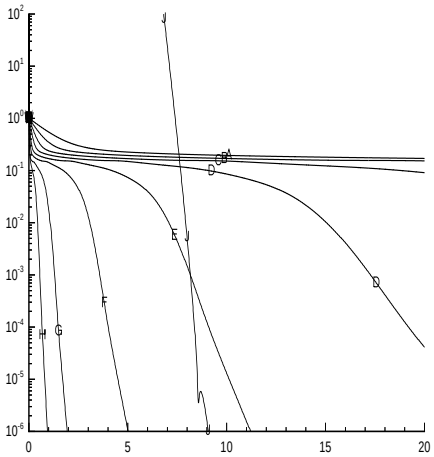




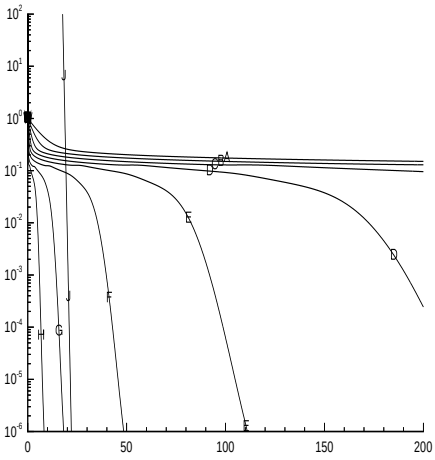
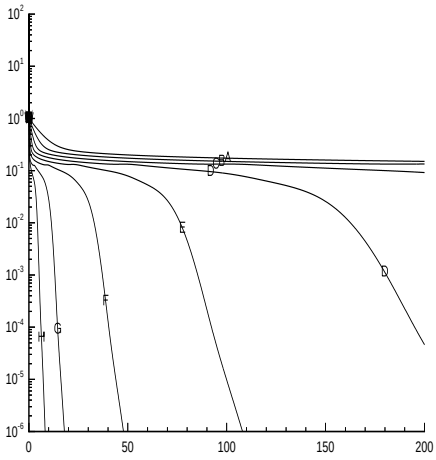
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

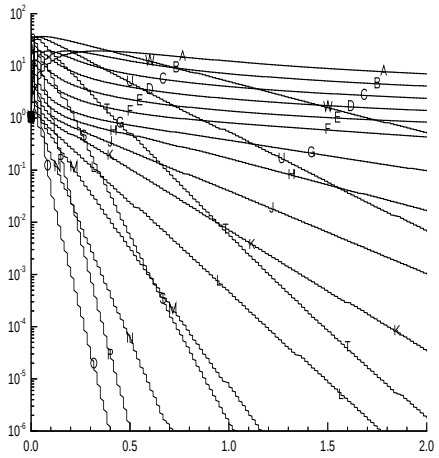
$n = 65 \times 65$

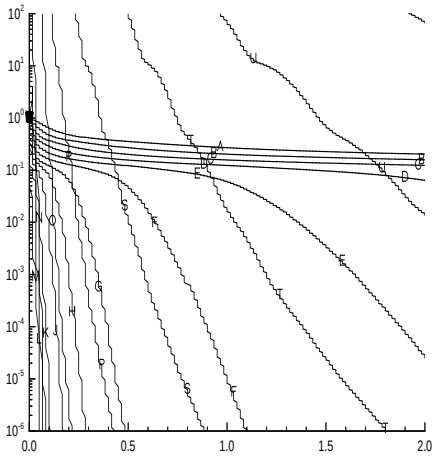



$n = 129 \times 129$

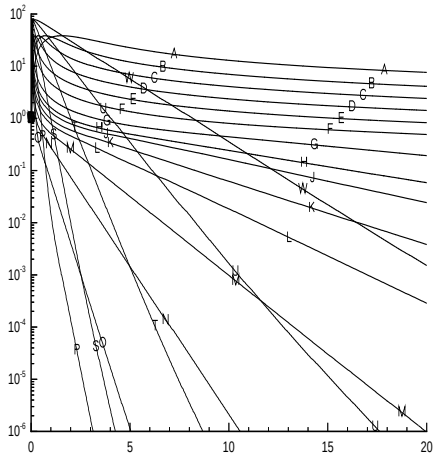
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

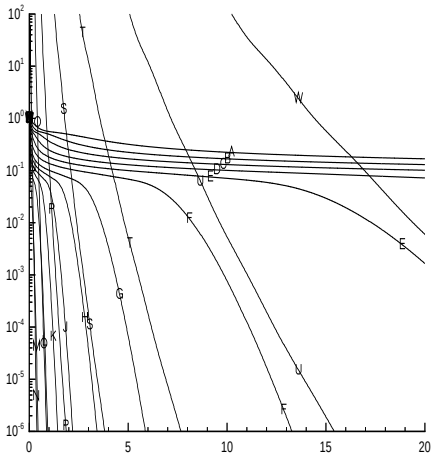
$n = 33 \times 33$



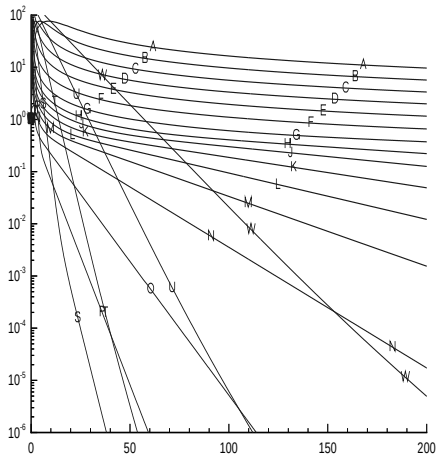


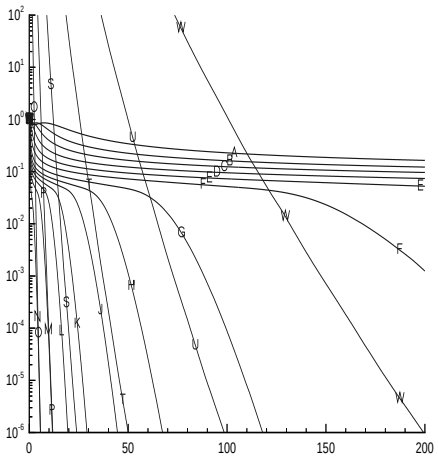
$n = 65 \times 65$



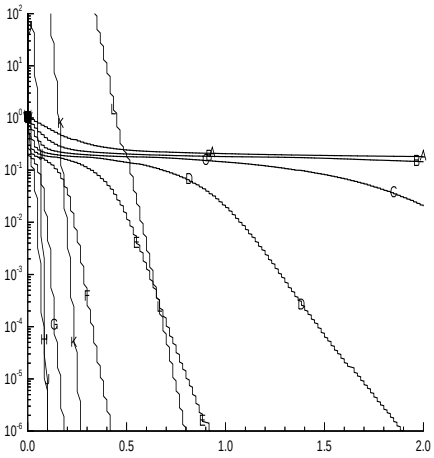
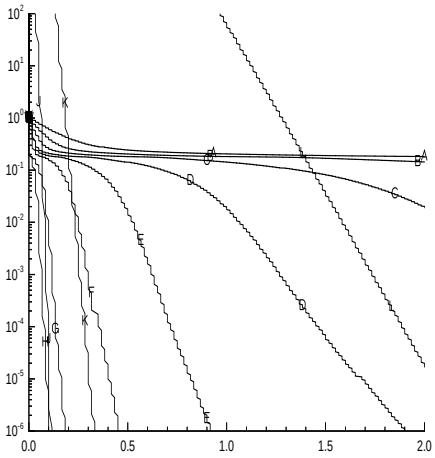


$n = 129 \times 129$

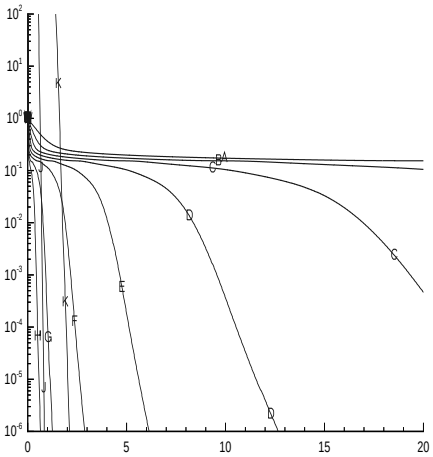
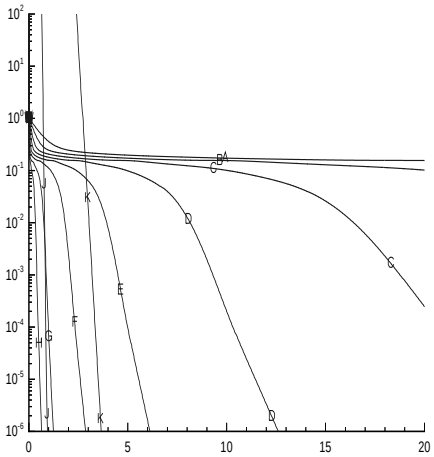




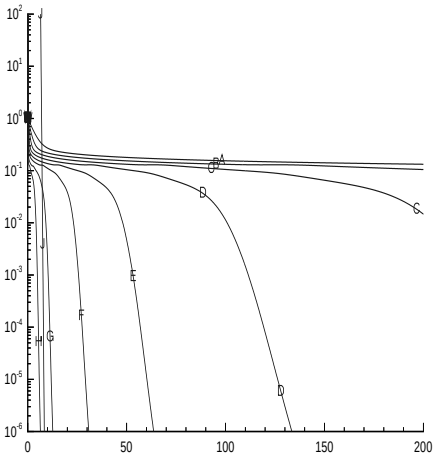
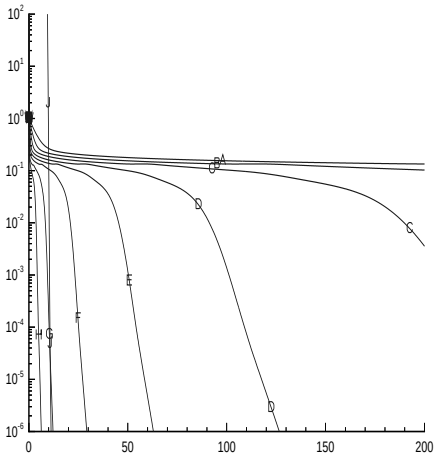
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

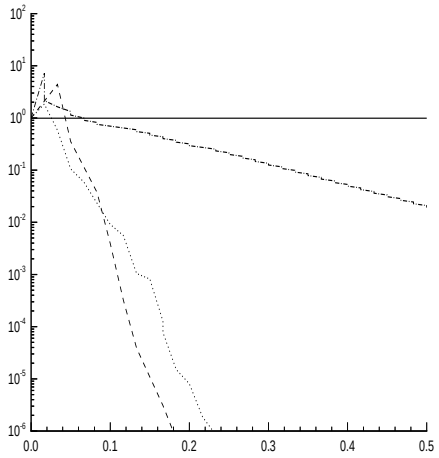
$n = 65 \times 65$

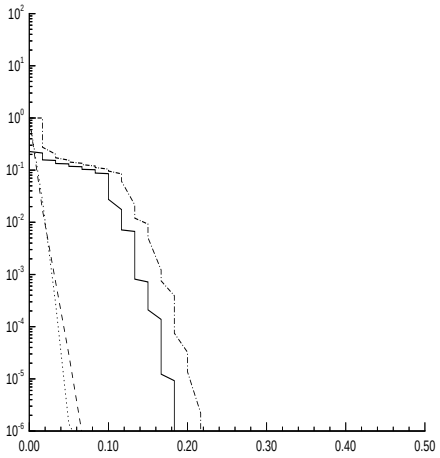



$n = 129 \times 129$

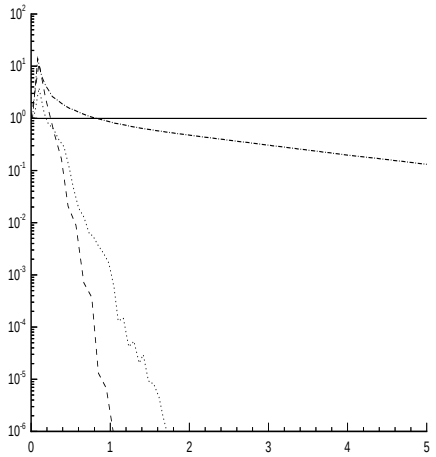
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

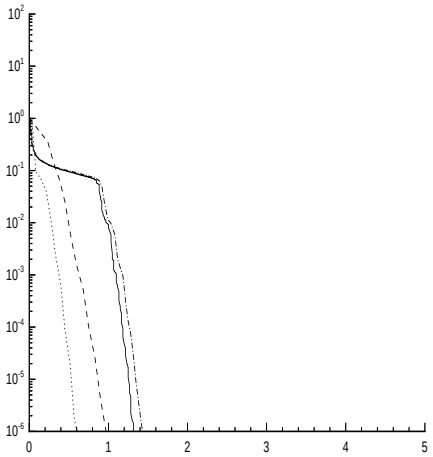
$n = 33 \times 33$



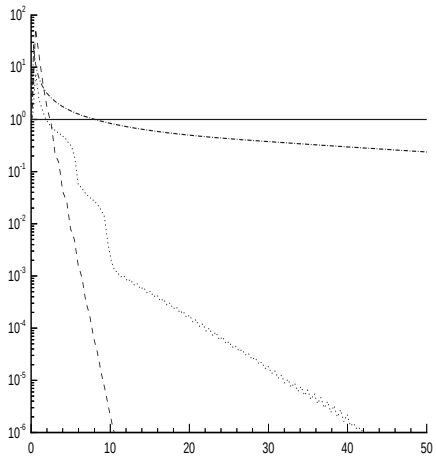


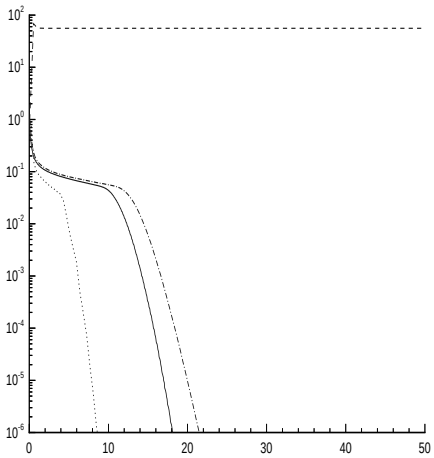
$n = 65 \times 65$



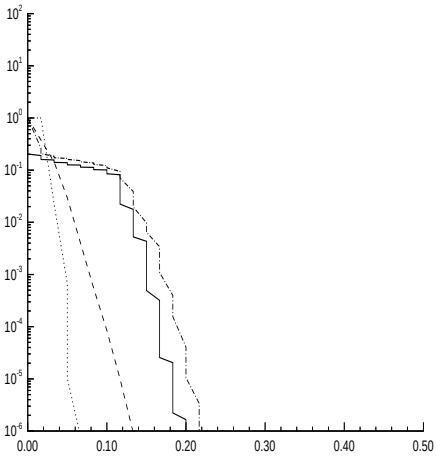
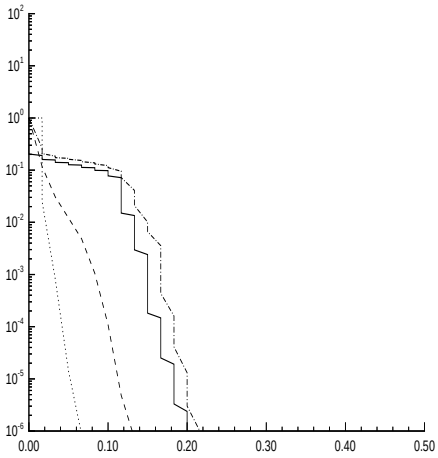


$n = 129 \times 129$

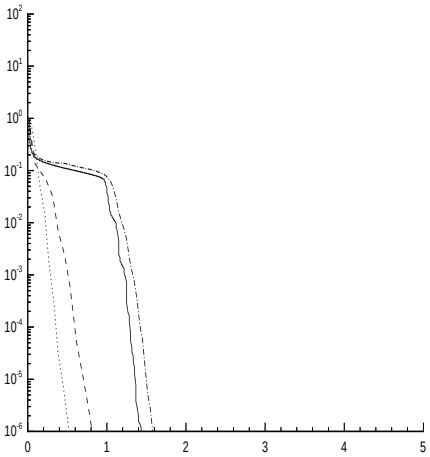
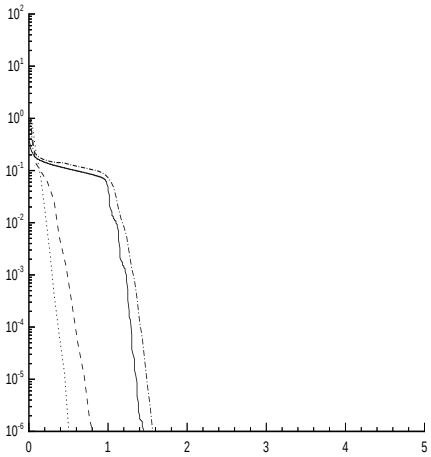




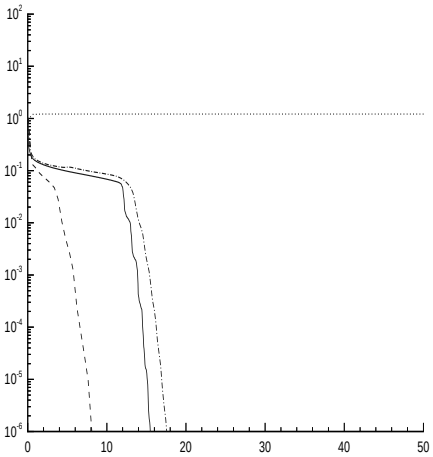
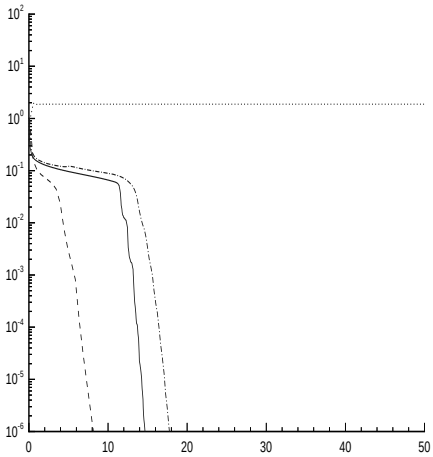
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

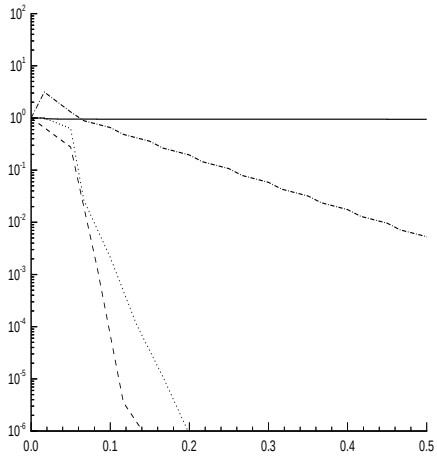
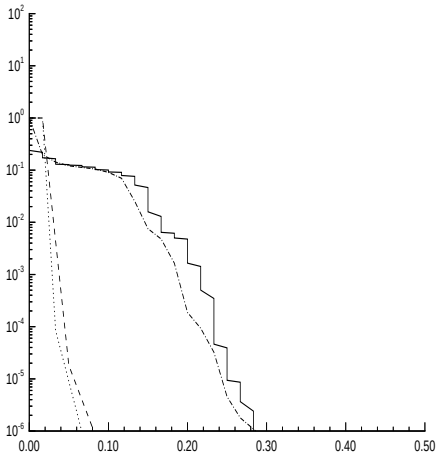
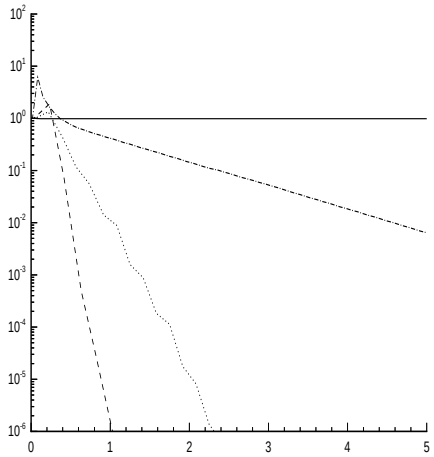
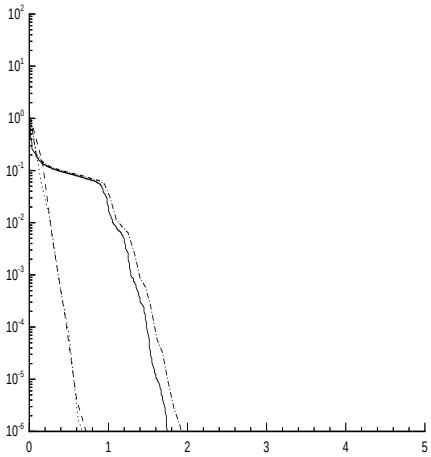
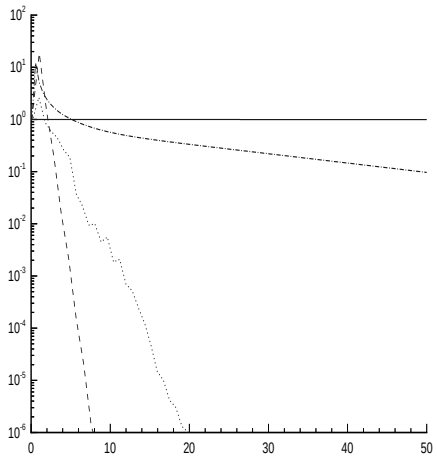
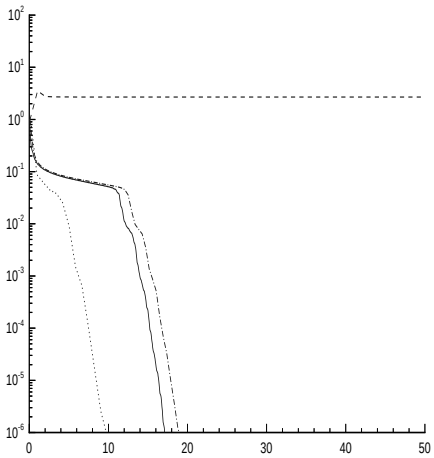
$n = 33 \times 33$

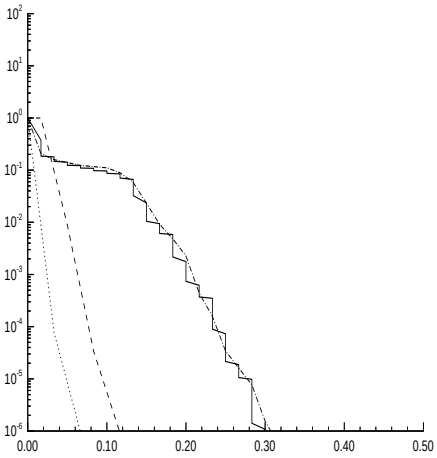
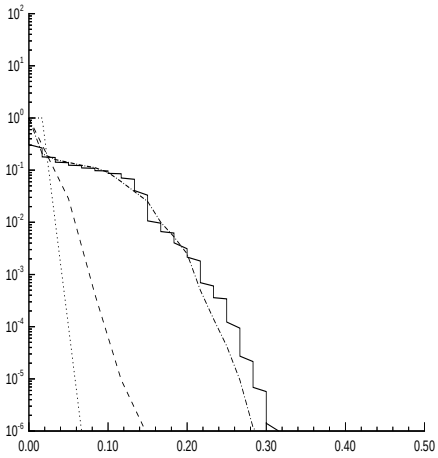
$n = 65 \times 65$

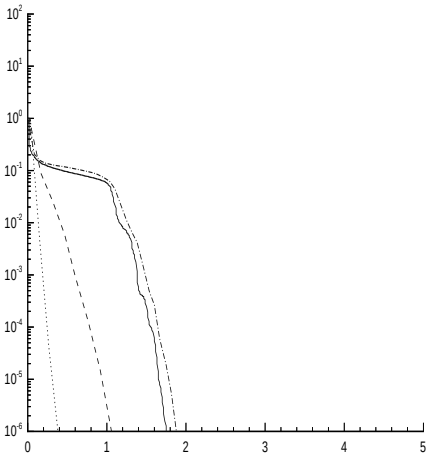
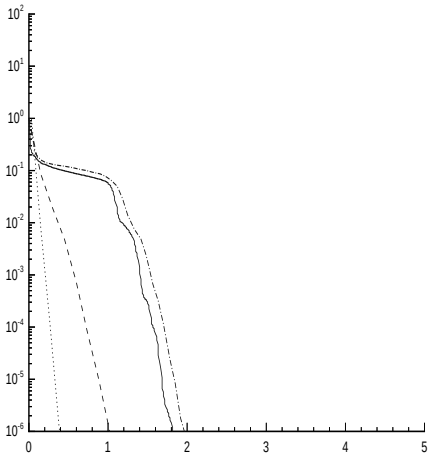
$n = 129 \times 129$

	$\varepsilon = 10^0$	$\varepsilon = 10^{-2}$
$n = 33 \times 33$		
$n = 65 \times 65$		
$n = 129 \times 129$		

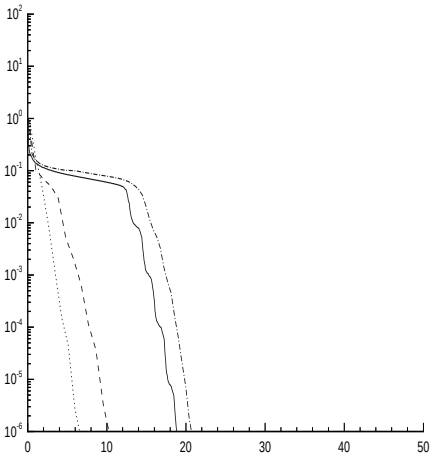
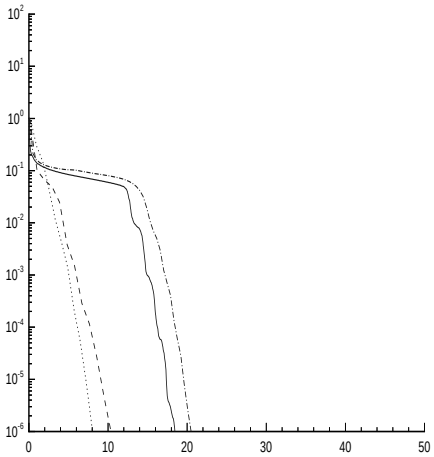
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

