

Gemischt genaue Methoden

Dominik Göddeke

Lehrstuhl für Angewandte Mathematik und Numerik
Fakultät für Mathematik
Technische Universität Dortmund
dominik.goeddeke@math.tu-dortmund.de

Vorlesung Wissenschaftliches Rechnen
Dortmund, 6. Juli 2010

Übersicht

- 1 Fließkommaformate und Fließkommagenauigkeit
- 2 Doppelte gegen einfache Genauigkeit
- 3 Gemischt-genaue iterative Verfeinerung
- 4 Gemischt-genaue iterative Verfeinerung für dünn besetzte Systeme

Fließkommaformate und Fließkommagenauigkeit

Formate

IEEE-754 Standard

Einfache Genauigkeit

- Format: s23e8 (sign, 23 bit Mantisse, 8 bit Exponent: 32 bit)
- Größter „counting integer“: 2^{24}
- Genauigkeit der Mantisse ca. 7–8 Nachkommastellen
- Eigentlich immer in Hardware verfügbar

Doppelte Genauigkeit

- Format: s52e11 (sign, 52 bit Mantisse, 11 bit Exponent: 64 bit)
- Größter „counting integer“: 2^{53}
- Genauigkeit der Mantisse ca. 15–16 Nachkommastellen
- Fast immer in Hardware verfügbar

Beispiel (S.M. Rump, 1988)

Polynomauswertung (Potenzen per Multiplikation)

$$f(x,y) = (333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

für $x_0 = 77617$ und $y_0 = 33096$ (exakt in s23e8 darstellbar) ergibt:

Beispiel (S.M. Rump, 1988)

Polynomauswertung (Potenzen per Multiplikation)

$$f(x,y) = (333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

für $x_0 = 77617$ und $y_0 = 33096$ (exakt in s23e8 darstellbar) ergibt:

einfach s23e8 1.1726...

Beispiel (S.M. Rump, 1988)

Polynomauswertung (Potenzen per Multiplikation)

$$f(x,y) = (333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

für $x_0 = 77617$ und $y_0 = 33096$ (exakt in s23e8 darstellbar) ergibt:

einfach s23e8	1.1726...
doppelt s52e11	1.17260394005318...

Beispiel (S.M. Rump, 1988)

Polynomauswertung (Potenzen per Multiplikation)

$$f(x,y) = (333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

für $x_0 = 77617$ und $y_0 = 33096$ (exakt in s23e8 darstellbar) ergibt:

einfach s23e8	1.1726...
doppelt s52e11	1.17260394005318...
quad s112e15	1.172603940053178631...

Beispiel (S.M. Rump, 1988)

Polynomauswertung (Potenzen per Multiplikation)

$$f(x,y) = (333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

für $x_0 = 77617$ und $y_0 = 33096$ (exakt in s23e8 darstellbar) ergibt:

einfach s23e8	1.1726...
doppelt s52e11	1.17260394005318...
quad s112e15	1.172603940053178631...

Noch nicht einmal das Vorzeichen stimmt!

Exaktes Ergebnis $-0.8273\dots$

Computational Precision \neq Result Accuracy

Operationen

Multiplikation $a \cdot b$

- $s_a \cdot s_b, \quad m_a \cdot m_b, \quad e_a + e_b$
- Exaktes Ergebnis: $s23e8 \cdot s23e8 = s46e9$
- Dominante Fehlerquelle: Mantisse abgeschnitten von 46 auf 23 bit

Addition $a + b$

- $e_{\text{diff}} = e_a - e_b, \quad m_a + (m_b \gg e_{\text{diff}})$
- Exaktes Ergebnis: $s23e8 + s23e8 = s278e8$
- Dominante Fehlerquelle: Mantisse abgeschnitten von 278 auf 23 bit

Fehlerquellen

Darstellungsfehler (representation error)

