Technische Universität Ilmenau
Fakultät für Mathematik
und Naturwissenschaften
Institut für Mathematik
http://wcms3.rz.tu-ilmenau.de/~math/index_de.html

Postfach 10 05 65
D - 98684 Ilmenau
Germany
Tel.: 03677/69 3267
Fax: 03677/69 3272
Telex: 33 84 23 tuil d.
email: werner.neundorf@tu-ilmenau.de

Preprint No. M 09/05

Abstiegsverfahren

Teil III

Werner Neundorf

Mai 2005

‡MSC (2000): 65F10, 65G50, 65-05, 65Y20

Zusammenfassung

Die Entwicklung moderner numerischer Algorithmen hat zu einem hohen Bedarf an effizienten, robusten iterativen Gleichungssystemlösern geführt. So entstand eine Vielzahl von Verfahren, die man zur Gruppe der Projektionsmethoden und Krylov-Unterraum-Methoden zählt.
Gegenstand der Betrachtungen in dieser mehrteiligen Arbeit sind grundlegende Abstiegsverfahren als Vertreter dieser Algorithmengruppe, die auf Minimierungsaufgaben nach Umformulierung eines regulären linearen Gleichungssystems führen.
Unter bestimmten Voraussetzungen an die Matrix des Gleichungssystems werden geeignete Funktionale konstruiert und damit der Weg des Abstiegs illustriert.

Das Verhalten der Abstiegsverfahren ist in den normalen gutartigen Fällen hinreichend bekannt und untersucht worden. Hier soll zunächst eine Gegenüberstellung zu anderen Situationen gemacht werden, wo man unter veränderten Voraussetzungen arbeitet, und damit der typische Charakter der Minimierungsaufgabe nicht mehr vorhanden ist. Dabei ist teilweise noch mit zufrieden stellenden Ergebnissen zu rechnen, es können aber auch starke Abweichungen vom Normalfall auftreten. Diese Darstellungen findet der Leser in den Teilen I (Preprint No. M 19/04 IfMath TUI) und II (Preprint No. M 20/04).
Des Weiteren betrachten wir in diesem Teil III die Abstiegsverfahren als polynomiale Iterationsverfahren, machen einige Aufwandsuntersuchungen zu Matrizen und zum Matrix-Vektor-Produkt, das bei Abstiegsverfahren auftritt, und untersuchen den Einfluss von Rundungsfehlern bei der Implementation der Verfahren im Zusammenhang mit ihrer (eventuell) theoretischen Endlichkeit sowie mit der Notation des Formelapparates und seiner numerischen Auswertung (Verhalten und "Fehlererinnerung").

Praktische Rechnungen mit kleindimensionierten Beispielen, wo man dies auch gut illustrieren kann, demonstrieren die unterschiedlichen Situationen.

Eileen Caddy

Spuren auf dem Weg zum Licht

Hört auf zu versuchen,
alles gedanklich außuarbeiten,
das bringt euch nirgendwohin.
Lebt auf der Intuition und Inspiration,
und laßt euer ganzes Leben
eine Offenbarung sein.

Inhaltsverzeichnis

Teil I

Teil II

Kapitel 6

Polynomiale Iterationsverfahren

Polynomiale Iterationsverfahren (polynomial based iteration methods) entstehen durch verschiedene Zugänge bei der iterativen Lösung von LGS.
Wenn man bereits mit einem Basisverfahren berechnete Iterierte noch einmal kombiniert, um die Konvergenz zu verbessern, ergibt sich die sogenannte Semiiteration, das Beschleunigungsverfahren bzw. die polynomiale Konvergenzbeschleunigung (siehe [17] Abschnitt 3.6).
Wenn man bei Abstiegsverfahren die Iterierten und Residua (Residuenvektoren) untersucht, stößt man auf polynomiale Darstellungen. Um den dabei auftretenden Lösungsfehler möglichst klein zu machen, sind der Einsatz und Eigenschaften von speziellen Polynomen notwendig sowie damit Abschätzungen durchzuführen. Dies wurde im Teil I Abschnitt 3.6 der Reihe *Abstiegsverfahren* (Preprint No. M 19/04 TU Ilmenau 2004) beim Nachweis der Konvergenz des Verfahrens der konjugierten Gradienten (CG) angewendet.
Bei Abstiegsverfahren mit einer "Minimierung" des Residuums (minimal residual property, extremal property) treten orthogonale Polynome auf im Zusammenhang mit sogenannten Kernpolynomen sowie mit speziellen oder modifizierten Skalarprodukten und reproduzierenden Eigenschaften (reproducing property).
In allen Fällen erweisen sich Systeme von orthogonalen Polynomen und speziell die Tschebyscheff-Polynome als sehr hilfreich.

6.1 Polynomiale Konvergenzbeschleunigung

Die Idee dieser "Nachbesserung" der Vektoren einer Basisiteration geht auf R.S. Varga und P.L. Tschebyscheff zurück.

Wir gehen von der Normalform des LGS $Ax = b,\ A(n,n)$ regulär, mit der Voraussetzung $a_{ii} \neq 0,\ i = 1, 2, ..., n$, was z. B. durch eventuelle Zeilenvertauschungen zu erreichen ist, und $x^* = A^{-1}b$ als exakte Lösung aus.

Iterationsverfahren (IV) beruhen auf einer regulären Zerlegung (Splitting) von A und

der allgemeinen Umformung des LGS gemäß

$$\begin{aligned} A &= N - P, \quad \det(N) \neq 0, \\ Nx - Px &= b, \\ Nx &= Px + b, \qquad (6.1) \\ x &= \underbrace{N^{-1}P}_{H}x + \underbrace{N^{-1}b}_{c}, \\ x &= Hx + c. \qquad (6.2) \end{aligned}$$

Die Beziehung (6.2) ist die Fixpunktgleichung oder iterierfähige Form.
Für reguläres N sind die Darstellungen $Ax = b$ und $x = Hx + c$ mit den Beziehungen für H und c konsistent, d. h., dass die Lösung bzw. der Grenzvektor des konvergenten IV auch der Normalform genügt.
Die allgemeine Iterationsform (allgemeines Iterationsverfahren, Fixpunktiteration) lautet

$$x^{(k+1)} = Hx^{(k)} + c, \quad k = 0, 1, ..., \quad H = N^{-1}P, \quad c = N^{-1}b. \qquad (6.3)$$

Da die Iterationsmatrix H konstant ist, wird (6.3) auch lineare stationäre Fixpunktiteration genannt. Sie wird das Basisiterationsverfahren (Basis-IV) sein.
Damit erhebt sich die Frage, wann mit einem beliebigen Startvektor $x^{(0)}$ die Aussage $\lim\limits_{k\to\infty} x^{(k)} = x^*$ und somit die Konvergenz des Verfahrens gelten. Hinreichende Konvergenzbedingung ist $\|H\| < 1$, als hinreichend und notwendig erweist sich die Ungleichung mit dem Spektralradius $\rho = \rho(H) = \max(|\lambda(H)|) < 1$. Mehr zu Konvergenzuntersuchungen findet man u. a. in [17].

Im Teil I der Reihe *Abstiegsverfahren* (Preprint No. M 19/04) genannte IV waren:

(1) Gesamtschrittverfahren (GSV), Jacobi-IV (JA)

Zerlegung der Koeffizientenmatrix $A = N - P = D - (E + F)$, $D =$diag(A),
E linke (untere) Dreiecksmatrix (Diagonale und oberes Dreieck sind Null), F rechte (obere) Dreiecksmatrix.

Matrix-Vektor-Form des GSV

$$\begin{aligned} x^{(k+1)} &= D^{-1}[b + (E + F)x^{(k)}] \\ &= (I - D^{-1}A)x^{(k)} + D^{-1}b \qquad (6.4) \\ &= x^{(k)} + D^{-1}r^{(k)} \quad \text{mit } r^{(k)} = b - Ax^{(k)} \quad \text{als Residuum,} \end{aligned}$$

$H = J = I - D^{-1}A$ Iterationsmatrix,
$W = D$ Wichtung, Skalierungsmatrix,
$D^{-1}r^{(k)}$ gewichtetes Residuum (weighted residual),
Verbesserung (update) für $x^{(k)}$.

(2) Relaxationsverfahren mit parameterabhängiger Skalierung (Vorkonditionierung)
Reguläre Vorkonditionierungsmatrix $W_\omega = W(\omega)$.
Matrix-Vektor-Form des IV

$$\begin{aligned} x^{(k+1)} &= (I - W_\omega^{-1}A)x^{(k)} + W_\omega^{-1}b \\ &= x^{(k)} + W_\omega^{-1}\, r^{(k)} \;\text{ mit } r^{(k)} = b - Ax^{(k)} \;\text{ als Residuum,} \\ H &= I - W_\omega^{-1}A \;\text{ Iterationsmatrix,} \\ W &= W_\omega \qquad\quad \text{Wichtung, Skalierungsmatrix.} \end{aligned} \tag{6.5}$$

Das GSV gehört zu dieser Verfahrensgruppe, wobei $W_\omega = D$ ist.
(3) Richardson-Verfahren mit festen Parameter (RF, Extrapolationsverfahren)
Matrix-Vektor-Form des IV

$$\begin{aligned} x^{(k+1)} &= (I - \omega A)x^{(k)} + \omega b \\ &= x^{(k)} + \omega r^{(k)} \;\text{ mit } r^{(k)} = b - Ax^{(k)} \;\text{ als Residuum,} \\ H &= I - \omega A \;\text{ Iterationsmatrix,} \\ W &= \omega^{-1} I \qquad \text{Wichtung, Skalierungsmatrix.} \end{aligned} \tag{6.6}$$

Für den Fall $A = A^T > 0$ kann man sogar ein optimales $\omega = \omega_{opt}$ wählen.
(4) In Betracht können noch weitere Basis-IV kommen (siehe [17]).
– Relaxationsverfahren von Jacobi (JOR, JA(ω)) mit $W_\omega = \frac{1}{\omega} D$ und

$$H \;=\; J_\omega \;=\; I - \omega D^{-1}A. \tag{6.7}$$

Falls $A = A^T > 0$ gilt, ist JOR für einen bestimmten Parameterbereich $\omega > 0$ konvergent. Unter diesen gibt es einen optimalen Wert $\omega = \omega_0$, und man erhält das Verfahren JOR-OE (Jacobi optimum extrapolated).
– Einzelschrittverfahren (ESV), Gauss-Seidel-IV (GS) mit $W_1 = D - E$ und

$$H \;=\; H_1 \;=\; I - (D - E)^{-1}A. \tag{6.8}$$

– Überrelaxationsverfahren von Gauß-Seidel (Successive OverRelaxation, SOR) mit

$$\begin{aligned} W_\omega &= \omega^{-1}\, D - E \;=\; \omega^{-1}(D - \omega E), \\ H &= H_\omega \;=\; I - \omega(D - \omega E)^{-1}A \;=\; (D - \omega E)^{-1}((1 - \omega)D + \omega F). \end{aligned} \tag{6.9}$$

Praktisch berechnet man den neuen Iterationsvektor $x^{(k+1)}$ komponentenweise

$$x_i^{(k+1)} \;=\; (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, ..., n.$$

Für $A = A^T > 0$ und konsistent geordnet sowie $\omega \in (0, 2)$ ist das SOR konvergent.

– Symmetrisches IV von Gauß-Seidel (SGS) mit

$$\widetilde{W}_1 = (D-E)D^{-1}(D-F), \tag{6.10}$$

$$H = \tilde{H}_1 = I - \widetilde{W}_1^{-1}A. \tag{6.11}$$

Praktisch führt man zwei Halbschritte durch.
Vorwärtsiteration (forward sweep):

$x_1^{(k+\frac{1}{2})}, x_2^{(k+\frac{1}{2})}, ..., x_n^{(k+\frac{1}{2})}$ nach dem ESV auf der Basis von $x_i^{(k)}$,

Rückwärtsiteration (backward sweep):

$x_n^{(k+1)}, x_{n-1}^{(k+1)}, ..., x_1^{(k+1)}$ nach dem ESV auf der Basis von $x_i^{(k+\frac{1}{2})}$.

Die Zusammenfassung beider Teilschritte liefert

$$\begin{aligned} x^{(k+\frac{1}{2})} &= D^{-1}(b + Ex^{(k+\frac{1}{2})} + Fx^{(k)}), \\ x^{(k+1)} &= D^{-1}(b + Ex^{(k+\frac{1}{2})} + Fx^{(k+1)}) \\ &= x^{(k)} + [(D-E)D^{-1}(D-F)]^{-1}(b - Ax^{(k)}) \\ &= (I - \widetilde{W}_1^{-1}A)\, x^{(k)} + (D-F)^{-1}D(D-E)^{-1}\, b. \end{aligned}$$

– Symmetrisches SOR (SSOR) mit

$$\begin{aligned} \widetilde{W}_\omega &= \tfrac{1}{\omega(2-\omega)}\,(D-\omega E)D^{-1}(D-\omega F) = (\tfrac{1}{\omega}D - E)(\tfrac{2-\omega}{\omega}\,D)^{-1}(\tfrac{1}{\omega}D - F), \\ H &= \tilde{H}_\omega = I - \widetilde{W}_\omega^{-1}A \\ &= (D-\omega F)^{-1}((1-\omega)D + \omega E)(D-\omega E)^{-1}((1-\omega)D + \omega F). \end{aligned} \tag{6.12}$$

SSOR mit $\omega = 1$ führt zum SGS.
Auch hier sind es zwei Halbschritte, aber mit Einbeziehung der Relaxation.
Vorwärtsiteration:

$x_1^{(k+\frac{1}{2})}, x_2^{(k+\frac{1}{2})}, ..., x_n^{(k+\frac{1}{2})}$ nach SOR auf der Basis von $x_i^{(k)}$,

Rückwärtsiteration:

$x_n^{(k+1)}, x_{n-1}^{(k+1)}, ..., x_1^{(k+1)}$ nach SOR auf der Basis von $x_i^{(k+\frac{1}{2})}$.

Wir fassen beide Iterationen

$$\begin{aligned} x^{(k+\frac{1}{2})} &= (1-\omega)x^{(k)} + \omega D^{-1}(b + Ex^{(k+\frac{1}{2})} + Fx^{(k)}) \\ &= (D-\omega E)^{-1}((1-\omega)D + \omega F)\, x^{(k)} + \omega(D-\omega E)^{-1}b, \\ x^{(k+1)} &= (1-\omega)x^{(k+\frac{1}{2})} + \omega D^{-1}(b + Ex^{(k+\frac{1}{2})} + Fx^{(k+1)}) \\ &= (D-\omega F)^{-1}((1-\omega)D + \omega E)\, x^{(k+\frac{1}{2})} + \omega(D-\omega F)^{-1}b, \end{aligned}$$

wiederum zusammen und erhalten verschiedene Darstellungen des IV.

$$\begin{aligned} x^{(k+1)} &= (D-\omega F)^{-1}((1-\omega)D+\omega E)(D-\omega E)^{-1}((1-\omega)D+\omega F)\,x^{(k)}+c \\ &= x^{(k)}+[(\tfrac{1}{\omega}D-E)(\tfrac{2-\omega}{\omega}D)^{-1}(\tfrac{1}{\omega}D-F)]^{-1}(b-Ax^{(k)}) \\ &= (I-\widetilde{W}_\omega^{-1}A)\,x^{(k)}+\omega(2-\omega)\,(D-\omega F)^{-1}D(D-\omega E)^{-1}\,b \\ &= \tilde{H}_\omega\,x^{(k)}+c. \end{aligned}$$

Damit können wir nun einen Konvergenzsatz des SSOR beweisen.

Satz 6.1 *Sei* $A=A^T>0$ *und* $\omega\in(0,2)$.
Dann gelten die folgenden Aussagen.
(1) Alle Eigenwerte von $\tilde{H}_\omega$ *(6.12) sind reell und nicht negativ.*
(2) Es gilt $\rho(\tilde{H}_\omega)<1$, *d. h. das SSOR ist konvergent.*

Beweis.
Wegen $A=A^T$ gilt für die Matrixzerlegung $A=D-E-F=D-E-E^T,\ F=E^T$. Außerdem liefert $A>0$ dazu $D>0$ sowie die Existenz der beiden spd Quadratwurzelmatrizen $A^{1/2}$ und $A^{-1/2}$ (siehe [18], Kap. 2).
Zu (1): Wir zeigen die Ähnlichkeit der Matrix $\tilde{H}_\omega$ zu einer symmetrischen und positiv semidefiniten Matrix.
Wir notieren mit (6.12)

$$\begin{aligned} A^{1/2}\tilde{H}_\omega A^{-1/2} &= \\ &= A^{1/2}\underbrace{(D-\omega F)^{-1}((1-\omega)D+\omega E)}_{=H_b}\ \underbrace{(D-\omega E)^{-1}((1-\omega)D+\omega F)}_{=H_\omega}A^{-1/2} \\ &= A^{1/2}H_bA^{-1/2}\ A^{1/2}H_\omega A^{-1/2},\ \text{ wobei gemäß SOR} \\ &\quad H_\omega=I-\omega(D-\omega E)^{-1}A,\ \ H_b=I-\omega(D-\omega E^T)^{-1}A, \\ &= A^{1/2}[I-\omega(D-\omega E^T)^{-1}A]A^{-1/2}\ A^{1/2}H_\omega A^{-1/2} \\ &= [I-\omega A^{1/2}(D-\omega E^T)^{-1}A^{1/2}]\ A^{1/2}H_\omega A^{-1/2} \\ &= [I-\omega A^{1/2}(D-\omega E)^{-1}A^{1/2}]^T\ A^{1/2}H_\omega A^{-1/2} \\ &= [A^{1/2}A^{-1/2}-\omega A^{1/2}(D-\omega E)^{-1}AA^{-1/2}]^T\ A^{1/2}H_\omega A^{-1/2} \\ &= [A^{1/2}(I-\omega(D-\omega E)^{-1}A)A^{-1/2}]^T\ A^{1/2}H_\omega A^{-1/2} \\ &= (A^{1/2}H_\omega A^{-1/2})^T\ \underbrace{A^{1/2}H_\omega A^{-1/2}}_{=C} \\ &= C^TC. \end{aligned}$$

Da die Matrix C^TC symmetrisch und positiv semidefinit ist, gilt dies auch für die zu ihr ähnliche Matrix $\tilde{H}_\omega$, die damit nur reelle EW ≥ 0 hat.

Zu (2): Da ähnliche Matrizen dieselben EW besitzen, ergibt sich gemäß Teil (1) für den Spektralradius

$$\begin{aligned}\rho(\tilde{H}_\omega) &= \rho(A^{1/2}\tilde{H}_\omega A^{-1/2}) \\ &= \rho((A^{1/2}H_\omega A^{-1/2})^T(A^{1/2}H_\omega A^{-1/2})).\end{aligned}$$

Man zeigt, dass der letzte Spektralradius kleiner Eins ist.
Im SOR ist die Skalierungsmatrix $W_\omega = \omega^{-1}\,D - E$ (6.9) und damit gilt

$$W_\omega + W_\omega^T - A = \omega^{-1}\,D - E + \omega^{-1}\,D - E^T - D + E + E^T = \frac{2-\omega}{\omega}\,D > 0.$$

Die spd-Eigenschaft der Matrix $W_\omega + W_\omega^T - A$ notieren wir als $W_\omega + W_\omega^T > A$.
Mit einer beliebigen regulären Matrix K folgt aus der Definition der Definitheit auch

$$W_\omega + W_\omega^T > A \quad \rightarrow \quad K^T(W_\omega + W_\omega^T)K > K^T A K.$$

Wir betrachten also unter Verwendung von (6.9) den Matrixausdruck

$$\begin{aligned}&(A^{1/2}H_\omega A^{-1/2})^T(A^{1/2}H_\omega A^{-1/2}) \;= \\ &\quad= (A^{1/2}(I - W_\omega^{-1}A)A^{-1/2})^T(A^{1/2}(I - W_\omega^{-1}A)A^{-1/2}) \\ &\quad= (I - A^{1/2}W_\omega^{-T}A^{1/2})\,(I - A^{1/2}W_\omega^{-1}A^{-1/2}) \\ &\quad= I - A^{1/2}(W_\omega^{-T} + W_\omega^{-1})A^{1/2} + A^{1/2}W_\omega^{-T}A\,W_\omega^{-1}A^{1/2} \\ &\quad= I - (W_\omega^{-1}A^{1/2})^T(\underbrace{W_\omega + W_\omega^T}_{>A})(\underbrace{W_\omega^{-1}A^{1/2}}_{=K}) + A^{1/2}W_\omega^{-T}A\,W_\omega^{-1}A^{1/2} \\ &\quad< I - (W_\omega^{-1}A^{1/2})^T A\,(W_\omega^{-1}A^{1/2}) + A^{1/2}W_\omega^{-T}A\,W_\omega^{-1}A^{1/2} \\ &\quad= I.\end{aligned}$$

Daraus folgt für die Spektralradien

$$\rho((A^{1/2}H_\omega A^{-1/2})^T(A^{1/2}H_\omega A^{-1/2})) < \rho(I) = 1,$$

wobei die Folgerung $B = C^TC,\ 0 < B = B^T < I \ \rightarrow\ \rho(B) < 1$ mit $\rho(B) = \lambda_{max}(B)$ und $Bv = \lambda_{max}(B)v,\ \|v\|_2 = 1$, einfach zu zeigen ist. □

Die Implementation des SSOR ist nicht viel teurer als die von SOR. Jeder Iterationsschritt ist zwar ein Doppelschritt (Vorwärts- und Rückwärtsiteration), aber die in der Vorwärtsrechnung zu berechnenden Größen $\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1/2)},\ i = 1, 2, ..., n$, tauchen in dieser Form auch in der Rückwärtsrechnung wieder auf und brauchen nur einmal ermittelt zu werden. Ebenso können die Werte $\sum_{j=i+1}^{n} a_{ij}x_j^{(k+1)},\ j = n, n-1, ..., 1$ für die nächste Stufe verwendet werden. Dazu bedarf es etwas Speicherplatz in Form von Vektoren.

6.1.1 Tschebyscheff-Beschleunigung

Nun steht die Frage, wie man aus der Folge der Vektoren $\{x^{(k)}\}$ eine zweite (sekundäre) Folge $\{v^{(k)}\}$ konstruieren kann, die möglichst schneller gegen die Lösung x^* konvergiert als $\{x^{(k)}\}$ selbst.

Das **Beschleunigungsverfahren** bzw. die Verbesserung durch eine gewichtete reelle Linearkombination ist

$$v^{(k)} \;=\; \sum_{i=0}^{k} \alpha_{ki} x^{(i)}, \quad k = 0, 1, ..., \quad \sum_{i=0}^{k} \alpha_{ki} = 1. \tag{6.13}$$

Die Bedingung für die Koeffizienten ist plausibel, denn bei $x^{(0)} = x^{(1)} = ... = x^{(k)} = x^*$ sollte natürlich $v^{(k)} = x^*$ sein. Die Bestimmung der Koeffizienten kann z. B. gemäß

$$\begin{aligned} \text{Cesaro}: \quad & \alpha_{ki} = \frac{1}{k+1} \;\Rightarrow\; v^{(k)} = \text{arithmetisches Mittel}, \\ \text{Tschebyscheff}: \quad & \text{Koeffizienten } \alpha_{ki} \text{ auf der Basis von Eigenschaften} \\ & \text{der Tschebyscheff-Polynome}, \end{aligned}$$

erfolgen, wobei wir uns für den zweiten Fall interessieren.

Möglich ist nun eine Rückführung von $v^{(k)}$ auf den Startvektor $x^{(0)}$ der Basisiteration durch die folgende Betrachtung.

$$\begin{aligned} v^{(0)} &= x^{(0)}, \\ v^{(1)} &= \alpha_{10} x^{(0)} + \alpha_{11} x^{(1)} \\ &= (\alpha_{10} + \alpha_{11} H) x^{(0)} + \alpha_{11} c, \\ v^{(2)} &= \alpha_{20} x^{(0)} + \alpha_{21} x^{(1)} + \alpha_{22} x^{(2)} \\ &= (\alpha_{20} + \alpha_{21} H + \alpha_{22} H^2) x^{(0)} + \alpha_{21} c + \alpha_{22}(H + I) c, \\ v^{(k)} &= \Big(\sum_{i=0}^{k} \alpha_{ki} H^i \Big) x^{(0)} + \Big(\sum_{i=1}^{k} \alpha_{ki} \sum_{j=0}^{i-1} H^j \Big) c \;=\; p_k(H)\, x^{(0)} + d. \end{aligned} \tag{6.14}$$

Für die beiden Fehler $e^{(k)} = x^* - x^{(k)}$ der Basisfolge und $d^{(k)} = x^* - v^{(k)}$ der modifizierten Folge gelten unter Berücksichtigung von (6.13) und $d^{(0)} = e^{(0)}$ die Beziehungen

$$\begin{aligned} e^{(k)} &= H^k e^{(0)}, \\ d^{(k)} &= x^* - \sum_{i=1}^{k} \alpha_{ki} x^{(i)} = \sum_{i=1}^{k} \alpha_{ki} (x^* - x^{(i)}) = \sum_{i=1}^{k} \alpha_{ki} H^i\, e^{(0)} = p_k(H)\, d^{(0)}. \end{aligned} \tag{6.15}$$

Dabei ist

$$p_k(x) \;=\; \sum_{i=0}^{k} \alpha_{ki} x^i \tag{6.16}$$

ein Polynom $k-$ten Grades, d. h. $p_k(x) \in \mathcal{P}_k(x)$, mit $p_k(1) = 1$, also $p_k(x) \in \mathcal{P}_k^1(x)$.

Daraus leitet sich die Minimierungsaufgabe

$$\min_{p_k(x) \in \mathcal{P}_k^1(x)} \|p_k(H)\| \tag{6.17}$$

für das Matrixpolynom $p_k(H)$ ab, die gerade auf die Wahl der Tschebyscheff-Polynome mit den Koeffizienten α_{ki} führt.

Eine rekursive Berechnung von $v^{(k)}$ ist durchführbar.

$$\begin{aligned}
v^{(0)} &= x^{(0)}, \\
v^{(1)} &= Hv^{(0)} + c \;=\; Hx^{(0)} + c \;=\; x^{(1)}, \\
v^{(k)} &= (1-\omega_k)v^{(k-2)} + \omega_k \underbrace{(Hv^{(k-1)} + c)}_{\tilde{v}^{(k-1)}}, \quad k = 2, 3, ..., \\
&\text{wobei} \quad \omega_1 = 1, \;\; \omega_k = \frac{2\,T_{k-1}(1/\rho)}{\rho\,T_k(1/\rho)} < 2, \\
&\qquad T_k(x) \;=\; \begin{cases} \cos(k\,\arccos(x)), & \text{falls } x \in [-1,1], \\ \cosh(k\,\operatorname{arcosh}(x)), & \text{sonst}, \end{cases} \\
&\qquad \rho \;=\; \rho(H) \;=\; \max(|\lambda(H)|) < 1.
\end{aligned} \tag{6.18}$$

Bemerkung 6.1 Es gibt eine Ähnlichkeit zur stationären oder instationären Extrapolation (hier ändert sich die Übergangsmatrix H in jedem Schritt). Anstelle von $\tilde{v}^{(k-1)}$ wird zumeist wieder eine Extrapolationsformel derselben Gestalt eingesetzt.
Die obige Formelzeile mit drei aufeinander folgenden Iterierten $v^{(i)},\ i = k, k-1, k-2$, wird auch als dreistufiges Verfahren oder, wie schon bei Polynomsystemen benutzt, als Drei-Term-Rekursion bezeichnet (siehe [33]). Eine weitere Vergrößerung der Anzahl der einbezogenen Stufen (stage) ist im Allgemeinen nicht mehr sinnvoll und meist schon zu aufwändig. Ein gewisses Maß an "Vergangenheit" sollte berücksichtigt werden. Dazu reichen meist die Zwei- und Drei-Term-Rekursionen aus.

Wenden wir uns also der sekundären Folge $\{v^{(k)}\}$ und ihrem Fehler $d^{(k)}$ zu.
Damit letzterer möglichst klein wird, kommt man auf die Idee anstelle der Norm $\|p_k(H)\|$ in (6.17) den Spektralradius $\rho(p_k(H))$ zu minimeren. Dieser ist zwar allgemein nicht bekannt, aber häufig lässt sich eine geeignete Abschätzung finden.
Wir nehmen an, dass die Iterationsmatrix H nur reelle Eigenwerte $\lambda(H)$ sowie das Spektrum $\sigma(H)$ hat. Ferner gehen wir von Schätzungen a und b für ihren kleinsten bzw. größten Eigenwert aus, so dass

$$a \;\le\; \lambda_{min} \;\le\; \lambda(H) \;\le\; \lambda_{max} \;\le\; b \;<\; 1 \tag{6.19}$$

mit $a < b$ gilt. Statt die Aufgabe

$$\min_{p_k(\lambda) \in \mathcal{P}_k^1(\lambda)} \max_{\lambda \in \sigma(H)} |p_k(\lambda)| \tag{6.20}$$

zu lösen, betrachten wir das etwas einfachere Ersatzproblem

$$\min_{p_k(\lambda) \in \mathcal{P}_k^1(\lambda)} \max_{\lambda \in [a,b]} |p_k(\lambda)|. \tag{6.21}$$

Dafür ist die Lösung bekannt und durch ein geeignet transformiertes und skaliertes Tschebyscheff-Polynom gegeben.
Im Teil I *Abstiegsverfahren* sind in den Lemmata (3.12) – (3.15) die Grundlagen und Formeln dazu beschrieben. Die wichtigste Eigenschaft finden wir in der folgenden Aussage.
Mit $-\infty < a < b < 1$, der bijektiven Intervallabbildung

$$g(x) \;=\; \frac{2x-(a+b)}{b-a} \;:\; [a,b] \to [-1,1], \tag{6.22}$$

dem k-ten Tschebyscheff-Polynom $T_k(x) = \cos(k \arccos(x)) : [-1,1] \to [-1,1]$ sowie dem skalierten Polynom

$$p_k(x) \;=\; \frac{T_k(g(x))}{T_k(g(1))} \tag{6.23}$$

gelten $p_k(x) \in \mathcal{P}_k^1(x)$, also $p_k(1) = 1$, und

$$\max_{a \le x \le b} |p_k(x)| \;\le\; \max_{a \le x \le b} \{|q_k(x)| : \; q_k(x) \in \mathcal{P}_k^1(x)\}. \tag{6.24}$$

Die gesuchten Koeffizienten α_{ki} sind damit gerade die Koeffizienten des Polynoms $p_k(x)$ aus (6.23), die wir aber nicht explizit ermitteln müssen.
Wir nutzen die Drei-Term-Rekursion $T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)$ der Tschebyscheff-Polynome (siehe [33]), um eine einfache Bildungsvorschrift für die Vektoren $\{v^{(k)}\}$ zu erhalten, ohne die Basisfolge $\{x^{(k)}\}$ direkt zu berechnen und schon gar nicht alle Vektoren $x^{(0)}, x^{(1)}, ..., x^{(k)}$ abspeichern zu müssen. Für (6.22) erhält man

$$g(H) \;=\; \frac{2H-(a+b)I}{b-a} \;=\; \frac{2}{b-a}H - \frac{a+b}{b-a}I. \tag{6.25}$$

Dann gelten wegen (6.15), (6.23)

$$\begin{aligned}
d^{(k+1)} &= p_{k+1}(H)\, d^{(0)} \\
&= \frac{T_{k+1}(g(H))}{T_{k+1}(g(1))}\, d^{(0)} \\
&= \frac{2g(H)T_k(g(H)) - T_{k-1}(g(H))}{T_{k+1}(g(1))}\, d^{(0)} \\
&= 2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\,\frac{T_k(g(H))}{T_k(g(1))} d^{(0)} - \frac{T_{k-1}(g(1))}{T_{k+1}(g(1))}\,\frac{T_{k-1}(g(H))}{T_{k-1}(g(1))} d^{(0)} \\
&= 2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\, d^{(k)} - \frac{T_{k-1}(g(1))}{T_{k+1}(g(1))}\, d^{(k-1)}
\end{aligned} \tag{6.26}$$

sowie mit $d^{(k)} = x^* - v^{(k)}$, $T_{k+1} + T_{k-1} = 2xT_k$, $(I-H)x^* = c$ und (6.25) weiter

$$\begin{aligned}
v^{(k+1)} &= x^* - d^{(k+1)} \\
&= x^* - 2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\,x^* + \frac{T_{k-1}(g(1))}{T_{k+1}(g(1))}\,x^* \\
&\qquad +2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\,v^{(k)} - \frac{T_{k-1}(g(1))}{T_{k+1}(g(1))}\,v^{(k-1)} \\
&= \Big(I - 2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))} + \frac{T_{k-1}(g(1))}{T_{k+1}(g(1))}I\Big)\,x^* \\
&\quad +2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\,v^{(k)} - \frac{2g(1)T_k(g(1)) - T_{k+1}(g(1))}{T_{k+1}(g(1))}\,v^{(k-1)} \\
&= \Big(\frac{T_{k+1}(g(1))}{T_{k+1}(g(1))}I + \frac{T_{k-1}(g(1))}{T_{k+1}(g(1))}I - 2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\Big)\,x^* \\
&\quad \underbrace{+2\Big(\frac{2}{b-a}H - \frac{a+b}{b-a}I\Big)\,\frac{T_k(g(1))}{T_{k+1}(g(1))}\,v^{(k)} + \Big(1 - 2g(1)\frac{T_k(g(1))}{T_{k+1}(g(1))}\Big)\,v^{(k-1)}}_{S_2} \qquad (6.27) \\
&= \Big(\frac{2g(1)T_k(g(1))}{T_{k+1}(g(1))}I - 2g(H)\frac{T_k(g(1))}{T_{k+1}(g(1))}\Big)\,x^* + S_2 \\
&= 2\left(g(1)I - g(H)\right)\frac{T_k(g(1))}{T_{k+1}(g(1))}\,x^* + S_2 \\
&= \frac{4}{b-a}(I-H)\,\frac{T_k(g(1))}{T_{k+1}(g(1))}\,x^* + S_2 \\
&= \frac{4}{b-a}\,\frac{T_k(g(1))}{T_{k+1}(g(1))}\,c \\
&\quad +2\Big(\frac{2}{b-a}H - \frac{a+b}{b-a}I\Big)\,\frac{T_k(g(1))}{T_{k+1}(g(1))}\,v^{(k)} + \Big(1 - 2g(1)\frac{T_k(g(1))}{T_{k+1}(g(1))}\Big)\,v^{(k-1)}.
\end{aligned}$$

Mit den Abkürzungen

$$\begin{aligned}
\rho_k &= 2g(1)\frac{T_{k-1}(g(1))}{T_k(g(1))}, \\
\gamma &= \frac{2}{2-b-a}, \\
g(1) &= \frac{2-a-b}{b-a}, \\
&\gamma = 1 \text{ und } g(1) = 1/b, \quad \text{falls} \quad a = -b,
\end{aligned} \qquad (6.28)$$

vereinfacht sich die Darstellung zur gewünschten Form

$$\begin{aligned} v^{(k+1)} &= \frac{2\rho_{k+1}}{g(1)(b-a)}\, c + \frac{\rho_{k+1}}{g(1)}\Big(\frac{2}{b-a}H - \frac{a+b}{b-a}I\Big)\,\frac{T_k(g(1))}{T_{k+1}(g(1))}\, v^{(k)} \\ &\quad +(1-\rho_{k+1})\, v^{(k-1)} \\ &= \frac{\rho_{k+1}}{2-b-a}\,[2Hv^{(k)} + 2c - (b+a)v^{(k)}] + (1-\rho_{k+1})\, v^{(k-1)} \\ &= \rho_{k+1}[\gamma(Hv^{(k)}+c) + (1-\gamma)v^{(k)}] + (1-\rho_{k+1})\, v^{(k-1)}, \\ &\quad k = 1, 2, \dots\ . \end{aligned} \tag{6.29}$$

Die Drei-Term-Rekursion der Tschebyscheff-Polynome erlaubt auch die rekursive Berechnung der Parameter ρ_k aus (6.28) gemäß

$$\begin{aligned} \rho_{k+1} &= \frac{2g(1)T_k(g(1))}{T_{k+1}(g(1))} \\ &= \frac{2g(1)T_k(g(1))}{2g(1)T_k(g(1)) - T_{k-1}(g(1))} \\ &= \frac{4g(1)^2\, T_k(g(1))}{4g(1)^2\, T_k(g(1)) - 2g(1)T_{k-1}(g(1))} \\ &= \frac{4g(1)^2}{4g(1)^2 - \frac{2g(1)T_{k-1}(g(1))}{T_k(g(1))}} \\ &= \frac{4g(1)^2}{4g(1)^2 - \rho_k} \\ &= \frac{1}{1 - \frac{1}{4g(1)^2}\rho_k}. \end{aligned} \tag{6.30}$$

Um alle Rekursionen für $v^{(k+1)}$ und ρ_{k+1} zu verwenden, brauchen wir noch die Startgrößen $v^{(0)},\ v^{(1)}$ und ρ_1. Aus (6.28) folgt sofort

$$\rho_1 \;=\; 2g(1)\frac{T_0(g(1))}{T_1(g(1))} \;=\; 2g(1)\frac{1}{g(1)} \;=\; 2.$$

Zur Bestimmung von $v^{(0)},\ v^{(1)}$ nehmen wir die Beziehungen (6.14), (6.23) und erhalten

$$\begin{aligned} p_0(x) &= \frac{T_0(g(x))}{T_0(g(1))} \;=\; 1 \;=\; \alpha_{00}, \\ p_1(x) &= \frac{T_1(g(x))}{T_1(g(1))} \;=\; \frac{-b-a}{2-b-a} + \frac{2}{2-b-a}x = \alpha_{10} + \alpha_{11}x = (1-\gamma) + \gamma x, \end{aligned}$$

$$
\begin{aligned}
v^{(0)} &= \alpha_{00}x^{(0)} = x^{(0)},\\
v^{(1)} &= \alpha_{10}x^{(0)} + \alpha_{11}x^{(1)}\\
&= \alpha_{10}x^{(0)} + \alpha_{11}(Hx^{(0)} + c)\\
&= \gamma(Hv^{(0)} + c) + (1-\gamma)v^{(0)}.
\end{aligned}
$$

Bei der Berechnung von $v^{(1)}$ tritt kein Parameter ρ_1 auf, man könnte sich diesen jedoch in (6.29) wertemäßig als Eins denken.

Algorithmus der Tschebyscheff-Beschleunigung einer Basisiteration aufgrund einer Matrixzerlegung

$$
\begin{aligned}
& & & H,\ \sigma(H),\ a \le \lambda_{min}(H) \le \lambda(H) \le \lambda_{max}(H) \le b < 1\\
& \gamma &=& \frac{2}{2-b-a},\quad \rho_1 = 2\\
& & & v^{(0)}\ \text{Startvektor}\ (= x^{(0)})\\
& v^{(1)} &=& \gamma(Hv^{(0)} + c) + (1-\gamma)v^{(0)}\\
& \underline{\mathbf{k = 1, 2, ...}} & & \\
& \rho_{k+1} &=& \frac{1}{1 - \frac{1}{4g(1)^2}\,\rho_k}\\
& v^{(k+1)} &=& \rho_{k+1}[\,\gamma(Hv^{(k)} + c) + (1-\gamma)v^{(k)}\,] + (1-\rho_{k+1})\,v^{(k-1)}\\
& \underline{\mathbf{end\ k}} & &
\end{aligned}
\tag{6.31}
$$

Die Tschebyscheff-beschleunigten IV kürzt man oft mit der Bezeichnung "T-Basisiteration" ab, z. B. T-GSV, T-ESV, T-SGS, T-SOR, T-SSOR.

Als mögliches Abbruchkriterium empfiehlt sich im T-IV neben der Beschränkung der Anzahl der Iterationen z. B. der Test

$$
\frac{\|b - Av^{(k)}\|}{\|b - Av^{(0)}\|} < \varepsilon \tag{6.32}
$$

bei vorgegebener Toleranz $\varepsilon > 0$.

Natürlich sollte die Basisiteration (6.3) möglichst konvergieren, also $-1 < a \le \lambda_{min}(H) \le \lambda_{max}(H) \le b < 1$ gelten. Dann bietet sich manchmal an mit $b = -a < 1$ zu arbeiten, was zur Formel $\gamma = 1$ führt. Falls insbesondere $\rho(H) = b = -a < 1$ ist, dann ist neben $\gamma = 1$ noch $g(1) = 1/\rho(H)$ und wir gelangen zu den eingangs notierten Beziehungen (6.18).

6.1.2 Modellproblem mit Vergleich von Iterationsverfahren

Als Modellproblem wird eine einfache eindimensionale Zweipunktrandwertaufgabe (RWA) mit homogenen Randbedingungen gewählt. Das zur numerischen Behandlung verwendete Diskretisierungsverfahren führt auf ein LGS, dessen Eigenschaften dargestellt werden. Des Weiteren werden für die Lösung des LGS einige IV untersucht, insbesondere die Fragen der Konvergenz im Zusammenhang mit dem Spektrum der jeweiligen Iterationsmatrix sowie der Effizienz des Verfahrens.

- Zweipunktrandwertaufgabe mit Dirichletschen (homogenen bzw. inhomogenen) Randbedingungen (RB):

$$\begin{aligned} -U''(x) &= F(x), \quad x \in \Omega = (0,1) \subset \mathbb{R}, \\ U &= \varphi \quad \text{für} \quad x \in \partial\Omega \quad \text{bzw.} \quad U(0) = \varphi_0, \quad U(1) = \varphi_1. \end{aligned} \tag{6.33}$$

- Gitter: $\Omega_h = \{x \mid x = ih,\ i = 1(1)N,\ h = 1/N\}$, h Maschenweite.
- Gitterfunktion: $u_h = (u_1, u_2, ..., u_{N-1})^T$ mit $u_i \approx U_i = U(ih)$.
- Analog für rechte Seite: $f_h = (f_1, f_2, ..., f_{N-1})^T$ mit $f_i = F_i = F(ih)$, d. h. auf dem Gitter wird die rechte Seite exakt dargestellt.
- Approximation der Ableitungen (Operatoren) mittels Differenzenausdrücken:

$$U_{xx} \sim \frac{1}{h^2}(U_{i+1} - 2U_i + U_{i-1}) \quad \text{zentraler Differenzenquotient 2. Ordnung.}$$

- Diskretisierte Aufgabe als LGS:

$$\begin{aligned} -\frac{1}{h^2}(u_{i+1} - 2u_i + u_{i-1}) &= f_i, \quad i = 1, 2, ..., N-1, \\ u_0 = \varphi_0, \quad u_N &= \varphi_1. \end{aligned} \tag{6.34}$$

- Matrixschreibweise (Matrix-Vektor-Notation) des LGS:

$$A_h u_h = b_h \quad \text{bzw.} \quad Au = b \tag{6.35}$$

mit

$$A_h = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b_h = \begin{pmatrix} f_1 + \varphi_0/h^2 \\ f_2 \\ f_3 \\ \cdots \\ f_{N-2} \\ f_{N-1} + \varphi_1/h^2 \end{pmatrix},$$

$$A = \begin{pmatrix} 2 & -1 & & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b = h^2 \begin{pmatrix} f_1 + \varphi_0/h^2 \\ f_2 \\ f_3 \\ \cdots \\ f_{N-2} \\ f_{N-1} + \varphi_1/h^2 \end{pmatrix},$$

$$A = \text{tridiag}(-1, 2, -1).$$

Für die rechte Seite $F(x) = \pi^2 \sin(\pi x)$ und homogene RB $U(0) = 0,\ U(1) = 0$ ist die exakte Lösung $U(x) = \sin(\pi x)$, so dass auch ein Vergleich mit der Näherungslösung aus (6.35) möglich ist.

Da die Systemmatrix symmetrisch und tridiagonal ist, betrachten wir zunächst einige mit ihr zusammenhängende Matrix-Eigenwertprobleme.

EW und EV von speziellen Tridiagonalmatrizen

Für Tridiagonalmatrizen mit jeweils gleichen Elementen auf den Diagonalen und Nebendiagonalen kann man das EWP direkt lösen. Wir fassen die Ergebnisse zusammen.

$A_1 = \text{tridiag}(1, 0, 1)$,

EW $\lambda_i = 2\cos(i\pi/(n+1)),\quad i = 1, 2, ..., n,$

$-2 < \lambda_n < \lambda_{n-1} < .... < \lambda_2 < \lambda_1 = 2\cos(\pi/(n+1)) < 2,$

$\lambda_{n+1-i} = -\lambda_i,\quad i = 1, 2, ..., [\frac{n}{2}],$ falls n ungerade, dann $\lambda_{\frac{n+1}{2}} = 0,$

EV $v^{(i)} = (v_1^{(i)}, v_2^{(i)}, ..., v_n^{(i)})^T$ mit $v_j^{(i)} = \sin(ij\pi/(n+1)).$

$A_2 = \text{tridiag}(d, 0, d^{-1}),\quad d > 0,$

EW $\lambda_i = 2\cos(i\pi/(n+1)),\ i = 1, 2, ..., n,$ von d unabhängig.

EW und EV der Systemmatrix $A(n, n),\ n = N - 1$

$A = \text{tridiag}(-1, 2, -1)$,

$$\text{EW}\quad \lambda_i = 2[1 - \cos(i\pi/(n+1))] = 4\sin^2(i\pi/(2(n+1))),\quad i = 1, 2, ..., n, \tag{6.36}$$

EV $v^{(i)} = (v_1^{(i)}, v_2^{(i)}, ..., v_n^{(i)})^T$ mit $v_j^{(i)} = \sin(ij\pi/(n+1)).$

Wegen $h = 1/(n+1),\quad \cos(x) = 1 - 2\sin^2(x/2),\quad$ gelten die Beziehungen

$\lambda_i = 4\sin^2(i\pi h/2),$

$v^{(i)},\quad v_j^{(i)} = \sin(ij\pi h),$

$0 < 2[1 - \cos(\pi h)] = 4\sin^2(\pi h/2) = \lambda_{min} = \lambda_1 < \lambda_2 < .... < \lambda_n = \lambda_{max} =$

$= 4\sin^2(n\pi h/2) = 4[1 - \sin^2(\pi h/2)] = 4\cos^2(\pi h/2) = 4 - \lambda_{min} < 4,$

$$\lambda_1 \approx \pi^2 h^2, \qquad \lim_{h\to 0} \lambda_1 = 0,$$
$$\lambda_n \approx 4 - \pi^2 h^2, \quad \lim_{h\to 0} \lambda_n = 4,$$
$$\lambda_{\frac{n+1}{2}} = 2, \quad \text{falls } n \text{ ungerade.}$$

Damit hat A das reelle Spektrum $\sigma(A) \in (0,4)$.

Insgesamt gilt $A = A^T > 0$, darüber hinaus ist A irreduzibel diagonaldominant und als spd Tridiagonalmatrix konsistent geordnet (vergl. [17]).

Vergleich der Eigenschaften der Basis-IV

Die Kenntnis der EW der Matrix A ist ein gute Ausgangsbasis für Abschätzungen der EW der Iterationsmatrix H und zugehörige Konvergenzbetrachtungen.

EW: $\lambda = \lambda(A) \in (0,4)$,

$$\lambda_1 = \lambda_{min} = 4\sin^2(\pi h/2), \quad \lambda_n = \lambda_{max} = 4 - \lambda_{min} = 4\cos^2(\pi h/2).$$

Die spektrale Konditionszahl von A ist

$$\kappa \;=\; \mathrm{cond}_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{4\cos^2(\pi h/2)}{4\sin^2(\pi h/2)} \approx \frac{1}{(\pi h/2)^2} = \frac{4}{\pi^2 h^2} = \frac{4(n+1)^2}{\pi^2}. \qquad (6.37)$$

Die Konvergenz der untersuchten Basis-IV ist wegen der günstigen Eigenschaften der Matrix A gesichert (siehe Konvergenzsätze in [13], [14], [15], [17], [31]).

IV	Iterationsmatrix H	EW $\mu = \mu(H)$	$\rho(H)$	
			$\max\|\mu(H)\|$	$\approx$
RF	$R = I - \omega A$ $\omega_{opt}^{\mathrm{RF}} = \dfrac{2}{\lambda_{min}+\lambda_{max}} = \dfrac{1}{2}$	$1 - \omega\lambda_i$	$\dfrac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}}$ $= \cos(\pi h)$	$1-\frac{\pi^2 h^2}{2}$
GSV, JA	$J = I - D^{-1}A$ $= I - \frac{1}{2}A$	$\mu_i = 1 - \frac{1}{2}\lambda_i$ $\mu_i \in (-1,1)$ $\mu_i = -\mu_{n+1-i}$ $\mu_{\frac{n+1}{2}} = 0$, n ung.	$\mu_1 = 1 - \frac{1}{2}\lambda_{min}$ $= \lvert 1 - \frac{1}{2}\lambda_{max}\rvert$ $= 1 - 2\sin^2(\frac{\pi h}{2})$ $= \cos(\pi h)$	$1-\frac{\pi^2 h^2}{2}$
ESV, GS	$H_1 = I - (D-E)^{-1}A$	0 sowie $\mu_i^2(J) \in [0,1)$ $0, ..., \mu_2^2, \mu_1^2$	$\mu_1^2 = \cos^2(\pi h)$ $< \mu_1$	$1-\pi^2 h^2$

IV	Iterationsmatrix H	EW $\mu = \mu(H)$	$\rho(H)$	
			$\max\|\mu(H)\|$	$\approx$
JOR, JA(ω)	$J_\omega = I - \omega D^{-1}A$ $= I - \frac{\omega}{2}A,\ \omega \in (0,1]$	$1 - \frac{\omega}{2}\lambda_i$	$1 - \frac{\omega}{2}\lambda_{min}$ $= 1 - 2\omega\ \sin^2(\frac{\pi h}{2})$	$1 - \frac{\omega\pi^2h^2}{2}$
	$\omega_{opt}^{\text{JOR}} = \dfrac{2}{2 - \mu_1 - \mu_n} = 1$ $\mu_n < \mu_1 = 1 - \frac{1}{2}\lambda_{min} < 1$		$1 - 2\sin^2(\frac{\pi h}{2})$	$1 - \frac{\pi^2h^2}{2}$
SOR	$H_\omega = I - \omega(D - \omega E)^{-1}A$ $\omega = \dfrac{2}{1 + \sqrt{\gamma\Gamma}}$ $\gamma = \mathcal{O}(h^2),\ \Gamma = \mathcal{O}(1)$	$\omega < 1 \to$ $\mu \in (0, \mu_1)$ $\omega \in [1, 1+\delta]$ $\to \mu \in [0, \mu_1^2]$ $\omega > 1+\delta \to$ $\mu \in \mathbb{C}, \delta \geq 0$ $\|\mu| \leq \mu_1^2$	$\|H_\omega\|_A$ $= \sqrt{\dfrac{\sqrt{\Gamma} - \sqrt{\gamma}}{\sqrt{\Gamma} + \sqrt{\gamma}}}$	$1 - \|\mathcal{O}(h)\|$
	$\omega_{opt} = \dfrac{2}{1 + \sqrt{1 - \mu_1^2}}$ $= \dfrac{2}{1 + \sqrt{(1-\mu_1)(1+\mu_1)}}$ $= \dfrac{2}{1 + \sin(\pi h)}$ $\approx \dfrac{2}{1 + \pi h} \in [1, 2)$ $\approx 2(1 - \pi h)$	$\mu \in \mathbb{C}$ $\|\mu| = \rho(H_{\omega_{opt}})$	$\rho(H_{\omega_{opt}}) = \omega_{opt} - 1$	$\frac{1-\pi h}{1+\pi h}$ $\approx 1 - \begin{Bmatrix}1\\2\end{Bmatrix}\pi h$
		μ EW von J, dann λ' EW von H_ω mit $(\lambda' + \omega - 1)^2 = \lambda'\omega^2\mu^2$ $\lambda'_{1,2} = 1 - \omega + \frac{1}{2}\omega^2\mu^2 \pm \omega\mu\sqrt{1 - \omega + \frac{1}{4}\omega^2\mu^2}$ $\rho(H_\omega) =$ $\begin{cases} 1 - \omega + d_1 + \omega\mu_1\sqrt{1 - \omega + \frac{1}{2}d_1}, & \omega \in (0, \omega_{opt}) \\ \omega - 1, & \omega \in [\omega_{opt}, 2) \end{cases}$ wobei μ_1 aus GSV und $d_1 = \frac{1}{2}\omega^2\mu_1^2$		
SSOR	$\tilde{H}_\omega = I - [\frac{1}{\omega(2-\omega)}(D - \omega E)\cdot$ $\cdot D^{-1}(D - \omega F)]^{-1}A$ $\omega_{opt} \approx$ wie bei SOR	$\in [0, 1)$	< 1	$1 - \|\mathcal{O}(h)\|$

Tab. 6.1 Vergleich ausgewählter IV mit Iterationsmatrix H, EW $\mu(H)$, Spektralradius $\rho(H)$ ($\approx$ heißt gerundet) sowie EW $\lambda_i(A)$, $\lambda'(H_\omega)$

Für diese Modellgleichung sind die Vorteile von SOR und SSOR gegenüber den anderen IV in Bezug auf den kleineren Spektralradius der Iterationsmatrix deutlich erkennbar. Weiterhin braucht bei gleicher Genauigkeit das ESV ungefähr die Hälfte der Iterationsschritte des GSV, was durch den Faktor 2 im gerundeten Spektralradius ausgedrückt wird.

Wir vergleichen die IV RF, GSV, JOR, ESV, SGS, SOR, SSOR und einige Tschebyscheff-beschleunigten Versionen (siehe auch [31]).

Dabei testen wir wachsende Dimensionen n. Als rechte Seite wählen wir einfach $b = h^2(1, 1, ..., 1)^T$. Der Startvektor ist der Nullvektor. Die Abbruchbedingung ist (6.32) mit der Toleranz $\varepsilon = 10^{-6}$.

Da hier der Spektralradius der Jacobi-Iterationsmatrix

$$\rho(J) \;=\; \mu_1 \;=\; 1 - \frac{1}{2}\lambda_{min} \;=\; \cos(\pi h) \;=\; \cos\left(\frac{\pi}{n+1}\right) < 1 \tag{6.38}$$

bekannt ist, wird im SOR auch der optimale Relaxationsparameter

$$\omega_{opt} \;=\; \frac{2}{1+\sqrt{1-\mu_1^2}} \;=\; \frac{2}{1+\sin(\pi h)}, \quad \mu_1^2 = \rho(H_1) < \mu_1, \tag{6.39}$$

gewählt.

n	4	8	16	32	64
$\rho(J) = \mu_1$	0.809 016	0.939 692	0.982 973	0.995 471	0.998 832
ω_{opt}	1.259 616	1.490 290	1.689 546	1.826 390	1.907 826

Tab. 6.2 Spektralradius $\rho(J)$ und optimaler Parameter ω_{opt} für SOR in Abhängigkeit von der Dimension n

Für das SSOR ist der optimale Parameter im Allgemeinen nicht bekannt, deshalb wird meistens ω_{opt} vom SOR verwendet. Andere Parameterwerte ω nahe ω_{opt} führen in Testrechnungen eher zu einer geringfügigen Vergrößerung der Iterationszahlen.

Gleiches gilt für T-SSOR. Für die T-IV haben wir außerdem die Intervallgrenzen $a,\ b$ zu wählen. Wir vergleichen u. a. die folgenden Varianten bei $n \geq 4$.

$$\begin{aligned}
&\text{T-GSV}: && a = -\cos(\pi h), && b = \cos(\pi h), && h = \tfrac{1}{n+1},\\
&\text{T-ESV}: && a = 0, && b = \cos^2(\pi h),\\
&\text{T-SGS}: && a = 0, && b = \cos^2(\pi h),\\
&\text{T-SOR}: && a = -\omega_{opt} + 1, && b = \omega_{opt} - 1,\\
& && a = 0, && b = \omega_{opt} - 1,\\
& && a = -1 + \pi h, && b = 1 - \pi h,\\
& && a = 0, && b = \cos^2(\pi h), && \cos(\pi h),
\end{aligned}$$

$$\begin{aligned}\text{T-SSOR}: \quad & a=0, \quad b=\tfrac{1-\pi h}{1+\pi h} \approx \rho(H_\omega) \text{ vom SOR},\\ & a=0, \quad b=1-2\pi h, \quad 1-\pi h, \quad 1-\tfrac{\pi h}{2}, \quad 1-\tfrac{\pi h}{4},\\ & \qquad\qquad \cos^2(\pi h), \quad \cos(\pi h),\\ & a=0, \quad b=1-\tfrac{\pi h}{2(1-h)} = 1-\tfrac{\pi}{2n}.\end{aligned}$$

Es wird das gute Konvergenzverhalten des optimalen SOR im Vergleich mit GSV und ESV bestätigt. SSOR mit seinen Parameterwerten kann die Güte von SOR nicht erreichen. Allerdings sei hier darauf hingewiesen, dass wir für das SOR u. a. den optimalen Prameter gewählt haben, während dies für SSOR nicht der Fall ist. SSOR ist aber im Allgemeinen weniger empfindlich auf Veränderungen dieser. Die mit Abstand besten Resultate werden allerdings mit T-SSOR erzielt.

Versuchen wir, die Auswertung etwas detaillierter zu machen, um damit auch die Ergebnisse in den nachfolgenden Tabellen noch zu untermauern.

Um die Tschebyscheff-Beschleunigung auf konvergente Basis-IV anzuwenden, sollten die EW der Iterationsmatrizen reell und paarweise verschieden sein sowie günstige untere und obere Schranken dafür vorliegen. Wie verhält sich dies in unserem Modellproblem?

Die Iterationsmatrix des GSV $J = I - D^{-1}A$ erfüllt $\rho(J) < 1$ und ist ähnlich zur Matrix $\tilde{J} = D^{1/2}JD^{-1/2} = I - D^{-1/2}AD^{-1/2}$, die ihrerseits symmetrisch und somit nur reelle EW hat. Dies ist eine zusätzliche Bestätigung zum Ergebnis in Tabelle 6.1, dass J nur reelle EW hat.

Die Iterationsmatrix des ESV $H_1 = I - (D-E)^{-1}A$ hat zwar nur reelle EW im Intervall $[0, \rho(J)^2]$, aber die Null-EW sind i. Allg. mehrfach und das beeinträchtigt das beschleunigte IV.

Das SOR mit $H_\omega = I - \omega(D - \omega E)^{-1}A$ hat im Bereich seiner Konvergenz bei $\omega \in (0,2)$ für Parameterwerte $\omega \leq 1$ reelle EW seiner Iterationsmatrix, in manchen Situationen z. B. bei n gerade auch für $\omega \leq 1 + \delta,\ 0 < \delta \ll 1$.

Wenn μ EW von J ist, dann ist λ' EW von H_ω mit

$$(\lambda' + \omega - 1)^2 \ = \ \lambda' \omega^2 \mu^2 \tag{6.40}$$

bzw.

$$\lambda' \ = \ 1 - \omega + \frac{1}{2}\omega^2\mu^2 \pm \omega\mu\sqrt{1 - \omega + \frac{1}{4}\omega^2\mu^2}. \tag{6.41}$$

Das heißt, für $\omega \leq 1$ ist der Radikand nicht negativ und die EW λ' sind reell. Insbesondere gilt beim ESV mit $\omega = 1$ die Beziehung $\lambda' = \mu^2$ oder $\lambda' = 0$.
Für ungerades n ist $\mu = 0$ EW von J, so dass der zugehörige EW von H_ω reell und gleich $1 - \omega$ ist. Das heißt für diesen EW

$$\lambda'(H_\omega) \ = \ \begin{cases} 1 - \omega \geq 0 & \text{für } \omega \leq 1, \\ -(\omega - 1) < 0 & \text{für } \omega > 1, \end{cases} \tag{6.42}$$

insbesondere ist $\lambda'(H_{\omega_{opt}}) = -(\omega_{opt} - 1) < 0$ zu $\mu = 0$.

Für $n = 4$ wollen wir die Schranken der reellen EW $\lambda'(H_\omega)$ (6.40) in Abhängigkeit von $\mu(J)$ für $\omega = 0(0.2)1$ angeben. Dabei sind

$$\begin{aligned} \rho(J) &= \mu_1 = \cos(\tfrac{\pi}{n+1}) = 0.809\,016\,994, \\ \rho(H_1) &= \mu_1^2 = 0.654\,508\,497, \quad \omega_{opt} = 1.259\,616\,183. \end{aligned}$$

ω	$u \le \lambda'(H_\omega) \le o$	
0	1,	1
0.2	0.667778,	0.958403
0.4	0.396285,	0.908436
0.6	0.188982,	0.846641
0.8	0.052170,	0.766715
1 (ESV)	0,	0.654509

Tab. 6.3
Schranken der reellen EW $\lambda'(H_\omega)$ für $\omega = 0(0.2)1$

Für wachsendes $\omega < 1$ nimmt die Konvergenzrate des SOR langsam zu.

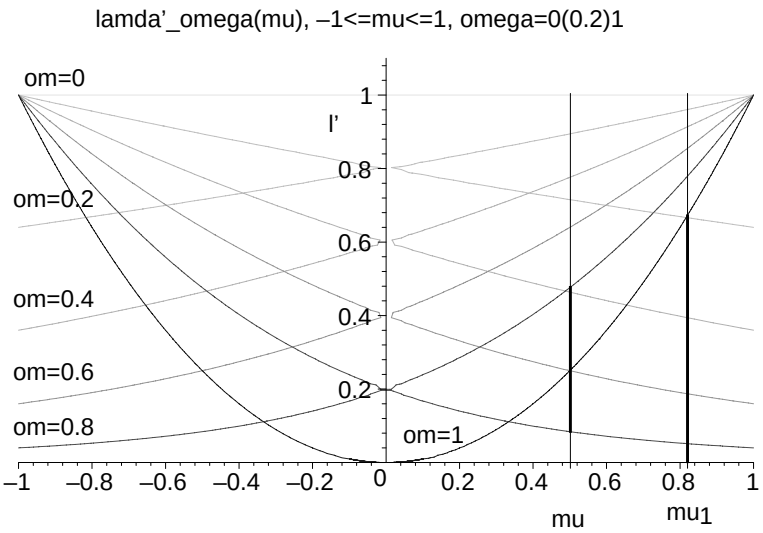


Abb. 6.1 $n = 4$, Schranken der EW $\lambda'(H_\omega)$ in Abhängigkeit von $\mu(J)$ für $\omega = 0(0.2)1$

Für größere ω werden im SOR immer mehr der EW komplex. Insbesondere haben wir bei $\omega = \omega_{opt}$ auch die Stelle, ab der die EW i. Allg. komplex und in jedem Fall betragsgleich sind und wo das beste Konvergenzverhalten mit

$$\rho(H_{\omega_{opt}}) = \omega_{opt} - 1 = \frac{1 - \sqrt{1-\mu_1^2}}{1 + \sqrt{1-\mu_1^2}} = \left(\frac{1 - \sqrt{1-\mu_1^2}}{\mu_1}\right)^2 \tag{6.43}$$

vorliegt.

Für $\mu = \pm\mu_1$ und $\omega = \omega_{opt}$ gilt $\omega_{opt} - 1 = \frac{1}{4}\omega_{opt}^2\mu_1^2$ und mit (6.40)

$$\begin{aligned}(\lambda' + \omega_{opt} - 1)^2 &= \lambda'\omega_{opt}^2\mu_1^2 \\ &= \lambda' 4(\omega_{opt} - 1),\end{aligned}$$

$$[(\omega_{opt} - 1) + (\omega_{opt} - 1)]^2 = [2(\omega_{opt} - 1)]^2 = 4(\omega_{opt} - 1)(\omega_{opt} - 1)$$

ist das zugehörige EW-Paar von $H_{\omega_{opt}}$ reell und

$$\lambda'(H_\omega) = \omega_{opt} - 1.$$

Damit versagt aber beim SOR i. Allg. die Tschebyscheff-Beschleunigung in Parameterbereich $\omega \in (1, 2)$. Für andere Werte ω treten jedoch auch Probleme auf.

Betrachten wir in der Beziehung (6.41) den betragskleinsten nicht verschwindenden EW $\mu_s(J) = 0.309\,016\,994$ für den Fall $n = 4$.
Der Radikand $1 - \omega + \frac{1}{4}\omega^2\mu^2$ ist zwar für ein anfängliches $\omega > 1$ negativ, wird jedoch bei μ nahe $\mu_s(J)$ positiv (Nulldurchgang). Somit sind der zugehörige EW von H_ω und auch alle anderen reell. Erst wenn ω eine Schranke $1 + \delta$ überschreitet, wird der Radikand $1 - \omega + \frac{1}{4}\omega^2\mu_s^2$ negativ und es entsteht ein erstes konjugiert komplexes EW-Paar von H_ω. Ihre Anzahl nimmt stetig zu mit wachsendem ω. In der Abb. 6.2 ist noch $1 - \omega + \frac{1}{4}\omega^2\mu_s^2 > 0$ für $\omega = 1.02$, aber $1 - \omega + \frac{1}{4}\omega^2\mu_s^2 < 0$ für $\omega = 1.03$.
Je kleiner dieses $\mu_s(J)$ ist, desto früher, d. h. schon für ω sehr nahe der Eins, treten komplexe EW von H_ω auf.

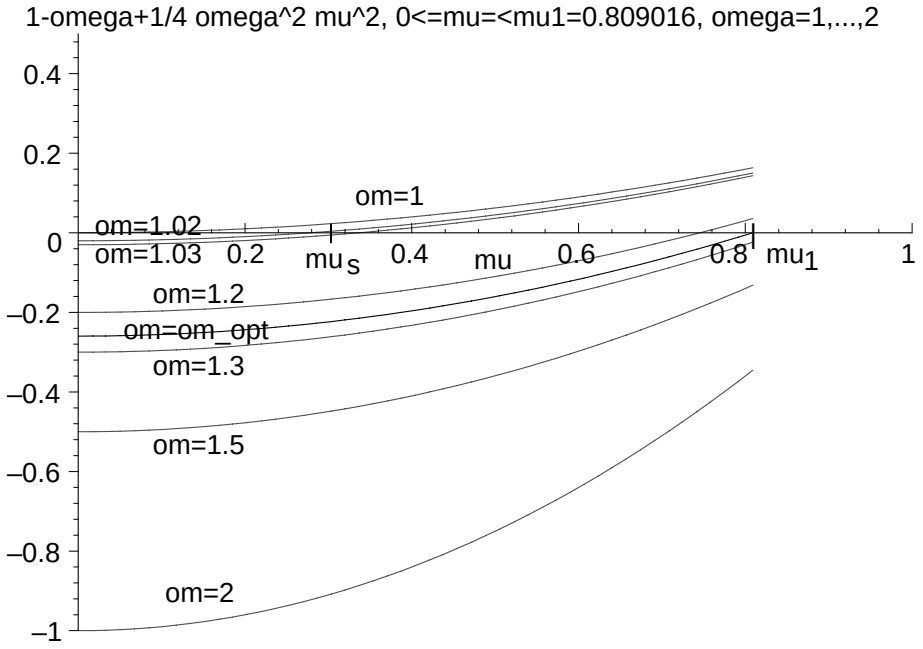


Abb. 6.2 $n = 4$, Funktion/Radikand $1 - \omega + \frac{1}{4}\omega^2\mu^2$ in der Formel der EW $\lambda'(H_\omega)$ in Abhängigkeit von $\mu(J) \in [0, \mu_1]$ für $\omega = 1, ..., 2$, Nulldurchgang der Funktion im Intervall $(0, \mu_1)$ bei $\omega \in (1, \omega_{opt})$

Das SSOR mit $\tilde{H}_\omega = I - [\frac{1}{\omega(2-\omega)}(D-\omega E)D^{-1}(D-\omega F)]^{-1}A$ hat für $\omega \in (0,2)$ nur reelle und nicht negative EW und es gilt $\tilde{H}_\omega < 1$ (vergl. [31]). Damit ist es bestens geeignet für seine Beschleunigung. Insbesondere trifft das auch für das SGS mit $\tilde{H}_1$ als Sonderfall des SSOR bei $\omega = 1$ zu.

Wir wollen noch einige EW und/oder ihre Beträge für verschiedene Dimensionen und Iterationsmatrizen tabellarisch darstellen (Angaben mit 6 Nachkommastellen).

$\mu(J)$	$\lambda(H_{0.99})$	$\lambda(H_1)$	$\lambda(H_{1.02})$	$\lambda(H_{1.03})$	$\lvert\lambda(H_{1.03})\rvert$
–0.809016	0.000151	0	0.000624	0.001421	0.001421
–0.309016	0.000887	0	0.007752	0.020653+0.021758i	0.03
0.309016	0.112703	0.095491	0.051596	0.020653 –0.021758i	0.03
0.809016	0.661332	0.654508	0.640325	0.632946	0.632946
$\rho(J)$	$\rho(H_{0.99})$	$\rho(H_1)$	$\rho(H_{1.02})$	$\rho(H_{1.03})$	
0.809016	0.661332	0.654508	0.640325	0.632946	

$\lambda(H_{1.2})$	$\lvert\lambda(H_{1.2})\rvert$	$\lambda(H_{\omega_{opt}})$	$\lambda(H_{1.3})$
0.088012	0.088012	0.259616	0.253059+0.161123i
–0.131246+0.150912i	0.2	–0.183861+0.183291i	0.253059 –0.161123i
–0.131246 –0.150912i	0.2	–0.183861 –0.183291i	–0.219309+0.204702i
0.454479	0.454479	0.259616	–0.219309 –0.204702i
$\rho(H_{1.2})$		$\rho(H_{\omega_{opt}}) = \lvert\lambda(H_{\omega_{opt}})\rvert$	$\rho(H_{1.3}) = \lvert\lambda(H_{1.3})\rvert$
0.454479		0.259616	0.3

$\lambda(\tilde{H}_1)$	$\lambda(\tilde{H}_{\omega_{opt}-0.1})$	$\lambda(\tilde{H}_{\omega_{opt}})$	$\lambda(\tilde{H}_{\omega_{opt}+0.1})$
0	0.000013	0.000413	0.003526
0.131126	0.229284	0.304322	0.386839
0.221477	0.277540	0.335287	0.405732
0.538020	0.493807	0.488591	0.505399
$\rho(\tilde{H}_1)$	$\rho(\tilde{H}_{\omega_{opt}-0.1})$	$\rho(\tilde{H}_{\omega_{opt}})$	$\rho(\tilde{H}_{\omega_{opt}+0.1})$
0.538020	0.493807	0.488591	0.505399

Tab. 6.4 $n = 4$, EW und Spektralradius der Iterationsmatrizen von Basis-IV

Bezüglich des Verhaltens der EW der Matrix H_ω in Abhängigkeit von $\omega \in (0,2)$ und speziell $\omega = \omega_{opt} = 1.259616$ findet man die gemachten Aussagen bestätigt. Das SSOR mit seinen Parametern kann die Güte von SOR nicht erreichen, der Spektralradius ist größer. Es zeigen sich jedoch die starke Empfindlichkeit des Spektralradius $\rho(H_\omega)$ von ω, dass der optimale Parameter ω_{opt} von SOR auch für das SSOR gut geeignet ist und SSOR i. Allg. weniger empfindlich auf Veränderungen von ω ist.

Für die Dimension $n = 5$ gilt

$$\mu_1 = \cos\left(\frac{\pi}{6}\right) = \frac{\sqrt{3}}{2}, \quad \mu_1^2 = \frac{3}{4}, \quad \omega_{opt} = \frac{4}{3} = 1.333333...$$

und $\mu = 0$ ist "mittlerer" EW von J.
Damit ist für $\omega > 1$ ein EW von H_ω gleich $1 - \omega < 0$ (siehe (6.42)).

$\mu(J)$	$\lambda(H_{0.99})$	$\lambda(H_1)$	$\lambda(H_{1.07})$	$\lambda(H_{1.08})$	$\lvert\lambda(H_{1.03})\rvert$
–0.866025	0.000132	0	0.052007	0.065800+0.045501i	0.08
–0.5	0.000377	0	0.006884	0.065800 –0.045501i	0.08
0	0.01	0	–0.07	–0.08	0.08
0.5	0.264647	0.25	0.094217	0.009068	0.009068
0.866025	0.754942	0.75	0.711790	0.705731	0.705731

$\rho(J)$	$\rho(H_{0.99})$	$\rho(H_1)$	$\rho(H_{1.07})$	$\rho(H_{1.08})$
0.866025	0.754942	0.75	0.711790	0.705731

$\lambda(H_{1.3})$	$\lvert\lambda(H_{1.3})\rvert$	$\lambda(H_{\omega_{opt}})$	$\lambda(H_{1.4})$
–0.088750+0.286571i	0.3	–0.111111+0.314269i	–0.155000+0.368747i
–0.088750 –0.286571i	0.3	–0.111111 –0.314269i	–0.155000 –0.368747i
–0.3	0.3	–0.333333	–0.4
0.187500	0.187500	0.333333	0.335000+0.218574i
0.48	0.48	0.333333	0.335000 –0.218574i

$\rho(H_{1.3})$	$\rho(H_{\omega_{opt}}) = \lvert\lambda(H_{\omega_{opt}})\rvert$	$\rho(H_{1.4}) = \lvert\lambda(H_{1.4})\rvert$
0.48	0.333333	0.4

$\lambda(\tilde{H}_1)$	$\lambda(\tilde{H}_{\omega_{opt}-0.1})$	$\lambda(\tilde{H}_{\omega_{opt}})$	$\lambda(\tilde{H}_{\omega_{opt}+0.1})$
0	0.000028	0.000547	0.004146
0.123010	0.280912	0.362771	0.450379
0.168944	0.298603	0.373760	0.456775
0.296273	0.352995	0.408110	0.476989
0.634428	0.565693	0.559241	0.573786

$\rho(\tilde{H}_1)$	$\rho(\tilde{H}_{\omega_{opt}-0.1})$	$\rho(\tilde{H}_{\omega_{opt}})$	$\rho(\tilde{H}_{\omega_{opt}+0.1})$
0.634428	0.565693	0.559241	0.573786

Tab. 6.5 $n = 5$, EW und Spektralradius der Iterationsmatrizen von Basis-IV

Die Bewertung der Eigenschaften der Iterationsmatrizen von Basis-IV trifft auch auf höhere Matrixdimensionen zu. Wir stellen noch einige Vergleiche für $n = 8, 16, 32, 64$ an.