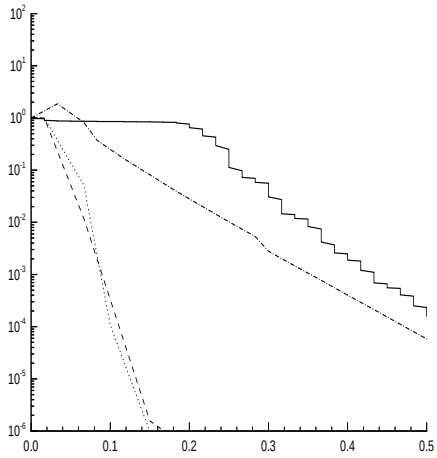
$n = 65 \times 65$

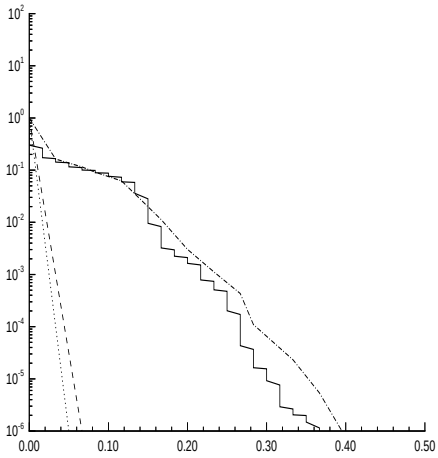



$n = 129 \times 129$

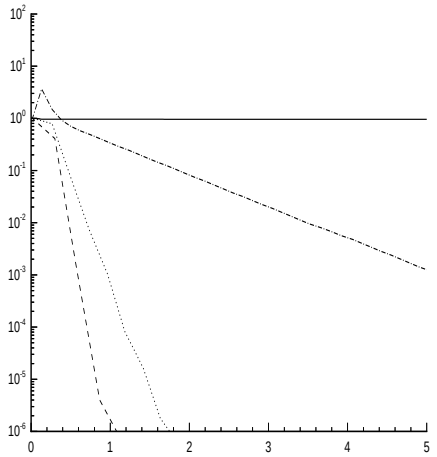
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

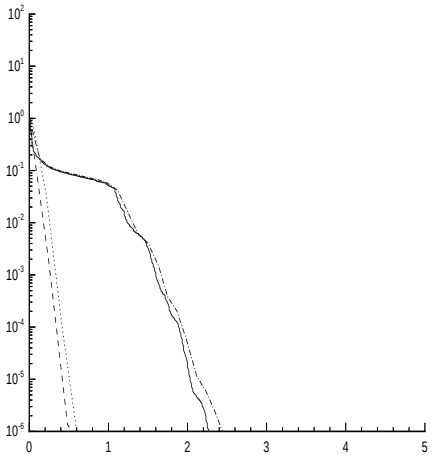
$n = 33 \times 33$



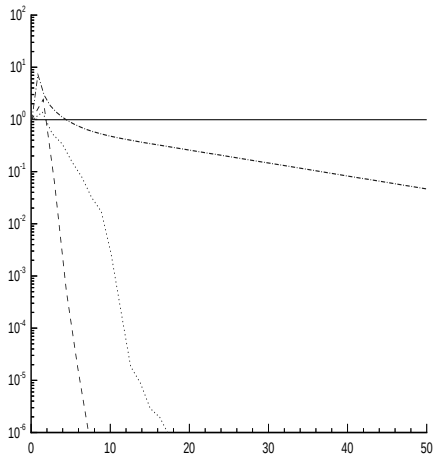


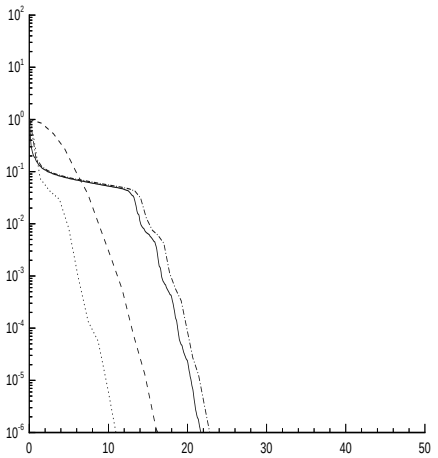
$n = 65 \times 65$



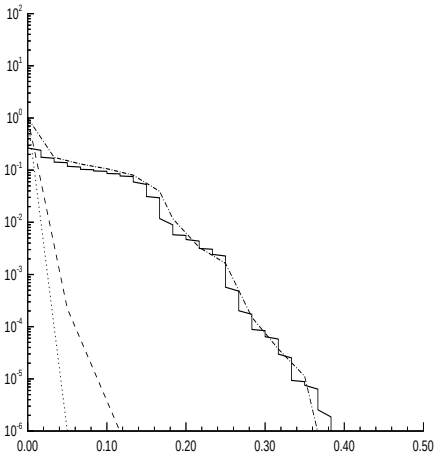
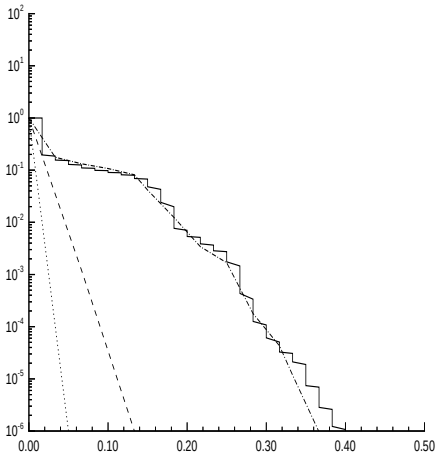


$n = 129 \times 129$

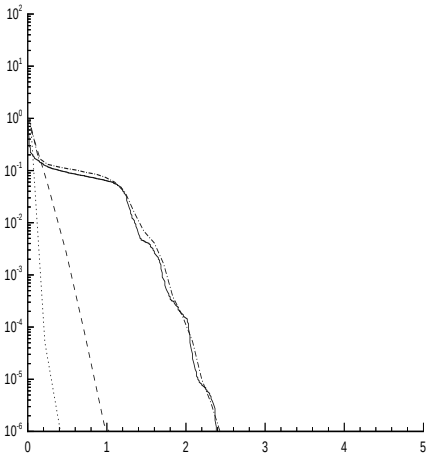
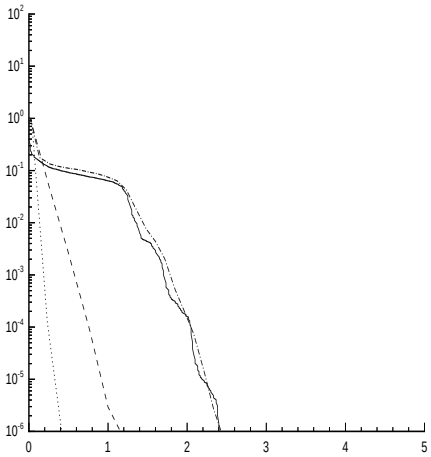




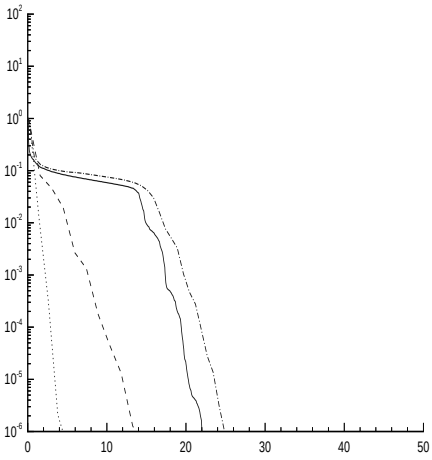
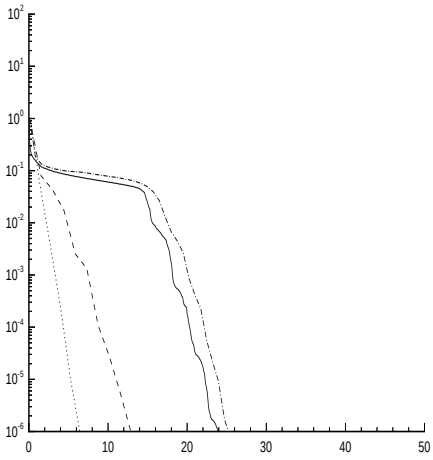
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

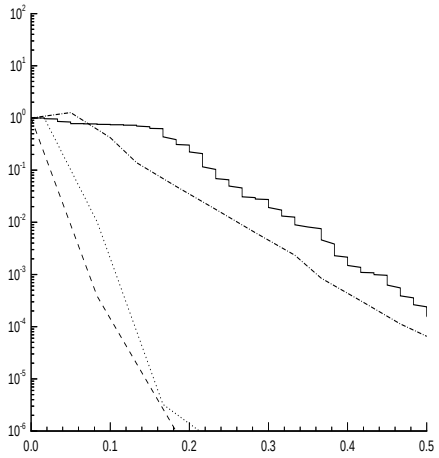
$n = 65 \times 65$

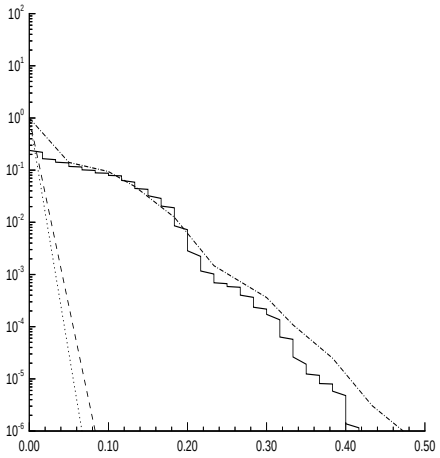



$n = 129 \times 129$

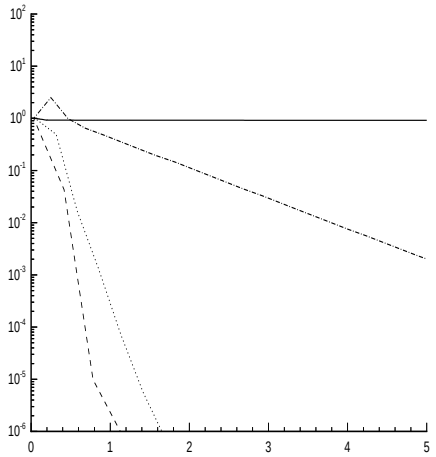
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

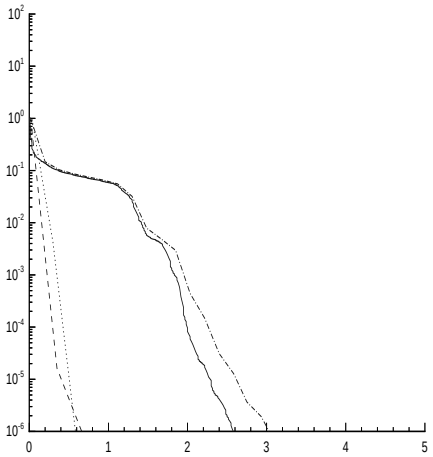
$n = 33 \times 33$



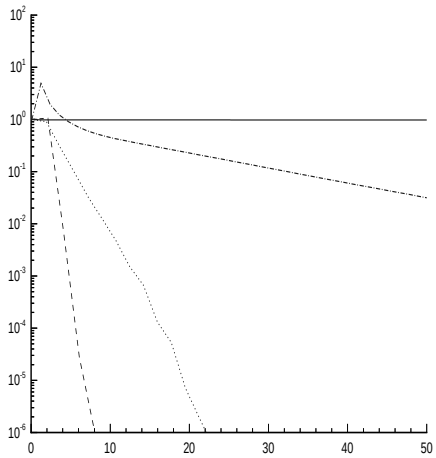


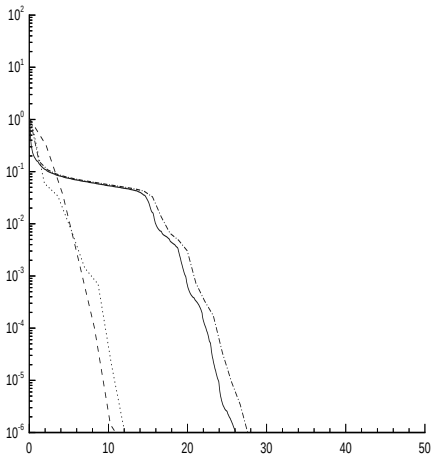
$n = 65 \times 65$



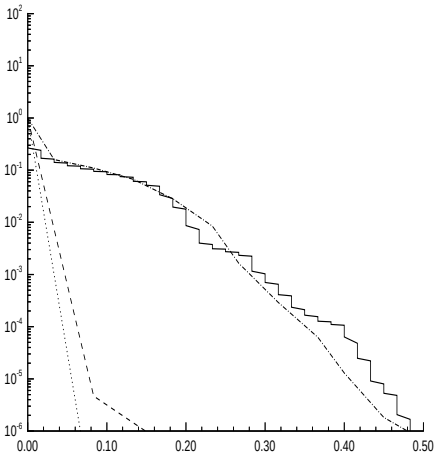
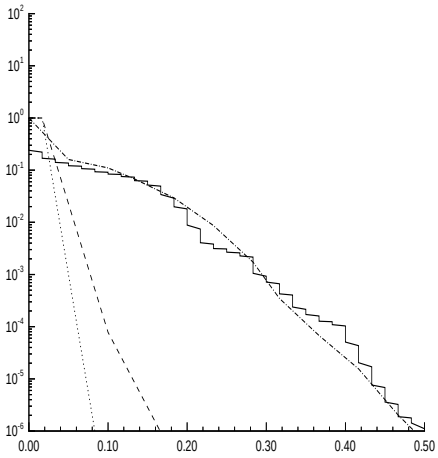


$n = 129 \times 129$

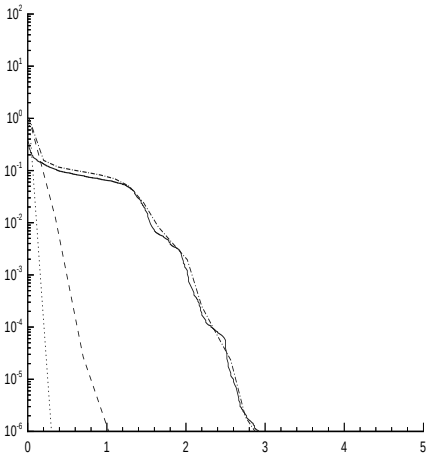
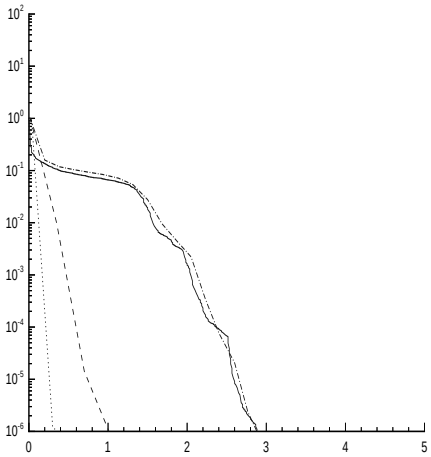




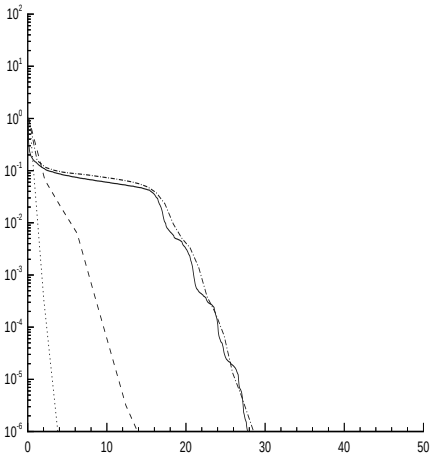
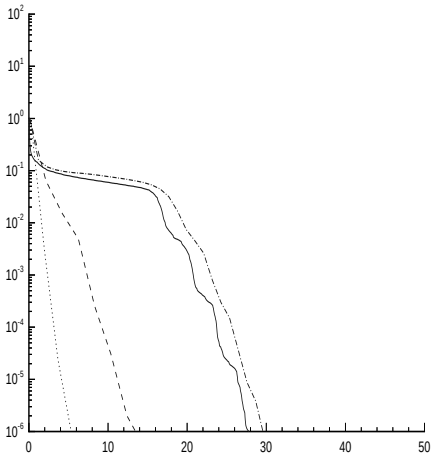
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

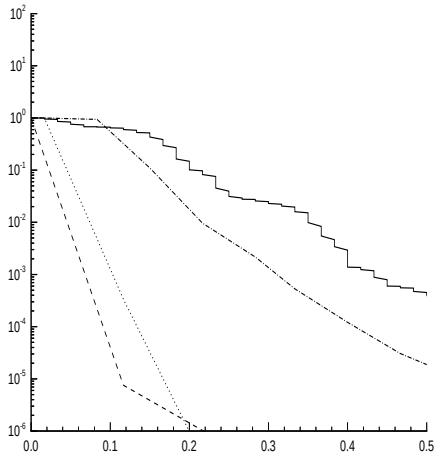
$n = 65 \times 65$

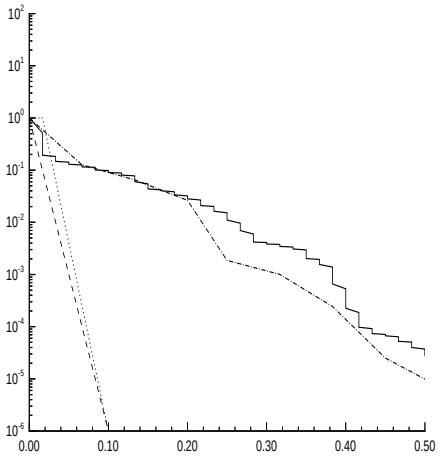



$n = 129 \times 129$

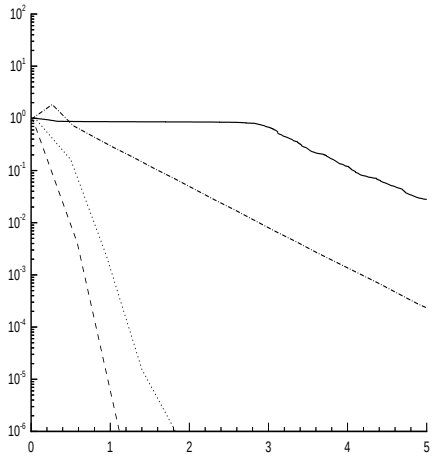
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

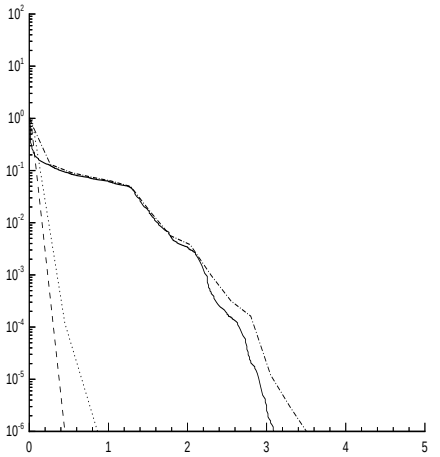
$n = 33 \times 33$



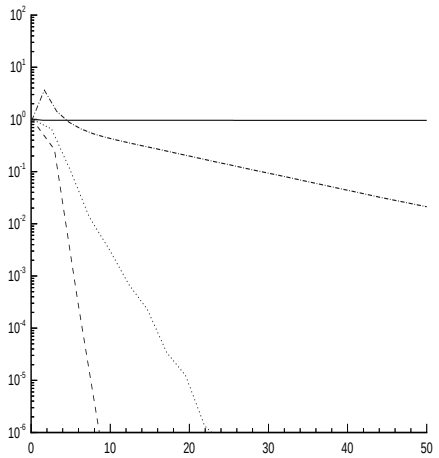


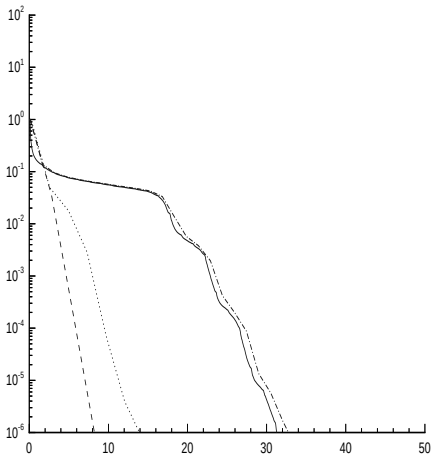
$n = 65 \times 65$



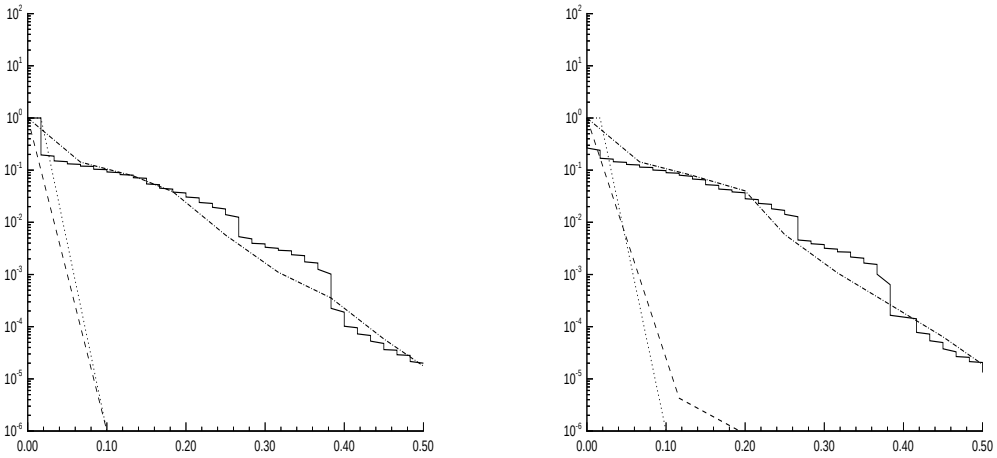


$n = 129 \times 129$

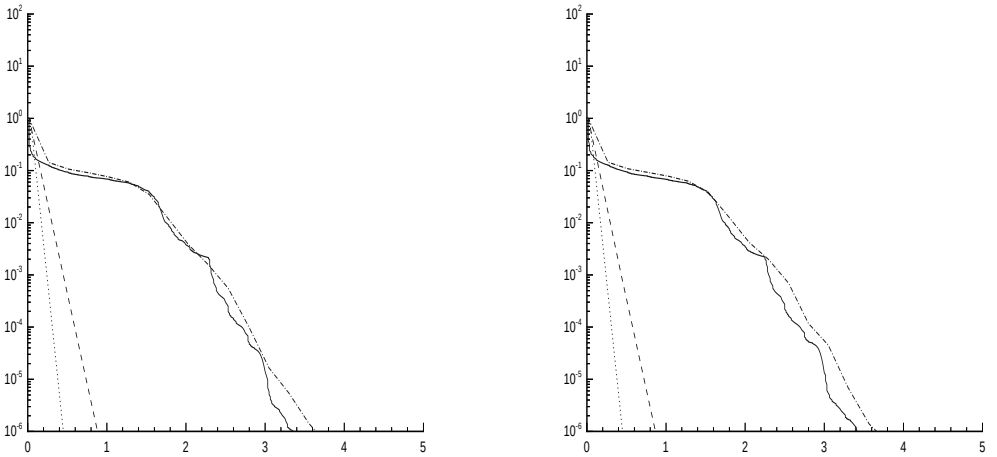




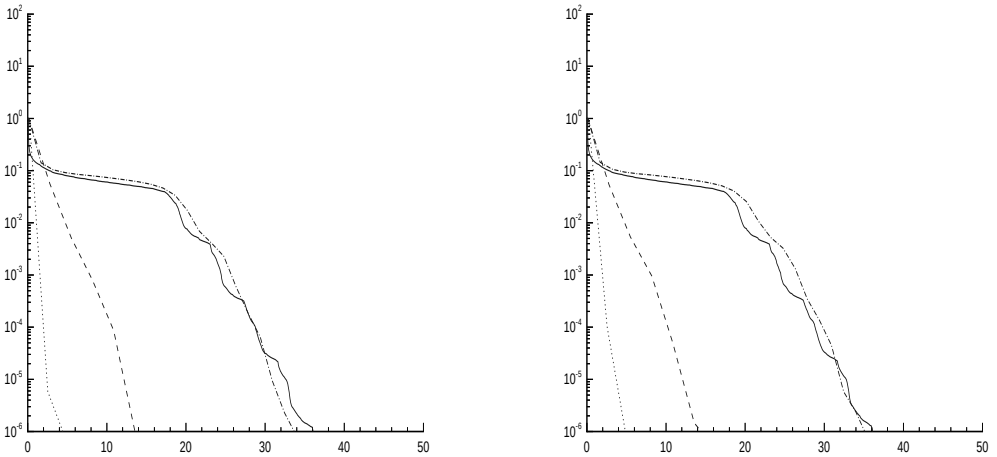
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