- Runden von rationalen und irrationalen Zahlen wie π , $\sqrt{2}$, $\frac{1}{3}$
- Prinzipiell unvermeidbar

Abschneidefehler (roundoff error)

- $a = 1$ $b = 0.00000004$ $c = 0.9998$ $d = 1.0002$
- Alle in s23e8 exakt repräsentierbar
- Berechne $a + b$ und $c \cdot d$
- $1.00000004 = 1 + 0.00000004 =_{\text{fl}} 1$
- $0.99999996 = 0.9998 \cdot 1.0002 =_{\text{fl}} 1$

Konsequenz: Fließkomma-Arithmetik ist nicht kommutativ!

- $1 + 0.00000004 - 1 =_{\text{fl}} 0$
- $1 - 1 + 0.00000004 =_{\text{fl}} 0.00000004$

Fehlerquellen

Auslöschung (cancellation error)

- Nimm entweder Ergebnis von $a + b$ oder von $c \cdot d$, subtrahiere 1 und multipliziere mit 10^8
- $4 = (1.00000004 - 1) \cdot 10^8 =_{\text{fl}} 0$
- $-4 = (0.99999996 - 1) \cdot 10^8 =_{\text{fl}} 0$
- Auslöschung verstärkt den kleinen Abschneidefehler 0.00000004 zum absoluten Fehler 4 und zu einem relativen Fehler der Ordnung 1

Rump's Beispiel

- Ausmultiplizieren und so zusammenfassen:
- $5.5 \cdot 77617^8 - 2 - 5.5 \cdot 77617^8 + \frac{33096}{2 \cdot 77617}$
- Sehr viel transparenter, man sieht sofort, dass Auslöschung hier den Fehler verursacht ($5.5 \cdot 77617^8$ hat 40 Stellen, 2 nur eine)
- Auch klar: $f(x, y) = \frac{x}{2y} - 2$ geht dann ohne jede Auslöschung

Moral

Mehr Rechengenauigkeit erhöht in der Regel die Ergebnisgenauigkeit

- Rump's Beispiel ist bewusst didaktisch konstruiert um diese heuristische Annahme zu widerlegen
- Abschneidefehler und Darstellungsfehler treten acht Größenordnungen später auf beim Übergang von single auf double
- Praxis: Wenn ich sowieso nur „auf 1% Genauigkeit“ lösen will, wird mir double schon reichen
- Praxis: 99% der Anwender verwenden deshalb doppelte Genauigkeit und sind damit glücklich
- Double ist das schnellste native Format auf CPUs

Das ist nur die halbe Wahrheit

- Fehler können sich schnell aufschaukeln
- Nicht für alle Probleme reicht doppelte Genauigkeit (als das Maximum von dem, was effizient in Hardware unterstützt wird)
- Probleme sind schlecht konditioniert (gleich)

Moral

Beispiele

- Skalarprodukte von langen Vektoren mit sehr stark variierenden Einträgen (und damit auch Matrix-Vektor-Operationen)
 - Ein „sicherer“ Algorithmus sortiert die Einträge zunächst nach ihren absoluten Werten um, so daß die anschließende Summation möglichst wenig falsch macht
 - Viel zu teuer in der Praxis
- Interaktion physikalischer Felder mit stark unterschiedlichen absoluten Werten
 - Astrophysik (Supernovae)
 - u.U. Pressure drop bei laminaren Strömungen um Hindernisse