$n=8$		$n=16$		$n=32$		$n=64$	
$\mu(J)$	$\lambda(H_1)$	$\mu(J)$	$\lambda(H_1)$	$\mu(J)$	$\lambda(H_1)$	$\mu(J)$	$\lambda(H_1)$
–0.939692	0	±0.092268	8×0	±0.047581	16×0	±0.024163	32×0
–0.766044	0	±0.273662	0.008513	±0.142314	0.002264	±0.072434	0.000583
–0.5	0	±0.445738	0.074891	±0.235758	0.020253	±0.120536	0.005246
–0.173648	0	±0.602634	0.198682	±0.327067	0.055582	±0.168357	0.014529
0.173648	0.25	±0.739008	...	...	...	...	...
0.5	0.301536	±0.850217	0.722869	±0.959492	0.920626	±0.981370	0.979122
0.766044	0.586824	±0.932472	0.869504	±0.981928	0.964183	±0.995331	0.990685
0.939692	0.883022	±0.982973	0.966236	±0.995471	0.990964	±0.998832	0.997665
$\rho(J)$	$\rho(H_1)$	$\rho(J)$	$\rho(H_1)$	$\rho(J)$	$\rho(H_1)$	$\rho(J)$	$\rho(H_1)$
0.939692	0.883022	0.982973	0.966236	0.995471	0.990964	0.998832	0.997665

$n=8$	$n=16$	$n=32$	$n=64$
$\omega_{opt}=1.490290$ $\lambda(H_{\omega_{opt}})$	$\omega_{opt}=1.689546$ $\lambda(H_{\omega_{opt}})$	$\omega_{opt}=1.826390$ $\lambda(H_{\omega_{opt}})$	$\omega_{opt}=1.907826$ $\lambda(H_{\omega_{opt}})$
$0.161367+0.462974i$	$-0.677395\pm 0.128879i$	$-0.822614\pm 0.078909i$	$-0.906763\pm 0.043911i$
$0.161367-0.462974i$	$-0.582655\pm 0.368764i$	$-0.792610\pm 0.233858i$	$-0.898277\pm 0.131323i$
$-0.212669+0.441765i$	$-0.405970\pm 0.557371i$	$-0.111144\pm 0.818882i$	$-0.063788\pm 0.905582i$
$-0.212669-0.441765i$	$-0.171202\pm 0.667955i$	...	...
$-0.456805+0.178083i$	$0.089941\pm 0.683655i$	$0.047214\pm 0.825040i$	$0.024111\pm 0.907506i$
$-0.456805-0.178083i$	$0.342192\pm 0.598647i$	$0.709077\pm 0.424417i$	$0.874080\pm 0.245219i$
0.490290	$0.551483\pm 0.413933i$	$0.781724\pm 0.268006i$	$0.895122\pm 0.151344i$
0.490290	2×0.689546	2×0.826390	2×0.907826
$\rho(H_{\omega_{opt}})$	$\rho(H_{\omega_{opt}})$	$\rho(H_{\omega_{opt}})$	$\rho(H_{\omega_{opt}})$
0.490290	0.689546	0.826390	0.907826

$n=8$		$n=16$		$n=32$		$n=64$	
$\lambda(\tilde{H}_1)$	$\lambda(\tilde{H}_{\omega_{opt}})$	$\lambda(\tilde{H}_1)$	$\lambda(\tilde{H}_{\omega_{opt}})$	$\lambda(\tilde{H}_1)$	$\lambda(\tilde{H}_{\omega_{opt}})$	$\lambda(\tilde{H}_1)$	$\lambda(\tilde{H}_{\omega_{opt}})$
0	0.000845	0	0.001241	0	0.001517	0	0.001681
0.115353	0.501584	0.112109	0.691765	0.111354	0.826748	0.111171	0.907877
0.129316	0.502995	0.115171	0.691831	0.112088	0.826751	0.111351	0.907878
0.157423	0.505916	...	...	...	...	...	...
0.209969	0.511728	0.479482	0.704042	0.775994	0.835344	0.930577	0.912883
0.309465	0.524409	0.619098	0.714007	0.860025	0.841651	0.959692	0.916446
0.498829	0.558535	0.785006	0.739464	0.932455	0.857379	0.981663	0.925221
0.801870	0.689131	0.936004	0.826319	0.982200	0.907816	0.995350	0.952454
$\rho(\tilde{H}_1)$	$\rho(\tilde{H}_{\omega_{opt}})$	$\rho(\tilde{H}_1)$	$\rho(\tilde{H}_{\omega_{opt}})$	$\rho(\tilde{H}_1)$	$\rho(\tilde{H}_{\omega_{opt}})$	$\rho(\tilde{H}_1)$	$\rho(\tilde{H}_{\omega_{opt}})$
0.801870	0.689131	0.936004	0.826319	0.982200	0.907816	0.995350	0.952454

Tab. 6.6 $n=8,16,32,64$, (ausgewählte) EW und Spektralradius der Iterationsmatrizen von Basis-IV

Noch einige Bemerkungen zur Anwendung der Tschebyscheff-Beschleunigung auf die konvergenten Basis-IV.

In allen Varianten ist zunächst eine geeignete Wahl der EW-Grenzen zu treffen. Die Kenntnis der EW der Iterationsmatrizen in unserem Modellproblem gestattet einige zusätzliche Testrechnungen.

Das T-GSV ist wesentlich besser als das GSV. Als EW-Grenzen bieten sich $b \gtrapprox \mu_1$ und $a \lessapprox -\mu_1$ an.

Das T-ESV scheint bei kleinen Dimensionen n mit dem sinnvollen EW-Intervall $[a,b] = [0,\mu_1^2]$ noch zu funktionieren und das ESV zu übertreffen, aber es versagt bei großen n.

T-SGS konvergiert gut bei $[a,b] = [0,\mu_1^2]$. Es kann aber noch verbessert werden, wenn die obere Schranke $b = \lambda_{max}(\tilde{H}_1)$ gewählt wird, was natürlich die Kenntnis über gewisse EW der Matrix $\tilde{H}_1$ voraussetzt.

T-GSV und T-SGS liegen bezüglich der Konvergenzrate in der Nachbarschaft von SOR mit optimalem Relaxationsparameter.

Das T-SOR hat i. Allg. wegen der komplexen EW von H_ω Konvergenzprobleme. Diese verstärken sich bei ungünstiger Wahl der EW-Grenzen a und b sowie des (eigentlich unbekannten optimalen) Relaxationsparameters ω. Dazu sind einige Varianten getestet worden. SOR nicht mit optimalen Parameter zu rechnen, um z. B. bei $\omega < 1$ reelle EW $\lambda(H_\omega)$ zu garantieren, und dann anschließend das entsprechende T-SOR zu nehmen, ist nicht besser als die Rechnung mit optimalem SOR und trotzdem gibt es mit T-SOR bei wachsender Matrixdimension erneut Konvergenzprobleme.

T-SSOR erzielt die besten Resultate mit $\omega \approx \omega_{opt}$ vom SOR.
Auch der Einfluss einer geschickten Wahl der EW-Grenzen, insbesondere der oberen Schranke, ist erkennbar. Man vergleiche dazu nur die Ergebnisse mit den Schranken $b = 1 - \frac{3}{4}\frac{\pi}{n+1}$, $b = 1 - \frac{\pi}{n+1}$ sowie $b = \lambda_{max}(\tilde{H}_{\omega_{opt}})$ bei Kenntnis der entsprechenden EW von $\tilde{H}_{\omega_{opt}}$ wie im Modellproblem.

Es gilt

$$1 - \frac{\pi}{n+1} < \lambda_{max}(\tilde{H}_{\omega_{opt}}) < 1 - \frac{3}{4}\frac{\pi}{n+1}.$$

Die Iterationsanzahlen, die man zu $b = \lambda_{max}(\tilde{H}_{\omega_{opt}})$ erhält, werden durch benachbarte b-Werte nicht unterschritten. Günstiger ist es, die obere Schranke b eher etwas größer als zu klein zu wählen. Meistens werden also die notwendigen Informationen für T-SSOR auf der Grundlage von SOR genommen.

Die geringe Empfindlichkeit von T-SSOR auf Veränderungen in der Wahl ω ist relativ zu sehen. Sehr abweichende Parameterwerte wie $\omega = 0.9$ oder 1 liefern schon sichtbar schlechtere Iterationsanzahlen.

Günstige Varianten der IV sind in der folgenden Tabelle fett hervorgehoben.

	n	4	8	16	32	64
RF	$\omega = \omega_{opt}^{\mathrm{RF}} = \frac{1}{2}$	66	222	800	3 025	11 741
GSV		66	222	800	3 025	11 741
JOR	$\omega = \omega_{opt}^{\mathrm{JOR}} = 1$	66	222	800	3 025	11 741
	$\omega = \omega_{opt}^{\mathrm{JOR}} - 0.1$	74	247	890	3 362	13 046
	$\omega = \omega_{opt}^{\mathrm{JOR}} + 0.1$	59	Divergenz			
T-GSV	$a = -\cos(\frac{\pi}{n+1}),\ b = \cos(\frac{\pi}{n+1})$	22	41	78	152	300
ESV		34	112	402	1 514	5 872
T-ESV	$a = 0,\ b = \cos(\frac{\pi}{n+1}) \approx 1 - \frac{1}{2}(\frac{\pi}{n+1})^2$	24	76	406	Divergenz	
	$a = 0,\ b = 1 - \frac{1}{2}(\frac{\pi}{n+1})^2$	23	76	434	Divergenz	
	$a = 0,\ b = \cos^2(\frac{\pi}{n+1}) \approx 1 - (\frac{\pi}{n+1})^2$	**16**	**50**	**193**	Divergenz	
	$a = 0,\ b = 1 - (\frac{\pi}{n+1})^2$	**16**	**49**	**194**	Divergenz	
	$a = 0,\ b = 1 - 2(\frac{\pi}{n+1})^2$	29	49	124	Divergenz	
	$a = 0,\ b = 1 - 4(\frac{\pi}{n+1})^2$	44	76	140	Divergenz	
	$\cos(\frac{\pi}{n+1}) > 1-\frac{1}{2}(\frac{\pi}{n+1})^2 > \cos^2(\frac{\pi}{n+1}) > 1-(\frac{\pi}{n+1})^2 > 1-2(\frac{\pi}{n+1})^2$					
SGS		23	63	208	765	2 944
T-SGS	$a = 0,\ b = \cos^2(\frac{\pi}{n+1})$	**11**	**21**	**38**	**73**	**143**
	$b =$	0.655	0.884	0.967	0.991	0.998
	$a = 0,\ b = \cos(\frac{\pi}{n+1})$	16	29	53	102	199
	$a = 0,\ b = \lambda_{max}(\tilde{H}_1)$	**9**	**16**	**27**	**56**	**114**
	von n abhängiges $\lambda_{max}(\tilde{H}_1)$	0.539	0.802	0.937	0.983	0.996
SOR	$\omega = \omega_{opt} + 0.1$	15	29	67	198	Div.
	$\omega = \omega_{opt} + 0.05$	14	26	53	124	384
	$\omega = \omega_{opt}$	**14**	**26**	**50**	**97**	**192**
	$\omega = \omega_{opt} - 0.05$	19	36	74	167	414
	$\omega = \omega_{opt} - 0.1$	22	43	93	224	601
	$\omega = 1$ (ESV)	34	112	402	1 514	5 872
	$\omega = 0.9$	42	138	492	1 851	7 177
T-SOR	$\omega = \omega_{opt}$					
	$a = -\omega_{opt} + 1,\ b = \omega_{opt} - 1$	13	33	397	Divergenz	
	$a = 0,\ b = \omega_{opt} - 1$	18	373	Divergenz		
	$a = -1 + \frac{\pi}{n+1},\ b = 1 - \frac{\pi}{n+1}$	14	48	Divergenz		
	$a = 0,\ b = \cos^2(\frac{\pi}{n+1})$	69	Divergenz			
	$a = 0,\ b = \cos(\frac{\pi}{n+1})$	Divergenz				

	n	4	8	16	32	64
T-SOR	$\omega = 1, \quad a = 0,\ b = \cos^2(\frac{\pi}{n+1})$	16	50	193	Divergenz	
	$\omega = 0.9, \quad a = 0,\ b = \cos^2(\frac{\pi}{n+1})$	19	36	99	Divergenz	
	$\omega = 0.9, \quad a = 0,\ b = \cos(\frac{\pi}{n+1})$	17	50	154	Divergenz	
SSOR	$\omega = \omega_{opt} + 0.1$	21	40	83	221	Div.
	$\omega = \omega_{opt} + 0.05$	20	39	76	160	413
	$\omega = \omega_{opt}$	**20**	**38**	**74**	**148**	**297**
	$\omega = \omega_{opt} - 0.05$	20	38	75	152	321
	$\omega = \omega_{opt} - 0.1$	20	39	78	164	380
	$\omega = 1$ (SGS)	23	63	208	765	2 944
T-SSOR	$\omega = \omega_{opt}$					
	$a = 0,\ b = \cos(\pi h) \approx 1 - \frac{1}{2}(\frac{\pi}{n+1})^2$	16	30	55	108	205
	$a = 0,\ b = \cos^2(\pi h) \approx 1 - (\frac{\pi}{n+1})^2$	11	21	39	79	155
	$a = 0,\ b = 1 - \frac{\pi h}{4} = 1 - \frac{\pi}{4(n+1)}$	18	23	35	47	66
	$a = 0,\ b = 1 - \frac{\pi h}{2} = 1 - \frac{\pi}{2(n+1)}$	12	16	23	34	48
	$a = 0,\ b = 1 - \frac{\pi h}{2(1-h)} = 1 - \frac{\pi}{2n}$	11	16	23	34	48
	$a = 0,\ b = 1 - \frac{3}{4}\pi h = 1 - \frac{3}{4}\frac{\pi}{n+1}$	**9**	**14**	**18**	**28**	**40**
	$b =$	0.528	0.738	0.861	0.928	0.963
	$a = 0,\ b = 1 - \pi h = 1 - \frac{\pi}{n+1}$	**13**	**16**	**21**	**28**	**37**
	$b =$	0.371	0.650	0.815	0.904	0.951
	$a = 0,\ b = \frac{1-\pi h}{1+\pi h} = 1 - \frac{2\pi}{n+1+\pi}$	16	24	36	53	78
	$a = 0,\ b = 1 - 2\pi h = 1 - \frac{2\pi}{n+1}$	23	30	41	57	80
	$\cos(\pi h) > \cos^2(\pi h) > 1-(\pi h)^2 > 1-\frac{\pi h}{4} > 1-\frac{\pi h}{2} > 1-\pi h > 1-2\pi h$					
	$\omega = \omega_{opt}, \quad a = 0,\ b = \lambda_{max}(\tilde{H}_{\omega_{opt}})$	**9**	**12**	**17**	**25**	**36**
	von n abhängiges $\lambda_{max}(\tilde{H}_{\omega_{opt}})$	0.489	0.690	0.827	0.908	0.953
	$\omega = \omega_{opt}+0.1, \quad a = 0,\ b = 1-\frac{\pi}{n+1}$	14	18	26	55	Div.
	$\omega = \omega_{opt}+0.05, \quad a = 0,\ b = 1-\frac{\pi}{n+1}$	14	17	22	33	70
	$\omega = \omega_{opt}-0.05, \quad a = 0,\ b = 1-\frac{\pi}{n+1}$	13	17	22	30	48
	$\omega = \omega_{opt}-0.1, \quad a = 0,\ b = 1-\frac{\pi}{n+1}$	14	17	23	36	64
	$\omega = 1, \quad a = 0,\ b = 1-\frac{\pi}{n+1}$	16	33	83	226	632
	$\omega = 0.9, \quad a = 0,\ b = 1-\frac{\pi}{n+1}$	18	40	101	277	776

Tab. 6.7 Anzahl der Iterationen der verschiedenen IV für die diskretisierte RWA, $\varepsilon = 10^{-6}$ in (6.32), Startvektor ist Nullvektor

6.2 Auf Polynomen basierende Iterationen

Die Abstiegsverfahren in den Teilen I und II der Reihe *Abstiegsverfahren* (Preprint No. M 19/04, 20/04) werden zunächst als polynomiale IV dargestellt.

6.2.1 GV

Wir notieren die Formeln kompakt gemäß Teil I Abschnitt 3.2 der Reihe *Abstiegsverfahren* (Preprint No. M 19/04).

$$\begin{aligned} & x^{(0)}, \quad r^{(0)} = b - Ax^{(0)}, \\ x^{(k+1)} &= x^{(k)} + \alpha_k r^{(k)}, \quad \alpha_k = \frac{r^{(k)T} r^{(k)}}{r^{(k)T} A r^{(k)}}, \\ & r^{(k+1)} = b - Ax^{(k+1)} = A(x^* - x^{(k+1)}) = Ae^{(k+1)}, \\ r^{(k+1)} &= r^{(k)} - \alpha_k A r^{(k)} \;=\; (I - \alpha_k A) r^{(k)}. \end{aligned} \tag{6.44}$$

Für die ersten Schritte erhält man die folgenden Darstellungen mit den Polynomen $q_i(A)$ und $p_i(A)$.

1.
$$\begin{aligned} x^{(1)} &= x^{(0)} + \alpha_0 r^{(0)} = x^{(0)} + \alpha_0 I r^{(0)} \\ &= x^{(0)} + q_0(A) r^{(0)}, \\ & \quad q_0(A) = \alpha_0 I, \end{aligned}$$

$$\begin{aligned} r^{(1)} &= b - Ax^{(1)} = b - A(x^{(0)} + q_0(A) r^{(0)}) = r^{(0)} - A q_0(A) r^{(0)} \\ &= [I - A q_0(A)] r^{(0)} \\ &= p_1(A) r^{(0)}, \\ & \quad p_1(A) = I - A q_0(A) = I - \alpha_0 A. \end{aligned}$$

2.
$$\begin{aligned} x^{(2)} &= x^{(1)} + \alpha_1 r^{(1)} = x^{(0)} + \alpha_0 r^{(0)} + \alpha_1 (I - \alpha_0 A) r^{(0)} \\ &= x^{(0)} + [(\alpha_0 + \alpha_1) I - \alpha_0 \alpha_1 A] r^{(0)} \\ &= x^{(0)} + q_1(A) r^{(0)}, \\ & \quad q_1(A) = (\alpha_0 + \alpha_1) I - \alpha_0 \alpha_1 A, \end{aligned}$$

$$\begin{aligned} r^{(2)} &= b - Ax^{(2)} = b - A(x^{(0)} + q_1(A) r^{(0)}) = r^{(0)} - A q_1(A) r^{(0)} \\ &= [I - A q_1(A)] r^{(0)} \\ &= p_2(A) r^{(0)}, \\ & \quad p_2(A) = I - A q_1(A) = I - (\alpha_0 + \alpha_1) A + \alpha_0 \alpha_1 A^2. \end{aligned}$$

3.
$$\begin{aligned}
x^{(3)} &= x^{(2)} + \alpha_2 r^{(2)} = x^{(0)} + q_1(A)r^{(0)} + \alpha_2[I - Aq_1(A)]r^{(0)} \\
&= x^{(0)} + [(\alpha_0+\alpha_1+\alpha_2)I - (\alpha_0\alpha_1+\alpha_0\alpha_2+\alpha_1\alpha_2)A + \alpha_0\alpha_1\alpha_2A^2]r^{(0)} \\
&= x^{(0)} + q_2(A)r^{(0)}, \\
&\quad q_2(A) = (\alpha_0+\alpha_1+\alpha_2)I - (\alpha_0\alpha_1+\alpha_0\alpha_2+\alpha_1\alpha_2)A + \alpha_0\alpha_1\alpha_2A^2, \\
r^{(3)} &= b - Ax^{(3)} = b - A(x^{(0)} + q_2(A)r^{(0)}) = r^{(0)} - Aq_2(A)r^{(0)} \\
&= [I - Aq_2(A)]r^{(0)} \\
&= p_3(A)r^{(0)}, \\
&\quad p_3(A) = I - Aq_2(A) \\
&\qquad = I-(\alpha_0+\alpha_1+\alpha_2)A+(\alpha_0\alpha_1+\alpha_0\alpha_2+\alpha_1\alpha_2)A^2-\alpha_0\alpha_1\alpha_2A^3.
\end{aligned}$$

4. Allgemein

$$\begin{aligned}
x^{(k)} &= x^{(0)} + q_{k-1}(A)r^{(0)}, \quad q_{k-1}(A) \in \mathcal{P}_{k-1}(A), \\
r^{(k)} &= p_k(A)r^{(0)}, \quad p_k(A) = I - Aq_{k-1}(A) \in \mathcal{P}_k(A),\ p_k(0) = I.
\end{aligned} \tag{6.45}$$

Betrachtung als reelle Polynome:

$$\begin{aligned}
q_0(x) &= \alpha_0, \\
q_1(x) &= \alpha_0 + \alpha_1 - \alpha_0\alpha_1 x, \\
q_2(x) &= \alpha_0 + \alpha_1 + \alpha_2 - (\alpha_0\alpha_1 + \alpha_0\alpha_2 + \alpha_1\alpha_2)x + \alpha_0\alpha_1\alpha_2x^2, \\
q_{k-1}(x) &\in \mathcal{P}_{k-1}(x), \quad q_{k-1}(0) = \sum_{i=0}^{k-1}\alpha_i, \\
p_0(x) &= 1, \\
p_1(x) &= 1 - \alpha_0 x, && p_1(0) = 1, \\
p_2(x) &= 1 - (\alpha_0+\alpha_1)x + \alpha_0\alpha_1x^2, && p_2(0) = 1, \\
p_3(x) &= 1-(\alpha_0+\alpha_1+\alpha_2)x+(\alpha_0\alpha_1+\alpha_0\alpha_2+\alpha_1\alpha_2)x^2-\alpha_0\alpha_1\alpha_2x^3, && p_3(0) = 1, \\
p_k(x) &= 1 - xq_{k-1}(x) \in \mathcal{P}_k(x),\ p_k(0) = 1,\ \text{d. h. } p_k(x) \in \mathcal{P}_k^1(x).
\end{aligned}$$

Betrachtung im Zusammenhang mit Vektorräumen und Krylov-Unterraum $\mathcal{K}_k(A, r^{(0)})$:

$$\begin{aligned}
q_{k-1}(A)r^{(0)} &= \tilde{\alpha}_0 Ir^{(0)} + \tilde{\alpha}_1 Ar^{(0)} + ... + \tilde{\alpha}_{k-1}A^{k-1}r^{(0)} \\
&= \left(\sum_{i=0}^{k-1}\tilde{\alpha}_i A^i\right) r^{(0)} \\
&\in \operatorname{span}\{r^{(0)}, Ar^{(0)}, ..., A^{k-1}r^{(0)}\} = \mathcal{K}_k(A, r^{(0)}),
\end{aligned}$$

$$
\begin{aligned}
x^{(k)} &= x^{(k-1)} + \alpha_{k-1} r^{(k-1)} \\
&= x^{(0)} + \alpha_0 r^{(0)} + \alpha_1 r^{(1)} + ... + \alpha_{k-1} r^{(k-1)} \\
&\in x^{(0)} + \mathcal{R}_k, \quad \mathcal{R}_k = \text{span}\{r^{(0)}, r^{(1)}, ..., r^{(k-1)}\}, \\
x^{(k)} &= x^{(0)} + q_{k-1}(A) r^{(0)} \ \in \ x^{(0)} + \mathcal{K}_k(A, r^{(0)}), \\
r^{(k)} &= p_k(A) r^{(0)} \ = \ [I - A q_{k-1}(A)] r^{(0)} \ \in \ \mathcal{K}_{k+1}(A, r^{(0)}), \\
e^{(k)} &= x^* - x^{(k)} \ = \ A^{-1}(b - A x^{(k)}) \ = \ A^{-1} r^{(k)} \\
&= A^{-1} p_k(A) r^{(0)} \ = \ A^{-1} p_k(A) A \, A^{-1} r^{(0)} \ = \ p_k(A) e^{(0)} \\
&= [I - A q_{k-1}(A)] e^{(0)} = e^{(0)} - q_{k-1}(A) \, A e^{(0)} = e^{(0)} - q_{k-1}(A) \, r^{(0)} \\
&= e^{(0)} - \left(\sum_{i=0}^{k-1} \tilde{\alpha}_i A^i \right) r^{(0)} \ \in \ e^{(0)} + \mathcal{K}_k(A, r^{(0)}).
\end{aligned}
\tag{6.46}
$$

In den Beziehungen $x^{(k)} = x^{(0)} + q_{k-1}(A) r^{(0)}$ kann man $q_{k-1}(A)$ als Iterationspolynom (iteration polynomial) und in $r^{(k)} = p_k(A) r^{(0)}$ den Term $p_k(A)$ als Residuumpolynom (residual polynomial) bezeichnen.
Zusammen ergibt sich das auf Polynomen basierende AV (polynomial based method).

$$
\boxed{
\begin{aligned}
x^{(k)} &= x^{(0)} + q_{k-1}(A) r^{(0)}, \quad q_{k-1}(A) \in \mathcal{P}_{k-1}(A) \\
r^{(k)} &= p_k(A) r^{(0)}, \quad p_k(A) \in \mathcal{P}_k(A) \\
& \qquad p_k(x) = 1 - x q_{k-1}(x) \in \mathcal{P}_k^1(x)
\end{aligned}
}
\tag{6.47}
$$

Das GV beruht auf der Minimierung des Funktionals

$$
Q(x) \ = \ \frac{1}{2} x^T A x - x^T b \ = \ \frac{1}{2} \Big(\|e(x)\|_A^2 - x^{*T} b \Big) \tag{6.48}
$$

und damit der von der A-Norm $\|e(x)\|_A$, d. h. "minimale Fehlereigenschaft" (minimal error property, minimal energy norm) und bedeutet

$$
\begin{aligned}
\|e^{(k)}\|_A \ = \ \|e(x^{(k)})\|_A &= \|x^* - x^{(k)}\|_A \\
&= \|p_k(A) e^{(0)}\|_A \\
&= \min_{\substack{p(x) \in \mathcal{P}_k(x) \\ p(0) = 1}} \|p(A) e^{(0)}\|_A
\end{aligned}
\tag{6.49}
$$

mit

$$
\begin{aligned}
\|e^{(k)}\|_A^2 &= (p_k(A) e^{(0)})^T A p_k(A) e^{(0)} \\
&= e^{(0)T} p_k(A) \, A p_k(A) e^{(0)} \\
&= (p_k(A), p_k(A))_{GV} \\
&= (p_k, p_k)_{GV},
\end{aligned}
\tag{6.50}
$$

wobei in der letzten Beziehung das modifizierte Skalarprodukt

$$(p,q)_{GV} \;=\; e^{(0)T} p(A)\, Aq(A) e^{(0)} \tag{6.51}$$

definiert wurde. In der Minimierungsstrategie (6.49) widerspiegelt sich das gute Konvergenzverhalten des AV. Sowohl dort als in der Formel für $r^{(k)}$ tritt das Residuumpolynom $p_k(A)$ als "Wachstumsfaktor" auf.
Natürlich ist in (6.50) die praktische Nutzung wegen der Verwendung der exakten Lösung x^* sehr eingeschränkt.
Man erhält jedoch damit eine äquivalente Formulierung zur Problemstellung des GV.

> Finde ein solches Polynom $p_k(x) \in \mathcal{P}_k^1(x)$, so dass
> $$(p_k, p_k)_{GV} \;\leq\; (p,p)_{GV}$$
> für alle $p(x) \in \mathcal{P}_k^1(x)$ gilt.

(6.52)

Auf diese Art haben wir den Zugang zur Minimierungsaufgabe über die Möglichkeit der Betrachtung von Systemen von Polynomen und später der damit verbundenen Ermittlung im Rahmen von Drei-Term-Rekursionen gemacht.
Da das GV im Allgemeinen ein unendliches IV ist, Orthogonalitätseigenschaften nur schwach ausgeprägt sind und die Dimensionen der Unterräume nicht stetig wachsen müssen, verzichten wir hier auf weitere Untersuchungen dazu.

6.2.2 CG

Die Vorgehensweise lehnt sich sehr an das GV an.
Wir notieren die Formeln kompakt gemäß Teil I Abschnitt 3.6 der Reihe *Abstiegsverfahren* (Preprint No. M 19/04).

$$\begin{aligned}
& x^{(0)}, \quad r^{(0)} = b - Ax^{(0)}, \quad p^{(0)} = r^{(0)}, \\
x^{(k+1)} \;&=\; x^{(k)} + \alpha_k p^{(k)}, \quad \alpha_k = \frac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}, \; k = 0, 1, ..., \\
& r^{(k+1)} = b - Ax^{(k+1)} = A(x^* - x^{(k+1)}) = Ae^{(k+1)}, \\
r^{(k+1)} \;&=\; r^{(k)} - \alpha_k A p^{(k)}, \\
p^{(k+1)} \;&=\; r^{(k+1)} + \beta_k p^{(k)}, \quad \beta_k = \frac{r^{(k+1)T} r^{(k+1)}}{r^{(k)T} r^{(k)}}.
\end{aligned} \tag{6.53}$$

Man sieht, dass beim CG zwei gekoppelte Zwei-Term-Rekursionen verwendet werden: eine für die Berechnung des Residuumvektors unter Benutzung einer Suchrichtung und eine für die Berechnung der nächsten Suchrichtung unter Benutzung eines neu bestimmten Residuums.

Für die ersten Schritte erhält man die folgenden Darstellungen mit den Polynomen $q_i(A),\ p_i(A)$ und $\tilde{p}_i(A)$.

1.
$$\begin{aligned} x^{(1)} &= x^{(0)} + \alpha_0 p^{(0)} = x^{(0)} + \alpha_0 I r^{(0)} \\ &= x^{(0)} + q_0(A) r^{(0)}, \\ &\quad q_0(A) = \alpha_0 I, \end{aligned}$$

$$\begin{aligned} r^{(1)} &= b - Ax^{(1)} = b - A(x^{(0)} + q_0(A) r^{(0)}) = r^{(0)} - A q_0(A) r^{(0)} \\ &= [I - A q_0(A)] r^{(0)} \\ &= p_1(A) r^{(0)}, \\ &\quad p_1(A) = I - A q_0(A) = I - \alpha_0 A, \end{aligned}$$

$$\begin{aligned} p^{(1)} &= r^{(1)} + \beta_0 p^{(0)} = p_1(A) r^{(0)} + \beta_0 r^{(0)} \\ &= [\beta_0 I + p_1(A)] r^{(0)} \\ &= [(1+\beta_0) I - \alpha_0 A] r^{(0)}, \\ &\quad \tilde{p}_1(A) = (1+\beta_0) I - \alpha_0 A, \quad \tilde{p}_0(A) = I. \end{aligned}$$

2.
$$\begin{aligned} x^{(2)} &= x^{(1)} + \alpha_1 p^{(1)} \\ &= x^{(0)} + q_0(A) r^{(0)} + \alpha_1 [\beta_0 I + p_1(A)] r^{(0)} \\ &= x^{(0)} + [q_0(A) + \alpha_1 \beta_0 I + \alpha_1 p_1(A)] r^{(0)} \\ &= x^{(0)} + [(\alpha_0 + \alpha_1 \beta_0 + \alpha_1) I - \alpha_0 \alpha_1 A] r^{(0)} \\ &= x^{(0)} + q_1(A) r^{(0)} \\ &\quad q_1(A) = (\alpha_0 + \alpha_1 (1+\beta_0)) I - \alpha_0 \alpha_1 A, \end{aligned}$$

$$\begin{aligned} r^{(2)} &= b - Ax^{(2)} = b - A(x^{(0)} + q_1(A) r^{(0)}) = r^{(0)} - A q_1(A) r^{(0)} \\ &= [I - A q_1(A)] r^{(0)} \\ &= p_2(A) r^{(0)}, \\ &\quad p_2(A) = I - A q_1(A) = I - (\alpha_0 + \alpha_1 (1+\beta_0)) A + \alpha_0 \alpha_1 A^2, \end{aligned}$$

$$\begin{aligned} p^{(2)} &= r^{(2)} + \beta_1 p^{(1)} = p_2(A) r^{(0)} + \beta_1 [\beta_0 I + p_1(A)] r^{(0)} \\ &= [\beta_0 \beta_1 I + \beta_1 p_1(A) + p_2(A)] r^{(0)} \\ &= [(1 + \beta_1 (1+\beta_0)) I - (\alpha_0 (1+\beta_1) + \alpha_1 (1+\beta_0)) A + \alpha_0 \alpha_1 A^2] r^{(0)}, \\ &\quad \tilde{p}_2(A) = (1 + \beta_1 (1+\beta_0)) I - (\alpha_0 (1+\beta_1) + \alpha_1 (1+\beta_0)) A + \alpha_0 \alpha_1 A^2. \end{aligned}$$

3. Allgemein

$$\begin{aligned} x^{(k)} &= x^{(0)} + q_{k-1}(A) r^{(0)}, \quad q_{k-1}(A) \in \mathcal{P}_{k-1}(A), \\ r^{(k)} &= p_k(A) r^{(0)}, \quad p_k(A) = I - A q_{k-1}(A) \in \mathcal{P}_k(A),\ p_k(0) = I, \\ p^{(k)} &= r^{(k)} + \beta_{k-1} p^{(k-1)} = \tilde{p}_k(A) r^{(0)}, \quad \tilde{p}_k(A) \in \mathcal{P}_k(A). \end{aligned} \tag{6.54}$$

Betrachtung als reelle Polynome:

$$\begin{aligned}
q_0(x) &= \alpha_0, \\
q_1(x) &= \alpha_0 + \alpha_1(1+\beta_0) - \alpha_0\alpha_1 x, \\
q_{k-1}(x) &\in \mathcal{P}_{k-1}(x), \\
\\
p_0(x) &= 1, \\
p_1(x) &= 1 - \alpha_0 x, & p_1(0) &= 1, \\
p_2(x) &= 1 - (\alpha_0 + \alpha_1(1+\beta_0))x + \alpha_0\alpha_1 x^2, & p_2(0) &= 1, \\
p_k(x) &= 1 - x q_{k-1}(x) \in \mathcal{P}_k^1(x).
\end{aligned}$$

Betrachtung im Zusammenhang mit den Unterräumen $\mathcal{R}_k = \text{span}\{r^{(0)}, r^{(1)}, ..., r^{(k-1)}\}$, $\mathcal{P}_k = \text{span}\{p^{(0)}, p^{(1)}, ..., p^{(k-1)}\}$ (nicht verwechseln mit den Polynomräumen $\mathcal{P}_k(A)$ bzw. $\mathcal{P}_k(x)$) und dem Krylov-Unterraum $\mathcal{K}_k(A, r^{(0)}) = \text{span}\{r^{(0)}, Ar^{(0)}, ..., A^{k-1}r^{(0)}\}$:

$$\begin{aligned}
q_{k-1}(A)r^{(0)} &= \tilde{\alpha}_0 I r^{(0)} + \tilde{\alpha}_1 A r^{(0)} + ... + \tilde{\alpha}_{k-1} A^{k-1} r^{(0)} = \left(\sum_{i=0}^{k-1} \tilde{\alpha}_i A^i\right) r^{(0)} \\
&\in \mathcal{K}_k(A, r^{(0)}), \\
x^{(k)} &= x^{(k-1)} + \alpha_{k-1} p^{(k-1)} \\
&= x^{(0)} + \alpha_0 p^{(0)} + \alpha_1 p^{(1)} + ... + \alpha_{k-1} p^{(k-1)} \\
&\in x^{(0)} + \mathcal{P}_k, \\
x^{(k)} &= x^{(0)} + q_{k-1}(A) r^{(0)} \in x^{(0)} + \mathcal{K}_k(A, r^{(0)}) = x^{(0)} + K_k, \\
r^{(k)} &= p_k(A) r^{(0)} = [I - A q_{k-1}(A)] r^{(0)} \in \mathcal{K}_{k+1}(A, r^{(0)}), \\
p^{(k)} &= r^{(k)} + \beta_{k-1} p^{(k-1)} = \tilde{p}_k(A) r^{(0)} \in \mathcal{K}_{k+1}(A, r^{(0)}), \\
e^{(k)} &= x^* - x^{(k)} = A^{-1}(b - A x^{(k)}) = A^{-1} r^{(k)} \\
&= A^{-1} p_k(A) r^{(0)} = A^{-1} p_k(A) A\, A^{-1} r^{(0)} = p_k(A) e^{(0)} \\
&= [I - A q_{k-1}(A)] e^{(0)} = e^{(0)} - q_{k-1}(A)\, A e^{(0)} = e^{(0)} - q_{k-1}(A)\, r^{(0)} \\
&= e^{(0)} - \left(\sum_{i=0}^{k-1} \tilde{\alpha}_i A^i\right) r^{(0)} \in e^{(0)} + \mathcal{K}_k(A, r^{(0)}).
\end{aligned} \tag{6.55}$$

Es gilt weiterhin

$$K_k = \mathcal{K}_k = \mathcal{K}_k(A, r^{(0)}) = \mathcal{R}_k = \mathcal{P}_k, \quad r^{(k)} \perp L_k = \mathcal{K}_k(A, r^{(0)}). \tag{6.56}$$

Das CG ist eine orthogonale Krylov-Unterraum-Methode mit $K_k = L_k = \mathcal{K}_k(A, r^{(0)})$.

Die mit (6.55), (6.56) verbundene Bedingung

$$r^{(k)} \perp \mathcal{R}_k \;\Leftrightarrow\; (r^{(k)}, r^{(j)}) = 0,\; j<k \;\Leftrightarrow\; (p^{(k)}, Ap^{(j)}) = 0,\; j<k \;\Leftrightarrow\; p^{(k)} \perp A\mathcal{P}_k \quad (6.57)$$

wird im ersten Teil als Galerkin-Bedingung bezeichnet. Sie gestattet eine effiziente Implementierung des Algorithmus.
Mit den Beziehungen $x^{(k)} = x^{(0)} + q_{k-1}(A)r^{(0)}$ und $r^{(k)} = p_k(A)r^{(0)}$ ergibt sich das auf Polynomen basierende AV (polynomial based method).

$$\begin{aligned}
x^{(k)} &= x^{(0)} + q_{k-1}(A)r^{(0)}, \quad q_{k-1}(A) \in \mathcal{P}_{k-1}(A) \\
r^{(k)} &= p_k(A)r^{(0)}, \quad p_k(A) \in \mathcal{P}_k(A) \\
&\quad p_k(x) = 1 - xq_{k-1}(x) \in \mathcal{P}_k^1(x) \\
p^{(k)} &= r^{(k)} + \beta_{k-1}p^{(k-1)} = \tilde{p}_k(A)r^{(0)}, \quad \tilde{p}_k(A) \in \mathcal{P}_k(A) \\
&\quad \tilde{p}_k(x) = p_k(x) + \beta_{k-1}\tilde{p}_{k-1}(x) \in \mathcal{P}_k(x) \\
&\quad \tilde{p}_k(0) = 1 + \beta_{k-1}\{1 + \beta_{k-2}[... + \beta_1(1 + \beta_0)]\}
\end{aligned} \quad (6.58)$$

Das CG beruht ebenfalls auf der Minimierung des Funktionals (6.48) und damit der von der A-Norm $\|e(x)\|_A$, d. h. "minimale Fehlereigenschaft" (minimal error property, minimal energy norm) und liefert die Bedingung (6.49).
Analog zur A-Norm

$$\|e^{(k)}\|_A^2 \;=\; e^{(0)T} p_k(A)\, Ap_k(A)e^{(0)} \;=\; (p_k, p_k)_{GAL} \quad (6.59)$$

(vergleiche (6.50)) betrachtet man auch die euklidischen Normen

$$\begin{aligned}
\|e^{(k)}\|_2^2 &= (p_k(A)e^{(0)})^T\, p_k(A)e^{(0)} \;=\; e^{(0)T} p_k(A)\, p_k(A)e^{(0)} \\
&= (p_k(A), p_k(A))_{CGe} \;=\; (p_k, p_k)_{CGe}
\end{aligned} \quad (6.60)$$

bzw.

$$\begin{aligned}
\|r^{(k)}\|_2^2 &= (p_k(A)r^{(0)})^T\, p_k(A)r^{(0)} \;=\; r^{(0)T} p_k(A)\, p_k(A)r^{(0)} \\
&= (p_k(A), p_k(A))_{CGr} \;=\; (p_k, p_k)_{CGr}.
\end{aligned} \quad (6.61)$$

In der Beziehung (6.59) erscheint das modifizierte Skalarprodukt

$$(p, q)_{GAL} \;=\; e^{(0)T} p(A) Aq(A) e^{(0)}. \quad (6.62)$$

Man erhält damit eine äquivalente Formulierung zur Problemstellung des CG.

Finde ein solches Polynom $p_k(x) \in \mathcal{P}_k^1(x)$, so dass
$$(p_k, p_k)_{GAL} \;\leq\; (p, p)_{GAL}$$
für alle $p(x) \in \mathcal{P}_k^1(x)$ gilt. (6.63)

6.2.3 CR

Wir notieren die Formeln kompakt gemäß Teil II Abschnitt 4.3 der Reihe *Abstiegsverfahren* (Preprint No. M 20/04).

$$
\begin{aligned}
& x^{(0)}, \quad r^{(0)} = b - Ax^{(0)}, \quad p^{(0)} = r^{(0)}, \\
x^{(k+1)} &= x^{(k)} + \alpha_k p^{(k)}, \quad \alpha_k = \frac{r^{(k)T} A r^{(k)}}{(Ap^{(k)})^T A p^{(k)}}, \ k = 0, 1, ..., \\
& r^{(k+1)} = b - Ax^{(k+1)} = A(x^* - x^{(k+1)}) = Ae^{(k+1)}, \\
r^{(k+1)} &= r^{(k)} - \alpha_k A p^{(k)}, \\
p^{(k+1)} &= r^{(k+1)} + \beta_k p^{(k)}, \quad \beta_k = \frac{r^{(k+1)T} A r^{(k+1)}}{r^{(k)T} A r^{(k)}}.
\end{aligned}
\tag{6.64}
$$

Die ersten Schritte kann man wie im CG nachvollziehen, aber mit den veränderten Schrittzahlen, und man erhält analoge Darstellungen mit den Polynomen $q_i(A)$, $p_i(A)$ und $\tilde{p}_i(A)$.
Allgemein gelten auch hier die Beziehungen (6.53), aus denen sich die entsprechenden reellen Polynome $q_{k-1}(x) \in \mathcal{P}_{k-1}(x)$ und $p_k(x) = 1 - xq_{k-1}(x) \in \mathcal{P}_k^1(x)$ ergeben.

Betrachtung des CR im Zusammenhang mit den Unterräumen $\mathcal{R}_k$, $\mathcal{P}_k$ und dem Krylov-Unterraum $\mathcal{K}_k(A, r^{(0)})$:

$$
\begin{aligned}
q_{k-1}(A)r^{(0)} &= \tilde{\alpha}_0 I r^{(0)} + \tilde{\alpha}_1 A r^{(0)} + ... + \tilde{\alpha}_{k-1} A^{k-1} r^{(0)} \ \in \ \mathcal{K}_k(A, r^{(0)}), \\
x^{(k)} &= x^{(0)} + \alpha_0 p^{(0)} + \alpha_1 p^{(1)} + ... + \alpha_{k-1} p^{(k-1)} \ \in \ x^{(0)} + \mathcal{P}_k, \\
x^{(k)} &= x^{(0)} + q_{k-1}(A) r^{(0)} \ \in \ x^{(0)} + \mathcal{K}_k(A, r^{(0)}) \ = \ x^{(0)} + K_k, \\
r^{(k)} &= p_k(A) r^{(0)} \ = \ [I - Aq_{k-1}(A)] r^{(0)} \ \in \ \mathcal{K}_{k+1}(A, r^{(0)}), \\
p^{(k)} &= r^{(k)} + \beta_{k-1} p^{(k-1)} \ = \ \tilde{p}_k(A) r^{(0)} \ \in \ \mathcal{K}_{k+1}(A, r^{(0)}), \\
e^{(k)} &= A^{-1} r^{(k)} \ = \ p_k(A) e^{(0)} \ \in \ e^{(0)} + \mathcal{K}_k(A, r^{(0)}).
\end{aligned}
\tag{6.65}
$$

Es gilt weiterhin

$$
K_k = \mathcal{K}_k = \mathcal{K}_k(A, r^{(0)}) = \mathcal{R}_k = \mathcal{P}_k, \quad r^{(k)} \perp L_k = A\,\mathcal{K}_k(A, r^{(0)}). \tag{6.66}
$$

Das CR ist eine schiefe Krylov-Unterraum-Methode mit $K_k = \mathcal{K}_k(A, r^{(0)}) = \mathcal{P}_k$ und $L_k = A\,\mathcal{K}_k(A, r^{(0)}) = A\,\mathcal{R}_k \subset \mathcal{K}_{k+1}(A, r^{(0)})$.
Mit den Beziehungen $x^{(k)} = x^{(0)} + q_{k-1}(A) r^{(0)}$ und $r^{(k)} = p_k(A) r^{(0)}$ ergibt sich das auf Polynomen basierende AV (polynomial based method) in der Gestalt (6.58).

Das CR beruht auf der Minimierung des Funktionals

$$\begin{aligned} R(x) &= \frac{1}{2}x^T A^T A x - x^T A^T b \\ &= \frac{1}{2}\Big(\|r(x)\|_2^2 - x^{*T} A^T b\Big) \;=\; \frac{1}{2}\Big(\|e(x)\|_{A^T A}^2 - b^T b\Big) \end{aligned} \tag{6.67}$$

und damit der von der euklidischen Norm $\|r(x)\|_2$, d. h. "minimale Residuumeigenschaft" (minimal residual property) und bedeutet

$$\begin{aligned} \|r^{(k)}\|_2 \;=\; \|r(x^{(k)})\|_2 &= \|b - Ax^{(k)}\|_2 \\ &= \|p_k(A)r^{(0)}\|_2 \\ &= \min_{\substack{p(x)\in\mathcal{P}_k(x)\\ p(0)=1}} \|p(A)r^{(0)}\|_2 \end{aligned} \tag{6.68}$$

mit

$$\begin{aligned} \|r^{(k)}\|_2^2 &= r^{(k)T}\, r^{(k)} \\ &= (p_k(A)r^{(0)})^T\, p_k(A)r^{(0)} \\ &= r^{(0)T} p_k(A)\, p_k(A) r^{(0)} \\ &= (p_k(A), p_k(A))_{MR} \\ &= (p_k, p_k)_{MR}, \end{aligned} \tag{6.69}$$

wobei in der letzten Beziehung das modifizierte Skalarprodukt (bezüglich A und $r^{(0)}$)

$$(p,q)_{MR} \;=\; r^{(0)T} p(A) q(A) r^{(0)} \tag{6.70}$$

definiert wurde.
Man erhält damit eine äquivalente Formulierung zur Problemstellung des CR.

> Finde ein solches Polynom $p_k(x) \in \mathcal{P}_k^1(x)$, so dass
>
> $$(p_k, p_k)_{MR} \;\le\; (p,p)_{MR}$$
>
> für alle $p(x) \in \mathcal{P}_k^1(x)$ gilt.

(6.71)

Bleibt die Frage nach der expliziten Lösung dieses Problems.
Von den Tschebyscheff-Polynomen wissen wir schon, dass solche "Minimalpolynome", wie sie in (6.63) und (6.71) zu finden sind, gewisse Orthogonalitätseigenschaften haben und mit diesen zusammen ihre verkürzte rekursive Berechnung (Drei-Term-Rekursion) erlauben.

6.2.4 Orthogonale Residuumpolynome

Das auf Polynomen basierende AV verwendet in der Formel für das Residuum $r^{(k)} = p_k(A)r^{(0)}$ das Residuumpolynom $p_k(x) = 1 - xq_{k-1}(x) \in \mathcal{P}_k(x)$.
Hier sollen seine Orthogonalitätseigenschaften in Verbindung gebracht werden mit der rekursiven Berechnung als Drei-Term-Rekursion, wie dies bei den Untersuchungen in [33], Kap. 1.1, schon gemacht worden ist.
Die Polynome k-ten Grades $p_k = p_k(x) \in \mathcal{P}_k(x)$ sollen ein Orthogonalsystem (OGS) bilden im Sinne von

$$(p_k, p_j) \begin{cases} = 0 & \text{für } j \neq k, \\ > 0 & \text{für } j = k. \end{cases} \tag{6.72}$$

Die Fourier-Reihenentwicklung des Polynoms $xp_{j-1}(x)$ nach dem OGS $\{p_k\}$ liefert

$$\begin{aligned} xp_{j-1}(x) &= \sum_{i=0}^{j} \frac{(xp_{j-1}, p_i)}{(p_i, p_i)} p_i(x) \\ &= \sum_{i=0}^{j} \frac{(p_{j-1}, xp_i)}{(p_i, p_i)} p_i(x) \\ &= \sum_{i=j-2}^{j} \frac{(xp_{j-1}, p_i)}{(p_i, p_i)} p_i(x). \end{aligned} \tag{6.73}$$

Das Polynomsystem $\{p_0, p_1, ..., p_{j-1}, xp_{j-1}\}$ ist eine Basis in $\mathcal{P}_j(x)$, so dass wir für $p_j(x)$ mit folgendem Ansatz arbeiten können.

$$\begin{aligned} p_j(x) &= \delta_j xp_{j-1}(x) + \delta_{j-1}p_{j-1}(x) + \delta_{j-2}p_{j-2}(x) + ... + \delta_1 p_1(x) + \delta_0, \\ \tfrac{1}{\delta_j}p_j(x) &= (x + \tfrac{\delta_{j-1}}{\delta_j})p_{j-1}(x) + \tfrac{\delta_{j-2}}{\delta_j}p_{j-2}(x) + ... + \tfrac{\delta_1}{\delta_j}p_1(x) + \tfrac{\delta_0}{\delta_j}, \\ \gamma_j p_j(x) &= (x - \alpha_j)p_{j-1}(x) - \beta_j p_{j-2}(x) + \tilde{\delta}_{j-3}p_{j-3}(x) + ... + \tilde{\delta}_1 p_1(x) + \tilde{\delta}_0. \end{aligned} \tag{6.74}$$

Wir zeigen nun, dass $\tilde{\delta}_{j-3} = ... = \tilde{\delta}_1 = \tilde{\delta}_0 = 0$ sind. Dazu untersuchen wir die Beziehung

$$\gamma_j p_j(x) - xp_{j-1}(x) = -\alpha_j p_{j-1}(x) - \beta_j p_{j-2}(x) + \sum_{i=0}^{j-3} \tilde{\delta}_i p_i(x) \ \in \ \mathcal{P}_{j-1}(x)$$

und ihr Skalarprodukt mit $p_k(x)$ für $k = 0, 1, ..., j-1, j$

$$(p_k(x), \gamma_j p_j(x) - xp_{j-1}(x)) = \Big(p_k(x), -\alpha_j p_{j-1}(x) - \beta_j p_{j-2}(x) + \sum_{i=0}^{j-3} \tilde{\delta}_i p_i(x)\Big).$$

Für $k = 0, 1, ..., j-3$ ist $xp_k(x) \in \mathcal{P}_{j-2}(x)$ sowie

$$(p_k(x), \gamma_j p_j(x) - xp_{j-1}(x)) = \tilde{\delta}_k(p_k(x), p_k(x))$$

und somit

$$\tilde{\delta}_k = \frac{\gamma_j(p_k, p_j) - (p_k, xp_{j-1})}{(p_k, p_k)} = -\frac{(p_k, xp_{j-1})}{(p_k, p_k)} = -\frac{(xp_k, p_{j-1})}{(p_k, p_k)} = 0.$$

Für $k = j - 2$ gelten

$$\begin{aligned} \tilde{\delta}_{j-2} = -\beta_j &= \frac{\gamma_j(p_{j-2}, p_j) - (p_{j-2}, xp_{j-1})}{(p_{j-2}, p_{j-2})} = -\frac{(xp_{j-1}, p_{j-2})}{(p_{j-2}, p_{j-2})}, \\ \beta_j &= \frac{(xp_{j-1}, p_{j-2})}{(p_{j-2}, p_{j-2})} = \frac{(p_{j-1}, xp_{j-2})}{(p_{j-2}, p_{j-2})}. \end{aligned}$$

Falls der Hauptkoeffizient von $p_k,\ k = 0, 1, ...,$ gleich Eins ist, gilt

$$\beta_j = \frac{(p_{j-1}, xp_{j-2})}{(p_{j-2}, p_{j-2})} = \frac{(p_{j-1}, p_{j-1} + c_{j-2}p_{j-2} + ... + c_0p_0)}{(p_{j-2}, p_{j-2})} = \frac{(p_{j-1}, p_{j-1})}{(p_{j-2}, p_{j-2})}.$$

Für $k = j - 1$ erhält man

$$\begin{aligned} \tilde{\delta}_{j-1} = -\alpha_j &= \frac{\gamma_j(p_{j-1}, p_j) - (p_{j-1}, xp_{j-1})}{(p_{j-1}, p_{j-1})} = -\frac{(p_{j-1}, xp_{j-1})}{(p_{j-1}, p_{j-1})}, \\ \alpha_j &= \frac{(xp_{j-1}, p_{j-1})}{(p_{j-1}, p_{j-1})} = \frac{(p_{j-1}, xp_{j-1})}{(p_{j-1}, p_{j-1})}. \end{aligned}$$

Für $k = j$ folgt sofort

$$\gamma_j = \frac{(xp_{j-1}, p_j)}{(p_j, p_j)} = \frac{(xp_j, p_{j-1})}{(p_j, p_j)}.$$

Damit kann man die Drei-Term-Rekursion der Polynome zusammenfassend notieren.

$$\begin{aligned} p_{-1}(x) = 0, &\quad p_0(x) = p_0, \\ \gamma_j p_j(x) &= (x - \alpha_j)p_{j-1}(x) - \beta_j p_{j-2}(x), \quad j = 1, 2, ..., \\ \gamma_j = \frac{(xp_{j-1}, p_j)}{(p_j, p_j)},\ \alpha_j &= \frac{(xp_{j-1}, p_{j-1})}{(p_{j-1}, p_{j-1})},\ \beta_j = \frac{(xp_{j-1}, p_{j-2})}{(p_{j-2}, p_{j-2})},\ \beta_1 = 0. \end{aligned} \tag{6.75}$$

Die Zusatzbedingung (Skalierung, interpolatory constraint)
$p_j(0) = p_{j-1}(0) = p_{j-2}(0) = 1$, damit auch $p_0 = 1$, führt auf die Beziehung

$$\gamma_j = -(\alpha_j + \beta_j),\ j > 1, \quad \gamma_1 = -\alpha_1, \tag{6.76}$$

zwischen den Rekursionsparametern und die beiden Darstellungen

$$\begin{aligned} -(\alpha_j + \beta_j)p_j(x) &= (x - \alpha_j)p_{j-1}(x) - \beta_j p_{j-2}(x), \\ \gamma_j p_j(x) &= (x + \gamma_j + \beta_j)p_{j-1}(x) - \beta_j p_{j-2}(x). \end{aligned} \tag{6.77}$$

Der Algorithmus zur Drei-Term-Rekursion der skalierten Polynome lautet somit

$$
\begin{aligned}
p_{-1}(x) &= 0, \quad p_0(x) = 1 \\
&\underline{\textbf{j = 1,2,...}} \\
\alpha_j &= \frac{(xp_{j-1}, p_{j-1})}{(p_{j-1}, p_{j-1})}, \quad \beta_j = \begin{cases} 0 & \text{für } j = 1 \\ \dfrac{(xp_{j-1}, p_{j-2})}{(p_{j-2}, p_{j-2})} & \text{für } j \geq 2 \end{cases} \\
-(\alpha_j + \beta_j)p_j(x) &= (x - \alpha_j)p_{j-1}(x) - \beta_j p_{j-2}(x) \\
&\underline{\textbf{end j}}
\end{aligned}
\tag{6.78}
$$

Natürlich findet man viel Ähnlichkeit zum Satz 1.1 in [33] und zu den Ergebnissen im Zusammenhang mit der dortigen Rekursionsformel (1.5).

6.2.5 Polynomiale Iterationsverfahren

Wir betrachten die allgemeine Form polynomialer IV gemäß

$$
\begin{aligned}
x^{(k)} &= x^{(0)} + q_{k-1}(A)r^{(0)}, \quad q_{k-1}(A) \in \mathcal{P}_{k-1}(A), \\
r^{(k)} &= p_k(A)r^{(0)}, \quad p_k(A) \in \mathcal{P}_k(A), \\
&\quad p_k(x) = 1 - xq_{k-1}(x) \in \mathcal{P}_k^1(x).
\end{aligned}
\tag{6.79}
$$

Ziel ist es, entweder die A-Norm $\|e(x^{(k)})\|_A$ bzw. $\|e(x^{(k)})\|_A^2$ wie bei GV, CG oder besser noch die euklidische Norm $\|r(x^{(k)})\|_2$ bzw. $\|r(x^{(k)})\|_2^2 = (p_k, p_k)_{CGr}$ (6.61) beim CG bzw. $\|r(x^{(k)})\|_2^2 = (p_k, p_k)_{MR}$ (6.69) beim CR kontinuierlich zu verkleinern.
In jedem Fall ist es eine "minimale Eigenschaft" und bedeutet bei $s(x^{(k)}) = e(x^{(k)})$ bzw. $s(x^{(k)}) = r(x^{(k)})$

$$
\begin{aligned}
\|s(x^{(k)})\|_2 &= \|p_k(A)s^{(0)}\|_2 \\
&= \min_{\substack{p(x) \in \mathcal{P}_k(x) \\ p(0) = 1}} \|p(A)s^{(0)}\|_2.
\end{aligned}
\tag{6.80}
$$

Wenn $\|r^{(k)}\|_2 = \|r(x^{(k)})\|_2$ monoton fallend ist, dann gilt das wegen der Minimaleigenschaft von $p_k(A)$ bzw. $p_k(x)$ und $e^{(k)} = p_k(A)e^{(0)}$ auch für den Fehler $\|e^{(k)}\|_2$.
Und wie verhält es sich dann mit dem Fehler $\|e^{(k)}\|_A$?
Wegen

$$
\|e^{(k)}\|_A^2 = (Ae^{(k)}, e^{(k)}) = e^{(0)T}p_k(A)Ap_k(A)e^{(0)}
$$

ist die A-Norm meistens auch abnehmend.

Wo wenden wir nun die Drei-Term-Rekursion mit ihren Eigenschaften an?
Der entscheidende Aspekt ist, das Residuum $r^{(k)} = p_k(A)r^{(0)}$ und damit das Residuumpolynom $p_k(x)$ als Wachstumsfaktor geeignet in die Rekursionsformel einzubetten und daraus wiederum die rekursiven Beziehungen als GV (6.44), CG (6.53), CR (6.64) oder anderer AV zu erhalten.
Wir werden sehen, dass dies möglich ist. Betrachten wir also die Drei-Term-Rekursion (6.77) und wenden diese einfach auf das Residuum $r^{(k)}$ (6.79) an.

$$
\begin{aligned}
\gamma_k p_k(x) &= (x + \gamma_k + \beta_k)p_{k-1}(x) - \beta_k p_{k-2}(x), \\
\gamma_k p_k(A)r^{(0)} &= (A + (\gamma_k + \beta_k)I)p_{k-1}(A)r^{(0)} - \beta_k p_{k-2}(A)r^{(0)}, \\
\gamma_k r^{(k)} &= (A + (\gamma_k + \beta_k)I)r^{(k-1)} - \beta_k r^{(k-2)} \\
&= \gamma_k r^{(k-1)} + Ar^{(k-1)} + \beta_k(r^{(k-1)} - r^{(k-2)}) \\
&= \gamma_k r^{(k-1)} + Ar^{(k-1)} + \beta_k[b - Ax^{(k-1)} - (b - Ax^{(k-2)})] \\
&= \gamma_k r^{(k-1)} + Ar^{(k-1)} + \beta_k A(x^{(k-2)} - x^{(k-1)}) \\
&= \gamma_k r^{(k-1)} + A[r^{(k-1)} + \beta_k(x^{(k-2)} - x^{(k-1)})] \\
r^{(k)} - r^{(k-1)} &= \frac{1}{\gamma_k} A[\underbrace{r^{(k-1)} + \beta_k(x^{(k-2)} - x^{(k-1)})}_{-w^{(k-1)}}], \\
A^{-1}(r^{(k)} - r^{(k-1)}) &= -(x^{(k)} - x^{(k-1)}) \;=\; -\frac{1}{\gamma_k} w^{(k-1)}, \\
w^{(k-1)} &= -r^{(k-1)} - \beta_k(x^{(k-2)} - x^{(k-1)}) \\
&= \beta_k(x^{(k-1)} - x^{(k-2)}) - r^{(k-1)}, \quad x^{(k-1)} = x^* - A^{-1}r^{(k-1)} \\
&= \beta_k(x^* - A^{-1}r^{(k-1)} - x^* + A^{-1}r^{(k-2)}) - r^{(k-1)} \\
&= \beta_k A^{-1}(r^{(k-2)} - r^{(k-1)}) - r^{(k-1)} \\
w^{(k-1)} &= \frac{\beta_k}{\gamma_{k-1}} w^{(k-2)} - r^{(k-1)}, \\
-w^{(k-1)} &= r^{(k-1)} + \frac{\beta_k}{\gamma_{k-1}}(-w^{(k-2)}).
\end{aligned}
\tag{6.81}
$$

Somit folgen die Aufdatierungsformeln (update formulae)

$$
\begin{aligned}
x^{(k)} &= x^{(k-1)} + \frac{1}{\gamma_k} w^{(k-1)} \;=\; x^{(k-1)} + \Big(-\frac{1}{\gamma_k}\Big)(-w^{(k-1)}), \\
r^{(k)} &= r^{(k-1)} - \frac{1}{\gamma_k} A w^{(k-1)}.
\end{aligned}
\tag{6.82}
$$

Der Vektor $-w^{(k-1)}$ entspricht der Suchrichtung, die Größe $-\frac{1}{\gamma_k}$ ist die Schrittzahl.

Gegeben sei also die Drei-Term-Rekursion (6.77), (6.78) für ein System von orthogonalen Residuumpolynomen $p_k(x)$

$$\gamma_k p_k(x) \;=\; (x+\gamma_k+\beta_k)p_{k-1}(x) - \beta_k p_{k-2}(x).$$

Dann ist der Algorithmus des auf diesen Polynomen basierenden AV mit $r^{(k)} = p_k(A)r^{(0)}$ wie folgt.

$$\begin{aligned}
&x^{(0)}, \quad r^{(0)} \;=\; b - Ax^{(0)},\; w^{(-1)} = 0,\; \beta_1 = 0 \\
&\underline{\textbf{k = 1,2,...}} \\
&w^{(k-1)} \;=\; \begin{cases} -r^{(k-1)} & \text{für } k = 1 \\ \dfrac{\beta_k}{\gamma_{k-1}} w^{(k-2)} - r^{(k-1)} & \text{für } k \geq 2 \end{cases} \\
&x^{(k)} \;=\; x^{(k-1)} + \frac{1}{\gamma_k} w^{(k-1)} \\
&r^{(k)} \;=\; r^{(k-1)} - \frac{1}{\gamma_k} A w^{(k-1)} \\
&\underline{\textbf{end k}}
\end{aligned} \tag{6.83}$$

Dabei ist

$$\frac{\beta_k}{\gamma_{k-1}} \;=\; \frac{(xp_{k-1}, p_{k-2})}{(p_{k-2}, p_{k-2})} \frac{(p_{k-1}, p_{k-1})}{(p_{k-1}, xp_{k-2})} \;=\; \frac{(p_{k-1}, p_{k-1})}{(p_{k-2}, p_{k-2})}$$

und

$$\frac{1}{\gamma_k} \;=\; -(\alpha_k + \beta_k)^{-1} \;=\; -\Big(\frac{(xp_{k-1}, p_{k-1})}{(p_{k-1}, p_{k-1})} + \frac{(xp_{k-1}, p_{k-2})}{(p_{k-2}, p_{k-2})}\Big)^{-1}.$$

Man erkennt, dass sich im Algorithmus (6.83) die Formeln von CG und CR widerspiegeln: $x^{(0)}$, $r^{(0)}$ und Schrittzahlen anders bezeichnet.

CG	CR
$k = 0, 1, ...$	
$\tilde\beta_{k-1} = \dfrac{r^{(k)T} r^{(k)}}{r^{(k-1)T} r^{(k-1)}}$ $(k \geq 1),\ \tilde\beta_{-1} = 0$	$\tilde\beta_{k-1} = \dfrac{r^{(k)T} A r^{(k)}}{r^{(k-1)T} A r^{(k-1)}}$ $(k \geq 1),\ \tilde\beta_{-1} = 0$
$p^{(k)} = \begin{cases} r^{(k)} & \text{für } k = 0 \\ r^{(k)} + \tilde\beta_{k-1} p^{(k-1)} & \text{für } k \geq 1 \end{cases}$	
$\tilde\alpha_k = \dfrac{r^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}}$	$\tilde\alpha_k = \dfrac{r^{(k)T} A r^{(k)}}{(Ap^{(k)})^T A p^{(k)}}$
$x^{(k+1)} = x^{(k)} + \tilde\alpha_k p^{(k)}$	
$r^{(k+1)} = r^{(k)} - \tilde\alpha_k A p^{(k)}$	

6.2.6 Explizite Lösung des CG als polynomiales Iterationsverfahren mittels orthogonaler Residuumpolynome

Die orthogonalen Residuumpolynome des CG findet man in zwei Schritten.
Dabei wollen wir uns auf die euklidische Norm des Residuums $r^{(k)}$ konzentrieren.
Zunächst verwendet man die Skalarprodukte (bezüglich A und $r^{(0)}$) (6.61), (6.70), um ein orthonormales Polynomsystem $\{\psi_k\}$ zu erzeugen, d. h.

$$(\psi_j, \psi_k)_{MR} \;=\; (\psi_j, \psi_k)_{CGr} \;=\; \delta_{jk}. \tag{6.84}$$

Daraus berechnet man die orthogonalen Residuumpolynome, auch Kernpolynome (kernel polynomial) genannt,

$$p_k^{GAL}(x) \;=\; \frac{\psi_k(x)}{\psi_k(0)}, \quad p_k^{GAL}(0) = 1, \quad (p_k^{GAL}, p_k^{GAL})_{MR} = \frac{1}{\psi_k(0)^2}. \tag{6.85}$$

Man erhält damit die nächste äquivalente Formulierung zur Problemstellung des CG (extremal property).

$p_k^{GAL}(x)$ ist ein solches Polynom, so dass

$$(p_k^{GAL}, p_k^{GAL})_{MR} \;\leq\; (p, p)_{MR}$$

für alle $p(x) \in \mathcal{P}_k^1(x)$ gilt. (6.86)

Die Orthogonalität der Residuumpolynome $p_k^{GAL}(x)$ ergibt bei $j \neq k$ mit (6.70) und dem modifizierten Skalarprodukt (6.62) die Beziehung

$$\begin{aligned}
0 \;=\; (p_j^{GAL}, p_k^{GAL})_{MR} \;&=\; (r^{GAL\,(j)})^T\, r^{GAL\,(k)} \\
&=\; (p_j^{GAL}(A) r^{(0)})^T\, p_k^{GAL}(A) r^{(0)} \\
&=\; r^{(0)T} p_j^{GAL}(A)\, p_k^{GAL}(A) r^{(0)} \\
&=\; (A e^{(0)})^T p_j^{GAL}(A)\, p_k^{GAL}(A) A e^{(0)} \\
&=\; e^{(0)T} p_j^{GAL}(A)\, A\, (A p_k^{GAL}(A))\, e^{(0)} \\
&=\; (p_j^{GAL}, A p_k^{GAL})_{GAL},
\end{aligned}$$

so dass die Residuumpolynome $p_k^{GAL}(x)$ auch orthogonal bezüglich des Skalarprodukts $(\cdot, x \cdot)_{GAL}$ sind. Die Residua $r^{GAL\,(k)} = p_k^{GAL}(A) r^{(0)}$ erfüllen die so genannte Galerkin-Bedingung (daher auch die Bezeichnung)

$$(p_j^{GAL}, p_k^{GAL})_{MR} \;=\; (r^{GAL\,(j)})^T\, r^{GAL\,(k)} = 0, \quad j \neq k. \tag{6.87}$$

Somit findet man die Koeffizienten der Drei-Term-Rekursion sowie eine neue Darstellung des CG aus (6.83) (vergl. [6]).

$$
\begin{aligned}
& x^{(0)},\ r^{(0)} = b - Ax^{(0)},\ x^{GAL\,(0)} = x^{(0)},\ r^{GAL\,(0)} = r^{(0)},\ w^{(-1)} = 0 \\
& \underline{\mathbf{k = 1,2,...}} \\
w^{(k-1)} &= \frac{\beta_k}{\gamma_{k-1}} w^{(k-2)} - r^{(k-1)},\ \text{wobei} \\
\frac{\beta_k}{\gamma_{k-1}} &= \begin{cases} 0 & \text{für } k = 1 \\ \dfrac{(p_{k-1}^{GAL}, p_{k-1}^{GAL})_{MR}}{(p_{k-2}^{GAL}, p_{k-2}^{GAL})_{MR}} = \dfrac{(r^{GAL\,(k-1)})^T\, r^{GAL\,(k-1)}}{(r^{GAL\,(k-2)})^T\, r^{GAL\,(k-2)}} & \text{für } k \geq 2 \end{cases} \\
x^{GAL\,(k)} &= x^{GAL\,(k-1)} + \frac{1}{\gamma_k} w^{(k-1)},\ \text{wobei} \\
\frac{1}{\gamma_k} &= -\Big(\frac{(xp_{k-1}^{GAL}, p_{k-1}^{GAL})_{MR}}{(p_{k-1}^{GAL}, p_{k-1}^{GAL})_{MR}} + \underbrace{\frac{(xp_{k-1}^{GAL}, p_{k-2}^{GAL})_{MR}}{(p_{k-2}^{GAL}, p_{k-2}^{GAL})_{MR}}}_{=0} \Big)^{-1}. \\
&= -\frac{(p_{k-1}^{GAL}, p_{k-1}^{GAL})_{MR}}{(xp_{k-1}^{GAL}, p_{k-1}^{GAL})_{MR}} = -\frac{(r^{GAL\,(k-1)})^T\, r^{GAL\,(k-1)}}{w^{(k-1)T} A w^{(k-1)}} \\
r^{GAL\,(k)} &= r^{GAL\,(k-1)} - \frac{1}{\gamma_k} A w^{(k-1)} \\
& \underline{\mathbf{end\ k}}
\end{aligned}
\tag{6.88}
$$

Definiert man $p^{(k-1)} = -w^{(k-1)}$, $\tilde{\beta}_k = \frac{\beta_k}{\gamma_{k-1}}$ und $\tilde{\alpha}_k = \frac{1}{\gamma_k}$, so hat man die ursprüngliche Form des CG.