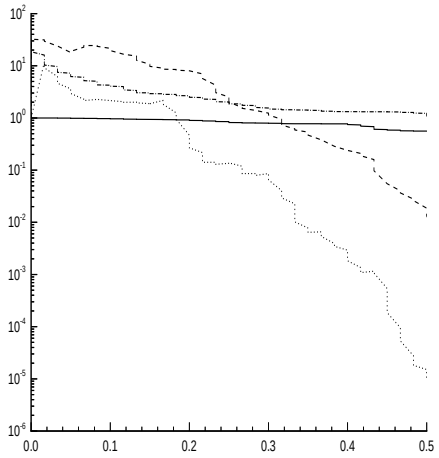
$n = 65 \times 65$

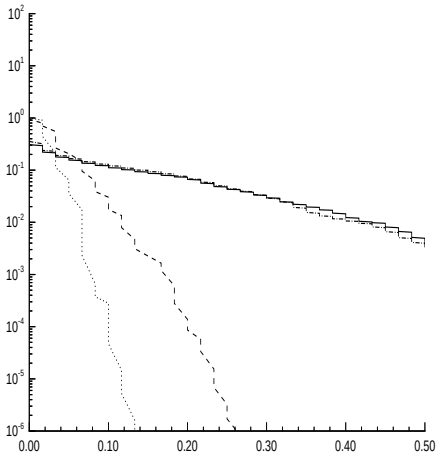


$n = 129 \times 129$

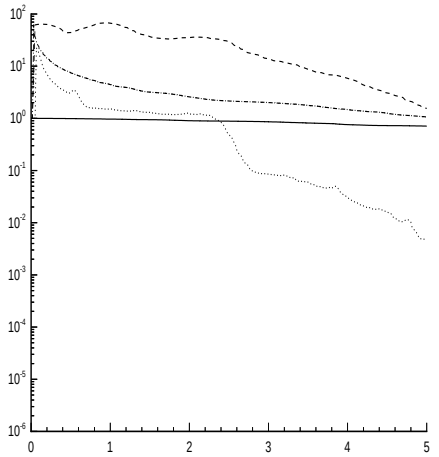
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

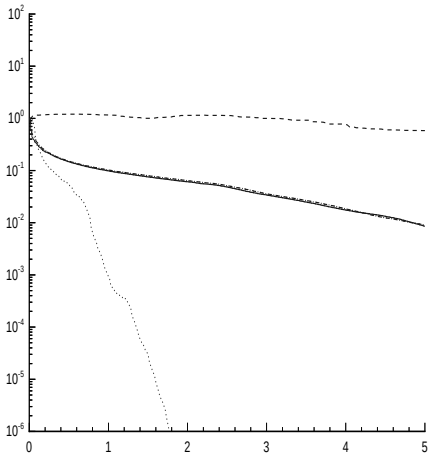
$n = 33 \times 33$



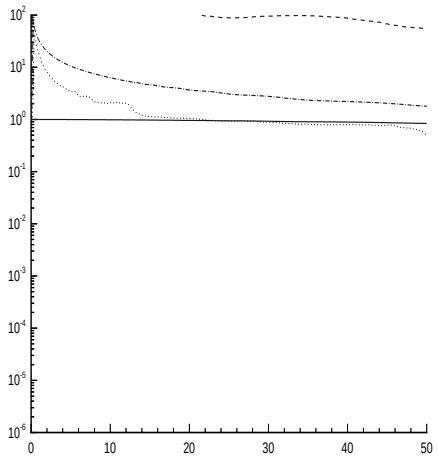


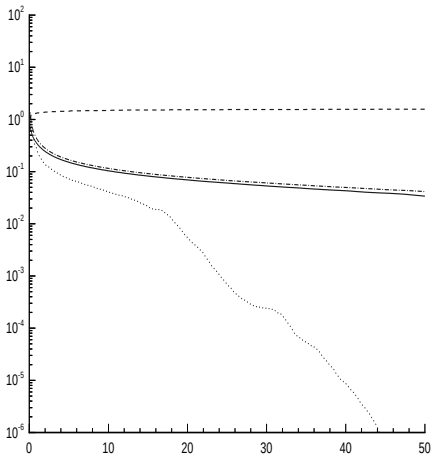
$n = 65 \times 65$



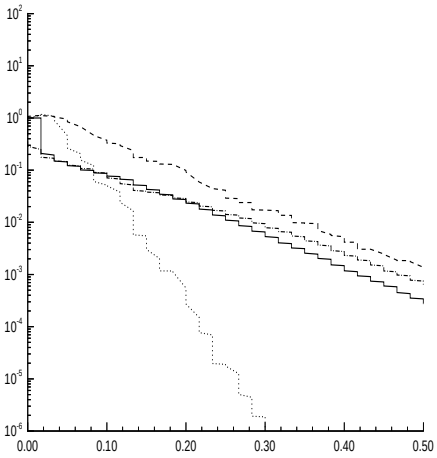
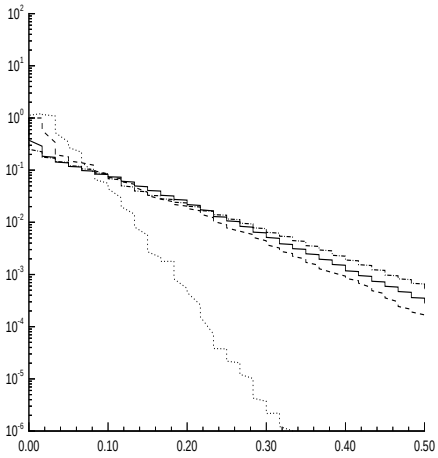


$n = 129 \times 129$

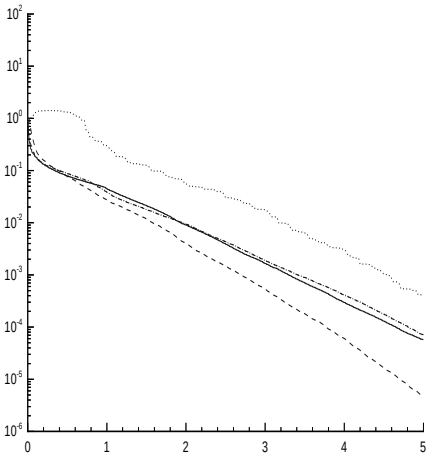
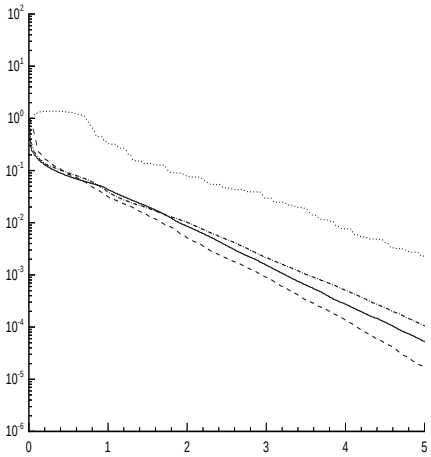




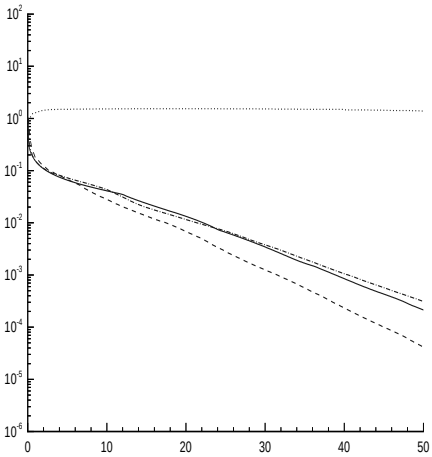
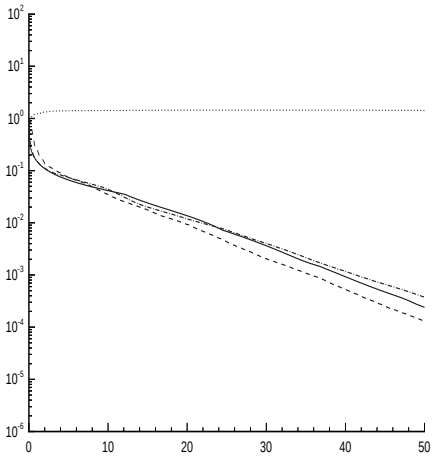
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

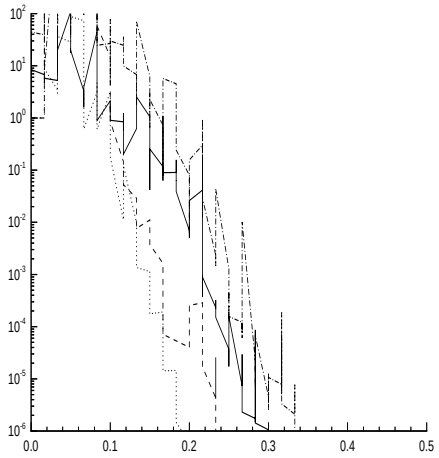
$n = 65 \times 65$

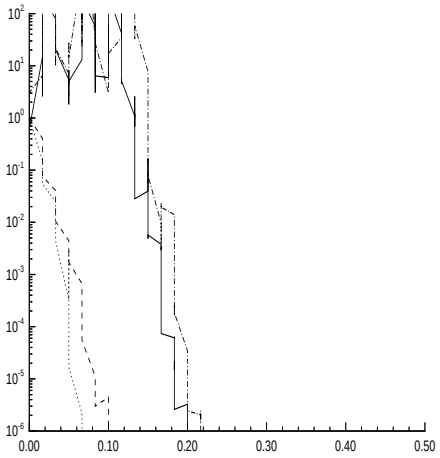



$n = 129 \times 129$

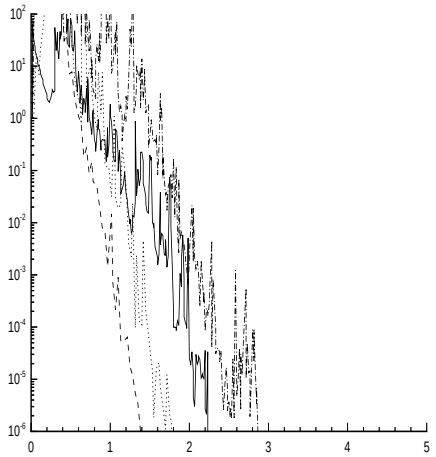
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

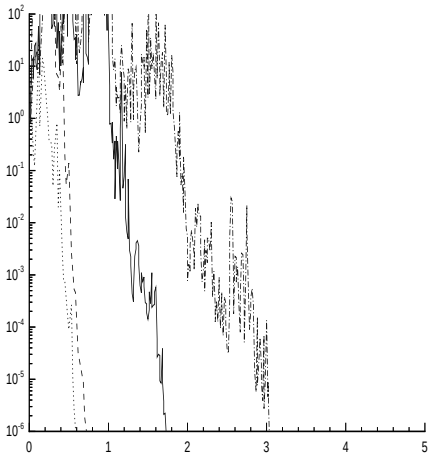
$n = 33 \times 33$



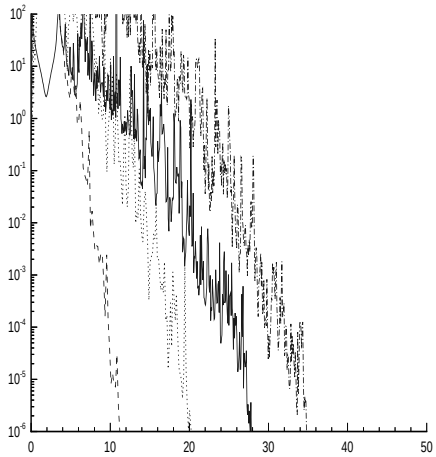


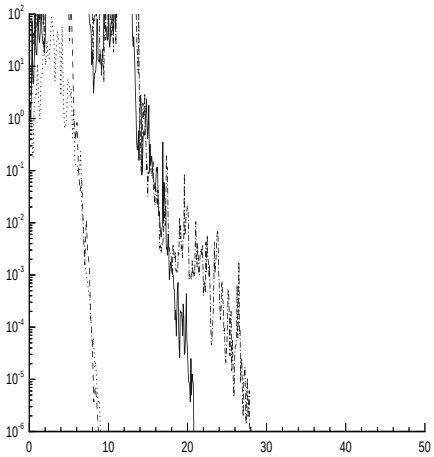
$n = 65 \times 65$



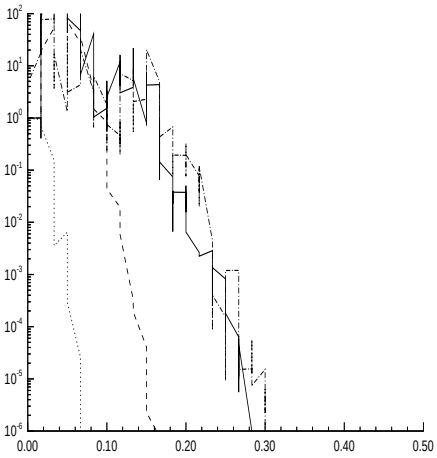
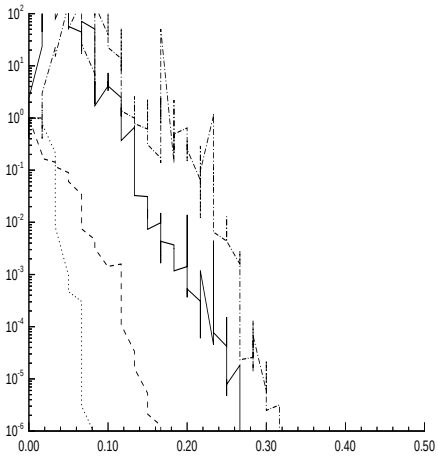


$n = 129 \times 129$

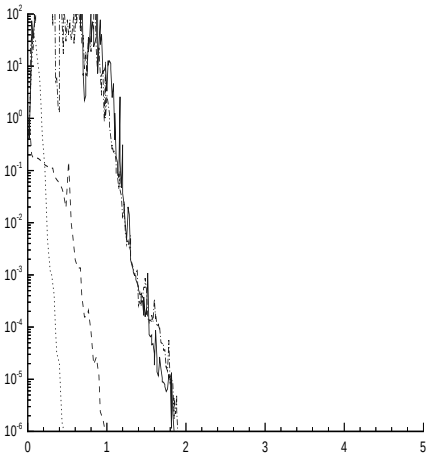
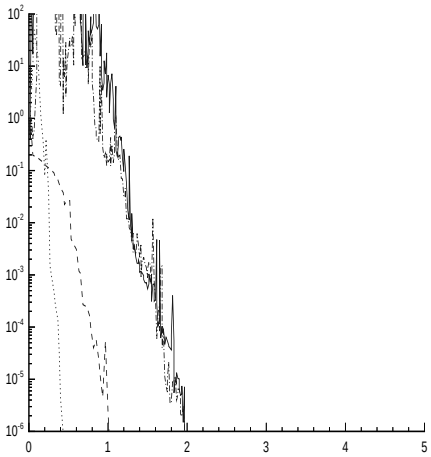




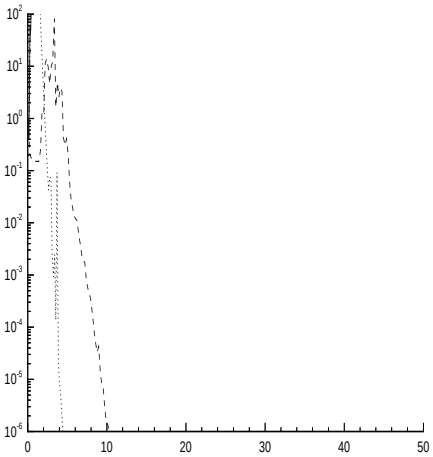
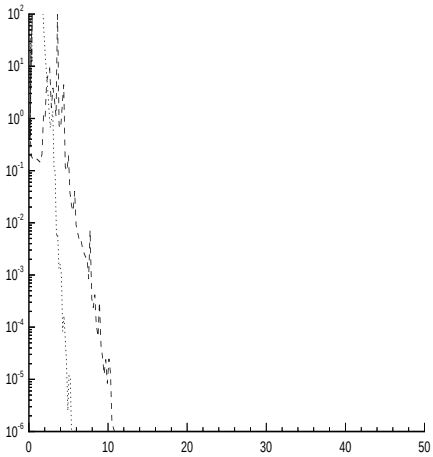
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

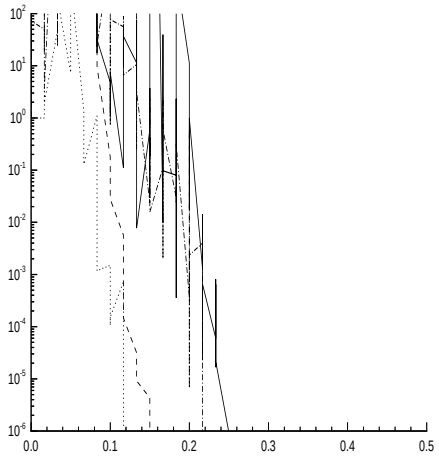
$n = 65 \times 65$

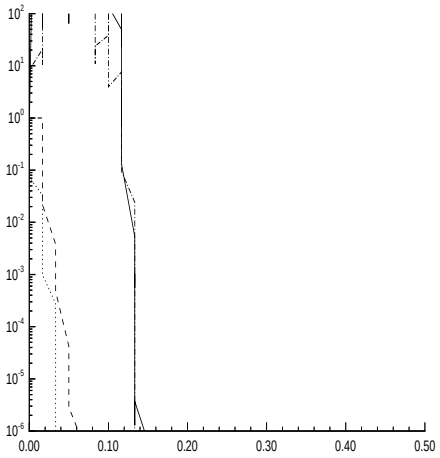



$n = 129 \times 129$

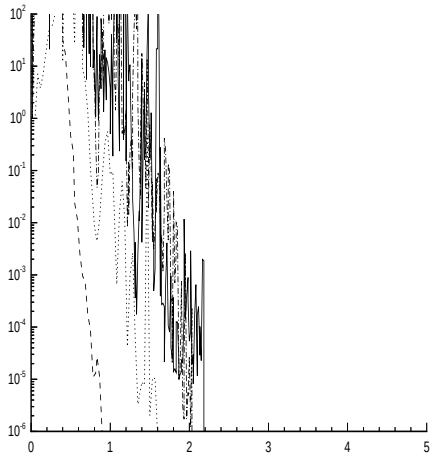
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

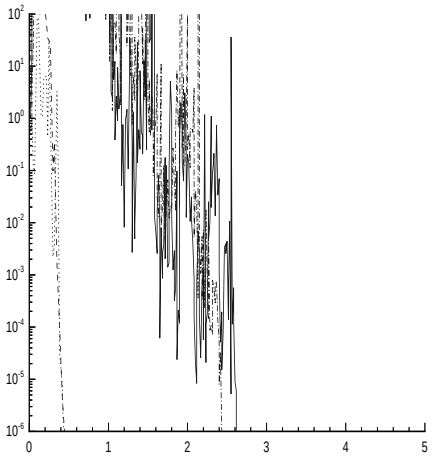
$n = 33 \times 33$



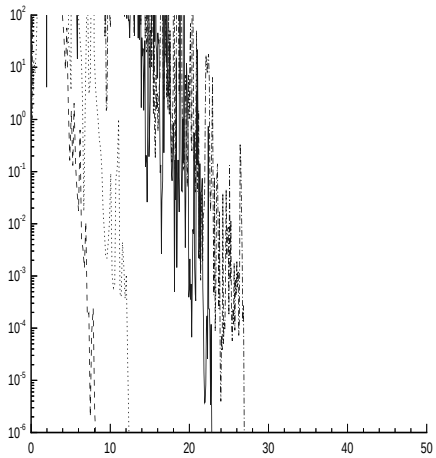


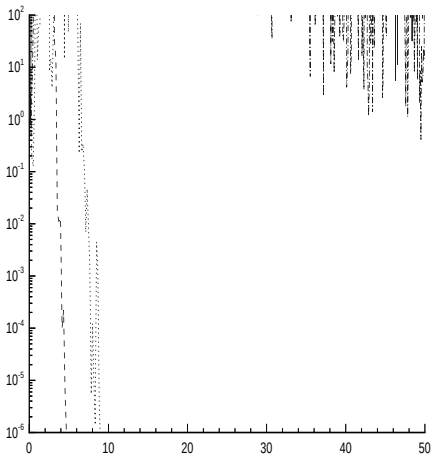
$n = 65 \times 65$



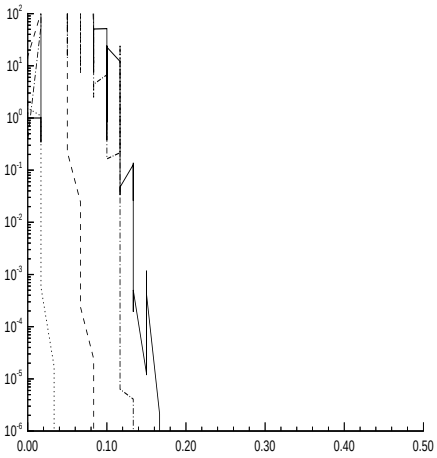
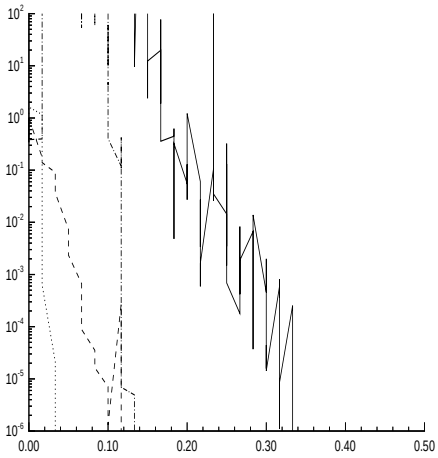


$n = 129 \times 129$

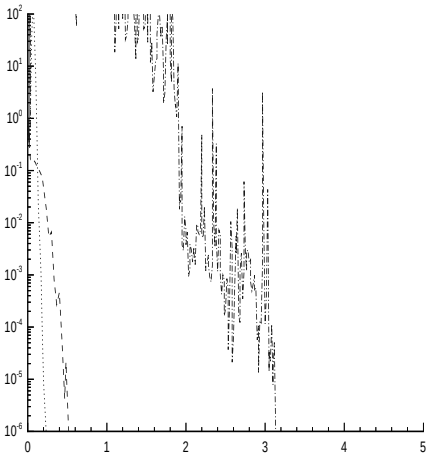
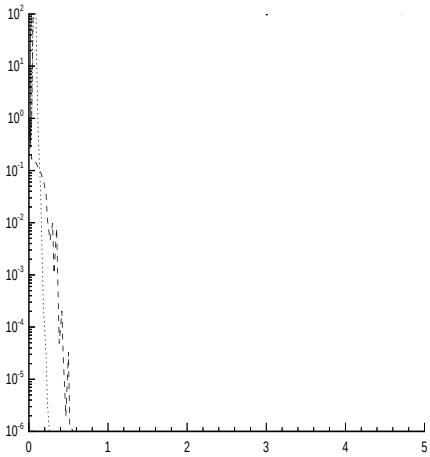




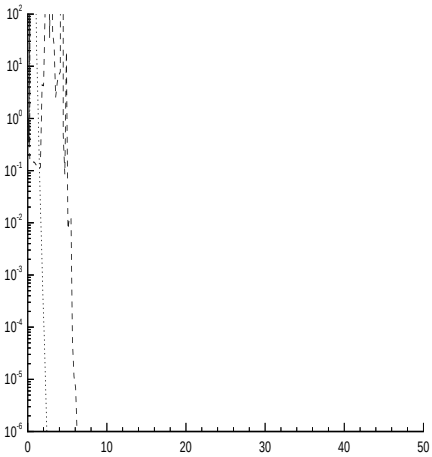
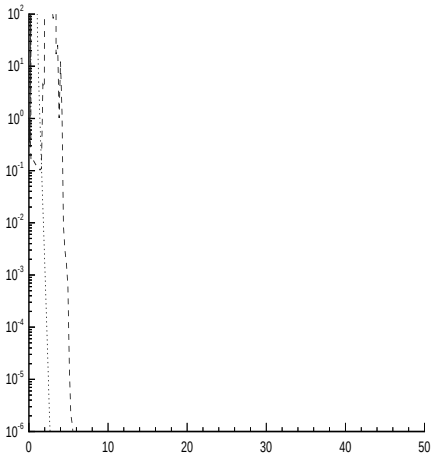
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

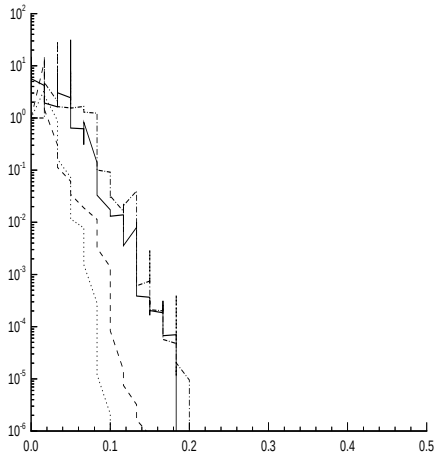
$n = 65 \times 65$

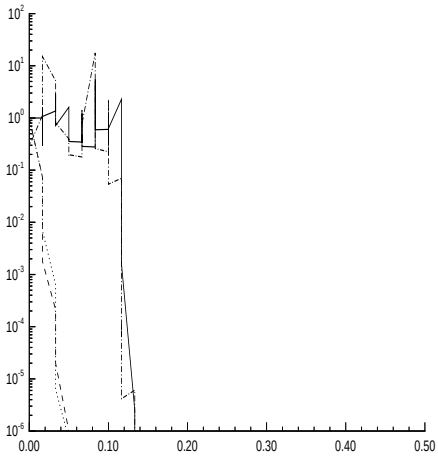



$n = 129 \times 129$

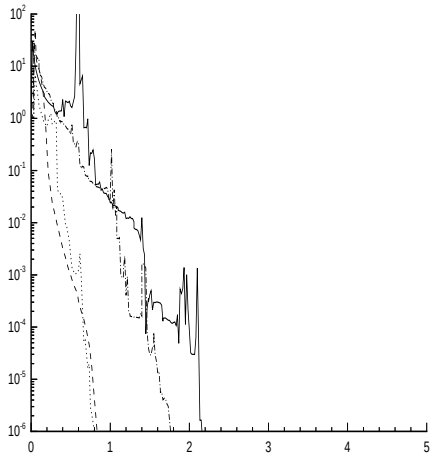
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