Doppelte gegen einfache Genauigkeit

Vorteile einfacher Genauigkeit

Rechengeschwindigkeit

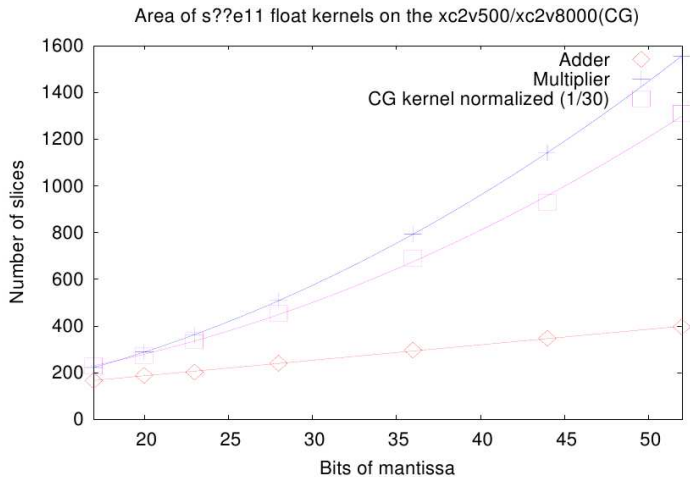
- Doppelt so schnelle Rechnung in einfacher wie in doppelter Genauigkeit in CPUs mit SSE
- Oft sogar mehr (GPUs, Cell-Prozessor etc.)

Rechenressourcen

- Größe eines Fließkomma-Addierers ist linear in der Bitlänge der Operanden
- Größe eines Fließkomma-Multiplizierers ist quadratisch in der Bitlänge der Operanden
- Multiplizierer dominieren (sind viel größer), also: 64 bit zu 32 bit bedeutet Vervierfachung der Ressourceneffizienz!
- (Auf CPUs nur 2x wegen diverser Hardware-Kompromisse)

Vorteile einfacher Genauigkeit

Beispiele auf einem FPGA



Vorteile einfacher Genauigkeit

Memory wall

- $64 \text{ bit} = 1 \text{ double} = 2 \text{ floats}$
- Also: Mehr Variablen pro Bandbreite (Datendurchsatz verdoppelt)
- Also: Mehr Variablen pro Speicherhierarchie-Ebene (effektive Nutzung schneller Caches verdoppelt)
- Gilt universell, für alle Hardwareplattformen (außer für FPGAs, aber das führt zu weit)

Idee: Benutze einfache Genauigkeit

- Um der memory wall Problematik entgegenzuwirken
- Um theoretisch doppelt so schnell rechnen zu können ohne den Programmieraufwand zu erhöhen

Konditionszahl und Störungssatz

Konditionszahl einer Matrix \mathbf{A}

- Konditionszahl $\text{cond}_2(\mathbf{A}) := \lambda_{\max}/\lambda_{\min}$
- (größter durch kleinster Eigenwert, deshalb auch *Spektralkondition*)

Störungssatz für LGS $\mathbf{Ax} = \mathbf{b}$

- $\delta\mathbf{A}$: Fehler (Störung) in der Matrix (Eingabedaten)
- $\delta\mathbf{b}$: Fehler (Störung) in der rechten Seite (Eingabedaten)
- $\frac{\|\mathbf{x} + \delta\mathbf{x}\|}{\|\mathbf{x}\|}$: Relativer Fehler in der Lösung
- Dann gilt:

$$\text{cond}_2(\mathbf{A}) \sim 10^s; \frac{\|\mathbf{A} + \delta\mathbf{A}\|}{\|\mathbf{A}\|}, \frac{\|\mathbf{b} + \delta\mathbf{b}\|}{\|\mathbf{b}\|} \sim 10^{-k} (k > s) \quad \Rightarrow \quad \frac{\|\mathbf{x} + \delta\mathbf{x}\|}{\|\mathbf{x}\|} \sim 10^{s-k}$$

Anwendung auf den Wechsel von doppelter zu einfacher Genauigkeit (DP \rightarrow SP)