6.2.7 Explizite Lösung des CR als polynomiales Iterationsverfahren mittels orthogonaler Residuumpolynome

Die orthogonalen Residuumpolynome des CR findet man in zwei Schritten.
Zunächst verwendet man das modifizierte Skalarprodukt (bezüglich A und $r^{(0)}$) (6.70), um ein orthonormales Polynomsystem $\{\psi_k\}$ zu erzeugen, d. h.

$$(\psi_j, \psi_k)_{MR} = \delta_{jk}. \tag{6.89}$$

Daraus berechnet man die skalierten oder Kernpolynome (kernel polynomial)

$$p_k^{MR}(x) = \frac{\sum_{i=0}^{k} \psi_i(0)\psi_i(x)}{\sum_{i=0}^{k} \psi_i(0)^2},\quad p_k^{MR}(0) = 1,\quad (p_k^{MR}, p_k^{MR})_{MR} = \frac{1}{\sum_{i=0}^{k} \psi_i(0)^2}. \tag{6.90}$$

Man erhält damit die nächste äquivalente Formulierung zur Problemstellung des CR (extremal property).

$$\boxed{\begin{array}{l} p_k^{MR}(x) \text{ ist ein solches Polynom, so dass} \\ \qquad (p_k^{MR}, p_k^{MR})_{MR} \;\leq\; (p,p)_{MR} \\ \text{für alle } \; p(x) \in \mathcal{P}_k^1(x) \text{ gilt.} \end{array}} \tag{6.91}$$

Unter Verwendung des Ansatzes

$$q_k(x) = q_k(0)p_k^{MR}(x) + xq_{k-1}(x) = q_k(0)p_k^{MR}(x) + x\sum_{i=0}^{k-1}\delta_i p_i^{MR}(x)$$

zeigt man die Eigenschaft (reproducing property)

$$(p_k^{MR}, q_k)_{MR} \;=\; q_k(0) \;\; \forall\; q_k(x) \in \mathcal{P}_k(x). \tag{6.92}$$

Insbesondere folgt daraus für den speziellen Fall $q_k(x) = xq_{k-1}(x)$ die Beziehung

$$(p_k^{MR}, xq_{k-1})_{MR} \;=\; 0 \;\; \forall\; q_{k-1}(x) \in \mathcal{P}_{k-1}(x). \tag{6.93}$$

Die Kernpolynome $p_k^{MR}(x)$ sind damit orthogonal bezüglich des modifizierten Skalarprodukts

$$(p, xq)_{MR} \;=\; r^{(0)T}p(A)\,Aq(A)r^{(0)}, \tag{6.94}$$

das somit die Grundlage für die Drei-Term-Rekursion der Polynome bildet.
Die Residua $r^{MR\,(k)} = p_k^{MR}(A)r^{(0)}$ erfüllen damit die A-Orthogonalität

$$(p_j^{MR}, xp_k^{MR})_{MR} \;=\; (r^{MR\,(j)})^T\,Ar^{MR\,(k)} = 0, \quad j \neq k. \tag{6.95}$$

Aus (6.78) entnehmen wir den Koeffizienten α_j der Drei-Term-Rekursion gemäß

$$\begin{aligned} \alpha_j &= \frac{(xp_{j-1}, p_{j-1})}{(p_{j-1}, p_{j-1})} \\ &= \frac{(xp_{j-1}^{MR}, xp_{j-1}^{MR})_{MR}}{(p_{j-1}^{MR}, xp_{j-1}^{MR})_{MR}} \\ &= \frac{(Ar^{MR\,(j-1)})^T Ar^{MR\,(j-1)}}{(r^{MR\,(j-1)})^T Ar^{MR\,(j-1)}} \end{aligned}$$

und finden eine neue Darstellung des CR aus (6.83) (vergl. [6]).

CR als polynomiales AV

$$
\begin{aligned}
& x^{(0)},\ r^{(0)} = b - Ax^{(0)},\ x^{MR\,(0)} = x^{(0)},\ r^{MR\,(0)} = r^{(0)},\ w^{(-1)} = 0 \\
& \underline{\textbf{k = 1,2,...}} \\
& w^{(k-1)} = \frac{\beta_k}{\gamma_{k-1}} w^{(k-2)} - r^{(k-1)},\ \text{wobei} \\
& \frac{\beta_k}{\gamma_{k-1}} = \begin{cases} 0 & \text{für } k = 1 \\ \dfrac{(p_{k-1}^{MR}, xp_{k-1}^{MR})_{MR}}{(p_{k-2}^{MR}, xp_{k-2}^{MR})_{MR}} = \dfrac{(r^{MR\,(k-1)})^T\, Ar^{MR\,(k-1)}}{(r^{MR\,(k-2)})^T\, Ar^{MR\,(k-2)}} & \text{für } k \geq 2 \end{cases} \\
& x^{MR\,(k)} = x^{MR\,(k-1)} + \frac{1}{\gamma_k} w^{(k-1)},\ \text{wobei} \\
& \frac{1}{\gamma_k} = -\Big(\frac{(xp_{k-1}^{MR}, xp_{k-1}^{MR})_{MR}}{(p_{k-1}^{MR}, xp_{k-1}^{MR})_{MR}} + \underbrace{\frac{(xp_{k-1}^{MR}, xp_{k-2}^{MR})_{MR}}{(p_{k-2}^{MR}, xp_{k-2}^{MR})_{MR}}}_{=0} \Big)^{-1}. \\
& \qquad = -\frac{(p_{k-1}^{MR}, xp_{k-1}^{MR})_{MR}}{(xp_{k-1}^{MR}, xp_{k-1}^{MR})_{MR}} = -\frac{(r^{MR\,(k-1)})^T\, Ar^{MR\,(k-1)}}{(Aw^{(k-1)})^T Aw^{(k-1)}} \\
& r^{MR\,(k)} = r^{MR\,(k-1)} - \frac{1}{\gamma_k} Aw^{(k-1)} \\
& \underline{\textbf{end k}}
\end{aligned}
\tag{6.96}
$$

6.3 Vergleich des Fehlerverhaltens von CG und CR

Die Funktionale, das Abstiegsverhalten sowie die Entwicklung der verschiedenen Fehler im CG und CR haben wir ausführlich erläutert. Für ausgewählte Modellprobleme sollen diese Eigenschaften zusammen mit numerischen Ergebnissen noch einmal gegenübergestellt werden.
Zunächst aber einige Vorbetrachtungen.

6.3.1 Möglichkeiten der Generierung von Matrizen und Vektoren

Wir beziehen uns hier bez. der Modellprobleme auf das Preprint No. M 10/03 *Spezielle Aspekte zu CAS Maple und Matlab*, IfMath TU Ilmenau 2003.
Als Matrizen des zu lösenden LGS treten so genannte Laplace-Matrizen auf.
Solche erhält man z. B. bei elliptischen Randwertaufgaben.
Zunächst soll kurz ihre Entstehung im ein- und zweidimensionalen Fall erläutert werden.

(1) Tridiagonalmatrix, 1D-Laplace-Matrix
Als Modellproblem für die Erzeugung einer solchen Matrix wird die Stabdurchbiegung als einfache eindimensionale Zweipunktrandwertaufgabe (RWA) mit inhomogenen Randbedingungen gewählt.
Das zur numerischen Behandlung verwendete Diskretisierungsverfahren führt mit dem 3-Punkte-Differenzenstern auf ein LGS, dessen Eigenschaften kurz dargestellt werden.

- Zweipunktrandwertaufgabe mit inhomogenen Randbedingungen:

$$\begin{aligned} -U''(x) &= F(x), \quad x \in \Omega = (0,1) \subset \mathbb{R}, \\ U &= \varphi \ \text{ für } \ x \in \partial\Omega \quad \text{bzw. } \ U(0) = \varphi_0, \ \ U(1) = \varphi_1. \end{aligned}$$

- Gitter: $\overline{\Omega}_h = \{x \mid x = ih,\ i = 0(1)N{+}1,\ h = 1/(N{+}1)\}$, h Maschenweite.
- Gitterfunktion: $u_h = (u_1, u_2, ..., u_N)^T$ mit $u_i \approx U_i = U(ih)$.
- Analog für rechte Seite: $f_h = (f_1, f_2, ..., f_N)^T$ mit $f_i = F_i = F(ih)$,
  d. h. auf dem Gitter wird die rechte Seite exakt dargestellt.
- Approximation der Ableitungen (Operatoren) mittels Differenzenausdrücken:

$$U''(x_i) \approx \frac{1}{h^2}(U_{i+1} - 2U_i + U_{i-1}) \ \text{ zentraler Differenzenquotient 2. Ordnung.}$$

- Diskretisierte Aufgabe als LGS:

$$\begin{aligned} -\frac{1}{h^2}(u_{i+1} - 2u_i + u_{i-1}) &= f_i, \quad i = 1, 2, ..., N, \\ u_0 = \varphi_0, \quad u_{N+1} &= \varphi_1. \end{aligned}$$

- Matrixschreibweise des LGS:

$$A_h u_h = b_h \quad \text{bzw.} \quad Au = b$$

mit

$$A_h = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b_h = \begin{pmatrix} f_1 + \varphi_0/h^2 \\ f_2 \\ f_3 \\ \cdots \\ f_{N-1} \\ f_N + \varphi_1/h^2 \end{pmatrix},$$

$$A = \begin{pmatrix} 2 & -1 & & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b = h^2 \begin{pmatrix} f_1 + \varphi_0/h^2 \\ f_2 \\ f_3 \\ \cdots \\ f_{N-1} \\ f_N + \varphi_1/h^2 \end{pmatrix},$$

$$A = \text{tridiag}(-1, 2, -1).$$

Die Koeffizientenmatrix $A(n, n)$ ist schwach besetzt. Sie hat etwa $3n$ nichtverschwindende Elemente ($n = N$) an Stelle von n^2 Elementen bei voll besetzten Matrizen. Sie ist eine Tridiagonalmatrix, d. h. ihre Bandbreite ist 3. Dazu ist sie symmetrisch und positiv definit (spd).

```
> n := 8:
  B1 := band([-1,2,-1],n);   # spd 1D-Laplace-Matrix
```

$$B1 := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

(2) Weitere RWA

$T''(x) = -\sin(\pi x),\ 0 < x < 1,\ T(0) = T(1) = 0,\ T_{exakt}(x) = \frac{1}{\pi^2}\sin(\pi x)$

Definition der Komponenten im LGS $Ax = b$

```
> n:=8;
  h:=1/(n+1);
  i:='i':
  xii:=vector(n,[seq(i*h,i=1..n)]);
  Texakt:= evalf(evalm(sin(Pi*xii)/(Pi*Pi)));

  B11 := band([-1,2,-1],n):  # Tridiagonalmatrix
  i := 'i': j := 'j':
  b := h*h*sin(Pi*xii);
  evalm(%);
  evalf(%);
  T := vector(n):
  B11E := matrix(n,n+1):
  B11E := augment(B11,b):   # erweiterte Matrix
  # B11E := concat(B11,b);  # Synonym fuer augment
  B11E:
  % = evalm(%);
```

$$n := 8$$

$$h := \frac{1}{9}$$

$$xii := \left[\frac{1}{9}, \frac{2}{9}, \frac{1}{3}, \frac{4}{9}, \frac{5}{9}, \frac{2}{3}, \frac{7}{9}, \frac{8}{9}\right]$$

$$Texakt := [0.03465388573,\ 0.06512800142,\ 0.08774671895,\ 0.09978188716,\ 0.09978188716,\ 0.08774671895,\ 0.06512800142,\ 0.03465388573]$$

$$b := \frac{1}{81}\sin(\pi\, xii)$$

$$\left[\frac{1}{81}\sin\left(\frac{\pi}{9}\right), \frac{1}{81}\sin\left(\frac{2\pi}{9}\right), \frac{\sqrt{3}}{162}, \frac{1}{81}\sin\left(\frac{4\pi}{9}\right), \frac{1}{81}\sin\left(\frac{4\pi}{9}\right), \frac{\sqrt{3}}{162}, \frac{1}{81}\sin\left(\frac{2\pi}{9}\right), \frac{1}{81}\sin\left(\frac{\pi}{9}\right)\right]$$

$$[\ 0.004222470904,\ 0.007935649501,\ 0.01069167165,\ 0.01215812041,\ 0.01215812041,\ 0.01069167165,\ 0.007935649501,\ 0.004222470904]$$

$$B11E := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{81}\sin(\frac{\pi}{9}) \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{81}\sin(\frac{2\pi}{9}) \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & \frac{\sqrt{3}}{162} \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & \frac{1}{81}\sin(\frac{4\pi}{9}) \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & \frac{1}{81}\sin(\frac{4\pi}{9}) \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & \frac{\sqrt{3}}{162} \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & \frac{1}{81}\sin(\frac{2\pi}{9}) \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & \frac{1}{81}\sin(\frac{\pi}{9}) \end{bmatrix}$$

(3) Blocktridiagonalmatrizen, 2D-Laplace-Matrix

Betrachten wir den Temperaturverlauf in einer dünnen quadratischen Platte gegeben durch die partielle Differentialgleichung für eine Funktion $U(x, y)$ auf dem Einheitsquadrat.

$$-\Delta U(x,y) = -\left(\frac{\partial U}{\partial x^2} + \frac{\partial U}{\partial y^2}\right) = Q(x,y), \quad (x,y) \in \Omega = (0,1)^2.$$

Auf dem Rand des Gebietes sei $U(x, y)$ gleich Null.
Das ist eine elliptische Randwertaufgabe bzw. die Poisson-Gleichung.

Der Diskretisierungsparameter bzw. die Maschenweite des quadratischen Gitters sei $h = 1/(N + 1)$. Man diskretisiert die partiellen Ableitungen mittels zentraler Differenzenquotienten 2. Ordnung

$$\Delta U(x_i, y_j) \approx \frac{1}{h^2}(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{ij})$$

und notiert die Differenzenformel (5-Punkte-Differenzenstern) für alle inneren (zweidimensionalen) Knoten (x_i, y_j), $i, j = 1, 2, ..., N$, in linearer Reihenfolge zeilenweise gemäß $(j - 1)N + i$.
Die diskretisierte RWA schreibt man als LGS $Au = h^2 q$.
Welche Struktur und Eigenschaften hat die Matrix A? Wie groß ist ihre Bandbreite?
A besitzt die folgende Blockstruktur.

$$A = \begin{pmatrix} B & -I & & & \\ -I & B & -I & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & B \end{pmatrix}$$

mit der $(N \times N)$-Matrix

$$B = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{pmatrix}$$

und der $(N \times N)$-Einheitsmatrix I. A ist eine dünn besetzte symmetrische Matrix mit Bandstruktur. Die Bandbreite beträgt $2N + 1$. Die Matrix ist irreduzibel diagonaldominant.

```
> n1 := 4:   # n1=N
  n1q := n1^2:
  T := band([-1,4,-1],n1);
  mI := diag(-1$n1);
  B2 := band([mI,T,mI],n1):
  evalm(B2);
  B2 := band([evalm(mI),evalm(T),evalm(mI)],n1);   # Error
```

$$T := \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix}$$

$$mI := \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} T & mI & 0 & 0 \\ mI & T & mI & 0 \\ 0 & mI & T & mI \\ 0 & 0 & mI & T \end{bmatrix}$$

```
Error, (in evalm) unnamed vector or array with undefined entries.
```

Korrekt: Blocktridiagonalmatrizen vom 2D-Laplace-Operator auf quadratischem Gitter

Variante 1

```
> n1 := 4:
  n1q := n1^2:
  T := band([-1,4,-1],n1):    # Diagonalblock
  B21 := diag(T$n1):          # Blockdiagonalmatrix
  for i from 1 to n1q-n1 do   # Blocknebendiagonalen ergaenzen
   B21[i,i+n1] := -1;
   B21[i+n1,i] := -1;
  end do:
  B21 := evalm(B21);           # Blocktridiagonalmatrix
```

$$B21 := \begin{bmatrix} 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{bmatrix}$$

Variante 2

```
> n1 := 4:
  n1q := n1^2:
  B22 := band([-1,0$(n1-2),-1,4,-1,0$(n1-2),-1],n1q);
  # 5 Diagonalen belegt
  # evtl. noch einige -1 zu Null machen, z.B. B22[4,5], B22[5,4],...
  B23:=evalm(B22):
  for i to n1-1 do
    ii := i*n1;
    B23[ii,ii+1] := 0;
    B23[ii+1,ii] := 0;
  end do:
  B23 := evalm(B23):  # wie B21
```

$$B22 := \begin{bmatrix} 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{bmatrix}$$

(4) Poisson-Gleichung auf Rechteckgebiet

Gegeben sei die partielle Differentialgleichung für eine Funktion $U(x, y)$ auf einem Rechteckgebiet.

$$-\Delta U(x,y) = -\left(\frac{\partial U}{\partial x^2} + \frac{\partial U}{\partial y^2}\right) = Q(x,y), \quad (x,y) \in \Omega = (a,b) \times (c,d).$$

Auf dem Rand des Gebietes sei die Funktion $U(x, y)$ vorgegeben (Dirichletsche Randbedingungen).
Zur Lösung verwenden wir die finite Differenzenmethode auf einem (2×5)-Gitter (x_i, y_j) mit der Maschenweite h. Wir führen die Nummerierung der Gitterpunkte im rechteckigen Gebiet zeilenweise durch.

So erhalten wir eine Matrix mit der Blockstruktur

$$A = \begin{pmatrix} B & -I \\ -I & B \end{pmatrix}$$

Blocktridiagonalmatrizen vom 2D-Laplace-Operator auf Rechteckgitter 2×5, 5-Punkte-Differenzenstern

```
> m := 2:
> n := 5:
  mn := m*n:
  B24 := band([-1,0$(n-2),-1,4,-1,0$(n-2),-1],mn);

  # evtl. noch einige -1 zu Null machen, z.B. B24[5,6], B24[6,5],...
  B25 := evalm(B24):
  for i to m-1 do
    ii := i*n;
    B25[ii,ii+1] := 0;
    B25[ii+1,ii] := 0;
  end do:
  B25 := evalm(B25);
```

$$B25 := \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 \end{bmatrix}$$

(5) Van der Vorst Matrix mit Shift

```
> c := 0:    # Shift
  n := 100:
  i := 'i':
  C1 := evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
  seq(C1[i,i],i=1..n);
```

-9, -7, -5, -3, -1, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181, 183, 185, 187, 189

```
> c := 0.97:   # Shift
  n := 100:
  i := 'i':
  C2 := evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
  seq(C2[i,i],i=1..n);
```

-9.97, -7.97, -5.97, -3.97, -1.97, 0.03, 2.03, 4.03, 6.03, 8.03, 10.03, 12.03, 14.03, 16.03, 18.03, 20.03, 22.03, 24.03, 26.03, 28.03, 30.03, 32.03, 34.03, 36.03, 38.03, 40.03, 42.03, 44.03, 46.03, 48.03, 50.03, 52.03, 54.03, 56.03, 58.03, 60.03, 62.03, 64.03, 66.03, 68.03, 70.03, 72.03, 74.03, 76.03, 78.03, 80.03, 82.03, 84.03, 86.03, 88.03, 90.03, 92.03, 94.03, 96.03, 98.03, 100.03, 102.03, 104.03, 106.03, 108.03, 110.03, 112.03, 114.03, 116.03, 118.03, 120.03, 122.03, 124.03, 126.03, 128.03, 130.03, 132.03, 134.03, 136.03, 138.03, 140.03, 142.03, 144.03, 146.03, 148.03, 150.03, 152.03, 154.03, 156.03, 158.03, 160.03, 162.03, 164.03, 166.03, 168.03, 170.03, 172.03, 174.03, 176.03, 178.03, 180.03, 182.03, 184.03, 186.03, 188.03

(6) Laplace-Matrizen mit/ohne Shift

```
> # 1D-Laplace-Matrix
  c := 0.0:       # Shift
  n := 100:

> A1 := band([-1.0,2.0,-1.0],n):
  A1 := evalm(A1+c*diag(1$n)):

> # 2D-Laplace-Matrix
  c := 0.0:       # Shift
  n1 := 10:
  n := n1^2:

> T := band([-1.0,4.0,-1.0],n1):
  A2 := diag(T$n1):
  for i from 1 to n-n1 do
   A2[i,i+n1] := -1.0;
   A2[i+n1,i] := -1.0;
  end do:
  A2 := evalm(A2+c*diag(1$n)):
```

Dazu werden in den LGS geeignete rechte Seiten definiert, so dass eine einfache exakte Lösung, zum Beispiel x^* als Einsvektor, erhalten wird, die dann auch in Fehlerbetrachtungen einbezogen wird.

```
> xs1 := evalm(vector(n,[1.0$n])):
  b1 := vector(n,[1.0+c,c$(n-2),1+c]):
  # b1 := evalm(A1&*xs1):

> xs2 := evalm(vector(n,[1.0$n])):
  b21 := 2.0+c,(1.0+c)$(n1-2),2.0+c:
  b22 := 1.0+c,c$(n1-2),1.0+c:
  b2 := vector(n,[b21,seq(b22,i=2..n1-1),b21]):
  # b2 := evalm(A2&*xs2):
```

6.3.2 Aufwand und Zeitmessungen beim Umgang mit Matrizen

Bei der Einschätzung der Effizienz von Algorithmen werden oft drei wichtige Probleme betrachtet: Aufwand an Operationen, Rechenzeit, Speicherbedarf. Aber dazu kann man auch noch andere Aspekte einbeziehen, wie die Vorbereitung von Daten, die Auswertung der Ergebnisse, Dateiarbeit allgemein, Ergibtanweisungen, die Arbeit mit Steuerstrukturen u. ä. Aus dieser Vielfalt wird man sich jedoch auf einige wesentliche Fragen beschränken. Manchmal bleibt es dann bei der Bestimmnung der Anzahl der arithmetischen Operationen, also bei der Berechnung einer sogenannten Komplexitätsfunktion in Abhängigkeit vom Problemumfang (Dimension), sowie bei Rechenzeitvergleichen. Die Programmiersprachen und CAS bieten oft mehrere Möglichkeiten der Unterstützung bei Aufwandsbetrachtungen.
In Maple kann man Zeitmessungen vornehmen und dann auch bei Kenntnis der Komplexitätsfunktion näherungsweise eine durchschnittliche Anzahl von Gleitpunkt-Operationen (GP-Operationen) (*floating point operations*, $flops$) pro Sekunde für die benutzte Rechnerplattform ermitteln. Die Kommandos für die Zeit sind grob notiert

```
> ta := time():
  y := evalm(A&*x);   # Algorithmus
  te := time():       # bzw. tdiff := time()-ta:
  tdiff := te-ta:
```

Matlab bietet zwei Varianten, einmal zur Bestimmung der Anzahl der durchgeführten GP-Operationen mit `flops` sowie die Zeitmessungen mit `clock, etime`.

```
t0 = clock;
flops(0);
y = A*x;      % Algorithmus
fl = flops;   % oder einfach flops
t1 = etime(clock,t0);
```

Das Kommando `flops(0)` (nicht `flops = 0`) setzt den Zähler auf Null zurück. So kann die Eingabe von `flops(0)` unmittelbar vor dem Beginn des Algorithmus und der Aufruf `flops` gleich nach seiner Beendigung die $flops$ ermitteln. Die Funktion `clock` gibt die aktuelle Zeit mit der Genauigkeit auf eine Hundertstel Sekunde an. Mit zwei solchen Zeiten kann `etime` die abgelaufene Zeit (Zeitdifferenz) in Sekunden bestimmen. Seit der Version 4.0 gibt es die bequemere Variante einer Stoppuhr mit `tic, toc`, ab der Version 6 ist das Kommando `flops` nicht mehr verfügbar.

Bei der Anzahl der GP-Operationen wird man bestrebt sein, die verschiedenen arithmetischen Operationen extra zu zählen, denn auf den Rechnern sind die Zeiten dafür durchaus nicht gleich. Inzwischen dauert auf modernen Computern die Auswertung einer Addition, Subtraktion oder Multiplikation ungefähr gleich lang, während die Division schon mehr Zeit braucht und die Berechnung von Standardfunktionen natürlich wesentlich länger dauern kann.

6.3.3 Matrixgenerierung sowie Operationen auf Matrizen

Möglichkeiten der Generierung von Matrizen und Vektoren sind im Kap. 6.3.1 schon gezeigt worden, wobei noch keine Aufwandsbetrachtungen durchgeführt worden sind.

In den AV spielt die Matrix-Vektor-Multiplikation ein wichtige Rolle und macht zusammen mit der Iterationsanzahl den wesentlichen Aufwand aus. Die klassische Matrix-Vektor-Multiplikation n-dimensionaler Matrizen und Vektoren $y = Ax$ braucht n^2 Multiplikationen und $n^2 - n$ Additionen.
Diesen Aufwand drücken wir durch die Komplexitätsfunktion $\mathcal{K}(n) = 2n^2 - n$ aus. Wir notieren dies auch in der Form

$$\mathcal{K}(n) = \mathcal{O}(2n^2),$$

wo nur der führende Ausdruck (Potenz) steht. Im Operationsmix $\{+, *\}$ ist der Aufwand $\mathcal{O}(n^2)$ und er muss also verdoppelt werden.
Für eine Bandmatrix mit der Bandbreite $bw = \alpha + \beta + 1 \ll n$ erhält man die Komplexität $\mathcal{O}(2n(\alpha + \beta + 1))$, im Fall einer Diagonalmatrix die Größe $\mathcal{O}(n)$.
Somit wird schon deutlich, dass man einfache Matrixstrukturen auch adäquat verarbeiten sollte, um den Aufwand klein zu halten.

Mit dem CAS Maple Version 9.5 untersuchen wir nun den Aufwand und machen dazu Zeitmessungen auf einem PC Pentium IV (2.4 GHz) für folgende Aufgaben:
– Generierung/Erzeugung von Matrizen,
– Umspeichern von Matrizen,
– Matrix-Vektor-Multiplikation,
– eventuell Lösung des LGS und Berechnung von A^{-1}.

Dabei nehmen wir verschiedene Matrixstrukturen. Außerdem sind exakte Rechnungen (mit Rationalarithmetik) wie solche in einer Gleitpunktarithmetik (GPA) möglich. Um Vergleiche eventuell zu höheren Programmiersprachen und Matlab anzustellen, wo das *double*-Format gebräuchlich ist, kann man in Maple die Rechengenauigkeit `Digits:=16` nehmen.
Will man die exakte (symbolische) Rechnung in Maple bezüglich der Ergebnisgenauigkeit möglichst gut nachvollziehen, kann man die Einstellung im Kommando `Digits:=...` entsprechend erhöhen. Dabei gibt es hier keine signifikanten Unterschiede, ob man z. B. mit `Digits:=30` oder `Digits:=50`, rechnet, jedoch schon sichtbare Zeitdifferenzen zwischen Rechnungen mit `Digits:=16` bzw. `Digits:=50`.

Exakte Berechnungen mit Maple werden meist nicht zum Vergleich herangezogen, auch wenn sie bei kleindimensionierten Problemen oder einfachen Matrixstrukturen in der Rechenzeit manchmal mit der GPA konkurrieren können. Unsere Betrachtungen beziehen sich zwar nur auf einige Situationen, sind aber durchaus hinreichend aussagekräftig und können zu Vergleichen herangezogen werden.

Nicht zu vernachlässigen sind Einflüsse durch Auslastung/Belastung des Rechners bzw. des Rechnernetzes, was sich auch auf Rechenzeiten in numerischen Algorithmen auswirken kann. So sind die Rechnungen mehrmals wiederholt worden, um dann die besten Zeiten abzulesen.

Rechnungen in Maple (Datei: *timearr2.mws*)

Erzeugung von GP-Zahlen in Matrizen, die exakt (symbolisch) definiert sind

Berechnungsfunktionen `evalm, evalf`

```
> H := hilbert(4);
  x := vector(4,[seq(i,i=1..4)]);
  y := evalm(H&*x);

  Digits := 16:
  Hf := evalf(H);          # noch keine GPZ erzeugt
  Hf := evalf(evalm(H));   # GPZ
  Hf := evalf(hilbert(4));
  xf := evalf(x);          # noch keine GPZ erzeugt
  xf := evalf(evalm(x));   # GPZ
  xf := evalf(vector(4,[seq(i,i=1..4)]));

  y := Hf&*xf;
  y;
  y := evalm(Hf&*xf);      # Berechnung
```

$$H := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

$$x := [1, 2, 3, 4]$$

$$y := \left[4, \frac{163}{60}, \frac{21}{10}, \frac{241}{140}\right]$$

$$Hf := H$$

$$Hf := \begin{bmatrix} 1. & 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 \\ 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 \\ 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 \\ 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 & 0.1428571428571429 \end{bmatrix}$$

$$Hf := \begin{bmatrix} 1. & 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 \\ 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 \\ 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 \\ 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 & 0.1428571428571429 \end{bmatrix}$$

$$xf := x$$

$$xf := [1., 2., 3., 4.]$$

$$xf := [1., 2., 3., 4.]$$

$$y := Hf \ \&* \ xf$$

$$Hf \ \&* \ xf$$

$$y := [4.000000000000000, 2.716666666666667, 2.100000000000000, 1.721428571428572]$$

Achtung: Die komponentenweise Belegung eines Tableaus heißt noch nicht, dass man es mit einer Matrix zu tun hat.

```
> for i from 1 to 4 do
   for j from 1 to 4 do
    H1[i,j] := 1.0/(i+j-1.0);
   end do:
  end do:
  H1[1,1];
  H1;
  evalm(H1);
  H11:=evalm(H1);
  evalm(H1&*x);
  whattype(H1);
  type(H1,symbol),type(H1,table),type(H1,array),type(H1,matrix);
```

$$1.000000000000000$$

$$H1$$

$$H1$$

$$H11 := H1$$

$$H1\&*[1,\ 2,\ 3,\ 4]$$

$$symbol$$

$$true,\ true,\ false,\ false$$

Deshalb vorher: Matrixdefinition

```
> H2 := matrix(4,4,[]);
  for i from 1 to 4 do
   for j from 1 to 4 do
    H2[i,j] := 1.0/(i+j-1.0);
   end do:
  end do:
  H2;
  evalm(H2);
  evalm(H2&*x);
  whattype(H2);
  type(H2,symbol),type(H2,table),type(H2,array),type(H2,matrix);
```

$$H2 := \text{array}(1..4, 1..4, [])$$

$$H2$$

$$\begin{bmatrix} 1.000000000000000 & 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 \\ 0.5000000000000000 & 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 \\ 0.3333333333333333 & 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 \\ 0.2500000000000000 & 0.2000000000000000 & 0.1666666666666667 & 0.1428571428571429 \end{bmatrix}$$

$$[4.000000000000000,\ 2.716666666666667,\ 2.100000000000000,\ 1.721428571428572]$$

$$symbol$$

$$true,\ true,\ true,\ true$$

Generierung und Umspeichern von Matrizen

Die Rechnungen wurden auf dem PC Pentium IV (2.4 GHz) durchgeführt.

– Diagonalmatrizen, z. B. Van der Vorst Matrix (`DM`)
– Tridiagonalmatrizen vom 1D-Laplace-Operator auf äquidistantem Gitter mit 3-Punkte-Differenzenstern (`TDM`)
– Blocktridiagonalmatrizen vom 2D-Laplace-Operator auf quadratischem Gitter mit 5-Punkte-Differenzenstern (`BTD`)
– (voll besetzte) Hilbert-Matrix (`H-M`)

Matrizen der Dimensionen 9,16,25,...,625 in verschiedenen Formaten, das Umspeichern erfolgt stets wie eine voll besetzte Matrix (`vM`)

```
> Digits := 50:   # 16
> kmax := 25:
> time1 := vector(kmax,[0$kmax]):
  time2 := evalm(time1): time3 := evalm(time1): time4 := evalm(time1):
  time5 := evalm(time1): time6 := evalm(time1): time7 := evalm(time1):
  time8 := evalm(time1): time9 := evalm(time1):

> for n1 from 3 to kmax do
   n1q := n1^2:

   sta := time():
   B91 := band([-1.0,2.0,-1.0],n1q):  # TDM  Tridiagonalmatrix
   time1[n1] := time()-sta;

   sta := time():
   B92 := evalm(B91):
   time2[n1] := time()-sta;

   sta := time():
   T := band([-1.0,4.0,-1.0],n1):
   B93 := diag(T$n1):
   for i from 1 to n1q-n1 do
    B93[i,i+n1] := -1.0;
    B93[i+n1,i] := -1.0;
   end do:                              # BTD  Blocktridiagonalmatrix
   time3[n1] := time()-sta;

   sta := time():
   B94 := evalm(B93):
   time4[n1] := time()-sta;

   sta := time():
   B95 := evalf(evalm(hilbert(n1q))): # H-M  Hilbert-Matrix
   # B95 := evalm(hilbert(n1q)):
   time5[n1] := time()-sta;

   sta := time():
   B96 := evalm(B95):
   time6[n1] := time()-sta;
```

```
 sta := time():
 for i from 1 to n1q do
  for j from 1 to n1q do
   B97[i,j] := 1.0/(i+j-1.0);
  end do:
 end do:                                  # H-M  Hilbert-Matrix
 time7[n1] := time()-sta;

 B98:=matrix(n1q,n1q,[]):
 sta := time():
 for i from 1 to n1q do
  for j from 1 to n1q do
   B98[i,j] := 1.0/(i+j-1.0);
  end do:
 end do:                                  # H-M  Hilbert-Matrix
 time8[n1] := time()-sta;

 sta := time():
 B99 := evalm(B98);
 time9[n1] := time()-sta;
end do:

printf(‘Zeiten in Sekunden\n‘):
printf(‘      Gen(TDM) Umsp(vM) Gen(BTD) Umsp(vM) Gen(H-M) Umsp(vM)
                                         Gen(H-M)  Gen(H-M) Umsp(vM)\n‘):
printf(‘
                                         p.H.table p.H.matrix \n‘):
printf(‘Dim.                                    time[3..kmax] \n‘):
printf(‘n1^2   time1    time2    time3    time4    time5    time6
                                            time7     time8    time9\n‘):
for k from 3 to kmax do
 printf(‘%3d  %7.3f  %7.3f  %7.3f  %7.3f  %7.3f  %7.3f  %7.3f
         %7.3f  %7.3f\n‘,k^2,time1[k],time2[k],time3[k],time4[k],
         time5[k],time6[k],time7[k],time8[k],time9[k]):
end do:
```

Digits := 50

```
Zeiten in Sekunden
      Gen(TDM) Umsp(vM) Gen(BTD) Umsp(vM) Gen(H-M) Umsp(vM) Gen(H-M)  Gen(H-M) Umsp(vM)
                                                             p.H.table p.H.matrix
Dim.                                 time[3..kmax]
n1^2   time1    time2    time3    time4    time5    time6    time7     time8    time9
-------------------------------------------------------------------------------------
  9    0.000    0.000    0.000    0.000    0.000    0.000    0.000     0.000    0.000
 16    0.000    0.000    0.016    0.000    0.000    0.000    0.000     0.015    0.000
 25    0.000    0.000    0.000    0.016    0.063    0.015    0.000     0.016    0.000
 36    0.000    0.015    0.000    0.000    0.032    0.000    0.015     0.016    0.016
 49    0.000    0.015    0.047    0.016    0.047    0.015    0.031     0.032    0.078
 64    0.000    0.015    0.016    0.016    0.062    0.094    0.047     0.047    0.093
 81    0.000    0.047    0.000    0.032    0.156    0.062    0.141     0.141    0.062
100    0.000    0.047    0.016    0.109    0.219    0.078    0.203     0.187    0.079
121    0.000    0.140    0.016    0.140    0.297    0.188    0.328     0.266    0.203
144    0.000    0.187    0.000    0.125    0.531    0.172    0.500     0.422    0.250
169    0.016    0.219    0.015    0.235    0.703    0.343    0.594     0.625    0.344
196    0.000    0.297    0.015    0.391    0.891    0.453    0.859     0.860    0.531
225    0.015    0.360    0.109    0.391    1.219    0.703    1.140     1.172    0.672
256    0.000    0.578    0.031    0.719    1.719    0.891    1.702     1.735    0.875
```

289	0.125	0.734	0.047	0.906	2.500	1.281	2.282	2.438	1.422
324	0.000	1.093	0.032	1.297	3.140	1.797	3.172	3.391	1.765
361	0.000	1.720	0.046	1.610	4.562	2.438	4.547	4.578	2.656
400	0.000	2.156	0.282	2.250	6.125	3.359	6.484	6.750	3.344
441	0.016	3.093	0.079	2.985	7.718	4.532	8.437	8.781	4.578
484	0.000	3.672	0.390	3.891	10.125	5.890	11.578	11.953	6.360
529	0.000	5.234	0.469	5.078	13.781	7.781	15.671	15.969	8.281
576	0.000	6.516	0.531	6.891	17.671	10.093	20.532	20.922	10.672
625	0.000	8.609	0.625	9.031	22.890	13.828	27.375	28.501	14.765

Digits := 16

Zeiten in Sekunden

	Gen(TDM)	Umsp(vM)	Gen(BTD)	Umsp(vM)	Gen(H-M)	Umsp(vM)	Gen(H-M) p.H.table	Gen(H-M) p.H.matrix	Umsp(vM)
Dim.				time[3..kmax]					
n1^2	time1	time2	time3	time4	time5	time6	time7	time8	time9
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
25	0.000	0.000	0.000	0.000	0.000	0.016	0.000	0.000	0.016
36	0.000	0.000	0.000	0.000	0.016	0.015	0.016	0.015	0.000
49	0.000	0.015	0.000	0.016	0.031	0.016	0.031	0.031	0.015
64	0.000	0.015	0.000	0.016	0.062	0.078	0.047	0.047	0.032
81	0.000	0.031	0.000	0.032	0.141	0.047	0.094	0.078	0.062
100	0.000	0.047	0.015	0.047	0.188	0.093	0.141	0.156	0.062
121	0.000	0.078	0.016	0.078	0.281	0.172	0.219	0.265	0.172
144	0.000	0.110	0.000	0.172	0.484	0.188	0.344	0.359	0.235
169	0.015	0.204	0.015	0.219	0.735	0.344	0.515	0.500	0.406
196	0.000	0.281	0.015	0.297	1.000	0.516	0.813	0.734	0.516
225	0.000	0.453	0.032	0.375	1.453	0.703	1.125	1.157	0.687
256	0.000	0.594	0.031	0.625	1.891	0.969	1.578	1.609	1.000
289	0.000	0.828	0.031	0.922	2.734	1.406	2.218	2.266	1.281
324	0.000	1.203	0.047	1.125	3.406	1.969	3.125	3.016	1.938
361	0.015	1.532	0.047	1.797	4.890	2.641	4.204	4.421	2.704
400	0.000	2.141	0.047	2.484	6.563	3.640	5.953	6.250	3.656
441	0.000	2.844	0.078	3.313	8.422	4.860	7.733	8.548	4.516
484	0.015	3.656	0.375	3.875	11.079	6.218	10.828	11.235	6.375
529	0.000	5.328	0.093	5.579	14.250	8.516	14.782	15.109	8.453
576	0.000	6.750	0.547	7.156	18.828	10.953	19.376	20.265	11.078
625	0.016	8.953	0.094	9.579	23.593	14.235	25.985	26.874	14.500

Bemerkenswert ist die schnelle Generierung von sparsen Matrizen, wie es die Tridiagonal- bzw. Blocktridiagonalmatrizen sind. Lange dauern ihre Umspeicherungen, wobei die vielen Nulleinträge sich schon bemerkbar machen im Vergleich zur Umspeicherung von voll besetzten Matrizen.
Noch mehr Zeit kosten Generierung und Umspeicherung großer voll besetzter Matrizen. Dabei ist es günstiger, vordefinierte Versionen (in exakter Form) zu nehmen und nicht selbst die Matrixelemente als GP-Zahlen entsprechender Genauigkeit einzutragen (per Hand). In älteren Maple-Versionen war es zum Teil umgekehrt.

Bei der Generierung der Hilbert-Matrix kostet die zusätzliche Umwandlung ihrer Einträge in GP-Zahlen mittels dem `evalf`-Kommando, also
`B95:=evalf(evalm(hilbert(n1q))):` anstelle von `B95:=evalm(hilbert(n1q)):`, noch etwas Zeit.

Die Genauigkeit der GPA (`Digits:=...`) ist für den Zeitaufwand hier nicht ausschlaggebend, es sei denn bei der Matrixbelegung werden die Elemente durch komplizierte Ausdrücke ermittelt.

Zeitmessungen zur Matrix-Vektor-Multiplikation
(1) Voll besetzte Matrix mit verschiedenen Formaten und Strategien

```
> Digits := 16:  # 50
> kmax := 25:                       # kmax := 16:
  ftime := vector(kmax,[0$kmax]):
  etime := evalm(ftime):

> for k from 5 by 5 to kmax do    # for k from 1 by 1 to kmax do
   m := k^2:                      #  m := 40*k:

   Hf := evalf(evalm(hilbert(m)));    # Hilbert-Matrix
   xf := evalf(evalm(vector(m,[seq(i,i=1..m)])));

   # Variante  float
   sta := time():
   yf := evalm(Hf&*xf);
   ftime[k] := time()-sta:

   # Variante  float+loop
   yf:=vector(m,[]):
   sta := time():
   for i from 1 to m do
    z := 0.0;
    for j from 1 to m do z := z+Hf[i,j]*xf[j]; end do;
    yf[i] := z;
   end do;
   etime[k] := time()-sta:
  end do:

  printf(‘Zeiten in Sekunden fuer y=Ax, A(m,m) voll besetzt\n‘):
  printf(‘       float        float+loop\n‘):
  printf(‘m   ftime[1..kmax] etime[1..kmax]\n‘):
  for k from 5 by 5 to kmax do
   printf(‘%3d  %7.3f        %7.3f\n‘,k^2,ftime[k],etime[k]):
  end do:
```

Darstellung der Ausgaben in zusammengefasster Form

```
Zeiten in Sekunden fuer y=Ax, A(m,m) voll besetzt

           Digits=16              Digits=50
       float   float+loop      float   float+loop
  m   ftime[k]  etime[k]      ftime[k]  etime[k]
-------------------------------------------------
=k^2
 25    0.015     0.016         0.031     0.000
100    0.219     0.110         0.281     0.219
225    1.437     0.954         2.359     1.297
400    6.578     3.953        11.563     5.719
625   22.765    14.780        49.656    20.735
```

	Digits=16		Digits=50	
	float	float+loop	float	float+loop
m	ftime[k]	etime[k]	ftime[k]	etime[k]
=40k				
40	0.031	0.109	0.031	0.016
80	0.156	0.110	0.157	0.093
120	0.344	0.234	0.407	0.250
160	0.578	0.469	0.859	0.484
200	1.078	0.672	1.532	0.890
240	1.703	1.047	2.640	1.422
280	2.484	1.641	4.172	2.141
320	3.578	2.438	6.203	3.016
360	4.921	3.219	8.360	4.140
400	6.594	3.984	11.640	5.516
440	8.219	5.172	16.062	7.437
480	10.531	6.766	21.250	9.564
520	13.516	8.078	26.172	12.171
560	16.297	9.985	35.219	15.141
600	20.719	12.891	45.031	18.734
640	25.390	14.813	57.531	23.907

Die exakte Multiplikation, die hier nicht mit aufgeführt ist, dauert natürlich am längsten, denn die dabei zu speichernden rationalen Zahlen haben immer mehr Dezimalstellen.
In der GPA führen höhere Genauigkeiten generell zu längeren Rechenzeiten.
Warum aber bei der GPA `Digits:=16,50` die nutzerdefinierten Schleifen zu einer etwas kürzeren Rechenzeit führen, ist nicht nur damit zu erklären, dass eine einfache Hilfsvariable zur Erzeugung der Skalarprodukte genommen wurde. Denn eine innere Schleife der Form

```
yf[i]:=0.0; for j from 1 to m do yf[i]:=yf[i]+Hf[i,j]*xf[j]; end do;
```

vergrößert nur geringfügig die Zeiten in der Spalte `etime`.

Damit bringt es Maple mit der GP-Genauigkeit `Digits:=16` (Format *double*) pro Sekunde auf ca. 60 000..120 000 $flops$ + Aufwand für Steuerung.

Mit Matlab und den Anweisungen `tic; A*b; toc` mit Zeitmessung ist man um Größenordnungen schneller und erhält ca. 200 000 000 $flops$ per *sec*.

(2) Bandstrukturen im Vergleich zur voll besetzten Matrix

Matrizen der Dimension $n1q = 100, 225, 400, 625$

```
> Digits := 50:
> n1 := 10:       # 15, 20, 25
  n1q := n1^2:
> B9d := evalm(diag(seq(2.0,i=1..n1q))):   # DM   Diagonalmatrix
  B9t := band([-1.0,2.0,-1.0],n1q):        # TDM  Tridiagonalmatrix
```

```
> T := band([-1.0,4.0,-1.0],n1):
  B9m := diag(T$n1):
  for i from 1 to n1q-n1 do
   B9m[i,i+n1] := -1.0;
   B9m[i+n1,i] := -1.0;
  end do:
  evalm(B9m):                                  # BTM  Blocktridiagonalmatrix
  B9n := evalf(evalm(hilbert(n1q))):           # H-M  Hilbert-Matrix

> x := evalf(evalm(vector(n1q,[1.0$n1q]))):
> sta := time():  y := evalm(B9d&*x):  print(time()-sta);
  sta := time():  y := evalm(B9t&*x):  print(time()-sta);
  sta := time():  y := evalm(B9m&*x):  print(time()-sta);
  sta := time():  y := evalm(B9n&*x):  print(time()-sta);
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) wie voll besetzt behandelt
                    m=n1q
Matrix   100     225     400     625
-------------------------------------
DM      0.079   0.640   3.250  11.406
TDM     0.110   0.688   3.125  11.736
BTD     0.109   0.687   3.000  12.281
H-M     0.219   1.657  11.031  32.578
```

Die Matrix-Vektor-Multiplikationen für eine Diagonal-, Tridigonal- bzw. Blocktridiagonalmatrix dauern in der GPA, als volle Matrizen interpretiert, etwa gleich lang. Man hat etwas Zeiteinsparung durch viele Nullen in der Matrix im Vergleich mit richtig voll besetzten Matrizen.

Somit entstehen in numerischen Algorithmen mit vielen Matrix-Vektor- oder Matrix-Matrix-Produkten bei großen Dimensionen enorme Rechenzeiten.

(3) Bandmatrizen mit Berücksichtigung ihrer Strukturen

(3.1) Diagonalmatrizen

Erhebliche Zeiteinsparungen hat man durch die Berücksichtigung der Diagonalstruktur der Matrizen.

Wir vergleichen 3 Varianten von $y = Ax$ bei Behandlung der Matrix $A(m, m)$ als voll besetzte bis zur optimalen Berechnungsformel $y_i = a_{ii}x_i,\ i = 1, 2, ..., m$.

```
> Digits := 50:
> kmax := 10:
  ftime := vector(kmax,[0$kmax]):      # zum Speichern der Zeiten
  etime :=evalm(stime): dtime :=evalm(stime):

> for k from 1 to kmax do
   m := 100*k: i := 'i':
   C1 := evalm(diag(seq(-11+2*i,i=1..m))):   # Van der Vorst Matrix
   x := vector(m,[seq(i,i=1..m)]):
```

```
    C1f := evalf(evalm(C1)):
    xf := evalf(evalm(x)):

    # float
    sta := time():
    yf := evalm(C1f&*xf);
    ftime[k] := time()-sta:

    # loop+float
    yf := vector(m,[]):
    sta := time():
    for i from 1 to m do
     z := 0.0;
     for j from 1 to m do z:=z+C1f[i,j]*xf[j]; end do;
     yf[i] := z;
    end do;
    etime[k] := time()-sta:

    # loop+float+diagonal
    yf := vector(m,[]):
    sta := time():
    for i from 1 to m do
     yf[i] := C1f[i,i]*xf[i];
    end do;
    dtime[k] := time()-sta:
   end do:

   printf(‘Zeiten in Sekunden fuer y=Ax, A(m,m) diagonal\n‘):
   printf(‘      float       float+loop     float+loop+diagonal\n‘):
   printf(‘ m   ftime[1..kmax]   etime[1..kmax]      dtime[1..kmax]\n‘):
   for k from 1 to kmax do
    printf(‘%3d %7.3f        %7.3f        %7.3f        %7.3f\n‘,
            k*100,ftime[k],etime[k],dtime[k]):
   end do:
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) diagonal
         float   float+loop float+loop+diagonal
  m     ftime[k]   etime[k]   dtime[k]
-----------------------------------------------
=100k
 100     0.109      0.047      0.000
 200     0.593      0.250      0.000
 300     1.672      0.797      0.000
 400     3.563      1.672      0.000
 500     6.625      3.000      0.000
 600    11.031      4.469      0.000
 700    18.109      7.578      0.000
 800    26.625     10.453      0.000
 900    40.703     15.484      0.016
1000    60.812     22.453      0.000
```

Wir ergänzen noch, dass bei nur wenigen Operationen (z. B. wegen der Diagonalform) die Rechenzeit der exakten (symbolischen) Arithmetik mit der der GPA (1. Spalte `float`) konkurrieren kann. Dazu betrachten wir also die Diagonalmatrix $C1$.

```
    # symbolisch
  > sta := time(): yf := evalm(C1&*x); stime[k] := time()-sta:
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) diagonal
                                      m=100k
               100  200  300  400  500  600   700   800   900   1000
-------------------------------------------------------------------
float ftime[k] 0.1  0.6  1.7  3.6  6.6  11.0  18.1  26.6  40.7  60.8
symb. stime[k] 0.1  0.7  1.8  4.3  9.0  15.8  27.4  42.5  63.8  98.1
```

(3.2) Tridiagonalmatrizen

Ähnliche Zeiteinsparungen hat man durch die Berücksichtigung der Tridiagonalstruktur der Matrizen. Wir vergleichen analog zur Vorgehensweise in Punkt (3.1) auch hier 3 Varianten von $y = Ax$ bei Behandlung der Matrix $A(m, m)$, definiert als `band([-1.0,2.0,-1.0],m)`, als voll besetzte bis zur optimalen elementweisen Berechnungsformel.

```
    # loop+float+tridiagonal
  > yf[1] := B9t[1,1]*xf[1]+B9t[1,2]*xf[2];
    for i from 2 to m-1 do
     yf[i] := B9t[i,i-1]*xf[i-1]+B9t[i,i]*xf[i]+B9t[i,i+1]*xf[i+1];
    end do;
    yf[m] := B9t[m,m-1]*xf[m-1]+B9t[m,m]*xf[m];
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) tridiagonal
          float   float+loop float+loop+tridiag.
  m     ftime[k]   etime[k]   dtime[k]
-----------------------------------------------
=100k
 100     0.094      0.078      0.000
 200     0.359      0.203      0.000
 300     1.047      0.515      0.000
 400     2.296      0.985      0.015
 500     4.000      1.546      0.000
 600     6.531      2.297      0.000
 700    10.797      3.234      0.015
 800    15.906      4.375      0.015
 900    23.000      5.703      0.015
1000    33.797      7.437      0.015
```

(3.3) Blocktridiagonalmatrizen
Genauso verfahren wir bei den 2D-Laplace-Matrizen, definiert gemäß

```
> n1 := 10:    #  15,20,25,30,35
  n1q := n1^2:

> T := band([-1.0,4.0,-1.0],n1):
  B9m := diag(T$n1):
  for i from 1 to n1q-n1 do
   B9m[i,i+n1] := -1.0;
   B9m[i+n1,i] := -1.0;
  end do:
```

Wir vergleichen wiederum 3 Varianten von $y = Ax$ bei Behandlung der Matrix $A(m, m)$ als voll besetzte bis zur optimalen Berechnungsformel.

```
  # loop+float+blocktridiagonal
> yf[1] := B9m[1,1]*xf[1]+B9m[1,2]*xf[2]+B9m[1,1+n1]*xf[1+n1];
  for i from 2 to n1 do
   yf[i] := B9m[i,i-1]*xf[i-1]+B9m[i,i]*xf[i]
           +B9m[i,i+1]*xf[i+1]+B9m[i,i+n1]*xf[i+n1];
  end do;
  for i from n1+1 to n1q-n1 do
   yf[i] := B9m[i,i-n1]*xf[i-n1]+B9m[i,i-1]*xf[i-1]
           +B9m[i,i]*x[i]+B9m[i,i+1]*xf[i+1]+B9m[i,i+n1]*xf[i+n1];
  end do;
  for i from n1q-n1+1 to n1q-1 do
   yf[i] := B9m[i,i-n1]*xf[i-n1]+B9m[i,i-1]*xf[i-1]
           +B9m[i,i]*xf[i]+B9m[i,i+1]*xf[i+1];
  end do;
  yf[n1q] := B9m[n1q,n1q-n1]*xf[n1q-n1]+B9m[n1q,n1q-1]*xf[n1q-1]
            +B9m[n1q,n1q]*xf[n1q];
```

```
Zeiten in Sekunden fuer y=Ax, A(m,m) blocktridiagonal
         float   float+loop float+loop+blocktrid.
m=n1^2  ftime[k]   etime[k]    dtime[k]
-------------------------------------------------
 100     0.094      0.047       0.000
 225     0.500      0.282       0.015
 400     2.282      1.047       0.000
 625     7.875      2.797       0.015
 900    23.906      6.390       0.032
1225    66.859     13.344       0.031
```

Die nutzerdefinierten Schleifen mit der optimalen elementweisen Berechnung liefern die sichtbar kleinsten Rechenzeiten. Diese sollten in AV auch Verwendung finden.
Das volle Matrix-Vektor-Produkt mit Schleifensteuerung ist mehr als doppelt so schnell wie die Ausführung des Kommandos `y:=evalm(A&*x);`

6.3.4 Aufwandsbetrachtungen für spezielle Matrizen

Die nutzerdefinierten Schleifen mit der optimalen elementweisen Berechnung bei der Matrix-Vektor-Multiplikation $y = Ax, A = A(n,n)$, sind die Grundlage für die folgende Prozedur.
Dabei werden mittels des Eingangsparameters `trid` die Fälle der Tridiagonalstruktur (`trid=3` und `n1` beliebig) bzw. Blocktridiagonalstruktur (`trid=5` und `n=n1^2`) der Matrix unterschieden.

```
  # Prozedur fuer Matrix-Vektor-Multiplikation y=Ax

> Amx:=proc(n::posint,n1::posint,trid::posint,A::matrix,x::vector)
   local i,y;

  y := vector(n,[seq(0.0,i=1..n)]):

  if trid=3 then       # Tridiagonalmatrix   A(n,n)
   y[1] := A[1,1]*x[1]+A[1,2]*x[2];
   for i from 2 to n-1 do
    y[i] := A[i,i-1]*x[i-1]+A[i,i]*x[i]+A[i,i+1]*x[i+1];
   end do;
   y[n] := A[n,n-1]*x[n-1]+A[n,n]*x[n];

  elif trid=5 then     # Blocktridiagonalmatrix   A(n,n)=A(n1^2,n1^2)
   y[1] := A[1,1]*x[1]+A[1,2]*x[2]+A[1,1+n1]*x[1+n1];
   for i from 2 to n1 do
    y[i] := A[i,i-1]*x[i-1]+A[i,i]*x[i]+A[i,i+1]*x[i+1]+A[i,i+n1]*x[i+n1];
   end do;
   for i from n1+1 to n-n1 do
    y[i] := A[i,i-n1]*x[i-n1]+A[i,i-1]*x[i-1]+A[i,i]*x[i]
           +A[i,i+1]*x[i+1]+A[i,i+n1]*x[i+n1];
   end do;
   for i from n-n1+1 to n-1 do
    y[i] := A[i,i-n1]*x[i-n1]+A[i,i-1]*x[i-1]+A[i,i]*x[i]+A[i,i+1]*x[i+1];
   end do;
   y[n] := A[n,n-n1]*x[n-n1]+A[n,n-1]*x[n-1]+A[n,n]*x[n];
  else
  end if;

  evalm(y);

 end:
```

Zeitmessungen von Operationen für Matrizen (Datei: *timearr3.mws*)

Dabei testen wir sowohl Kommandos aus dem "alten" Maple-Paket `linalg` als auch aus `LinearAlgebra`. Letzteres beinhaltet effiziente und robuste Algorithmen der numerischen linearen Algebra. Dabei verzichten wir hier jedoch auf die Möglichkeiten der expliziten Nutzung von Matrixstrukturen durch entsprechende optionale Parameter, die zu einer zusätzlichen Effizienz bezüglich Aufwand, Speicherplatz und Rechenzeit führen können.

Wir ermitteln für die Test-Matrizen die Zeiten zu folgenden Kommandos:

– Generierung der Matrix A

– Lösung des LGS $Ax = b$
`linalg[linsolve](A,b)`
`LinearAlgebra[LinearSolve](A,b)`

– Matrix-Vektor-Produkt Ax
`evalm(A&*x)`
`LinearAlgebra[MatrixVektorMultiply](A,x)`
Prozeduraufruf `Amx(n,n1,trid,A,x)`

– Berechnung der Inversen A^{-1}
`linalg[inverse](A)`
`LinearAlgebra[LinearSolve](A,I)`

Die Rechenzeiten mit zusätzlichen Zuweisungen des Ergebnisses an eine Variable unterscheiden sich nicht signifikant.

(1) Tridiagonalmatrix
Maple-Kommandos und Rechenzeiten für die 1D-Laplace-Matrix mit einem Shift c

```
  # Tridiagonalmatrix mit Shift
> Digits := 50: # 30, 40, 50
  c := 0.0:     # Shift
  trid := 3:
  n1 := 1:      # nicht relevant bei TDM
> n := 625:     # 100, 200, 225, 625, 900, ...

> sta := time():
  A1 := band([-1.0,2.0,-1.0],n):
  A1 := evalm(A1+c*diag(1.0$n)):
  print(time()-sta);

> xs1 := evalm(vector(n,[1.0$n])):
  b1 := evalm(vector(n,[1.0+c,c$(n-2),1.0+c])):

  #  The LinearAlgebra package is an efficient and robust suite
  #  of commands for doing computational linear algebra
> In := Matrix(n,n,shape=identity,datatype=float):  # Einheitsmatrix
  A1m := Matrix(n,n,(i,j)->A1[i,j],datatype=float):
  # nicht  A1m:=evalm(A1):, dann ist A1m vom Typ matrix bzw. array
  b1m := Vector(n,i->b1[i],datatype=float):

> sta := time(): linsolve(A1,b1):               print(time()-sta);
  sta := time(): xs1 := linsolve(A1,b1):        print(time()-sta);
  sta := time(): LinearSolve(A1m,b1m):          print(time()-sta);
  sta := time(): xs1m := LinearSolve(A1m,b1m):  print(time()-sta);

> sta := time(): evalm(A1&*xs1):                      print(time()-sta);
  sta := time(): b1s := evalm(A1&*xs1):               print(time()-sta);
  sta := time(): MatrixVectorMultiply(A1m,xs1m):      print(time()-sta);
  sta := time(): b1sm:=MatrixVectorMultiply(A1m,xs1m):print(time()-sta);
```

```
  # Prozedur  Amx()
> sta := time(): Amx(n,n1,trid,A1,xs1):         print(time()-sta);
  sta := time(): b1s := Amx(n,n1,trid,A1,xs1):  print(time()-sta);

> sta := time(): inverse(A1):                   print(time()-sta);
  sta := time(): A1inv := inverse(A1):          print(time()-sta);
  sta := time(): LinearSolve(A1m,In):           print(time()-sta);
  sta := time(): A1minv := LinearSolve(A1m,In): print(time()-sta);
```

Zeiten in Sekunden fuer A1(n,n) tridiagonal mit Shift c=0
(MVMultiply = MatrixVectorMultiply)

	n=100 Digits			n=200 Digits			n=225 Digits		n=625 Digits		n=900 Digits
	30	40	50	50	30	40	50	30	40	50	50
Generierung A1	0.046	0.047	0.047	0.312	0.313	0.328	0.328	3.813	4.125	4.265	11.876
linsolve(A1,b1)	0.235	0.235	0.234	1.204	1.438	1.484	1.485	22.032	23.562	23.501	76.704
LinearSolve(A1m,b1m)	0	0	0	0.031	0.031	0.031	0.046	0.250	0.265	0.265	1.031
evalm(A1&*xs1)	0.063	0.078	0.109	0.641	0.672	0.672	0.763	9.718	11.062	10.094	32.234
MVMultiply(A1m,xs1m)	0	0	0	0	0	0	0	0.078	0.093	0.078	0.172
Prozedur Amx()	0	0	0	0	0	0	0	0.015	0.015	0.016	0.031
inverse(A1)	5.172	5.625	5.687	61.876	80	85	85	4150	4423	4369	22898
LinearSolve(A1m,In)	0.344	0.532	0.547	3.203	3.486	4.016	4.296	63	80	90	340

(2) Blocktridiagonalmatrix

Analog gehen wir für die 2D-Laplace-Matrix unter Einbeziehung eines Shift c vor.

```
  # Blocktridiagonalmatrix mit Shift

> Digits := 50:   # 30, 40, 50
  c := 0.0:       # Shift
  trid := 5:

> n1 := 25:       # 10, 15, 25, 30
  n := n1^2:
> sta := time():
  T := band([-1.0,4.0,-1.0],n1):
  A2 := diag(T$n1):
  for i from 1 to n-n1 do
   A2[i,i+n1] := -1.0;
   A2[i+n1,i] := -1.0;
  end do:
  A2 := evalm(A2+c*diag(1.0$n)):
  print(time()-sta);

> xs2 := evalm(vector(n,[1.0$n])):
  b2 := evalm(A2&*xs2):

> In := Matrix(n,n,shape=identity,datatype=float):  # Einheitsmatrix
  A2m := Matrix(n,n,(i,j)->A2[i,j],datatype=float):
  b2m := Vector(n,i->b2[i],datatype=float):
```

```
> sta := time(): linsolve(A2,b2):                print(time()-sta);
  sta := time(): xs1:=linsolve(A2,b2):           print(time()-sta);
  sta := time(): LinearSolve(A2m,b2m):           print(time()-sta);
  sta := time(): xs2m := LinearSolve(A2m,b2m):   print(time()-sta);

> sta := time(): evalm(A2&*xs2):                          print(time()-sta);
  sta := time(): b2s:=evalm(A2&*xs2):                     print(time()-sta);
  sta := time(): MatrixVectorMultiply(A2m,xs2m):          print(time()-sta);
  sta := time(): b2sm:=MatrixVectorMultiply(A2m,xs2m): print(time()-sta);

  # Prozedur  Amx()
> sta := time(): Amx(n,n1,trid,A2,xs2):          print(time()-sta);
  sta := time(): b2s := Amx(n,n1,trid,A2,xs2):   print(time()-sta);

> sta := time(): inverse(A2):                    print(time()-sta);
  sta := time(): A2inv := inverse(A2):           print(time()-sta);
  sta := time(): LinearSolve(A2m,In):            print(time()-sta);
  sta := time(): A2minv := LinearSolve(A2m,In): print(time()-sta);
```

Zeiten in Sekunden fuer A2(n,n), n=n1^2, blocktridiagonal mit Shift c=0
(MVMultiply = MatrixVectorMultiply)

	n=100 Digits			n=225 Digits			n=625 Digits			n=900 Digits
	30	40	50	30	40	50	30	40	50	50
Generierung A2	0.046	0.047	0.046	0.343	0.358	0.359	4.422	4.608	4.516	13
linsolve(A2,b2)	0.390	0.469	0.484	2.767	3.031	3.250	50.812	53.658	55.062	201
LinearSolve(A2m,b2m)	0.078	0.234	0.266	0.704	1.578	1.875	12.313	23.657	26.609	91
evalm(A2&*xs2)	0.062	0.062	0.063	0.594	0.594	0.620	9.938	10.875	9.812	66
MVMultiply(A2m,xs2m)	0	0	0	0.015	0.016	0.015	0.093	0.093	0.094	0.203
Prozedur Amx()	0	0	0	0	0	0	0.016	0.015	0.031	0.062
inverse(A2)	7.781	10.375	10.829	121	137	147	5446	6380	6780	52628
LinearSolve(A2m,In)	1.968	3.734	4.469	20	39	46	1267	2168	1624	7152

Für die Implementierung der AV ist sehr wichtig der Einsatz der zeitsparenden Prozedur `Amx()` für das Matrix-Vektor-Produkt Ax im Vergleich zu den Maple-Kommandos `evalm(A&*x)` bzw. `MatrixVektorMultiply(A,x)`.

Die erhaltenen Rechenzeiten für die verschiedenen Prozeduren unterscheiden sich zum Teil erheblich. Da die Algorithmen im Prinzip "black" Boxen sind, ist es unmöglich etwas über ihre Vorgehensweise und den Aufwand an arithmetischen Operationen zu erfahren.

Aus obigen und weiteren Rechnungen können wir die folgenden Zeiten entnehmen.

Zeiten in Sekunden fuer Ax, A1(n,n) tridiagonal mit Shift c=0
Digits=50

	n				
	625	900	1255	1600	2025
MatrixVectorMultiply(A1m,xs1m)	0.078	0.172	0.328	0.563	0.875
Prozedur Amx(n,n1,trid,A1,xs1)	0.016	0.031	0.046	0.047	0.078

Zur Nutzerprozedur `Amx()` lassen sich damit Aufwandsabschätzungen bezüglich der Größenordnung machen.
Legen wir bei der n-dimensionalen Tridiagonalmatrix pro Zeile 5 wesentliche Operationen zu Grunde ($3 \times \{*\}$ und $2 \times \{+\}$), so beträgt die Komplexität an Operationen $\mathcal{K}(n) = 5n + \mathcal{O}(1)$. Dabei vernachlässigen wir die Berechnungen der Indizes, die Steuerung der Schleifen und Ergibtanweisungen.
Zudem bemerkt man, dass die Rechenzeiten in Maple natürlich nicht auf die Genauigkeit einer Tausendstel Sekunde angegeben werden, sondern dort Zeitsprünge zwischen 0.015 und 0.017 Sekunden erkennbar sind.
Aus der Komplexitätsfunktion und der Rechenzeit ermitteln wir die ungefähre Anzahl der arithmetischen Operationen pro Sekunde *flops* für den PC Pentium IV (2.4GHz).

```
Anzahl der arithmetischen Operationen pro Sekunde (gerundet) bei Ax
mit Prozedur  Amx(n,n1,trid,A1,xs1), A1(n,n) tridiagonal
Digits=50
            n       625     900    1255    1600    2025
          ---------------------------------------------
          flops   195000  145000  135000  170000  130000
```

Damit nähern wir uns den Bereich 60 000..120 000 $flops$ pro Sekunde an, der im Zusammenhang mit Maple 9.5 bei der Genauigkeit `Digits:=16` schon genannt wurde.

6.3.5 Komplexität bei LGS und Invertierung von Matrizen

Da in den bisherigen Rechnungen auch die Lösung von LGS sowie die Invertierung von Matrizen einbezogen worden ist, sollen dazu einige Aufwandsbetrachtungen gemacht werden. Sie sind die Grundlage für die Rechenzeiten mit entsprechenden Befehlen im CAS Matlab und Prozeduren in höheren Programmiersprachen, gestatten jedoch nicht den direkten Bezug auf die Rechenzeiten der Maple-Kommandos.

Zentraler Punkt für die beiden genannten Problemstellungen ist der bekannte Gauß-Algorithmus (Gauß-Elimination, GA) für das LGS $Ax = b$ (vergl. [17]). Er basiert auf der LU-Faktorisierung der Matrix A mit der unteren und oberen Dreiecksmatrix L bzw. U gemäß $A = LU$. In dieser Grundvariante entspricht er vom Formelapparat und Rechenaufwand her auch der Darstellung als Resttableau-Algorithmus oder verketteter Gauß-Algorithmus (VGA).
Steht nun die LU-Faktorisierung der Matrix A zur Verfügung, so kann man das LGS $Ax = LUx = b$ durch die weniger aufwändigen gestaffelten LGS $Ly = b$ und $Ux = y$ der Reihe nach lösen.
Bezieht man die rechte Seite b beim GA gleich mit ein, so entsteht aus der rechten Seite der Vektor y und es bleibt nur die Rückwärtssubstitution gemäß $Ux = y$ übrig. Der Gesamtaufwand bleibt in jedem Fall der gleiche.
Ein vorliegende LU-Faktorisierung der Matrix A kann unter Verwendung der n Einheitsvektoren als rechte Seiten des LGS natürlich auch für die Ermittlung der inversen Matrix dienen.

Effizienz des Gauß-Algorithmus

Der arithmetische Hauptaufwand liegt in der LU-Faktorisierung der Matrix bzw. Vorwärtselimination des LGS mit rechter Seite.
Wir betrachten jeden der drei Schritte, also LU-Faktorisierung $A = LU,\ l_{ii} = 1$, in der einfachen Variante als Resttableau-Algorithmus

$$\begin{array}{rcll} k & = & 1,2,...,n & \\ p & = & a_{kk} & \\ i & = & k+1,k+2,...,n & \\ & & s \;=\; a_{ik}/p & \\ & & a_{ik} \;=\; s & \rightarrow\ L \\ & & a_{ij} \;=\; a_{ij} - s\,a_{kj}, \quad j = k+1,k+2,...,n & \rightarrow\ U \end{array} \tag{6.97}$$

die Vorwärtselimination $Ly = b,\ l_{ii} = 1$, gemäß

$$y_k \;=\; b_k - \sum_{i=1}^{k-1} l_{ki}\, y_i, \quad k = 1,2,...,n, \tag{6.98}$$

sowie Rückwärtssubstitution $Ux = y$ gemäß

$$x_k \;=\; \frac{1}{u_{kk}}\Big(y_k - \sum_{i=k+1}^{n} u_{ki}\, x_i\Big), \quad k = n, n-1, ..., 1, \tag{6.99}$$

für sich und fassen schließlich die Strichoperationen $\{+, -\}$ bzw. Punktoperationen $\{*,\ /\} = \{\cdot,\ :\}$ jeweils zusammen.

In der Faktorisierung (6.97) verursacht die Anweisung $a_{ij} = a_{ij} - sa_{kj}$ im Inneren aller Schleifen mit einer Addition und einer Multiplikation den Hauptaufwand.
Wir haben mit dieser Vorschrift $n-1$ Resttableaus der Reihe nach zu erzeugen, das sind also $(n-1)^2+(n-2)^2+...+1^2$ Elemente mit je 1 Addition und 1 Multiplikation. Dazu kommen noch $(n-1)+(n-2)+...+1$ Divisionen für die Spalten. Damit ergibt sich der Gesamtaufwand beim Durchlaufen der k-Schleife

$$\begin{aligned} T_{LU}(n) \;&=\; \sum_{k=1}^{n-1}(n-k)^2\ \{+,*\} \;+\sum_{k=1}^{n-1}(n-k)\ \{/\} \;=\; \sum_{k=1}^{n-1}k^2\ \{+,*\} \;+\sum_{k=1}^{n-1}k\ \{/\} \\ &=\; \frac{(n-1)n(2n-1)}{6}\ \{+,*\} \;+\frac{(n-1)n}{2}\ \{/\} \\ &=\; \frac{n^3}{3}-\frac{n^2}{2}+\frac{n}{6}\ \{+\} \;+\frac{n^3}{3}-\frac{n}{3}\ \{*\} \\ &=\; \frac{2n^3}{3}-\frac{n^2}{2}+\frac{n}{6}, \end{aligned}$$

$Ly = b$:

$$\begin{aligned} T_V(n) &= \sum_{k=1}^{n-1} k \ \{+,*\} \ = \ \frac{(n-1)n}{2} \ \{+,*\} \\ &= \frac{n^2}{2} - \frac{n}{2} \ \{+\} \ + \frac{n^2}{2} - \frac{n}{2} \ \{*\} \\ &= n^2 - n, \end{aligned}$$

$Ux = y$:

$$\begin{aligned} T_R(n) &= \sum_{k=1}^{n-1} k \ \{+,*\} \ + n \ \{/\} \ = \ \frac{(n-1)n}{2} \ \{+\} \ + \frac{(n+1)n}{2} \ \{*\} \\ &= \frac{n^2}{2} - \frac{n}{2} \ \{+\} \ + \frac{n^2}{2} + \frac{n}{2} \ \{*\} \\ &= n^2. \end{aligned}$$

Damit beträgt der Aufwand zur Lösung des LGS bei gegebener Faktorisierung LU $T_V(n) + T_R(n) = 2n^2 - n$ sowie der arithmetische Gesamtaufwand

$$T(n) = T_{LU}(n) + T_V(n) + T_R(n) = \frac{2n^3}{3} + \frac{3n^2}{2} + \frac{n}{6}. \tag{6.100}$$

Hat man m rechte Seiten im LGS zu verarbeiten, so ist die Komplexität

$$\begin{aligned} T_m(n) &= T_{LU}(n) + m[T_V(n) + T_R(n)] \ = \ \Big(\frac{2n}{3} + 2m\Big)n^2 - \frac{n^2}{2} + \frac{n}{6} - mn \\ &\approx \frac{2n^3}{3} + 2mn^2, \quad m \gg 1. \end{aligned}$$

Zur Berechnung der inversen Matrix A^{-1} muss das LGS n Mal gelöst werden, und zwar mit den n rechten Seiten als Einheitsvektoren.
Dazu kommen $2n^2$ *flops* für das Produkt $A^{-1}b$. Damit ist grob gerechnet

$$\begin{aligned} T_i(n) &= n[T_V(n) + T_R(n)] = 2n^3 - n^2, \\ T_{inv}(n) &= T_{LU}(n) + T_i(n) + 2n^2 = \frac{8n^3}{3} + \frac{n^2}{2} + \frac{n}{6}. \end{aligned} \tag{6.101}$$

Beachtet man jedoch, dass das gestaffelte System $Ly = b$ mit Einheitsvektoren auf der rechten Seite wegen der Nullen nur ca. die Hälfte von $T_V(n)$ braucht, dann ergibt sich eine genauere Abschätzung gemäß

$$\widetilde{T}_{inv}(n) = T_{LU}(n) + n\Big[\frac{1}{2}T_V(n) + T_R(n)\Big] + 2n^2 = \frac{13n^3}{6} + n^2 + \frac{n}{6}. \tag{6.102}$$

Wir machen nun die Aufwandsuntersuchung für die Vorwärtselimination im LGS $Ly = b$ mit $b = e_1, e_2, ..., e_n$ detaillierter. Zu lösen ist das System

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{pmatrix} Y = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Für die 1. Lösung $y = (y_1, y_2, ..., y_n)^T$ als 1. Spaltenvektor von Y braucht man folgende Multiplikationen und Additionen:

y_1 : 0,
y_2 : 1 $\{*\}$,
y_3 : 2 $\{*\}$, 1 $\{+\}$,
...
y_{n-1} : $n-2$ $\{*\}$, $n-3$ $\{+\}$,
y_n : $n-1$ $\{*\}$, $n-2$ $\{+\}$,

also insgesamt $(n-1)n/2$ $\{*\}$ und $(n-2)(n-1)/2$ $\{+\}$.
Für die weiteren Lösungsvektoren reduziert sich der Aufwand jeweils um die "untere Zeile". Damit gilt

$$\begin{aligned} \overline{T}_V(n) &= \sum_{k=2}^{n} \frac{(k-1)k}{2} \{*\} + \sum_{k=2}^{n-1} \frac{(k-1)k}{2} \{+\} \\ &= 2\sum_{k=2}^{n} \frac{(k-1)k}{2} - \sum_{k=1}^{n-1} k = \sum_{k=2}^{n} (k-1)k - \frac{(n-1)n}{2} \\ &= \sum_{k=2}^{n} k^2 - \sum_{k=2}^{n} k - \frac{(n-1)n}{2} \\ &= \frac{1}{6}n(n+1)(2n+1) - 1 - \Big(\frac{(n+1)n}{2} - 1\Big) - \frac{(n-1)n}{2} \\ &= \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - n^2 \\ &= \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6} \end{aligned}$$

und

$$\overline{T}_{inv}(n) = T_{LU}(n) + \overline{T}_V(n) + nT_R(n) + 2n^2 = 2n^3 + n^2 + \frac{n}{3}. \tag{6.103}$$

Etwas einfacher stellen sich die Komplexitätsfunktionen dar, wenn man 1 Addition und 1 Multiplikation zu einer Grundoperation $\{\circ\}$ zusammenfasst und in den Betrachtungen nur die führende Ordnung nimmt.

