Lösung linearer Gleichungssysteme

direkte/iterative Verfahren, schwach besetze Matrizen, Software-Pakete

Kittel Matthias
Institut für Astronomie
Universität Wien
Türkenschanzstraße 17
1180 Wien
kittel@astro.univie.ac.at
14. Mai 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Lineare Gleichungssysteme

1.1 Einführung

Trotz der Tatsache, dass fast alle realen Zusammenhänge in der Astronomie und der Physik *nicht*linear sind, sind Reduktionen auf einfachere lineare Modelle gang und gäbe. Viele mathematische Untersuchungs- und Lösungsmethoden, exakt oder iterativ, sind für lineare Modelle besser entwickelt, deshalb setzt man diese oft ein, auch wenn sie sich wesentlich vom ursprünglichen nichtlinearen Modell unterscheiden. Die numerische Lösung nichtlinearer Probleme wird in fast allen Fällen auf die Lösung von linearen Gleichungssystemen zurückgeführt. Diese Tatsache erklärt die zentrale Rolle, die den Lösungsverfahren linearer Gleichungssysteme in der Numerik zugewiesen wird. Um ein lineares Gleichungssystem mittels Software lösen zu können, bringt man es (fast) immer in die Standardform $Ax = b$

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ f_2(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots \\ a_{n1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Lineare Gleichungssysteme sind *angenehme* numerische Probleme, da die Daten des Problems in algebraischer Form vorliegen, das heißt in Matrizen und Vektoren. Diese Daten sind natürlich durch das Problem gegeben, und die Eigenschaften des Problems können zu speziellen Eigenschaften der Matrizen führen, die durch angepasste Lösungsverfahren schneller oder einfacher gelöst werden können.

Die Anzahl der arithmetischen Operationen bei der Lösung von großen linearen Gleichungssystemen ist sehr groß. Für ein vollbesetztes System mit 1200 Gleichungen ist ein Arbeitsaufwand von mehr als einem Gigaflop/s erforderlich. Auf grund dieser enormen Anzahl von Rechenoperationen stellt sich automatisch die Frage nach der Lösungsgenauigkeit, Robustheit und Effizienz der verwendeten Verfahren.

Um nun das gegebene Gleichungssystem effizient, schnell und problembezogen lösen zu können, muss man die Wahl zwischen zwei Klassen von Algorithmen treffen:

- **Direkte Verfahren**, die auf einer Faktorisierung der Systemmatrix beruhen und mit einer endlichen Anzahl von arithmetischen Operationen die (exakte) Lösung liefern.
- **Iterative Verfahren**, die auf der iterativen Fixpunktbestimmung von Gleichungssystemen beruhen und in den meisten Fällen einen unendlichen Prozess zur (exakten) Lösung benötigen.

1.2 Matrixnormen

Um für Konditionsberechnungenen den Grad der *Nachbarschaft* von Matrizen ausdrücken zu können, gibt es die zur Vektornorm äquivalente Matrixnorm. Diese Abbildung
$\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$, *die für alle* $A, B \in \mathbb{R}^{n\times n}$ *den Bedingungen*

1. $\|A\| \geq 0$,
2. $\|A\| = 0 \iff A = 0$ *(Definitheit)*
3. $\|\alpha A\| = \|\alpha\| \cdot \|A\|, \quad \alpha \in \mathbb{R}$ *(Homogenität)* und
4. $\|A + B\| \leq \|A\| + \|B\|$ *(Dreiecksungleichung)*

genügt, heißt Matrixnorm.
$\|\cdot\|_1$ *wird als Spaltensummen-,* $\|\cdot\|_\infty$ *als Zeilensummennorm und* $\|\cdot\|_2$ *als Euklidische Norm bezeichnet.*
All diese Normen sind äquivalent und untereinander durch konstante Relationen verbunden.

1.3 spezielle Matrixeigenschaften

Wenn man bei der numerischen Lösung linearer Gleichungssysteme spezielle Eigenschaften der Matrizen berücksichtigt, kann man sowohl die Zuverlässigkeit als auch die Effizienz der Lösungsalgorithmen erhöhen.

Die *transponierte Matrix* A^T ensteht durch die Spiegelung von A an der Hauptdiagonalen:

$$C = A^T \quad \rightarrow \quad c_{ij} = a_{ji} \quad \textit{für alle } i, j \in \{1, 2, \ldots, n\}$$

Eine *symmetrische Matrix* stimmt mit ihrer Transponierten überein, das heißt:

$$A^T = A$$

Die Eigenschaft, dass die *konjugierte Transposition* A^H *einer komplexen Matrix*

$$C = A^H \quad \rightarrow \quad c_{ij} = \overline{a}_{ji} \quad \textit{für alle } i, j, \in \{1, 2, \ldots .n\}$$

gleich der Ausgangsmatrix ist, nennt man *Hermitizität.*
Eine *orthogonale Matrix* ist eine quadratische Matrix mit folgender Eigenschaft

$$Q^T Q = QQ^T = I \quad \textit{mit } Q^T = Q^{-1}.$$

Bei komplexen Matrizen heißt diese Eigenschaft *unitär.*
Ein besonderer Typ von reellen symmetrischen/komplexen hermitischen Matrizen sind positiv definite Matrizen, die in vielen Anwendungen auftreten.