- Abschneidefehler in der 7.-8. Nachkommastelle verstärkt um s Stellen

Konditionszahl und Störungssatz

Konditionszahl bei PDE-Diskretisierungen auf Gittern

- Allgemein: Kondition hängt von der Gitterweite h ab
- Genauer: Von der minimalen Gitterweite bei anisotropen Gittern
- Beispiel: Man kann zeigen: Für elliptische PDEs zweiter Ordnung ist $\text{cond}_2(\mathbf{A}) \sim h_{\min}^{-2}$, unabhängig von der Ordnung der Finiten Elemente (oder Differenzen) oder der Anzahl der Raumdimensionen
- Beweis: in jedem guten Lehrbuch, bspw. O. Axelsson und A. Barker: *Finite Element Solution of Boundary Value Problems. Theory and Applications*, Academic Press (Kapitel 5.5)
- Das ist eine gute Gelegenheit, Werbung für mein Lieblingsbuch zu machen: „*Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*“ von R. Barrett et al., http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html

Numerisches Beispiel

Poisson-Problem auf Einheitsquadrat

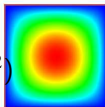
- $-\Delta u = f$ auf $\Omega = [0, 1]^2$
- Homogene Dirichlet-Randbedingungen: $u = 0$ auf Γ
- $\text{cond}_2(\mathbf{A}) \approx 2 \cdot 10^5$ für 1024×1024 Elemente (Verfeinerungsstufe $L = 10$) im isotropen Fall

Testfall und Fehlermessung

- Analytische Testfunktion $v(x, y) = x(1-x)y(1-y)$
- Definiere $f := -\Delta v$ in den Gitterpunkten (analytische Differenzierung und Auswertung)
- Messe den (relativen) Fehler in der L_2 -Norm

$$\frac{\sqrt{\int_{\Omega} (v-u)^2 d\mathbf{x}}}{\sqrt{\int_{\Omega} (v)^2 d\mathbf{x}}}$$

- Bilineare konforme Elemente: Pro Verfeinerung reduziert sich die L_2 -Norm des Fehlers um Faktor 4 (da Approximationsordnung h^2)



Numerisches Beispiel

Legende

- Level L bedeutet $N = (2^L + 1)^2$ Gitterpunkte
- Wir erwarten Fehlerreduktion um Faktor 4
- Relative L_2 -Fehler

Level	Daten+Rechnung in DP		Daten in SP, Rechnung in DP		Daten+Rechnung in SP	
	L_2 Fehler	Red.	L_2 Fehler	Red.	L_2 Fehler	Red.
5	1.1102363E-3	4.00	1.1102371E-3	4.00	1.1111655E-3	4.00
6	2.7752805E-4	4.00	2.7756739E-4	4.00	2.8704684E-4	3.87
7	6.9380072E-5	4.00	6.9419428E-5	4.00	1.2881795E-4	2.23
8	1.7344901E-5	4.00	1.7384278E-5	3.99	4.2133101E-4	0.31
9	4.3362353E-6	4.00	4.3757082E-6	3.97	2.1034461E-3	0.20
10	1.0841285E-6	4.00	1.1239630E-6	3.89	8.8208778E-3	0.24

Numerisches Beispiel

Beobachtungen zusammengefasst

- Datenfehler in der 7. Nachkommastelle verstärkt auf 2. Nachkommastelle
- Einfache Genauigkeit für relevante Problemgrößen nicht ausreichend
- Verfeinerung um Faktor 4 (ein Level, 4-facher Rechenaufwand) liefert sogar ein schlechteres Ergebnis!

Wichtige Bemerkungen

- Problem hier einfach feststellbar: Fehler gegen analytisch bekannte Referenzlösung
- In der Praxis gibt es keine analytische Referenz, deshalb: Löser liefert „ein“ Ergebnis, und wir haben keine Möglichkeit dessen Qualität zu bewerten

Moral

Generelles Problem

- Beispiel: Extrem-sehr-weit-draußen-Numerik wie ein schlauer Fehlerschätzer oder Fehlerindikator, der den (rein mathematischen) Diskretisierungsfehler mittels FEM-Integration und wilden Galerkin-Projektionen abschätzt (Kontext Adaptivität), findet „diese Sorte Fehler“ nicht, wenn er auch nur in einfacher Genauigkeit arbeitet!
- Wir brauchen also eigentlich Verfahren, die auch Fließkomma-Fehler einbeziehen