Verfahren	$T_o(n)$ bez. $\{\circ\}$
$A = LU$	$n^3/3$
$Ly = b, Ux = y$	n^2
A^{-1}	n^3
$A = LU$ & $Ax = b$	$n^3/3$
$A = LU$ & A^{-1}	$4n^3/3$

Tab. 6.8
Führende Ordnung der Komplexität von Verfahren im Operationsmix $\{+, *\}$

In älterer Literatur wird der wesentliche Aufwand meist nach der Anzahl der Multiplikationen/Divisionen gerechnet. Damit ergeben sich

$$T^*_{LU}(n) = \frac{n^3}{3} - \frac{n}{3}, \quad T^*_V(n) = \frac{n^2}{2} - \frac{n}{2}, \quad T^*_R(n) = \frac{n^2}{2} + \frac{n}{2},$$

$$T^*_{LU}(n) + T^*_V(n) = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} = \frac{1}{6}n(n-1)(2n+5), \tag{6.104}$$

$$T^*(n) = T^*_{LU}(n) + T^*_V(n) + T^*_R(n) = \frac{n^3}{3} + n^2 - \frac{n}{3}. \tag{6.105}$$

Bei Berücksichtigung der Additionen/Subtraktionen hat man ungefähr eine Verdopplung des Aufwands, was im Vergleich mit (6.100) nur größenordnungsmäßig richtig ist.

(1) Komplexität von Algorithmen zur Lösung von LGS

Wir vergleichen hier die Operationszahlen und Rechenzeiten von 5 Lösungsalgorithmen.
Die Implementation ist in einer älteren Version von Matlab gemacht worden, wo man die Anzahl der *flops* noch ermitteln konnte.

$Ax = b$, AB ist die erweiterte Koeffizientenmatrix $(A|b)$,

`lu` : `[L,U] = lu(A);` Lösung x als `U\(L\b)`
`gaussel` : `AT = gaussel(AB);` x mit Rückwärtseinsetzen
`A\b` : `A\b`
`rref` : `rref(AB)`
`inv` : `inv(A)*b`

Wir notieren die Matlab-Befehle. Die Generierung der voll besetzten Matrix und rechten Seite des LGS ist dort zu erkennen.
Die verschiedenen Ergebnisse werden so abgespeichert, um sie u. a. als vergleichendes Balkendiagramm (Grafik) darzustellen.

```
title('SPEEDTEST for SOLUTION of Ax=b for n = 20x20 .. 400x400 ')
clear n t s sn ns tn nt q tt ss echo off

h = 1;     % Skalierung des Balkenabstandes
na = 20;
nsw = 20;
ne = 200;

for n = na:nsw:ne
  A = rand(n)+n*eye(n);  % Matrix zufaellig und streng diagonaldominant
  b = rand(n,1);
  AB = [A b];
  %
  t0 = clock;  flops(0);
  [L U] = lu(A);
  U\(L\b);
  t(n) = etime(clock,t0);  s(n) = flops;  q(n) = 0;
  if (t(n)~=0), q(n) = s(n)./t(n); end;
  %
  t0 = clock;  flops(0);
  AT = gaussel(AB);
  for i = n:-1:1
    hh = AT(i,n+1);
    for j = i+1:n
      hh = hh - x(j)*AT(i,j);
    end;
    x(i) = hh/AT(i,i);
  end;
  t(n+2*h) = etime(clock,t0);  s(n+2*h) = flops;  q(n+2*h) = 0;
  if (t(n+2*h)~=0), q(n+2*h) = s(n+2*h)./t(n+2*h); end;
  %
  t0 = clock;  flops(0);
  A\b;
  t(n+4*h) = etime(clock,t0);  s(n+4*h) = flops;  q(n+4*h) = 0;
  if (t(n+4*h)~=0), q(n+4*h) = s(n+4*h)./t(n+4*h); end;
  %
  t0 = clock;  flops(0);
  rref(AB);
  t(n+6*h) = etime(clock,t0);  s(n+6*h) = flops;  q(n+6*h) = 0;
  if (t(n+6*h)~=0), q(n+6*h) = s(n+6*h)./t(n+6*h); end;
  %
  t0 = clock;  flops(0);
  inv(A)*b;
  t(n+8*h) = etime(clock,t0);  s(n+8*h) = flops;  q(n+8*h) = 0;
  if (t(n+8*h)~=0), q(n+8*h) = s(n+8*h)./t(n+8*h); end;
  %
```

```
% Grafik, Balkendiagramme
[ns,sn] = bar(s);
plot(ns,sn,'g')
xlabel('dimension n')
ylabel('flops')
print graph31.ps -dps
pause

[nt,tn] = bar(t);
plot(nt,tn)
xlabel('dimension n')
ylabel('elapsed time in sec')
print graph32.ps -dps
pause

bar(q)
xlabel('dimension n')
ylabel('quotient of flops / time(sec) ')
print graph33.ps -dps
pause
clf

% Tabellen fuer flops und Rechenzeiten
disp(' ')
disp('flops')
disp('         n        lu    gaussel       A\b      rref       inv')
ss = zeros(round((ne-na)/nsw)+1,6);
for j=na:nsw:ne
  ss(round((j-na)/nsw)+1,1) = j;
  for k=0:2:8
    ss(round((j-na)/nsw)+1,round(k/2)+2) = s(j+k*h);
  end
end;
disp(ss)

disp(' ')
disp('times')
disp('         n        lu   gaussel       A\b       rref       inv')
tt = zeros(round((ne-na)/nsw)+1,6);
for j=na:nsw:ne
  tt(round((j-na)/nsw)+1,1) = j;
  for k=0:2:8
    tt(round((j-na)/nsw)+1,round(k/2)+2) = t(j+k*h);
  end
end;
disp(tt)
```

Die Rechnungen wurden auf einem PC Pentium II (350MHz) durchgeführt.

n	`lu`	`gaussel`	`A\b`	`rref`	`inv`
20	6028	6221	8028	27435	18518
40	45238	46021	53482	132595	138062
60	149648	151421	168380	357758	454650
80	351258	354421	384674	748467	1064234
100	682068	687021	734390	1349225	2062840
120	1174078	1181221	1249514	2214142	3546454
140	1859288	1869021	1962066	3378712	5611096
160	2769698	2782421	2903782	4896733	8352502
180	3937308	3953421	4107164	6812084	11867174
200	5394118	5414021	5604018	9170035	16250918
250	10511393	10542521	10840204	17335976	31642579
300	18136168	18181021	18609462	29346143	54564812
350	28768443	28829521	29412540	45912719	86518365
400	42908218	42988021	43750234	67730467	129004034
$\mathcal{K}$	$\frac{2}{3}n^3$			n^3	$2n^3$

Tab. 6.9 Ergebnistableau für PC Pentium II: *flops*

n	`lu`	`gaussel`	`A\b`	`rref`	`inv`
20	0	0.05	0	0.39	0
40	0	0.49	0	1.43	0
60	0	1.54	0	3.02	0
80	0	3.51	0	5.28	0
100	0	6.70	0	8.02	0.05
120	0	11.53	0	11.48	0.11
140	0.06	17.96	0	15.43	0.22
160	0.06	26.69	0.11	20.32	0.22
180	0.11	37.79	0.11	25.76	0.33
200	0.11	56.35	0.11	34.49	0.44
250	0.21	99.26	0.32	50.92	0.49
300	0.38	169.34	0.66	74.70	1.26
350	0.71	271.50	0.66	104.63	1.49
400	1.10	400.79	1.04	140.56	3.89
Vergl.	$\frac{1}{3}T$	$100T$	$\frac{1}{3}T$	$35T$	T

Tab. 6.10 Ergebnistableau für PC Pentium II: *time* in *sec*

Bei der Anzahl der GP-Operationen werden mit `flops` alle arithmetischen Operationen extra gezählt. Die sonst übliche Angabe des Aufwandes für den GA mit $T^*(n) = \frac{n^3}{3} + n^2 - \frac{n}{3}$ im Operationsmix $\{+, *\}$ muss also verdoppelt werden. Trotzdem treten noch Ungenauigkeiten auf.

Sinnvoll ist es also, die Komplexitätsfunktion (6.100) $T(n) = \frac{2n^3}{3} + \frac{3n^2}{2} + \frac{n}{6}$ zu verwenden. Somit ergibt sich die Größenordnung $\mathcal{K} = \frac{2}{3}n^3 + \mathcal{O}(n^2)$ für die Verfahren `lu, gaussel` und `A\b`.

Der Zugang über die inverse Matrix hat auf der Basis von (6.105) $T^*(n) = \frac{n^3}{3} + n^2 - \frac{n}{3}$ bei n rechten Seiten die Komplexität $\hat{T}_{inv}(n) = 2(\frac{n^3}{3} + n \cdot n^2 - \frac{n}{3}) = \frac{8n^3}{3} - \frac{2n}{3}$.

Dieselbe Größenordnung besitzt auch (6.101) $T_{inv}(n) = \frac{8n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$, während die genauere Betrachtung mit den Einheitsvektoren die Beziehungen (6.102) $\widetilde{T}_{inv}(n) = \frac{13n^3}{6} + n^2 + \frac{n}{6}$ bzw. (6.103) $\overline{T}_{inv}(n) = 2n^3 + n^2 + \frac{n}{3}$ ergab, was auf die Größenordnung $\mathcal{K} = 2n^3 + \mathcal{O}(n^2)$ führt. Gleichzeitig bedeutet der Vergleich der führenden Koeffizienten von $T(n)$ und $\overline{T}_{inv}(n)$, dass die explizite Verwendung der inversen Matrix zur Lösung des LGS $2 : \frac{2}{3} = 3$ Mal mehr arithmetischen Aufwand erfordert. Die Verhältnisse sind bei kleinen Dimensionen n noch nicht so ausgeprägt, werden aber mit wachsendem n immer deutlicher.

Deshalb noch ein Vergleich von Ergebnissen aus den hergeleiteten Formeln mit den ermittelten *flops* am PC Pentium II.

Verf.	n	Formel	Wert	*flops*
`lu`	20	$T(n) = \frac{2n^3}{3} + \frac{3n^2}{2} + \frac{n}{6}$	5 937	6 028
		$2T^*(n) = \frac{2n^3}{3} + 2n^2 - \frac{2n}{3}$	6 120	
	200	$T(n)$	5 393 367	5 394 118
		$2T^*(n)$	5 413 200	
`inv`	20	$\overline{T}_{inv}(n) = 2n^3 + n^2 + \frac{n}{3}$	16 407	18 518
		$\widetilde{T}_{inv}(n) = \frac{13n^3}{6} + n^2 + \frac{n}{6}$	17 737	
		$T_{inv}(n) = \frac{8n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$	21 537	
		$\hat{T}_{inv}(n) = \frac{8n^3}{3} - \frac{2n}{3}$	21 320	
	200	$\overline{T}_{inv}(n)$	16 040 067	16 250 918
		$\widetilde{T}_{inv}(n)$	17 373 367	
		$T_{inv}(n)$	21 353 367	
		$\hat{T}_{inv}(n)$	21 333 200	

Tab. 6.11 Aufwand von Verfahren gemäß Formel und am PC Pentium II

Die Kommandos `lu`, `gaussel` und `A\b` bestätigen die Formel $T(n)$ für die *flops*, wobei `A\b` für kleinere Dimensionen etwas mehr braucht.
Genauso passen die *flops* bei `inv` am ehesten zu $\overline{T}_{inv}(n)$.

Die Kommandos `lu` und `A\b` zeigen sowohl für $flops$ als auch in der Rechenzeit $time = T$ ähnliches Verhalten. Entsprechend seiner Komplexität ist `inv` dreimal schlechter. Dass `gaussel` trotz gleicher $flops$ wie `lu` in der Rechenzeit deutlich schlechter ausfällt, liegt nicht hauptsächlich am timesharing Modus des Rechners. Es ist zu vermuten, dass auch die Struktur der einfachen Laufanweisungen in der Funktion `gaussel` zu Rechenzeitverlusten führt gegenüber solchen, die eventuell in den anderen Kommandos Teilvektoren und Untermatrizen nutzt und eine schnelle Adressenmanipulation beim Zugriff auf Feldkomponenten realisiert.
Auch `rref` wird im Vergleich mit `gaussel` trotz größerer $flops$ bezüglich der Rechenzeit immer günstiger, kann aber bei weitem nicht mit `lu` konkurrieren.
Der GA ohne Pivotstrategie (Funktion `gausseo`) zeigt ähnliche Komplexität wie `gaussel`. Das unterstreicht, dass die Auswertung von Bedingungen/Tests größenordnungsmäßig die Komplexität nicht beeinflusst.

(2) Komplexität von Algorithmen zur Bestimmung der Inversen

Analog zum LGS vergleichen wir hier die Operationszahlen und Rechenzeiten von 4 Lösungsalgorithmen.

A^{-1}, AE ist die um die Einheitsmatrix I erweiterte Koeffizientenmatrix

```
inv  : inv(A)
A\I  : A\I
lu   : [L,U] = lu(A); U\(L\I)
rref : rref(AE)
```

Wir notieren ausgewählte Matlab-Befehle.

```
title('SPEEDTEST for INVERSION of MATRICES n = 10x10 .. 400x400 ')
clear n t s sn ns tn nt q tt ss echo off
h = 1;     % Skalierung des Balkenabstandes
na = 10; nsw = 10; ne = 100;

for n=na:ns:ne
  A = rand(n)+n*eye(n);  % Matrix zufaellig und streng diagonaldominant
  I = eye(n);
  AE = [A I];
  %
  t0 = clock; flops(0);
  inv(A);
  t(n) = etime(clock,t0);  s(n) = flops;  q(n) = 0;
  if (t(n)~=0), q(n) = s(n)./t(n); end;
  %
  t0 = clock; flops(0);
  A\I;
  t(n+2*h) = etime(clock,t0);  s(n+2*h) = flops;  q(n+2*h) = 0;
  if (t(n+2*h)~=0), q(n+2*h) = s(n+2*h)./t(n+2*h); end;
  t0 = clock; flops(0);
```

```
  [L U] = lu(A);  U\(L\I);
  t(n+4*h) = etime(clock,t0);  s(n+4*h) = flops;  q(n+4*h) = 0;
  if (t(n+4*h)~=0), q(n+4*h) = s(n+4*h)./t(n+4*h); end;
  %
  t0 = clock; flops(0);
  rref(AE);
  t(n+6*h) = etime(clock,t0);  s(n+6*h) = flops;  q(n+6*h) = 0;
  if (t(n+6*h)~=0), q(n+6*h) = s(n+6*h)./t(n+6*h); end;
end      % Fortsetzung analog zu LGS
```

n	inv	A\I	lu	rref
20	17708	22840	22750	47497
40	134814	176676	176280	277181
60	447356	589548	588610	826784
80	1051446	1389568	1387740	1840280
100	2042818	2704470	2701670	3457743
150	6846281	9085008	9078745	11084328
200	16171200	21484502	21473320	25605056
250	31517269	41902646	41885395	49264454
300	54384734	72339686	72314970	84330436
350	86273097	114795124	114762045	133003654
400	128683576	171270178	171226620	197482589
$\mathcal{K}$	$2n^3$	$\frac{8}{3}n^3$		$3n^3$

Tab. 6.12 Ergebnistableau für PC Pentium II: *flops*

n	inv	A\I	lu	rref
20	0	0	0	0.55
40	0	0	0	2.04
60	0	0	0	4.72
80	0.06	0	0.05	8.63
100	0.06	0.06	0.11	13.89
150	0.22	0.22	0.27	31.53
200	0.44	0.61	0.87	59.65
250	0.68	1.21	1.48	100.02
300	1.26	2.08	3.40	154.40
350	1.49	2.25	4.01	225.14
400	3.89	6.32	9.39	312.91
Vergl.	T	$T..\frac{3}{2}T$	$T..2T$	

Tab. 6.13 Ergebnistableau für PC Pentium II: *times* in *sec*

Wir hatten schon vorher die Komplexität für die Bestimmung der Lösung eines LGS mittels `inv(A)*b` untersucht. Die alleinige Invertierung der Matrix macht gemäß (6.103) den Aufwand $2n^3 - n^2 + \frac{n}{3}$ und unterscheidet sich in den $flops$ eben nur durch $2n^2$ von den für die Lösung `inv(A)*b`. So beträgt die Größenordnung der Komplexität für die inverse Matrix auch $\mathcal{K}(n) = 2n^3 + \mathcal{O}(n^2)$. Auch die Rechenzeiten T sind ungefähr die gleichen.
Die Kommandos `A\I` und `lu` zeigen sowohl für $flops$ als auch in der Rechenzeit ähnliches Verhalten, sind aber nicht so gut wie `inv`. Ihre Komplexität ist $\approx \frac{8}{3}n^3$. Der Befehl `rref` fällt im Vergleich als sichtbar schlechteste Variante auf.

Die Kommandos `A\I` und `lu` geraten aber bezüglich der Rechenzeit im Vergleich mit `inv` mit wachsendem n immer mehr ins Hintertreffen. Das Verhältnis anders als bei den $flops$ verschlechtert sich. Der Befehl `rref` braucht die längste Rechenzeit.

`A\I` braucht ca. 4 Mal mehr $flops$ und $time$ als `A\b`. Ähnlich ist es bei `lu`, wobei die Rechenzeitrelation von `lu`(A^{-1}) : `lu`$(Ax = b)$ noch zunehmend schlechter wird.
Wird die inverse Matrix explizit gebraucht, so ist die built-in-Funktion `inv` für die Berechnung am besten geeignet.

Im CAS Maple ist es in der Tat sehr schwierig solche Zusammenhänge wie in Matlab zu erkennen.
Wir bemerken zunächst, dass wir in Maple mit zwei Bandmatrizen gerechnet haben. Die Rechenzeiten für die Blocktridiagonalmatrix sind größer und teilweise wesentlich größer als die für die Tridiagonalmatrix. Diese nehmen meistens noch zu mit wachsender Genauigkeit, wobei diese Zunahmen bei den Kommandos `evalm(A&*x)`, `linsolve, inverse` aus dem Paket `linalg` weniger stark ausgeprägt sind als bei dem anderen Kommado `LinearSolve` aus dem Paket `LinearAlgebra`.
Sind wir in der Lage, nun einige Komplexitätsfunktionen, wie sie in Tabelle 6.11 zusammengefasst sind, für die Maple-Kommandos nachzuvollziehen?
Dabei wollen wir von der Rechnerleistung des PC Pentium IV mit 150 000 $flops$ ausgehen und auch voraussetzen, dass die Ordnung der Komplexität erhalten bleibt. Wir probieren es wegen der Bandstruktur der Matrix aber auch mit anderer Ordnung. Zu bestimmen ist also der Faktor bei dem führenden Glied $n^k,\ k = 2, 3$.

Wir betrachten die Tridiagonalmatrix (1D-Laplace-Matrix mit Shift) $A1$ aus Abschnitt 6.3.4.

```
> A1 := band([-1.0,2.0,-1.0],n):
  A1 := evalm(A1+c*diag(1.0$n)):
```

und die Rechenzeiten der Algorithmen bei `Digits:=50`.
Als Komplexitätsfunktionen versuchen wir die Ansätze

$$\mathcal{K}(n) \ = \ \alpha\, n^k, \quad k = 2, 3,$$

zu verifizieren.

Kommando	n				$\mathcal{K}(n)$ theor.
	100	225	625	900	
`evalm(A&*x)`	$1.6n^2$	$2n^2$	$4n^2$	$6n^2$	$2n^2$
`MatrixVectorMultiply`			$0.030n^2$	$0.031n^2$	$2n^2$
`linsolve`	$0.035n^3$	$0.020n^3$	$0.015n^3$	$0.016n^3$	$\frac{2}{3}n^3$
Test mit n^2	$3.5n^2$	$4.4n^2$	$9.0n^2$	$14.2n^2$	
`LinearSolve`, $Ax=b$		$0.00060n^3$	$0.00016n^3$	$0.00021n^3$	$\frac{2}{3}n^3$
Test mit n^2		$0.14n^2$	$0.10n^2$	$0.19n^2$	
`inverse`	$0.85n^3$	$1.12n^3$	$2.68n^3$	$4.71n^3$	$2n^3$
Test mit n^2	$85n^2$	$251n^2$	$1677n^2$	$4240n^2$	
`LinearSolve`, A^{-1}	$0.082n^3$	$0.056n^3$	$0.055n^3$	$0.069n^3$	$2n^3$
Test mit n^2	$8.2n^2$	$12.7n^2$	$34.5n^2$	$62.9n^2$	

Tab. 6.14 Umrechnung der Rechenzeiten von Maple-Algorithmen für $A1$ auf die Komplexitätsfunktion $\mathcal{K}(n) = \alpha n^2$ bzw. αn^3

Mit diesen Ergebnissen liegen wir weit weg von der theoretisch erhaltenen Komplexität. Die Deutung des Aufwands an *flops* sowie Angabe eine geeigneten Komplexitätsfunktion für die Maple-Kommandos sind nicht machbar, auch wenn sich in Einzelfällen wie bei `linsolve` ein Trend beim Aufwand von ungefähr $0.02n^3$ bzw. bei `MatrixVectorMultiply` die Trendfunktion $0.03n^2$ (auch bei den weiteren Dimensionen $n = 1225, 1600, 2025$) andeuten. Es scheint für die gegebenen Dimensionen n auch nicht auszureichen, im Ansatz nur das führende Glied n^k zu betrachten.
Ansätze mit anderen Exponenten kommen, wie man leicht aus der Tabelle 6.14 entnehmen kann, erst gar nicht in Frage.

Unter denselben Voraussetzungen betrachten wir auch die Blocktridiagonalmatrix $A2$ aus Abschnitt 6.3.4 und bestätigen damit die genannten Resultate.

Kommando	n				$\mathcal{K}(n)$ theor.
	100	225	625	900	
`evalm(A&*x)`	n^2	$2n^2$	$4n^2$	$12n^2$	$2n^2$
`MatrixVectorMultiply`		$0.044n^2$	$0.036n^2$	$0.037n^2$	$2n^2$
`linsolve`	$0.072n^3$	$0.042n^3$	$0.033n^3$	$0.041n^3$	$\frac{2}{3}n^3$
`LinearSolve`, $Ax=b$	$0.040n^3$	$0.024n^3$	$0.016n^3$	$0.018n^3$	$\frac{2}{3}n^3$
`inverse`	$1.6n^3$	$1.9n^3$	$4.2n^3$	$10.8n^3$	$2n^3$
`LinearSolve`, A^{-1}	$0.67n^3$	$0.60n^3$	$0.99n^3$	$1.47n^3$	$2n^3$

Tab. 6.15 Umrechnung der Rechenzeiten von Maple-Algorithmen für $A2$ auf die Komplexitätsfunktion $\mathcal{K}(n) = \alpha n^2$ bzw. αn^3

6.3.6 CG und CR für spezielle Matrizen

Eine Implementierung des CG soll sich auf die Versionen 1 (3.96) und 2 (3.97) im Teil I stützen. Alle auftretenden Matrix-Vektor-Produkte werden mit der zeitsparenden Prozedur `Amx()` aus Abschnitt 6.3.4 berechnet. Ergänzt um die zusätzliche Abfrage des Nenners bei der Berechnung der Schrittzahl α, die Bestimmung von Werten des Funktionals sowie einige Ergebnisfelder, Zwischenausgaben und Dateiarbeit entsteht dann die erweiterte Maple-Prozedur in Anlehnung an diese Versionen des CG, die wir in den Beispielrechnungen verwenden.

Dieser Algorithmus liefert in Maple die folgenden Kommandos.

```
> cg6:=proc(n::posint, n1::posint, trid::posint, A::matrix, b::vector,
            x0::vector, maxiter::posint, eeps::numeric,
            aus::name, fileaus::name)
  local  k,i,x,p,r,sr2,v,xh,alpha,alphaold,alphanew,beta,
         Q,fh,fh1,fh2,h,d,Ax;
  global Qv,epsv,epsv2,rv,Ainv,xs,lp,file1;

  # Matrix-Vektor-Produkt y=Ax
  Ax:=proc(n::posint, n1::posint, trid::posint, A::matrix, x::vector)
   local i,y;

   y:=vector(n,[seq(0,i=1..n)]):
   if trid=3 then   # Tridiagonalmatrix   A(n,n)
    y[1]:=A[1,1]*x[1]+A[1,2]*x[2];
    for i from 2 to n-1 do
     y[i]:=A[i,i-1]*x[i-1]+A[i,i]*x[i]+A[i,i+1]*x[i+1];
    end do;
    y[n]:=A[n,n-1]*x[n-1]+A[n,n]*x[n];
   elif trid=5 then  # Blocktridiagonalmatrix   A(n,n)=A(n1^2,n1^2)
    y[1]:=A[1,1]*x[1]+A[1,2]*x[2]+A[1,1+n1]*x[1+n1];
    for i from 2 to n1 do
     y[i]:=A[i,i-1]*x[i-1]+A[i,i]*x[i]+A[i,i+1]*x[i+1]+A[i,i+n1]*x[i+n1];
    end do;
    for i from n1+1 to n-n1 do
     y[i]:= A[i,i-n1]*x[i-n1]+A[i,i-1]*x[i-1]
           +A[i,i]*x[i]+A[i,i+1]*x[i+1]+A[i,i+n1]*x[i+n1];
    end do;
    for i from n-n1+1 to n-1 do
     y[i]:=A[i,i-n1]*x[i-n1]+A[i,i-1]*x[i-1]+A[i,i]*x[i]+A[i,i+1]*x[i+1];
    end do;
    y[n]:=A[n,n-n1]*x[n-n1]+A[n,n-1]*x[n-1]+A[n,n]*x[n];
     else
   end if;
   evalm(y);
  end:

  fh2:=`%.16e`; fh1:=`%+.10e`;     # Ausgabeformate einstellen
  fh :=fh1;
  for i from 2 to n do
   fh:=cat(fh,` `,fh1);
  end do;
```

```
k:=0:
x:=evalf(evalm(x0)):
v:=Ax(n,n1,trid,A,x):   # v:=evalm(A&*x):
Q:=0.5*evalm(transpose(x)&*v)-evalm(transpose(x)&*b):
r:=evalm(b-v):
p:=evalm(r):  lp:=evalm(p):
alphaold:=evalm(transpose(r)&*r);

Qv[1]:=Q;
h:=evalm(transpose(xs)&*b);
epsv[1]:=abs(sqrt(2.0*Q+h));
epsv2[1]:=sqrt(evalm(transpose(evalm(xs-x))&*evalm(xs-x)));
sr2:=evalf(sqrt(alphaold));
rv[1]:=sr2;
fprintf(default,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sr2);
if fileaus=ja then
 fprintf(file1,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sr2);
end if;

if aus=ja then
 fprintf(default,`\n`):
 fprintf(default,`Schritt k = %g\n`,k);
 fprintf(default,`Startvektor            x =
                 [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(default,`Funktionswert       Q(x) = `||fh2||`\n`,Q);
 fprintf(default,`Residuum/Suchr.  r =b-Ax =
                 [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(default,`Anfangsfehlerquadrat r'r = `||fh2||`\n\n`,alphaold);
end if;

if fileaus=ja then
 fprintf(file1,`\n`):
 fprintf(file1,`Schritt k = %g\n`,k);
 fprintf(file1,`Startvektor            x =
               [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(file1,`Funktionswert       Q(x) = `||fh2||`\n`,Q);
 fprintf(file1,`Residuum/Suchr.  r =b-Ax =
               [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(file1,`Anfangsfehlerquadrat r'r = `||fh2||`\n\n`,alphaold);
end if;

while (sqrt(1.0*alphaold)>eeps) and (k<maxiter) do
       # absoluter Fehler mit ||r||
 v:=Ax(n,n1,trid,A,p);     # v:=evalm(A&*p);
 d:=evalm(transpose(v)&*p);
 if d=0 then
  lprint(`Abbruch wegen Nenner p'Ap=0`):
  RETURN(x,k);
 end if;

 alpha:=alphaold/d;
 x:=evalm(x+alpha*p);
 r:=evalm(r-alpha*v);
 alphanew:=evalm(transpose(r)&*r);
```

```
 beta:=alphanew/alphaold;
 p:=evalm(r+beta*p);  lp:=evalm(p);
 k:=k+1;
 alphaold:=alphanew;

 v:=Ax(n,n1,trid,A,x):     # v:=evalm(A&*x);
 Q:=0.5*evalm(transpose(x)&*v)-evalm(transpose(x)&*b);

 if aus=ja then
  fprintf(default,`\n`);
  fprintf(default,`Schritt k = %g\n`,k);
  fprintf(default,`Iterationsvektor      x =
                  [`||fh||`]\n`,seq(x[i],i=1..n));
  fprintf(default,`Funktionswert      Q(x) = `||fh2||`\n`,Q);
  fprintf(default,`Suchrichtung          p =
                  [`||fh||`]\n`,seq(p[i],i=1..n));
  fprintf(default,`Residuum       r = b-Ax =
                  [`||fh||`]\n`,seq(r[i],i=1..n));
  fprintf(default,`Fehlernormquadrat   r'r = `||fh2||`\n`,alphaold);
 end if;

 if fileaus=ja then
  fprintf(file1,`\n`);
  fprintf(file1,`Schritt k = %g\n`,k);
  fprintf(file1,`Iterationsvektor      x =
                [`||fh||`]\n`,seq(x[i],i=1..n));
  fprintf(file1,`Funktionswert      Q(x) = `||fh2||`\n`,Q);
  fprintf(file1,`Suchrichtung          p =
                [`||fh||`]\n`,seq(p[i],i=1..n));
  fprintf(file1,`Residuum       r = b-Ax =
                [`||fh||`]\n`,seq(r[i],i=1..n));
  fprintf(file1,`Fehlernormquadrat   r'r = `||fh2||`\n`,alphaold);
 end if;

 fprintf(default,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sqrt(alphaold));
 if fileaus=ja then
  fprintf(file1,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sqrt(alphaold));
 end if;

 Qv[k+1]:=Q;
 xh:=evalm(xs-x);
 v:=Ax(n,n1,trid,A,xh);
 epsv[k+1]:=abs(sqrt(evalm(transpose(xh)&*v)));
            # epsv[k+1]:=abs(sqrt(2.0*Q+h));
 epsv2[k+1]:=sqrt(evalm(transpose(xh)&*xh));
 rv[k+1]:=evalf(sqrt(alphaold));
end do:

lprint(`||xs-x||_2    = `,evalf(norm(xs-x,frobenius))):
lprint(`||A^(-1)r||_2 = `,evalf(norm(Ainv&*r,frobenius))):
lprint(` `):

[x,k];
end:
```

Die Implementierung des CR erfolgt in Maple gemäß Teil II, Abschnitt 4.3, mit der Prozedur `Amx()` für die Matrix-Vektor-Produkte wie in CG.
Ergänzt um die zusätzliche Abfrage des Nenners bei der Berechnung des Parameters β, die Bestimmung von Werten der Funktionale sowie einige Ergebnisfelder, Zwischenausgaben und Dateiarbeit entsteht dann die erweiterte Prozedur.

```
> cr6:=proc(n::posint, n1::posint, trid::posint, A::matrix, b::vector,
            x0::vector, maxiter::posint, eeps::numeric,
            aus::name, fileaus::name)
  local  k,i,x,xh,p,r,r2,sr2,v,t,s,alpha,alphar,alphas,beta,betar,
         Q,R,fh,fh1,fh2,h,Ax;
> global Qv,Rv,epsv,epsv2,rv,Ainv,xs,lp,file1;

  # Matrix-Vektor-Produkt y=Ax
  Ax:=proc(n::posint, n1::posint, trid::posint, A::matrix, x::vector)
   local i,y;
  ...
  fh2:=‘%.16e‘; fh1:=‘%+.10e‘;    # Ausgabeformate einstellen
  fh :=fh1;
  for i from 2 to n do
   fh:=cat(fh,‘ ‘,fh1);
  end do;

  k:=0:
  x:=evalm(x0):
  v:=Ax(n,n1,trid,A,x):   # v:=evalm(A&*x):
  Q:=0.5*evalm(transpose(x)&*v)-evalm(transpose(x)&*b):
  R:=0.5*evalm(transpose(v)&*v)-evalm(transpose(v)&*b):
  r:=evalm(b-v):
  t:=Ax(n,n1,trid,A,r):   # t:=evalm(A&*r):
  alphar:=evalm(transpose(t)&*r):
  p:=evalm(r): lp:=evalm(p):
  s:=evalm(t):
  alphas:=evalm(transpose(s)&*s);

  Qv[1]:=Q;
  Rv[1]:=R;
  h:=evalm(transpose(xs)&*b);
  epsv[1]:=abs(sqrt(2.0*Q+h));
  epsv2[1]:=sqrt(evalm(transpose(evalm(xs-x))&*evalm(xs-x)));
  r2:=evalm(transpose(r)&*r);
  sr2:=evalf(sqrt(r2));
  rv[1]:=sr2;
  fprintf(default,‘k = %3d,  ||r||_2 = ‘||fh2||‘\n‘,k,sr2);
  if fileaus=ja then
   fprintf(file1,‘k = %3d,  ||r||_2 = ‘||fh2||‘\n‘,k,sr2);
  end if;

  if aus=ja then
   fprintf(default,‘\n‘):
   fprintf(default,‘Schritt k = %g\n‘,k);
   fprintf(default,‘Startvektor            x =
                  [‘||fh||‘]\n‘,seq(x[i],i=1..n));
```

```
 fprintf(default,‘Funktionswert       Q(x) = ‘||fh2||‘\n‘,Q);
 fprintf(default,‘Funktionswert       R(x) = ‘||fh2||‘\n‘,R);
 fprintf(default,‘Residuum/Suchr.  r =b-Ax =
                 [‘||fh||‘]\n‘,seq(r[i],i=1..n));
 fprintf(default,‘Anfangsfehlerquadrat r’r = ‘||fh2||‘\n\n‘,r2);
end if;

if fileaus=ja then
 fprintf(file1,‘\n‘):
 fprintf(file1,‘Schritt k = %g\n‘,k);
 fprintf(file1,‘Startvektor            x =
               [‘||fh||‘]\n‘,seq(x[i],i=1..n));
 fprintf(file1,‘Funktionswert       Q(x) = ‘||fh2||‘\n‘,Q);
 fprintf(file1,‘Funktionswert       R(x) = ‘||fh2||‘\n‘,R);
 fprintf(file1,‘Residuum/Suchr.  r =b-Ax =
               [‘||fh||‘]\n‘,seq(r[i],i=1..n));
 fprintf(file1,‘Anfangsfehlerquadrat r’r = ‘||fh2||‘\n\n‘,r2);
end if;

while (sr2>eeps) and (k<maxiter) do
 if alphar=0 then
  lprint(‘Abbruch wegen Nenner r’Ar=0‘):
  RETURN(x,k);
 end if;

 alpha:=alphar/alphas;
 x:=evalm(x+alpha*p);
 r:=evalm(r-alpha*s);
 r2:=evalm(transpose(r)&*r);
 sr2:=evalf(sqrt(r2));
 betar:=1/alphar;

 t:=Ax(n,n1,trid,A,r):   # t:=evalm(A&*r):
 alphar:=evalm(transpose(t)&*r);
 beta:=alphar*betar;
 p:=evalm(r+beta*p);  lp:=evalm(p);
 s:=evalm(t+beta*s);
 k:=k+1;
 alphas:=evalm(transpose(s)&*s);

 v:=Ax(n,n1,trid,A,x):    # v:=evalm(A&*x):
 Q:=0.5*evalm(transpose(x)&*v)-evalm(transpose(x)&*b):
 R:=0.5*evalm(transpose(v)&*v)-evalm(transpose(v)&*b):

 if aus=ja then
  fprintf(default,‘\n‘);
  fprintf(default,‘Schritt k = %g\n‘,k);
  fprintf(default,‘Iterationsvektor      x =
                  [‘||fh||‘]\n‘,seq(x[i],i=1..n));
  fprintf(default,‘Funktionswert      Q(x) = ‘||fh2||‘\n‘,Q);
  fprintf(default,‘Funktionswert      R(x) = ‘||fh2||‘\n‘,R);
  fprintf(default,‘Suchrichtung          p =
                  [‘||fh||‘]\n‘,seq(p[i],i=1..n));
  fprintf(default,‘Residuum       r = b-Ax =
```

```
                  [‘||fh||‘]\n‘,seq(r[i],i=1..n));
 fprintf(default,‘Fehlernormquadrat   r’r = ‘||fh2||‘\n‘,r2);
end if;
if fileaus=ja then
 fprintf(file1,‘\n‘);
 fprintf(file1,‘Schritt k = %g\n‘,k);
 fprintf(file1,‘Iterationsvektor      x =
               [‘||fh||‘]\n‘,seq(x[i],i=1..n));
 fprintf(file1,‘Funktionswert      Q(x) = ‘||fh2||‘\n‘,Q);
 fprintf(file1,‘Funktionswert      R(x) = ‘||fh2||‘\n‘,R);
 fprintf(file1,‘Suchrichtung          p =
               [‘||fh||‘]\n‘,seq(p[i],i=1..n));
 fprintf(file1,‘Residuum       r = b-Ax =
               [‘||fh||‘]\n‘,seq(r[i],i=1..n));
 fprintf(file1,‘Fehlernormquadrat   r’r = ‘||fh2||‘\n‘,r2);
end if;

fprintf(default,‘k = %3d,  ||r||_2 = ‘||fh2||‘\n‘,k,sr2);
if fileaus=ja then
 fprintf(file1,‘k = %3d,  ||r||_2 = ‘||fh2||‘\n‘,k,sr2);
end if;

Qv[k+1]:=Q;
Rv[k+1]:=R;
xh:=evalm(xs-x);
v:=Ax(n,n1,trid,A,xh);
epsv[k+1]:=abs(sqrt(evalm(transpose(xh)&*v)));
          # epsv[k+1]:=abs(sqrt(2.0*Q+h));
epsv2[k+1]:=sqrt(evalm(transpose(xh)&*xh));
rv[k+1]:=sr2;
end do:

lprint(‘||xs-x||_2    = ‘,evalf(norm(xs-x,frobenius))):
lprint(‘||A^(-1)r||_2 = ‘,evalf(norm(Ainv&*r,frobenius))):
lprint(‘ ‘):

[x,k];
end:
```

Für die speziellen Tridiagonalmatrizen und Blocktridiagonalmatrizen mit Shift aus Abschnitt 6.3.4 stellen wir vergleichenden Berechnungen zu den AV CG und CR an. Mit den absoluten Fehlern

$$\|e^{(k)}\|_A, \quad \|e^{(k)}\|_2, \quad \|r^{(k)}\|_2, \quad k = 0, 1, 2, ...,$$

die man als Ergebnisvektoren `epsv[1..maxiter+1], epsv2[1..maxiter+1]` bzw. `rv[1..maxiter+1]` der obigen Prozeduren `cg6` und `cr6` erhält, folgen grafische Vergleiche der relativen Fehler der AV, die aus Gründen der besseren Übersicht als Funktionen $\log_{10}()$ in Abhängigkeit von der Iterationszahl k dargestellt werden, also

$$f(k) \;=\; \log_{10}\left(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}\right),\ \log_{10}\left(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2}\right),\ \log_{10}\left(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}\right),\ k=0,1,2,\ldots\,. \quad (6.106)$$

Test-LGS: $A1(n,n)$ und $A2(n,n)$, $n = n_1^2$, $\quad n = 100,\ 625$

```
  # Tridiagonalmatrix TDM mit Shift
> Digits := 50:
  c := 0.0:     # 0.0, 1.0, 2.0, -1.0, -1.5  Shift
  trid := 3:
  n1 := 1:      # nicht relevant bei TDM
> n := 100:     # 100, 625,
  A1 := band([-1.0,2.0,-1.0],n):
  A1 := evalm(A1+c*diag(1.0$n)):

  # Loesung
> xs1 := evalm(vector(n,[1.0$n])):
  # rechte Seite
> b1 := evalm(vector(n,[1.0+c,c$(n-2),1.0+c])):
  # b1 := evalm(A1&*xs1):

  # ------------------------------------------------
  # Blocktridiagonalmatrix BTM mit Shift
> Digits := 50:
  c := 0.0:       # 0.0, 1.0, 2.0, -1.0, -2.0  Shift
  trid := 5:
> n1 := 10:       # 10, 25
  n := n1^2:
> T := band([-1.0,4.0,-1.0],n1):
  A2 := diag(T$n1):
  for i from 1 to n-n1 do
   A2[i,i+n1] := -1.0;
   A2[i+n1,i] := -1.0;
  end do:
  A2 := evalm(A2+c*diag(1.0$n)):

  # Loesung
> xs2 := evalm(vector(n,[1.0$n])):
  # rechte Seite
> b21 := 2.0+c,(1.0+c)$(n1-2),2.0+c:
  b22 := 1.0+c,c$(n1-2),1.0+c:
  b2 := vector(n,[b21,seq(b22,i=2..n1-1),b21]):
  # b2 := evalm(A2&*xs2):
```

Die folgenden Abbildungen zeigen die Entwicklung und den Verlauf der drei genannten relativen Fehler von CG und CR in verschiedenen Gegenüberstellungen und geeigneten Ausschnitten (Zoomtechnik).
Dabei sind die Fehler in der Grafik zu **einem** AV gekennzeichnet mit den Linienarten

$\log_{10}(\|e^{(k)}\|_A/\|e^{(0)}\|_A)$	solid line	———
$\log_{10}(\|e^{(k)}\|_2/\|e^{(0)}\|_2)$	dot line	.....
$\log_{10}(\|r^{(k)}\|_2/\|r^{(0)}\|_2)$	dash line	- - - - -

und bei Gegenüberstellung der **zwei** AV als CG mit relativem Fehler (`solid line, bold`) bzw. CR mit relativem Fehler (`dash line, thin`).

Ergebnisse zu $A1(100, 100)$ tridiagonal

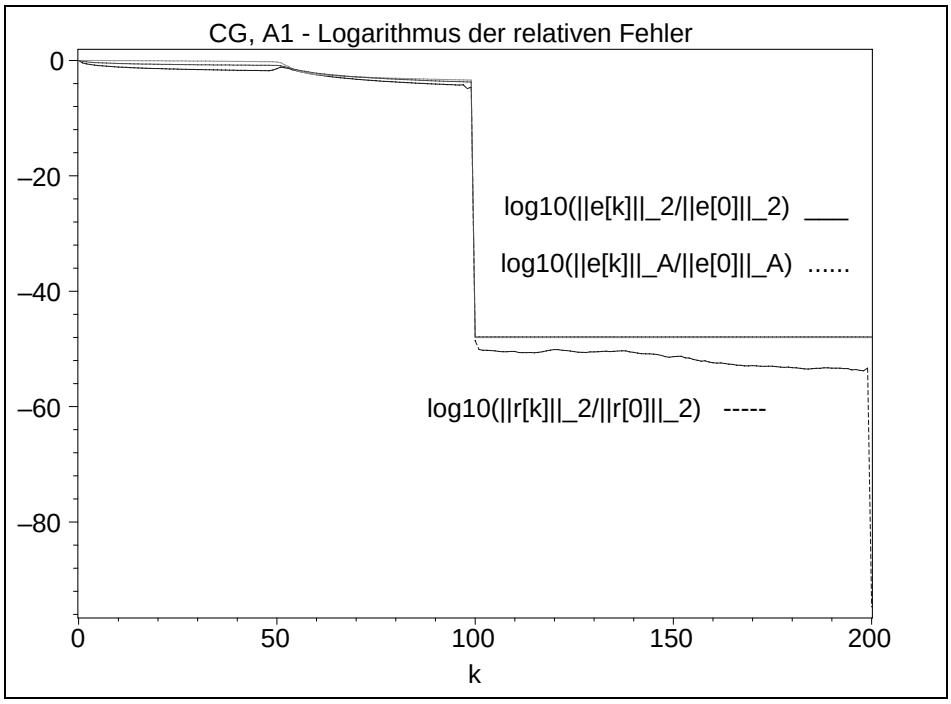


Abb. 6.3 Datei $ccga1c01.ps$, Matrix $A1(100, 100)$, Shift $c = 0$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

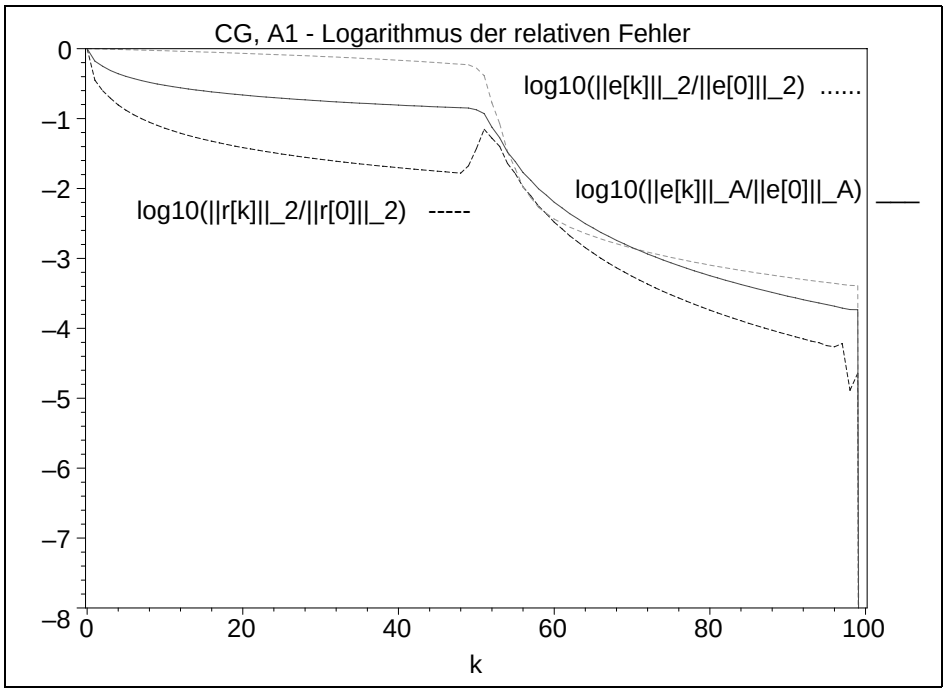


Abb. 6.4 Datei $ccga1c01a.ps$, Matrix $A1(100, 100)$, Shift $c = 0$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$ (Zoom)

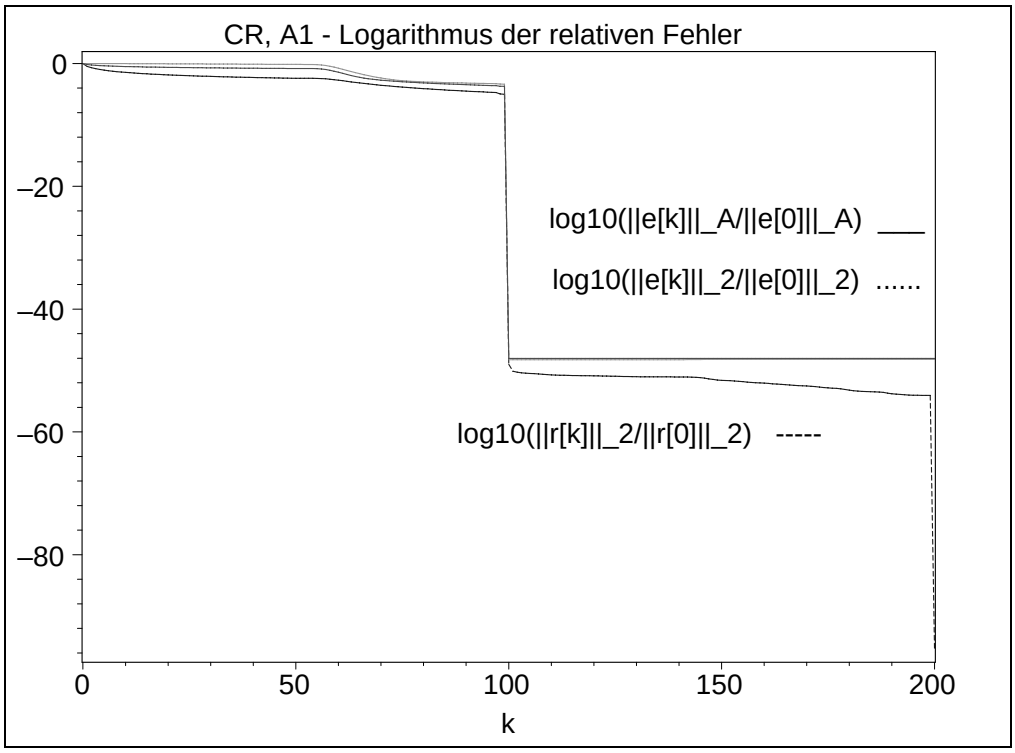


Abb. 6.5 Datei *ccra1c01.ps*, Matrix $A1(100, 100)$, Shift $c = 0$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

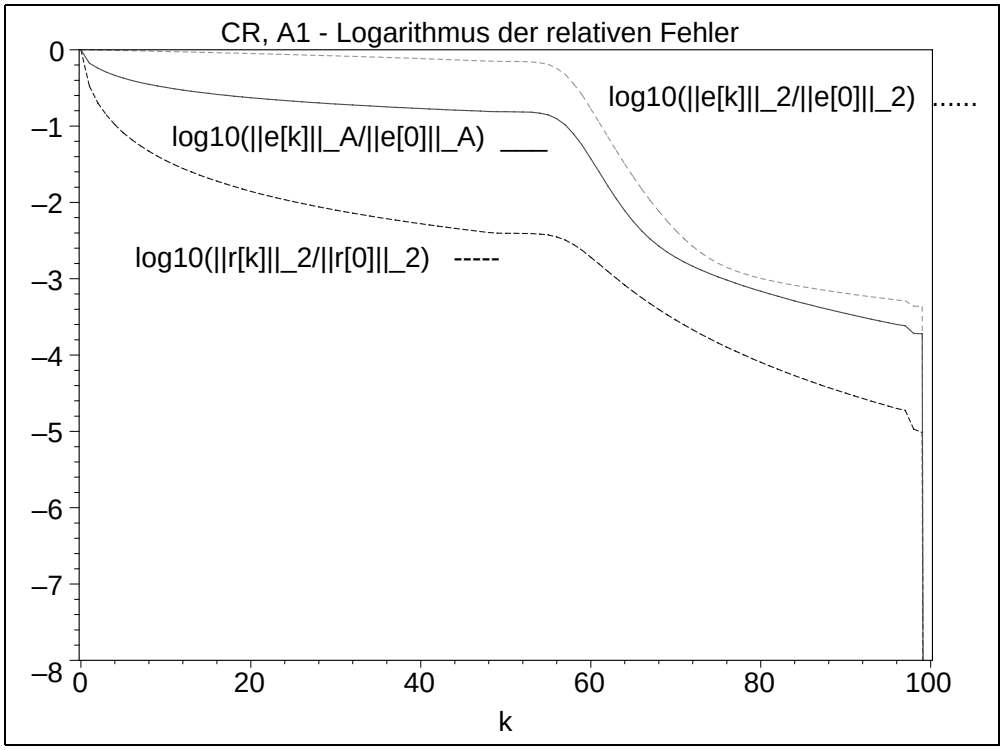


Abb. 6.6 Datei *ccra1c01a.ps*, Matrix $A1(100, 100)$, Shift $c = 0$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$ (Zoom)

Die beiden AV CG und CR sind für die spd Matrix $A1 = A1(100, 100)$, Shift $c = 0$, theoretisch endlich und brauchen höchstens $n = 100$ Schritte.
Die numerischen Rechnungen in der sehr genauen GPA mit `Digits:=50` widerspiegeln dieses Verhalten, wobei bei $n/2$ sichtbare Verbesserungen und erst kurz vor bzw. bei n die deutliche Fehlerverkleinerung stattfinden. Solche "Genauigkeitssprünge" treten bei Vielfachen von n auch auf. Dazwischen liegen i. Allg. nur kleine Genauigkeitszuwächse. Alle Fehler tendieren natürlich gegen Null. Die Konvergenz ist langsam, weil die spektrale Kondition der Matrix $\kappa = 4133.642$ groß und damit der Quotient

$$q \;=\; \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \;=\; 0.969$$

im Konvergenzfaktor q^k nahe der Eins liegt.

Im Fall A spd erzeugt das CG eine monoton fallende Folge von Werten $\|e^{(k)}\|_A$ und das gilt auch für $\|e^{(k)}\|_2$ (siehe [11]), während sich dieses Eigenschaft nicht auf die Residuumnorm $\|r^{(k)}\|_2$ überträgt. Das CR erzeugt eine monoton fallende Folge von Werten $\|r^{(k)}\|_2$ und das gilt auch für die Fehlernormen $\|e^{(k)}\|_{A^TA}$ (siehe 6.67) und $\|e^{(k)}\|_2$ (siehe Abschnitt 6.2.5) sowie meistens auch für $\|e^{(k)}\|_A$.

Ergebnisse aus Berechnungen mit Maple

Startvektor $x^{(0)} = (1, 0, 0, ..., 0)^T$, die rechte Seite b wird so gewählt, dass die exakte Lösung $x^* = (1, 1, ..., 1)^T$ ist, $Q(x^*) = -\frac{1}{2}x^{*T}b = -1$, $R(x^*) = -\frac{1}{2}b^Tb = -1$.

```
CG : Iterationsverlauf  mit verschiedenen Fehlern
```

i	Q(x[i])= (||e[i]||^2_A-xs'b)/2	||e[i]||_A=||xs-x[i]||_A =sqrt(2Q(x[i])+xs'b)	||e[i]||_2= ||xs-x[i]||_2	||r[i]||_2= ||b-Ax[i]||_2
0	0.0000000000000000e+00	1.4142135623730950e+00	9.9498743710661995e+00	1.7320508075688773e+00
1	-5.6250000000000000e-01	9.3541434669348535e-01	9.8955482415073901e+00	6.1237243569579452e-01
2	-6.8750000000000000e-01	7.9056941504209483e-01	9.8385402880711933e+00	4.3301270189221932e-01
3	-7.6250000000000000e-01	6.8920243760451109e-01	9.7684121022815167e+00	3.3911649915626341e-01
4	-8.0831408775981524e-01	6.1917027099205071e-01	9.6996148123665450e+00	2.7062205477269659e-01
5	-8.3904569892473118e-01	5.6736989887597812e-01	9.6308067140027897e+00	2.2752799967203039e-01
6	-8.6137820512820513e-01	5.2653925755216936e-01	9.5610868084256588e+00	1.9611613513818403e-01
7	-8.7825584225900682e-01	4.9344535207253332e-01	9.4909691227581839e+00	1.7215261469580199e-01
8	-8.9146719234018586e-01	4.6590301063593512e-01	9.4203521196363611e+00	1.5348899223289991e-01
9	-9.0209378369509165e-01	4.4250698594464777e-01	9.3492020888588396e+00	1.3846202502834299e-01
10	-9.1082456710596300e-01	4.2231607332432185e-01	9.2775249525063955e+00	1.2611239252975046e-01
...				
46	-9.7881977366426983e-01	2.0581655101439327e-01	6.1688697624376245e+00	2.9953363338121414e-02
47	-9.7925907123003103e-01	2.0367095409001733e-01	6.0598256587457008e+00	2.9332102762704438e-02
48	-9.7968051619624720e-01	2.0159109009950216e-01	5.9487834950868423e+00	2.8736089575687653e-02
49	-9.8008517515709714e-01	1.9957366982096040e-01	5.8356291881490573e+00	3.6634928535231181e-02
50	-9.8211802069014463e-01	1.8911361299417537e-01	5.2471557269007441e+00	6.4479624320772592e-02
51	-9.8607940425578742e-01	1.6685679934730007e-01	4.0887404493964836e+00	1.2158535049195702e-01
52	-9.9435950008762339e-01	1.0621205122185159e-01	1.6643246681351613e+00	9.0592126004430260e-02
53	-9.9717126206729499e-01	7.5216194169939331e-02	8.4123520685311445e-01	6.9347573583406189e-02
54	-9.9890394108280373e-01	4.6820058034912043e-02	3.3628798012720312e-01	3.9611556103418771e-02
55	-9.9941023480406673e-01	3.4344291983771472e-02	1.9049126103784081e-01	2.8271084925483177e-02
56	-9.9971350449555091e-01	2.3937230602101391e-02	1.0509291789416009e-01	1.8276869302777970e-02
57	-9.9982759751954063e-01	1.8568924603184164e-02	7.3785406338384905e-02	1.3607362642169915e-02
58	-9.9990209235651387e-01	1.3993401551169106e-02	5.3435987398372842e-02	9.6936086082524931e-03
59	-9.9993595367025126e-01	1.1317802768094459e-02	4.3782544119487365e-02	7.5238193711179504e-03
60	-9.9995967482665534e-01	8.9805538074949883e-03	3.6308002095191291e-02	5.7169838331495342e-03

```
...
 90  -9.9999991673786636e-01  4.0807385027313578e-04  5.3273587497804452e-03  1.4057903511741740e-04
 91  -9.9999992573122022e-01  3.8540570773331604e-04  5.1476996372223225e-03  1.3078020693730777e-04
 92  -9.9999993392591590e-01  3.6352189507044574e-04  4.9701240246364960e-03  1.2231797306058303e-04
 93  -9.9999994074344540e-01  3.4425732992578682e-04  4.8102858644735600e-03  1.1401571133444982e-04
 94  -9.9999994703929261e-01  3.2545570326615905e-04  4.6506826262118239e-03  1.0810957806864678e-04
 95  -9.9999995234314745e-01  3.0872917758548657e-04  4.5061757893700331e-03  9.8246024124274322e-05
 96  -9.9999995681687121e-01  2.9388136651406531e-04  4.3737333193217495e-03  9.4413159862330889e-05
 97  -9.9999996197675379e-01  2.7576528500379629e-04  4.2105710572729169e-03  1.0469652819685816e-04
 98  -9.9999996551160015e-01  2.6263434599001654e-04  4.0961672434221437e-03  2.2482133712107785e-05
 99  -9.9999996602734663e-01  2.6066320558506495e-04  4.0457695769842436e-03  3.9522331272582655e-05
-----------------------------------------------------------------------------------------------------
100  -1.0000000000000000e+00  1.8079823007983236e-48  9.0480053050382326e-48  4.6715407814001545e-49
-----------------------------------------------------------------------------------------------------
101  -1.0000000000000000e+00  1.8085906114983568e-48  9.0480108311164173e-48  1.3982821707364548e-50
102            "                        "                       "              1.0344256061401024e-50
...
197            "                        "                       "              3.3920116198710879e-54
198            "                        "                       "              2.8534290834278875e-54
199            "                        "                       "              8.5809146238921482e-54
200  -1.0000000000000000e+00  1.8085906114983568e-48  9.0480108311164173e-48  3.3217615111071637e-95


CR : Iterationsverlauf  mit verschiedenen Fehlern

 i         Q(x[i])=          ||e[i]||_A=        ||e[i]||_2=    ||r[i]||_2=            R(x[i])=
      (||e[i]||^2_A-xs'b)/2  ||xs-x[i]||_A=     ||xs-x[i]||_2  ||b-Ax[i]||_2=     (||r[i]||^2_2-b'b)/2
                             sqrt(2Q(x[i])+xs'b)               sqrt(2R(x[i])+b'b)
-----------------------------------------------------------------------------------------------------
  0   0.0000000000000000e+00  1.414213562e+00  9.949874371e+00  1.732050807e+00  5.0000000000000000e-01
  1  -5.5555555555555556e-01  9.428090415e-01  9.899494936e+00  5.773502691e-01 -8.3333333333333333e-01
  2  -6.7040000000000000e-01  8.119113252e-01  9.858032258e+00  3.464101615e-01 -9.4000000000000000e-01
  3  -7.4044436396559529e-01  7.204937696e-01  9.808541727e+00  2.423291238e-01 -9.7063829787234043e-01
  4  -7.8740927960223195e-01  6.520593844e-01  9.755755688e+00  1.805294829e-01 -9.8370455289878808e-01
  5  -8.1960000000000000e-01  6.006662967e-01  9.703006120e+00  1.414213562e-01 -9.9000000000000000e-01
  6  -8.4329553324099723e-01  5.598293789e-01  9.649823134e+00  1.147078669e-01 -9.9342105263157895e-01
  7  -8.6148883634549666e-01  5.263291055e-01  9.596238066e+00  9.545820586e-02 -9.9544386546660351e-01
  8  -8.7587927195268092e-01  4.982383526e-01  9.542363728e+00  8.106032845e-02 -9.9671461157532709e-01
  9  -8.8755383966027194e-01  4.742281314e-01  9.488181974e+00  6.995416863e-02 -9.9755320714555128e-01
 10  -8.9721635217058600e-01  4.533952973e-01  9.433693937e+00  6.117322823e-02 -9.9812891807340583e-01
...
 50  -9.7628182325316343e-01  2.177988831e-01  6.997921829e+00  6.826128219e-03 -9.9997670198676685e-01
 60  -9.9860483174079736e-01  5.282363598e-02  1.678498354e+00  3.314149263e-03 -9.9999450820732878e-01
 70  -9.9999639097466506e-01  2.686643011e-03  4.260487744e-02  4.984442837e-04 -9.9999987577664799e-01
 80  -9.9999952974180361e-01  9.698022441e-04  1.006478597e-02  1.394646453e-04 -9.9999999027480634e-01
 90  -9.9999987725492666e-01  4.954696223e-04  6.406609491e-03  5.487453852e-05 -9.9999999849439251e-01
...
 95  -9.9999992968875004e-01  3.749966665e-04  5.408111362e-03  3.717792949e-05 -9.9999999930890078e-01
 96  -9.9999993648351182e-01  3.564168575e-04  5.244180436e-03  3.459254818e-05 -9.9999999940167781e-01
 97  -9.9999994125486290e-01  3.427685431e-04  5.123225326e-03  3.284608656e-05 -9.9999999946056730e-01
 98  -9.9999996304271317e-01  2.718723480e-04  4.369213593e-03  1.855243161e-05 -9.9999999982790364e-01
 99  -9.9999996402323955e-01  2.682415346e-04  4.306339390e-03  1.679416127e-05 -9.9999999985897807e-01
-----------------------------------------------------------------------------------------------------
100  -1.0000000000000000e+00  1.285457117e-48  5.834989288e-48  1.742687231e-49 -1.0000000000000000e+00
-----------------------------------------------------------------------------------------------------
101            "              1.305220287e-48  5.840239721e-48  1.310656099e-50          "
102            "              1.305220287e-48  5.840239721e-48  9.045345342e-51          "
...
149            "              1.340074624e-48  6.562248090e-48  4.988403985e-52          "
150            "              1.340000000e-48  6.575180605e-48  4.323844535e-52          "
...
197            "                     "               "         1.554362920e-54          "
198            "                     "               "         1.515213524e-54          "
199            "                     "               "         1.500830512e-54          "
200  -1.0000000000000000e+00  1.340000000e-48  6.575180605e-48  4.449693400e-96 -1.0000000000000000e+00
```

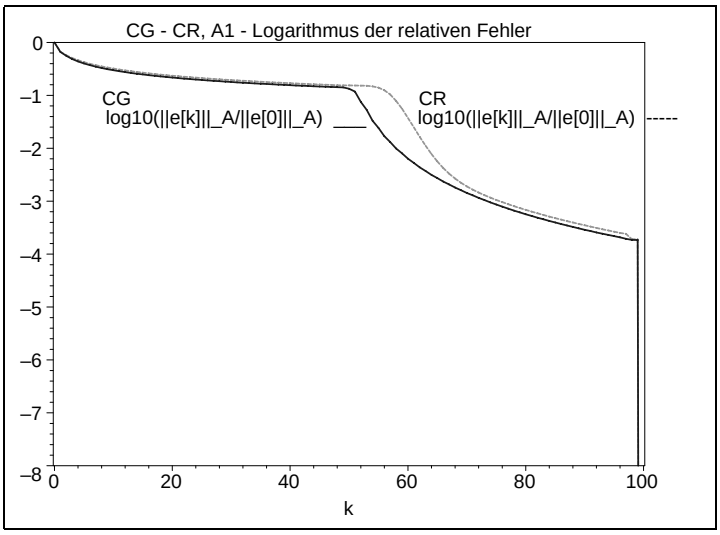


Abb. 6.7
Datei $ccgra1c011.ps$
Matrix $A1(100,100)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)100$

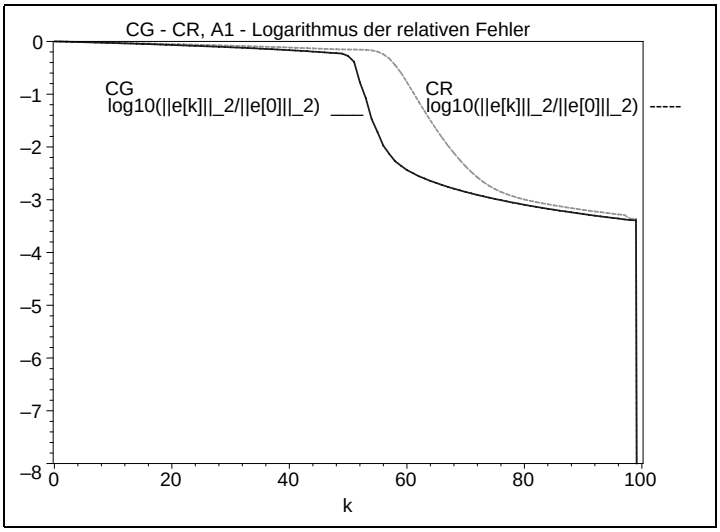


Abb. 6.8
Datei $ccgra1c012.ps$
Matrix $A1(100,100)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)100$

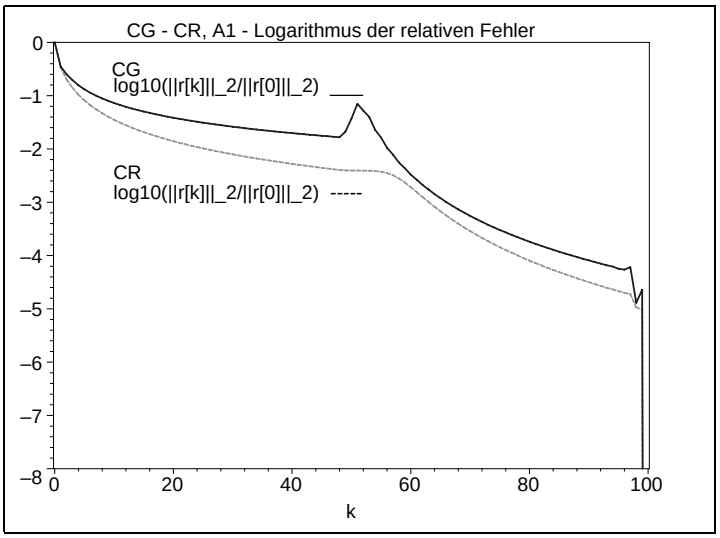


Abb. 6.9
Datei $ccgra1c013.ps$
Matrix $A1(100,100)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)100$

Für die Matrix $A1(100, 100)$ mit Shift $c = 1$ liegen die Fehler bei den ersten 100 Iterationen des CG sehr dicht beieinander. Es gilt $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$. Zusätzlich ist $\|e^{(0)}\|_2 = 9.949...$, $\|e^{(0)}\|_A = 10.049...$, $\|r^{(0)}\|_2 = 10.029...$, so dass sich die relativen Fehler ebensfalls nur minimal unterscheiden.

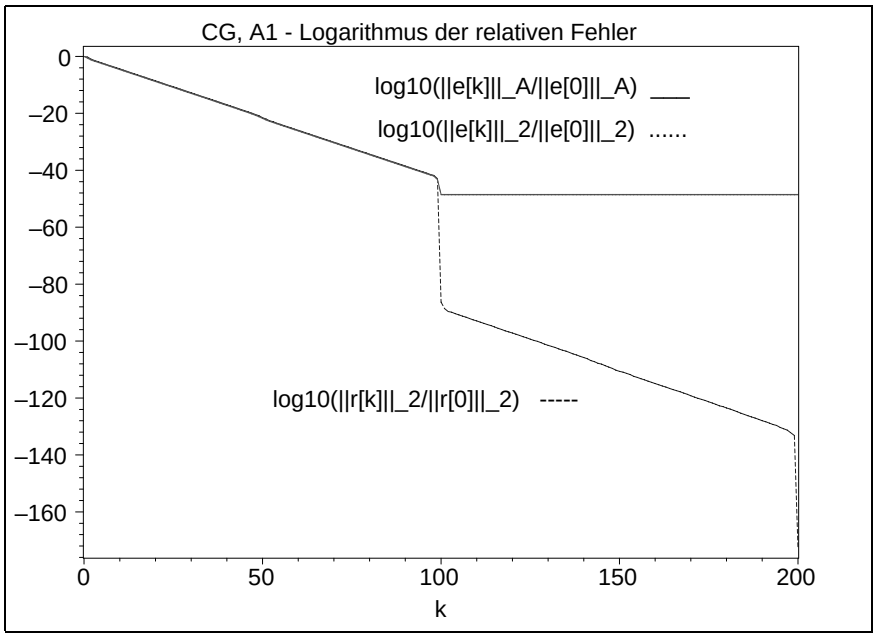


Abb. 6.10 Datei *ccga*1*c*11.*ps*, Matrix $A1(100, 100)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

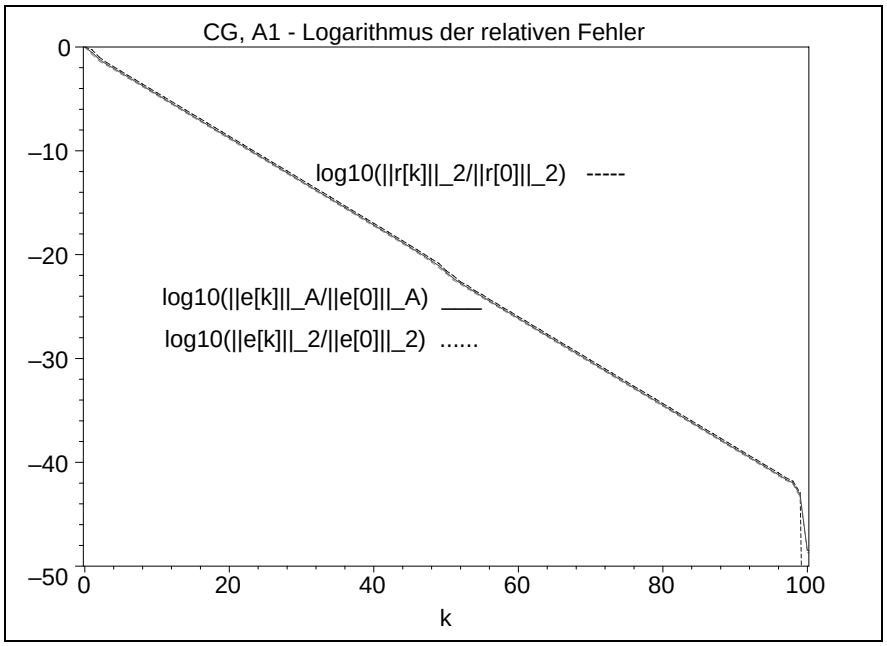


Abb. 6.11 Datei *ccga*1*c*11*a.ps*, Matrix $A1(100, 100)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Analoges gilt für das CR bei den ersten 100 Iterationen und genauso ist $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$.