1.4 speziell besetzte Matrizen

Für die Auswahl des *richtigen* Lösungsverfahrens spielen Matrizen mit spezieller Besetzungsstruktur eine wichtige Rolle. Besonderes Augenmerk gilt hier den verschwindenden Einträgen a_{ij}.

- **Diagonalmatrizen**, sind n×n-Matrizen

$$\mathbf{D} := \begin{pmatrix} d_{11} & & & \mathbf{0} \\ & d_{22} & & \\ & & \ddots & \\ \mathbf{0} & & & d_{nn} \end{pmatrix}$$

  mit $d_{ij} = 0$ für $i \neq j$.

- **Dreiecksmatrizen**:
  Eine n×n-Matrix $T = t_{ij}$ wird *obere Dreiecksmatrix* genannt, falls $t_{ij} = 0$ für alle $j < i$ gilt:

$$\mathbf{T} := \begin{pmatrix} t_{11} & t_{12} & \dots & t_{1n} \\ & t_{22} & \dots & t_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & t_{nn} \end{pmatrix}$$

- **Tridiagonale Matrizen**:
  Eine n×n-Matrix $A = a_{ij}$ wird *Tridiagonale Matrix* genannt, falls $a_{ij} = 0$ für alle $\mid j - i \mid > 1$ gilt, das heißt nur die Haut- und die beiden Nebendiagonalen besetzt sind:

$$\mathbf{A} := \begin{pmatrix} a_{11} & a_{12} & & & \mathbf{0} \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & \ddots & \ddots & \\ & & \ddots & a_{n-1,n-1} & a_{n-1,n} \\ \mathbf{0} & & & a_{n,n-1} & a_{nn} \end{pmatrix}$$

- **Bandmatrizen**:
  Eine n×n-Matrix $A = a_{ij}$ wird *Bandmatrix* genannt, falls nur die Haupdiagonale und einige Nebendiagonalen zu Null verschiedene Einträge haben (z.B. Pentadiagonale Matrix).

- **Blockmatrizen**:
  Eine Matrix nennt man Blockmatrix, wenn sich die Einträge in der Matrix selbst wieder zu sogennanten *Untermatrizen* zerlegen lassen. Mit dieser Methode kann man auf Lösungsmethoden zugreifen, die für oben genannten Matrizen verwendet werden.

1.5 Kondition

Die Kondition eines linearen Gleichungssystems ist am einfachsten im zweidimensionalen Fall darstellbar. Die Lösung von

$a_{11}x + a_{12}y = b_1$ *(Gerade g_1)*

$a_{21}x + a_{22}y = b_2 \quad$ *(Gerade g_2)*

ist der Schnittpunkt S der beiden Geraden in der x-y-Ebene. Stört man nun diese Geraden verschiebt sich der Schnittpunkt T der gestörten Geraden f_1 und f_2 bei gut konditionierten Problemen kaum. Je schleifender der Schnitt zwischen den Geraden, desto größer der Abstand von S und T. Diese anschauliche Konditionsbetrachtung lässt sich im groben auch auf den n-dimensionalen Fall übertragen. Wenn $Ax = b$ das zu lösende Gleichungssystem bestimmt, $\triangle Ax = \triangle b$ das gestörte System, dann ist *cond(A)* durch

$$cond(A) := \|A\| \cdot \|A^{-1}\|$$

definiert. Im allgemeinen gilt, je höher die Konditionszahl, desto schlechter konditioniert ist die Matrix. Der Wert hängt von der verwendeten Matrixnorm ab. Mittels der Konditionszahl und den Normenwerten von A und b läßt sich zum Beispiel der relative Fehler von x bestimmen.

1.6 Direkte Verfahren

Alle Verfahren zur numerischen Lösung von linearen Gleichungssystemen beruhen auf dem *klassischen* Gauß'schen Eliminationsverfahren, das auf der Vertauschung beziehungsweise Anpassung der Linearkombinationen innerhalb des Systems beruht. Durch Multiplikation von einzelnen (oder allen) Gleichungen und geeigneter Pivotierung gelangt man zu einem Dreiecks-System, das sich durch *Rücksubstition* rekursiv lösen läßt. Auf diesem Gebiet existiert eine Vielzahl von verschiedenen Softwarepaketen, die zuverlässig und schnell lösen. Für viele Probleme existieren *Black-Box*-Programme, die sich für Spezialfälle auch leicht modifizieren und anpassen lassen. Um das Lösen von direkten Verfahren zu verbessern gibt es spezielle Pivotstrategien und Faktorisierungsmöglichkeiten. Besonders bei symmetrischen, positiv definiten Matrizen gibt es die Möglichkeit der *Cholesky-Zerlegung*. Der Rechenaufwand reduziert sich im Gegensatz zu herkömmlichen Verfahren um bis zu 50 Prozent. Zu signifikanten Effizienzsteigerungen führt die Berücksichtigung der Struktur, wenn A eine Bandmatrix ist. Da es eine große Anzahl von *symmetrisch angeordneten Nullen* gibt, empfiehlt sich eine besondere Speicherung wie bei iterativen Verfahren.