Im Folgenden: Ein Verfahren

- dass die Genauigkeitsprobleme löst und gleichzeitig noch Geschwindigkeitsvorteile bietet (besser zur Hardware passt, Hardware-orientierte Numerik)
- dass theoretisch komplett untermauert ist (keine Heuristik)

Exemplarisch für die Lösung linearer Gleichungssysteme

Gemischt-genaue iterative Verfeinerung

Gemischt-genaue iterative Verfeinerung

Idee

- Allgemein: Durch „mehr Genauigkeit“ an strategisch günstigen Stellen der Rechnung die Qualität der Rechnung verbessern (engl. precision vs. accuracy)
- Geschwindigkeit: Ergebnissenauigkeit des doppelt genauen mit idealerweise der Geschwindigkeit des einfach genauen Formats

Basialgorithmus für LGS, Standardform (Tafel!)

1 Initialisierung:

$$k = 0, \quad \mathbf{x}^{(0)} = \mathbf{0}$$

2 Defekt in doppelter Genauigkeit:

$$\mathbf{d}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$$

3 Löse in einfacher Genauigkeit:

$$\mathbf{A}\mathbf{c}^{(k)} = \mathbf{d}^{(k)}$$

4 Aktualisiere in doppelter Genauigkeit:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{c}^{(k)}$$

5 Konvergenztest, sonst $k = k + 1$ und weiter mit Schritt 2

Bemerkungen

Alternative Namen in der Literatur

- Deutsch: Nachiteration, (gemischt-genaue) iterative Verbesserung, (gemischt-genaues) iteratives Verfeinern
- Englisch: (mixed precision) iterative refinement

Sehr allgemeines sehr mächtiges Verfahren

- Numerisch untersuchte und theoretisch untermauerte Erweiterungen für
 - Nichtlineare Löser
 - Vorkonditionierte Krylov-Unterraum-Verfahren wie GMRES
 - ...
- Im Folgenden: Konzentration auf lineare Systeme $\mathbf{Ax} = \mathbf{b}$

Konvergenz

Man kann zeigen

- Iterative Verfeinerung konvergiert gegen die doppelt genaue Lösung des Systems, d.h. gegen $\mathbf{x}_{64} := \mathbf{A}_{64}^{-1} \mathbf{b}_{64} \neq (\mathbf{A}^{-1} \mathbf{b})_{64}$

Grobe Skizze des Konvergenzbeweises

- Berechnete Lösung $\mathbf{x}^{(k^*)}$ nach Konvergenz ist die exakte Lösung von $(\mathbf{A} + \delta \mathbf{A}) \mathbf{x} = \mathbf{b}$, wobei $\delta \mathbf{A}$ von \mathbf{b} abhängt und gleichmäßig beschränkt ist
- Konkrete obere Schranke m von $\|\delta \mathbf{A}\|_2$ hängt von den Details des Fließkommasystems ab, insbesondere von den Rundungsmodi bei Akkumulationen
- Falls $m \|\mathbf{A}^{-1}\|_2 \geq 1$, dann ist \mathbf{A} „zu schlecht konditioniert“ in der niedrigen Genauigkeit. Dann kann $(\mathbf{A} + \delta \mathbf{A})$ singular sein (in niedriger Genauigkeit), d.h. die Invertierung zur Lösung des Hilfsproblems ist nicht möglich
- Falls andererseits $m \|\mathbf{A}^{-1}\|_2 = 2^{-p}$ für ein $p > 1$, dann gilt

$$\|\mathbf{x}_{64} - \mathbf{x}^{(k+1)}\|_2 \leq \frac{2^{-p}}{1 - 2^{-p}} \|\mathbf{x}_{64} - \mathbf{x}^{(k)}\|_2$$

und das Verfahren konvergiert zur doppelt genauen Lösung

Konvergenz

Quantitative Konvergenzaussage

- Zahl der Iterationen k^* bis zur Konvergenz ist beschränkt durch

$$\left\lceil \frac{\log_{10} \epsilon_{\text{high}}}{\log_{10} \epsilon_{\text{low}} - \log_{10} \text{cond}(\mathbf{A})} \right\rceil$$