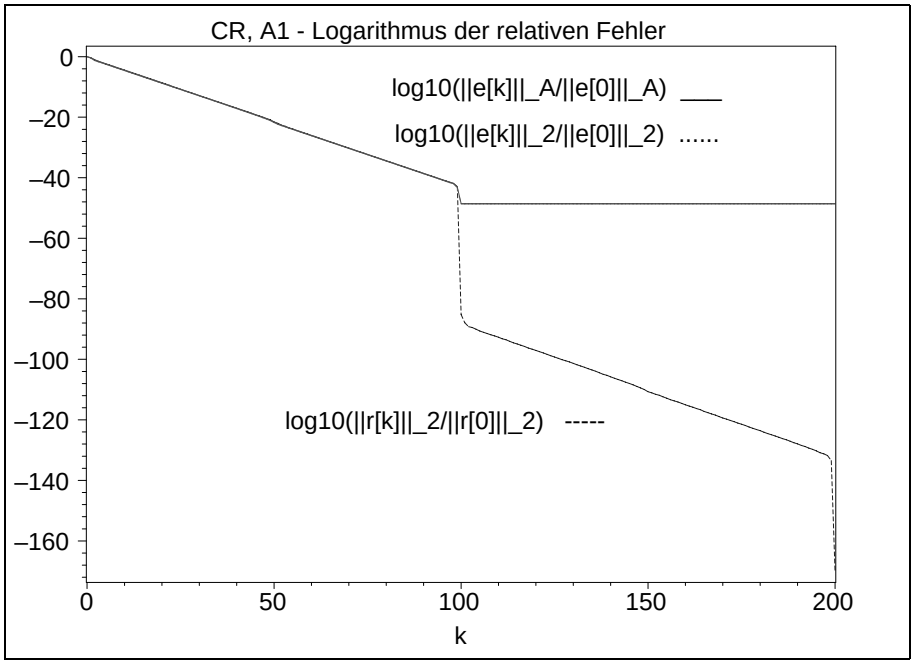


Abb. 6.12 Datei *ccra1c11.ps*, Matrix $A1(100, 100)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

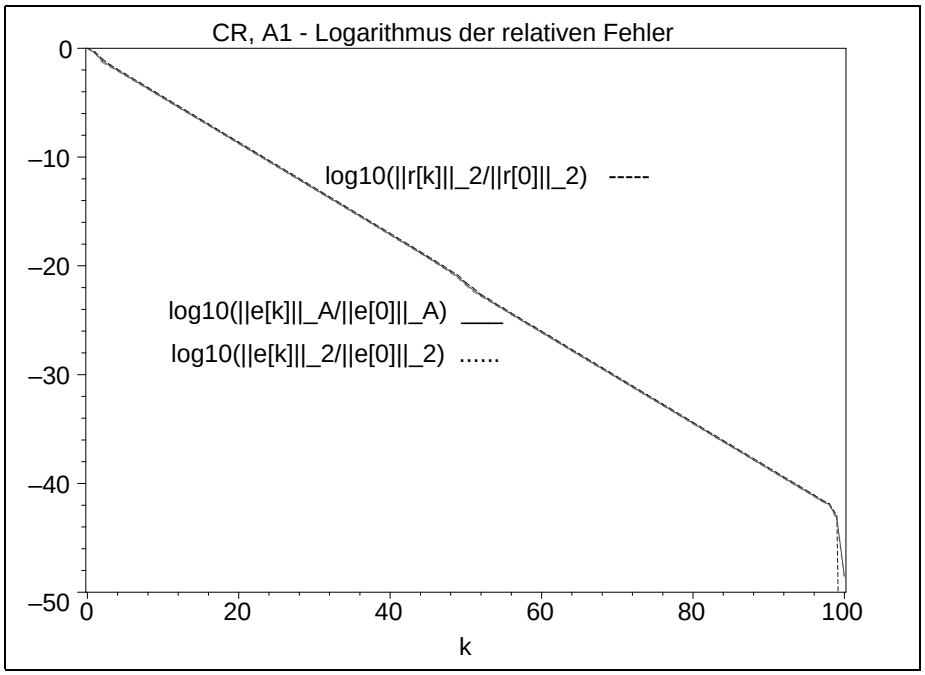


Abb. 6.13 Datei *ccra1c11a.ps*, Matrix $A1(100, 100)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Im Vergleich der AV ist bei den Fehlern $\|e^{(k)}\|_A$ und $\|e^{(k)}\|_2$ das CG minimal besser als CR, bei $\|r^{(k)}\|_2$ umgekehrt.

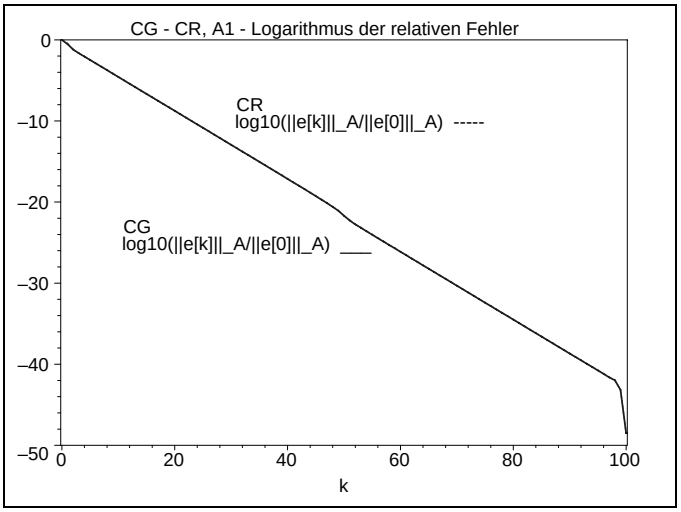


Abb. 6.14
Datei *ccgra1c111.ps*
Matrix $A1(100, 100)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)100$

Ein sehr ähnliches Bild wie in Abb. 6.14 ergibt sich zu CG versus CR für den Verlauf des relativen Fehlers $\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)100$.

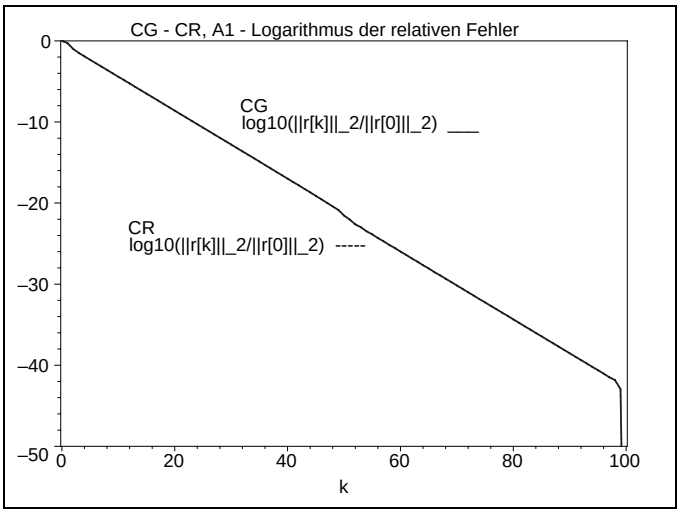


Abb. 6.15
Datei *ccgra1c113.ps*
Matrix $A1(100, 100)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)100$

Mit dem Schift $c = 1$ wird die Diagonaldominanz der Matrix $A1$ verstärkt.
Damit verbessert sich die spektrale Kondition der Matrix auf $\kappa = 4.994$ und man erhält den Quotienten $q = 0.381$ deutlich kleiner als Eins im Konvergenzfaktor q^k.
Somit nehmen die Fehler stetig ab.

Für die Matrix $A1(100, 100)$ mit Shift $c = 2$ folgen analoge Aussagen und Ergebnisse wie in der Situation zu $c = 1$. Die Fehler tendieren jedoch noch schneller gegen Null wegen $\kappa = 2.998$ und $q = 0.267$.

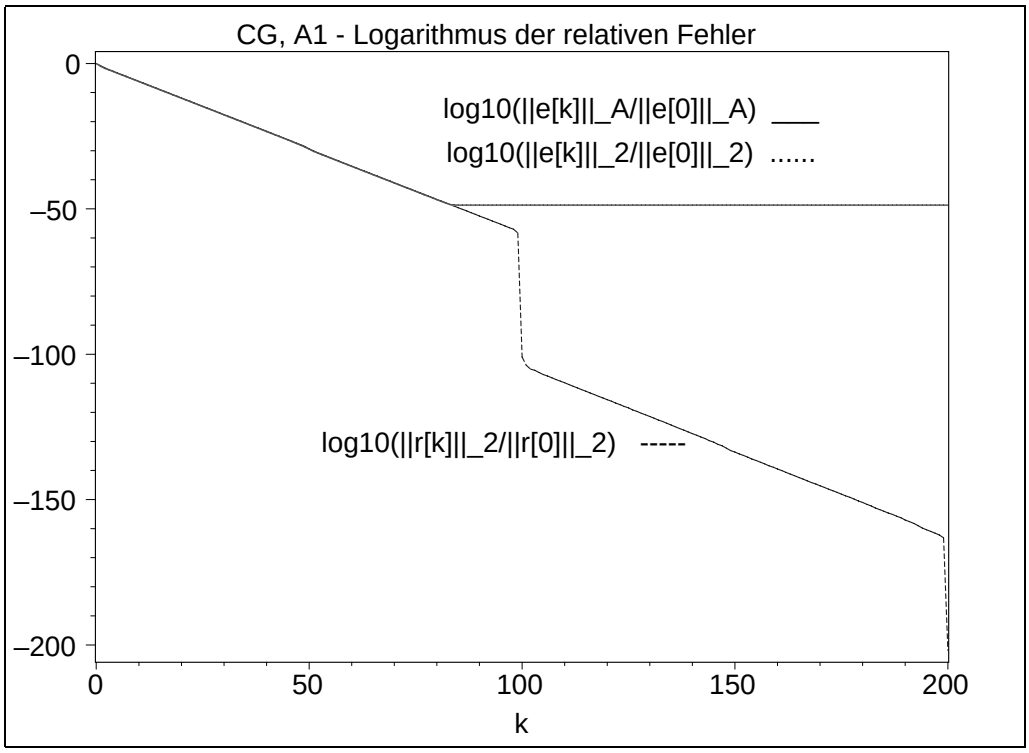


Abb. 6.16 Datei *ccga*1*c*21.*ps*, Matrix $A1(100, 100)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

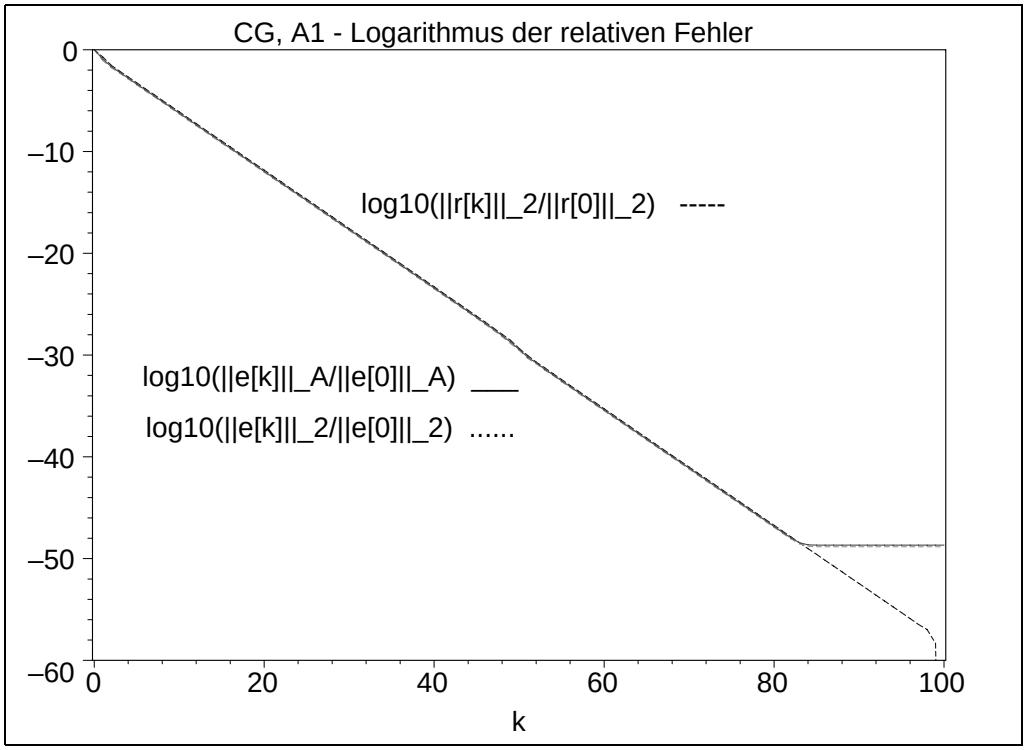


Abb. 6.17 Datei *ccga*1*c*21*a.ps*, Matrix $A1(100, 100)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

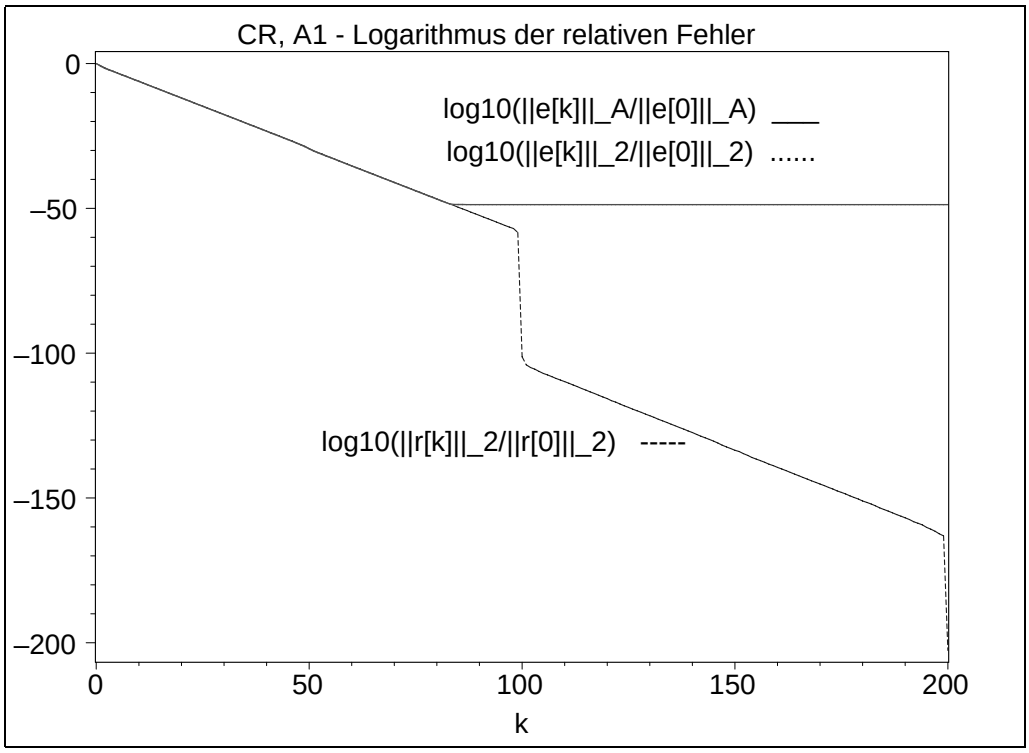


Abb. 6.18 Datei *ccra1c21.ps*, Matrix $A1(100, 100)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

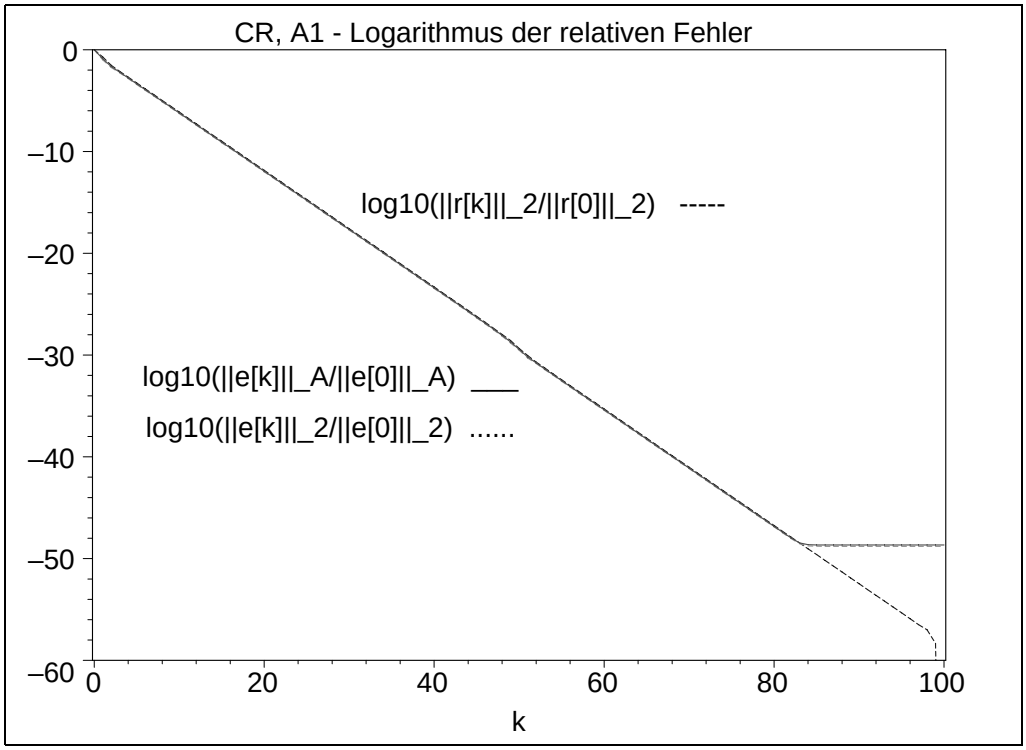


Abb. 6.19 Datei *ccra1c21a.ps*, Matrix $A1(100, 100)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Im Teil II, Abschnitte 5.2 und 5.4, sind die AV CG und CR unter veränderten Voraussetzungen getestet worden. Sowohl für symmetrische indefinite Koeffizientenmatrizen als auch im allgemeinen Fall haben wir in einigen Beispielen konvergente Iterationsfolgen erhalten. Andererseits konnten jedoch auch Probleme bei der Durchführung wie vorzeitiger Abbruch (Nulldivision bei Schrittzahl) und Divergenz auftreten.

Deshalb soll hier zusätzlich die symmetrische tridiagonale Matrix $A1(100, 100)$ mit dem Shift $c = -1$ betrachtet werden, womit dann ihre positive Definitheit verloren geht. Glücklicherweise ist die Matrix regulär. Aber die Regularität geht z. B. bei $n = 20$ und $n = 200$ verloren. Ihre spektrale Konditionszahl ist $\kappa = 166.503$.
Wie bei den anderen Shifts interessieren wir uns für die Fehlerverläufe von CG und CR. Dabei kann man beim CG kein monoton abnehmendes Verhalten der Fehler erwarten. Aber, wenn es nicht vorzeitig abbricht, ist es nach spätestens n Schritten (theoretisch) an der Lösung. Für das CR erhält man eine stetig abnehmende Folge von Fehlern $\|r^{(k)}\|_2$ und damit auch von $\|e^{(k)}\|_2$ mit spätestens $\|e^{(n)}\|_2 = 0,\ \|r^{(n)}\|_2 = 0$ (theoretisch), auch hier vorausgesetzt, dass kein vorzeitiger Abbruch eintritt.

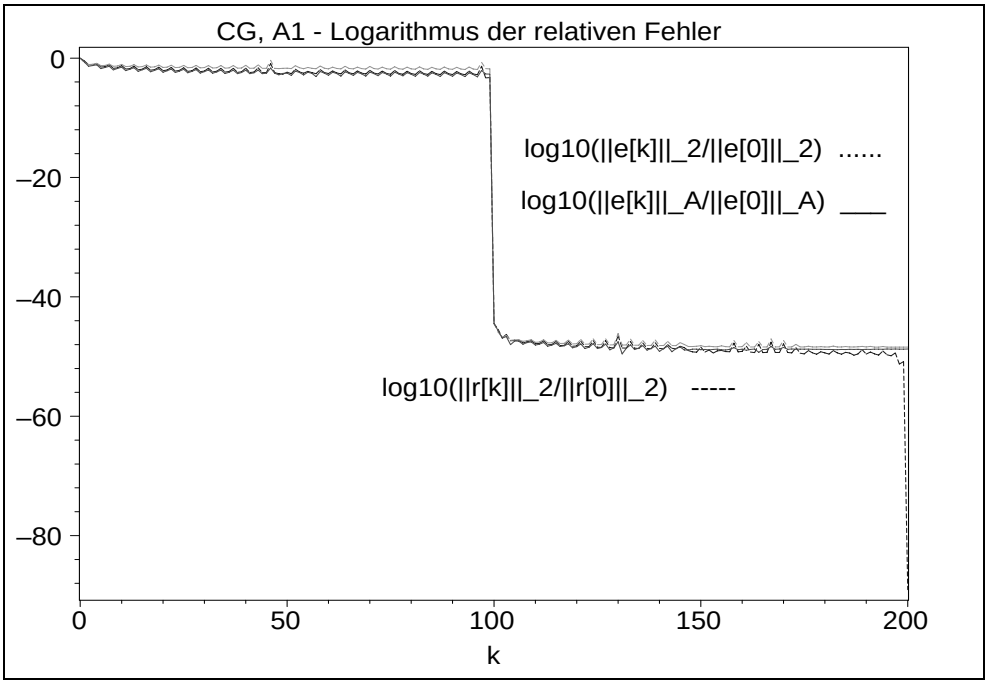
CG : Iterationsverlauf mit verschiedenen Fehlern

i	Q(x[i])= (||e[i]||^2_A-xs'b)/2	||e[i]||_A=||xs-x[i]||_A =sqrt(2Q(x[i])+xs'b)	||e[i]||_2= ||xs-x[i]||_2	||r[i]||_2= ||b-Ax[i]||_2
0	5.0000000000000000e-01	9.8488578017961047e+00	9.9498743710661995e+00	9.8994949366116653e+00
1	5.1585106382978723e+01	2.2738101868796012e+00	1.8062564173892619e+00	3.4320012916008675e+00
2	4.9084410170904544e+01	4.1087752653203986e-01	8.5032315286715664e-01	7.8400589240135916e-01
3	4.8844321013289037e+01	5.5799459981430547e-01	9.1701118116390518e-01	6.9315658787957674e-01
4	4.9272865572561313e+01	7.3873618100281636e-01	1.2668524149555819e+00	1.3054750076141928e+00
5	4.9014434904776772e+01	1.6991118136704115e-01	5.4873415373663436e-01	3.1914850663886181e-01
6	4.8962427479661435e+01	2.7412595768575148e-01	6.0786194603334466e-01	3.5009901510336904e-01
7	4.9092971754114268e+01	4.3121167450399132e-01	1.0082261436757950e+00	7.8080917520517559e-01
8	4.9005648098724350e+01	1.0628357092561661e-01	4.3537229457122492e-01	1.9755663662587520e-01
...				
97	4.8996369104379080e+01	8.5216144255884130e-02	1.9629319039133859e+00	4.1610229139626280e-01
98	4.9000230367447483e+01	2.1464736079582463e-02	1.7495680971123633e-01	4.8298592570897498e-03
99	4.9000204918032787e+01	2.0244408254472900e-02	1.5964153778344787e-01	5.2644395814201348e-03
100	4.9000000000000000e+01	3.3199610784314927e-44	3.3232983669691772e-44	3.3192332742482961e-44
101	4.9000000000000000e+01	2.1657483917343677e-45	1.2652586361293884e-45	3.7576485453894170e-45
102	4.9000000000000000e+01	1.2207506051606118e-46	1.4907823248214342e-46	1.3832573305141553e-46

CR : Iterationsverlauf mit verschiedenen Fehlern

i	Q(x[i])= (||e[i]||^2_A-xs'b)/2	||e[i]||_A= ||xs-x[i]||_A= sqrt(2Q(x[i])+xs'b)	||e[i]||_2= ||xs-x[i]||_2	||r[i]||_2= ||b-Ax[i]||_2= sqrt(2R(x[i])+b'b)	R(x[i])= (||r[i]||^2_2-b'b)/2
0	5.0000000000000000e-01	9.848857801e+00	9.949874371e+00	9.899494936e+00	0.0000000000000000e+00
1	5.0997010097049309e+01	1.998504489e+00	1.825410721e+00	3.242661173e+00	-4.3742574257425743e+01
2	4.9090243934537600e+01	4.248386388e-01	8.587296820e-01	7.620487324e-01	-4.8709640864714086e+01
3	4.8894734040468245e+01	4.588375737e-01	8.300702973e-01	5.127653697e-01	-4.8868535837786859e+01
4	4.8989058600444105e+01	1.479283580e-01	7.442914343e-01	4.772695514e-01	-4.8886106887638741e+01
5	4.9012012121454340e+01	1.549975577e-01	5.678943605e-01	2.652987958e-01	-4.8964808274470232e+01
6	4.8982435710741257e+01	1.874261948e-01	5.490603689e-01	2.114465761e-01	-4.8977645172707520e+01
7	4.8997024333341326e+01	7.714488523e-02	5.280027489e-01	2.040953004e-01	-4.8979172554159506e+01
8	4.9003630225893968e+01	8.520828473e-02	4.561727977e-01	1.419489615e-01	-4.8989925246154862e+01
...					
97	4.9000004080910611e+01	2.856890131e-03	1.364977852e-01	3.706761129e-03	-4.8999993129960965e+01
98	4.9000140746971011e+01	1.677778120e-02	1.370607766e-01	2.940571429e-03	-4.8999995676519834e+01
99	4.9000167638593095e+01	1.831057580e-02	1.392412813e-01	2.567226996e-03	-4.8999996704672774e+01
100	4.9000000000000000e+01	2.404643542e-43	2.405837469e-43	2.403440368e-43	-4.9000000000000000e+01
101	4.9000000000000000e+01	1.397385040e-45	8.379934024e-46	2.438296023e-45	-4.9000000000000000e+01
102	4.9000000000000000e+01	2.250986035e-46	2.460196567e-46	2.314869139e-46	-4.9000000000000000e+01

Die fehlende Definitheit der Matrix führt zu einer "Sattelpunktsituation" an der Lösung und zu oszillierendem Verhalten aller Fehler des CG.



.

Abb. 6.20 Datei *ccga*1*cm*11.*ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

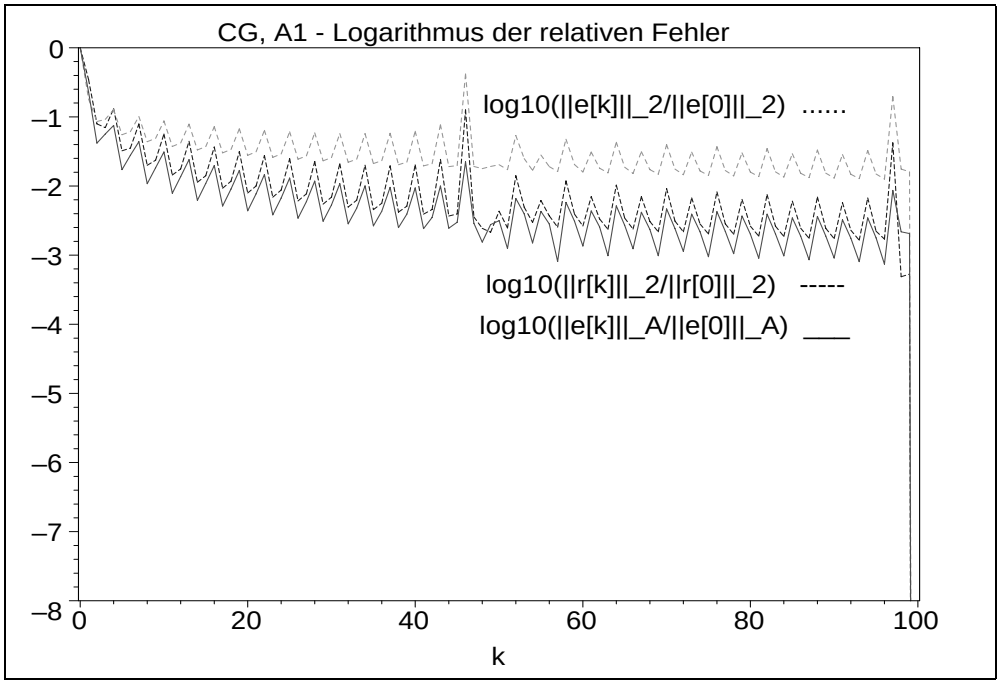


Abb. 6.21 Datei *ccga*1*cm*11*a.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Die fehlende Definitheit der Matrix hat keinen Einfluss auf das Abstiegsverhalten der Iterationsfolge $\|r^{(k)}\|_2$ im CR, das gilt auch für $\|e^{(k)}\|_2$. Man bemerkt jedoch ein oszillierendes Verhalten oder einen unregelmäßigen Verlauf der Fehler $\|e^{(k)}\|_A$.

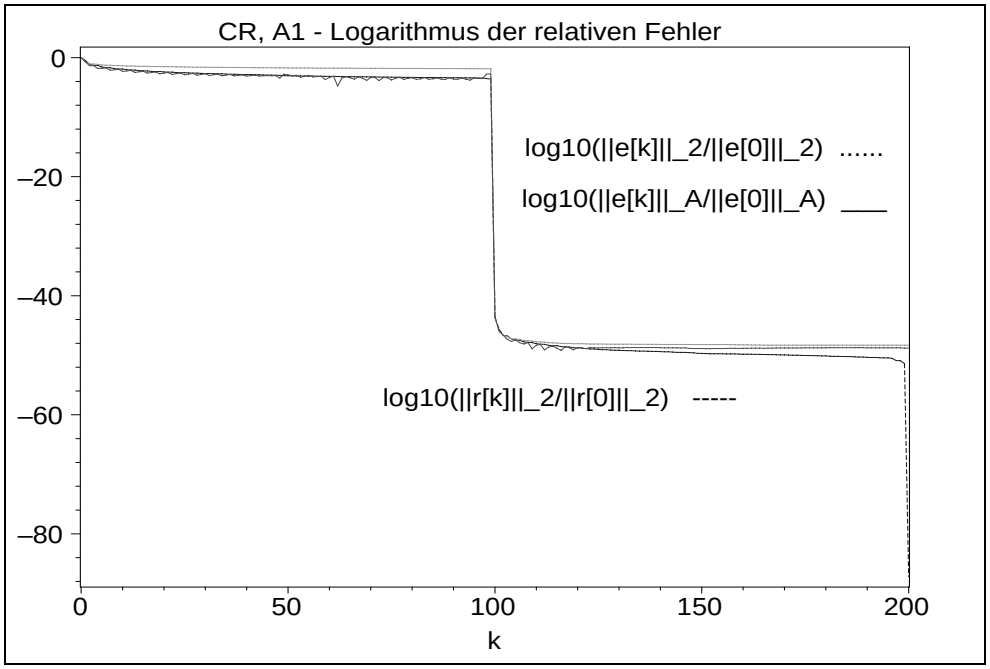


Abb. 6.22 Datei *ccra1cm11.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

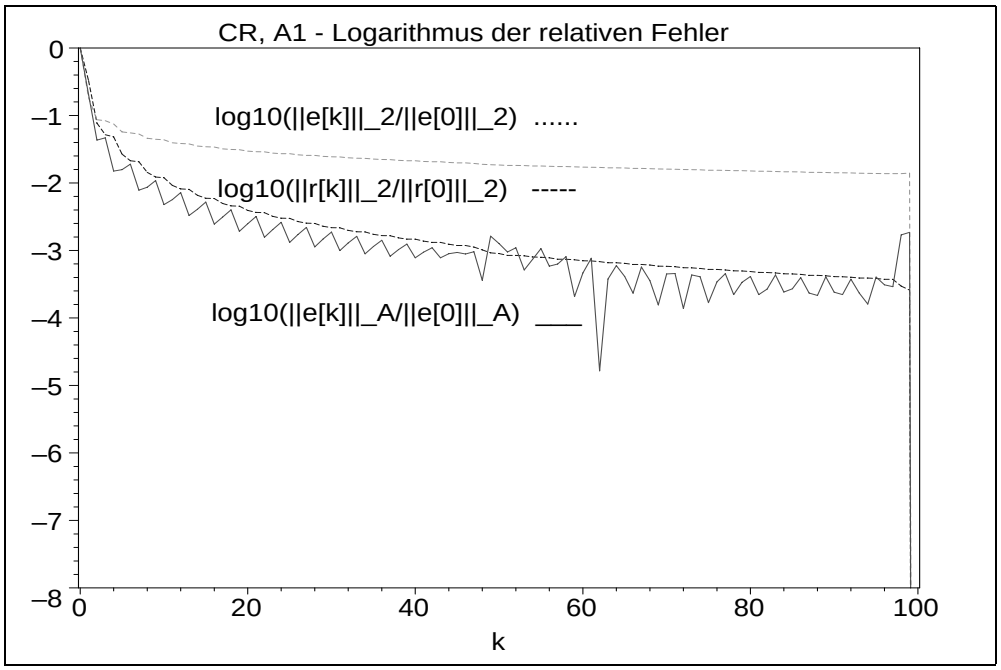


Abb. 6.23 Datei *ccra1cm11a.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

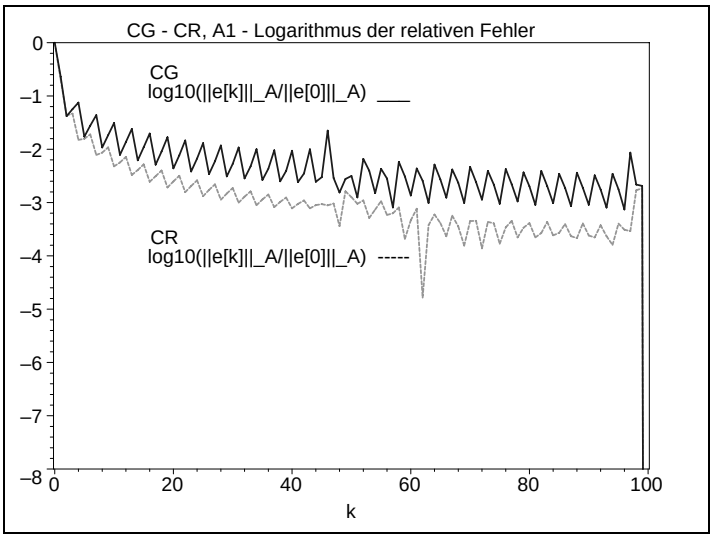


Abb. 6.24
Datei $ccgra1cm111.ps$
Matrix $A1(100,100)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k=0(1)100$

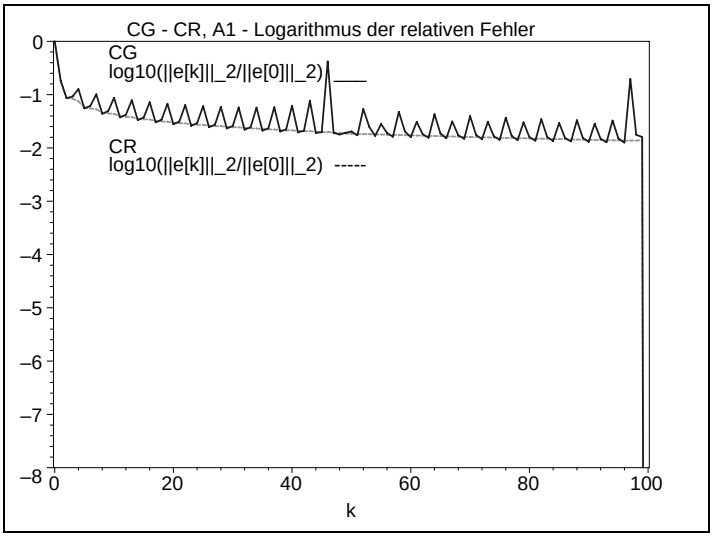


Abb. 6.25
Datei $ccgra1cm112.ps$
Matrix $A1(100,100)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k=0(1)100$

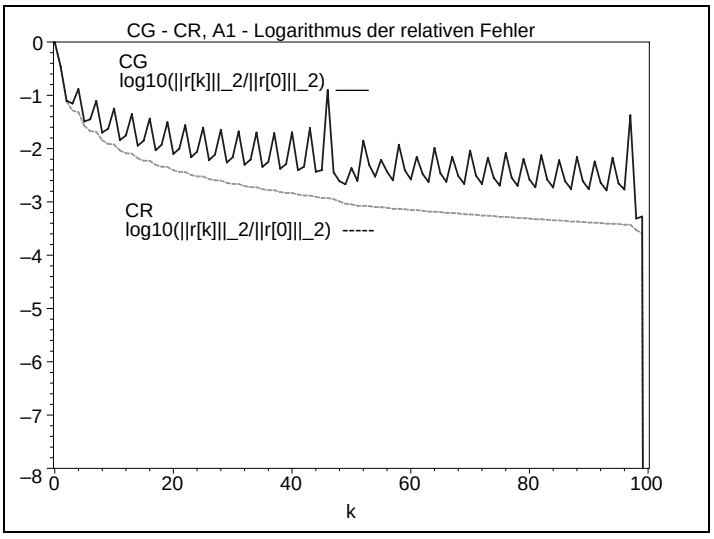


Abb. 6.26
Datei $ccgra1cm113.ps$
Matrix $A1(100,100)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k=0(1)100$

Ergebnisse zu $A2(100,100) = A2(10^2, 10^2)$ blocktridiagonal

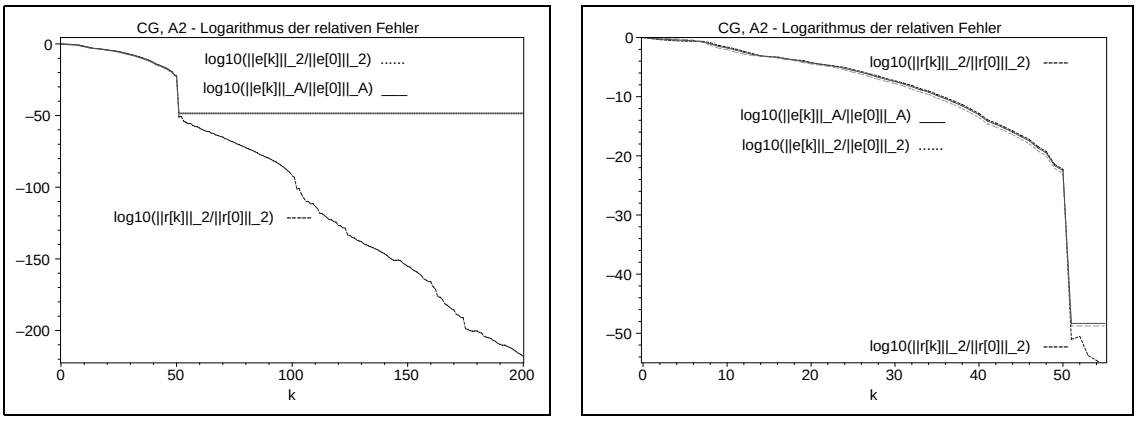


Abb. 6.27 Dateien *ccga2c01.ps*, *ccga2c01a.ps*, Matrix $A2(100,100)$, Shift $c = 0$, CG: Verlauf der 3 relativen Fehler für $k = 0(1)200$ bzw. $k = 0(1)55$

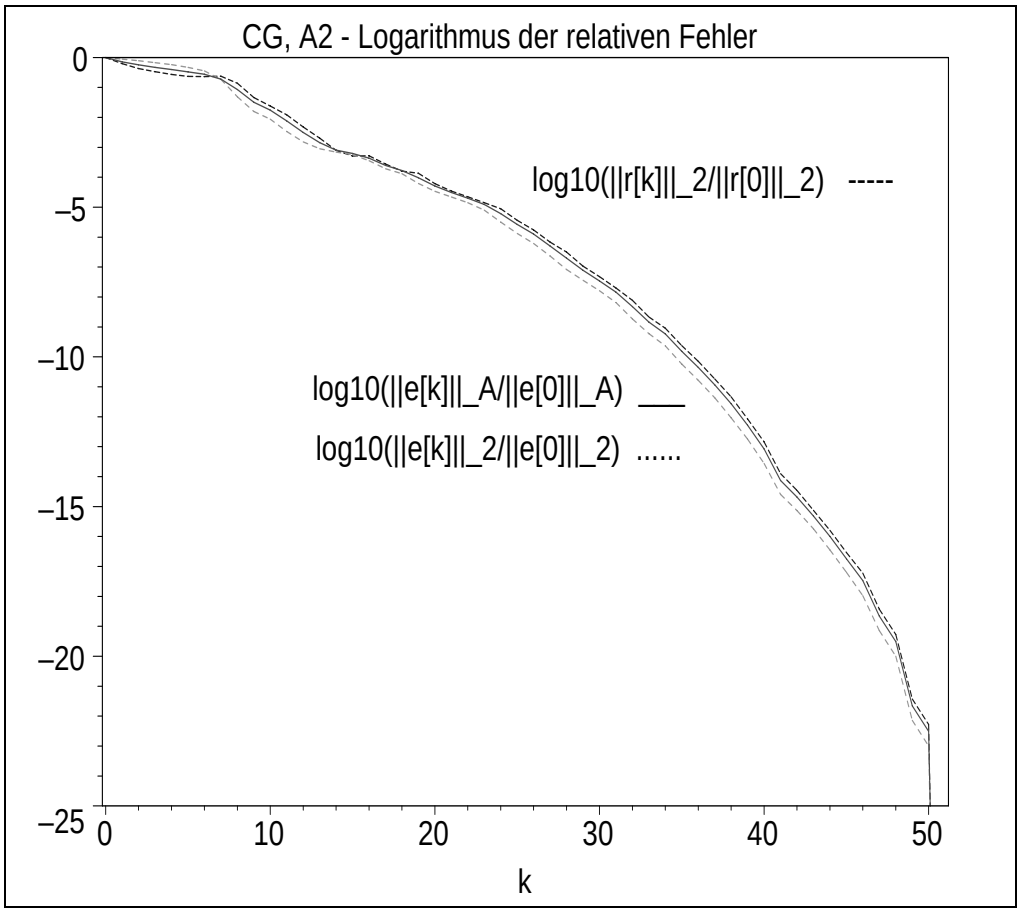


Abb. 6.28 Datei *ccga2c01aa.ps*, Matrix $A2(100,100)$, Shift $c = 0$, CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$ (Zoom)

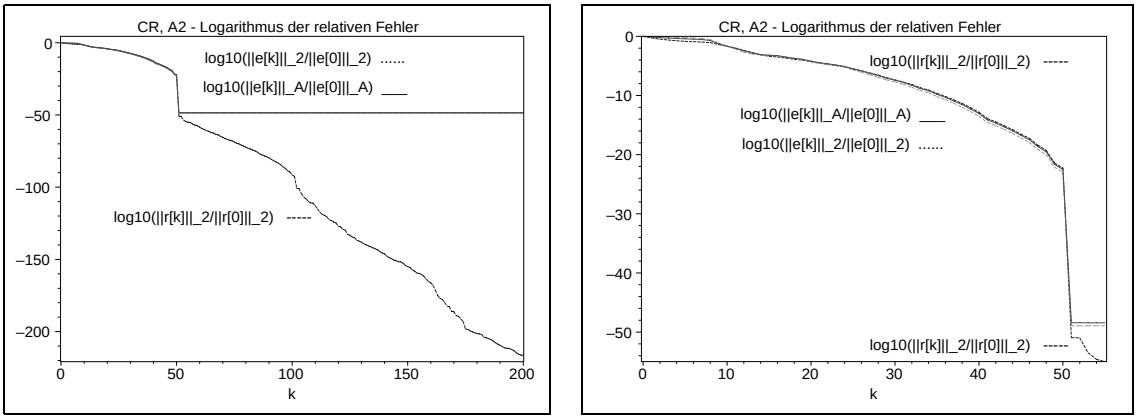


Abb. 6.29 Dateien $ccra2c01.ps$, $ccra2c01a.ps$, Matrix $A2(100, 100)$, Shift $c = 0$, CR: Verlauf der 3 relativen Fehler für $k = 0(1)200$ bzw. $k = 0(1)55$

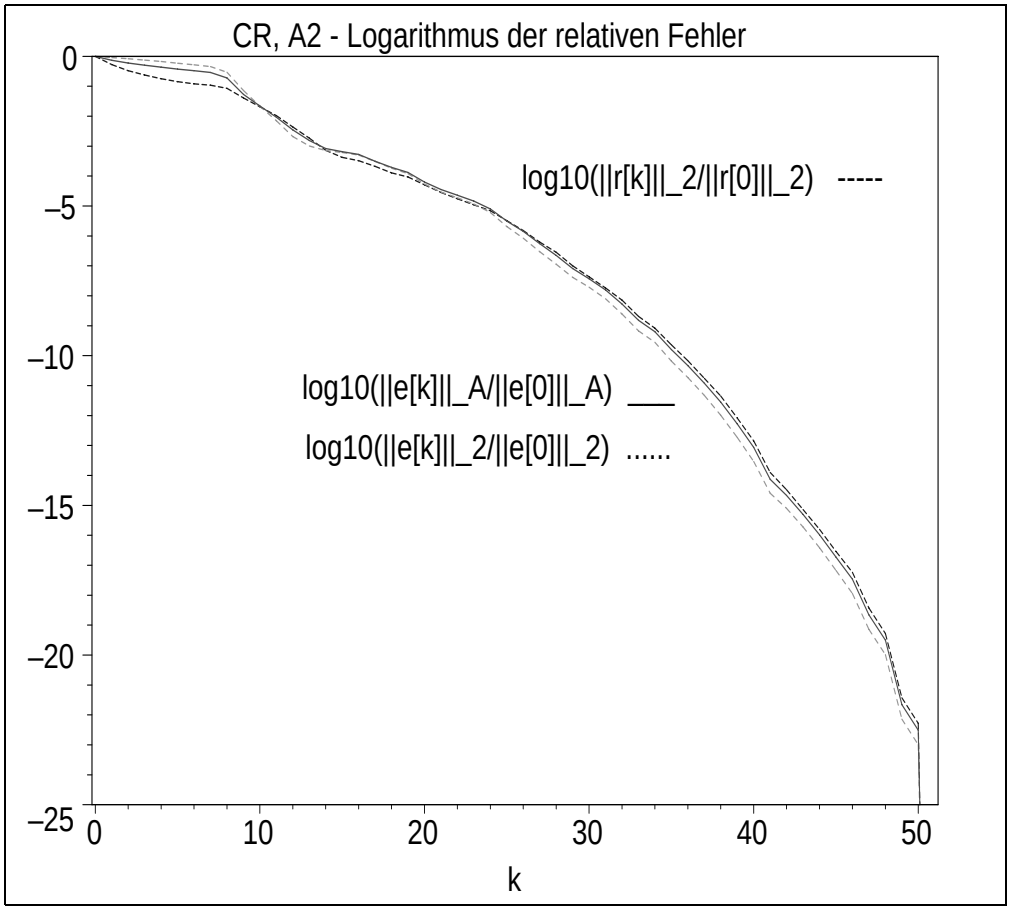


Abb. 6.30 Datei $ccra2c01aa.ps$, Matrix $A2(100, 100)$, Shift $c = 0$, CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$ (Zoom)

Die beiden AV CG und CR sind für die spd Matrix $A2 = A2(100, 100) = A2(10^2, 10^2)$, Shift $c = 0$, theoretisch endlich und brauchen höchstens $n = 100$ Schritte.
Die numerischen Rechnungen in der sehr genauen GPA mit `Digits:=50` widerspiegeln dieses Verhalten. Nach stetigen Genauigkeitszuwächsen findet bei ungefähr $n/2$ eine deutliche Fehlerverkleinerung statt und das AV könnte dort schon beendet werden. Solche "Genauigkeitssprünge" treten in abgeschwächter Form auch bei Vielfachen von $n/2$ auch auf. Alle Fehler tendieren natürlich gegen Null.
Die Konvergenz ist linear. Die spektrale Matrixkondition beträgt $\kappa = 48.374$, womit sich der Quotient

$$q \;=\; \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \;=\; 0.748$$

im Konvergenzfaktor q^k ergibt.

Im Fall A spd erzeugt das CG eine monoton fallende Folge von Werten $\|e^{(k)}\|_A$ und das gilt auch für $\|e^{(k)}\|_2$, während sich dieses Eigenschaft nicht auf die Residuumnorm $\|r^{(k)}\|_2$ überträgt. Unter anderem ist $\|r^{(15)}\|_2 < \|r^{(16)}\|_2$ und $\|r^{(55)}\|_2 < \|r^{(56)}\|_2$.

Das CR erzeugt eine monoton fallende Folge von Werten $\|r^{(k)}\|_2$ und das gilt ebenfalls für die Fehlernorm $\|e^{(k)}\|_2$ sowie meistens auch für $\|e^{(k)}\|_A$.

Ergebnisse aus Berechnungen mit Maple

Startvektor $x^{(0)} = (1, 0, 0, ..., 0)^T$, die rechte Seite b wird so gewählt, dass die exakte Lösung $x^* = (1, 1, ..., 1)^T$ ist, $Q(x^*) = -\frac{1}{2}x^{*T}b = -20,\ R(x^*) = -\frac{1}{2}b^Tb = -24.$

```
CG : Anfaenglicher Iterationsverlauf  mit verschiedenen Fehlern
```

i	Q(x[i])= (||e[i]||^2_A-xs'b)/2	||e[i]||_A=||xs-x[i]||_A =sqrt(2Q(x[i])+xs'b)	||e[i]||_2= ||xs-x[i]||_2	||r[i]||_2= ||b-Ax[i]||_2
0	0.0000000000000000e+00	6.3245553203367587e+00	9.9498743710661995e+00	7.3484692283495343e+00
1	-9.8513513513513514e+00	4.5052521901994895e+00	8.7749435764513632e+00	4.5625672293594894e+00
2	-1.3534340407314537e+01	3.5960143472142774e+00	7.7916154491590559e+00	3.1443314810131379e+00
3	-1.5641769106395488e+01	2.9523654562416597e+00	6.7385350857346332e+00	2.4713248962242881e+00
4	-1.6881449902766298e+01	2.4974187062780251e+00	5.7549797018094259e+00	2.0128868138977892e+00
5	-1.7782031107433942e+01	2.1061666090630428e+00	4.6908090567818488e+00	1.7284519868010718e+00
6	-1.8472331750529316e+01	1.7479520871412259e+00	3.5442005646010258e+00	1.6940679806435351e+00
7	-1.9272356202718886e+01	1.2063530140726756e+00	1.8594203053074264e+00	1.7593421078431320e+00
8	-1.9855980189202056e+01	5.3669322857279186e-01	4.8288438015877369e-01	1.0095700822883297e+00
...				
14	-1.9999987108101611e+01	5.0777747861388104e-03	6.8533618928396889e-03	5.8559794279624553e-03
15	-1.9999992167360939e+01	3.9579386201860088e-03	5.6964174118295831e-03	3.7535504783425540e-03
16	-1.9999996196787741e+01	2.7579747131895826e-03	3.5560352295934717e-03	3.8557885860914568e-03
17	-1.9999998767501060e+01	1.5700311715283447e-03	1.9221139112787468e-03	2.0249917391132715e-03
...				
48	-2.0000000000000000e+01	1.9716985036655688e-19	9.9848163058816136e-20	3.9330537658980067e-19
49	-2.0000000000000000e+01	1.4028708219873577e-21	7.2161703322381503e-22	2.7528771971485625e-21
50	-2.0000000000000000e+01	1.9546453661351455e-22	9.9038432642523064e-23	3.8961808467506797e-22
51	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	6.7258890898546815e-51
52	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	2.1829029696763784e-50
53	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	1.2166805798733452e-53
54	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	1.8707826331324879e-54
55	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	1.7551718856583891e-55
56	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	2.7530120833168704e-55
57	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	1.1649241141506141e-56
58	-2.0000000000000000e+01	2.9320300134889479e-48	1.7097953093864774e-48	1.7748317292946816e-57

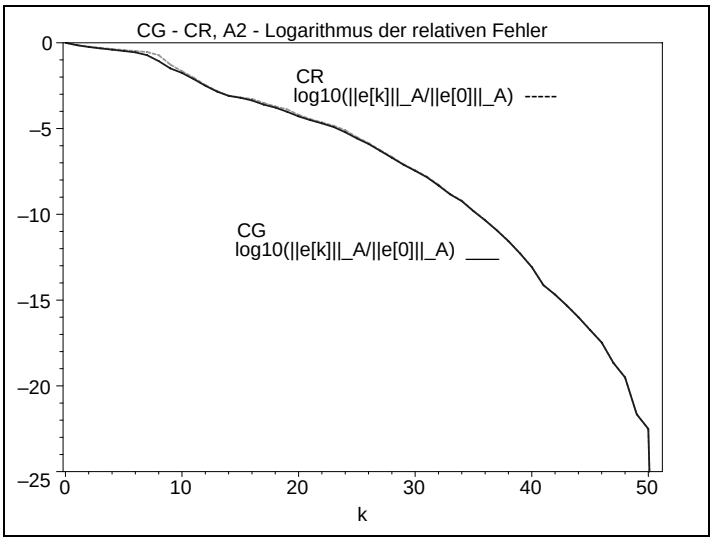


Abb. 6.31
Datei *ccgra2c011.ps*
Matrix $A2(100,100)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)51$

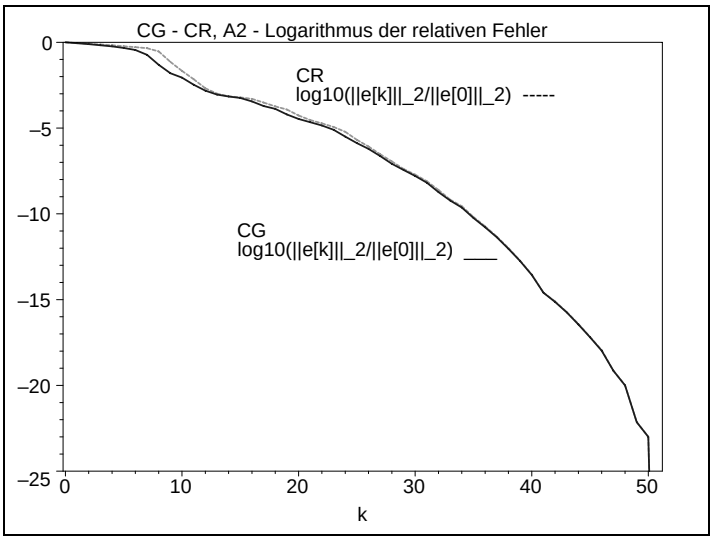


Abb. 6.32
Datei *ccgra2c012.ps*
Matrix $A2(100,100)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)51$

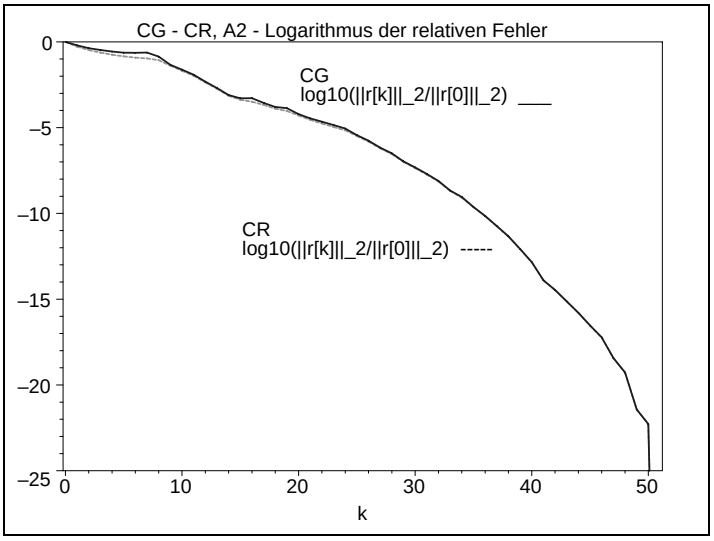


Abb. 6.33
Datei *ccgra2c013.ps*
Matrix $A2(100,100)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)51$

Für die Matrix $A2(100, 100)$ mit Shift $c = 1$ liegen die Fehler bei den ersten 51 Iterationen des CG sehr dicht beieinander. Es gilt $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$. Zusätzlich ist $\|e^{(0)}\|_A = 11.789...$, $\|e^{(0)}\|_2 = 9.949...$, $\|r^{(0)}\|_2 = 15.264...$, so dass sich die relativen Fehler ebensfalls nur minimal unterscheiden.

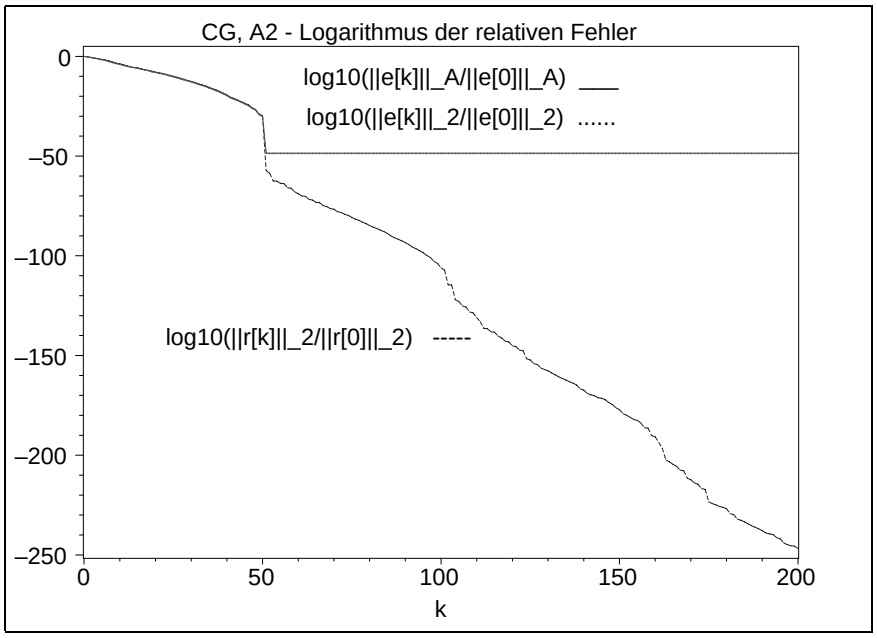


Abb. 6.34 Datei *ccga2c11.ps*, Matrix $A2(100, 100)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

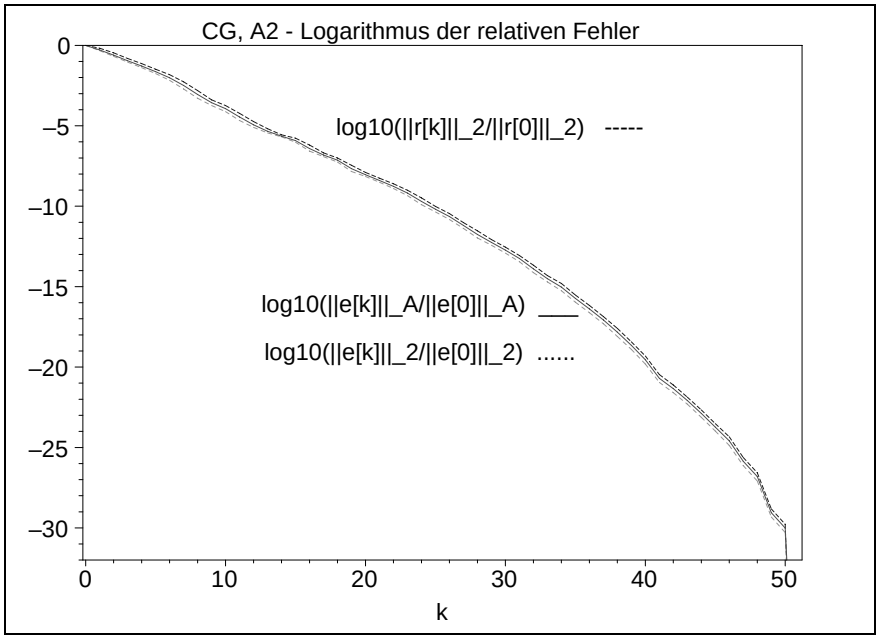


Abb. 6.35 Datei *ccga2c11a.ps*, Matrix $A2(100, 100)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$

Analoges gilt für das CR bei den ersten 51 Iterationen und genauso ist $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$.

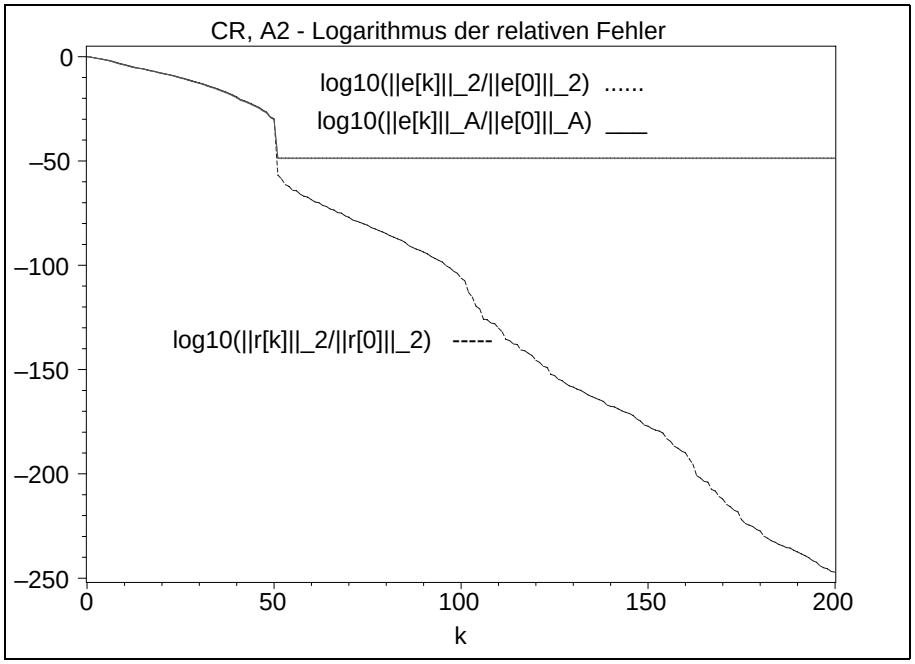


Abb. 6.36 Datei *ccra2c11.ps*, Matrix $A2(100, 100)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

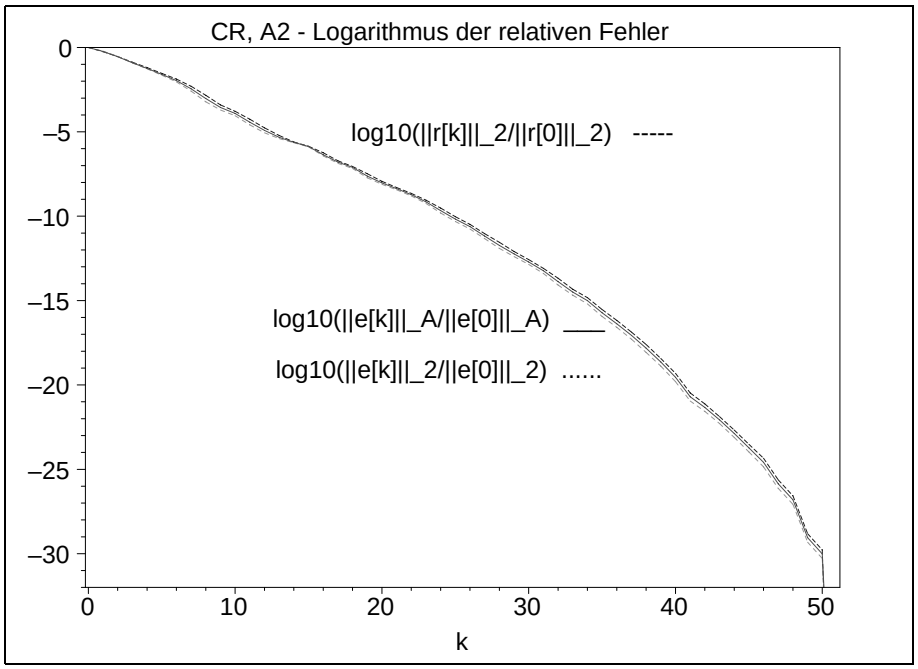


Abb. 6.37 Datei *ccra2c11a.ps*, Matrix $A2(100, 100)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$

Im Vergleich der AV ist bei den Fehlern $\|e^{(k)}\|_A$ und $\|e^{(k)}\|_2$ das CG minimal besser als CR, bei $\|r^{(k)}\|_2$ umgekehrt.

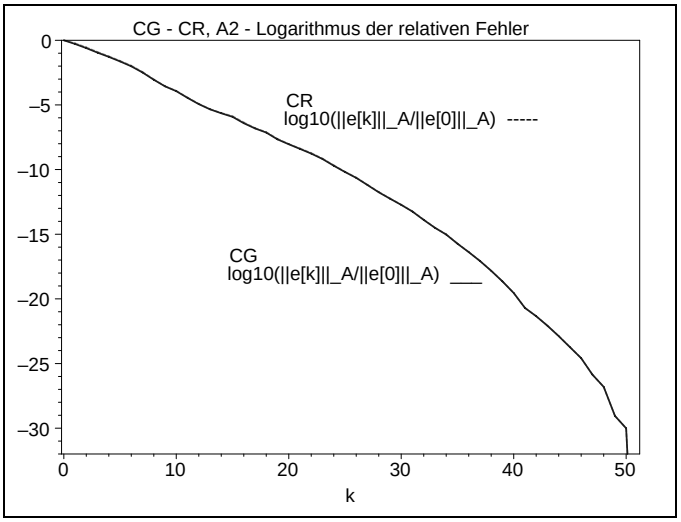


Abb. 6.38
Datei *ccgra2c111.ps*
Matrix $A2(100, 100)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)51$

Ein sehr ähnliches Bild ergibt sich zu CG versus CR für den Verlauf des relativen Fehlers $\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)51$.

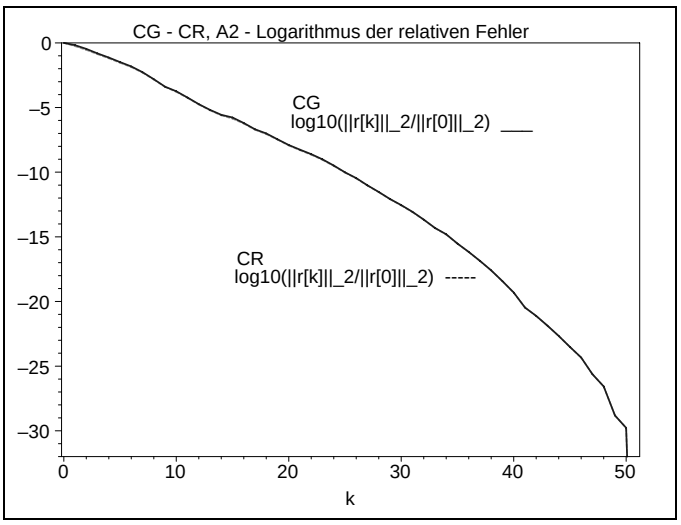


Abb. 6.39
Datei *ccgra2c113.ps*
Matrix $A2(100, 100)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)51$

Mit dem Schift $c = 1$ wird die Diagonaldominanz der Matrix $A2$ verstärkt.
Damit verbessert sich die spektrale Kondition der Matrix auf $\kappa = 7.605$ und man erhält den Quotienten $q = 0.467$ deutlich kleiner als Eins im Konvergenzfaktor q^k. Somit nehmen die Fehler stetig ab.

Für die Matrix $A2(100, 100)$ mit Shift $c = 2$ folgen analoge Aussagen und Ergebnisse wie in der Situation zu $c = 1$. Die Fehler tendieren jedoch noch schneller gegen Null wegen $\kappa = 4.550$ und $q = 0.361$.