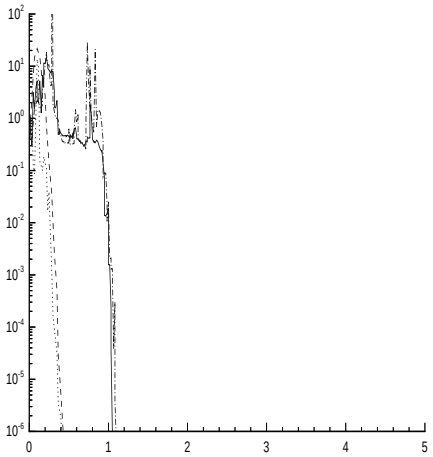
$n = 33 \times 33$



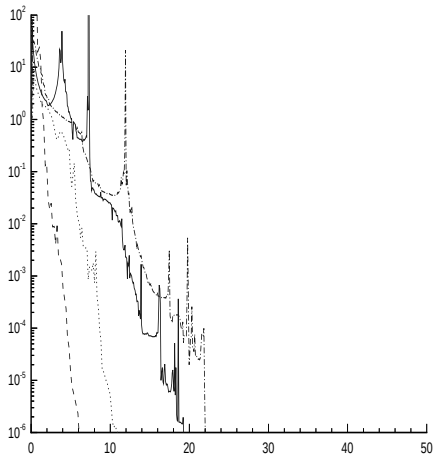


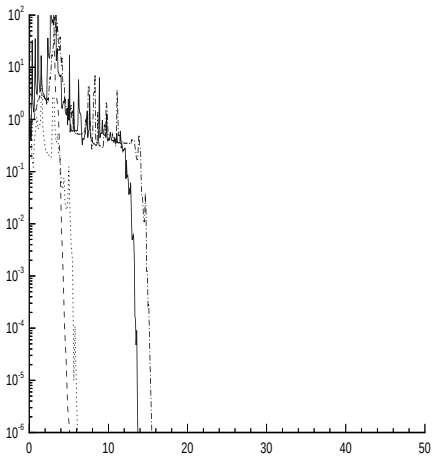
$n = 65 \times 65$



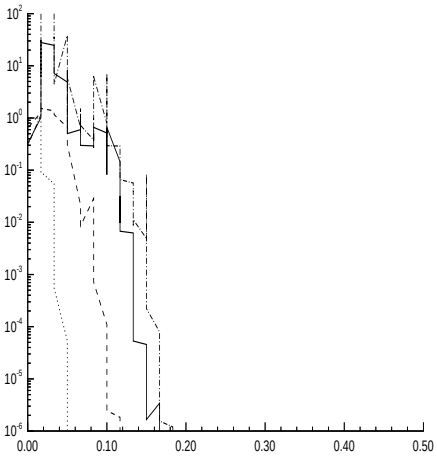
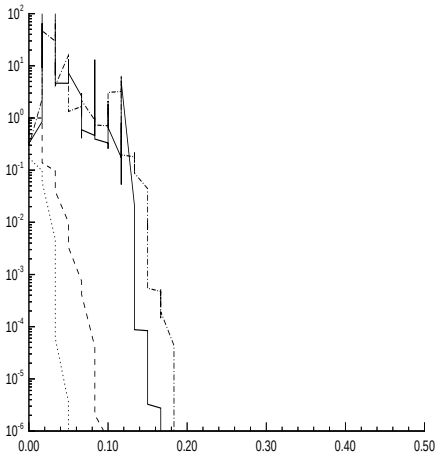


$n = 129 \times 129$

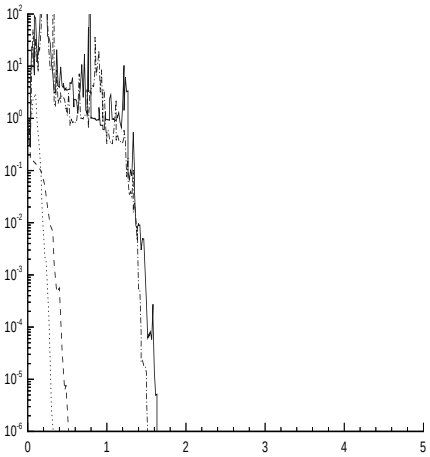
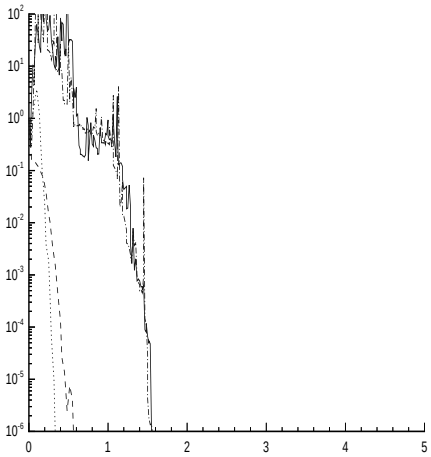




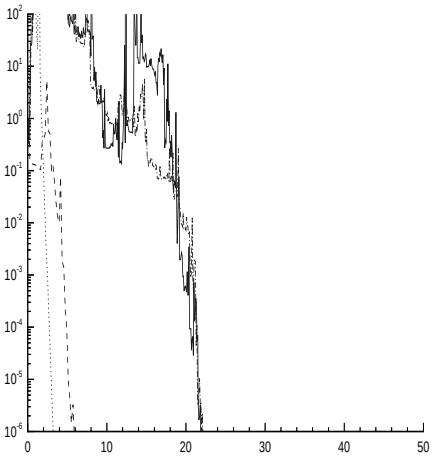
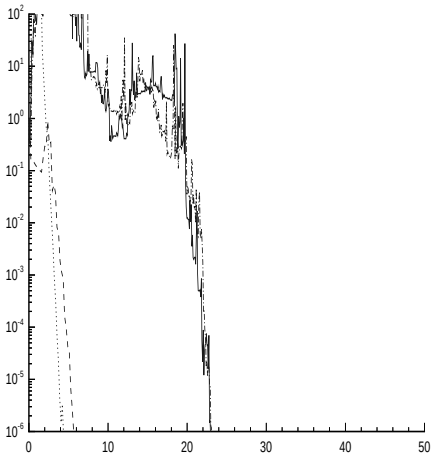
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

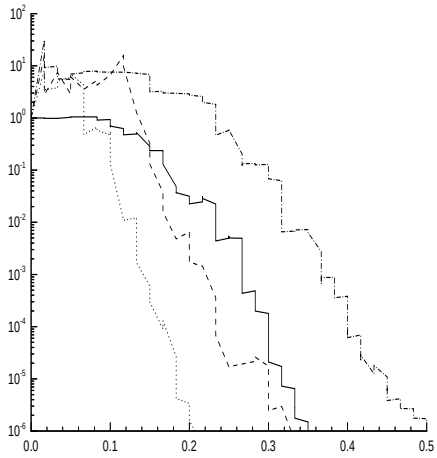
$n = 65 \times 65$

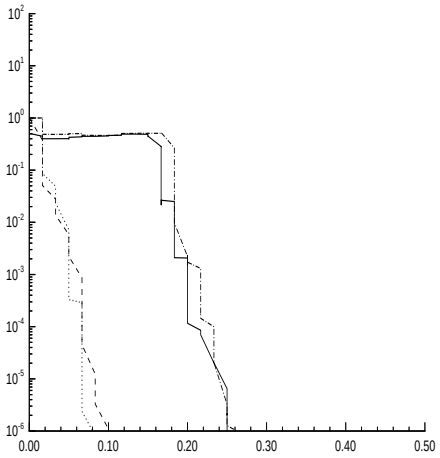



$n = 129 \times 129$

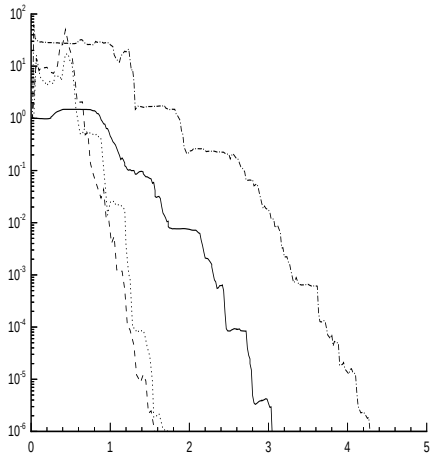
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

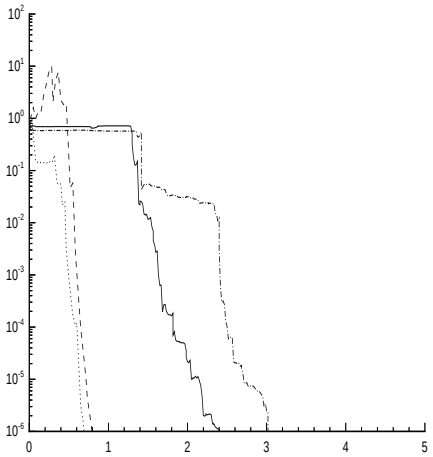
$n = 33 \times 33$



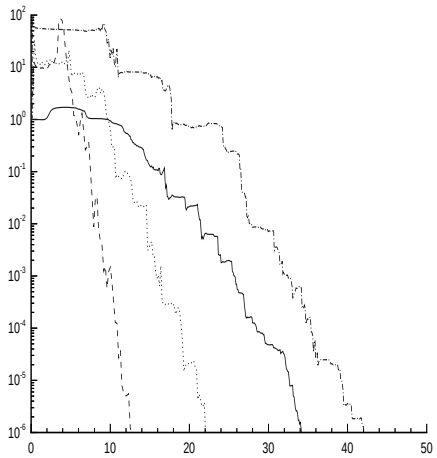


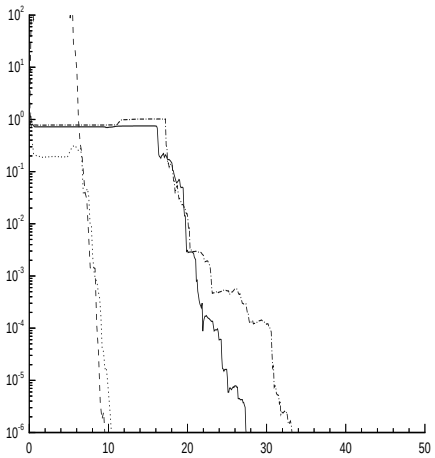
$n = 65 \times 65$



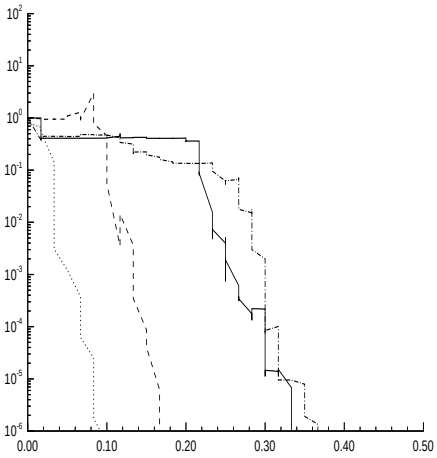
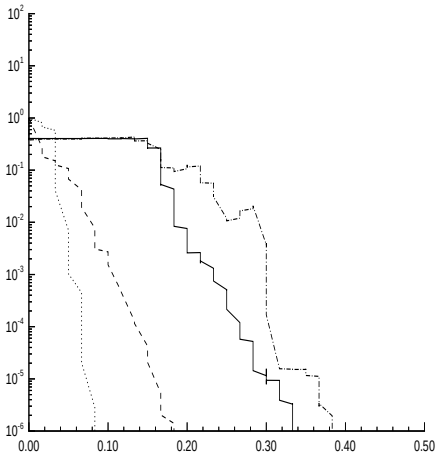


$n = 129 \times 129$

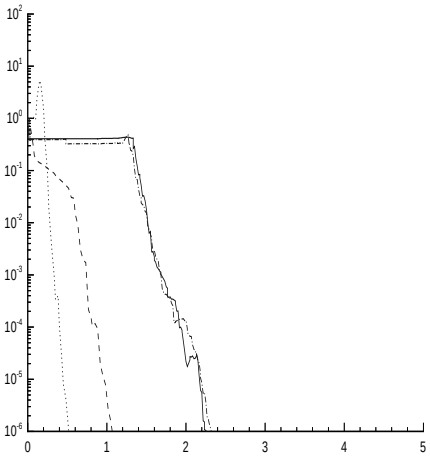
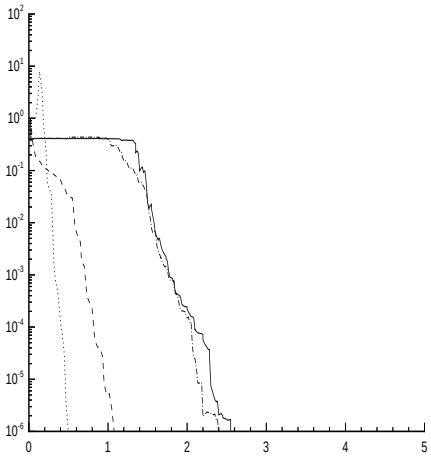




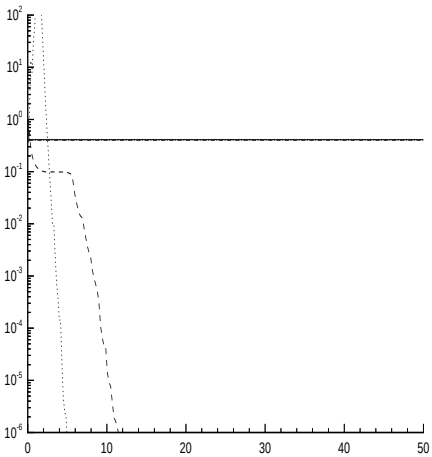
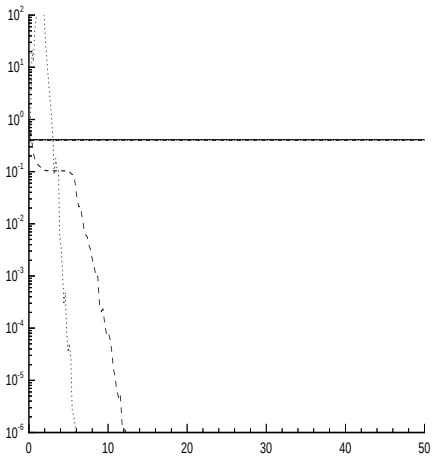
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

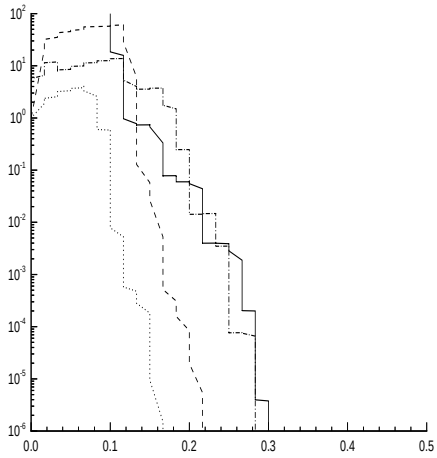
$n = 65 \times 65$

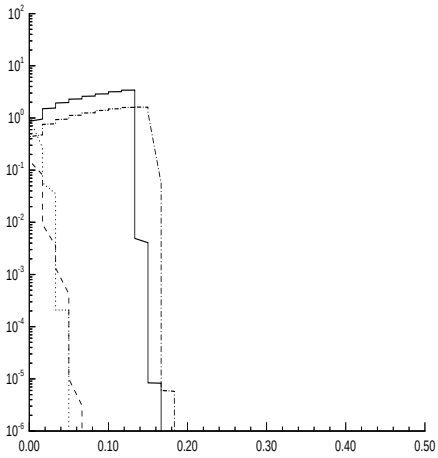



$n = 129 \times 129$

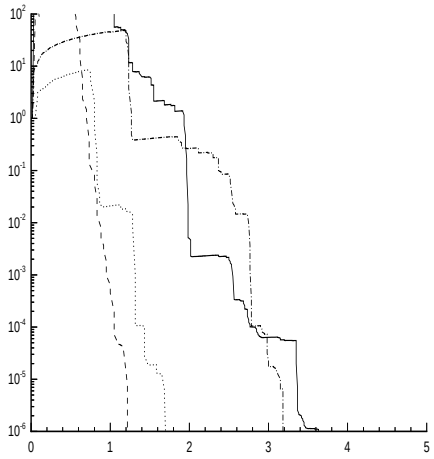
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

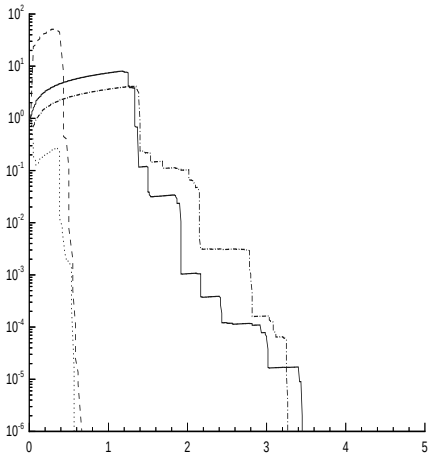
$n = 33 \times 33$



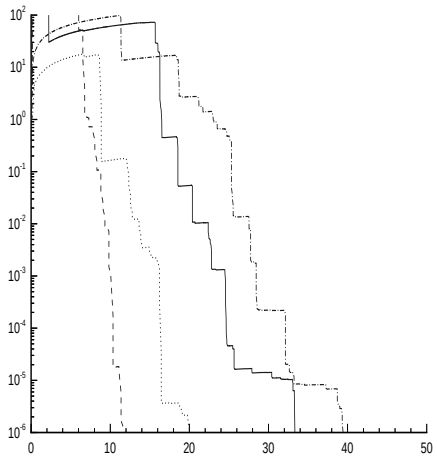


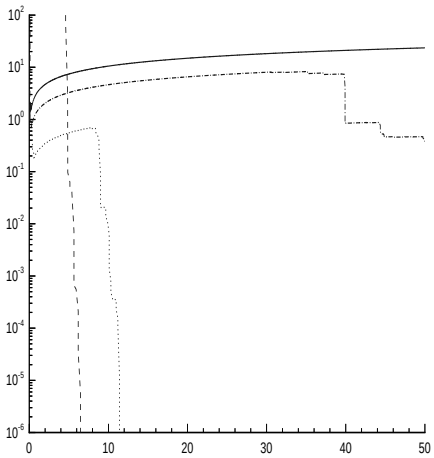
$n = 65 \times 65$



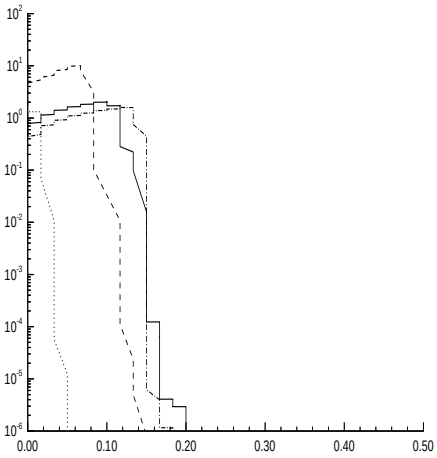
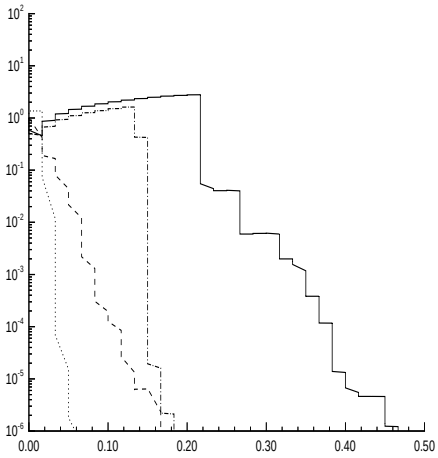


$n = 129 \times 129$

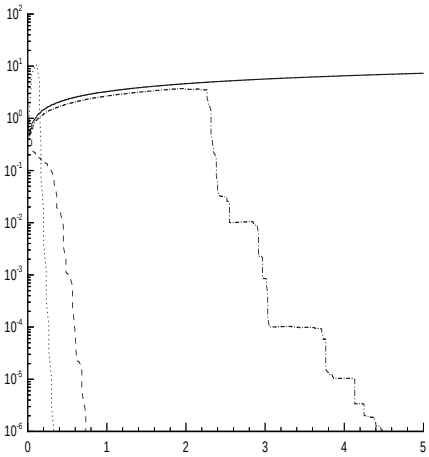
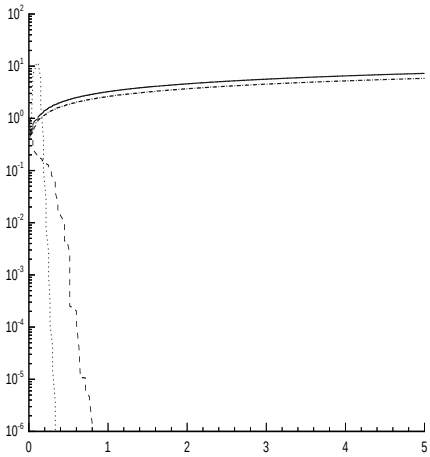




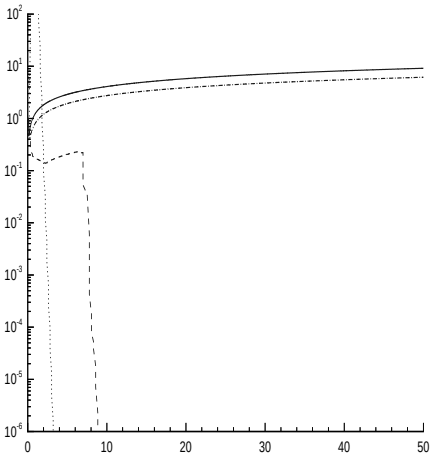
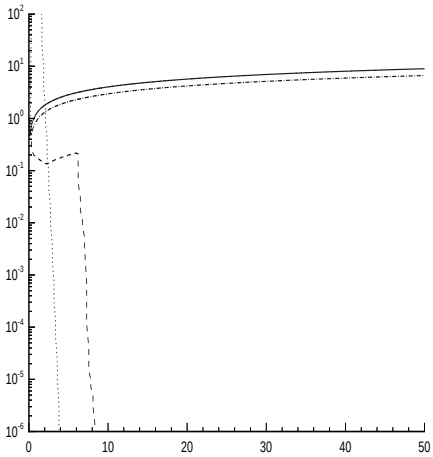
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

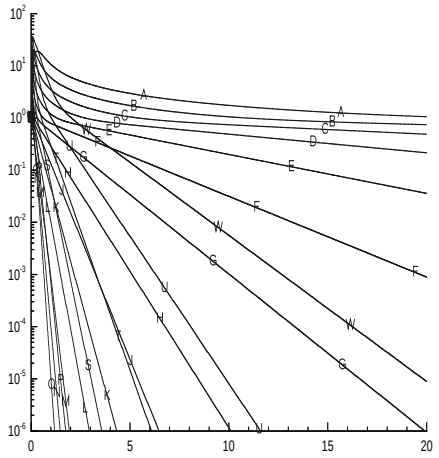
$n = 65 \times 65$

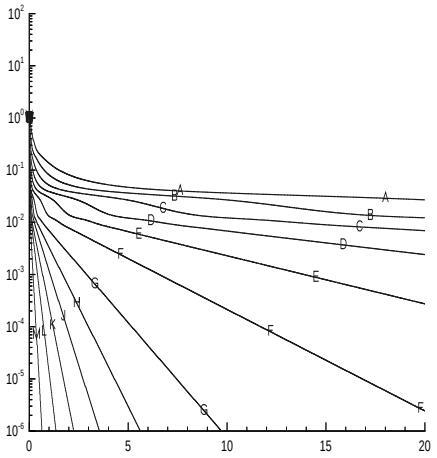



$n = 129 \times 129$

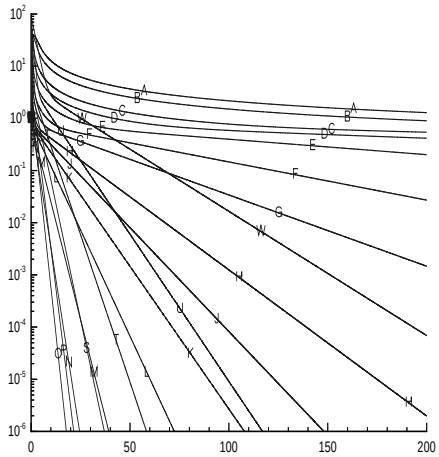
$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