- Praxis: $\log_{10} \epsilon_{\text{double}} \approx -16$ und $\log_{10} \epsilon_{\text{single}} \approx -8$
- Also reichen typischerweise 3–4 Iterationen, wenn \mathbf{A} nicht zu schlecht konditioniert ist

Bemerkungen

Details des Beweises

- Beweistechniken erfordern einige Einarbeitung (technisch frickelige Abschätzungen, Vorwärts- und Rückwärts-Fehleranalyse)
- Bei Interesse: Referenzen [115], [203] (Lehrbücher) und [248], [60], [142] sowie Kapitel 4.3 meiner Diss

Verhältnis zwischen den Genauigkeiten

- In der allgemeinen Theorie egal solange die Fehlerberechnung genauer ist (sonst keine Konvergenz, da jedes $\mathbf{d}^{(k)} = \mathbf{0}$)
- double und single sind in gängiger Hardware unterstützt
- Quad-double wird oft verwendet, wenn double nicht ausreicht, da quad in HW idR nicht unterstützt wird und teuer in Software emuliert werden muss

Bemerkungen

Details des Fließkommaformats

- Führt hier zu weit, nur Ideen
- Rundung bei Darstellungsfehlern, insbesondere von Zwischenergebnissen ($s23e8 * s23e8 = s46e9 \Rightarrow s23e8$) (normal, immer abrunden etc.)
- Behandlung von *denormalised numbers*, d.h. Werte die eigentlich Null wären in Fließkomma
- Mathematische Handhabung ist sehr technisch

Rechenaufwand pro Schritt

In doppelter Genauigkeit

- eine Matrix-Vektor Multiplikation
- zwei Vektor-Vektor-Operationen
- eine Normberechnung zur Konvergenzkontrolle

In einfacher Genauigkeit

- Lösung eines LGS
- mit jeweils verschiedener rechter Seite aber gleicher Systemmatrix

Laufzeitgewinn also, wenn die Zeit in einfacher Genauigkeit über alle Schritte die Lösung in doppelter Genauigkeit unterschreitet

Anwendung: A dicht besetzt

Konkreter Algorithmus

- Lösung mit fester Matrix für verschiedene rechte Seiten:
LU-Zerlegung
- Berechne $\mathbf{PA} = \mathbf{LU}$ in einfacher Genauigkeit
- Löse $\mathbf{Ly} = \mathbf{Pd}$ und $\mathbf{Uc} = \mathbf{y}$ in einfacher Genauigkeit

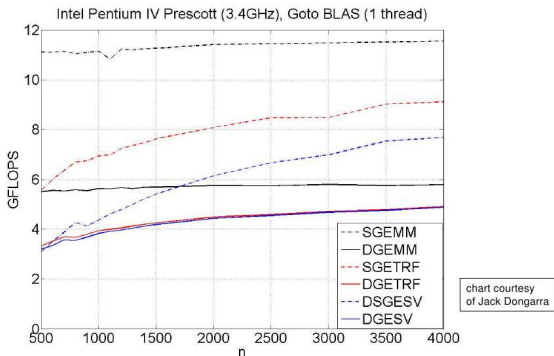
Grund für Speedup

- LU-Zerlegung dominiert mit $\mathcal{O}(N^3)$ die Rechenzeit, ist compute-bound und deshalb in einfacher Genauigkeit mindestens doppelt so schnell
- Alle anderen Operationen (in single und double) sind $\mathcal{O}(N^2)$ und $\mathcal{O}(N)$ und deshalb „vernachlässigbar“