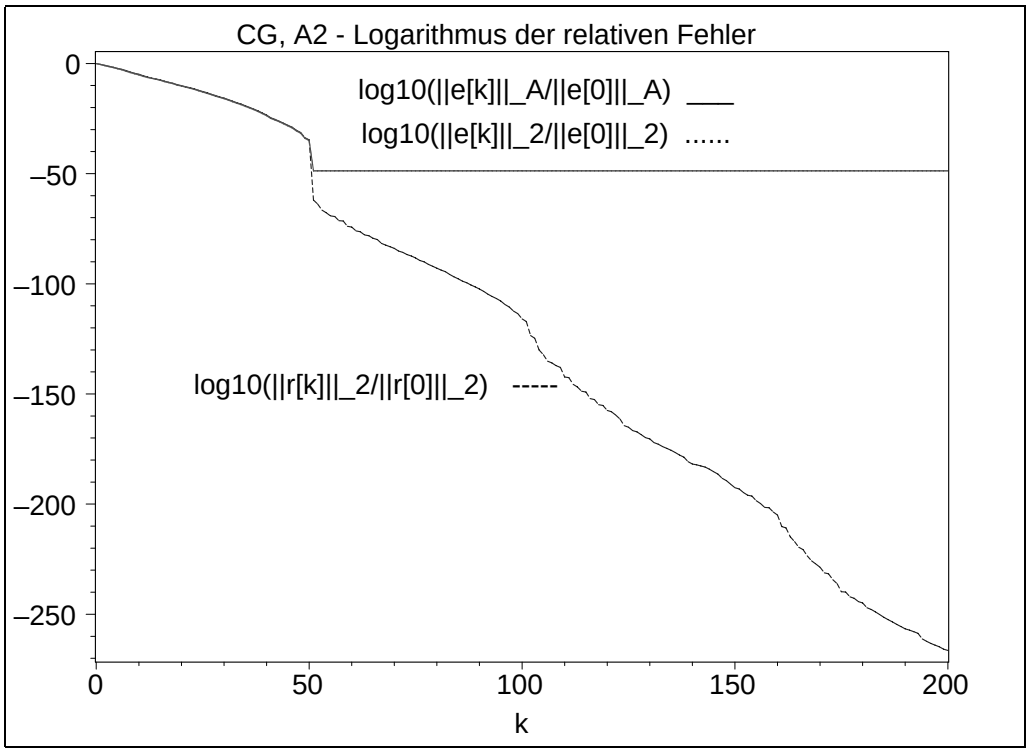


Abb. 6.40 Datei *ccga2c21.ps*, Matrix $A2(100, 100)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

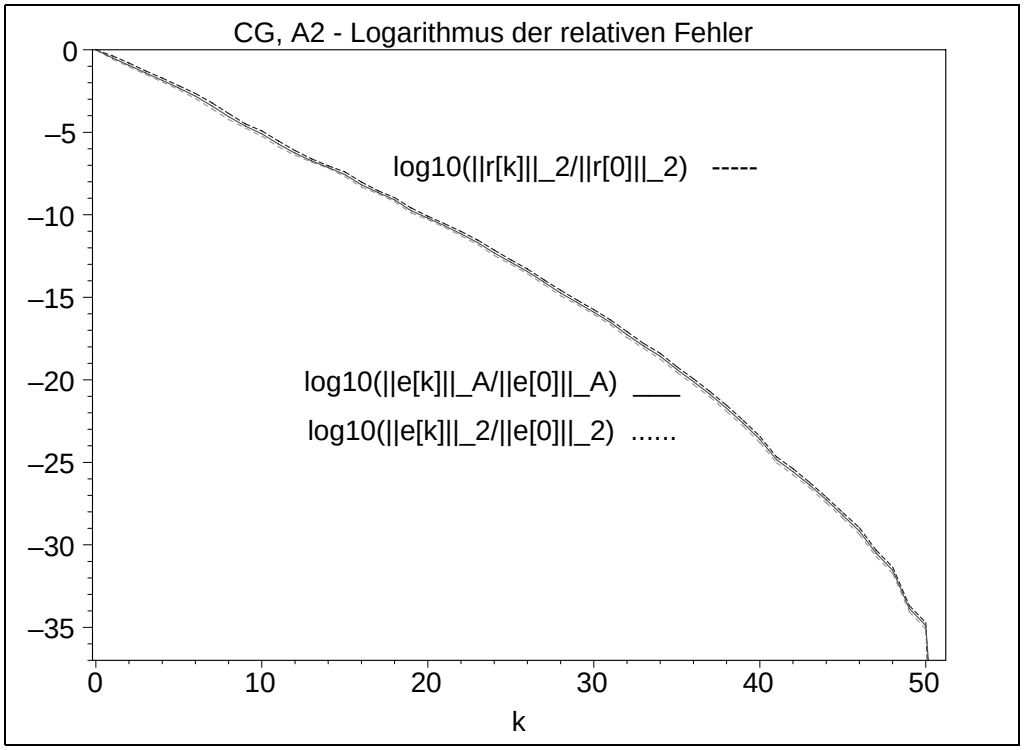


Abb. 6.41 Datei *ccga2c21a.ps*, Matrix $A2(100, 100)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$

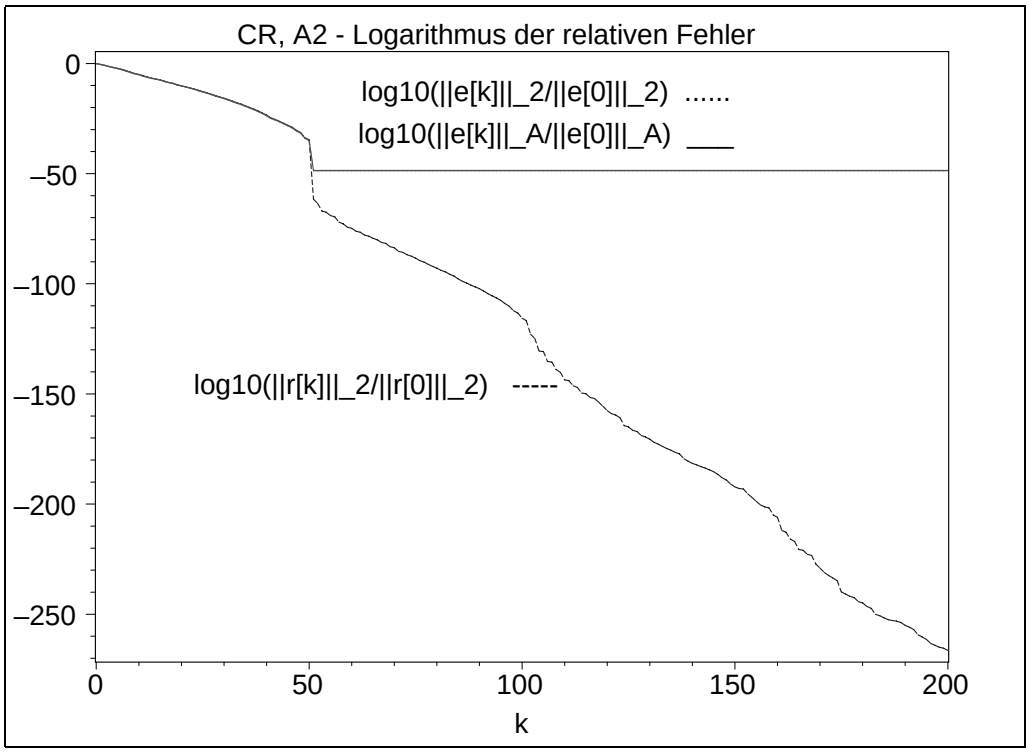


Abb. 6.42 Datei *ccra2c21.ps*, Matrix $A2(100, 100)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

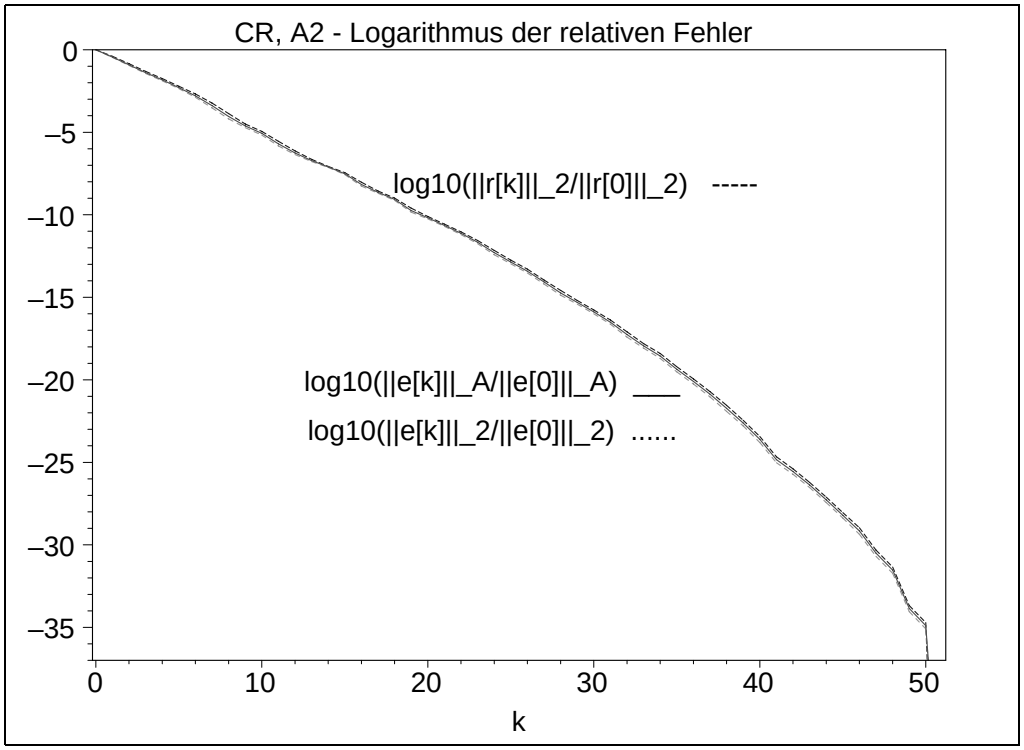


Abb. 6.43 Datei *ccra2c21a.ps*, Matrix $A2(100, 100)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$

Auch hier betrachten wir die symmetrische, aber indefinite Matrix $A2(100, 100)$ mit dem Shift $c = -1$. Glücklicherweise ist die Matrix regulär. Aber die Regularität geht z. B. bei $c = -2$ und $n = 25$ verloren.
Ihre spektrale Konditionszahl ist $\kappa = 879.881$.

Wie bei den anderen Shifts interessieren wir uns für die Fehlerverläufe von CG und CR und hoffen, dass sie nicht vorzeitig abbrechen (durch Nulldivision bei Schrittzahl) bzw. divergieren.
Dabei kann man beim CG kein monoton abnehmendes Verhalten der Fehler erwarten. Aber, wenn es nicht vorzeitig abbricht, ist es nach spätestens n Schritten (theoretisch) an der Lösung. Für das CR erhält man eine stetig abnehmende Folge von Fehlern $\|r^{(k)}\|_2$ und damit auch von $\|e^{(k)}\|_2$ mit spätestens $\|e^{(n)}\|_2 = 0,\ \|r^{(n)}\|_2 = 0$ (theoretisch).

```
CG : Iterationsverlauf  mit verschiedenen Fehlern
 i         Q(x[i])=          ||e[i]||_A=||xs-x[i]||_A    ||e[i]||_2=             ||r[i]||_2=
      (||e[i]||^2_A-xs'b)/2    =sqrt(2Q(x[i])+xs'b)      ||xs-x[i]||_2           ||b-Ax[i]||_2
-------------------------------------------------------------------------------------------------
 0   5.0000000000000000e-01  7.6811457478686082e+00  9.9498743710661995e+00  8.5440037453175312e+00
 1  -3.8014285714285714e+02  2.8640630479891924e+01  9.6274056097948945e+01  1.5895577616277492e+02
 2   4.2263157894736842e+01  4.9524050510306287e+00  5.7401004749572004e+00  8.4784434166940751e+00
 3   3.2082148888596109e+01  2.0406611127750287e+00  4.0982854971145940e+00  3.6388043898694255e+00
 4   2.9769399250935273e+01  6.7911817685102091e-01  3.8766901193628043e+00  2.2560366345056125e+00
 5   2.8546283412690182e+01  1.7051196951005040e+00  4.1830570348801798e+00  1.9313839449966625e+00
...
10   3.0139476569892512e+01  5.2816014596429388e-01  7.9718756675341811e-01  5.5943048326901513e-01
20   2.9995077448375796e+01  9.9222493661509616e-02  3.2109189169362970e-01  2.3617572647323721e-01
30   3.0000039486300362e+01  8.8666529539958949e-03  9.9585167165932137e-02  2.2029733231324989e-03
40   3.0000000000997010e+01  4.4654456934837801e-05  2.7901783583434680e-05  7.4747838840469196e-05
...
48   3.0000000000000000e+01  2.1831126948092119e-10  1.3052936357593784e-10  3.7163403184534419e-10
49   3.0000000000000000e+01  2.0921013955463767e-12  1.2549570468929551e-12  3.5479882685026833e-12
50   3.0000000000000000e+01  3.9137829687061318e-13  2.3410701266209325e-13  6.6572249902600902e-13
51   3.0000000000000000e+01  2.2318442171802916e-38  2.4380868858361903e-38  2.0430492035672758e-38
52   3.0000000000000000e+01  6.8337651910550953e-41  2.6134404793578894e-41  1.7870037703766514e-40
53   3.0000000000000000e+01  4.9425346707508284e-44  6.5463219370964334e-44  4.8569646172067141e-44

CR : Iterationsverlauf  mit verschiedenen Fehlern
 i         Q(x[i])=          ||e[i]||_A=       ||e[i]||_2=    ||r[i]||_2=            R(x[i])=
      (||e[i]||^2_A-xs'b)/2  ||xs-x[i]||_A=    ||xs-x[i]||_2  ||b-Ax[i]||_2=     (||r[i]||^2_2-b'b)/2
                             sqrt(2Q(x[i])+xs'b)              sqrt(2R(x[i])+b'b)
-------------------------------------------------------------------------------------------------
 0   5.0000000000000000e-01  7.681145747e+00  9.949874371e+00  8.544003745e+00  2.5000000000000000e+00
 1  -1.6899740278877857e+00  7.961152432e+00  1.012970643e+01  8.531687967e+00  2.3948497854077253e+00
 2   3.1412023563759688e+01  1.680490145e+00  5.944544637e+00  6.013901238e+00 -1.5916495945984938e+01
 3   3.2034020869888482e+01  2.016938705e+00  4.250535307e+00  3.113269951e+00 -2.9153775105678179e+01
 4   3.0037874940117726e+01  2.752269613e-01  3.910449426e+00  1.826813065e+00 -3.2331377011042964e+01
 5   2.8961805371795838e+01  1.440968166e+00  3.948146615e+00  1.327180912e+00 -3.3119295412182870e+01
...
10   3.0083078239141794e+01  4.076229609e-01  9.751443521e-01  4.643817539e-01 -3.3892174793288975e+01
20   2.9999059348202254e+01  4.337399676e-02  1.346813514e-01  3.285907706e-02 -3.3999460140527293e+01
30   3.0000041069419310e+01  9.063047976e-03  1.009955085e-01  1.356052185e-03 -3.3999999080561234e+01
40   3.0000000004157378e+01  9.118527977e-05  8.943409974e-04  7.198381264e-05 -3.3999999997409165e+01
...
48   3.0000000000000000e+01  2.218575047e-10  1.367047224e-10  3.651439934e-10 -3.4000000000000000e+01
49   3.0000000000000000e+01  2.092206223e-12  1.255144912e-12  3.547820790e-12 -3.4000000000000000e+01
50   3.0000000000000000e+01  3.975721247e-13  2.448326808e-13  6.543032159e-13 -3.4000000000000000e+01
51   3.0000000000000000e+01  1.193458995e-38  1.303745519e-38  1.092503055e-38 -3.4000000000000000e+01
52   3.0000000000000000e+01  5.089727710e-41  1.954999014e-41  1.331658975e-40 -3.4000000000000000e+01
53   3.0000000000000000e+01  9.298591001e-44  1.215119994e-43  8.208477763e-44 -3.4000000000000000e+01
```

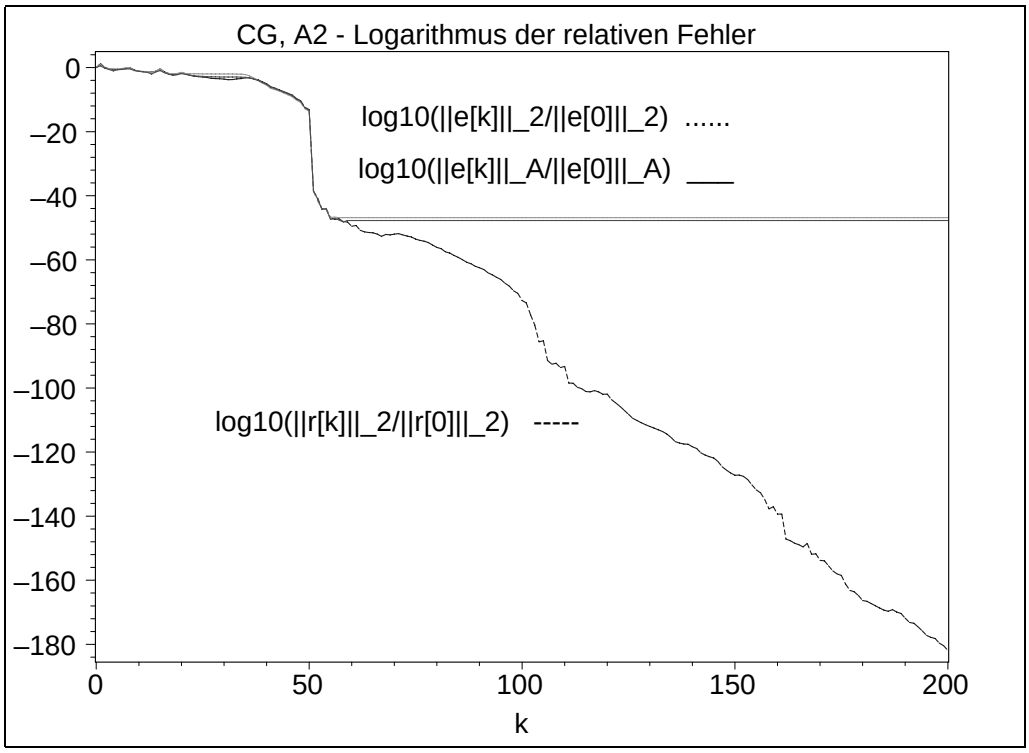


Abb. 6.44 Datei *ccga2cm11.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

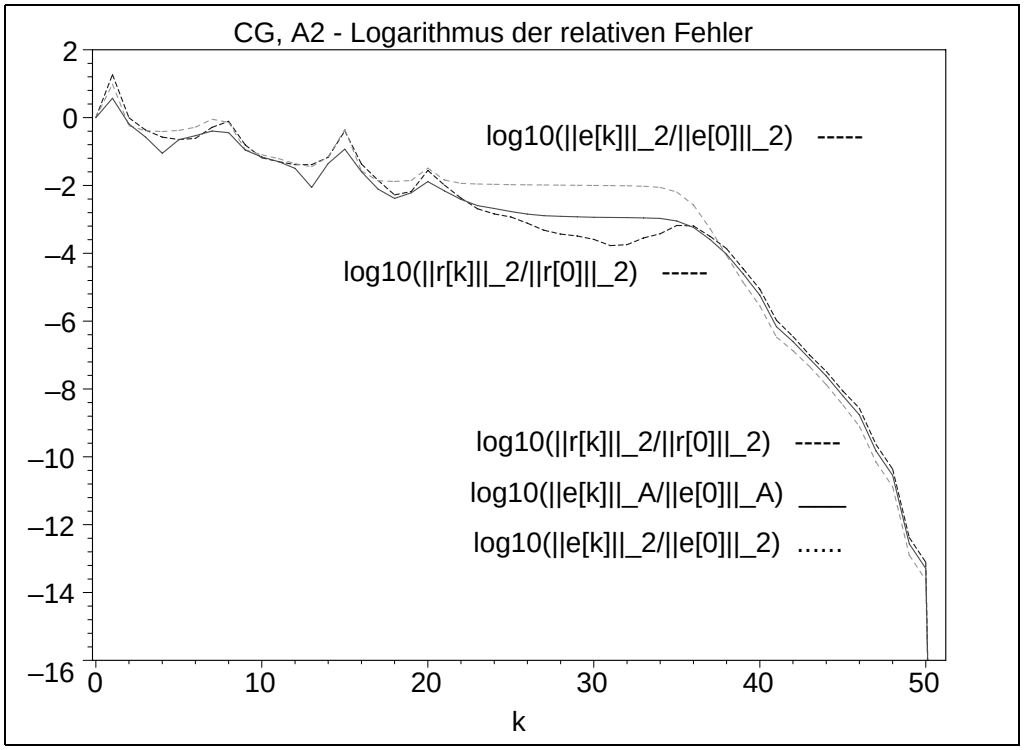


Abb. 6.45 Datei *ccga2cm11a.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$

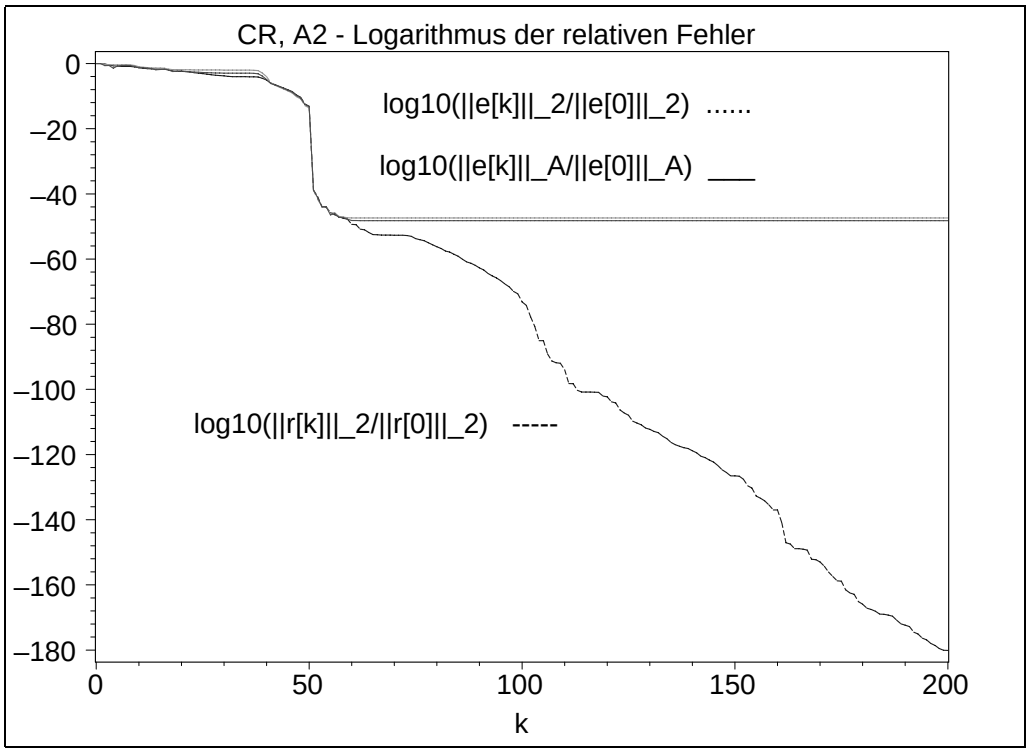


Abb. 6.46 Datei *ccra2cm11.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

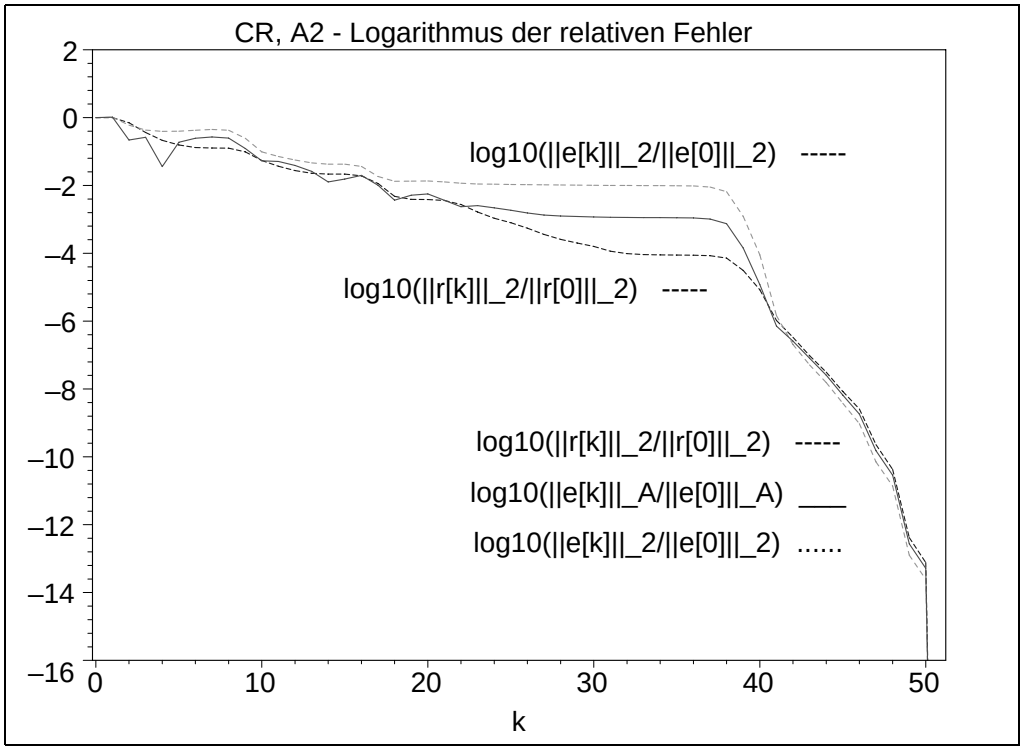


Abb. 6.47 Datei *ccra2cm11a.ps*, Matrix $A1(100, 100)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)51$

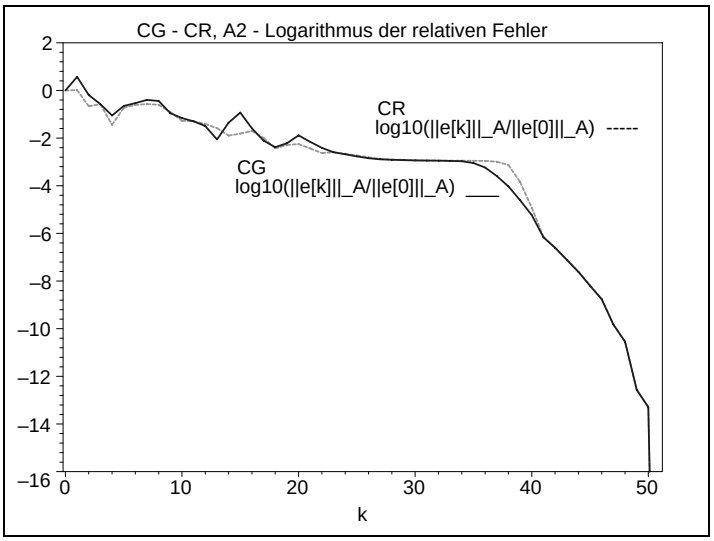


Abb. 6.48
Datei *ccgra2cm111.ps*
Matrix $A1(100,100)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k=0(1)51$

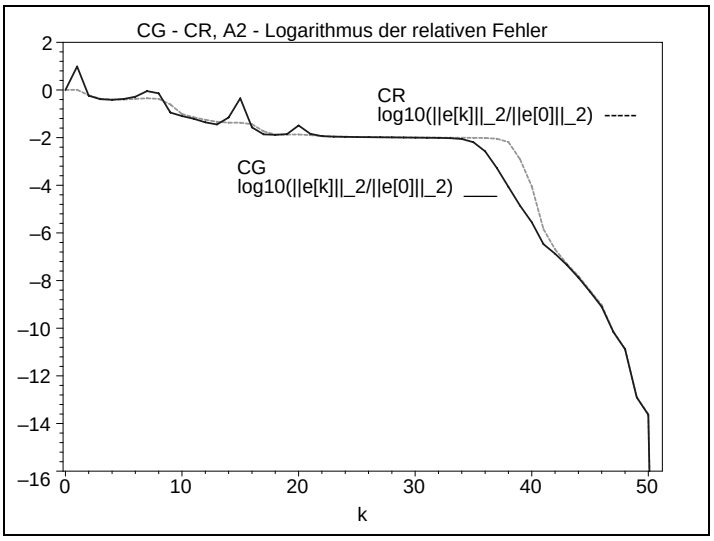


Abb. 6.49
Datei *ccgra2cm112.ps*
Matrix $A1(100,100)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k=0(1)51$

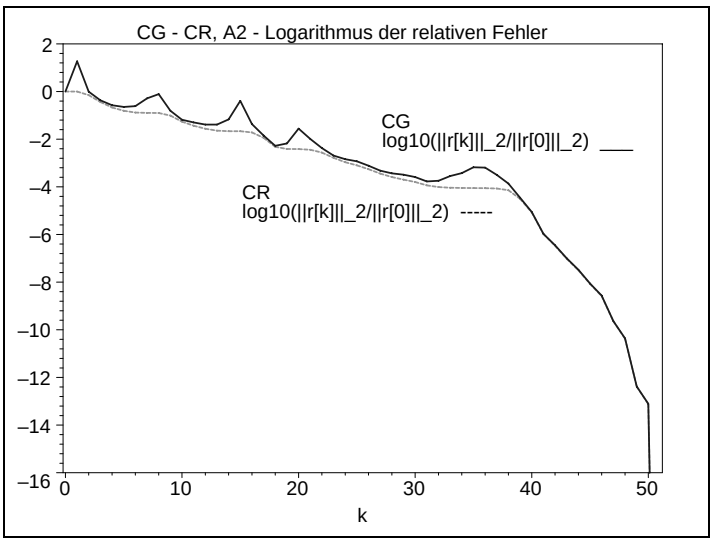


Abb. 6.50
Datei *ccgra2cm113.ps*
Matrix $A1(100,100)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k=0(1)51$

Ergebnisse zu $A1(625, 625)$ tridiagonal

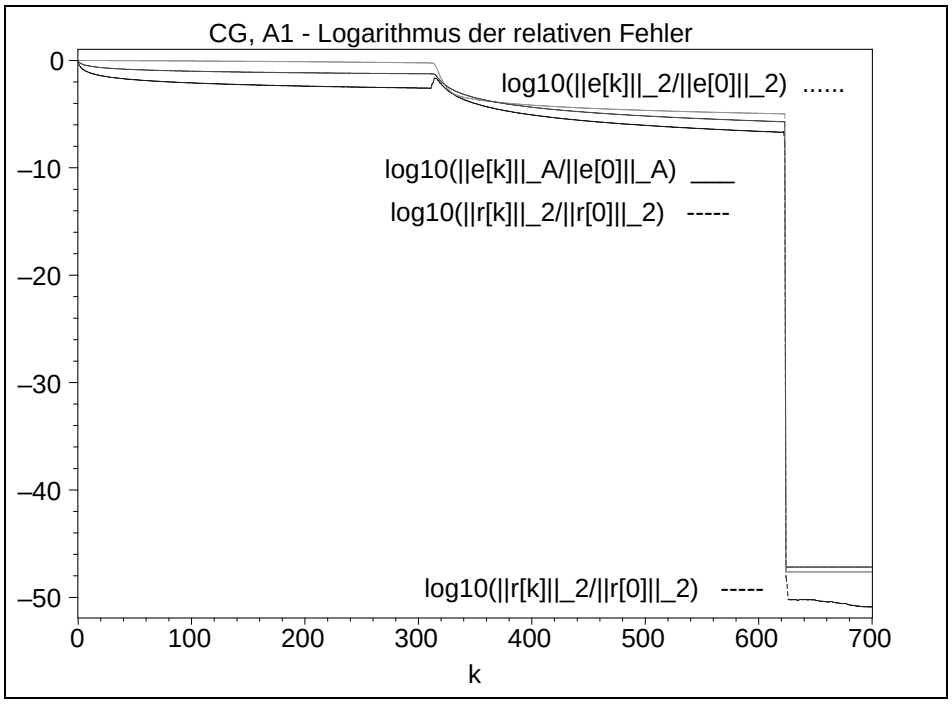


Abb. 6.51 Datei *ccga1c06.ps*, Matrix $A1(625, 625)$, Shift $c = 0$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

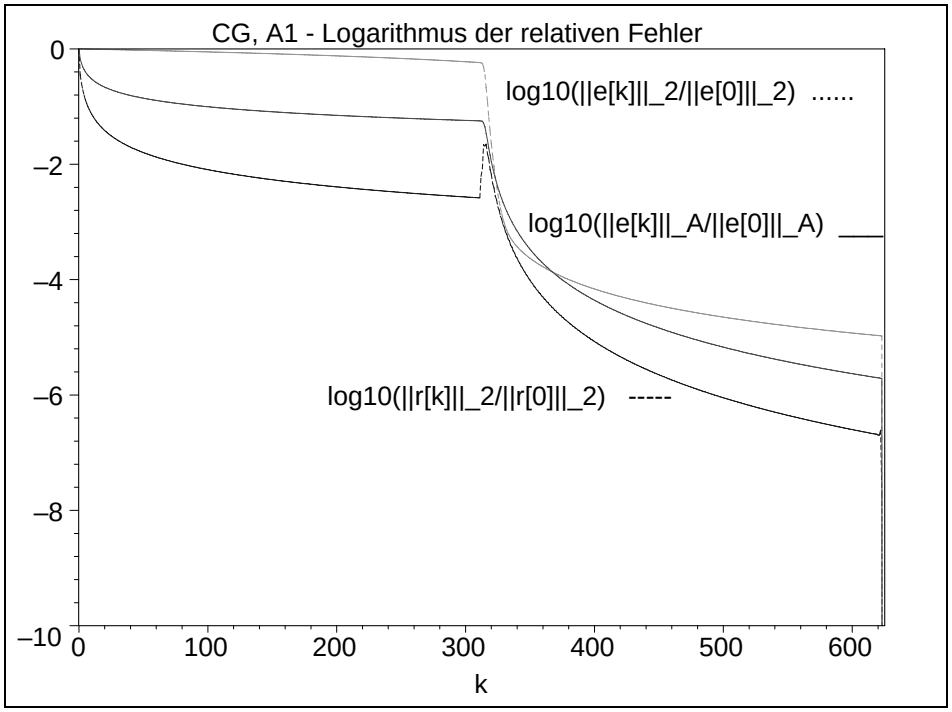


Abb. 6.52 Datei *ccga1c06a.ps*, Matrix $A1(625, 625)$, Shift $c = 0$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)625$

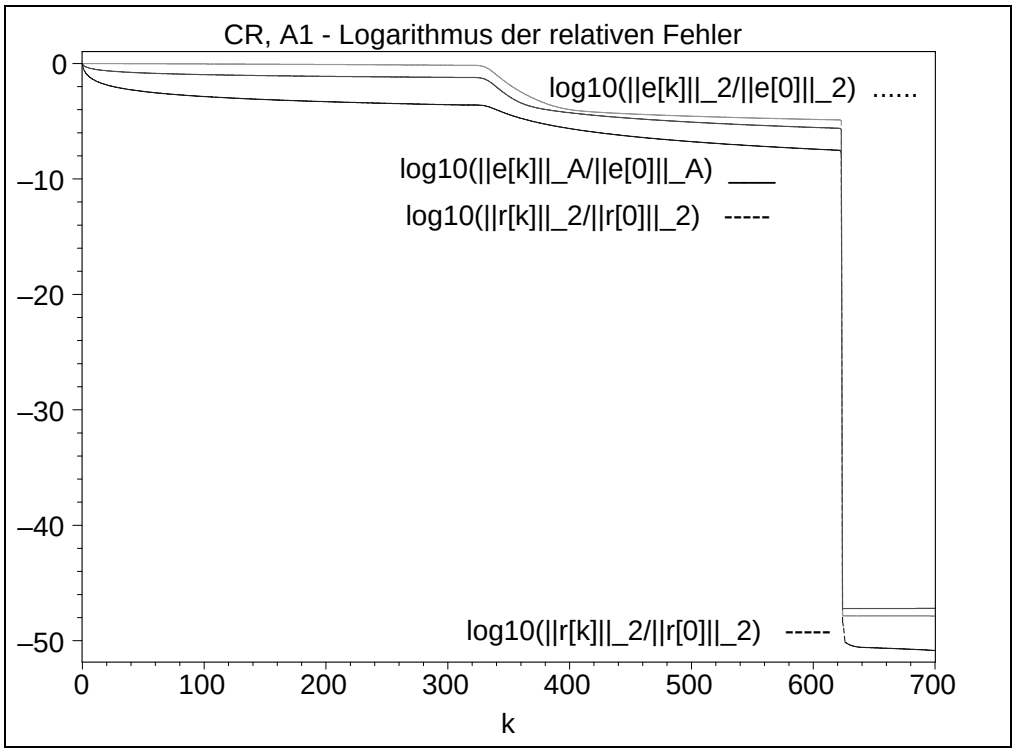


Abb. 6.53 Datei *ccra1c06.ps*, Matrix $A1(625, 625)$, Shift $c = 0$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

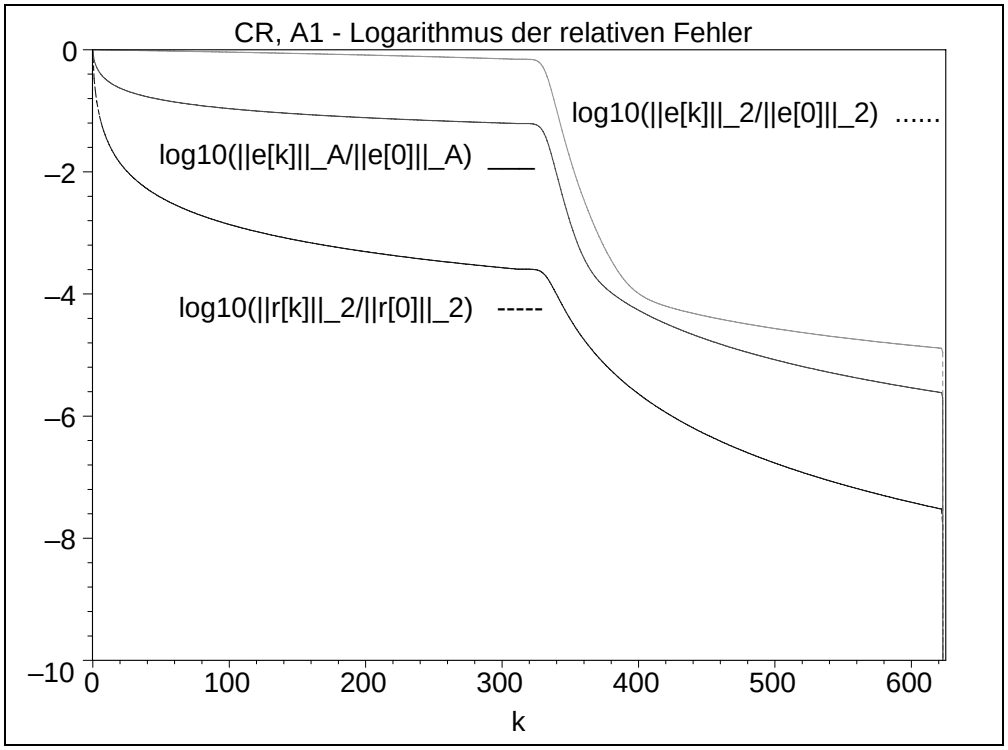


Abb. 6.54 Datei *ccra1c06a.ps*, Matrix $A1(625, 625)$, Shift $c = 0$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)625$

Die beiden AV CG und CR sind für die spd Matrix $A1 = A1(625, 625)$, Shift $c = 0$, theoretisch endlich und brauchen höchstens $n = 625$ Schritte.
Die numerischen Rechnungen in der sehr genauen GPA mit `Digits:=50` widerspiegeln dieses Verhalten, wobei bei $n/2$ sichtbare Verbesserungen und erst kurz vor bzw. bei n die deutliche Fehlerverkleinerung stattfinden. Dazwischen liegen i. Allg. nur kleine Genauigkeitszuwächse. Alle Fehler tendieren natürlich gegen Null. Die Konvergenz ist langsam, weil die spektrale Kondition der Matrix $\kappa = 158\,820$ groß ist und damit der Quotient

$$q \;=\; \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \;=\; 0.995$$

im Konvergenzfaktor q^k nahe der Eins liegt.
Im Fall A spd erzeugt das CG eine monoton fallende Folge von Werten $\|e^{(k)}\|_A$ und das gilt auch für $\|e^{(k)}\|_2$, während sich dieses Eigenschaft nicht auf die Residuumnorm $\|r^{(k)}\|_2$ überträgt. Das CR erzeugt eine monoton fallende Folge von Werten $\|r^{(k)}\|_2$ und das gilt ebenfalls für die Fehlernorm $\|e^{(k)}\|_2$ sowie meistens auch für $\|e^{(k)}\|_A$.

Ergebnisse aus Berechnungen mit Maple

Startvektor $x^{(0)} = (1, 0, 0, ..., 0)^T$, die rechte Seite b wird so gewählt, dass die exakte Lösung $x^* = (1, 1, ..., 1)^T$ ist, $Q(x^*) = -\frac{1}{2}x^{*T}b = -1,\ R(x^*) = -\frac{1}{2}b^Tb = -1$.

```
CG : Iterationsverlauf  mit verschiedenen Fehlern
```

i	Q(x[i])= (||e[i]||^2_A-xs'b)/2	||e[i]||_A=||xs-x[i]||_A =sqrt(2Q(x[i])+xs'b)	||e[i]||_2= ||xs-x[i]||_2	||r[i]||_2= ||b-Ax[i]||_2
0	0.0000000000000000e+00	1.4142135623730950e+00	2.4979991993593593e+01	1.7320508075688773e+00
1	-5.6250000000000000e-01	9.3541434669348535e-01	2.4958402893614808e+01	6.1237243569579452e-01
2	-6.8750000000000000e-01	7.9056941504209483e-01	2.4935855208915535e+01	4.3301270189221932e-01
3	-7.6250000000000000e-01	6.8920243760451109e-01	2.4908269209240533e+01	3.3911649915626341e-01
4	-8.0831408775981524e-01	6.1917027099205071e-01	2.4881369084282329e+01	2.7062205477269659e-01
5	-8.3904569892473118e-01	5.6736989887597812e-01	2.4854626087762439e+01	2.2752799967203039e-01
...				
50	-9.8047403140162186e-01	1.9761562994043835e-01	2.3616119825056233e+01	2.7613889610297536e-02
100	-9.9011992910444232e-01	1.4057077146802377e-01	2.2159680237403487e+01	1.3972530257705367e-02
150	-9.9338684909410535e-01	1.1500565991197694e-01	2.0600575466479634e+01	9.3524077011361339e-03
200	-9.9503016314807738e-01	9.9697912234134740e-02	1.8913394940381473e+01	7.0284106787705719e-03
250	-9.9601932773418185e-01	8.9226366796123102e-02	1.7060173441576617e+01	5.6295207056824653e-03
300	-9.9668009952207039e-01	8.1484973804126720e-02	1.4979407005696002e+01	4.6950482816169730e-03
...				
310	-9.9678677538153574e-01	8.0165137291272221e-02	1.4527534747109462e+01	4.5441858343832732e-03
311	-9.9679706712430322e-01	8.0036652549900968e-02	1.4481572052434819e+01	4.5296311121810513e-03
312	-9.9683003507991338e-01	7.9623676379411393e-02	1.4332921358348564e+01	1.0703343471549254e-02
313	-9.9689464682548556e-01	7.8808034799942100e-02	1.4041099065034026e+01	1.4662776395532011e-02
314	-9.9734488983631175e-01	7.2871258582355316e-02	1.2005760660798753e+01	3.7797157547589645e-02
315	-9.9802241639057232e-01	6.2890120200675061e-02	8.9427058784863755e+00	3.5925040403972673e-02
316	-9.9882543425724348e-01	4.8467839703385264e-02	5.3120494745007032e+00	3.8776720661673730e-02
317	-9.9933532101634758e-01	3.6460361590429118e-02	3.0067891232432504e+00	2.5975356127762015e-02
318	-9.9963520024801326e-01	2.7011099643914543e-02	1.6510567315133995e+00	2.1678771827166718e-02
319	-9.9979062726190673e-01	2.0463271395027264e-02	9.4857941989078136e-01	1.4138063335461643e-02
320	-9.9987554783123363e-01	1.5776702365600175e-02	5.6491527409261623e-01	1.1626203235418421e-02
321	-9.9992269965145553e-01	1.2433852865823566e-02	3.5212199663935241e-01	8.0015738755703265e-03
...				

```
350  -9.9999988686840441e-01  4.7567130582143226e-04  5.8632185423773974e-03  1.7072647238247651e-04
400  -9.9999999807100149e-01  6.2112776601942474e-05  1.7210875369844417e-03  1.4737668408794171e-05
450  -9.9999999978917299e-01  2.0534215861292510e-05  8.8582582555243672e-04  3.8971579785263003e-06
500  -9.9999999995430365e-01  9.5599524201598944e-06  5.5991980755027226e-04  1.5556948138467165e-06
550  -9.9999999998580732e-01  5.3277913828247098e-06  3.9425460436191958e-04  7.7090834726128035e-07
600  -9.9999999999449463e-01  3.3182438887449411e-06  2.9675042535530738e-04  4.3660070588295646e-07
...
620  -9.9999999999605753e-01  2.8080125809096539e-06  2.6846407072757747e-04  3.5010132371988668e-07
621  -9.9999999999611532e-01  2.7873572038202198e-06  2.6727600737724298e-04  3.4973463338182752e-07
622  -9.9999999999619454e-01  2.7587899526274476e-06  2.6562993422529302e-04  4.2683019154222266e-07
623  -9.9999999999625457e-01  2.7369453019123751e-06  2.6437978095647279e-04  3.3706702424132511e-08
-------------------------------------------------------------------------------------------------
624  -1.0000000000000000e+00  9.3286762190570212e-48  5.7626091312876670e-47  1.3667411994497584e-48
-------------------------------------------------------------------------------------------------
625  -1.0000000000000000e+00  9.3207617714433620e-48  5.7625987193279388e-47  2.9172268680913105e-49
-------------------------------------------------------------------------------------------------
626          "                9.3222958545628662e-48  5.7625374619172759e-47  1.1749892383590725e-50
627          "                        "                       "               1.0293642597257787e-50
628          "                        "                       "               1.0532005433767769e-50
629          "                        "                       "               1.0243493802605785e-50
630  -1.0000000000000000e+00  9.3222958545628662e-48  5.7625374619172759e-47  1.0470717137190562e-50


CR : Iterationsverlauf  mit verschiedenen Fehlern

 i         Q(x[i])=          ||e[i]||_A=        ||e[i]||_2=    ||r[i]||_2=            R(x[i])=
      (||e[i]||^2_A-xs'b)/2  ||xs-x[i]||_A=     ||xs-x[i]||_2  ||b-Ax[i]||_2=     (||r[i]||^2_2-b'b)/2
                             sqrt(2Q(x[i])+xs'b)               sqrt(2R(x[i])+b'b)
-------------------------------------------------------------------------------------------------
  0   0.0000000000000000e+00  1.414213562e+00  2.497999199e+01  1.732050807e+00  5.0000000000000000e-01
  1  -5.5555555555555556e-01  9.428090415e-01  2.495996794e+01  5.773502691e-01 -8.3333333333333333e-01
  2  -6.7040000000000000e-01  8.119113252e-01  2.494355227e+01  3.464101615e-01 -9.4000000000000000e-01
  3  -7.4044436396559529e-01  7.204937696e-01  2.492403440e+01  2.423291238e-01 -9.7063829787234043e-01
  4  -7.8740927960223195e-01  6.520593844e-01  2.490330839e+01  1.805294829e-01 -9.8370455289878808e-01
  5  -8.1960000000000000e-01  6.006662967e-01  2.488269132e+01  1.414213562e-01 -9.9000000000000000e-01
...
 50  -9.7679113153830796e-01  2.154477591e-01  2.393368092e+01  6.586993754e-03 -9.9997830575664192e-01
100  -9.8820163206978180e-01  1.536122907e-01  2.283389100e+01  2.387871509e-03 -9.9999714903482679e-01
150  -9.9209020203535103e-01  1.257759751e-01  2.167846622e+01  1.310811399e-03 -9.9999914088673792e-01
200  -9.9405090587858027e-01  1.090788166e-01  2.045791401e+01  8.550143686e-04 -9.9999963447521470e-01
250  -9.9523264437929257e-01  9.764584600e-02  1.915977604e+01  6.133605379e-04 -9.9999981189442527e-01
300  -9.9602269973963768e-01  8.918856720e-02  1.776704829e+01  4.673942471e-04 -9.9999989077130886e-01
350  -9.9999780204179465e-01  2.096644082e-03  4.022210473e-01  6.703940441e-05 -9.9999999775285913e-01
400  -9.9999999705238811e-01  7.678036067e-05  2.545158776e-03  4.015185115e-06 -9.9999999999193914e-01
450  -9.9999999967767925e-01  2.538979114e-05  1.075945661e-03  8.583931774e-07 -9.9999999999963158e-01
500  -9.9999999992983162e-01  1.184638201e-05  6.796798429e-04  2.952989974e-07 -9.9999999999995640e-01
550  -9.9999999997814861e-01  6.610808249e-06  4.789329222e-04  1.305009176e-07 -9.9999999999999148e-01
600  -9.9999999999150889e-01  4.120947747e-06  3.606760058e-04  6.733794279e-08 -9.9999999999999773e-01
...
620  -9.9999999999391740e-01  3.487864149e-06  3.263281315e-04  5.331517602e-08 -9.9999999999999858e-01
621  -9.9999999999401610e-01  3.459449534e-06  3.247295062e-04  5.270626313e-08 -9.9999999999999861e-01
622  -9.9999999999408105e-01  3.440625223e-06  3.236699973e-04  5.230896714e-08 -9.9999999999999863e-01
623  -9.9999999999606746e-01  2.804473215e-06  2.789845127e-04  2.833374609e-08 -9.9999999999999960e-01
-------------------------------------------------------------------------------------------------
624  -1.0000000000000000e+00  8.817369222e-48  3.546671115e-47  8.248602403e-49 -1.0000000000000000e+00
-------------------------------------------------------------------------------------------------
625  -1.0000000000000000e+00  8.792792502e-48  3.546669141e-47  2.125523010e-49 -1.0000000000000000e+00
-------------------------------------------------------------------------------------------------
626          "                8.785727061e-48  3.546535492e-47  1.396040350e-50         "
627          "                      "                "          1.126377065e-50         "
628          "                      "                "          9.806332668e-51         "
629          "                      "                "          8.519348106e-51         "
630  -1.0000000000000000e+00  8.785727061e-48  3.546535492e-47  7.674959401e-51 -1.0000000000000000e+00
```

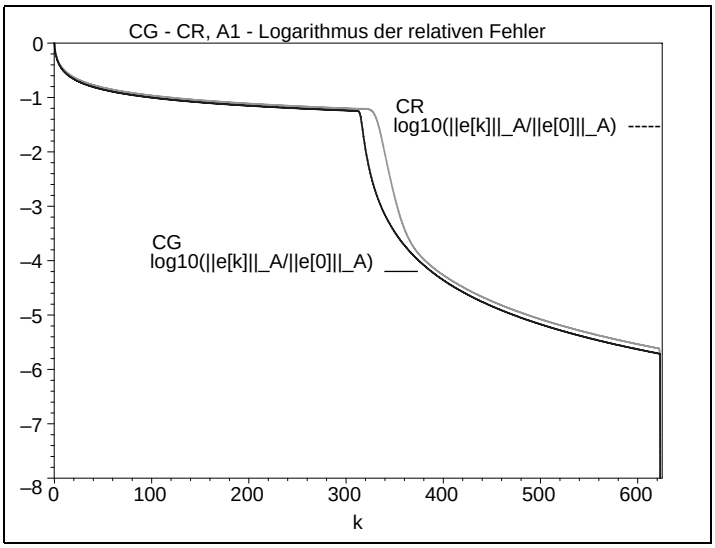


Abb. 6.55
Datei *ccgra1c061.ps*
Matrix $A1(625, 625)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)625$

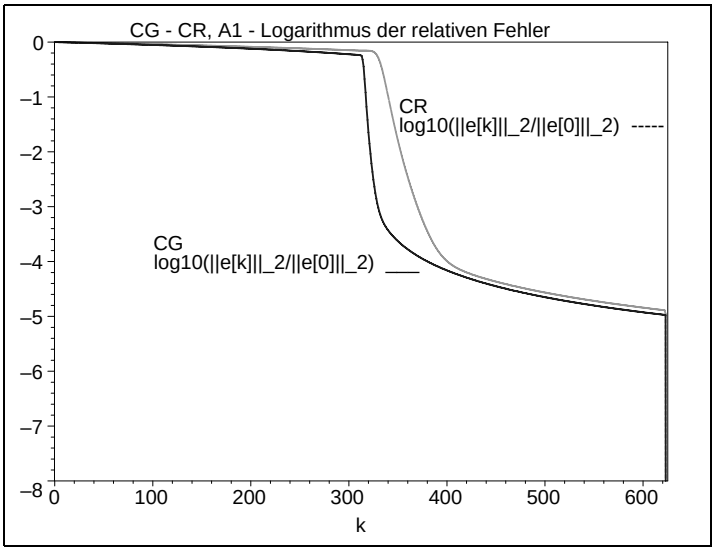


Abb. 6.56
Datei *ccgra1c062.ps*
Matrix $A1(625, 625)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)625$

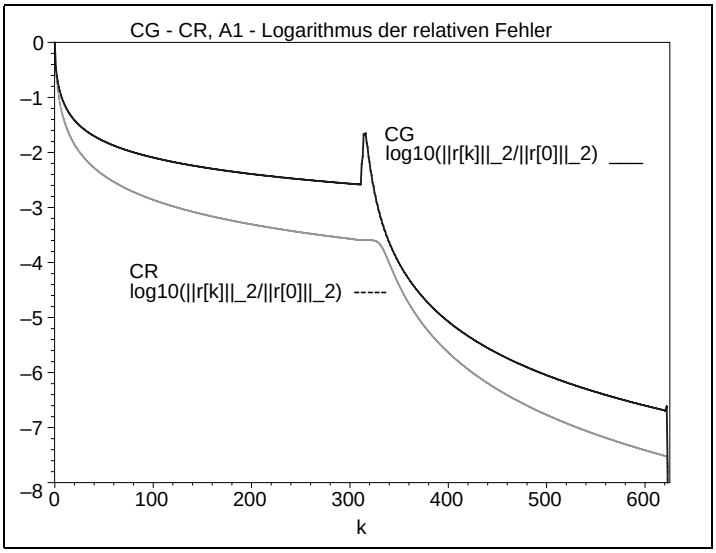


Abb. 6.57
Datei *ccgra1c063.ps*
Matrix $A1(625, 625)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)625$

Für die Matrix $A1(625, 625)$ mit Shift $c = 1$ liegen die Fehler bei den ersten 100 Iterationen des CG sehr dicht beieinander. Es gilt $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$. Zusätzlich ist $\|e^{(0)}\|_A = 25.019...$, $\|e^{(0)}\|_2 = 24.979...$, $\|r^{(0)}\|_2 = 25.119...$, so dass sich die relativen Fehler ebensfalls nur minimal unterscheiden.

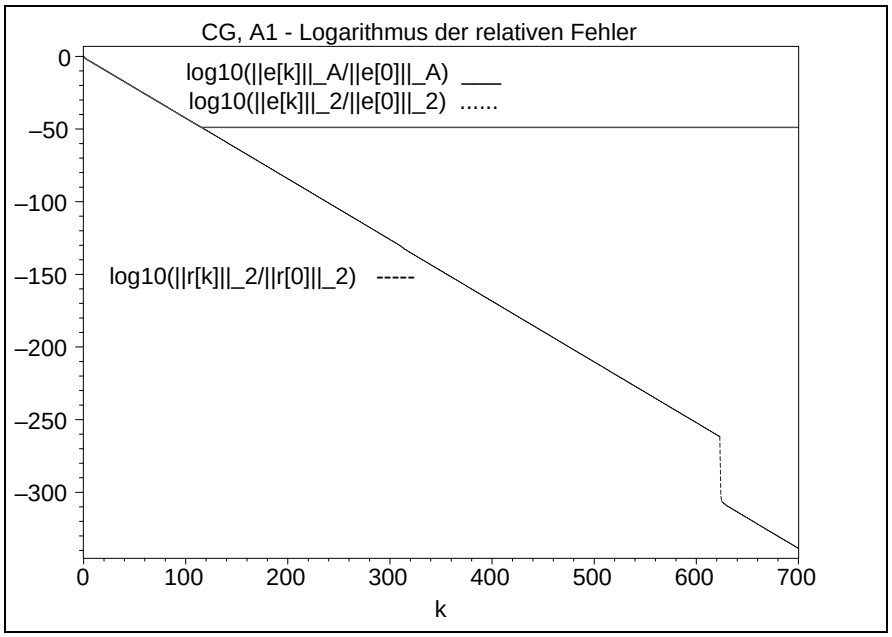


Abb. 6.58 Datei *ccga1c16.ps*, Matrix $A1(625, 625)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

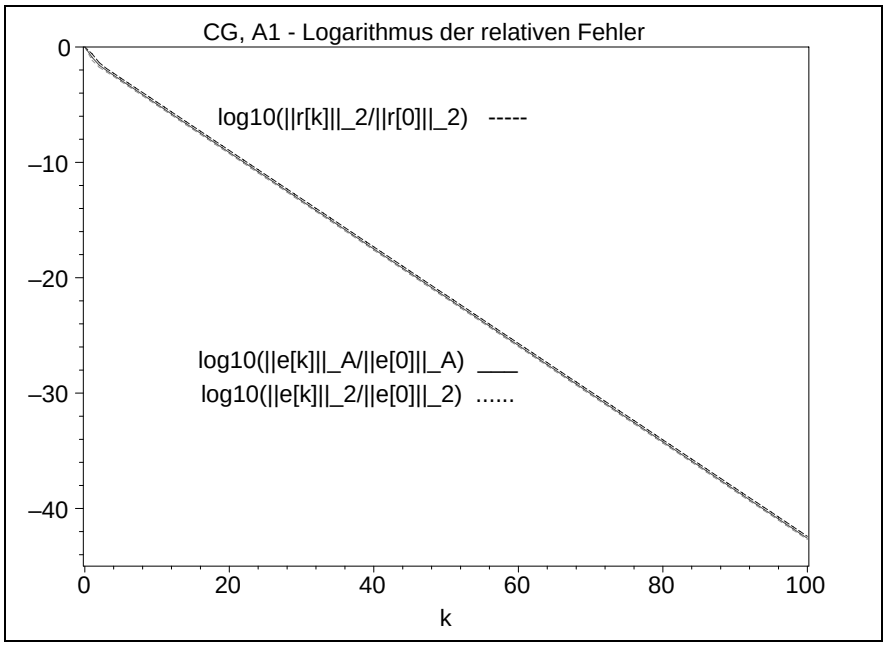


Abb. 6.59 Datei *ccga1c16a.ps*, Matrix $A1(625, 625)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Analoges gilt für das CR bei den ersten 100 Iterationen und genauso ist $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$.

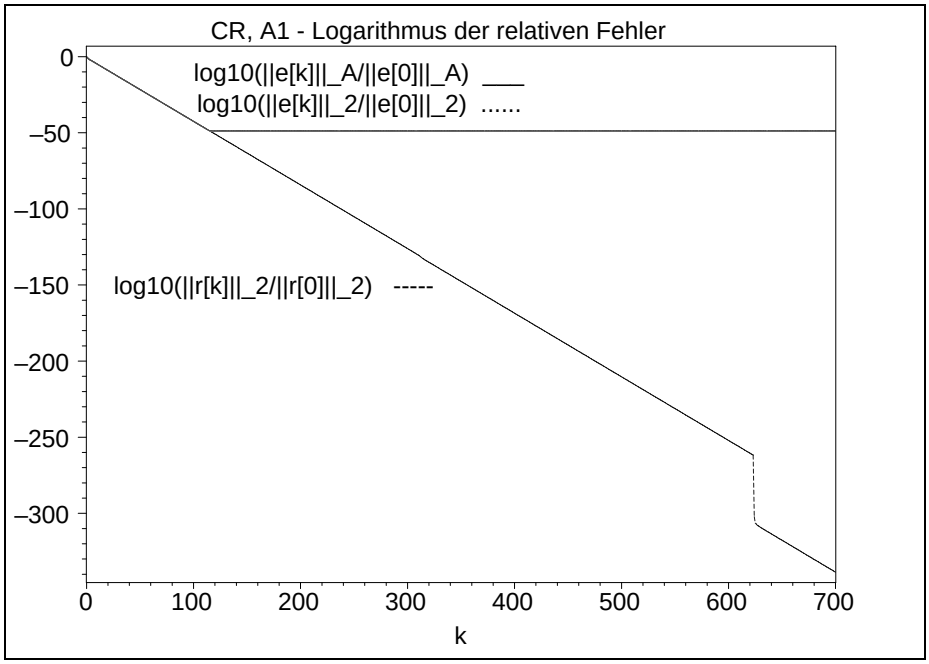


Abb. 6.60 Datei *ccra1c16.ps*, Matrix $A1(625, 625)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

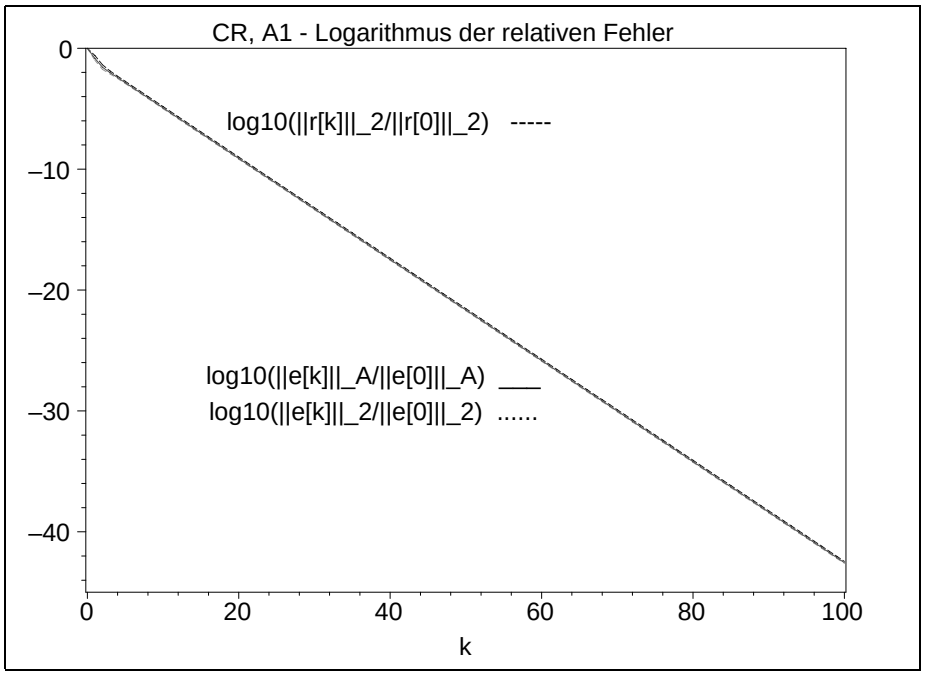


Abb. 6.61 Datei *ccra1c16a.ps*, Matrix $A1(625, 625)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Im Vergleich der AV ist bei den Fehlern $\|e^{(k)}\|_A$ und $\|e^{(k)}\|_2$ das CG minimal besser als CR, bei $\|r^{(k)}\|_2$ umgekehrt.

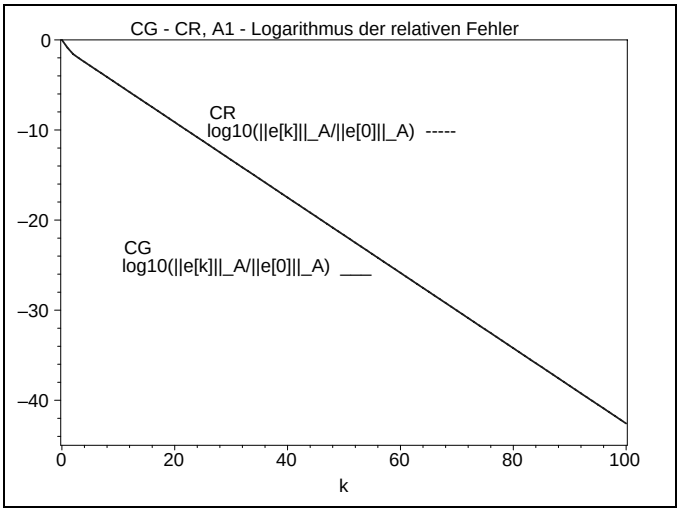


Abb. 6.62
Datei *ccgra1c161.ps*
Matrix $A1(625, 625)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)100$

Ein sehr ähnliches Bild ergibt sich zu CG versus CR für den Verlauf des relativen Fehlers $\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)100$.

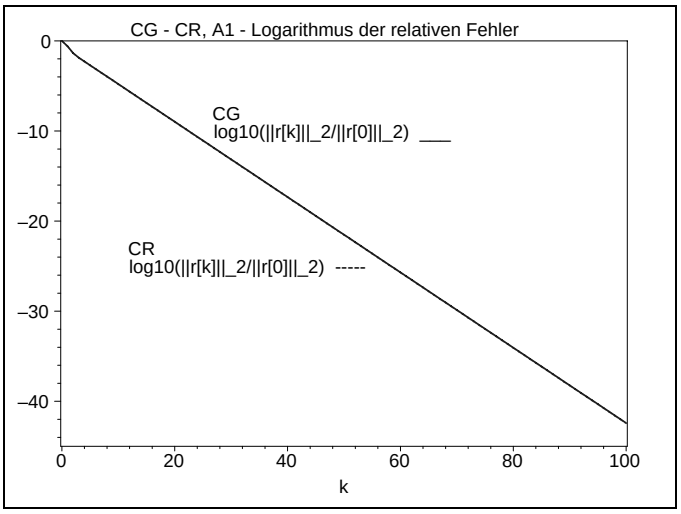


Abb. 6.63
Datei *ccgra1c163.ps*
Matrix $A1(625, 625)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)100$

Mit dem Schift $c = 1$ wird die Diagonaldominanz der Matrix $A1$ verstärkt.
Damit verbessert sich die spektrale Kondition der Matrix auf $\kappa = 4.999\,8$ und man erhält den Quotienten $q = 0.382$ deutlich kleiner als Eins im Konvergenzfaktor q^k.
Somit nehmen die Fehler stetig ab.

Für die Matrix $A1(625, 625)$ mit Shift $c = 2$ folgen analoge Aussagen und Ergebnisse wie in der Situation zu $c = 1$. Die Fehler tendieren jedoch noch schneller gegen Null wegen $\kappa = 2.999\,95$ und $q = 0.268$.

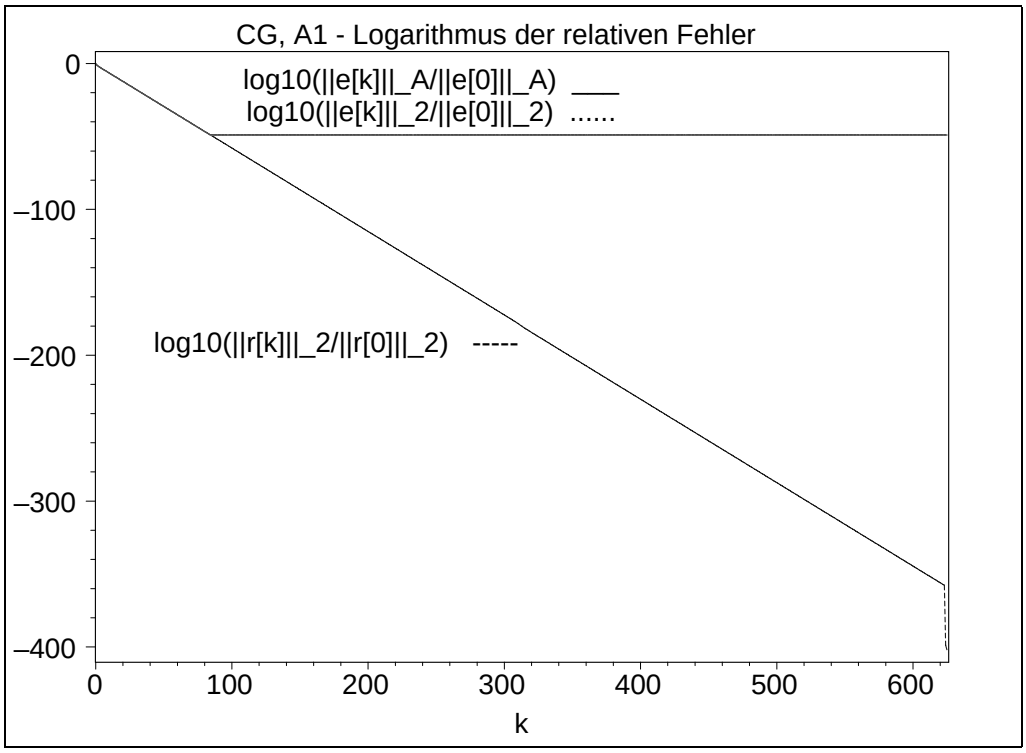


Abb. 6.64 Datei *ccga1c26.ps*, Matrix $A1(625, 625)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)625$

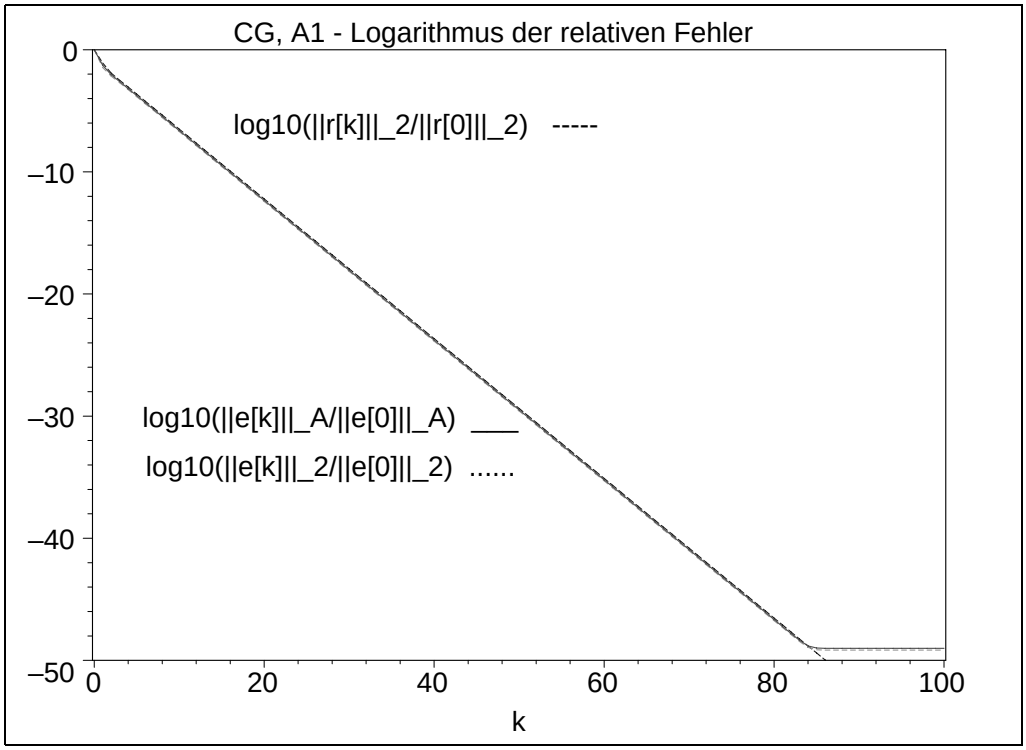


Abb. 6.65 Datei *ccga1c26a.ps*, Matrix $A1(625, 625)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

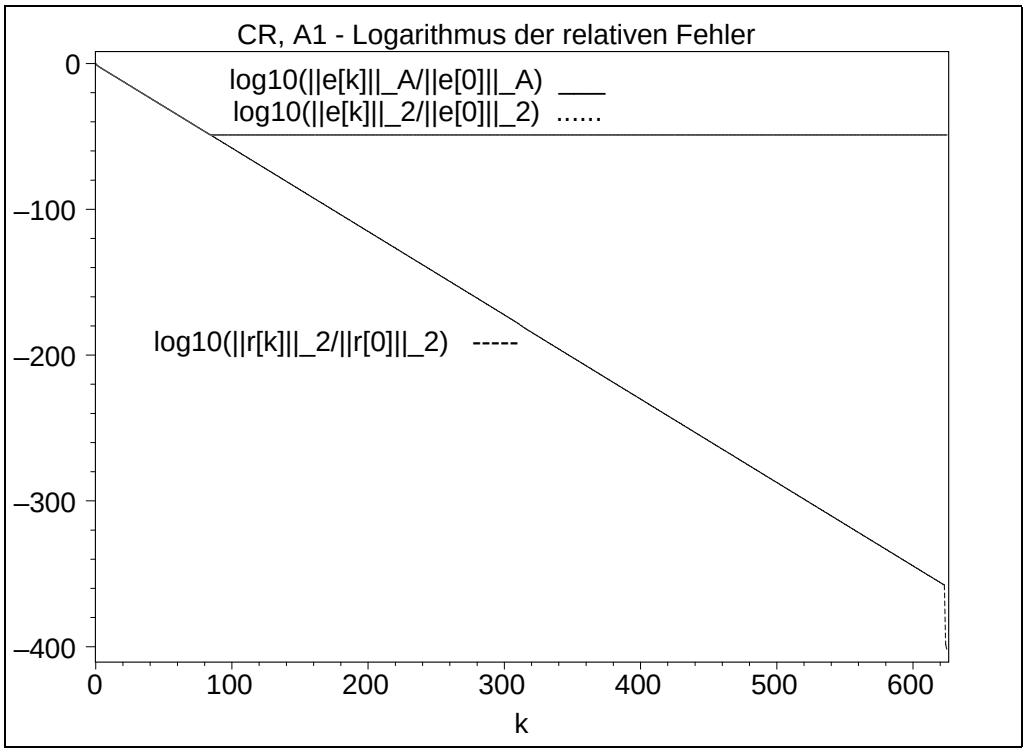


Abb. 6.66 Datei *ccra1c26.ps*, Matrix $A1(625, 625)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)625$

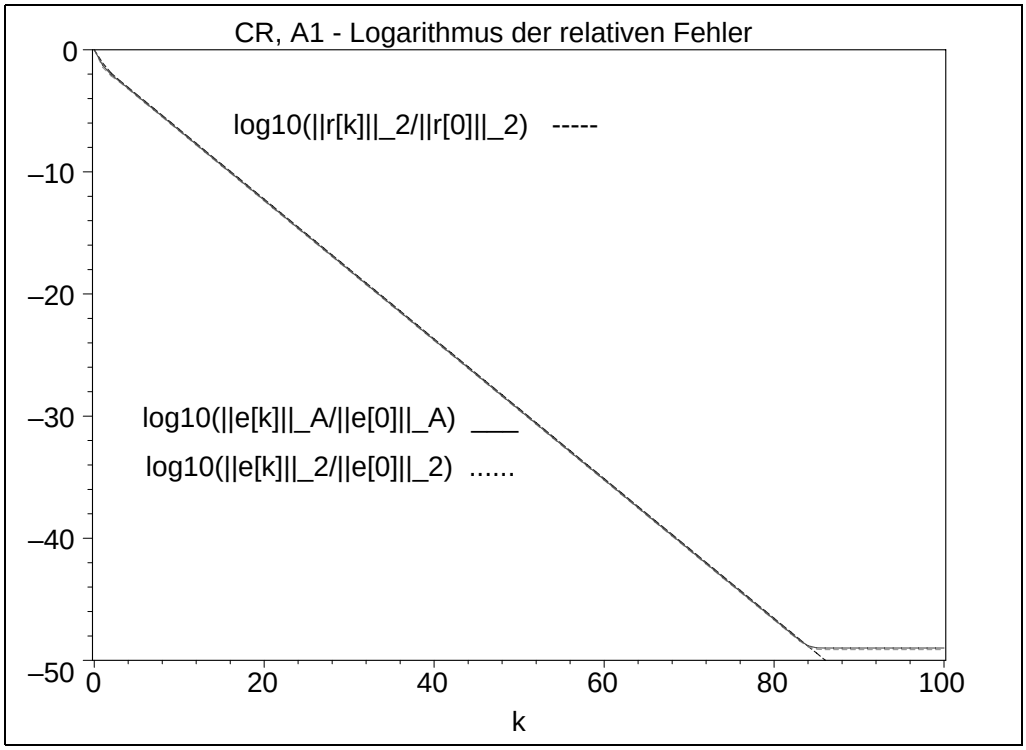


Abb. 6.67 Datei *ccra1c26a.ps*, Matrix $A1(625, 625)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Auch hier betrachten wir die symmetrische, aber indefinite Matrix $A1(625, 625)$ mit dem Shift $c = -1$. Glücklicherweise ist die Matrix regulär, ihr betragskleinster EW ist 0.002 898.... Die spektrale Konditionszahl beträgt $\kappa = 1\,034.887$.

Wie bei den anderen Shifts interessieren wir uns für die Fehlerverläufe von CG und CR und hoffen, dass sie nicht vorzeitig abbrechen (durch Nulldivision bei Schrittzahl) bzw. divergieren.