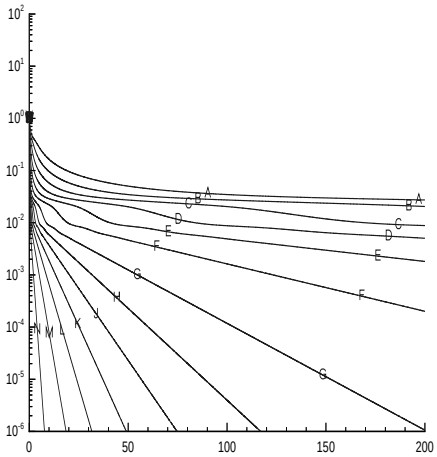
$n = 33 \times 33$



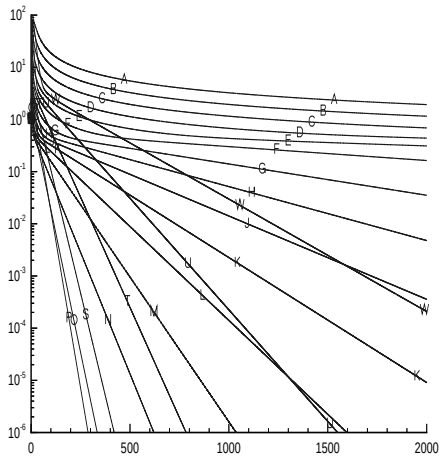


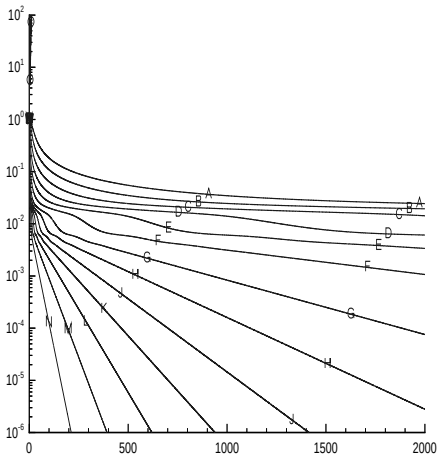
$n = 65 \times 65$



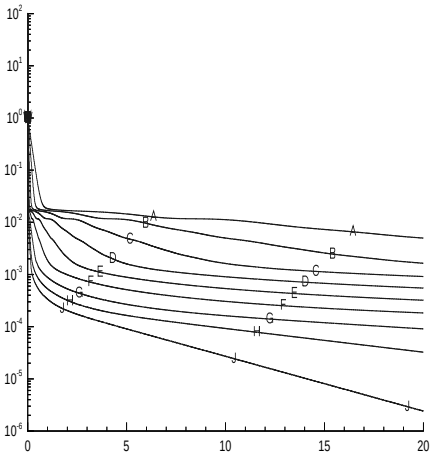


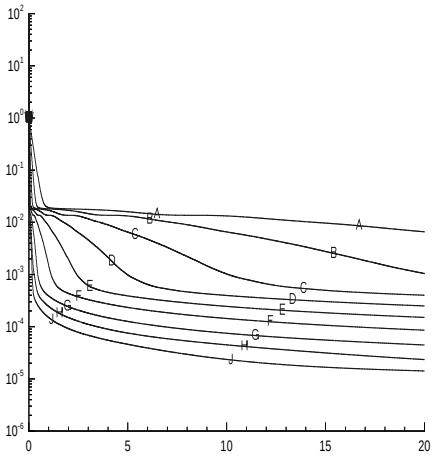
$n = 129 \times 129$



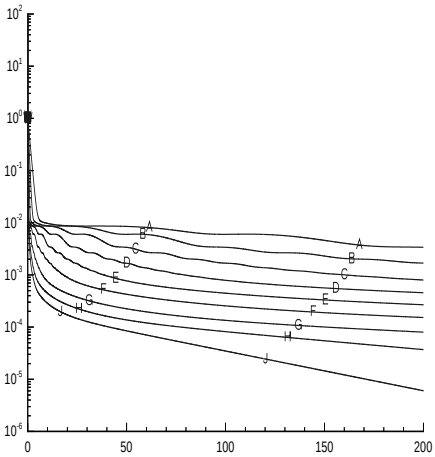


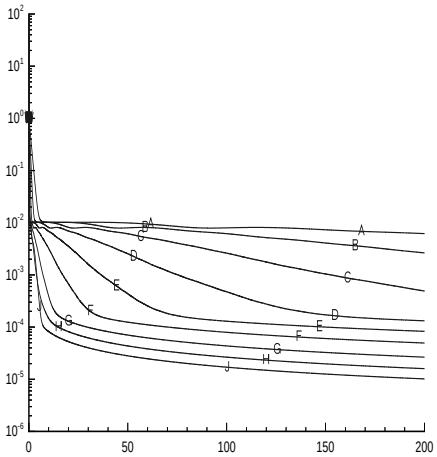
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



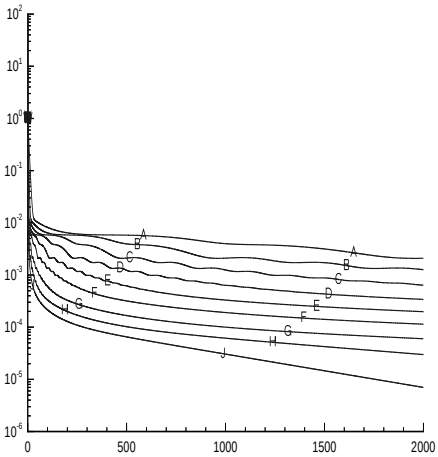


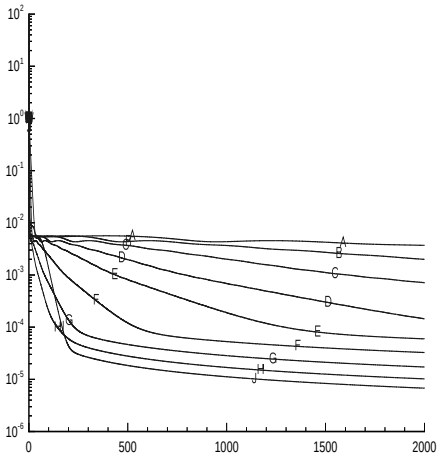
$n = 33 \times 33$





$n = 65 \times 65$

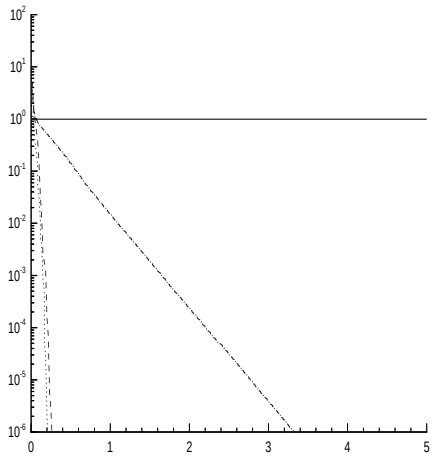
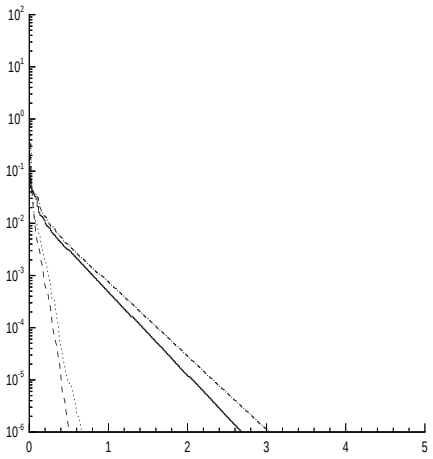




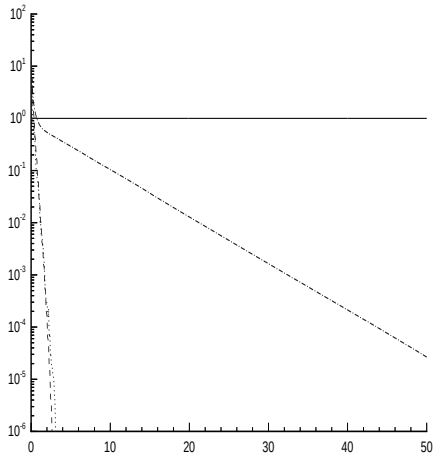
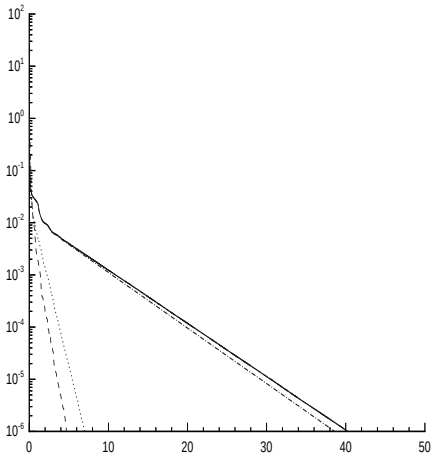
$n = 129 \times 129$

$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

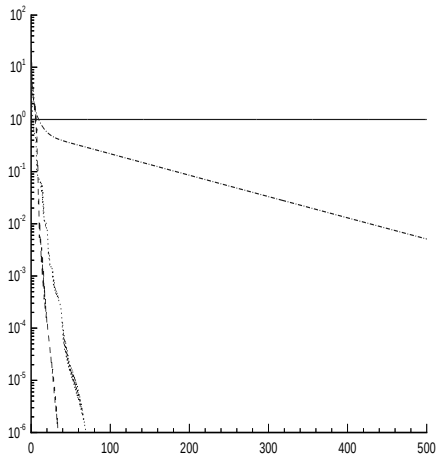
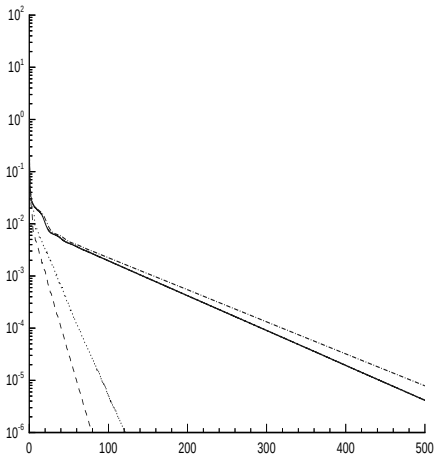
$n = 33 \times 33$

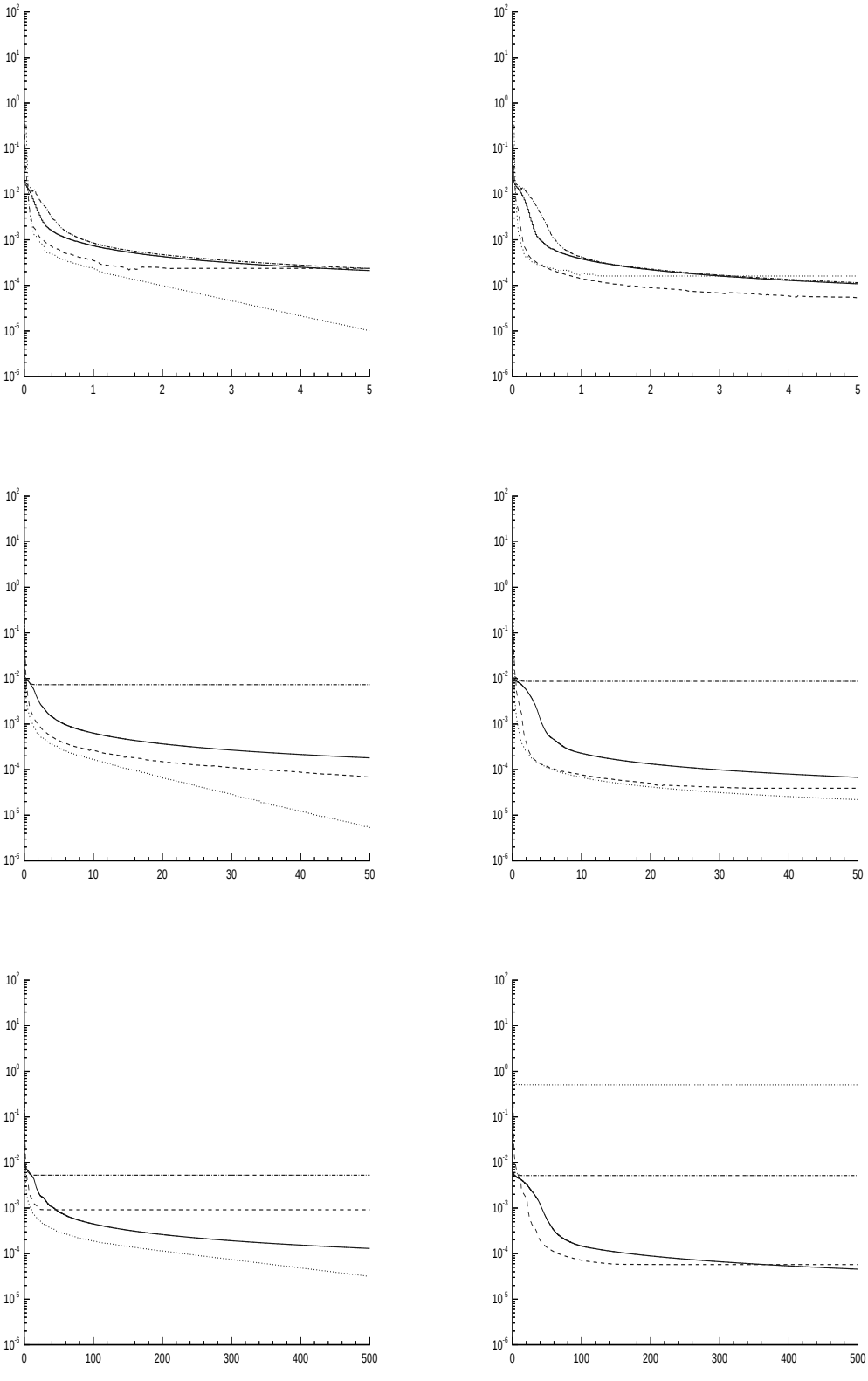
$n = 65 \times 65$

$n = 129 \times 129$

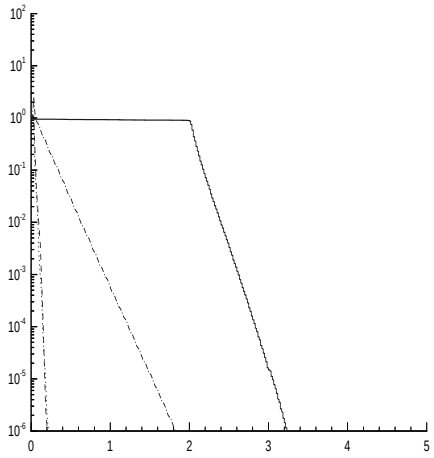
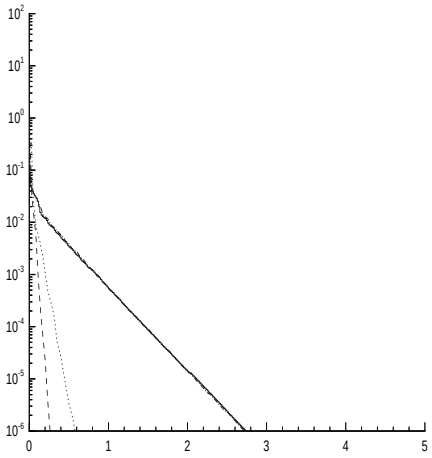



$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

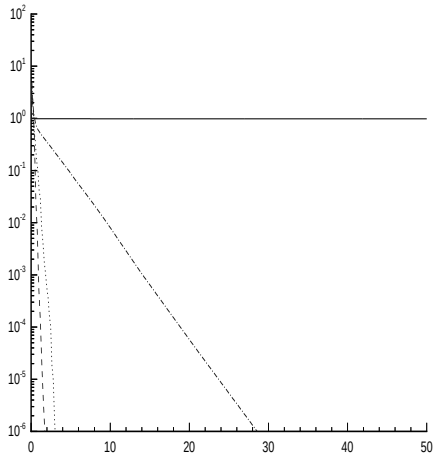
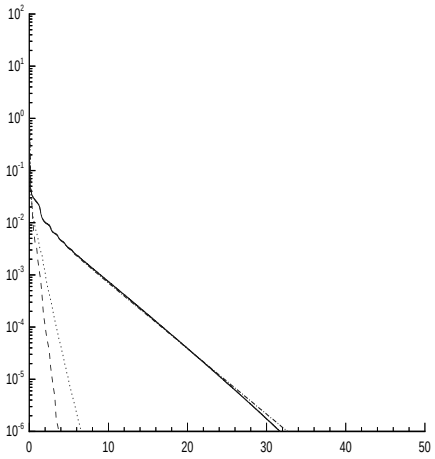


$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

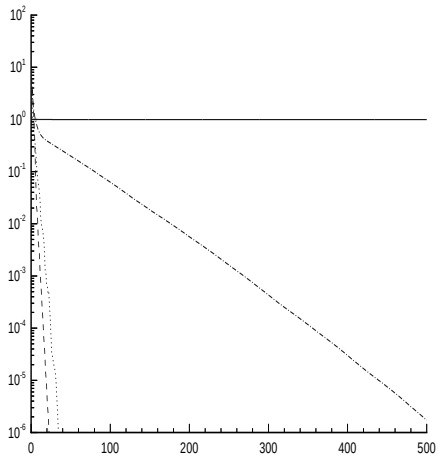
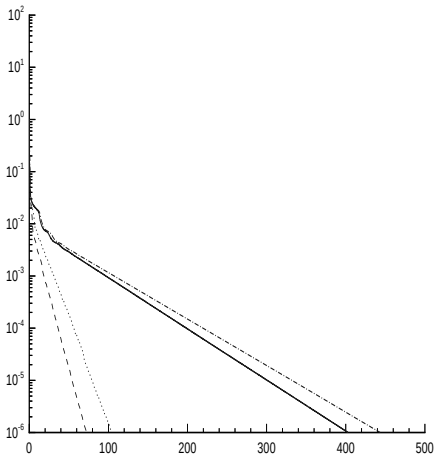
$n = 33 \times 33$

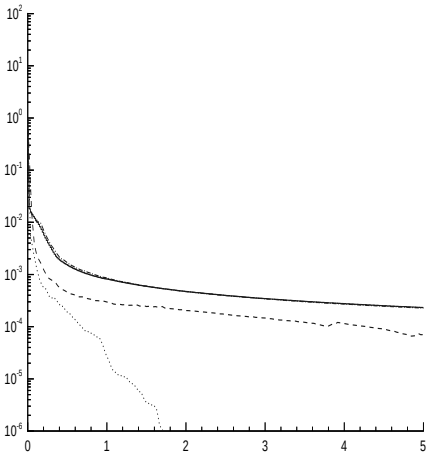
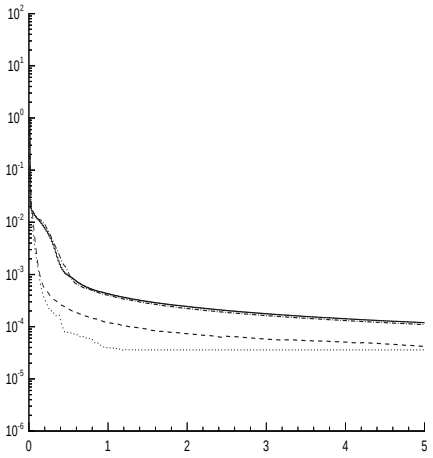
$n = 65 \times 65$

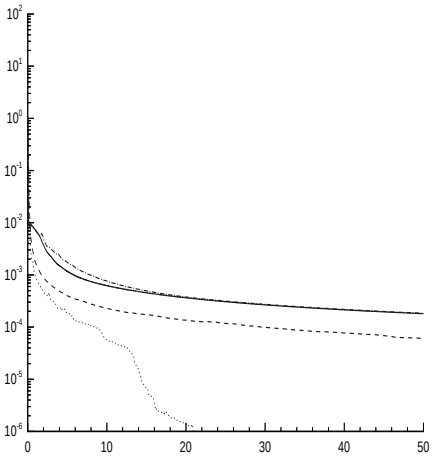
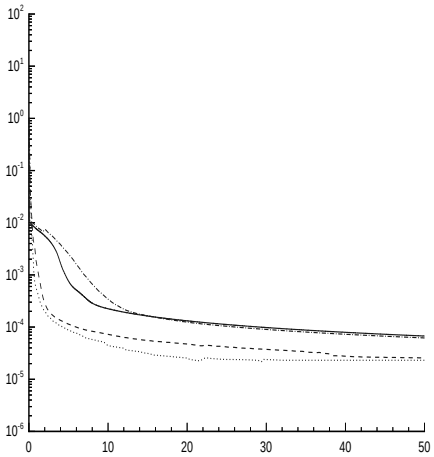
$n = 129 \times 129$

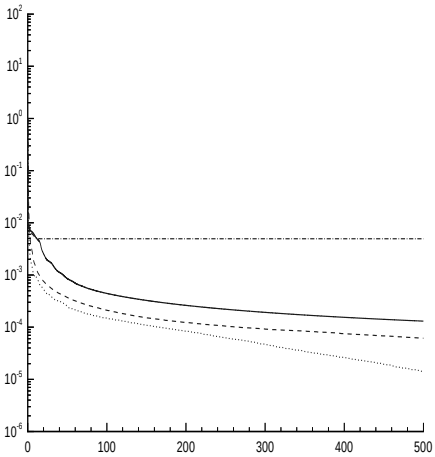
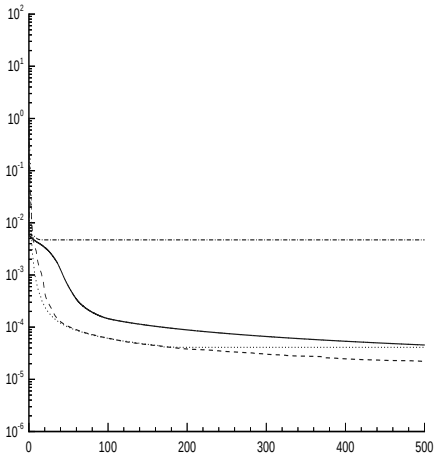
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

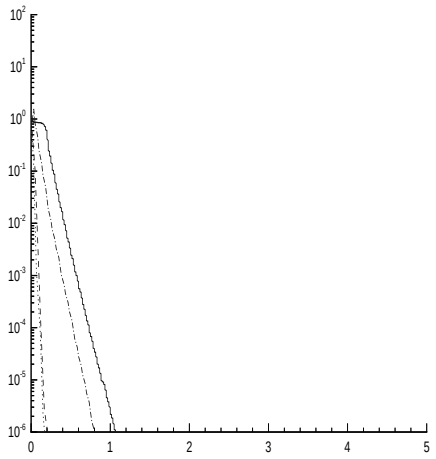
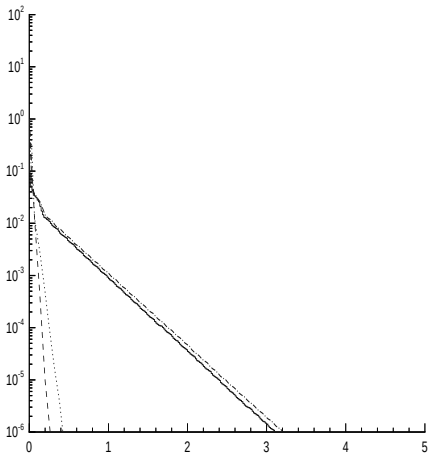
$n = 65 \times 65$

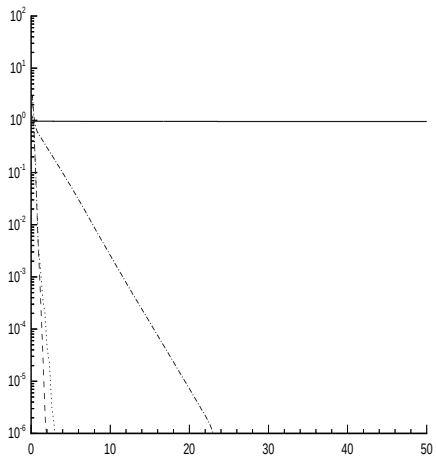
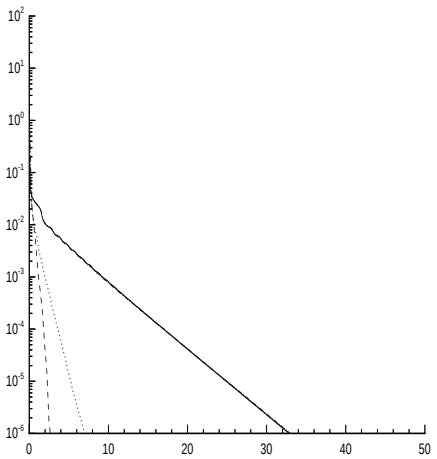
$n = 129 \times 129$

$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

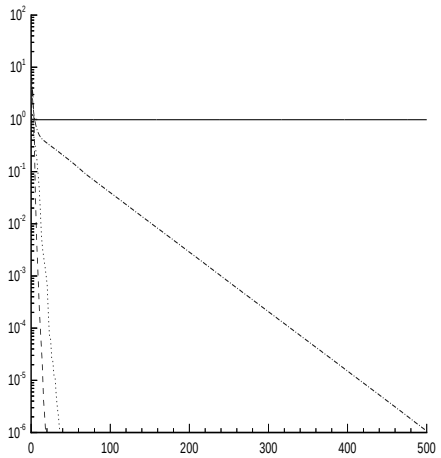
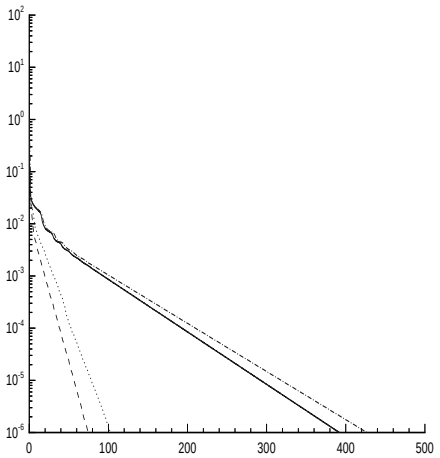
$n = 33 \times 33$