Anwendung: A dicht besetzt

Beispiel

- SGEMM/DGEMM: Lapack dense MatVec in single und double
- SGETRF/DGETRF: LU-Zerlegung berechnen
- DSGESV/DGESV: Lösen mit LU in mixed und double



Zusammenfassung

Gemischt-genaueres iteratives Verfeinern

- Universelles Mittel zur Sicherstellung der Genauigkeit berechneter Lösungen
- Konvergenz gegen doppelte Genauigkeit selbst wenn die große Mehrheit der Operationen in einfacher Genauigkeit durchgeführt wird
- Konvergenz wenn Systemmatrix nicht zu schlecht konditioniert ist
- Für dicht besetzte Systeme immer Speedup
 - Zahl der Iterationen hängt nicht von N ab
 - (Kondition kann zwar schnell von N abhängen, s. sog. Hilbertmatrizen, aber das geht logarithmisch in die Abschätzung ein, d.h. es passt wenn die Kondition nicht exponentiell mit N steigt)
 - Eine Größenordnung mehr Operationen in einfacher als in doppelter Genauigkeit
- Preis: 50% mehr Hauptspeicher nötig

Gemischt-genaue iterative
Verfeinerung für dünn besetzte
Systeme

Herausforderungen

Theorie

- Konvergenzbeweise für Nachiterationen basieren auf der „exakten“ Lösung der Hilfsprobleme
- Direkte Löser konvergieren immer, falls die Matrix nicht singulär ist in einfacher Genauigkeit (nur u.U. gegen eine sehr falsche Lösung)
- Dieser Fall wird von der Theorie abgedeckt
- Tabelle von vor 20 Folien basierte nebenbei auf einem direkten Löser

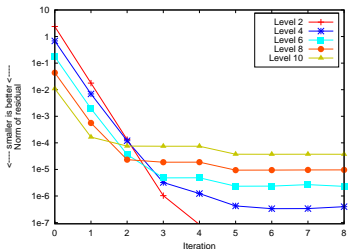
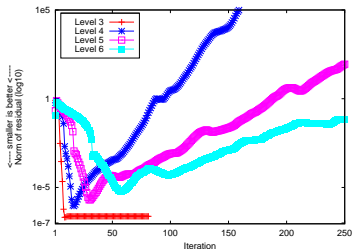
Praxis

- Direkte Löser für die Hilfsprobleme sind inakzeptabel teuer
- Selbst bei Verwendung von „sparse LU“ Techniken wie in den Bibliotheken UMFPACK und SUPERLU, die sonst bspw. für die Lösung von *Grobgitterproblemen* perfekte Dienste leisten
- Wir müssen also iterative Löser verwenden

Problem

Iterative Löser konvergieren nicht immer bei zu geringer Rechengenauigkeit

- Beispiele für CG (links) und MG (rechts) für das Problem aus der Tabelle



Strategie

Nachiteration wird also zu einem kaskadierten iterativen Verfahren

- Äußere Schleife in doppelter Genauigkeit (Korrektur zur Konvergenz)
- Innere Schleife in einfacher Genauigkeit (iterative Lösung der Hilfsprobleme)

Abbruchbedingung der inneren Schleife

- Heuristik muss sicherstellen, dass die innere Schleife rechtzeitig abbricht bevor „Unfug“ passiert
- Sehr abhängig vom tatsächlichen inneren iterativen Verfahren

Theorie

Interpretation der Methode

- Nachiteration ist (gemischt-genaue) vorkonditionierte Defektkorrektur

$$\mathbf{x}_{\text{DP}}^{(k+1)} = \mathbf{x}_{\text{DP}}^{(k)} + \mathbf{C}_{\text{SP}}^{-1}(\mathbf{b}_{\text{DP}} - \mathbf{A}_{\text{DP}}\mathbf{x}_{\text{DP}}^{(k)})$$