Dabei kann man beim CG kein monoton abnehmendes Verhalten der Fehler erwarten. Aber, wenn es nicht vorzeitig abbricht, ist es nach spätestens n Schritten (theoretisch) an der Lösung. Für das CR erhält man eine stetig abnehmende Folge von Fehlern $\|r^{(k)}\|_2$ und damit auch von $\|e^{(k)}\|_2$ mit spätestens $\|e^{(n)}\|_2 = 0,\ \|r^{(n)}\|_2 = 0$ (theoretisch). Beide AV brauchen in der Tat fast n Iterationen.

```
CG : Iterationsverlauf  mit verschiedenen Fehlern
 i         Q(x[i])=          ||e[i]||_A=||xs-x[i]||_A    ||e[i]||_2=            ||r[i]||_2=
      (||e[i]||^2_A-xs'b)/2    =sqrt(2Q(x[i])+xs'b)      ||xs-x[i]||_2          ||b-Ax[i]||_2
-------------------------------------------------------------------------------------------------
  0  5.0000000000000000e-01  2.4939927826679852e+01  2.4979991993593593e+01  2.4959967948697370e+01
  1  3.1401292407108239e+02  2.2418403471623000e+00  1.7432553584774996e+00  3.3341579064382761e+00
  2  3.1158068152538006e+02  4.0170020009966808e-01  8.5356883645968903e-01  7.8472783052230220e-01
  3  3.1133644498808987e+02  5.7193533185164706e-01  9.3138030507111592e-01  7.1149690404536075e-01
  4  3.1176003948257052e+02  7.2116500548837385e-01  1.2127886252790927e+00  1.2470031184572762e+00
  5  3.1151361172691490e+02  1.6499531456922303e-01  5.4937549881333289e-01  3.1866393998317714e-01
  6  3.1146077068207872e+02  2.8010468729131940e-01  6.1530632884676504e-01  3.5808084005097835e-01
  7  3.1158773438913982e+02  4.1888993575834759e-01  9.5919863455400144e-01  7.3939549682033328e-01
  8  3.1150528329490360e+02  1.0279391911583472e-01  4.3547116711167926e-01  1.9704002778862123e-01
...
100  3.1150046620455066e+02  3.0535374589537176e-02  2.8104659087723311e-01  5.5896840535477320e-02
200  3.1150000856586421e+02  4.1390492176779859e-03  8.7578197483290606e-02  7.7606388479017679e-03
300  3.1149998723573077e+02  5.0525774076371593e-03  7.9448249552207130e-02  6.6474795728549054e-03
400  3.1149997379866846e+02  7.2389683708156216e-03  1.3060474669808453e-01  1.2967426889921395e-02
500  3.1150000525086172e+02  3.2406362704260503e-03  6.3285806987344362e-02  4.2986284448929659e-03
600  3.1149999911390075e+02  1.3312394584734485e-03  5.0231457811378903e-02  2.5543004560417142e-03
...
621  3.1149999937379836e+02  1.1191082506948746e-03  4.9165525963360775e-02  2.5879883992581683e-03
622  3.1149957976607429e+02  2.8990823572797403e-02  3.5958508645979711e+00  3.0652096634968685e-01
623  3.1150000570958837e+02  3.3792272408370285e-03  6.9046839466747632e-02  2.3401992338940164e-04
624  3.1150000000000000e+02  6.9651679086490371e-41  6.9652579805852313e-41  6.9650791564782569e-41
625  3.1150000000000000e+02  9.8061521998029380e-45  8.0141858409635099e-45  1.4001734613259259e-44
626  3.1150000000000000e+02  3.3348391109317403e-45  4.2071127901923428e-45  4.9849084897927835e-45
```

```
CR : Iterationsverlauf  mit verschiedenen Fehlern
 i         Q(x[i])=          ||e[i]||_A=        ||e[i]||_2=    ||r[i]||_2=            R(x[i])=
      (||e[i]||^2_A-xs'b)/2  ||xs-x[i]||_A=     ||xs-x[i]||_2  ||b-Ax[i]||_2=     (||r[i]||^2_2-b'b)/2
                             sqrt(2Q(x[i])+xs'b)               sqrt(2R(x[i])+b'b)
-------------------------------------------------------------------------------------------------------
  0  5.0000000000000000e-01  2.493992782e+01  2.497999199e+01  2.495996794e+01  0.0000000000000000e+00
  1  3.1391657182374016e+02  2.198441185e+00  1.748008988e+00  3.304803358e+00 -3.0603913738019169e+02
  2  3.1158733584264929e+02  4.179374179e-01  8.615006598e-01  7.634988522e-01 -3.1120853475131593e+02
  3  3.1139064443382291e+02  4.676656202e-01  8.368485280e-01  5.205181674e-01 -3.1136453041866288e+02
  4  3.1149213459279539e+02  1.254225434e-01  7.440521066e-01  4.803505416e-01 -3.1138463167857223e+02
  5  3.1151160591200191e+02  1.523542713e-01  5.684882377e-01  2.655442589e-01 -3.1146474312327172e+02
  6  3.1148193175116490e+02  1.900960222e-01  5.511238133e-01  2.132947272e-01 -3.1147725267965980e+02
  7  3.1149756349125939e+02  6.980700166e-02  5.282349435e-01  2.049380474e-01 -3.1147900019835064e+02
  8  3.1150350198543963e+02  8.368972983e-02  4.564251081e-01  1.420384336e-01 -3.1148991254167966e+02
...
100  3.1149999895505291e+02  1.445646632e-03  1.342713283e-01  3.681668676e-03 -3.1149999322265788e+02
200  3.1150000033816435e+02  8.223920589e-04  9.469308182e-02  1.291834465e-03 -3.1149999916558186e+02
300  3.1149999982548701e+02  5.907842121e-04  7.734792495e-02  7.039972498e-04 -3.1149999975219394e+02
400  3.1149999995104426e+02  3.129081043e-04  6.647990904e-02  4.530433211e-04 -3.1149999989737587e+02
500  3.1150000000571062e+02  1.068701678e-04  5.949803893e-02  3.238578937e-04 -3.1149999994755803e+02
```

```
600  3.1149999998143008e+02  1.927169920e-04  5.434574010e-02  2.459824443e-04 -3.1149999996974632e+02
...
622  3.1149999999037397e+02  1.387517586e-04  5.343941830e-02  2.337542011e-04 -3.1149999997267949e+02
623  3.1150000427653409e+02  2.924562904e-03  5.561676958e-02  1.653830477e-04 -3.1149999998632422e+02
624  3.1150000000000000e+02  5.179435145e-41  5.179531780e-41  5.179338438e-41 -3.1150000000000000e+02
625  3.1150000000000000e+02  1.663151522e-44  1.222611052e-44  2.598124795e-44 -3.1150000000000000e+02
626  3.1150000000000000e+02  3.150982989e-45  6.098065036e-45  6.730733585e-45 -3.1150000000000000e+02
```

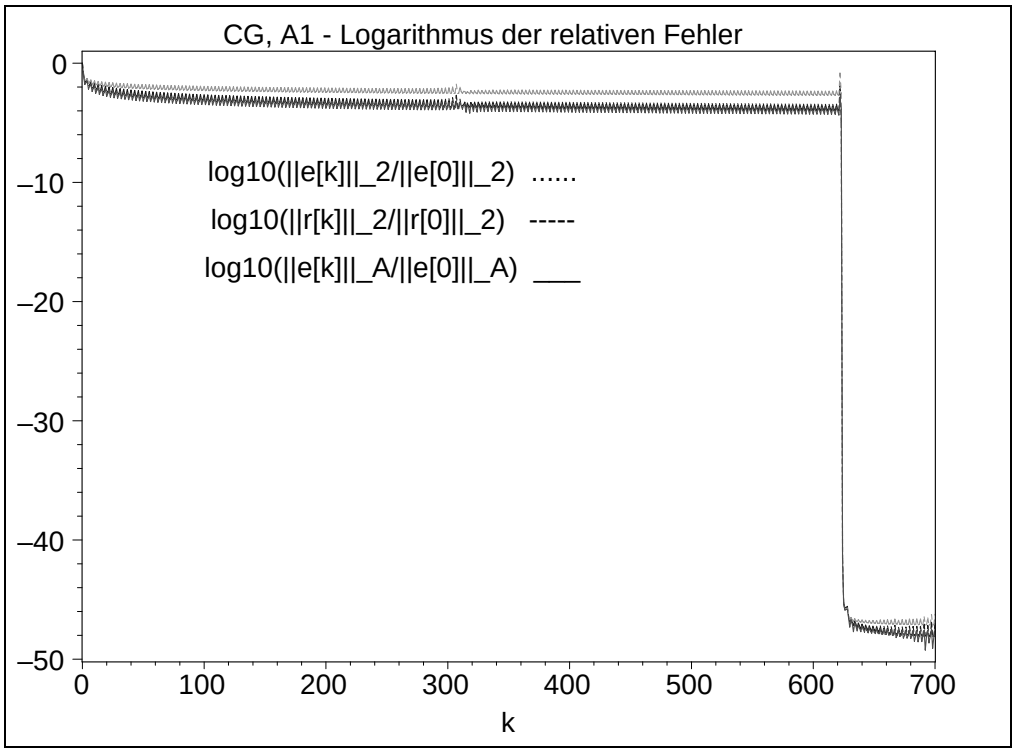


Abb. 6.68 Datei *ccga*1*cm*16.*ps*, Matrix $A1(625, 625)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

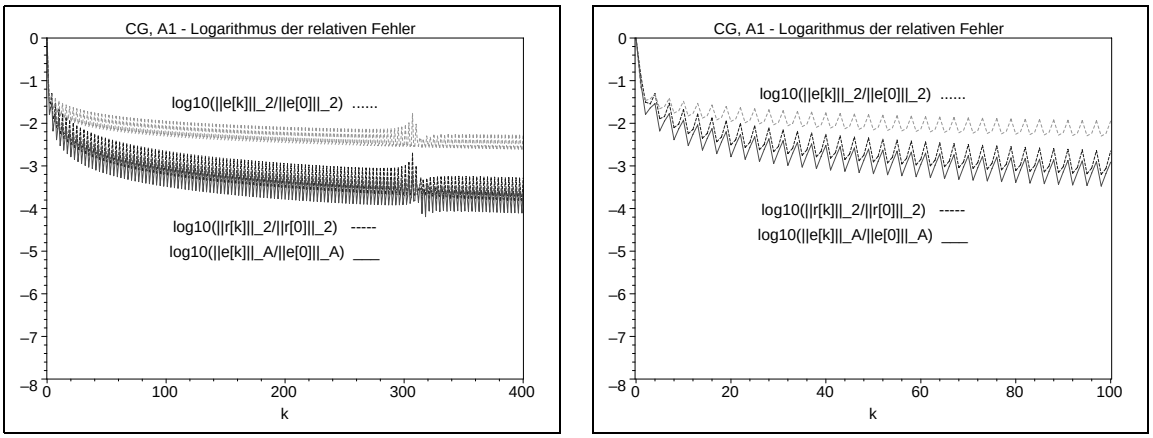


Abb. 6.69 Dateien *ccga*1*cm*16*a.ps*, *ccga*1*cm*16*aa.ps*, Matrix $A1(625, 625)$, $c = -1$,
CG: Verlauf der 3 relativen Fehler für $k = 0(1)400$ bzw. $k = 0(1)100$

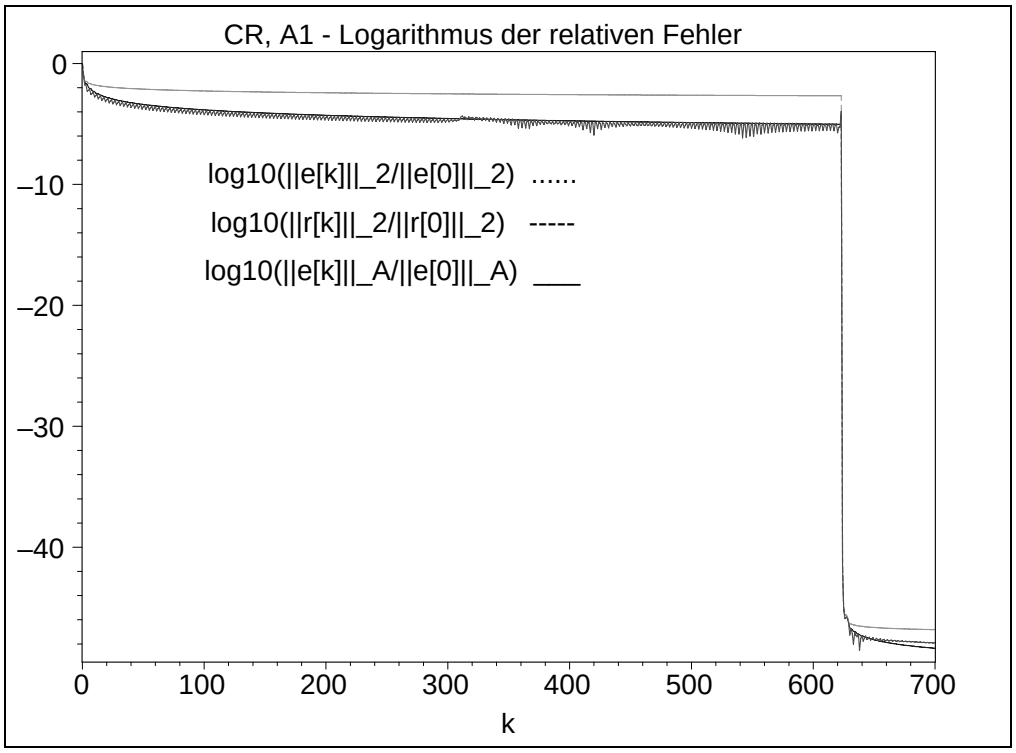


Abb. 6.70 Datei $ccra1cm16.ps$, Matrix $A1(625, 625)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

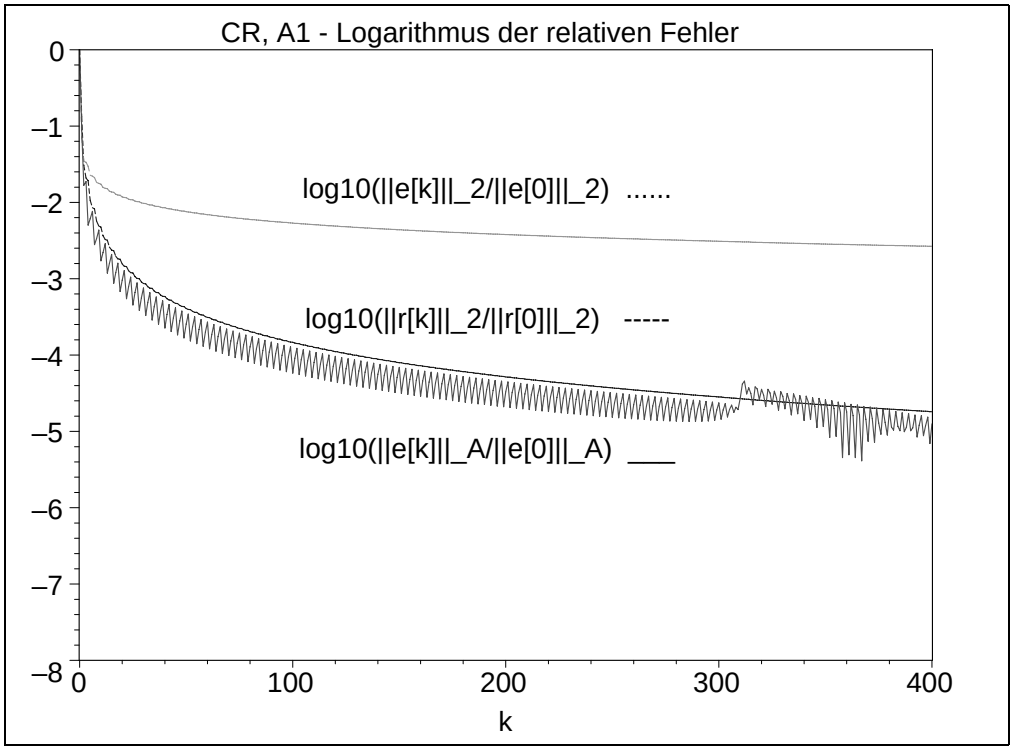


Abb. 6.71 Datei $ccra1cm16a.ps$, Matrix $A1(625, 625)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)400$

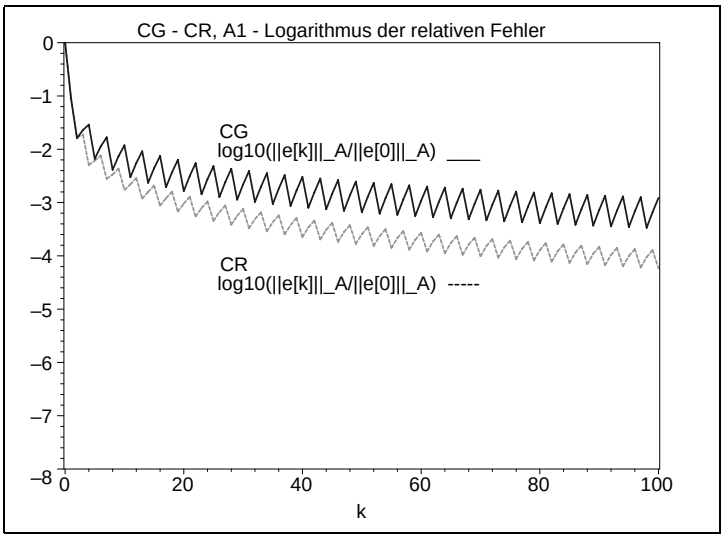


Abb. 6.72
Datei $ccgra1cm161.ps$
Matrix $A1(625, 625)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)100$

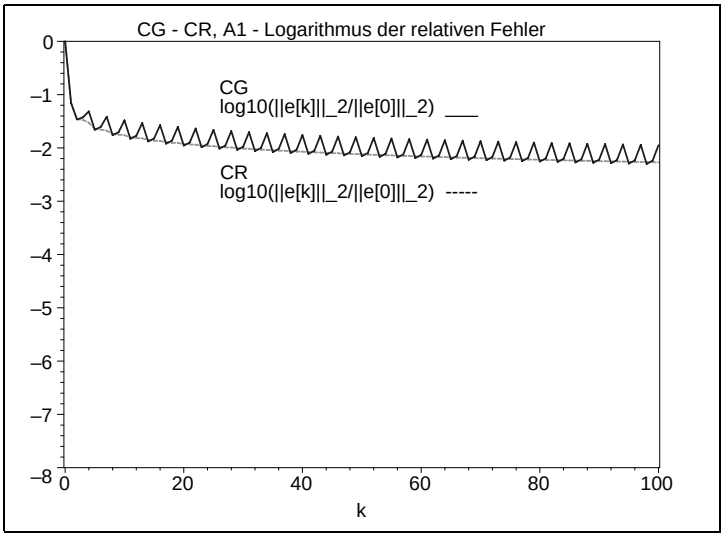


Abb. 6.73
Datei $ccgra1cm162.ps$
Matrix $A1(625, 625)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)100$

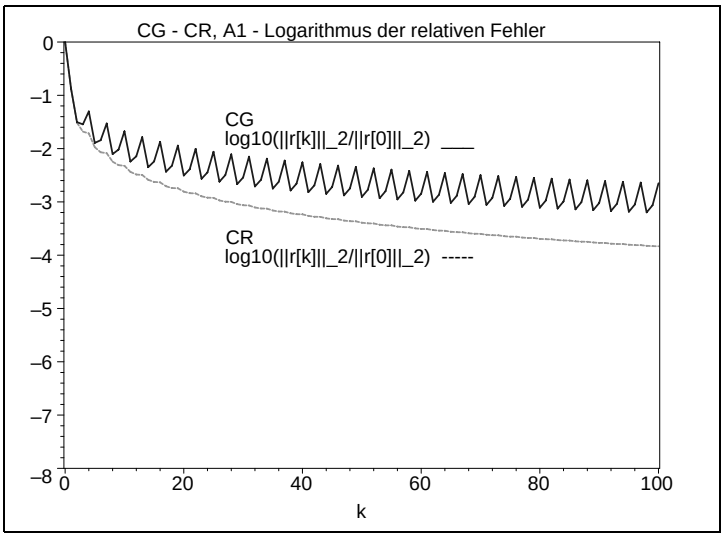


Abb. 6.74
Datei $ccgra1cm163.ps$
Matrix $A1(625, 625)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)100$

Als Ergänzung noch die symmetrische, aber indefinite Matrix $A1(625, 625)$ mit dem Shift $c = -1.5$. Glücklicherweise ist die Matrix regulär, ihr betragskleinster EW ist 0.003 398.... Die spektrale Konditionszahl beträgt $\kappa = 735.719$.
Mit $c = -2$ wird die Matrix singulär.

Wir interessieren uns wiederum für die Fehlerverläufe von CG und CR. Beide AV brechen zum Glück nicht vorzeitig ab, brauchen jedoch fast n Iterationen.

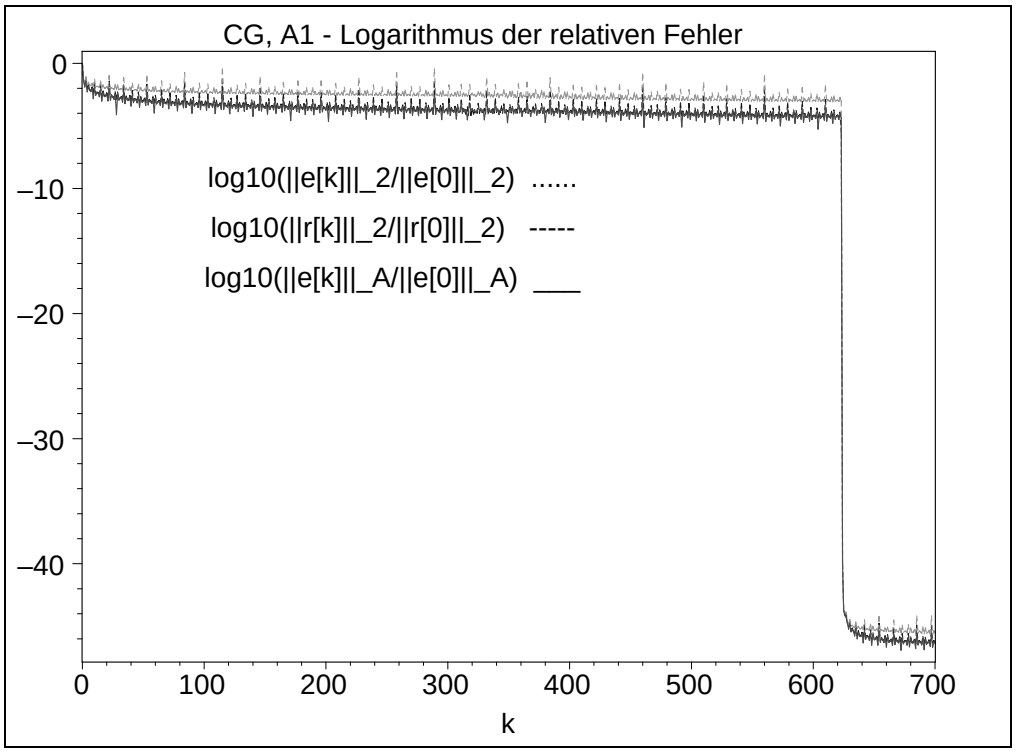


Abb. 6.75 Datei *ccga*1*cm*26.*ps*, Matrix $A1(625, 625)$, Shift $c = -1.5$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

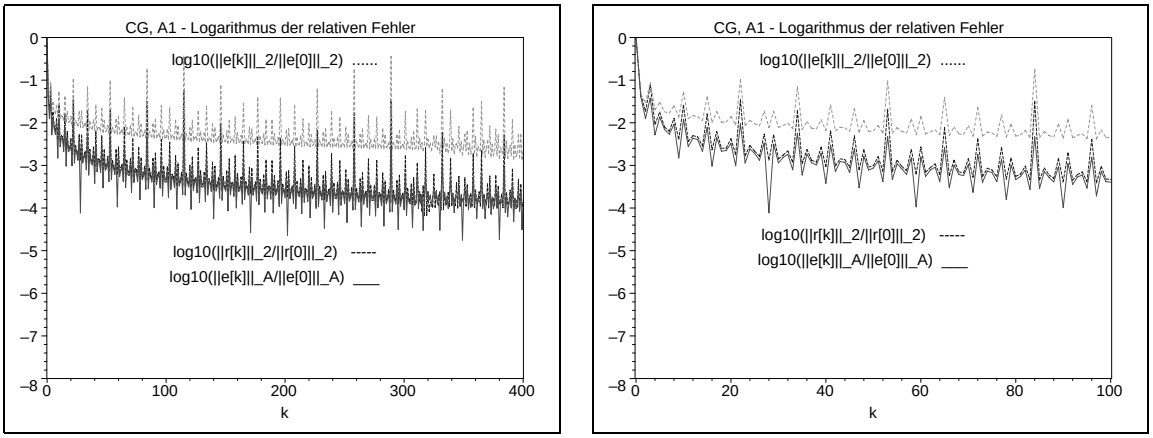


Abb. 6.76 Dateien *ccga*1*cm*26*a.ps*, *ccga*1*cm*26*aa.ps*, Matrix $A1(625, 625)$, $c = -1.5$,
CG: Verlauf der 3 relativen Fehler für $k = 0(1)400$ bzw. $k = 0(1)100$

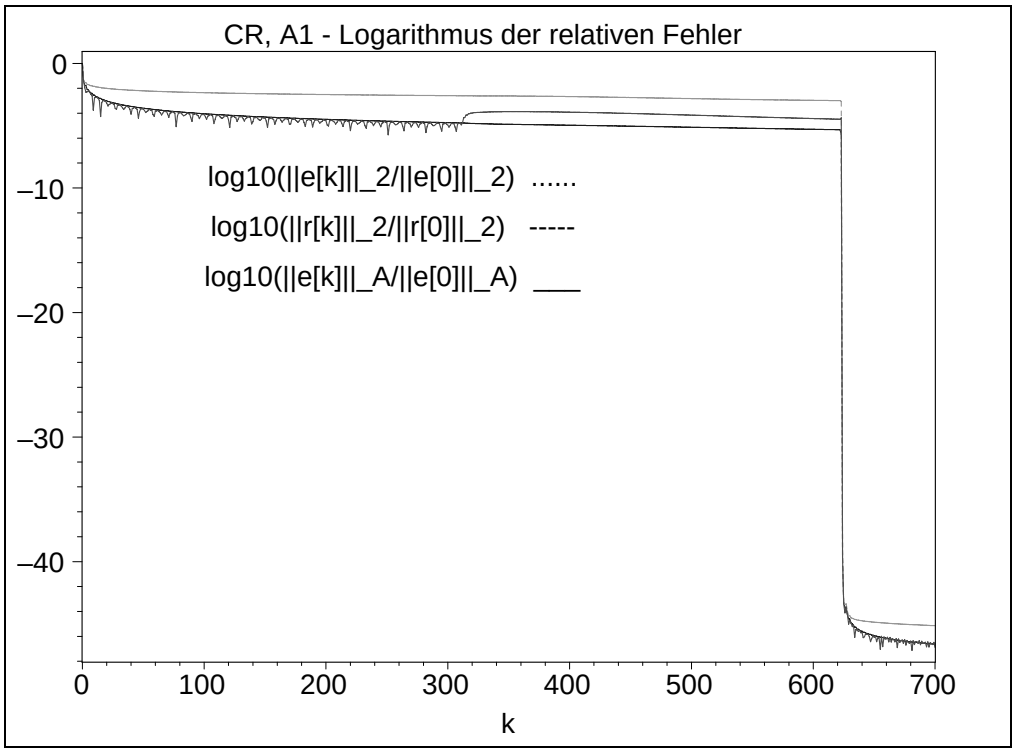


Abb. 6.77 Datei *ccra*1*cm*26.*ps*, Matrix $A1(625, 625)$, Shift $c = -1.5$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

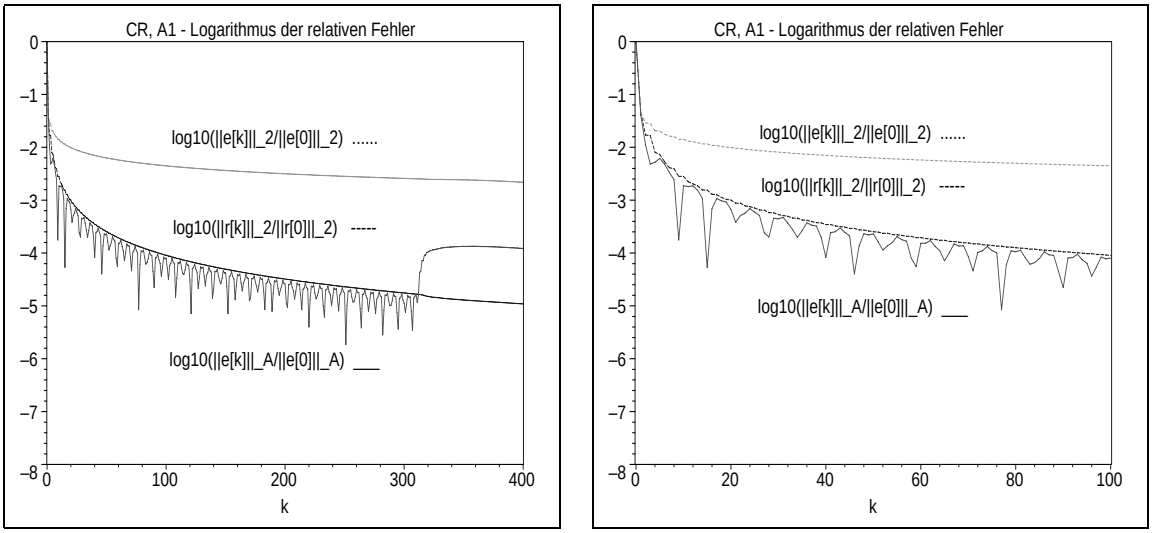


Abb. 6.78 Dateien *ccra*1*cm*26*a.ps*, *ccra*1*cm*26*aa.ps*, Matrix $A1(625, 625)$,$c = -1.5$,
CR: Verlauf der 3 relativen Fehler für $k = 0(1)400$ bzw. $k = 0(1)100$

Man bemerke den eigenartigen Sprung von $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$ bei der 313. Iteration zwischen den Werten $\|e^{(312)}\|_A = 0.000\,488...$ und $\|e^{(313)}\|_A = 0.001\,312...$ mit dem Übergang vom oszillierenden zu einem glatten Verlauf.

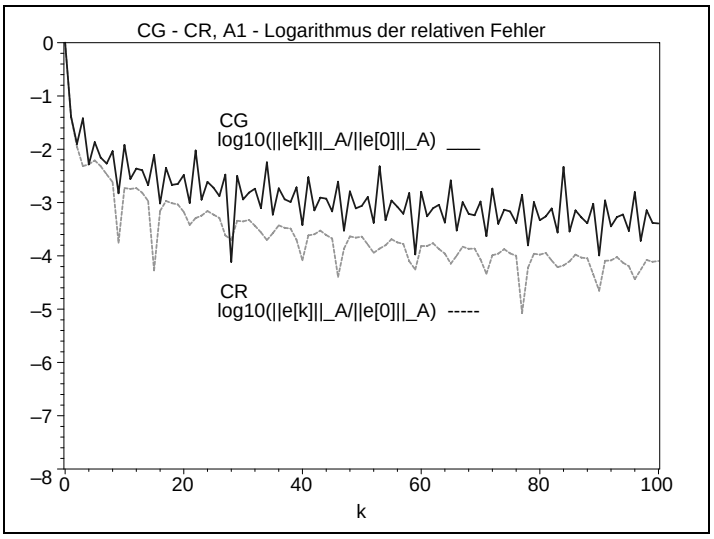


Abb. 6.79
Datei $ccgra1cm261.ps$
Matrix $A1(625, 625)$, $c{=}-1.5$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)100$

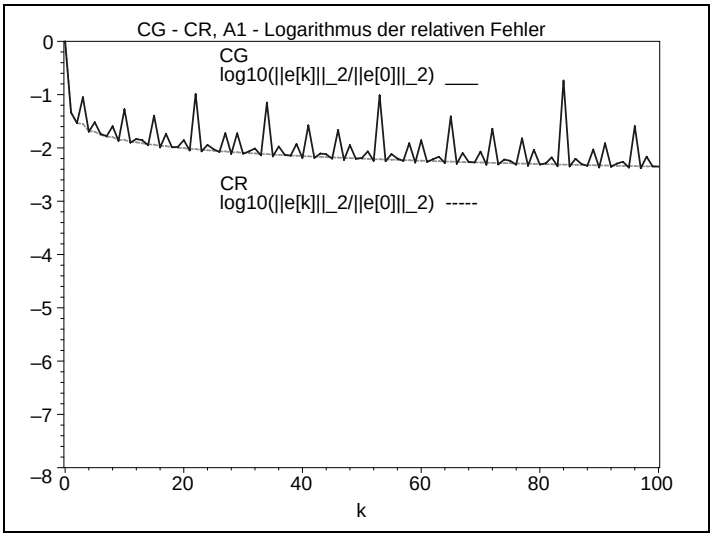


Abb. 6.80
Datei $ccgra1cm262.ps$
Matrix $A1(625, 625)$, $c{=}-1.5$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)100$

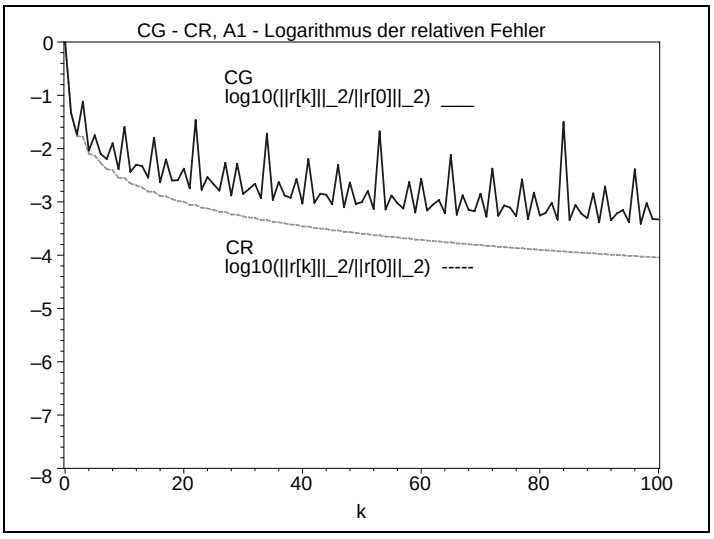


Abb. 6.81
Datei $ccgra1cm263.ps$
Matrix $A1(625, 625)$, $c{=}-1.5$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)100$

Ergebnisse zu $A2(625, 625) = A2(25^2, 25^2)$ blocktridiagonal

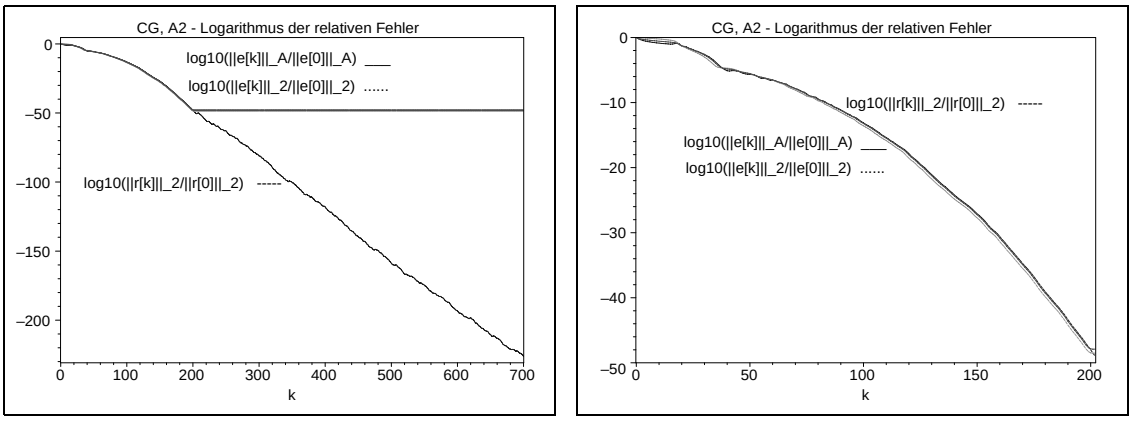


Abb. 6.82 Dateien *ccga2c06.ps*, *ccga2c06a.ps*, Matrix $A2(625, 625)$, Shift $c = 0$, CG: Verlauf der 3 relativen Fehler für $k = 0(1)700$ bzw. $k = 0(1)202$

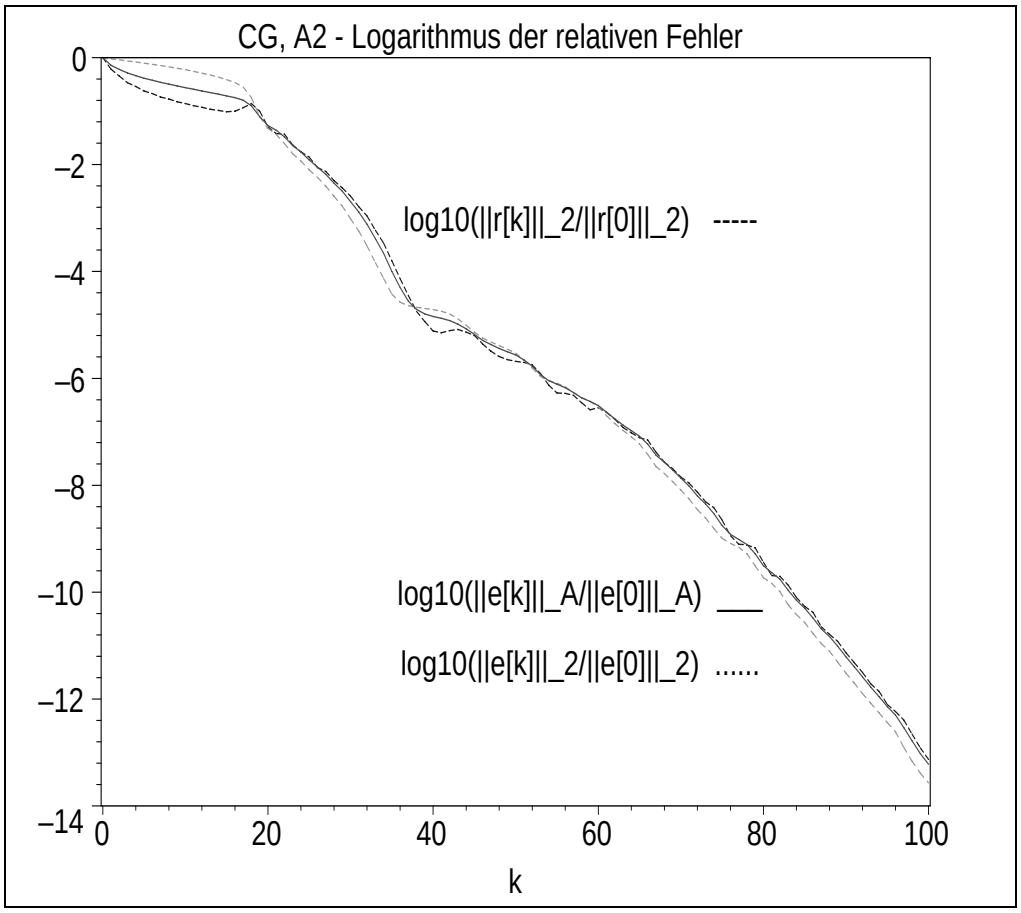


Abb. 6.83 Datei *ccga2c06aa.ps*, Matrix $A2(625, 625)$, Shift $c = 0$, CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

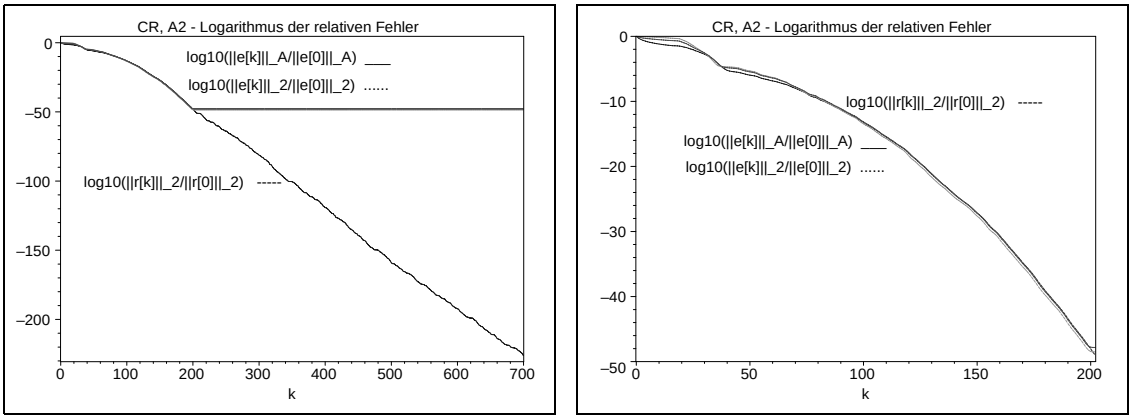


Abb. 6.84 Dateien *ccra2c06.ps*, *ccra2c06a.ps*, Matrix $A2(625, 625)$, Shift $c = 0$, CR: Verlauf der 3 relativen Fehler für $k = 0(1)700$ bzw. $k = 0(1)202$

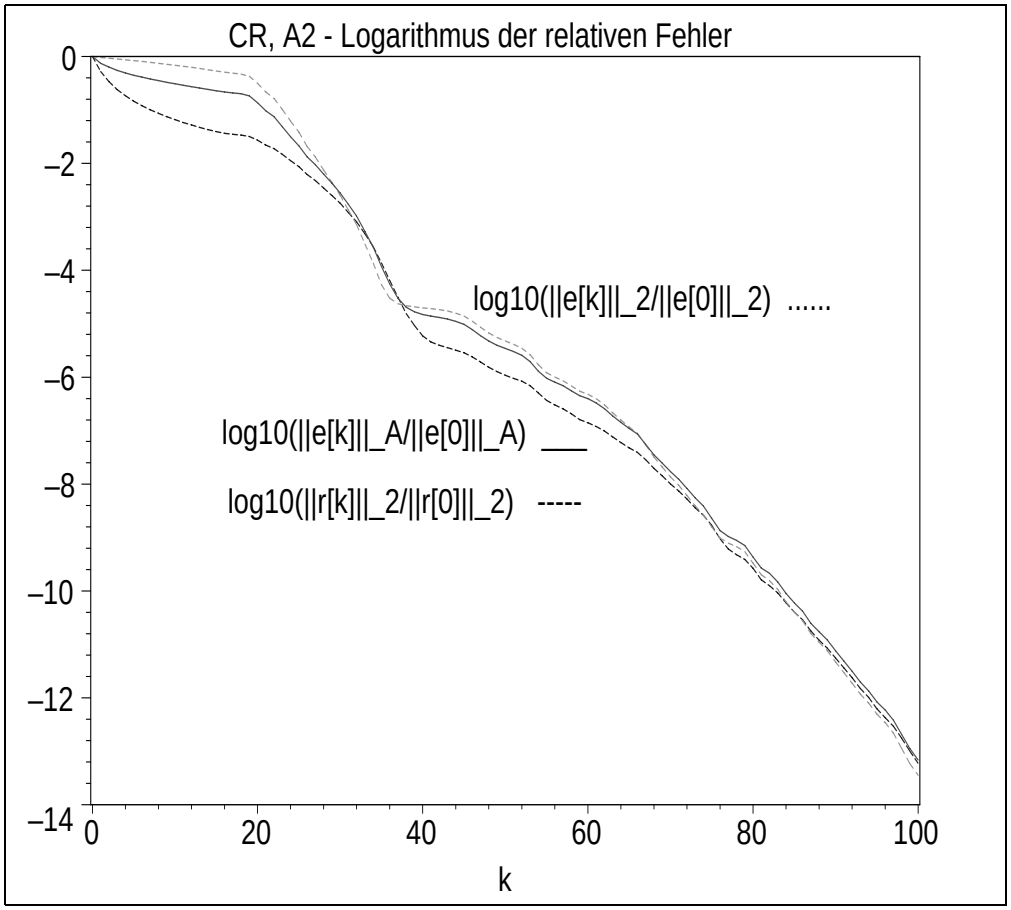


Abb. 6.85 Datei *ccra2c06aa.ps*, Matrix $A2(625, 625)$, Shift $c = 0$, CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)100$

Die beiden AV CG und CR sind für die spd Matrix $A2 = A2(625, 625) = A2(25^2, 25^2)$, Shift $c = 0$, theoretisch endlich und brauchen höchstens $n = 625$ Schritte.
Die numerischen Rechnungen in der sehr genauen GPA mit `Digits:=50` widerspiegeln dieses Verhalten. Nach stetigen Genauigkeitszuwächsen haben wir schon nach ungefähr $n/3$ Iterationen eine deutliche Fehlerverkleinerung und das AV könnte dort schon beendet werden. Alle Fehler tendieren natürlich gegen Null. Die Konvergenz ist linear. Der kleinste Eigenwert von $A2$ ist 0.029 164..., der größte 7.970 835....
Die spektrale Matrixkondition beträgt $\kappa = 273.306$, womit sich der Quotient

$$q \;=\; \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \;=\; 0.886$$

im Konvergenzfaktor q^k ergibt.

Im Fall A spd erzeugt das CG eine monoton fallende Folge von Werten $\|e^{(k)}\|_A$ und das gilt auch für $\|e^{(k)}\|_2$, während sich dieses Eigenschaft nicht auf die Residuumnorm $\|r^{(k)}\|_2$ überträgt. Unter anderem ist $\|r^{(15)}\|_2 < \|r^{(16)}\|_2$ und $\|r^{(41)}\|_2 < \|r^{(42)}\|_2$.

Das CR erzeugt eine monoton fallende Folge von Werten $\|r^{(k)}\|_2$ und das gilt ebenfalls für die Fehlernorm $\|e^{(k)}\|_2$ sowie meistens auch für $\|e^{(k)}\|_A$.

Ergebnisse aus Berechnungen mit Maple

Startvektor $x^{(0)} = (1, 0, 0, ..., 0)^T$, die rechte Seite b wird so gewählt, dass die exakte Lösung $x^* = (1, 1, ..., 1)^T$ ist, $Q(x^*) = -\frac{1}{2}x^{*T}b = -50$, $R(x^*) = -\frac{1}{2}b^Tb = -54$.

```
CG : Anfaenglicher Iterationsverlauf  mit verschiedenen Fehlern

 i         Q(x[i])=           ||e[i]||_A=||xs-x[i]||_A    ||e[i]||_2=             ||r[i]||_2=
      (||e[i]||^2_A-xs'b)/2     =sqrt(2Q(x[i])+xs'b)      ||xs-x[i]||_2           ||b-Ax[i]||_2
-------------------------------------------------------------------------------------------------
  0   0.0000000000000000e+00  1.0000000000000000e+01  2.4979991993593593e+01  1.0677078252031311e+01
  1  -2.4246268656716418e+01  7.1768699783796532e+00  2.3654868330279882e+01  6.4786005848471422e+00
  2  -3.1811834566583934e+01  6.0312793723083439e+00  2.2720159942466761e+01  4.8011963930399951e+00
  3  -3.6682143245391579e+01  5.1609798981605074e+00  2.1669609417944653e+01  3.6121656518492415e+00
  4  -3.9432633665018709e+01  4.5972527307036951e+00  2.0723557664163166e+01  3.0866372949429599e+00
  5  -4.1488296476674846e+01  4.1259431705551044e+00  1.9724511940407361e+01  2.5279388267313912e+00
  6  -4.2917804694398110e+01  3.7635608951103447e+00  1.8765280680698759e+01  2.2695996216288382e+00
  7  -4.4054041223640634e+01  3.4484659709381985e+00  1.7779847615620404e+01  1.9495930926177365e+00
...
 14  -4.7855645779626191e+01  2.0709197089089710e+00  1.0782489029769145e+01  1.1011649916365136e+00
 15  -4.8139332086284086e+01  1.9290764182457440e+00  9.7215412600992540e+00  1.0346971158332668e+00
 16  -4.8408531818665835e+01  1.7840785752506332e+00  8.5582181999500377e+00  1.0645177151972188e+00
 17  -4.8728879074882891e+01  1.5944409208980489e+00  7.0015827228819389e+00  1.2387016296730159e+00
 18  -4.9220805794740919e+01  1.2483542808506574e+00  4.4444295132409355e+00  1.4753975317687644e+00
 19  -4.9693206784227926e+01  7.8331758025985150e-01  1.9805596950111099e+00  1.0594187098673719e+00
 20  -4.9854876887192702e+01  5.3874504695133429e-01  1.1864969200471009e+00  5.4134860235505371e-01
...
 40  -4.9999999989638666e+01  1.4395369817733611e-04  4.8180970983747285e-04  8.2300715224627083e-05
 80  -5.0000000000000000e+01  3.1382775803578015e-09  4.6624442945897915e-09  3.8721698441249041e-09
120  -5.0000000000000000e+01  9.1650978500868356e-18  6.9021834237141433e-18  1.5753832863362914e-17
160  -5.0000000000000000e+01  1.5344182730234065e-30  1.0242212415564584e-30  2.7039732000847980e-30
197  -5.0000000000000000e+01  1.6000343746307452e-46  9.7422016505510704e-47  2.9542897827404282e-46
198  -5.0000000000000000e+01  5.2945001652658393e-47  3.2322221458309452e-47  9.4307627176405337e-47
199  -5.0000000000000000e+01  2.1152815415447656e-47  1.3075947384415402e-47  3.1267483259209200e-47
200  -5.0000000000000000e+01  1.3828622491050943e-47  8.2661357356385093e-48  1.0546305581034402e-47
201  -5.0000000000000000e+01  1.2743092246389806e-47  7.6032427292570372e-48  3.2954070300376729e-48
202  -5.0000000000000000e+01  1.2788440092521058e-47  7.5610382884892204e-48  1.1005655156933650e-48
```

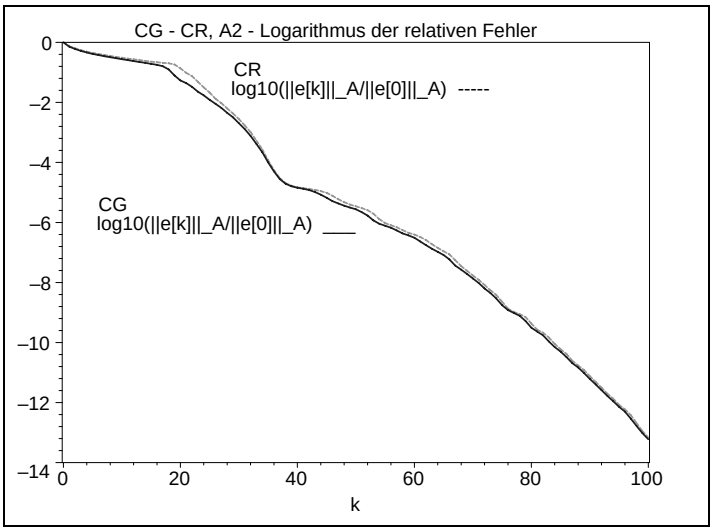


Abb. 6.86
Datei *ccgra2c061.ps*
Matrix $A2(625, 625)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)100$

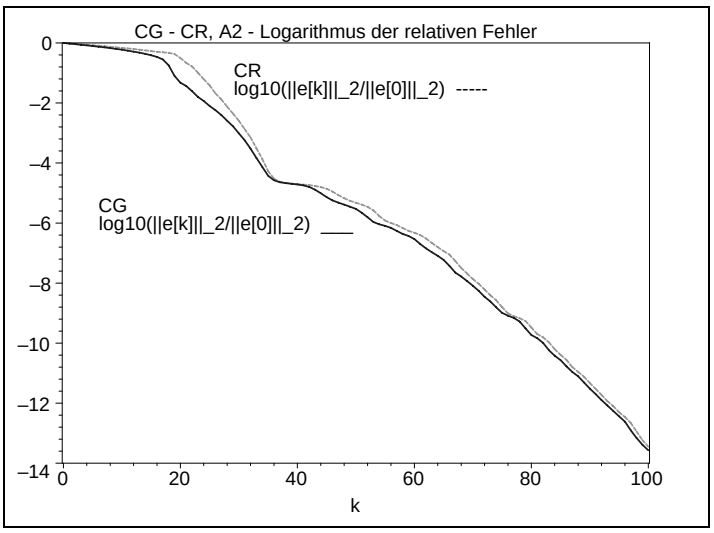


Abb. 6.87
Datei *ccgra2c062.ps*
Matrix $A2(625, 625)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)100$

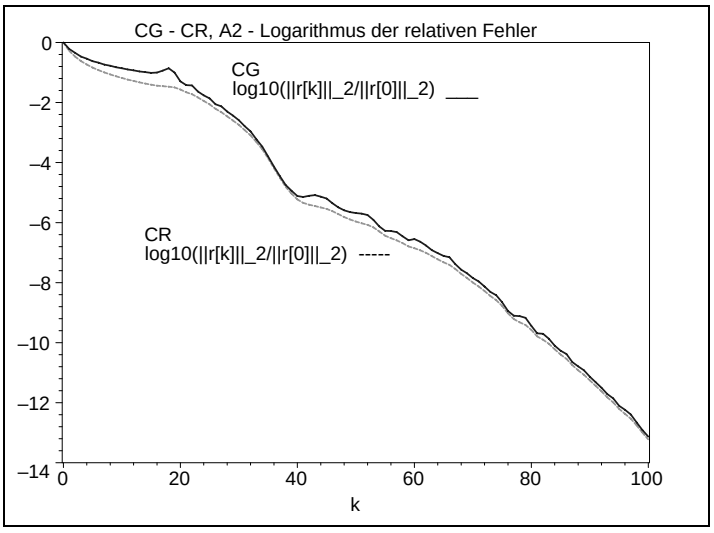


Abb. 6.88
Datei *ccgra2c063.ps*
Matrix $A2(625, 625)$, $c = 0$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)100$

Für die Matrix $A2(625, 625)$ mit Shift $c = 1$ liegen die Fehler bei den ersten 135 Iterationen des CG sehr dicht beieinander. Es gilt $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$. Zusätzlich ist $\|e^{(0)}\|_A = 26.907...$, $\|e^{(0)}\|_2 = 24.979...$, $\|r^{(0)}\|_2 = 30.626...$, so dass sich die relativen Fehler ebensfalls nur minimal unterscheiden.

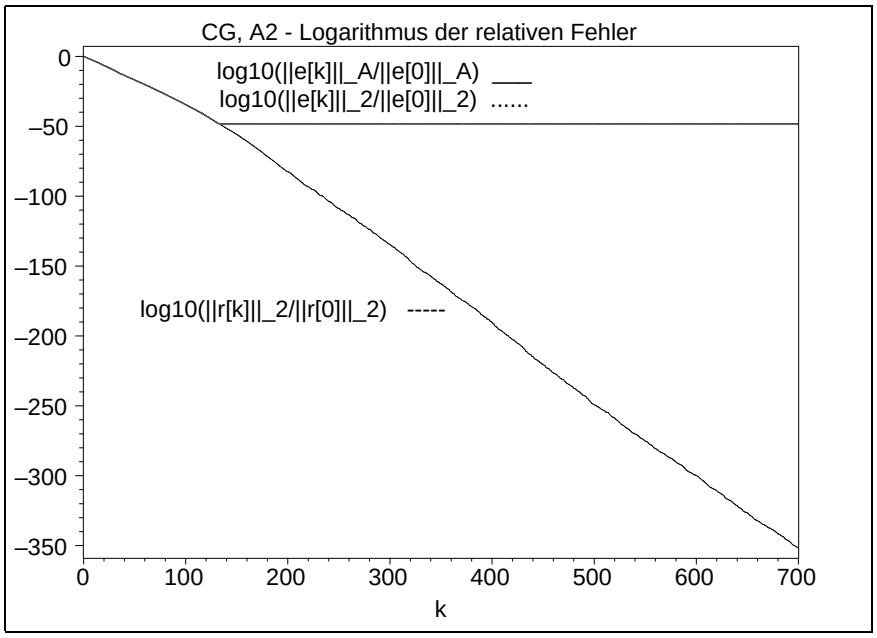


Abb. 6.89 Datei *ccga2c16.ps*, Matrix $A2(625, 625)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

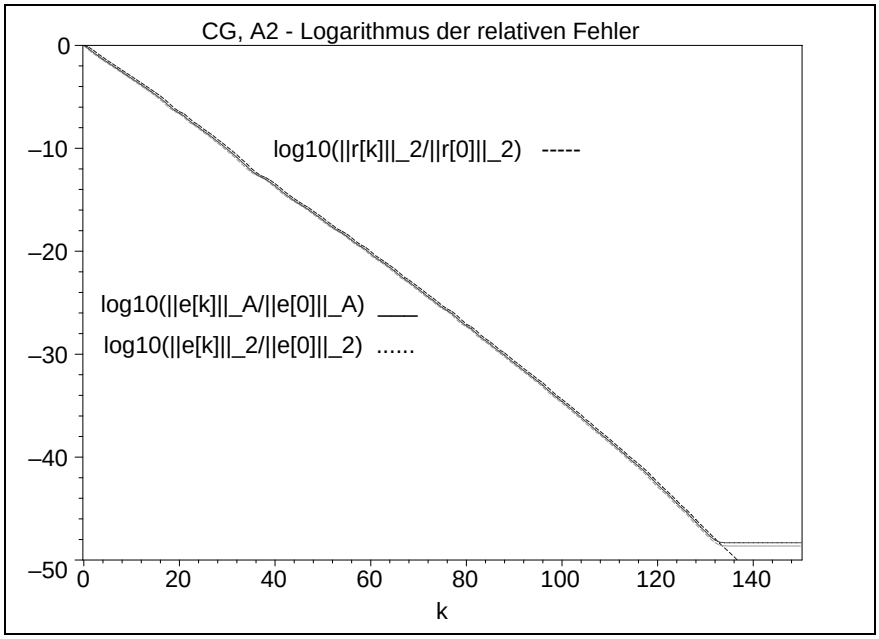


Abb. 6.90 Datei *ccga2c16a.ps*, Matrix $A2(625, 625)$, Shift $c = 1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)150$

Analoges gilt für das CR bei den ersten 135 Iterationen und genauso ist $\|e^{(k)}\|_2 < \|e^{(k)}\|_A < \|r^{(k)}\|_2$.

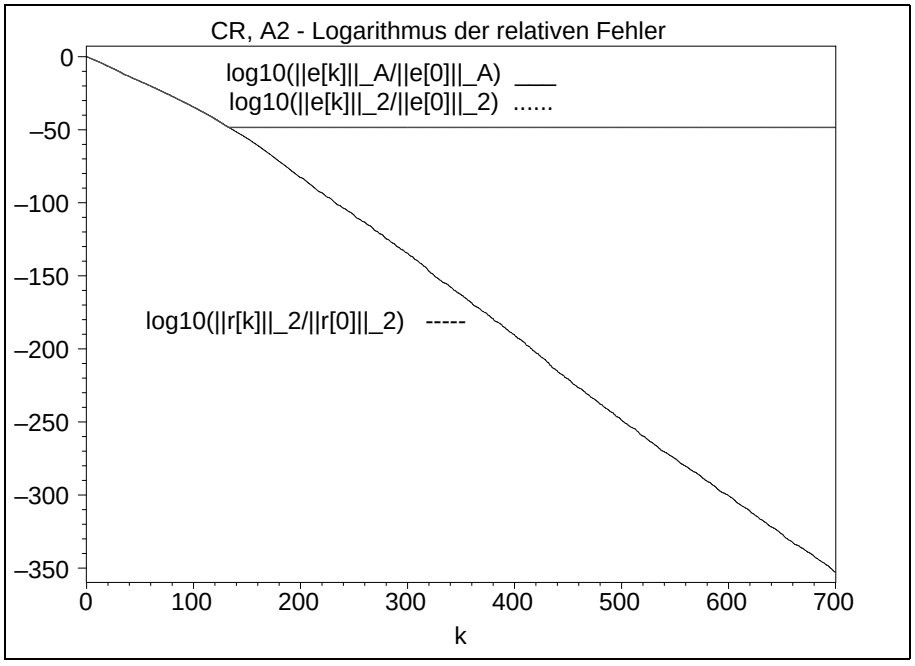


Abb. 6.91 Datei *ccra2c16.ps*, Matrix $A2(625, 625)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

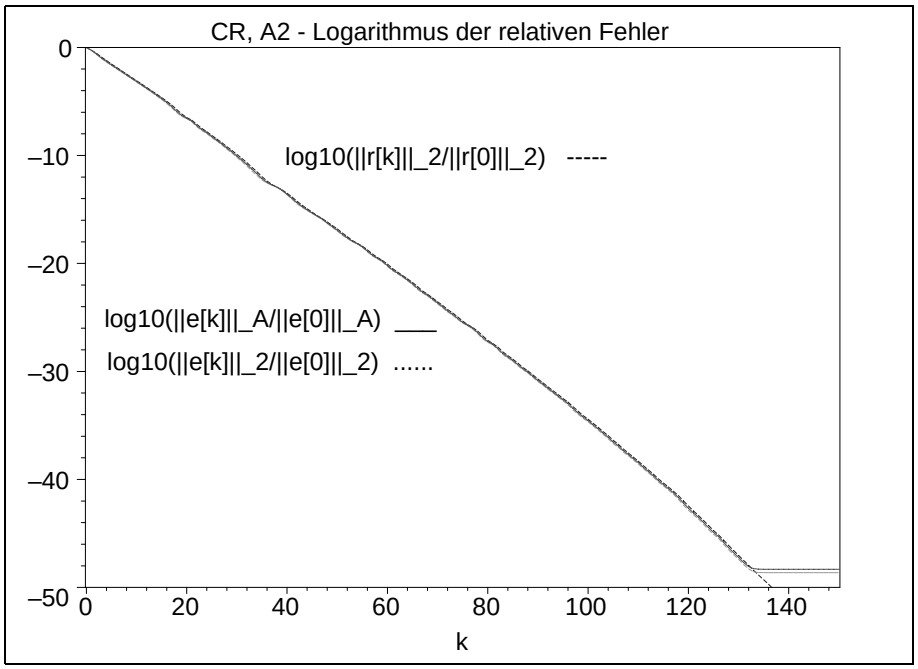


Abb. 6.92 Datei *ccra2c16a.ps*, Matrix $A2(625, 625)$, Shift $c = 1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)150$

Im Vergleich der AV ist bei den Fehlern $\|e^{(k)}\|_A$ und $\|e^{(k)}\|_2$ das CG minimal besser als CR, bei $\|r^{(k)}\|_2$ umgekehrt.

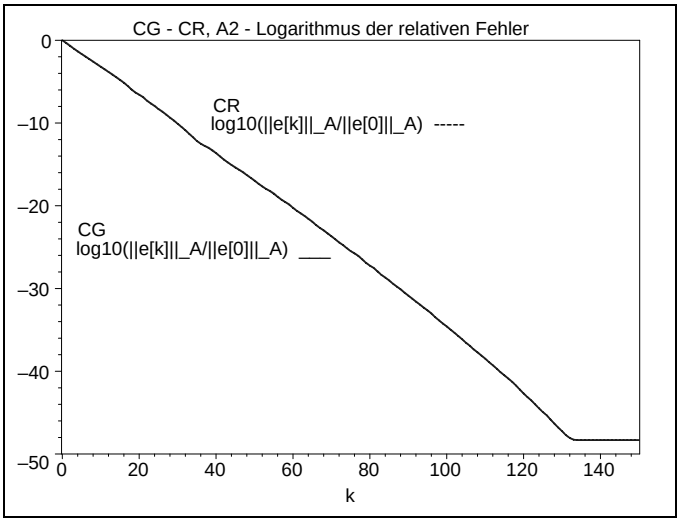


Abb. 6.93
Datei *ccgra2c161.ps*
Matrix $A2(625, 625)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)150$

Ein sehr ähnliches Bild ergibt sich zu CG versus CR für den Verlauf des relativen Fehlers $\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)150$.

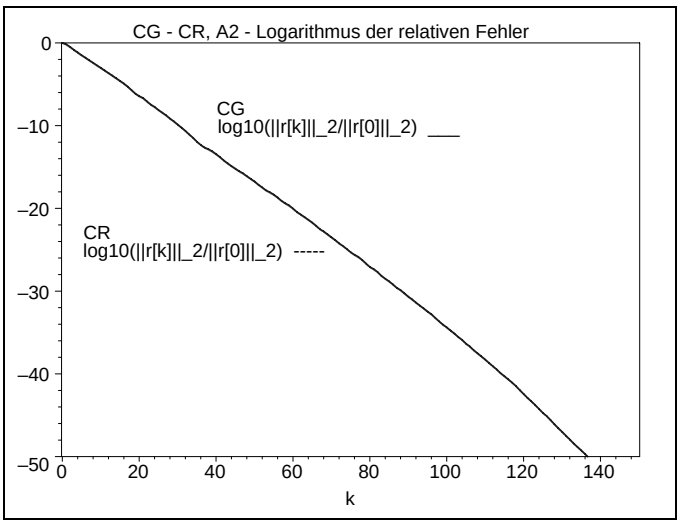


Abb. 6.94
Datei *ccgra2c163.ps*
Matrix $A2(625, 625)$, $c = 1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)150$

Mit dem Schift $c = 1$ wird die Diagonaldominanz der Matrix $A2$ verstärkt.
Damit verbessert sich die spektrale Kondition der Matrix auf $\kappa = 8.716$ und man erhält den Quotienten $q = 0.494$ deutlich kleiner als Eins im Konvergenzfaktor q^k. Somit nehmen die Fehler stetig ab.

Für die Matrix $A2(625, 625)$ mit Shift $c = 2$ folgen analoge Aussagen und Ergebnisse wie in der Situation zu $c = 1$. Die Fehler tendieren jedoch noch schneller gegen Null wegen $\kappa = 4.913$ und $q = 0.378$. Für die gleiche Genauigkeit benötigt man 105 Iterationen bei $c = 2$ statt 135 bei $c = 1$.

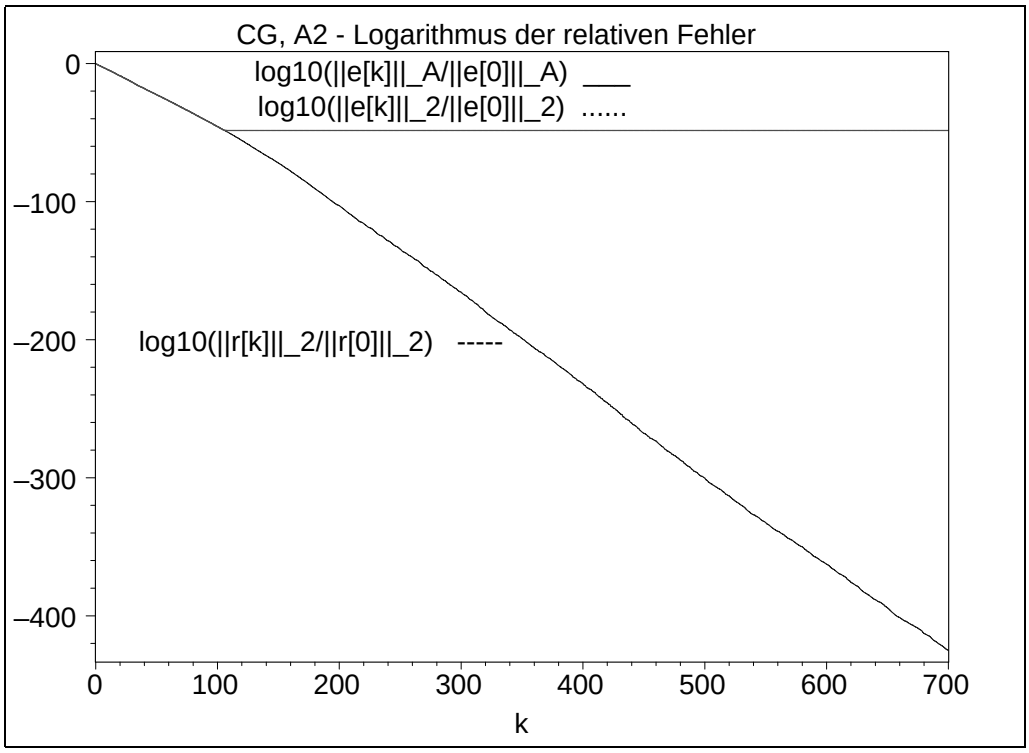


Abb. 6.95 Datei *ccga2c26.ps*, Matrix $A2(625, 625)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

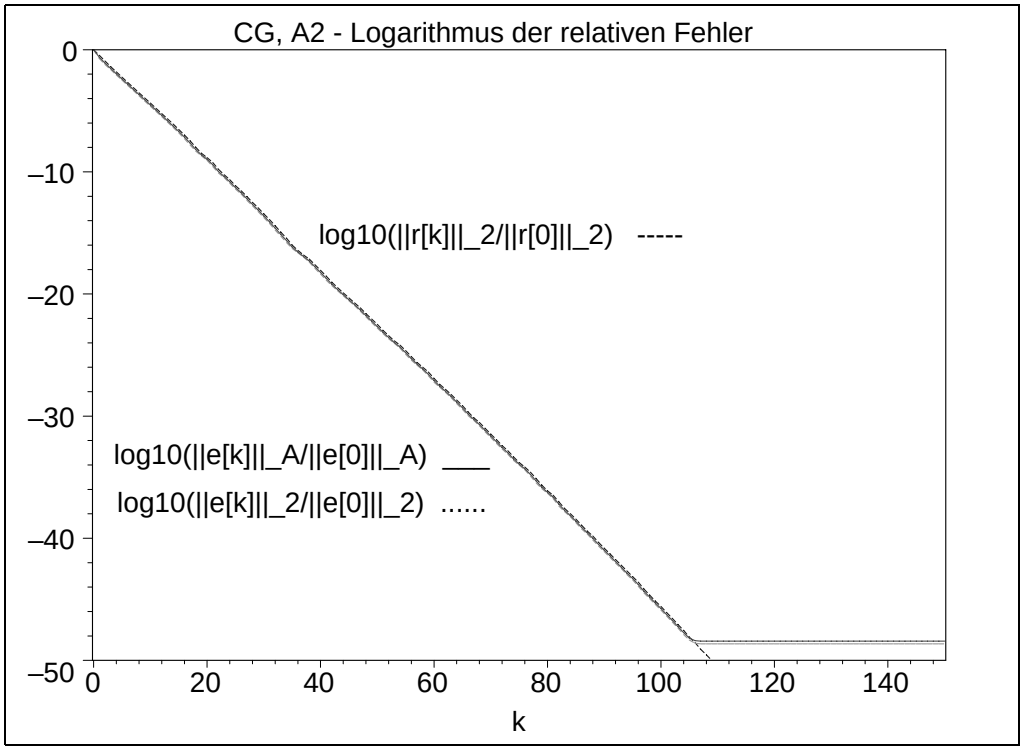


Abb. 6.96 Datei *ccga2c26a.ps*, Matrix $A2(625, 625)$, Shift $c = 2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)150$

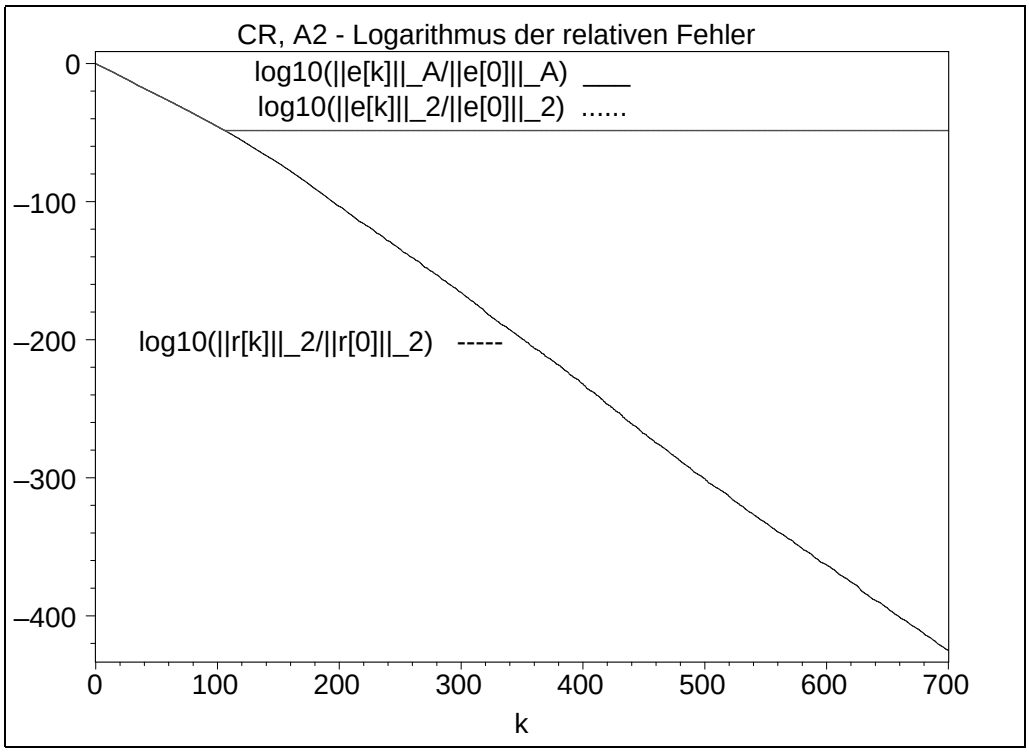


Abb. 6.97 Datei *ccra2c26.ps*, Matrix $A2(625, 625)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

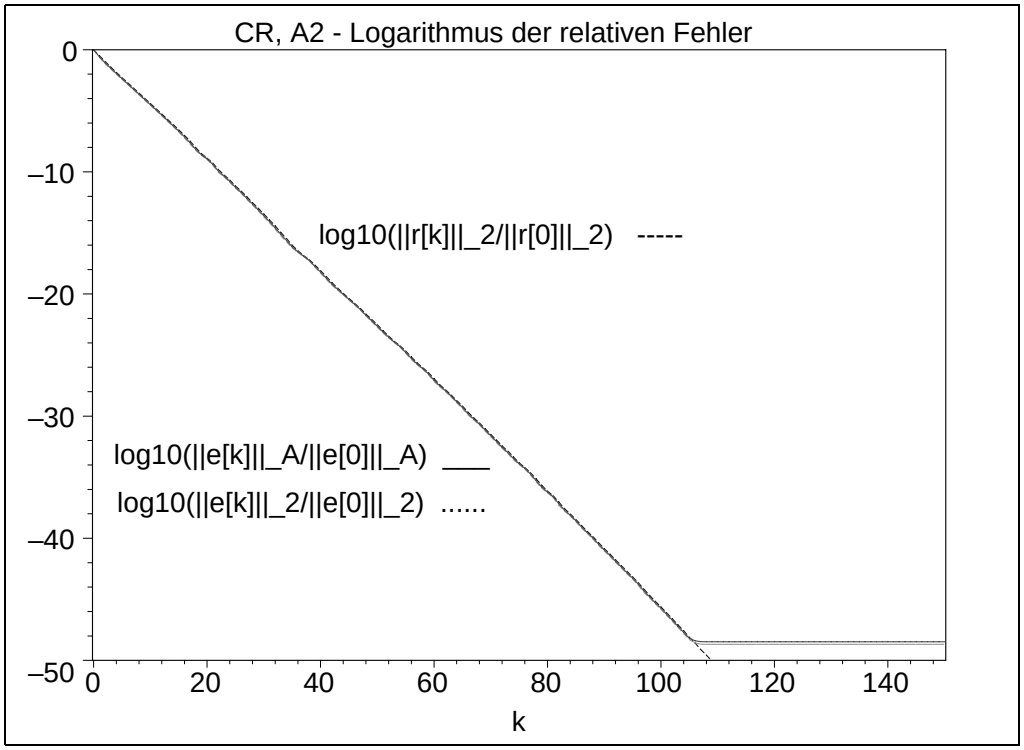


Abb. 6.98 Datei *ccra2c26a.ps*, Matrix $A2(625, 625)$, Shift $c = 2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)150$

Auch hier betrachten wir die symmetrische, aber indefinite Matrix $A2(625, 625)$ mit dem Shift $c = -1$. Die Matrix ist regulär, ihr kleinster Eigenwert ist $-0.970\,835...$, der betragskleinste $0.005\,957...$ und der größte $6.970\,835....$ Die spektrale Konditionszahl beträgt $\kappa = 1\,170.190$.