1.7 LAPACK und die BLAS

Das LAPACK (*Linear Algebra Package*) ist ein *public domain* Softwarepacket von Fortran 77-Programmen, die entwickelt wurden, um Standardprobleme der Linearen Algebra numerisch zu lösen. Das LAPACK wurde entwickelt, um lineare Gleichungssysteme, lineare Ausgleichs- und Eigenwertprobleme, Faktorisierungen von Matrizen, Singulärwertzerlegungen und Konditionsabschätzungen durchzuführen. Diese Programme wurden für dicht besetzte Matrizen und Bandmatrizen geschrieben, nicht aber für schwach besetzte Matrizen mit allgemeiner Besetzungsstruktur. Es sind sowohl *Black-Box-Programme*, als auch *Rechenprogramme* verfügbar. Das LAPACK ist über e-mail, ftp, oder internet `http://www.netlib.org/lapack` abrufbar. Eine wesentliche Besonderheit des Sofware-Packetes ist die Verwendung von Unterprogrammen zur Lösung elementarer Teilaufgaben, den sogenannten BLAS (*Basic Linear Algebra Subroutine*). Das LAPACK gliedert sich in drei Programmgruppen:

- **Black-box** oder **Treiberprogramme** gestalten die Lösung von Standardproblemen der Linearen Algebra so einfach wie möglich. Sie übernehmen den Aufruf der geeigneten Rechen- und Hilfsprogramme.
- **Rechenprogramme** führen bestimmte Verarbeitungsschritte wie Faktorisierungen oder Reduktionen durch.
- **Hilfsprogramme**, wie Normberechnungen, Blockalgorithmen oder die Erweiterungen zur BLAS.

2 Schwach besetzte Matrizen

Bei einer großen Anzahl astronomischer und physikalischer Modelle muss, um das dargestellte System genau zu beschreiben, eine sehr große Anzahl von Zustandsvariablen eingeführt werden. Dabei kommt es oft vor, dass diese Zustände *lokal beschränkt* sind und nur einige Zustände miteinander *wechselwirken*. Das mathematische Äquivalent liegt in einer *dünn oder schwachbesetzten Matrix (sparse matrix)* vor, ein System das über viele Gleichungen verfügt, in denen aber nur wenige Unbekannte vorkommen. Am stärksten ausgeprägt ist diese Form durch die Diskretisierung von Differentialgleichungen, wo es viele Stützstellen geben kann, die Differenzen aber immer nur die vorangegangene Stützstelle miteinbeziehen.

2.1 Speicherung

Es ist von Vorteil, wenn man solche schwach besetzten Matrizen in einer besonderen Form speichert, um das Lösungsverfahren zu beschleunigen, da großteils iterative Verfahren verwendet werden. Es werden nur die von Null verschiedenen Werte gespeichert, dafür muss aber Information über den *Platz* des Matrixeintrages mitgeliefert werden. Für Bandmatrizen kann man zu einer kompakten Speicherung auf ein rechteckiges Feld kommen, indem man die Diagonalen als Zeilen des Feldes aufzeichnet und die Spalten beibehält.

$$\begin{pmatrix} * & * & * & & \\ * & * & * & * & \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{pmatrix} \Rightarrow \begin{pmatrix} & & * & * & * \\ & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & \end{pmatrix}$$

Als Beispiele seien hier zwei einfache Speicherverfahren angegeben:

- **CCO-Format**:
  Es werden nur die Nicht-Null-Elemente (NNE) abgespeichert, und zwar in drei eindimensionalen Feldern *wert, zeilenindex, spaltenindex*, von der Länge der Anzahl der NNE. Hierbei handelt es sich um ein einfaches aber relativ ineffizientes Verfahren.

- **CRS-Format**:
  Bei dieser Speicherart handelt es sich um ein komprimiertes Zeilenspeicherformat, das wie das CCO-Format keinerlei Annahmen über die Struktur der schwach besetzten Matrix macht und nur das Minimum an Information speichert. Beim CRS (*compressed row storage*) werden *zeilenweise* aufeinanderfolgende NNE in benachbarte Elemente eines eindimensionales Feld gespeichert. Ein weiteres Feld *wert* enthält die Spaltenindizes dieser Elemente. In einem zusätzlichen Feld werden jene Positionen des Feldes *wert* gespeichert, bei denen eine neue Zeile der Matrix beginnt.

2.2 Iterative Verfahren

2.2.1 Fixpunkt

Iterationsverfahren werden wie folgt definiert:

$$\Phi(x_k) = x_{k+1}, \quad x_0 \ldots Startwert, \Phi \ldots Iterationsvorschrift$$

$$x_k \Rightarrow x_{k+1}, \quad Einzelschrittverfahren$$

$$\Phi(x_k, x_{k-1}) \Rightarrow x_{k+1}, \quad x_0, x_1 \ldots Startwerte \quad Mehrschrittverfahren$$