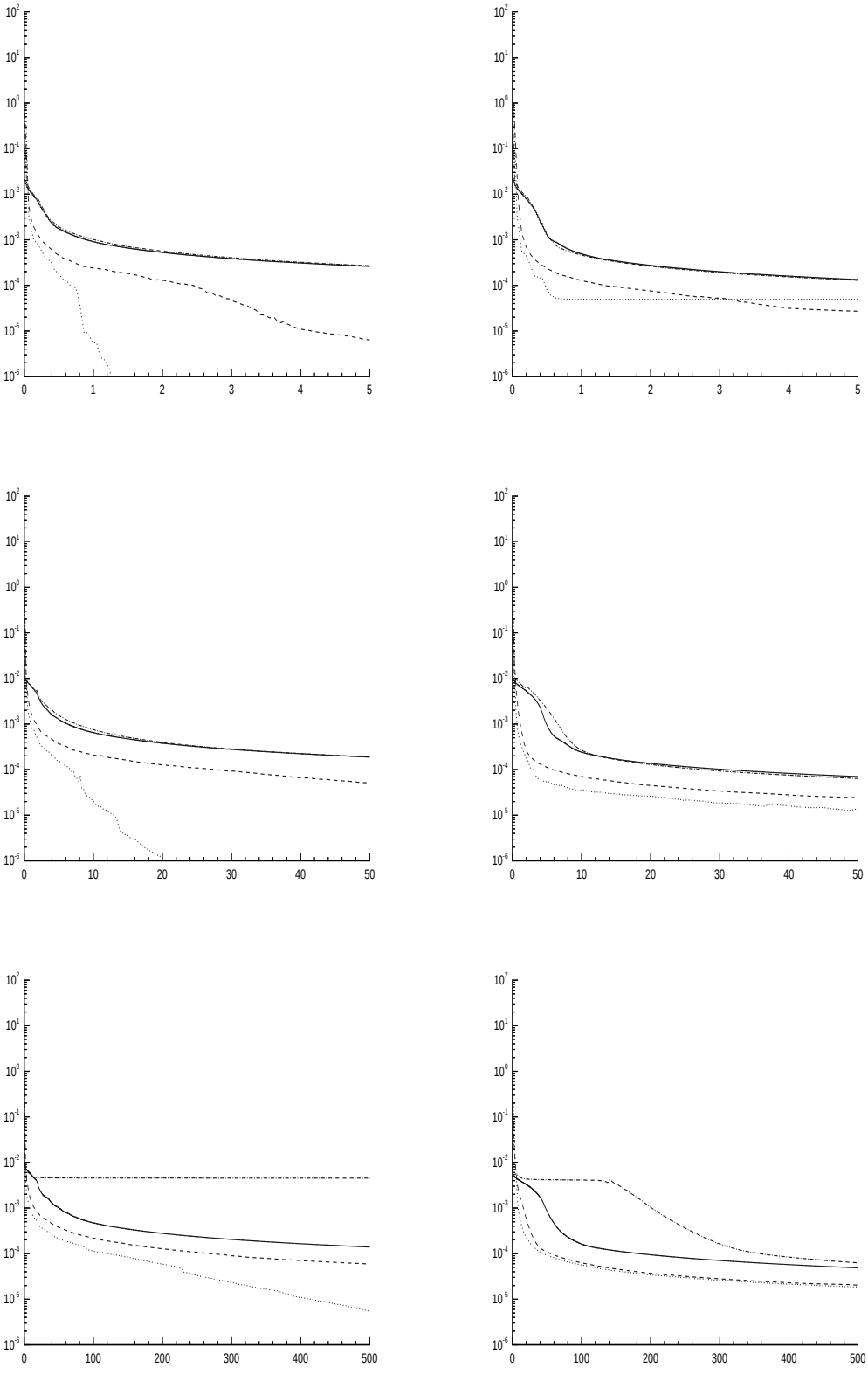
$n = 65 \times 65$

$n = 129 \times 129$

$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



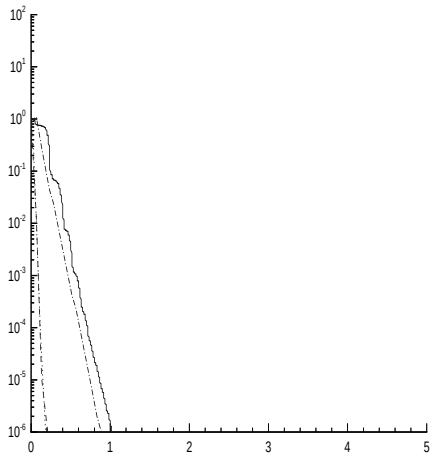
$n = 33 \times 33$

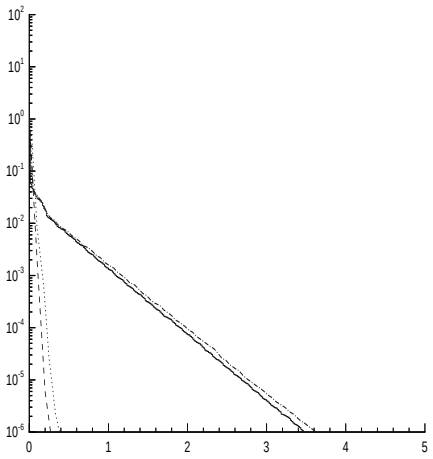
$n = 65 \times 65$

$n = 129 \times 129$

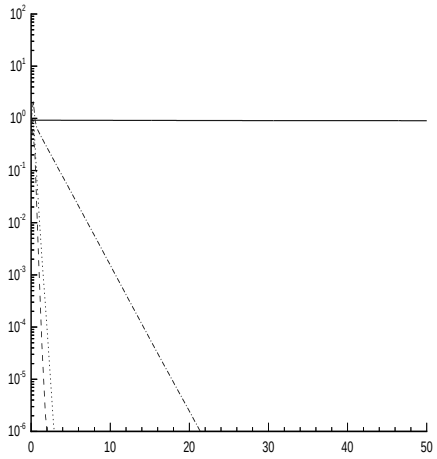
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

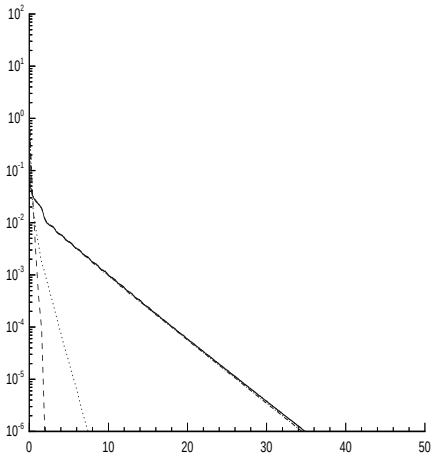
$n = 33 \times 33$



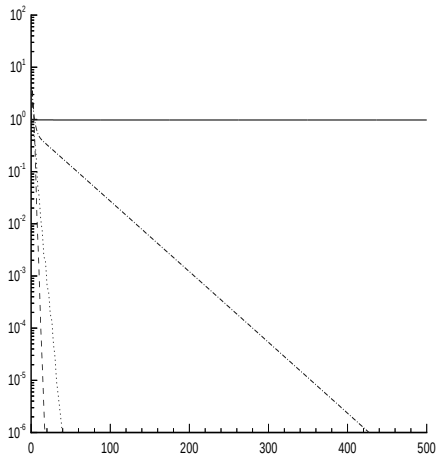


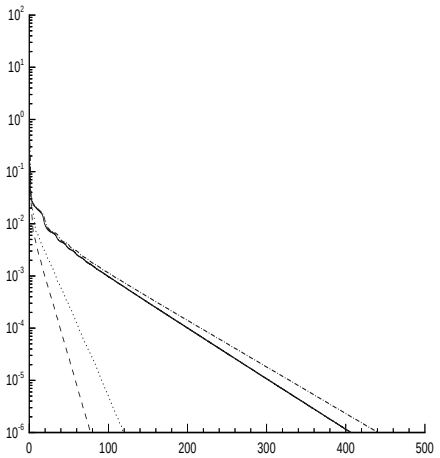
$n = 65 \times 65$



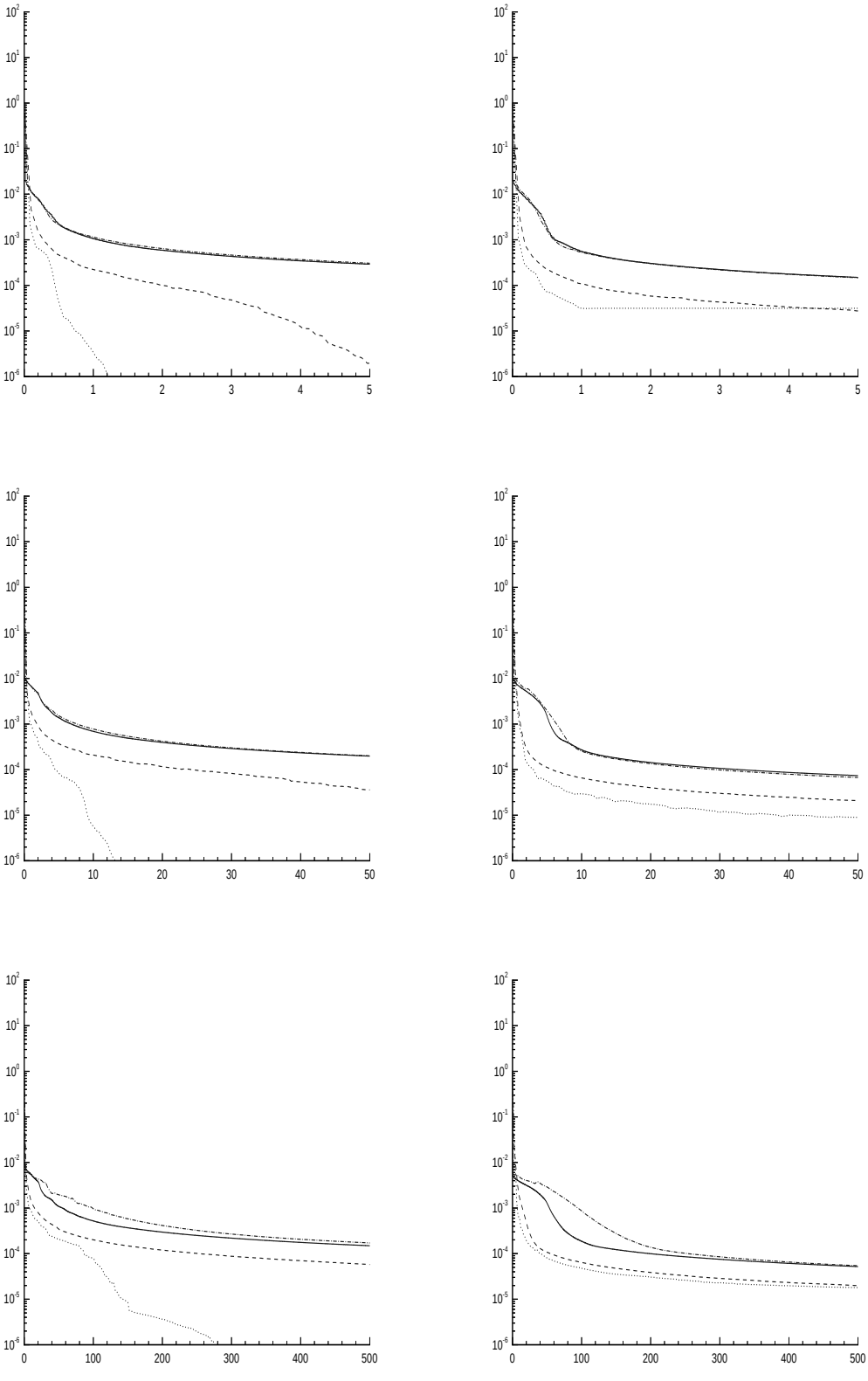


$n = 129 \times 129$



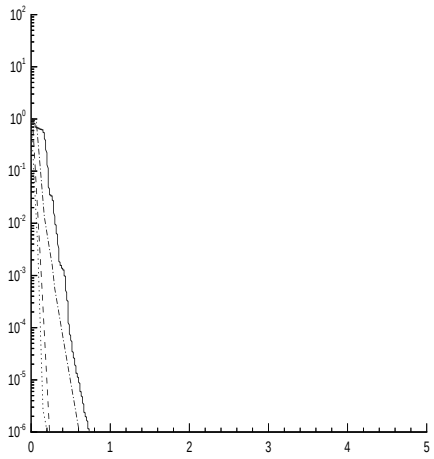


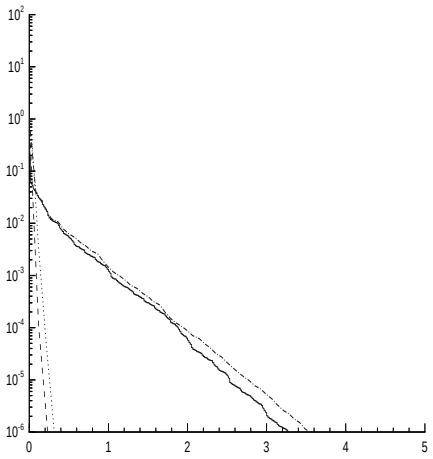
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



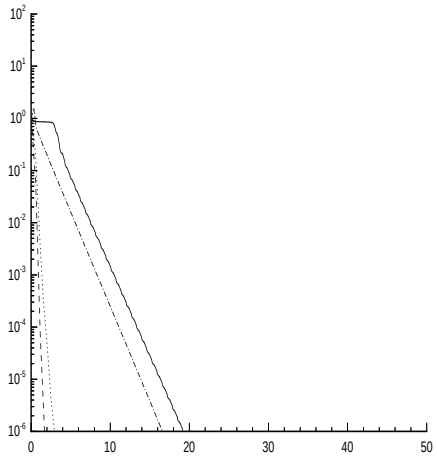
$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

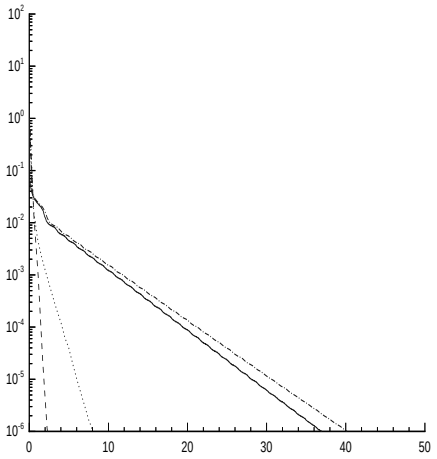
$n = 33 \times 33$



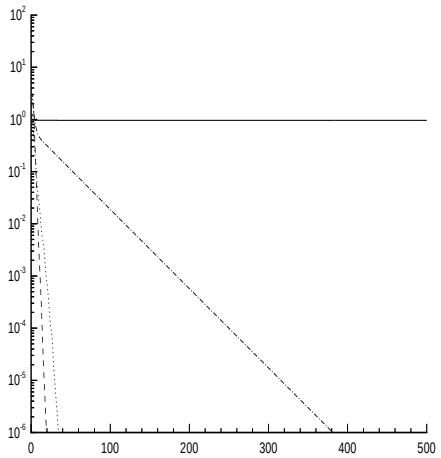


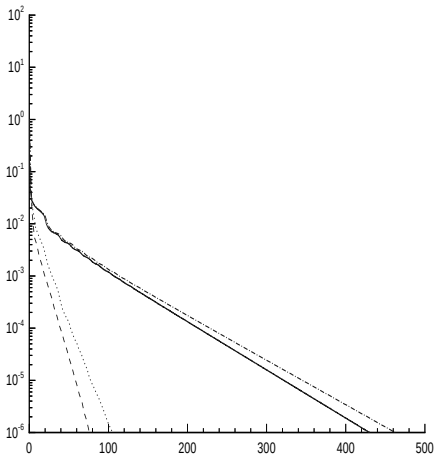
$n = 65 \times 65$



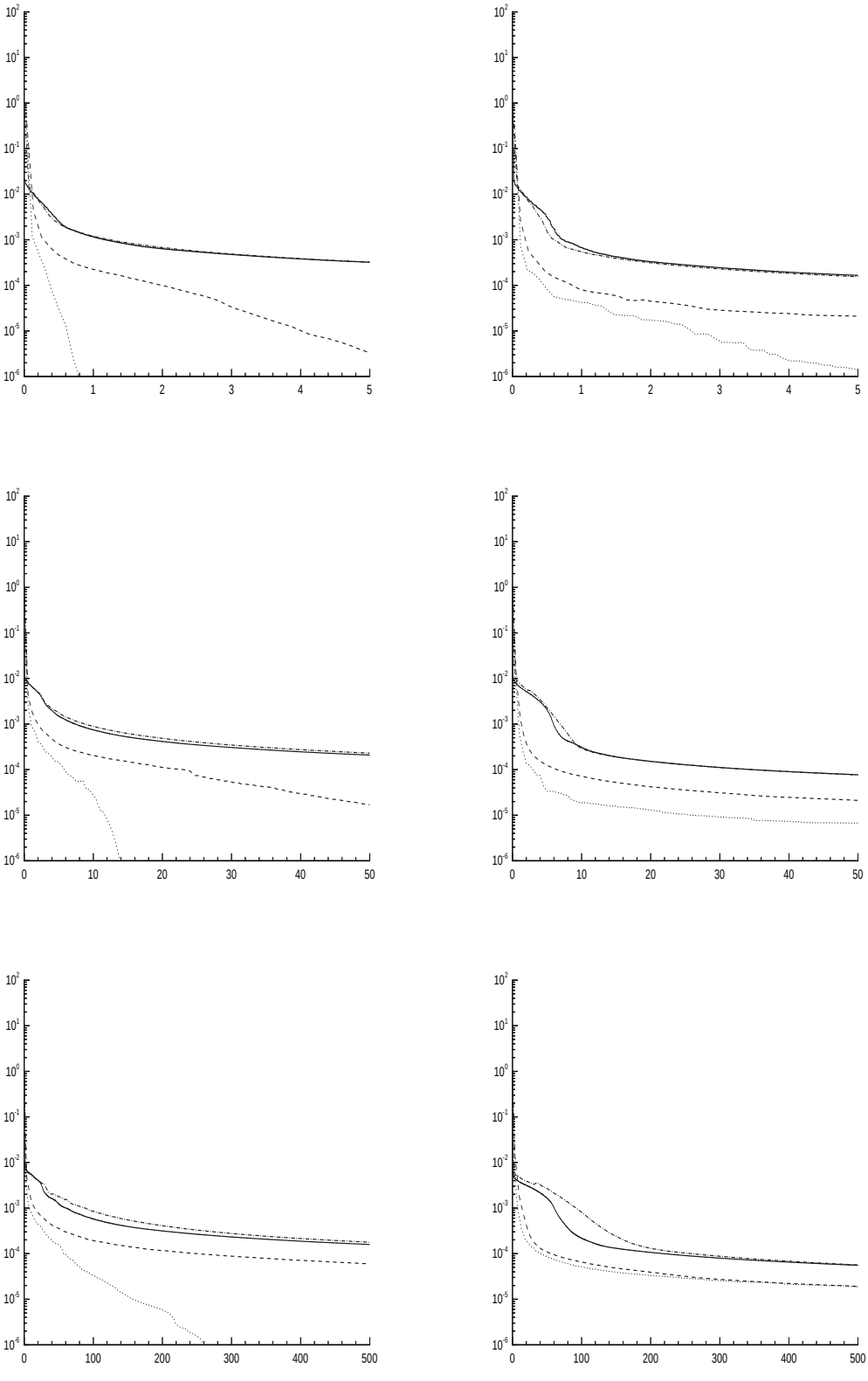


$n = 129 \times 129$



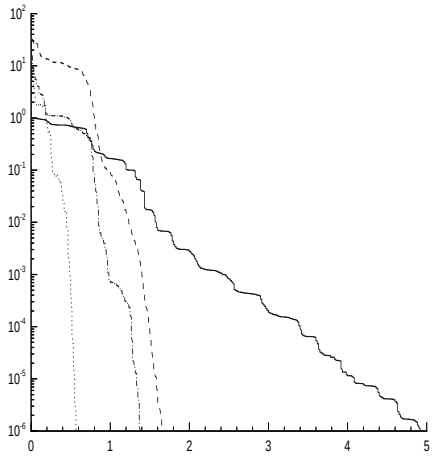
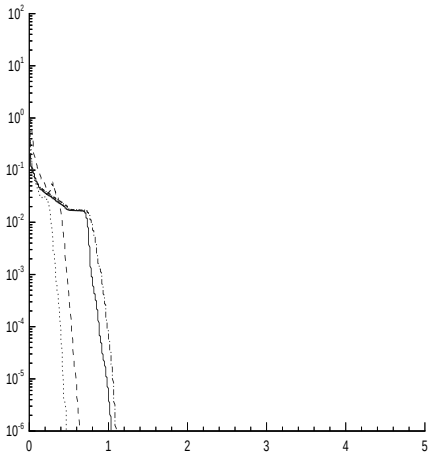


$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

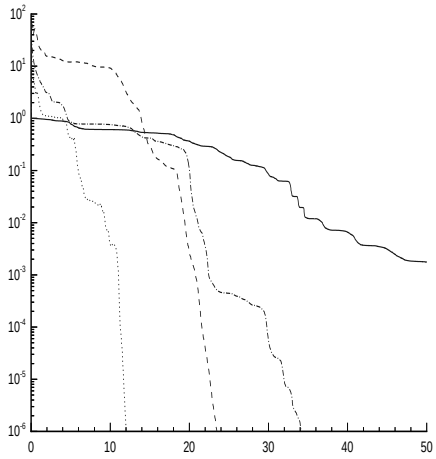
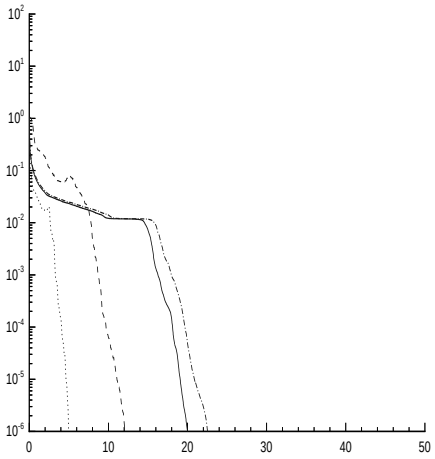


$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

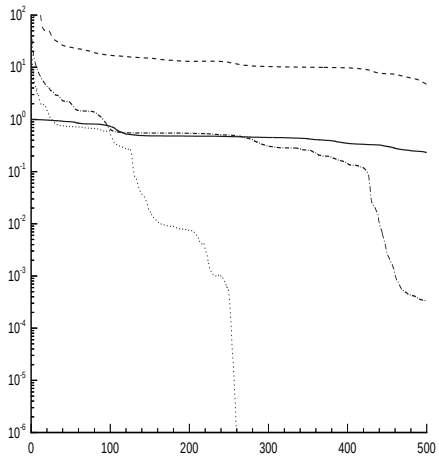
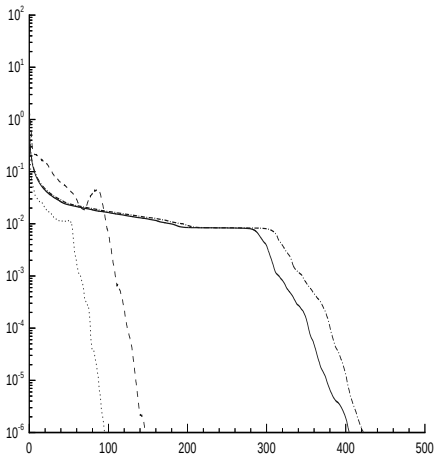
$n = 33 \times 33$

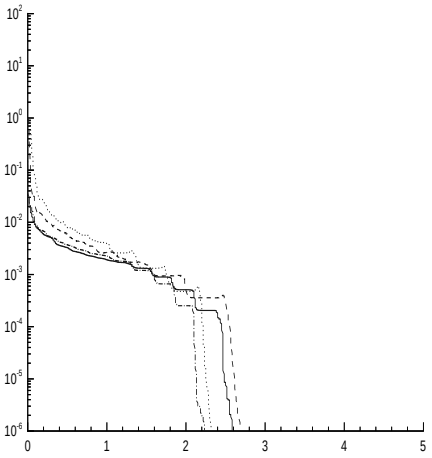
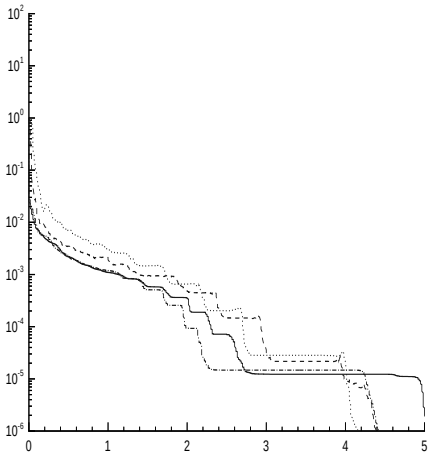
$n = 65 \times 65$

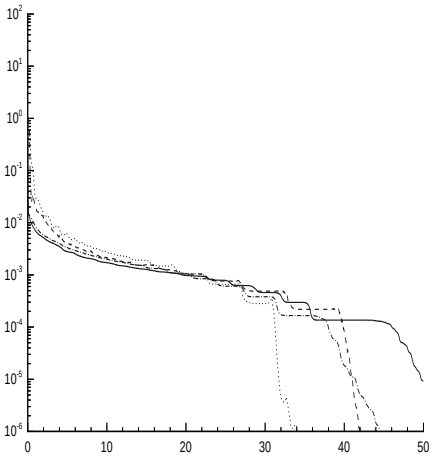
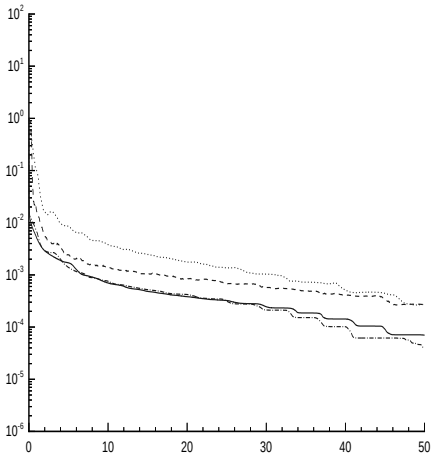
$n = 129 \times 129$

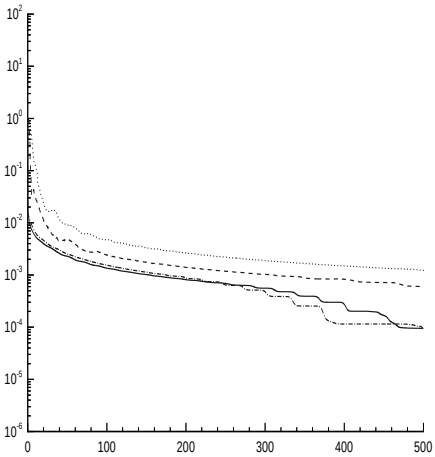
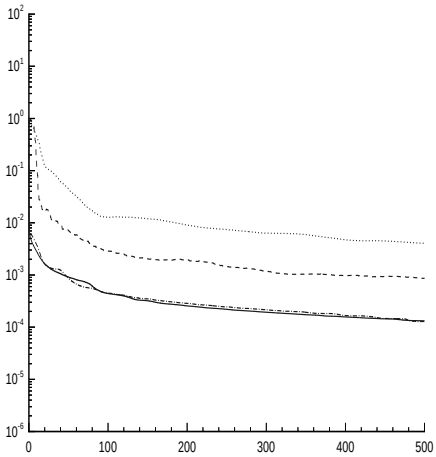
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