- Einfach genauer Vorkonditionierer \mathbf{C}_{SP} kapselt die approximative Lösung (je nach Abbruchbedingung) des iterativen Verfahrens für das Hilfsproblem
- Implementierung: Eigentlich ändert sich nichts, auch wenn diese Schreibweise kompakter ist

Erinnerung: Vorkonditionierung

- Einfach zu invertierender Vorkonditionierer \mathbf{C} ist „gut“, wenn $\text{cond}(\mathbf{C}^{-1}\mathbf{A}) \ll \text{cond}(\mathbf{A})$ (und wenn der VK schöne Eigenschaften wie M-Matrix hat)
- Grund: $\text{cond}(\mathbf{A}^{-1}\mathbf{A}) = \text{cond}(\mathbf{I}) = 1$

Theorie

Überlegungen zur Konvergenz des Verfahrens

- Nachiteration ist (gemischt-genaue) vorkonditionierte Defektkorrektur
- Konvergenz hängt also nur ab von der Qualität des Vorkonditionierers
- Qualitativ: Wenn die Anwendung von \mathbf{C}_{SP} in einer Reduktion in der doppelt genauen Residuenorm resultiert konvergiert das Verfahren
- Quantitativ: Fehlerreduktion durch \mathbf{C}_{SP} definiert die Konvergenzrate
- Man beachte den analogen Einfluss der Kondition in dieser und in der allgemeinen Theorie

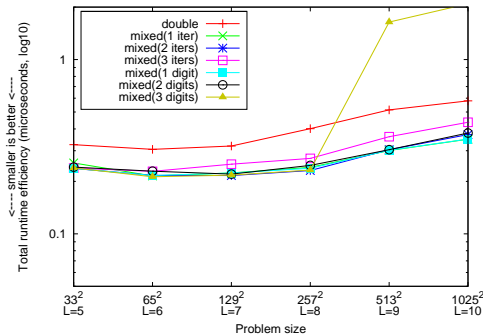
Unter der Voraussetzung eines optimalen Abbruchkriteriums

- Konvergenz sichergestellt wenn das System nicht zu schlecht konditioniert ist für die Anwendung von \mathbf{C}_{SP}
- Konvergenzrate ist proportional zur Kondition von \mathbf{C}_{SP}

Heuristische Abbruchkriterien

- Meine Erfahrung: keine Probleme mit „gewinne eine Stelle“ (für die Probleme (Poisson, anisotrope Diffusion, Strukturmechanik, Stokes, laminare Strömungssimulation) die ich bisher so betrachtet habe)
- Selbst für Probleme mit $\text{cond}(\mathbf{A}) \approx 10^{19}$, also mehr als grenzwertig für doppelte Genauigkeit
- Wenn Mehrgitter für die Anwendung von \mathbf{C}_{SP} verwendet wird
- Krylov-Unterraum-Verfahren leiden unter der restart-Problematik: In jedem Korrekturschritt geht der bisher aufgebaute Suchraum \mathbf{A} -orthogonaler Richtungen verloren
- Lösbares Problem, führt aber hier zu weit (s. meine Diss, Kap. 4.5, und Referenz [211])

Numerische Ergebnisse



- Poisson-Problem, Testfall aus der Tabelle von oben
- Mehrgitter-Löser mit ADI-TRIDI Glätter
- Abbruchkriterium variabel
- Totale Effizienz (Zeit pro Stelle Gewinn pro Unbekannte)
- 1.7-1.9-fach schneller als reine Rechnung in doppelter Genauigkeit ohne Verlust der Genauigkeit

Zusammenfassung

Zusammenfassung

- 1 Fließkommaformate und Fließkommagenauigkeit
- 2 Doppelte gegen einfache Genauigkeit
- 3 Gemischt-genaue iterative Verfeinerung
- 4 Gemischt-genaue iterative Verfeinerung für dünn besetzte Systeme