So ein Iterationsverfahren bleibt stehen, wenn $x_k = x_{k+1}$, das heißt wenn ein **Fixpunkt** x^* erreicht ist, für den gilt

$$\Phi(x^*) = x^*$$

Dieses Iterationsverfahren heißt *konvergent*, wenn es einen offenen Bereich $D \subseteq \mathbb{R}^n$ gibt, sodass $\forall x_0 \in D$ die Iterationsfolge gegen einen Fixpunkt x^* mit $\Phi(x^*) = x^*$ konvergiert. Für die Konvergenz dieser Fixpunktiteration muss nun folgende hinreichende Kontraktionsbedingung gelten, $\Phi : D \subset \mathbb{R}^n \to \mathbb{R}^n$ heißt kontrahierend auf $D_0 \subset D$, falls dort eine Lipschitz-Bedingung

$$\|\Phi(x) - \Phi(y)\| \leq L\|x - y\|$$

für alle $x, y \in D_0$ mit einr Lipschitz-Konstante (Kontraktionskonstante) $0 \leq L \leq 1$ erfüllt. Der dazugehörige Kontraktionssatz lautet: Ist $\Phi : D \subset \mathbb{R}^n \to \mathbb{R}^n$ eine kontrahierende Abbildung auf der abgeschlossenen Menge $D_0 \subset D$ und gilt $\Phi \cdot D_0 \subset D_0$ dann folgt

- Φ hat in D_0 genau einen Fixpunkt x^* und
- $x_{k+1} = \Phi(x_k)$ konvergiert für jeden Startwert $x_0 \in D_0$ gegen x^*.

Konvergente Verfahren sind in der Regel stabil, es kann aber vorkommen, dass durch eine Iteration die Werte aus dem *Einzugsbereich* herauslaufen. Ist nun die Iteration auf einen Fixpunkt gewährleistet, kommt es natürlich darauf an wie schnell der Fixpunkt erreicht wird.
Sei x^* Fixpunkt der Iterationsvorschrift Φ, und es gelte für alle Startwerte

in einer Umgebung des Startwertes und für alle Iterationsfolgen $\{x_k\}$,

$$\|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^p,$$

$p \geq 1$, wobei für $p = 1$ gelten muss $C < 1$. Dann heißt das durch Φ erzeugt Verfahren *konvergent von p-ter Ordung.*

2.2.2 Verfahrensübersicht

Es steht nun eine große Anzahl von Iterationsverfahren zu Verfügung, die sich grob in zwei Kategorien unterteilen lassen, stationäre und nichtstationäre Verfahren. Erstere werden dadurch charakterisiert, dass die Iterationsvorschrift unabhängig vom aktuellen Iterationsschritt definiert ist, bei zweiteren wird die Iterationsvorschrift bei jeder Iteration geändert. Zur ersten Klasse gehören das

- **Jacobi-Verfahren**: Ein Iterationsschritt des Jacobi- oder Gesamtschrittverfahrens entspricht der lokalen Lösung für eine Variable. Diese Methode ist leicht implementier- und anwendbar, konvergiert aber oft nur sehr langsam. Löst man unter der Annahme, dass die Werte $x_1, \ldots, x_{x-1}, x_{x+1}, \ldots, x_n$ gegeben sind, nur die i-te Gleichung des Systems $Ax = b$ nach x_i, der i-ten Komponente von x auf, so erhält man die Iterationsvorschrift

  $$x_i^{(k)} = (b^i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)})/a_{ii}$$

  für $a_{ii} \neq 0$. Außer bei starker Diagonaldominanz von A ist das Verfahren nur als vorkonditioniertes Verfahren für nichtstationäre Methoden zu empfehlen, die Parallelisierung ist aber einfach durchzuführen.

- **Gauß-Seidel-Verfahren**: Im Unterschied zum Jacobi-Verfahren werden beim Gauß-Seidel- oder Einzelschrittverfahren die erhaltenen Näherungswerte sofort nach ihrer Berechnung weiterverwendet. Die Konvergenzgeschwindigkeit erhöht sich dadurch, bleibt aber meist immer noch verhältnismäßig gering. Die Iterationsvorschrift lautet

  $$x_i^{(k)} = (b^i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)})/a_{ii}$$

Anwendungsgebiete sind strikt diagonaldominante oder symmetrische, positiv definite Matrizen.

- **SOR-Verfahren**: Dieses Verfahren wird aus dem Gauß-Seidel-Verfahren unter Einführung eines Extrapolationsoperators ω abgeleitet. Bei (fast) optimaler Wahl von ω kann eine wesentliche Konvergenzbeschleunigung erreicht werden. Man erhält folgende Iterationsvorschrift

$$x_i^{(k)} = \omega \cdot (b^i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)})/a_{ii} + (1-\omega) \cdot x_i^{(k-1)}$$

  Dieses Verfahren beschleunigt die oben genannten Verfahren und kann bei geeigneter Wahl von ω sogar Konvergenzverhalten erzwingen.

Zur zweiten Klasse der Iterationsverfahren gehören die sogenannten *Gradientenverfahren*, die ihre Iterationsvorschrift nach jedem Schritt auf Grund Überprüfung der Residuenvektoren verändern. Diese konjugierten Vektoren sind Gradienten eines quadratischen Funktionals, dessen Minimierung äquivalent zur Lösung des linearen Gleichungssystems ist. Bei positiv definiter Koeffizientenmatrix sind die Gradientenmethoden sehr effizient.

3 Auswahlverfahren und Vergleich

Steht man nun vor dem Problem, welche Klasse von Lösungsverfahren für das gestellte Problem nun am zweckmäßigsten ist, kann man sich an folgende Richtlinie halten:

- Kleine Systeme löst man am besten mit Programmen aus dem LAPACK, NAG- oder IMSL-Bibliothek, die alle auf direkten Verfahren beruhen. Auch für große Systeme mit Bandmatrix gibt es im LAPACK spezielle Programme.