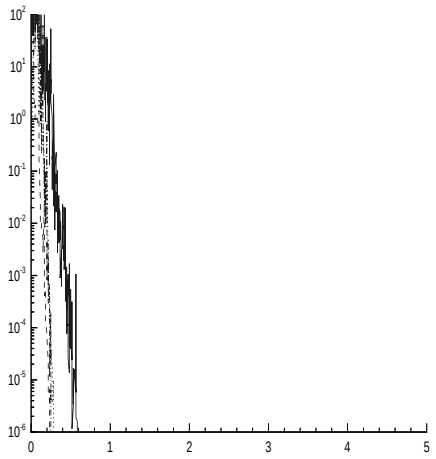
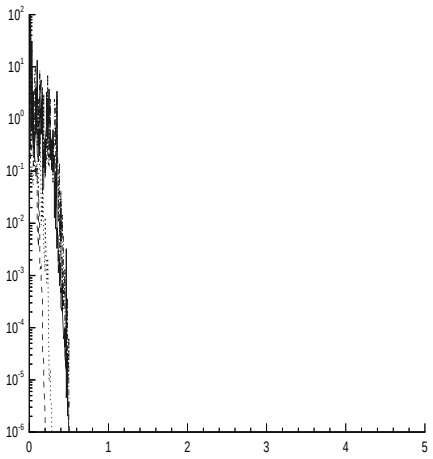
$n = 65 \times 65$

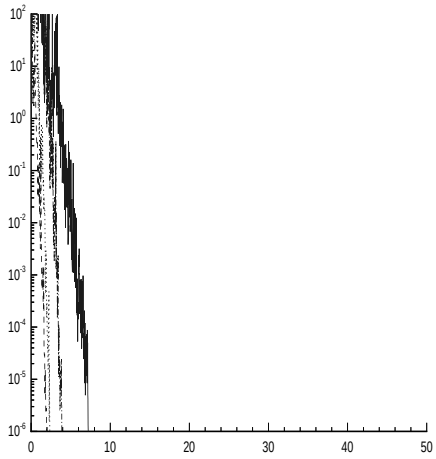
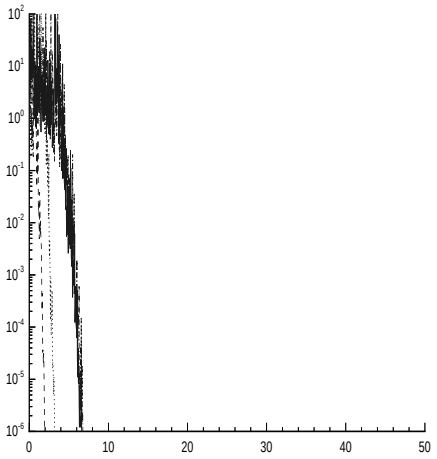
$n = 129 \times 129$

$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

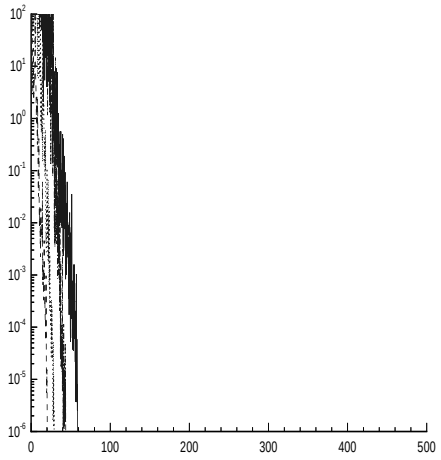
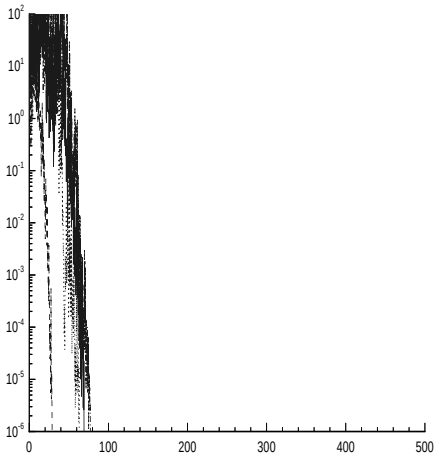
$n = 33 \times 33$

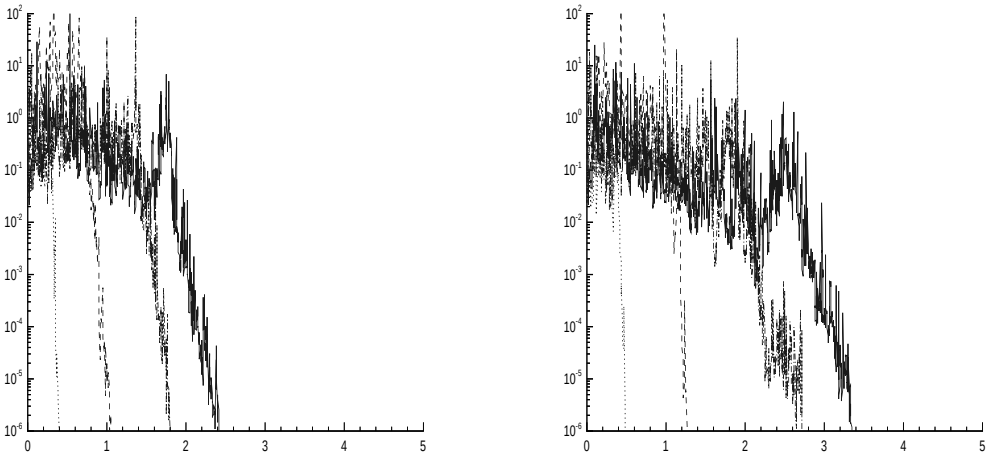
$n = 65 \times 65$

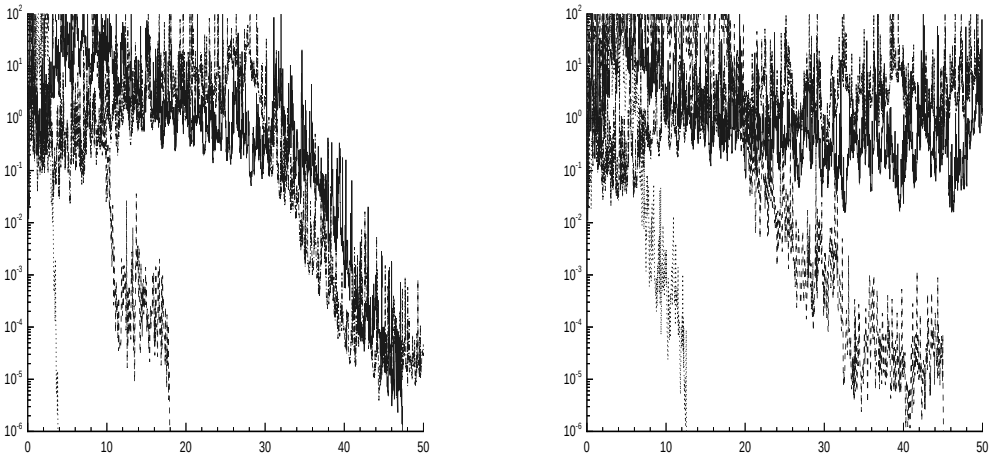
$n = 129 \times 129$

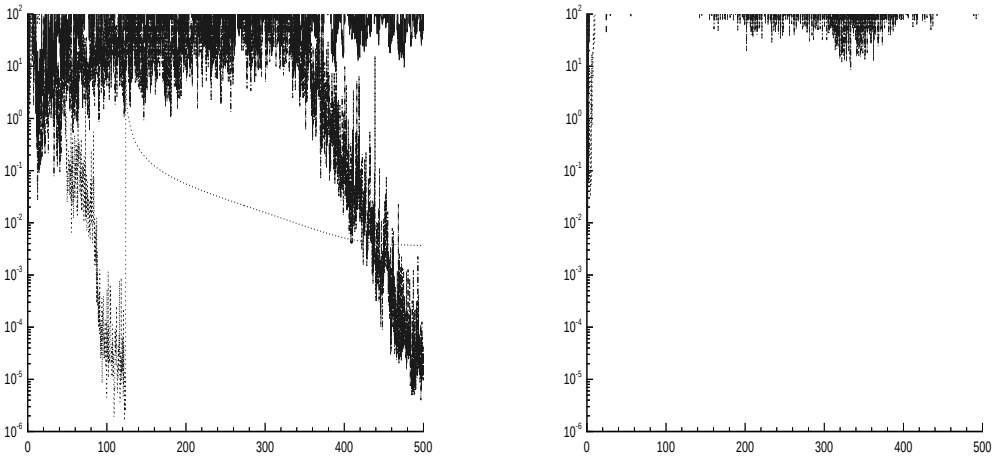
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



$n = 33 \times 33$



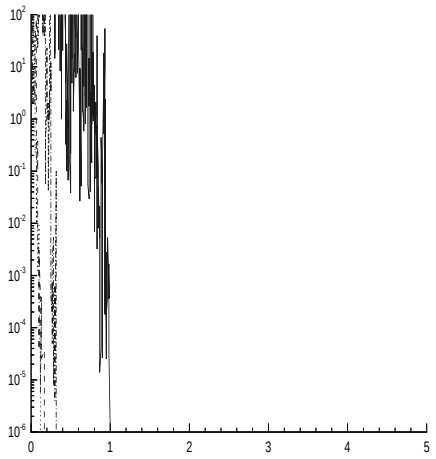
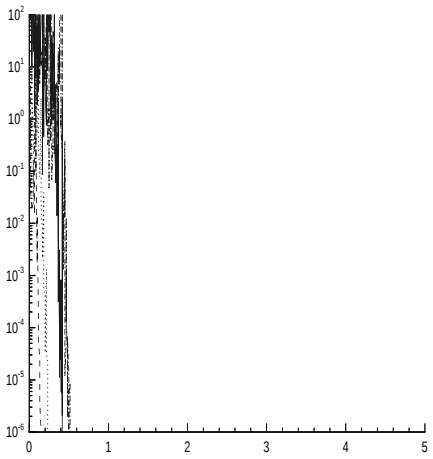
$n = 65 \times 65$



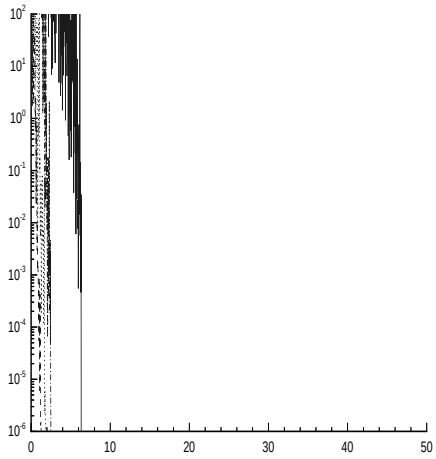
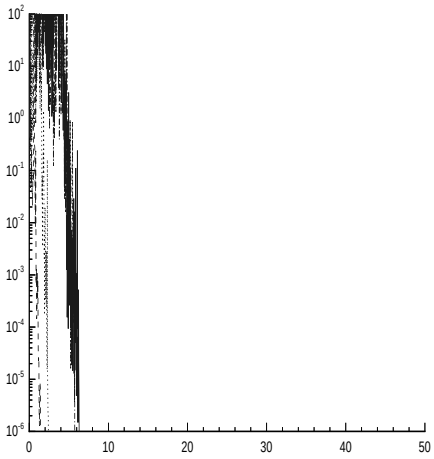
$n = 129 \times 129$

$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

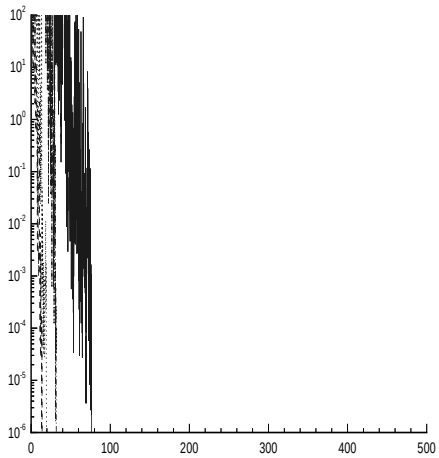
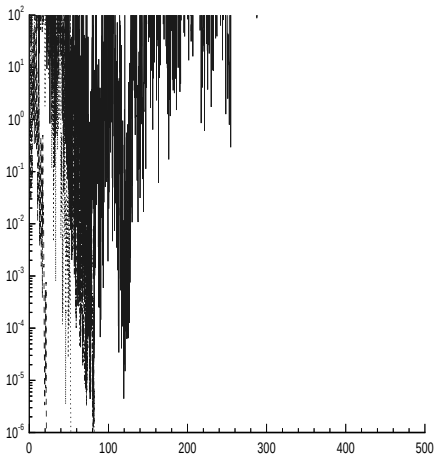
$n = 33 \times 33$

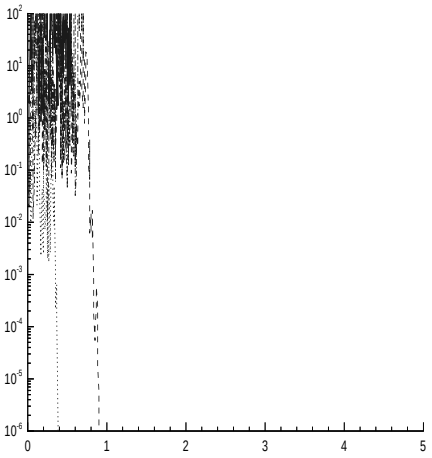
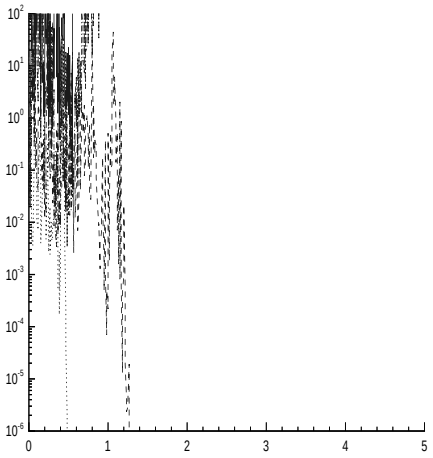
$n = 65 \times 65$

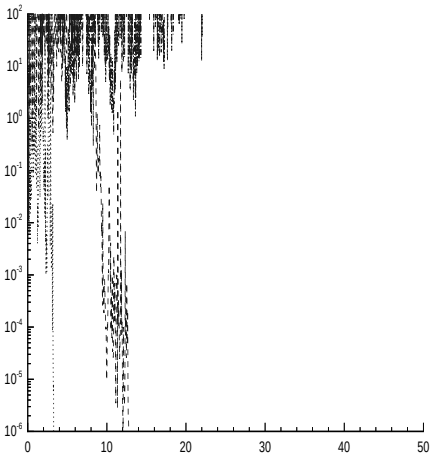
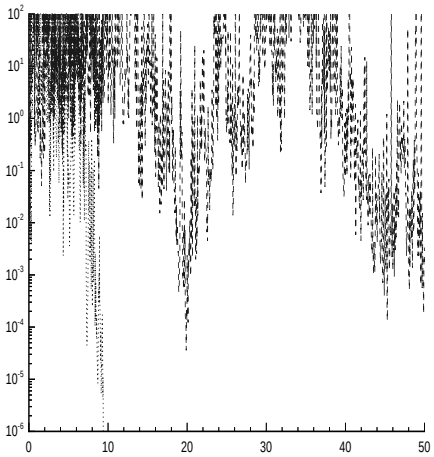
$n = 129 \times 129$

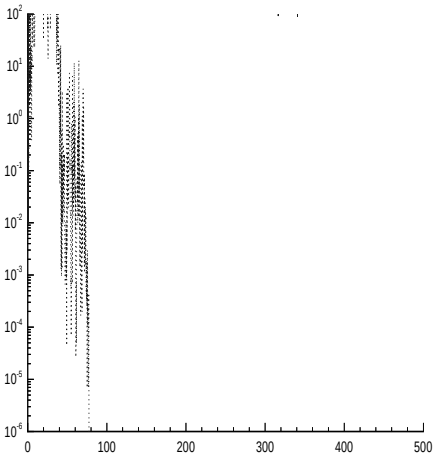
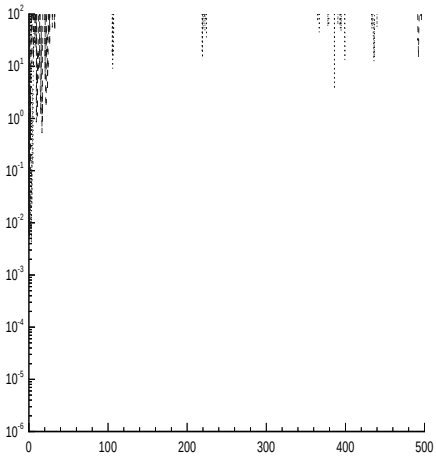
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

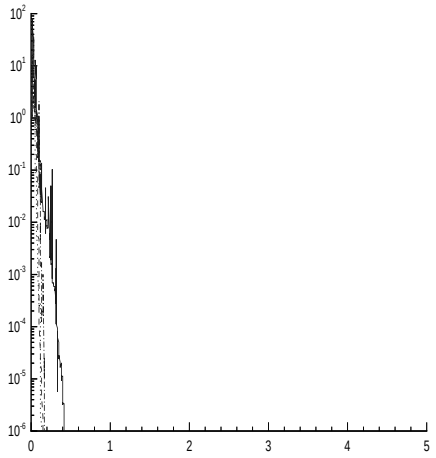
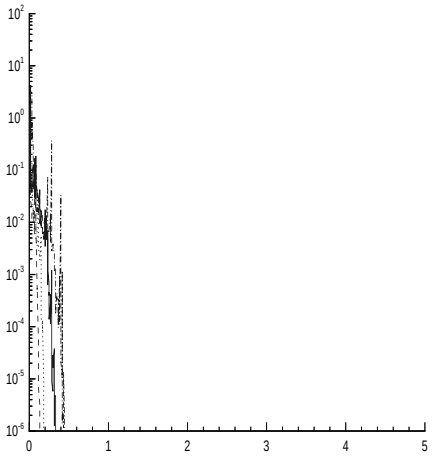
$n = 65 \times 65$

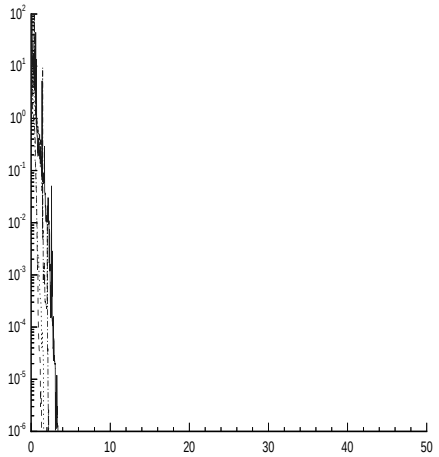
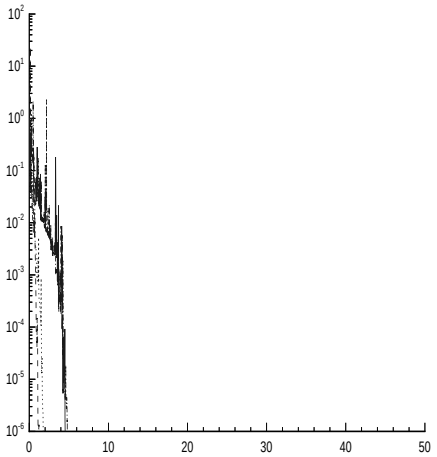
$n = 129 \times 129$

$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

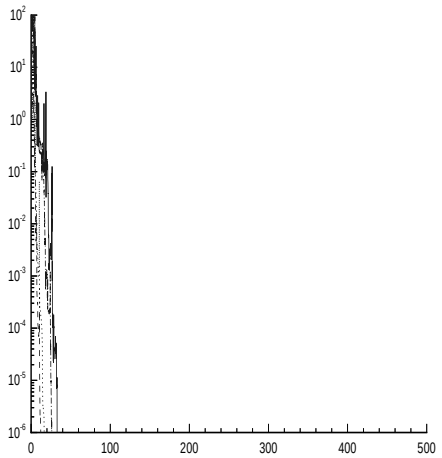
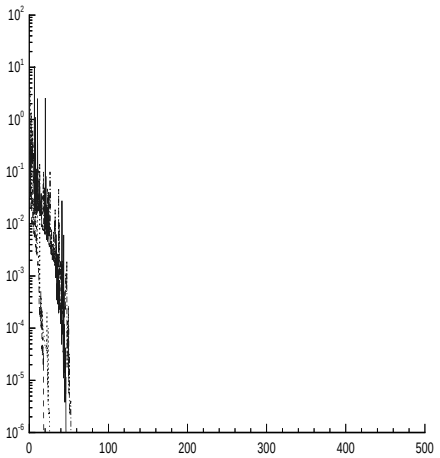
$n = 33 \times 33$

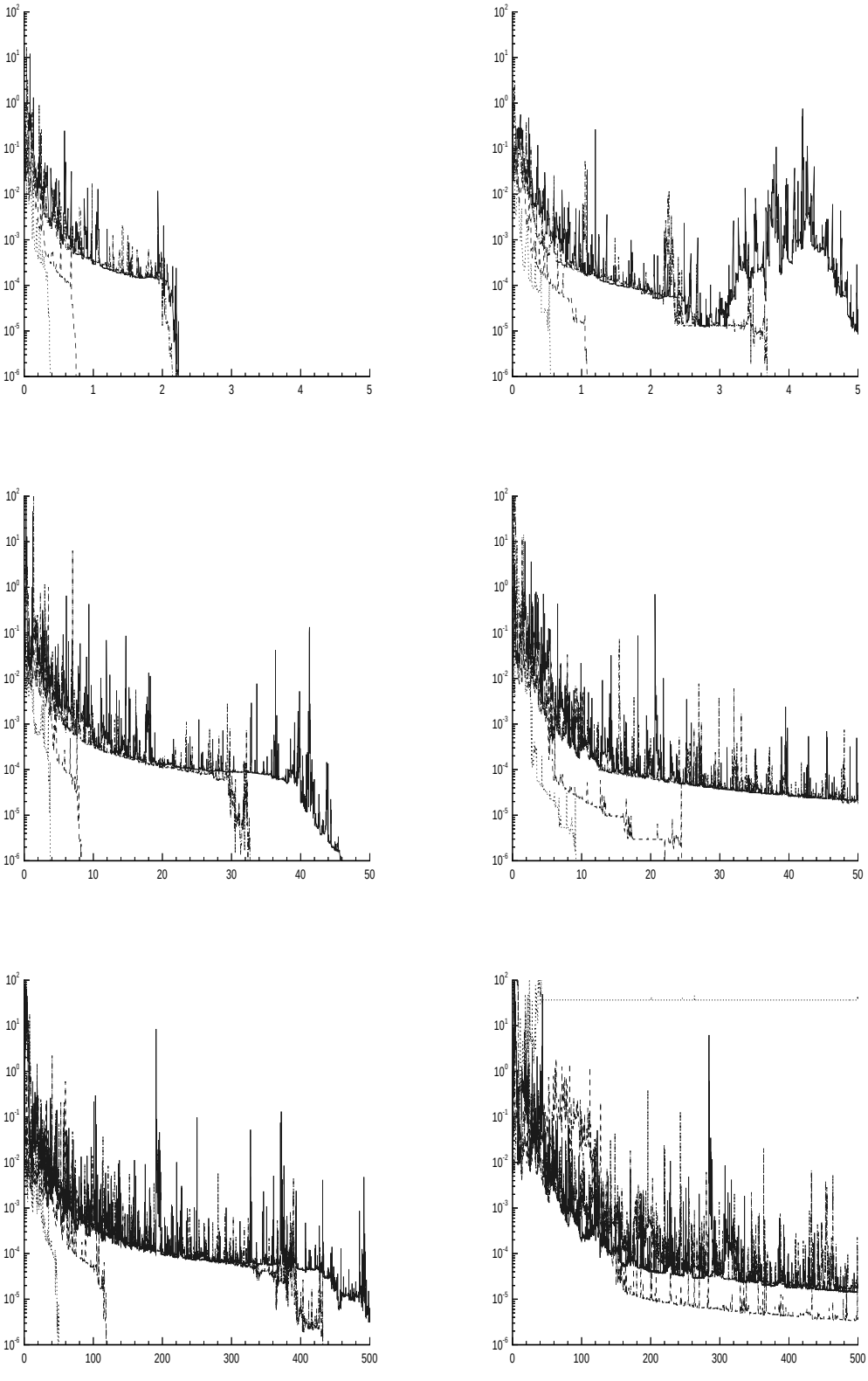
$n = 65 \times 65$

$n = 129 \times 129$

$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$



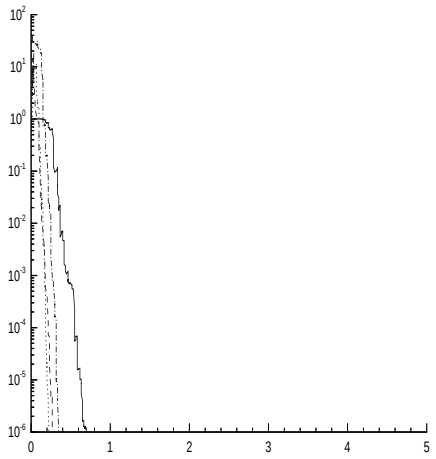
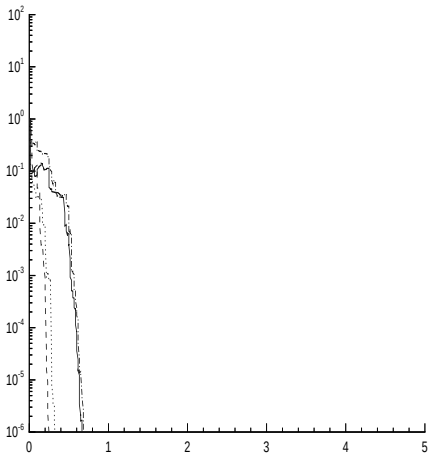
$n = 33 \times 33$