- Große schwach besetzte Systeme mit allgemeiner Besetzheitsstruktur löst man am besten mit iterativen Verfahren, wie ITPACK, SLAP oder TEMPLATES.
  Bei der Auswahl kann die angeführte Tabelle als Entscheidungshilfe zwischen direkten und iterativen Methoden eingesetzt werden. Bei sehr großen Problemen scheiden direkte Verfahren manchmal ganz aus, weil durch Iterationsverfahren der Geschwindigkeitsgewinn als der überragende Vorteil angesehen werden muss.

Tabelle 1: Vergleich der Lösungsverfahren

	direkte Verfahren	**iterative Vefahren**
Genauigkeit	nicht beinflussbar	wählbar
Rechenaufwand	vorhersagbar	meist nicht vorhersagbar, aber oft kleiner
neue rechte Seiten	rasch	kleine Zeitersparnis
Speicherbedarf	größer	kleiner
Startwertvorgabe	nicht erforderlich	meist vorteilhaft
Algorithmusparameter	nicht erforderlich	müssen gesetzt werden
Black-Box-Verwendung	möglich	oft nicht möglich
Robustheit	ja	nein

4 Software

Wie für lineare Systeme vollbesetzter Matrizen gibt es auch für große schwach besetzte Systeme komerzielle Softwareprodukte sowie *Public-domain*-Software, die neben Speicherungs- und elementaren Verarbeitungsmöglichkeiten auch Lösungsmethoden zu verschiedenen mathematischen Problemen in diesem Zusammenhang bietet. Der Unterschied zum Bereich der vollbesetzten Matrizen ist der, dass sich bei den schwach besetzten Matrizen die weitgehende Standardisierung von Schnittstellen und Funktionsmerkmalen noch nicht eingetreten ist. Diese Tatsache beruht auf der strikten Problemabhängigkeit der Datenstrukturen zur Matrixdarstellung.

4.1 elementare Software

- Die *Harwell-Boeing-Collection*
  ist eine Sammlung von großen schwach besetzten Matrizen aus verschiedenen Anwendungsgebieten. Zur Speicherung wurde ein eigenes Format entwickelt: das *Harwell-Boeing*-Format, das im wesentlichen dem CRS-Format entspricht. Im Anhang A sind zwei Matrizen astrophysikalischer Anwendungen angefügt.

- Analog zu der BLAS gibt es auch ein Package, das sich mit schwach besetzten Matrizen befasst, das **SPARSE-BLAS**. Dieses Package enthält eigentlich Unterprogramme für die BLAS, die die Matrixelemente speziell abspeichern, um auch mit den Programmen für vollbesetzte Matrizen, bei geringerem Rechenaufwand, Lösungen zu erzielen.

4.2 Softwarepakete für Gleichungssysteme

Es gibt eine Anzahl von Softwarepaketen, die sich für das iterative Lösen von Matrizen eigenen, genannt seinen hier nur das **ITPACK**, entwickelt an der Universität von Texas und das **SLAP** der Lawrence Livermore National Laboratory.

Literatur

[1] LAPACK-LinearAlgebraPACKage: `http://www.netlib.org/lapack`, 22 01 01 letzter Abruf

[2] Matrixmarket: `http://math.nist.gov/MatrixMarket`, 23 01 01 letzter Abruf

[3] Mauser, N.: 1999, Numerik 2, Vorlesung am Institut für Mathematik, Universität Wien

[4] Netlib Repository: `http://www.netlib.org`, 18 01 01 letzter Abruf

[5] Timmes: `http://www.journals.uchicag.edu/ApJ/journal/issues/ApJS/v124n1/40233/40233.html`, 12 01 01 letzter Abruf

[6] Timmes, F.X.: 1999, Integration of Nuclear Reaction Networks for Stellar Hydrodynamics, ApJS, vol. 124, iss. 1, 241-263

[7] Überhuber, Ch.: 1995, Computernumerik 1, 1995, Springer

[8] Überhuber, Ch.: 1995, Computernumerik 2, 1995, Springer

[9] Young, D., M.: 1971, Iterative Solution of Large Linear Systems, 1971, Academic Press

A Matrix MCAA

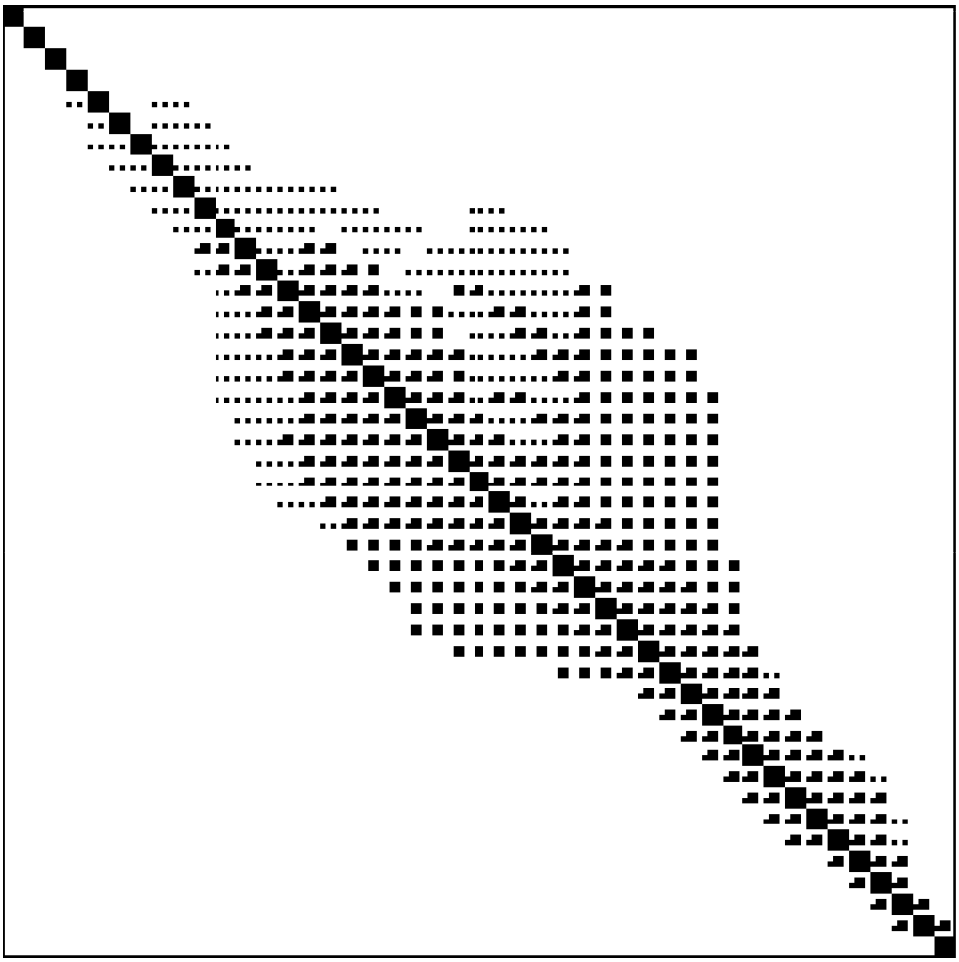

Abbildung 1: Matrix MCCA - Structure Plot

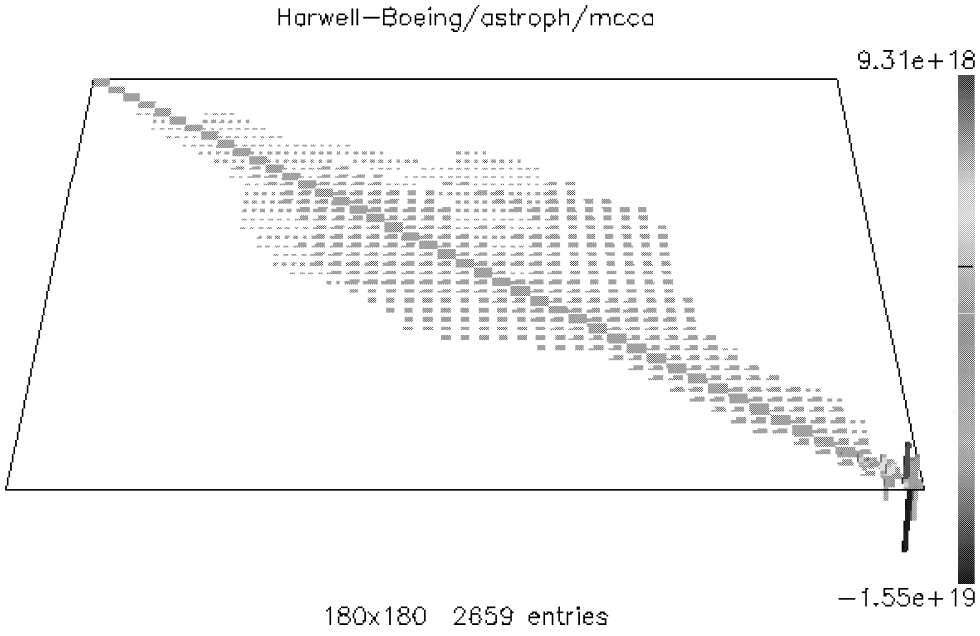

Abbildung 2: Matrix MCCA - City Plot

B Matrix MCFE

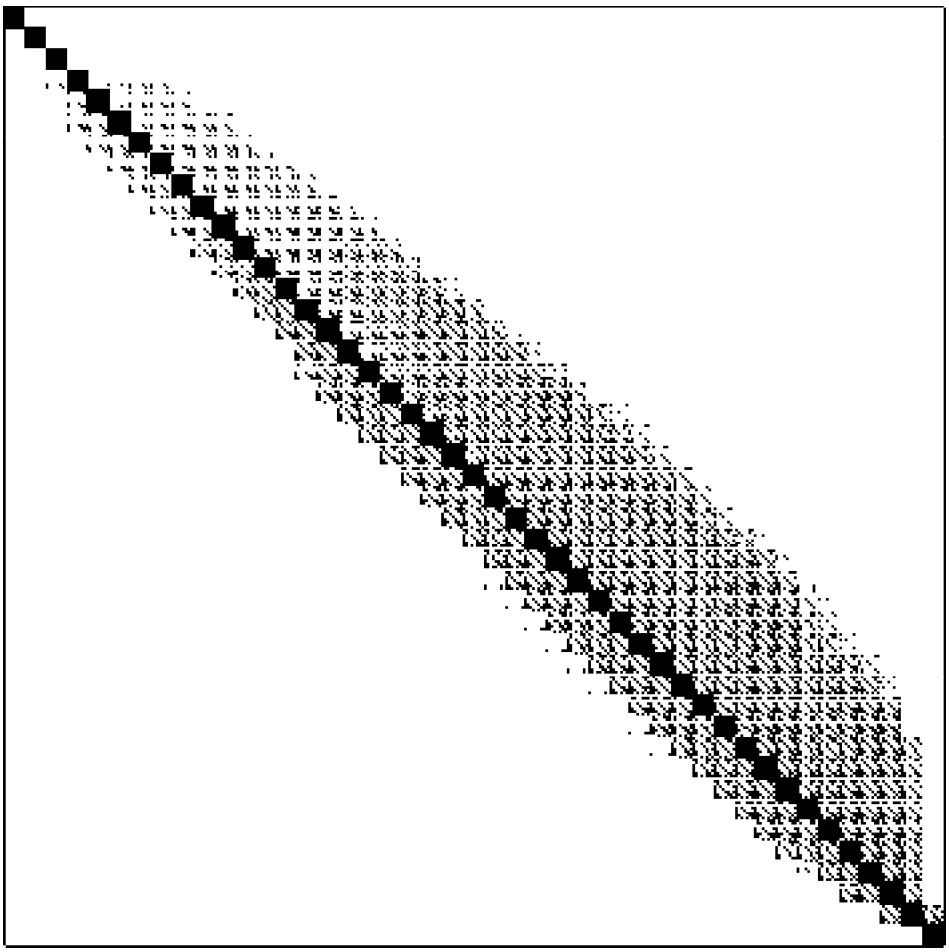

Abbildung 3: Matrix MCFE - Structure Plot

C Netzwerke

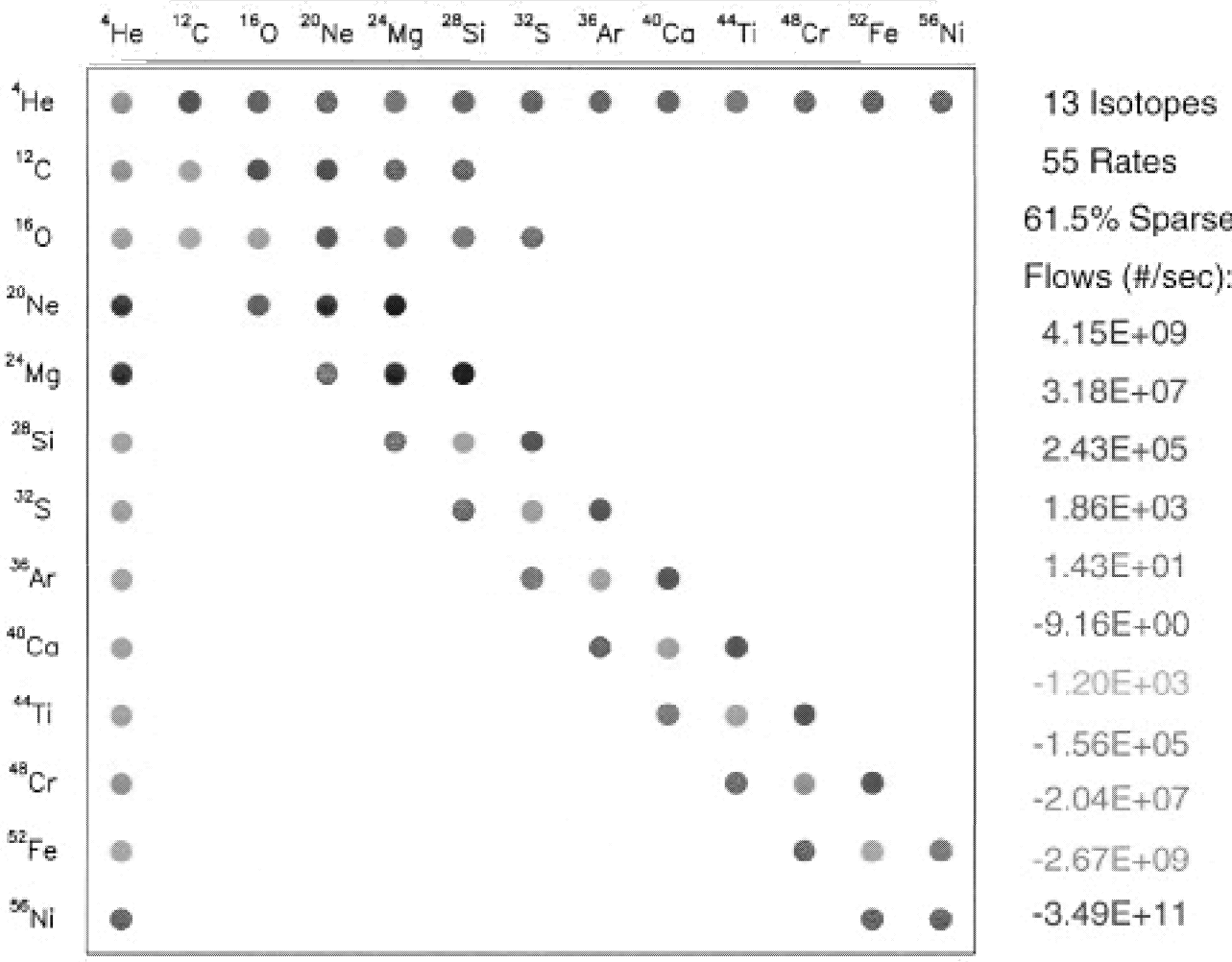