$n = 65 \times 65$

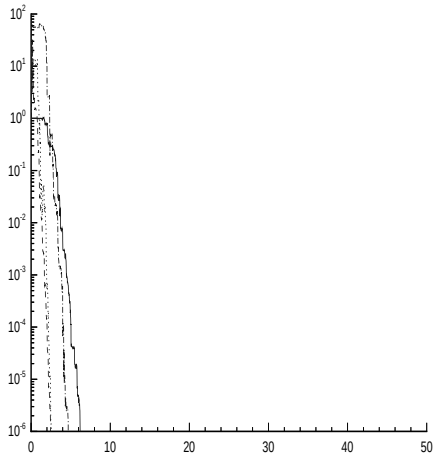
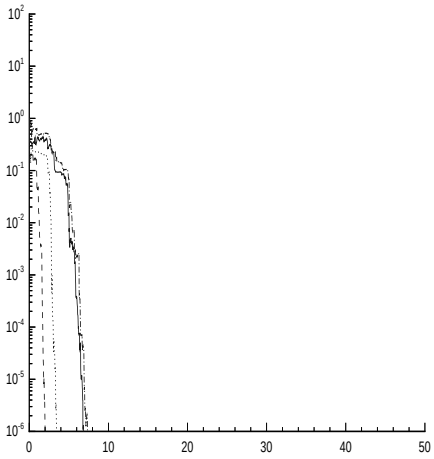
$n = 129 \times 129$

$\varepsilon = 10^{0}$ $\varepsilon = 10^{-2}$

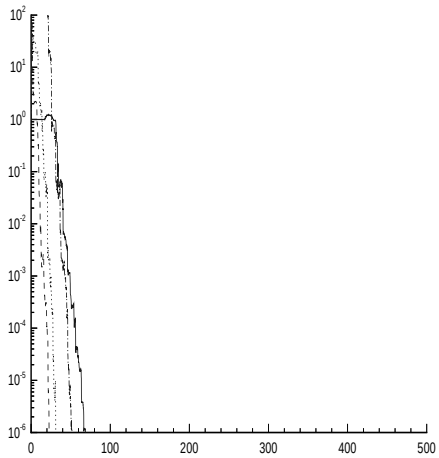
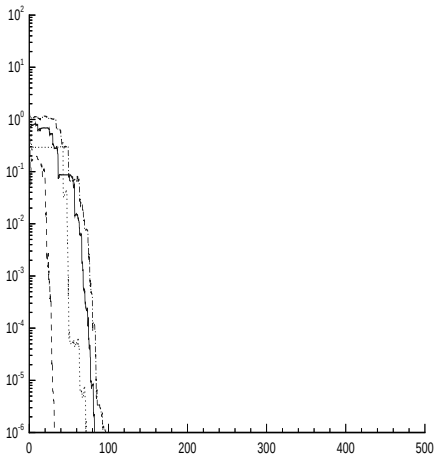
$n = 33 \times 33$

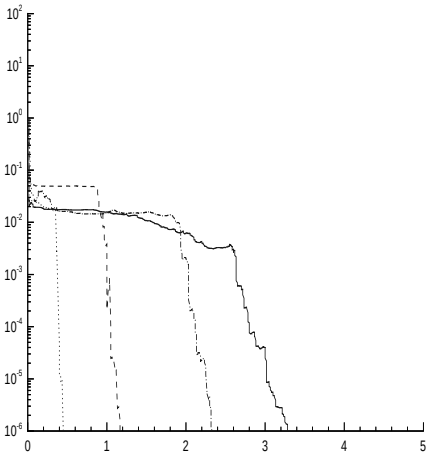
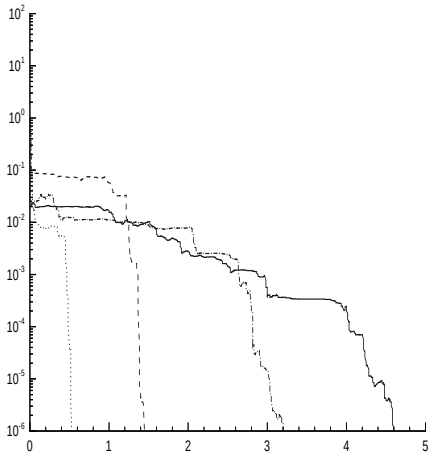
$n = 65 \times 65$

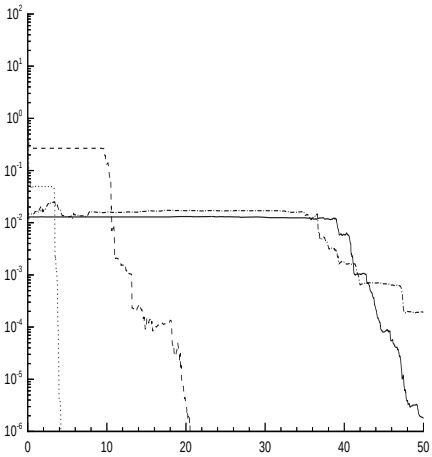
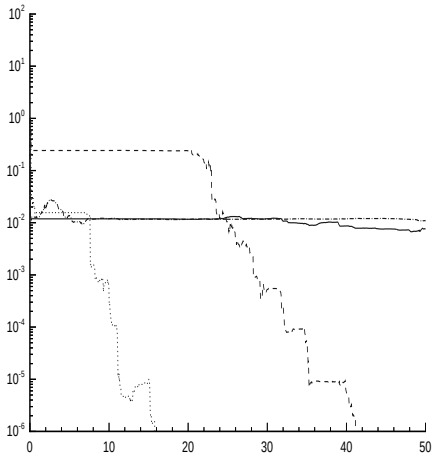
$n = 129 \times 129$

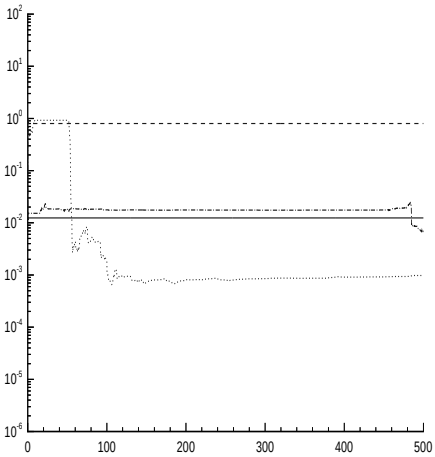
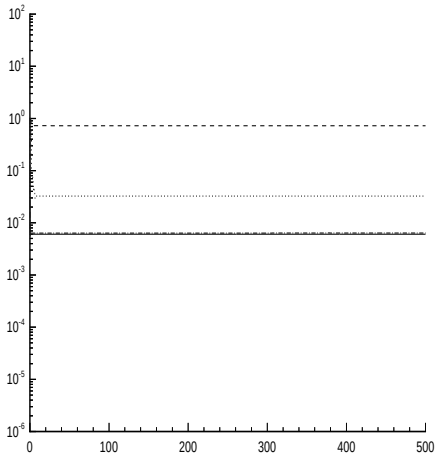
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

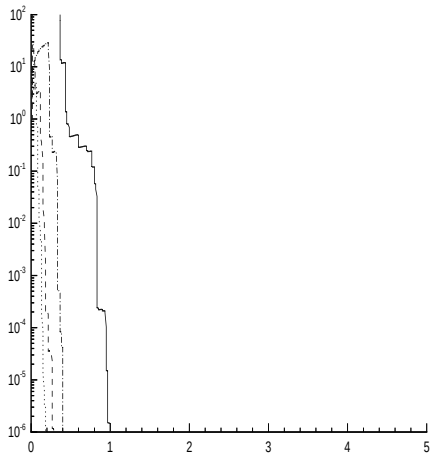
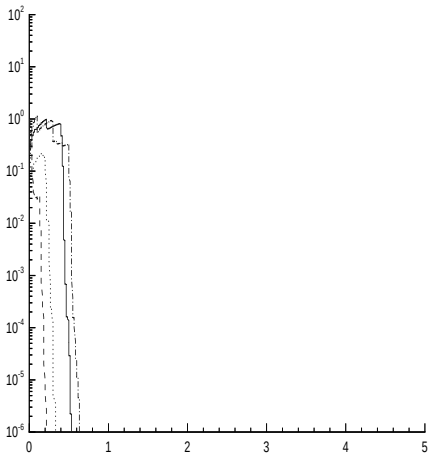
$n = 65 \times 65$

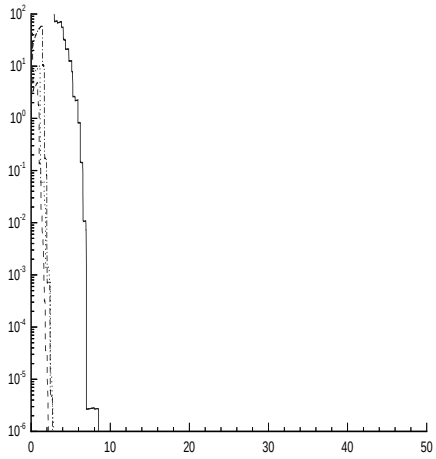
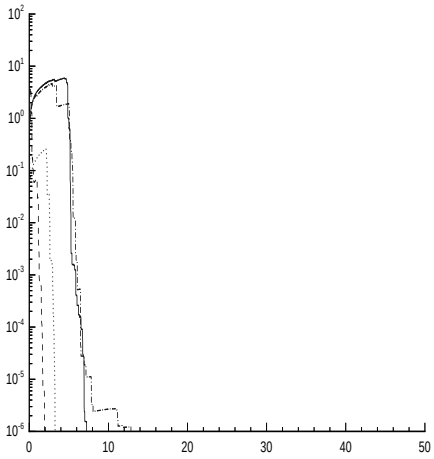
$n = 129 \times 129$

$\varepsilon = 10^0$ $\varepsilon = 10^{-2}$

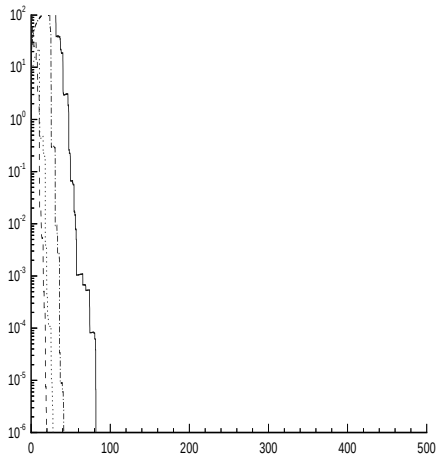
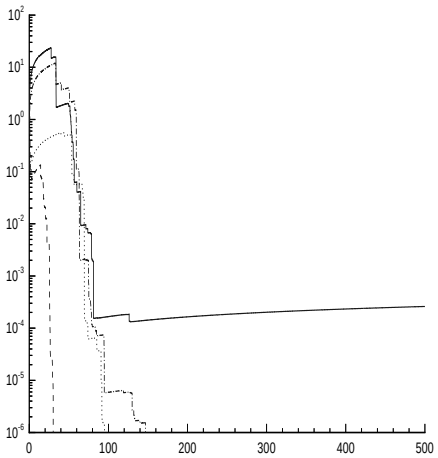
$n = 33 \times 33$

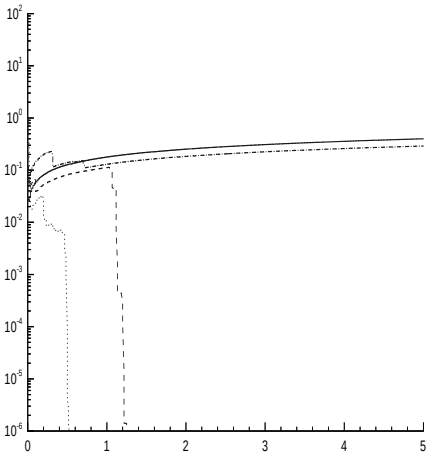
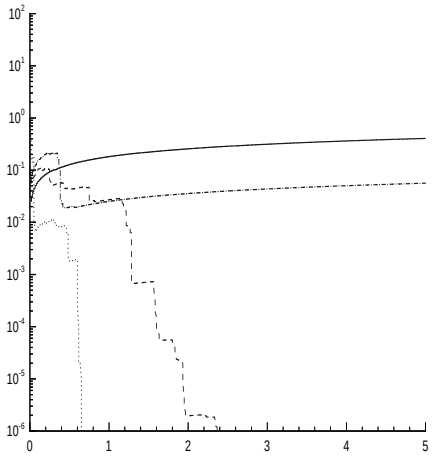
$n = 65 \times 65$

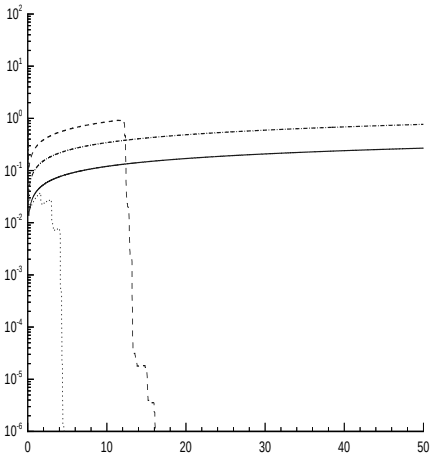
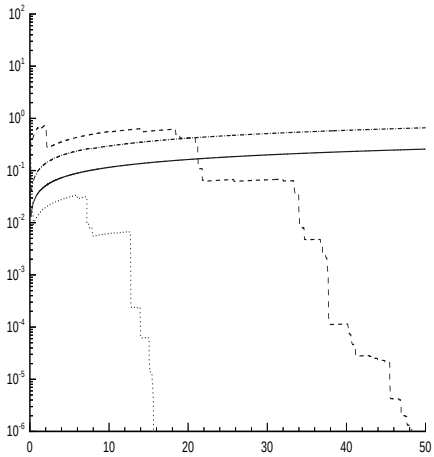
$n = 129 \times 129$

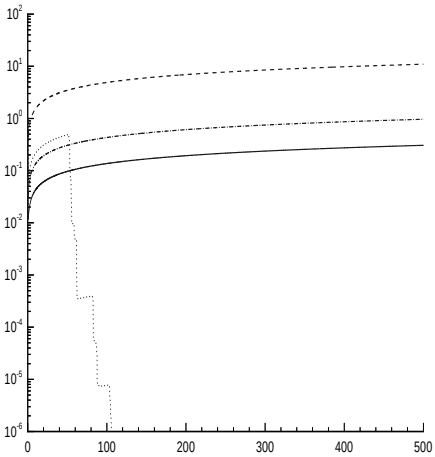
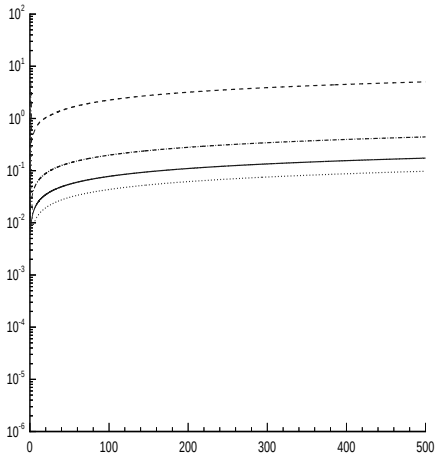
$\varepsilon = 10^{-4}$ $\varepsilon = 10^{-6}$

$n = 33 \times 33$

$n = 65 \times 65$

$n = 129 \times 129$

Literaturverzeichnis

[1] **M. Arioli, I. Duff, D. Ruiz**, *Stopping criteria for iterative solvers*, SIAM J. Matrix Anal. Appl. 13, 1992, pp.138–144

[2] **W. E. Arnoldi**, *The principle of minimized iteration in the solution of the matrix eigenvalue problem*, Quart. Appl. Math. 9, 1951, pp.17–29

[3] **O. Axelsson**, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994

[4] **R. E. Bank, T. F. Chan**, *An analysis of the composite step bi–conjugate gradient method*, Tech. Report CAM92, Dept. of Math., UCLA, Los Angeles, 1992

[5] **C. Bardos, D. Brezis, H. Brezis**, *Perturbations singulieres et prolongements maximaux d'operateurs positives*, Archive Rat. Mech. Anal. 53, 1973, pp.69–100

[6] **R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato,**

J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods*, SIAM, Philadelphia, 1993

[7] **D. Braess**, *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*, Springer-Verlag, Berlin, 1992

[8] **T. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, C. Tong**, *A quasi–minimal residual variant of the Bi–CGSTAB algorithm for nonsymmetric systems*, SIAM J. Sci. Stat. Comp. 15, 1994, pp.338–347

[9] **I. S. Duff**, *Sparse matrices and their uses*, Academic Press, London, 1981

[10] **I. S. Duff, A. M. Erisman, J. K. Reid**, *Direct methods for sparse matrices*, Oxford University Press, London, 1986

[11] **S. L. Eisenstat, H. C. Elman, M. H. Schultz**, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Num. Anal. 20, 1983, pp.345–357

[12] **H. C. Elman**, *Iterative methods for large sparse nonsymmetric systems of linear equations*, Ph.D. Dissertation, Comp. Sci. Dept., Yale Univ., New Haven, 1982

[13] X **V. Faber, T. Manteuffel**, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Num. Anal. 21, 1984, pp.352–362

[14] **A. Felgenhauer**, *Application of a generalized maximum principle to estimate corner layers in the n–dimensional case*, Report R-Mech. 03/84, Akademie der Wissenschaften, Berlin, 1984, pp.1–8

[15] **R. Fletcher**, *Conjugate gradient methods for indefinite systems*, Lecture Notes in Math. 506, Springer–Verlag, Berlin – New York, 1976, pp. 73–89

[16] **R. W. Freund**, *A transpose–free quasi–minimal residual algorithm for non–Hermitian linear systems*, SIAM J. Sci. Stat. Comp. 14, 1993, pp.470–482

[17] **R. W. Freund, G. H. Golub, N. Nachtigal**, *Iterative solution of linear systems*, Acta Numerica, 1992, pp. 57–100

[18] **R. W. Freund, M. Gutknecht, N. Nachtigal**, *An implementation of the look–ahead Lanczos algorithm for non–Hermitian matrices*, SIAM J. Sci. Comp. 14, 1993, pp.137–158

[19] **R. W. Freund, N. Nachtigal**, *QMR: A quasi–minimal residual method for non–Hermitian linear systems*, Num. Math. 60, 1991, pp.315–339

[20] **R. W. Freund, T. Szeto**, *A quasi–minimal residual squared algorithm for non–Hermitian linear systems*, Tech. Report, RIACS, NASA Ames, Ames, 1991

[21] **F. Gastaldi, L. Gastaldi, A. Quarteroni**, *Adaptive domain decomposition methods for advection dominated equations*, erscheint in: East–West J. Num. Math.

[22] **H. Goering, A. Felgenhauer, G. Lube, H.–G. Roos, L. Tobiska**, *Singularly perturbed differential equations*, Akademie–Verlag, Berlin, 1983

[23] **G. H. Golub, C. F. van Loan**, *Matrix computations, second edition*, The Johns Hopkins Univ. Press, Baltimore, 1989

[24] **W. Hackbusch**, *Iterative Lösung großer schwachbesetzter Gleichungssysteme*, Teubner, Stuttgart, 1991

[25] **W. Hackbusch**, *Multi-Grid methods and applications*, Springer–Verlag, Berlin, 1985

[26] **M. Hestenes, E. Stiefel**, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand. 49, 1952, pp.409–436

[27] **Y. Huang, H. A. van der Vorst**, *Some observations on the convergence behavior of GMRES*, Tech. Report 89–09, Fac. of Tech. Math and Inf., Delft Univ. of Tech., Delft, 1989

[28] **K. Jea, D. Young**, *Generalized conjugate–gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl. 34, 1980, pp.159–194

[29] **W. Kahan**, *Gauß-Seidel methods of solving large systems of linear equations*, Ph. D. Dissertation, Univ. of Toronto, 1958

[30] **C. Lanczos**, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Stand. 49, 1952, pp.33–53

[31] **J. L. Lions**, *Perturbations singulieres dans les problemes aux limite et en controle optimal*, Springer–Verlag, Berlin, 1973

[32] **A. Meister**, *Zur zeitgenauen numerischen Simulation reibungsbehafteter, kompressibler, turbulenter Strömungsfelder mit einer impliziten Finite–Volumen–Methode vom Box–Typ*, Dissertation & Forschungsbericht 96–08, Inst. für Strömungsmech., DLR, Göttingen, 1996

[33] **N. M. Nachtigal**, *A look–ahead variant of the Lanczos algorithm and its application to the quasi–minimal residual methods for non–Hermitian linear systems*, Ph.D. Dissertation, MIT, Cambridge, 1991

[34] **F. Nataf, F. Rogier**, *Factorization of the convection–diffusion operator and the Schwarz algorithm*, Math. Model. Meth. Appl. Sc. 5, 1995, pp.67–93

[35] **J. M. Ortega**, *Introduction to parallel and vector solution of linear systems*, Plenum Press, New York – London, 1988

[36] **O. Osterby, Z. Zlatev**, *Direct methods for sparse matrices*, Springer–Verlag, New York, 1983

[37] **O. A. Olejnik, R. U. Radkevic**, *Second order equations with nonnegative characteristic form*, Amer. Math. Soc., New York, 1973

[38] **C. C. Paige, M. A. Saunders**, *Solution of sparse indefinite systems of linear equations*, SIAM J. Num. Anal. 12, 1975, pp.617–629

[39] **B. N. Parlett, D. R. Taylor, Z. A. Liu**, *A look–ahead Lanczos algorithm for unsymmetric matrices*, Math.Comp. 44, 1985, pp.105–124

[40] **D. Peaceman, H. Rachford**, *The numerical solution of elliptic and parabolic differential equations*, Journal of SIAM 3, 1955, pp.28–41

[41] **C. Pommerell**, *Solution of large unsymmetric systems of linear equations*, Ph.D. Dissertation, Swiss Fed. Inst. of Tech., Zürich, 1992

[42] **A. Quarteroni**, *Mathematical aspects of domain decomposition methods*, Preprint

[43] **A. Quarteroni, A. Valli**, *Numerical approximation of partial differential equations*, Springer–Verlag, Berlin, Heidelberg, New York, 1994

[44] **G. Radicati di Brozolo, Y. Robert**, *Vector and parallel CG–like algorithms for sparse non–symmetric systems*, Tech. Report 681–M, IMAG/TIM3, Grenoble, 1987

[45] **H. G. Roos, M. Stynes, L. Tobiska**, *Numerical methods for singularly perturbed differential equations*, Springer–Verlag, Berlin, 1996

[46] **Y. Saad**, *Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals*, SIAM J. Num. Anal. 20, 1983, pp.784–811

[47] **Y. Saad**, *Iterative methods for sparse linear systems*, PWS Publishers, Cambridge, 1996

[48] **Y. Saad, M. Schultz**, *Conjugate gradient–like algorithms for solving nonsymmetric linear systems*, Math. Comp. 44, 1985, pp.417–424

[49] **Y. Saad, M. Schultz**, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp., 7, 1986, pp.856–869

[50] **G. L. G. Sleijpen, D. R. Fokkema**, *Bi–CGSTAB(l) for linear equations involving unsymmetric matrices with complex spectrum*, Tech. Report 772, Dept. of Math., Univ. of Utrecht, 1993

[51] **P. Sonneveld**, *CGS: A fast Lanczos–type solver for nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp. 10, 1989, pp.36–52

[52] **H. A. van der Vorst**, *Bi–CGSTAB: A fast and smoothly converging variant of Bi–CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp. 13, 1992, pp.631–644

[53] **H. A. van der Vorst, P. Sonneveld**, *CGSTAB: A more smoothly converging variant of CG–S*, Tech. Report 90–50, Fac. of Tech. Math., Delft Univ. of Tech., Delft, 1990

[54] **H. A. van der Vorst, C. Vuik**, *The rate of convergence of the GMRES method*, Preprint 654, Dept. of Math., Univ. of Utrecht, 1991

[55] **P. K. Vinsome**, *ORTHOMIN: An iterative method for solving sparse sets of simultaneous linear equations*, in Proceedings or the fourth SPE Symposium of Reservoir Simulation, Los Angeles, 1976, pp.149–159

Danksagung

Zum Schluß dieser Arbeit möchte ich allen danken, die mich während ihrer Fertigstellung und auf dem Weg dorthin unterstützt haben:
Ich danke Herrn Gert Lube für die intensive und motivierende Betreuung und für seine stetige Anteilnahme am Fortschritt meiner Arbeit. Diese hat mir dadurch wirklich Spaß gemacht. Dank an Andreas Auge für viele hilfreiche Anregungen und konstruktive Zusammenarbeit bei der Entwicklung von *BLANC* und an ihn und Frank–Christian Otto für die Unterstützung bei der Durchführung der Experimente.
Ganz herzlich danke ich meinen Eltern, daß sie mir das Studium und die damit verbundenen Erfahrungen ermöglicht haben, und meiner ganzen Familie für den starken Rückhalt.
Aus meinem Göttinger „sozialen Umfeld“, das mich psychologisch, kulinarisch und geduldig umsorgt hat, möchte ich hier dankend Christian, Christoph, Fritz, Henning, Moni und Jodel hervorheben, die Korrektur gelesen haben, sowie Schwatz, die dies beinahe getan hätte.

Göttingen, den 19.9.1996 Andreas P. Priesnitz