Abbildung 4: Netzwerk mit 13 Isotopen

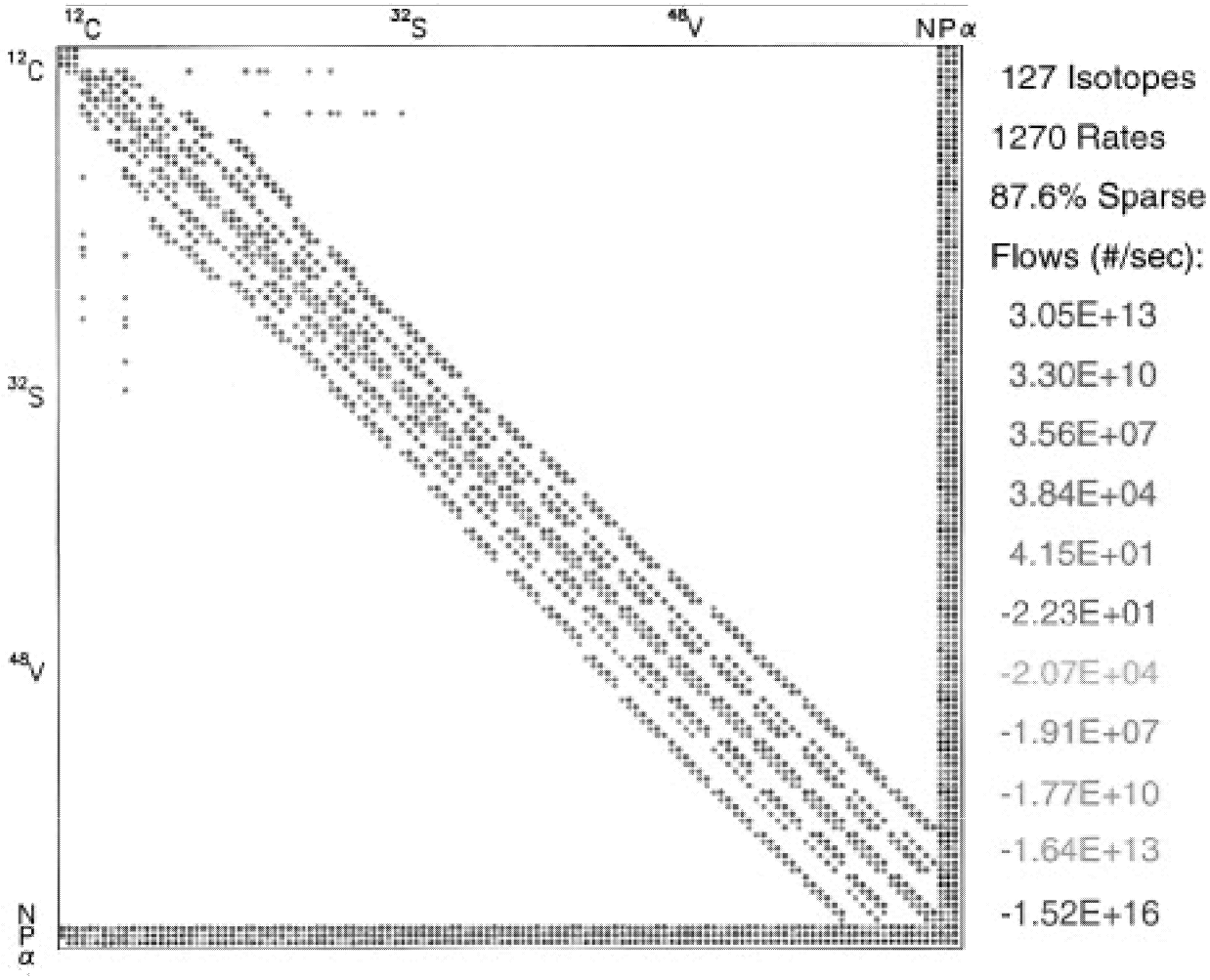

Abbildung 5: Netzwerk mit 127 Isotopen

D Iterationsverfahren - Testresultate

Tabelle 2: Symmetrische und positiv definite Matrix - Testresultate

Verfahren	Iterations-schritte	Fehler	Laufzeit
Gauß-Seidel	3000	$6,30 \cdot 10^{-2}$	88,6 s
SOR $\omega = 1,3$	3000	$1,09 \cdot 10^{-2}$	88,5 s
konjugierte Gradienten	635	$3,58 \cdot 10^{-11}$	14,9 s

Tabelle 3: Unsymmetrische Matrix - Testresultate

Verfahren	Iterations-schritte	Fehler	Laufzeit
Gauß-Seidel	3000	$5,19 \cdot 10^{-4}$	25,2 s
SOR $\omega = 1,3$	3000	$6,24 \cdot 10^{-7}$	25,2 s
konjugierte Gradienten	146	$3,55 \cdot 10^{-8}$	1,1 s