Wie bei den anderen Shifts interessieren wir uns für die Fehlerverläufe von CG und CR und hoffen, dass sie nicht vorzeitig abbrechen (durch Nulldivision bei Schrittzahl) bzw. divergieren. Dabei kann man beim CG kein monoton abnehmendes Verhalten der Fehler erwarten. Aber, wenn es nicht vorzeitig abbricht, ist es nach spätestens n Schritten (theoretisch) an der Lösung. Für das CR erhält man eine stetig abnehmende Folge von Fehlern $\|r^{(k)}\|_2$ und damit auch von $\|e^{(k)}\|_2$ mit spätestens $\|e^{(n)}\|_2 = 0,\ \|r^{(n)}\|_2 = 0$ (theoretisch). Die (nicht monotonen) Genauigkeitszuwächse verlaufen bis $n/3$ langsam, dann findet bis zu ungefähr $n/2$ eine deutliche Fehlerverkleinerung statt und die AV könnten dort schon beendet werden.

Ergebnisse aus Berechnungen mit Maple

Startvektor $x^{(0)} = (1, 0, 0, ..., 0)^T$, die rechte Seite b wird so gewählt, dass die exakte Lösung $x^* = (1, 1, ..., 1)^T$ ist, $Q(x^*) = -\frac{1}{2}x^{*T}b = 262.5,\ R(x^*) = -\frac{1}{2}b^Tb = -266.5$.

CG : Iterationsverlauf mit verschiedenen Fehlern

i	Q(x[i])= (||e[i]||^2_A-xs'b)/2	||e[i]||_A=||xs-x[i]||_A =sqrt(2Q(x[i])+xs'b)	||e[i]||_2= ||xs-x[i]||_2	||r[i]||_2= ||b-Ax[i]||_2
0	5.0000000000000000e-01	2.2891046284519194e+01	2.4979991993593593e+01	2.3194827009486404e+01
1	3.6412311557788945e+02	1.4256445249632844e+01	1.3799212740799812e+01	2.7155511084828164e+01
2	2.7890600897893404e+02	5.7281775424534529e+00	7.8152837586298128e+00	1.0136849271526673e+01
3	2.6165303328063944e+02	1.3015119817816225e+00	6.8685585931490625e+00	5.4579583283648835e+00
4	2.5346079493993937e+02	4.2518713668361687e+00	7.8896328807843567e+00	6.6284000347578574e+00
5	7.3737155163806157e+02	3.0817902317908063e+01	2.5005523755150053e+02	2.9385284400914876e+02
6	2.6847766604034682e+02	3.4576483454356143e+00	6.2499656771757403e+00	5.9228839233347481e+00
7	2.6329156303132896e+02	1.2582233755013161e+00	5.0328446299718658e+00	3.1470994704819448e+00
8	2.6086403345164358e+02	1.8088485554940289e+00	5.1922263335128460e+00	3.1282394608946812e+00
9	2.5510641450585664e+02	3.8454090794461278e+00	9.4627133765518735e+00	7.6346994961895579e+00
10	2.6792624549045779e+02	3.2943119131186676e+00	7.5796650601534016e+00	6.5697183259677816e+00
...				
100	2.6249989421575413e+02	1.4545394176034097e-02	7.7256539694886778e-02	2.0538275421704840e-02
200	2.6250000000000000e+02	2.7028021502383069e-10	2.7551834618656060e-10	3.8871640455545756e-10
300	"	1.0461669907175971e-37	2.4951436761032237e-37	1.7562353104982126e-37
...				
312	"	8.8927866875753823e-40	2.0366034152922196e-39	5.6030119247399507e-40
314	"	2.3382771222901942e-39	1.2295760952201635e-38	1.2144099006221374e-38
316	"	6.2780958109995400e-41	3.7426470015473519e-41	1.1177522520639691e-40
318	"	2.5589392980382011e-42	1.9173096661905205e-42	4.5517915648997081e-42
320	"	4.8887284620229962e-43	1.3030944512798586e-42	2.1137691411461684e-43
322	"	2.8520973568620286e-43	4.4491874923873876e-43	6.6006675198102887e-43
324	"	5.6749005502827977e-45	3.5621813824826495e-44	2.1173865289820016e-44
...				
399	"	1.0197164409775886e-46	6.3125715790001146e-46	5.8387864372079106e-50
400	"	1.0197078552212884e-46	6.3122140212448437e-46	2.6167811737579984e-50
401	"	"	"	9.7815910562654954e-51
500	"	"	"	3.4994099399227575e-75
600	"	"	"	5.0583790630684910e-95
...				
624	"	"	"	1.0742005265903856e-96
625	"	"	"	1.1563649047825915e-96
626	"	"	"	7.8582558452614581e-96
...				
700	2.6250000000000000e+02	1.0197078552212884e-46	6.3122140212448437e-46	1.491018738898050e-112

```
CR : Iterationsverlauf  mit verschiedenen Fehlern
 i         Q(x[i])=          ||e[i]||_A=        ||e[i]||_2=    ||r[i]||_2=            R(x[i])=
      (||e[i]||^2_A-xs'b)/2  ||xs-x[i]||_A=     ||xs-x[i]||_2  ||b-Ax[i]||_2=     (||r[i]||^2_2-b'b)/2
                             sqrt(2Q(x[i])+xs'b)               sqrt(2R(x[i])+b'b)
-------------------------------------------------------------------------------------------------------
  0  5.0000000000000000e-01  2.289104628e+01  2.497999199e+01  2.319482700e+01  2.5000000000000000e+00
  1  2.4256729829804353e+02  6.313905558e+00  1.418975110e+01  1.763689802e+01 -1.1096991404011461e+02
  2  2.7666539441782570e+02  5.322667454e+00  8.177146005e+00  8.788638755e+00 -2.2787991441091950e+02
  3  2.6281598860677741e+02  7.949699450e-01  6.959725977e+00  4.636605232e+00 -2.5575094595912759e+02
  4  2.5767856992457338e+02  3.105295501e+00  7.005621338e+00  3.799336650e+00 -2.5928252050994073e+02
  5  2.5783890931467487e+02  3.053224749e+00  6.990369084e+00  3.799019124e+00 -2.5928372684714327e+02
  6  2.6313714632722952e+02  1.128845717e+00  6.058302022e+00  3.197750981e+00 -2.6138719433059322e+02
  7  2.6325418165948293e+02  1.228154436e+00  5.297068421e+00  2.243028956e+00 -2.6398441055102524e+02
  8  2.6190659247829796e+02  1.089410410e+00  5.103388353e+00  1.822862184e+00 -2.6483858672889484e+02
  9  2.6119287809221931e+02  1.616862336e+00  5.150999030e+00  1.773025667e+00 -2.6492818999185007e+02
 10  2.6207609644362273e+02  9.207644176e-01  4.970364107e+00  1.711782875e+00 -2.6503489969336588e+02
...
100  2.6250019095695765e+02  1.954261792e-02  1.616582670e-01  7.880689373e-03 -2.6649996894736750e+02
200  2.6250000000000000e+02  2.996507877e-10  3.550387396e-10  3.038994438e-10 -2.6650000000000000e+02
300         "                3.459024000e-37  6.206265642e-37  2.615681020e-37 -2.6650000000000000e+02
...
390         "                8.510564904e-47  5.002696168e-46  7.966487387e-51 -2.6650000000000000e+02
391         "                8.511575823e-47  5.002654234e-46  6.505535981e-51 -2.6650000000000000e+02
392         "                       "                "         6.228324348e-51 -2.6650000000000000e+02
...
400         "                       "                "         5.412193432e-51 -2.6650000000000000e+02
500         "                8.511078721e-47  5.003758770e-46  6.623913395e-75 -2.6650000000000000e+02
600         "                       "                "         1.234945572e-95 -2.6650000000000000e+02
625         "                       "                "         1.303985059e-97 -2.6650000000000000e+02
700  2.6250000000000000e+02  8.511078721e-47  5.003758770e-46  7.93614812e-112 -2.6650000000000000e+02
```

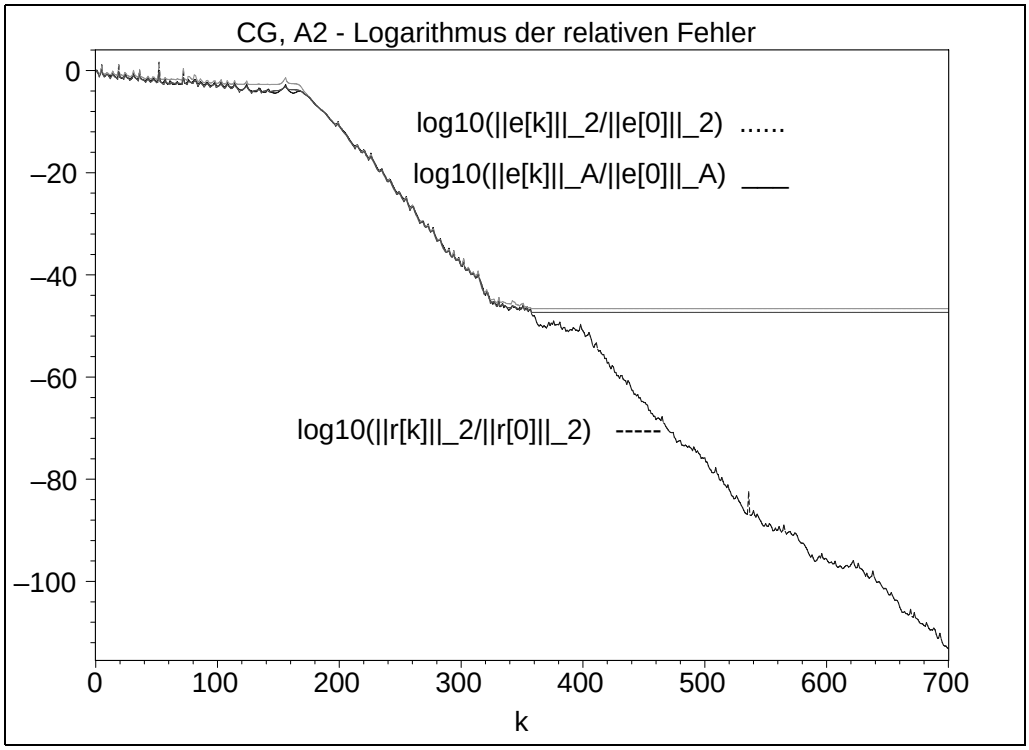


Abb. 6.99 Datei *ccga*2*cm*16.*ps*, Matrix $A2(625, 625)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

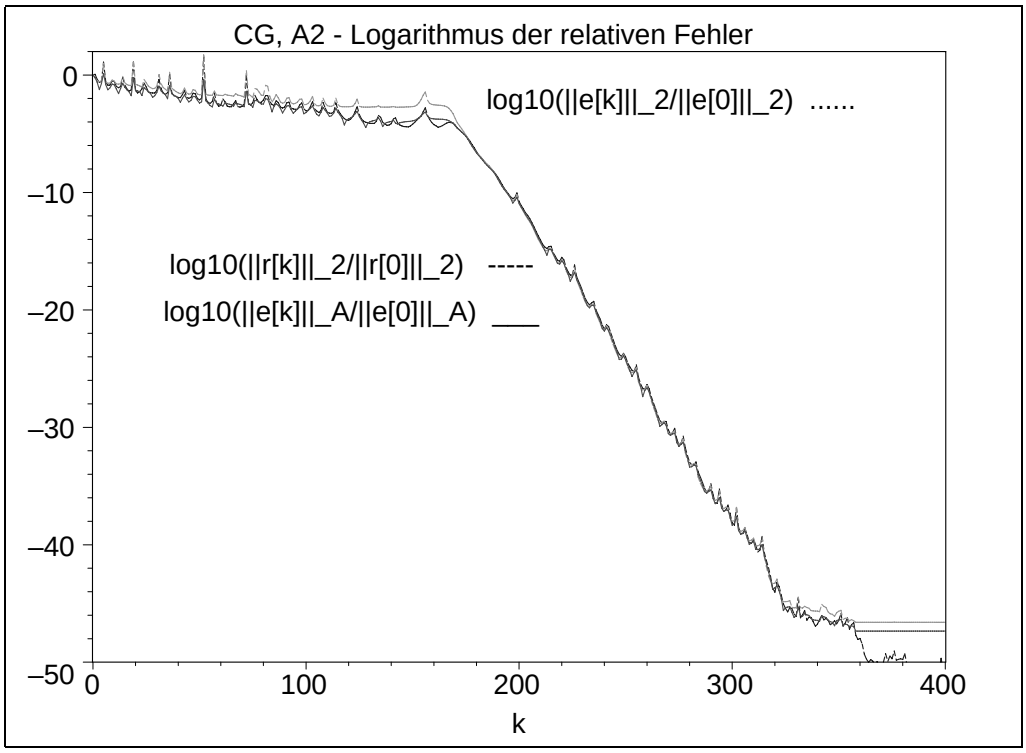


Abb. 6.100 Datei $ccga2cm16a.ps$, Matrix $A2(625, 625)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)400$

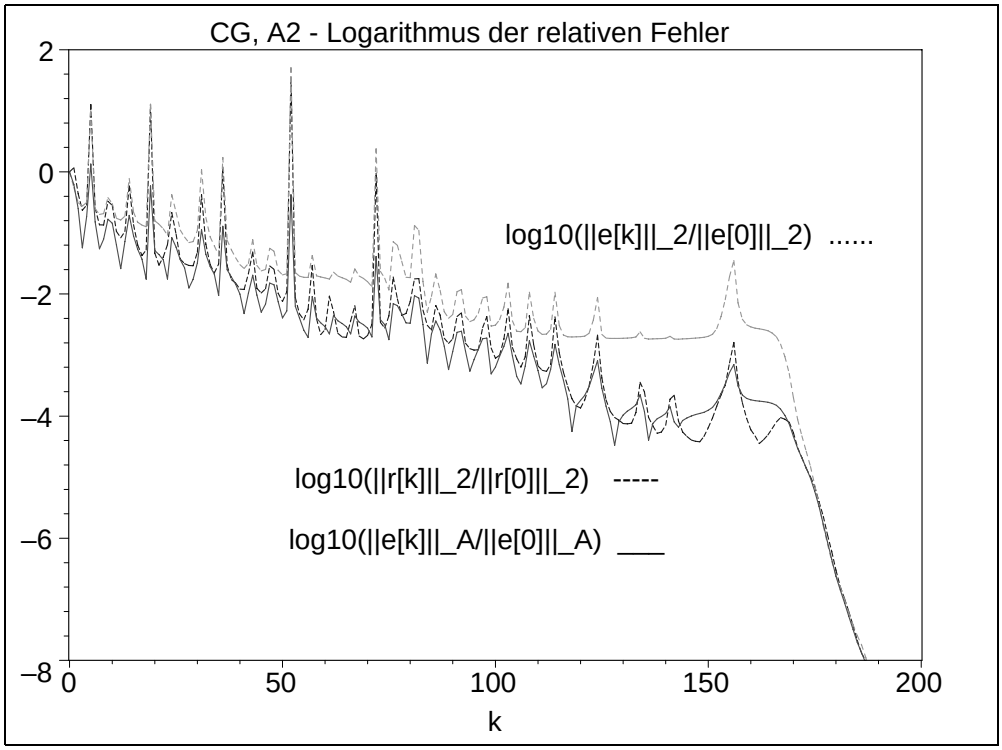


Abb. 6.101 Datei $ccga2cm16aa.ps$, Matrix $A2(625, 625)$, Shift $c = -1$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)200$

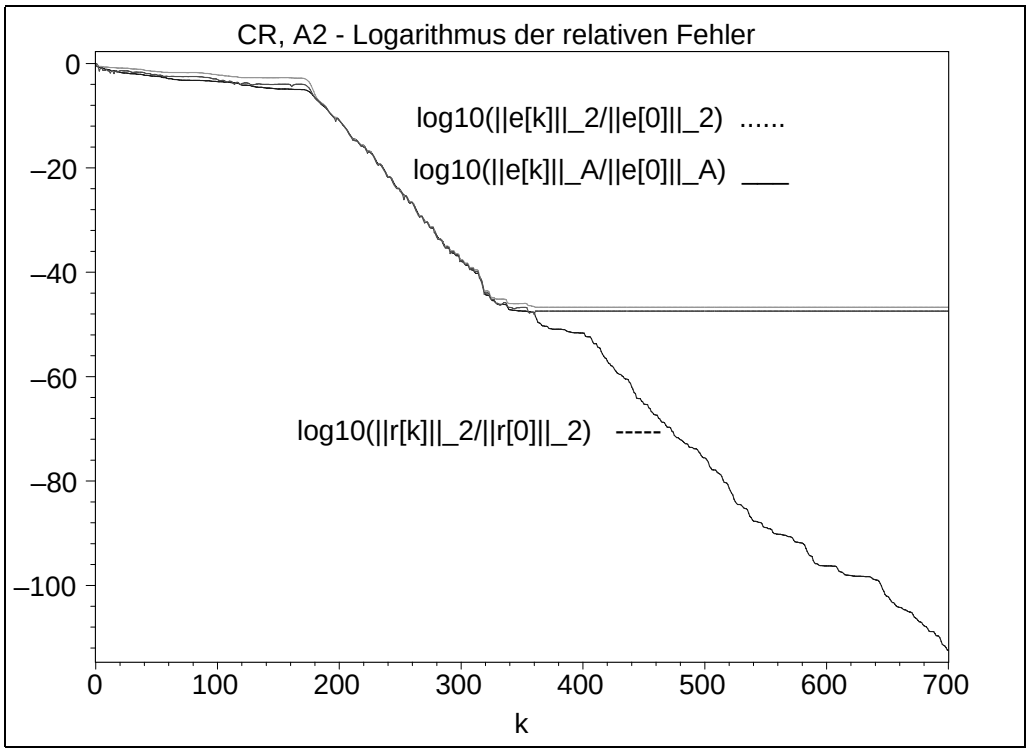


Abb. 6.102 Datei $ccra2cm16.ps$, Matrix $A2(625, 625)$, Shift $c = -1$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

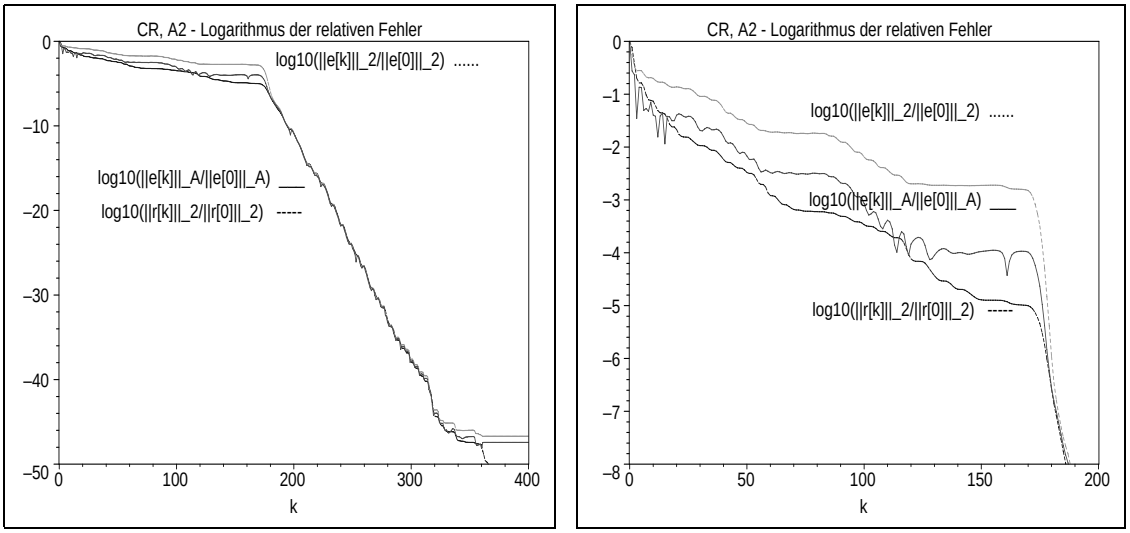


Abb. 6.103 Dateien $ccra2cm16a.ps$, $ccra2cm16aa.ps$, Matrix $A2(625, 625)$, $c=-1$,
CR: Verlauf der 3 relativen Fehler für $k = 0(1)400$ bzw. $k = 0(1)200$

Man bemerke auch im CR den unregelmäßigen Verlauf der Fehler $\|e^{(k)}\|_A$.

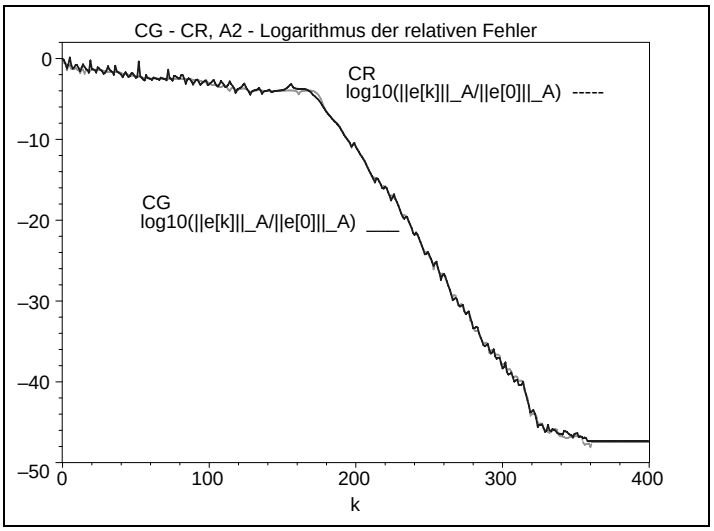


Abb. 6.104
Datei $ccgra2cm161.ps$
Matrix $A2(625, 625)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)400$

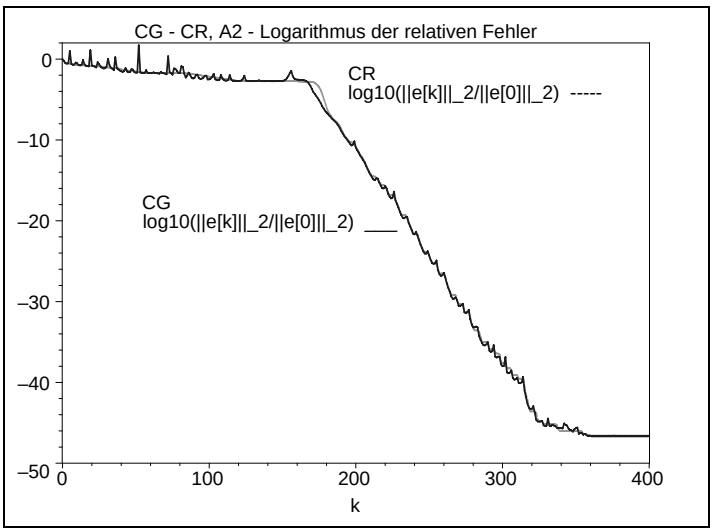


Abb. 6.105
Datei $ccgra2cm162.ps$
Matrix $A2(625, 625)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)400$

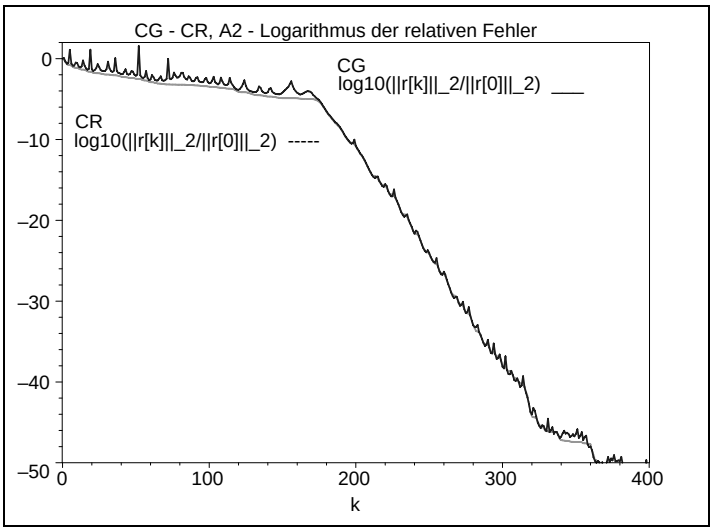


Abb. 6.106
Datei $ccgra2cm163.ps$
Matrix $A2(625, 625)$, $c=-1$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)400$

Als Ergänzung noch die symmetrische, aber indefinite Matrix $A2(625, 625)$ mit dem Shift $c = -2$. Glücklicherweise ist die Matrix regulär, ihr betragskleinster EW ist $-0.011\,985....$ Die spektrale Konditionszahl beträgt $\kappa = 498.175$.

Wir interessieren uns wiederum für die Fehlerverläufe von CG und CR. Beide AV brechen zum Glück nicht vorzeitig ab und brauchen für eine gute Genauigkeit ungefähr $2n/3$ Iterationen. Das Verhalten ist qualitativ ähnlich zu dem bei $c = -1$.

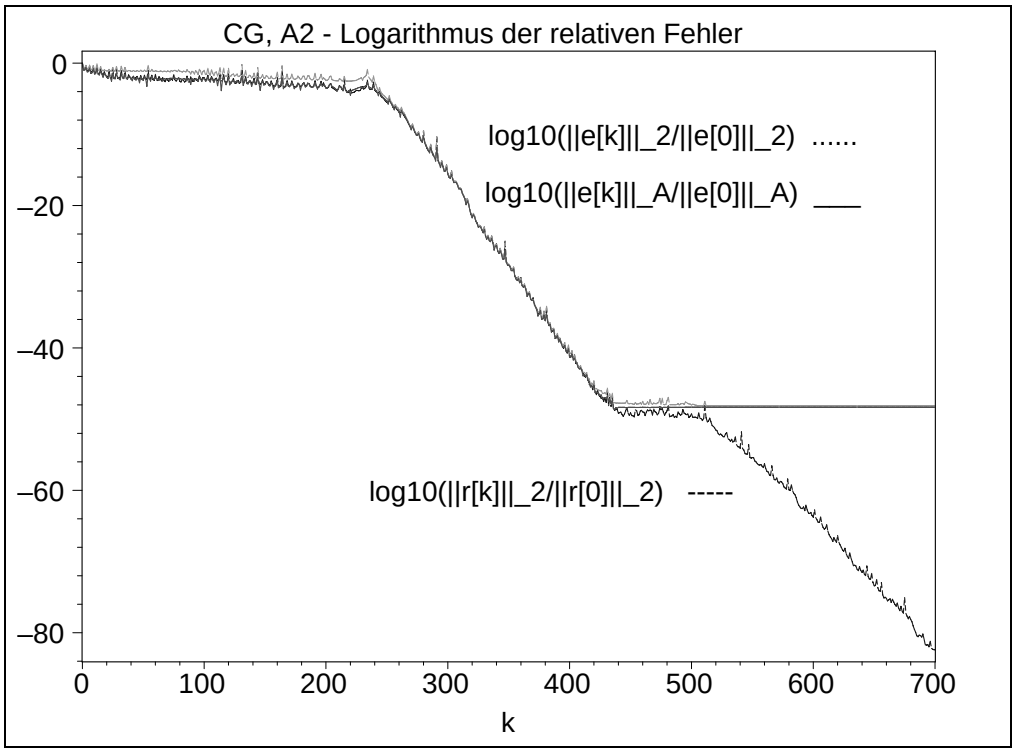


Abb. 6.107 Datei *ccga2cm26.ps*, Matrix $A2(625, 625)$, Shift $c = -2$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

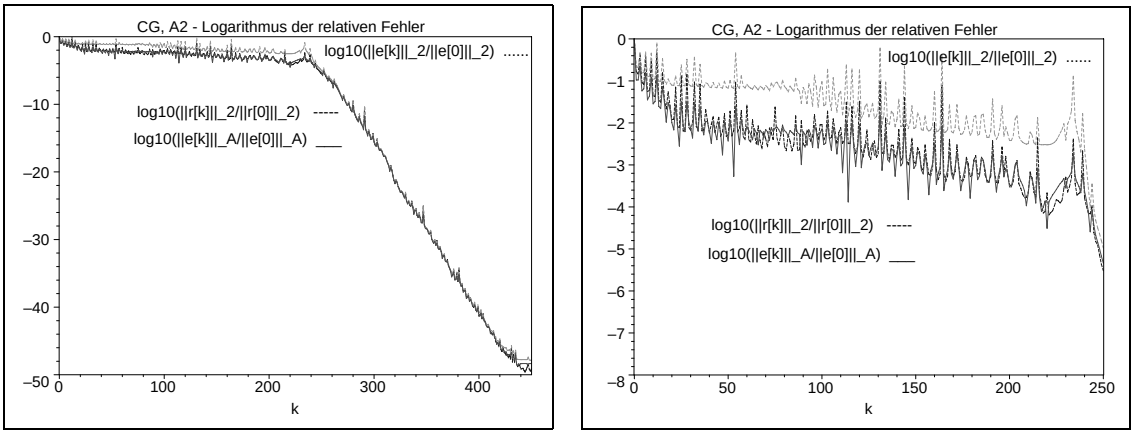


Abb. 6.108 Dateien *ccga2cm26a.ps*, *ccga2cm26aa.ps*, Matrix $A2(625, 625)$, $c = -2$,
CG: Verlauf der 3 relativen Fehler für $k = 0(1)450$ bzw. $k = 0(1)250$

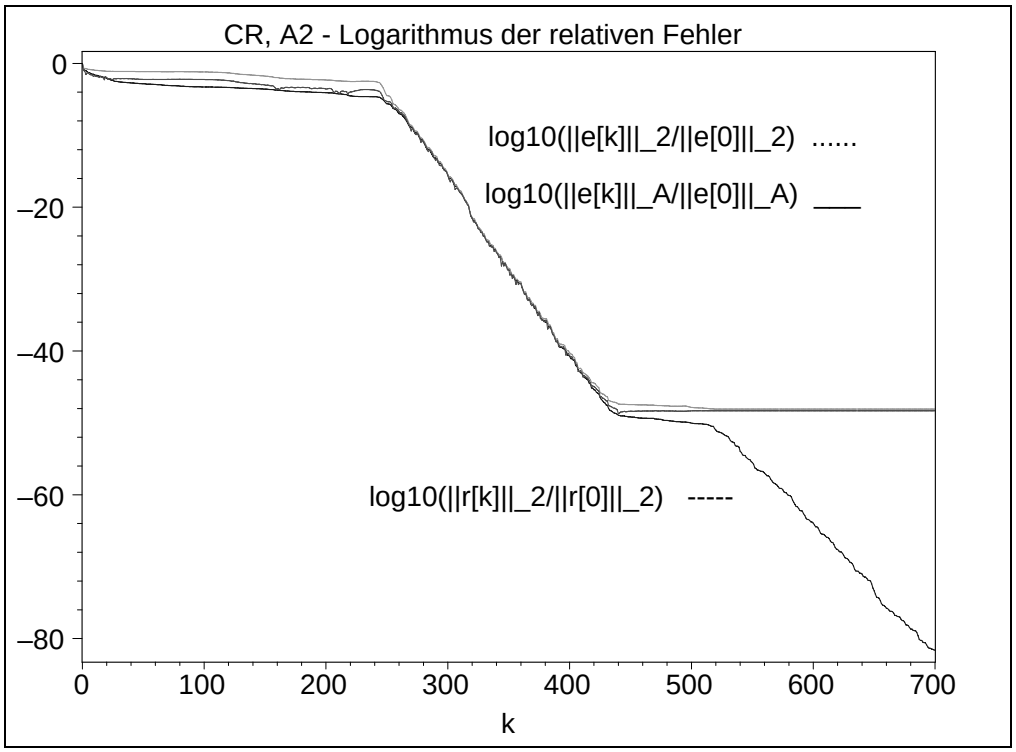


Abb. 6.109 Datei *ccra2cm26.ps*, Matrix $A2(625, 625)$, Shift $c = -2$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

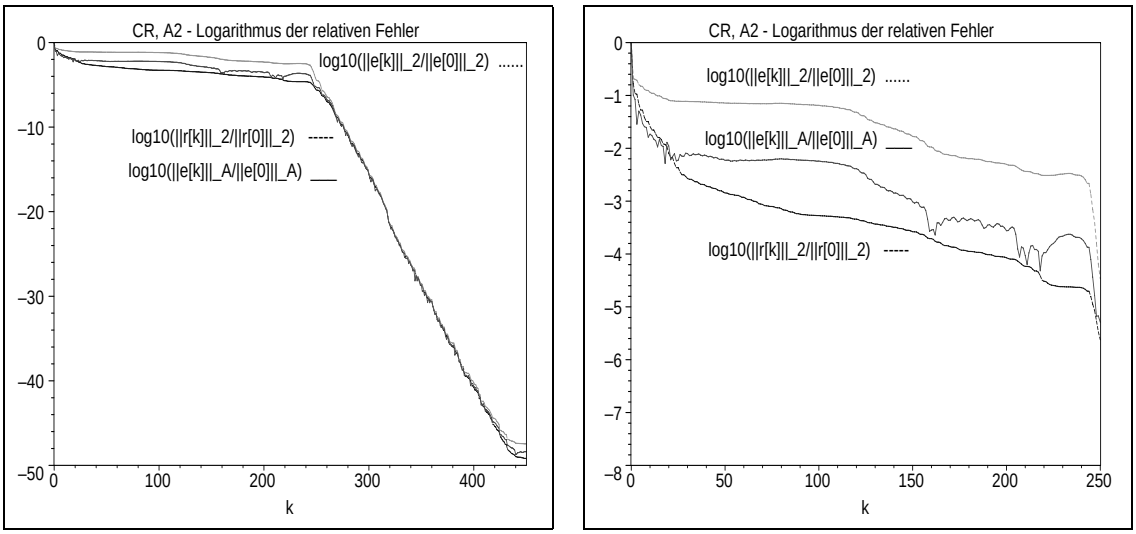


Abb. 6.110 Dateien *ccra2cm26a.ps*, *ccra2cm26aa.ps*, Matrix $A2(625, 625)$, $c = -2$,
CR: Verlauf der 3 relativen Fehler für $k = 0(1)450$ bzw. $k = 0(1)250$

Man bemerke die sichtbare Verbesserung der AV nach etwas mehr als $n/3$ Iterationen und auch im CR den unregelmäßigen Verlauf der Fehler $\|e^{(k)}\|_A$.

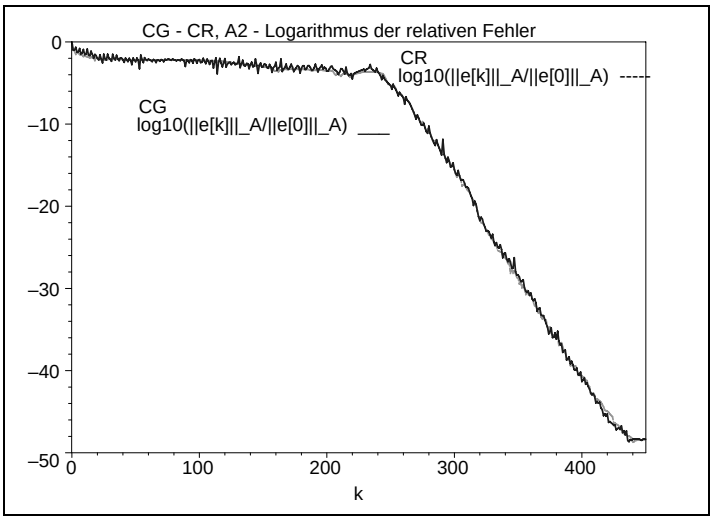


Abb. 6.111
Datei *ccgra2cm261.ps*
Matrix $A2(625, 625)$, $c{=}-2$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $k = 0(1)450$

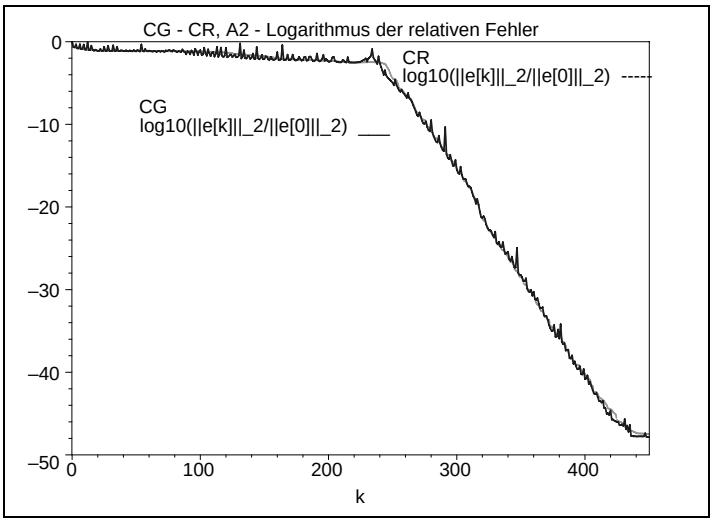


Abb. 6.112
Datei *ccgra2cm262.ps*
Matrix $A2(625, 625)$, $c{=}-2$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2})$, $k = 0(1)450$

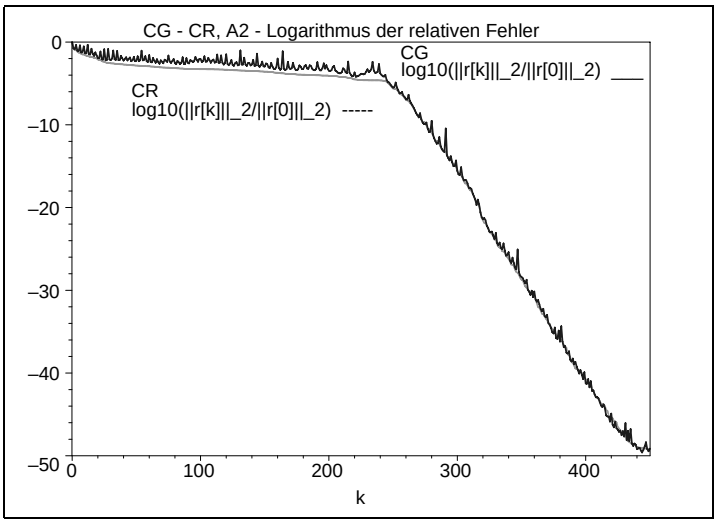


Abb. 6.113
Datei *ccgra2cm263.ps*
Matrix $A2(625, 625)$, $c{=}-2$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0(1)450$

Für die symmetrische Matrix $A2(625, 625)$ wählen wir nun den Shift c so, dass die Kondition möglichst schlecht wird.

Man kann zwei wesentliche Fälle unterscheiden.

(1) Sei $A2$ spd. Dann erzeugt das CG eine monoton fallende Folge von Werten $\|e^{(k)}\|_A$ und das gilt auch für $\|e^{(k)}\|_2$, während sich dieses Eigenschaft nicht auf die Residuumnorm $\|r^{(k)}\|_2$ überträgt. Das CR erzeugt monoton fallende Folgen von Werten $\|r^{(k)}\|_2$ und $\|e^{(k)}\|_2$ und das gilt dann meistens auch für die Fehlernorm $\|e^{(k)}\|_A$. Wird jedoch bei der spd Matrix die Kondition immer schlechter, so entstehen zwei zusätzliche Effekte. Bei den Fehlern taucht in gewissen Bereichen eine Stagnation auf ("Plateau-Situation" für $\|e^{(k)}\|_A$ im CG bzw. für $\|r^{(k)}\|_2$ im CR). Außerdem gibt es im CG für den Fehler $\|r^{(k)}\|_2$ auffällige Spitzen und Senken, ausgeprägter als bei gut konditionierten Matrizen.

(2) Sei $A2$ symmetrisch und indefinit. Dann erzeugt das CG oszillierende Folgen von Werten $\|e^{(k)}\|_A$, $\|e^{(k)}\|_2$ und $\|r^{(k)}\|_2$. Das CR erzeugt eine monoton fallende Folge von Werten $\|r^{(k)}\|_2$ und das gilt dann meistens auch für die Fehlernorm $\|e^{(k)}\|_2$, jedoch nicht für die Größen $\|e^{(k)}\|_A$, die zum Teil recht unregelmäßigen Verlauf haben. Kommt noch die schlechte Kondition der Matrix hinzu, dann überlagern sich "Plateau-Eigenschaft" und Oszillation.

Betrachten wir 3 geschiftete symmetrische Matrizen $A2$ mit einer Kondition κ schlechter als 10^5.

	$A2$ spd $c = -0.029\,1$	$A2$ indefinit $c = -0.029\,228$	$A2$ indefinit $c = -1.005\,9$
EW λ			
λ_{min}	0.000 064 503	−0.000 063 497	−0.976 735 497
	0.043 598 616	0.043 470 616	−0.933 201 383
			...
	...	...	−0.012 061 978
	...	...	0.000 057 007
			...
	7.898 201	7.898 073	6.921 401
λ_{max}	7.941 734	7.941 606	6.964 935
$\kappa \approx$	124 120	125 071	122 176
q	0.994 316	0.994 360	0.994 294

Tab. 6.16 Charakteristika von 3 Matrizen $A2(625, 625)$ mit Shift c

Die Ergebnisse aus Berechnungen mit Maple stellen wir grafisch dar.
Der Startvektor sei $x^{(0)} = (1, 0, 0, ..., 0)^T$, die rechte Seite b wird so gewählt, dass die exakte Lösung $x^* = (1, 1, ..., 1)^T$ ist.

Der Shift sei $c = -0.029\,1$. Die Matrix bleibt spd. Ihr kleinster Eigenwert ist $0.000\,064\,503$.
Die spektrale Konditionszahl beträgt $\kappa \approx 124\,120$.

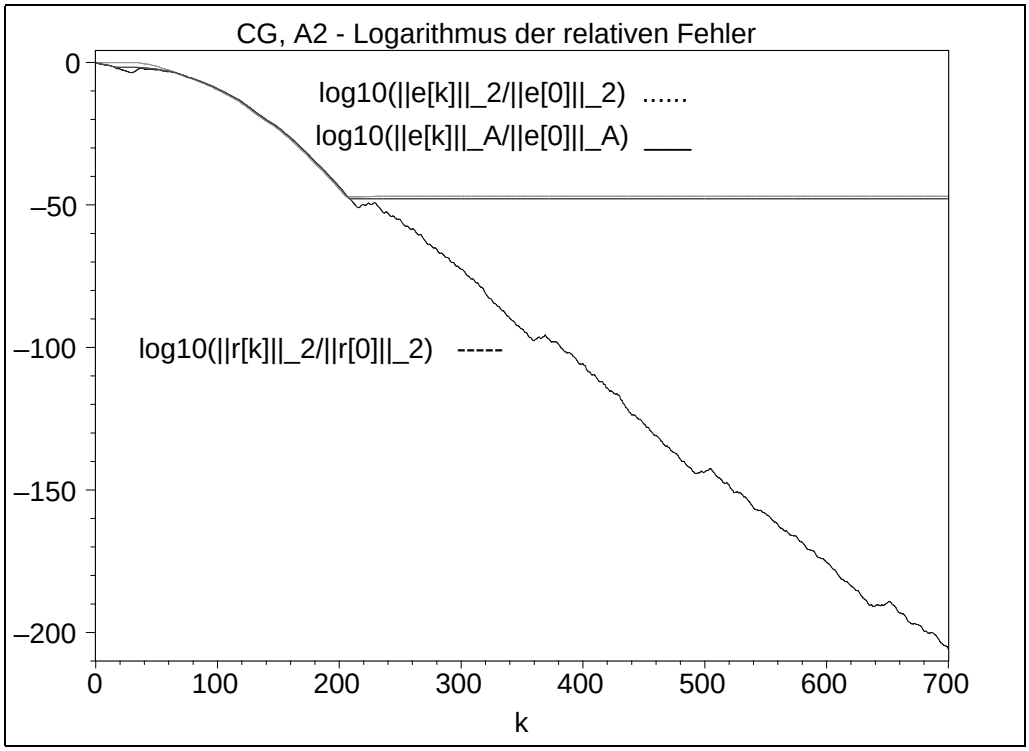
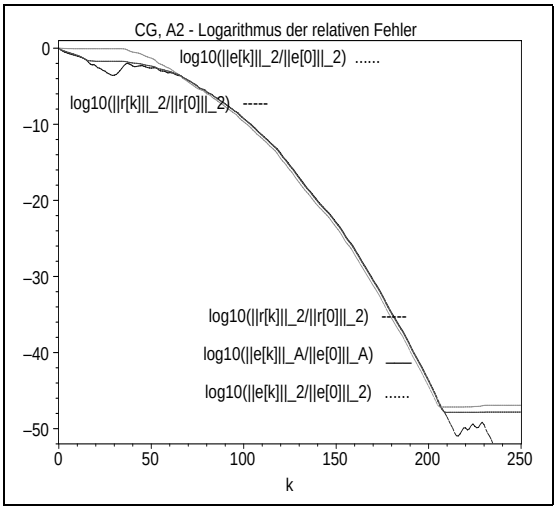


Abb. 6.114 Datei *ccga2cm06.ps*, Matrix $A2(625, 625)$, Shift $c = -0.0291$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$



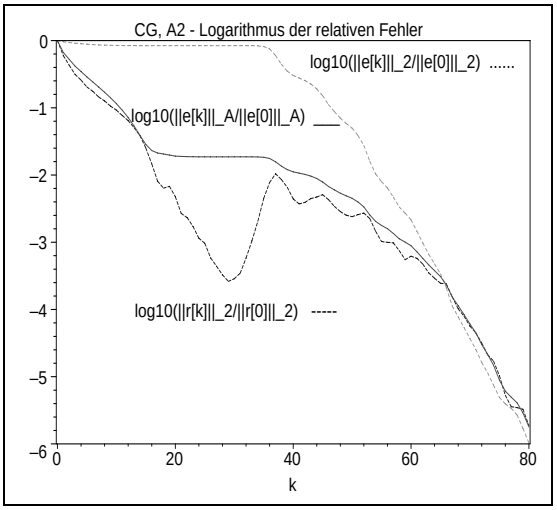


Abb. 6.115 Dateien *ccga2cm06a.ps*, *ccga2cm06aa.ps*, Matrix $A2$, $c = -0.0291$,
CG: Verlauf der 3 relativen Fehler für $k = 0(1)250$ bzw. $k = 0(1)80$

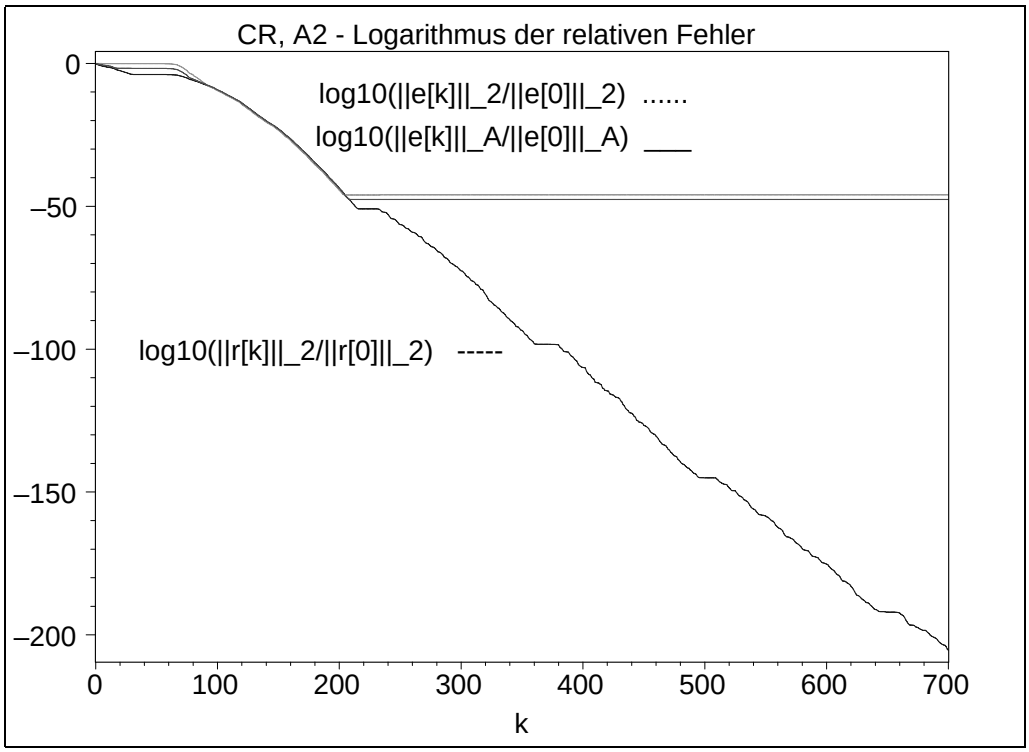


Abb. 6.116 Datei $ccra2cm06.ps$, Matrix $A2(625, 625)$, Shift $c = -0.0291$,
CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

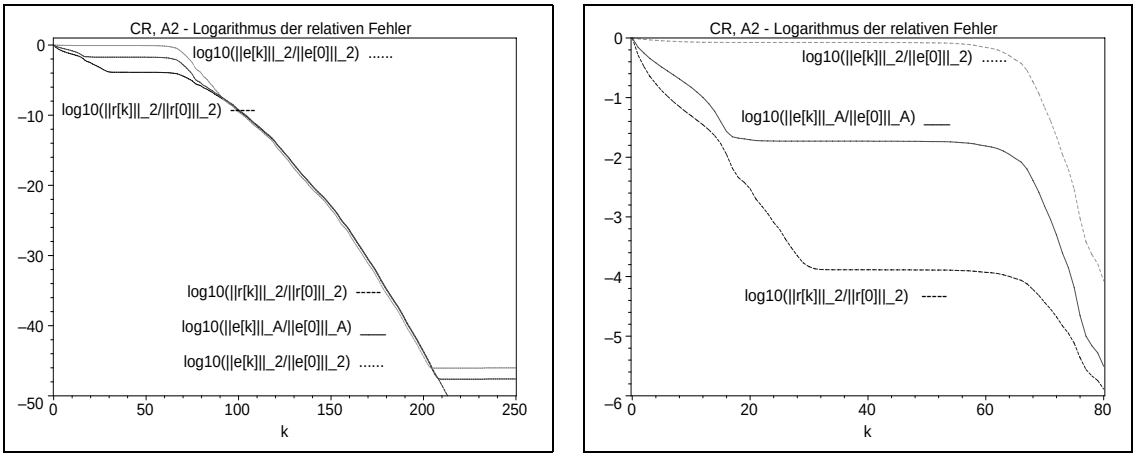


Abb. 6.117 Dateien $ccra2cm06a.ps$, $ccra2cm06aa.ps$, Matrix $A2$, $c = -0.0291$,
CR: Verlauf der 3 relativen Fehler für $k = 0(1)250$ bzw. $k = 0(1)80$

Beide AV CG und CR brauchen für eine gute Genauigkeit etwa $n/3$ Iterationen, eine sichtbare Verbesserung bemerkt man ungefähr bei $n/8$ Iterationen.
Das Konvergenzverhalten ist nicht ganz so gut wie bei $c = 0$. Die schlechte Kondition der spd Matrix verursacht bei den Fehlern im CG und CR die genannte "Plateau-Situation" sowie im CG für den Fehler $\|r^{(k)}\|_2$ auffällige Spitzen und Senken, ausgeprägter als bei $c = 0$.

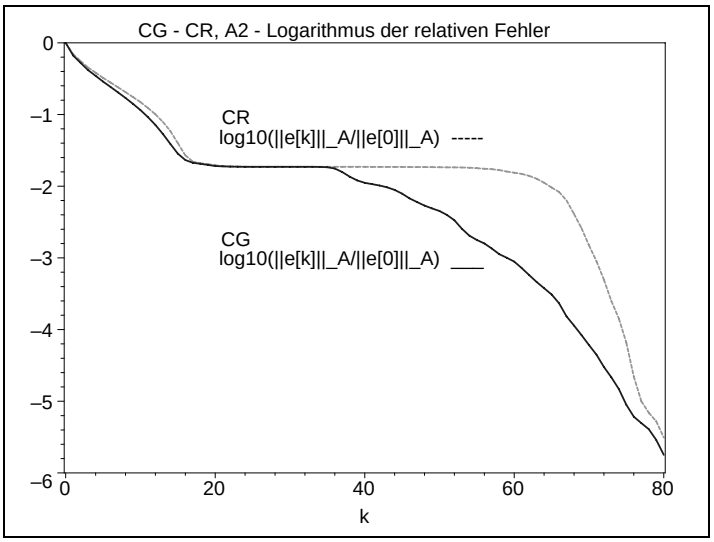


Abb. 6.118
Datei $ccgra2cm061.ps$
Matrix $A2(625, 625)$,
$c = -0.0291$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ k = 0(1)80$

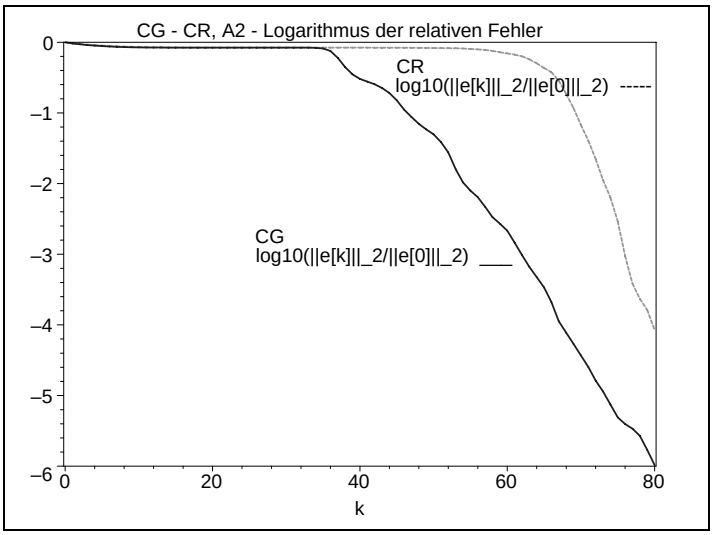


Abb. 6.119
Datei $ccgra2cm062.ps$
Matrix $A2(625, 625)$,
$c = -0.0291$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2}),\ k = 0(1)80$

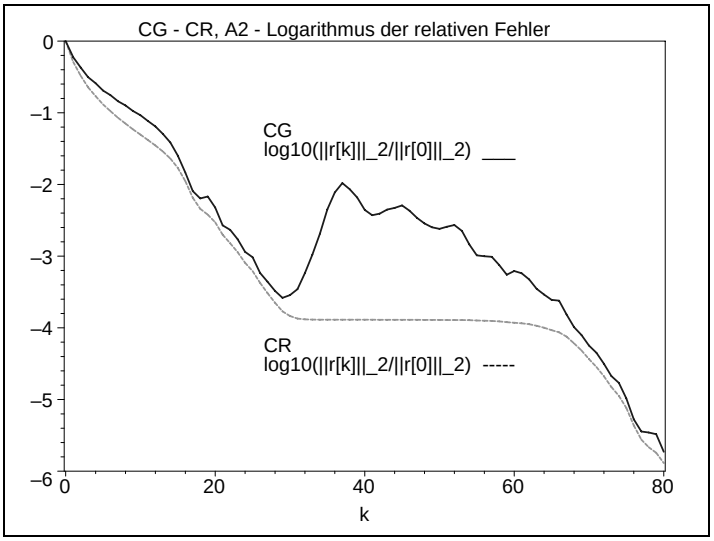


Abb. 6.120
Datei $ccgra2cm063.ps$
Matrix $A2(625, 625)$,
$c = -0.0291$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0(1)80$

Der Shift sei $c = -0.029\,228$. Die Matrix wird indefinit. Sie besitzt jedoch nur einen negativen EW $-0.000\,063\,497$, der auch der betragskleinste ist. Der nächste größere ist $0.043\,470\,616$.
Die spektrale Konditionszahl beträgt $\kappa \approx 125\,071$.

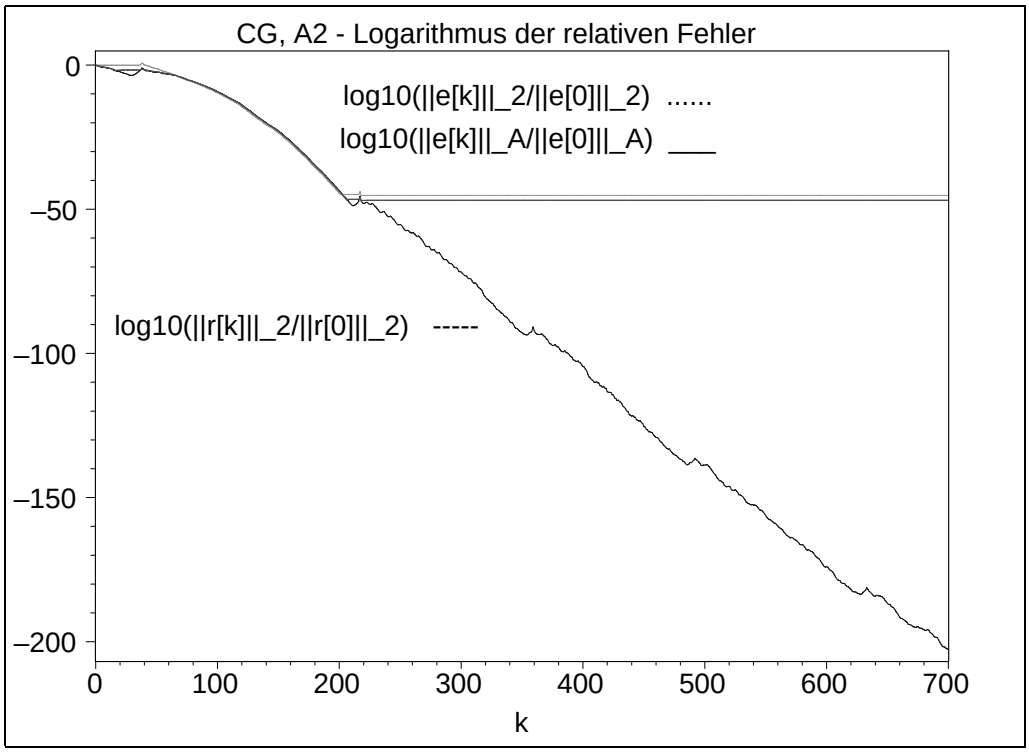
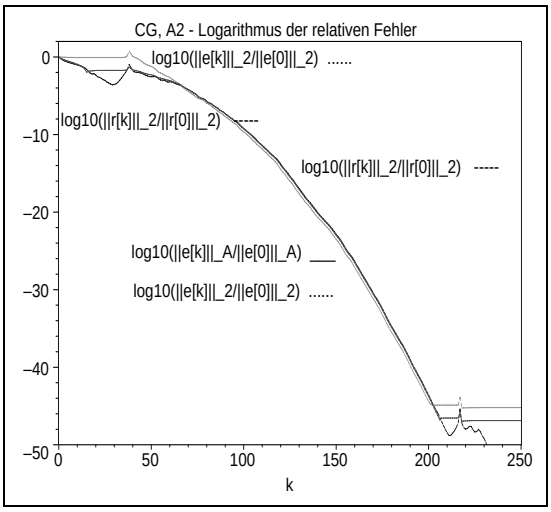


Abb. 6.121 Datei *ccga2cm46.ps*, Matrix $A2(625, 625)$, Shift $c = -0.029228$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$



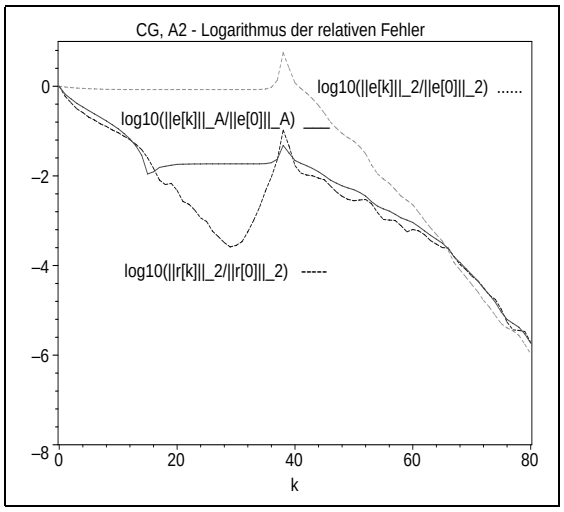


Abb. 6.122 Dateien *ccga2cm46a.ps*, *ccga2cm46aa.ps*, Matrix $A2$, $c = -0.029228$,
CG: Verlauf der 3 relativen Fehler für $k = 0(1)250$ bzw. $k = 0(1)80$

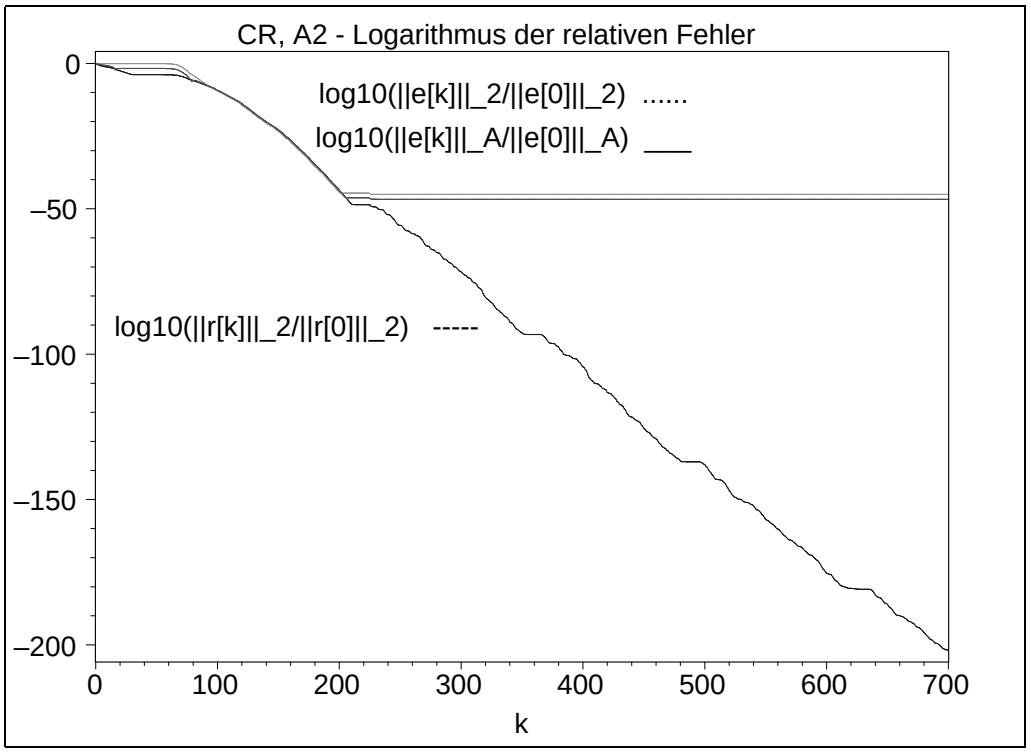


Abb. 6.123 Datei $ccra2cm46.ps$, Matrix $A2(625, 625)$, Shift $c = -0.029228$, CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

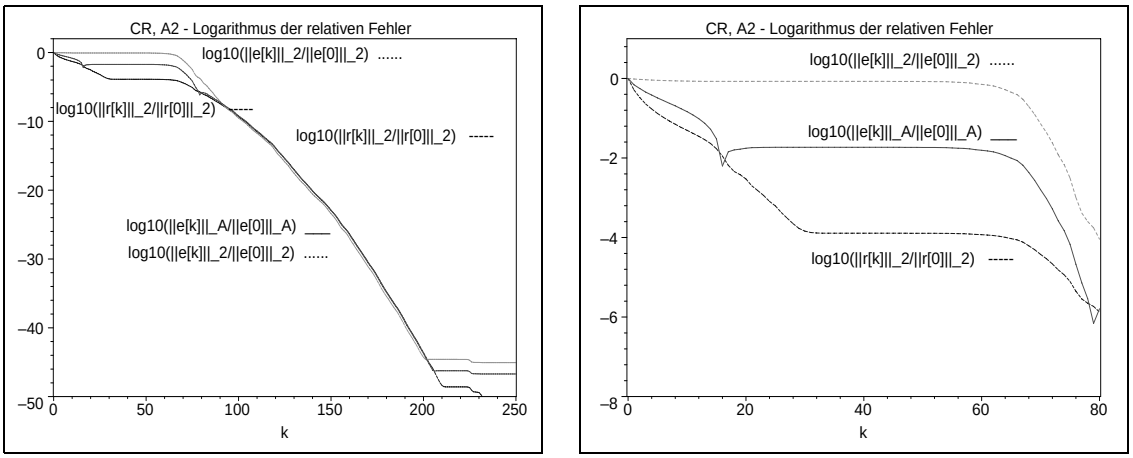


Abb. 6.124 Dateien $ccra2cm06a.ps$, $ccra2cm46aa.ps$, Matrix $A2$, $c = -0.029228$, CR: Verlauf der 3 relativen Fehler für $k = 0(1)250$ bzw. $k = 0(1)80$

Beide AV CG und CR brauchen für eine gute Genauigkeit etwa $n/3$ Iterationen, eine sichtbare Verbesserung bemerkt man ungefähr bei $n/8$ Iterationen.
Wegen der Indefinitheit der Matrix bemerkt man das beginnende oszillierende bzw. unregelmäßige Verhalten der Fehler, überlagert durch die genannte "Plateau-Situation" im Konvergenzprozess.

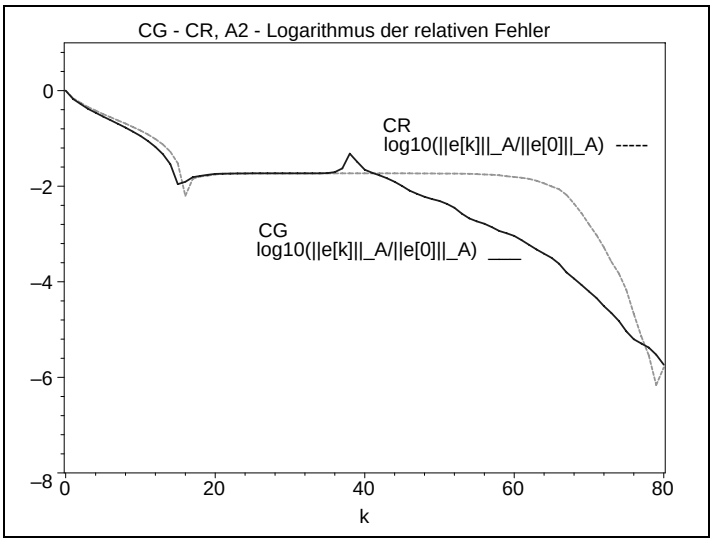


Abb. 6.125
Datei $ccgra2cm461.ps$
Matrix $A2(625, 625)$,
$c = -0.029228$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ k = 0(1)80$

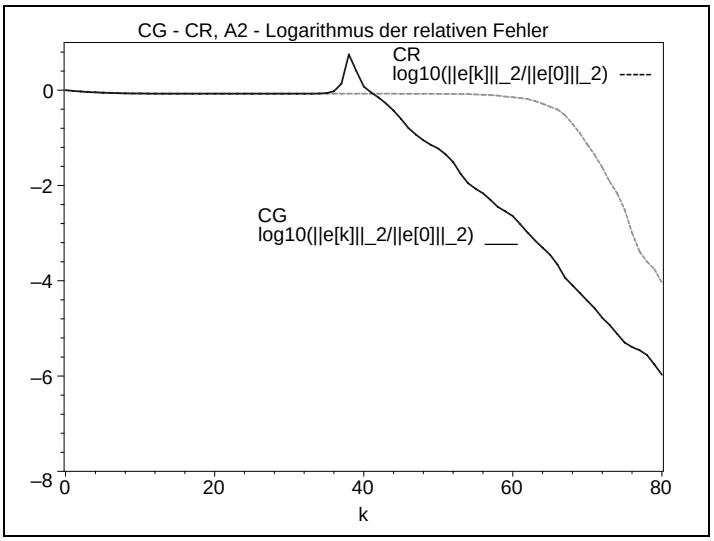


Abb. 6.126
Datei $ccgra2cm462.ps$
Matrix $A2(625, 625)$,
$c = -0.029228$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2}),\ k = 0(1)80$

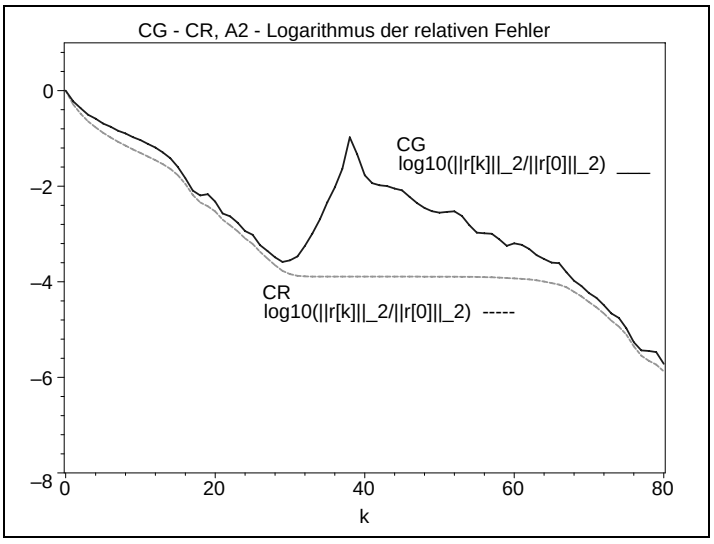


Abb. 6.127
Datei $ccgra2cm463.ps$
Matrix $A2(625, 625)$,
$c = -0.029228$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0(1)80$

Der Shift sei $c = -1.005\,9$. Die Matrix ist indefinit. Sie besitzt sowohl negative als auch positive EW. Ihr kleinster Eigenwert ist $-0.976\,735...$, der betragskleinste $0.000\,057...$, der größte $6.964\,935...$.
Die spektrale Konditionszahl beträgt $\kappa \approx 122\,176$.

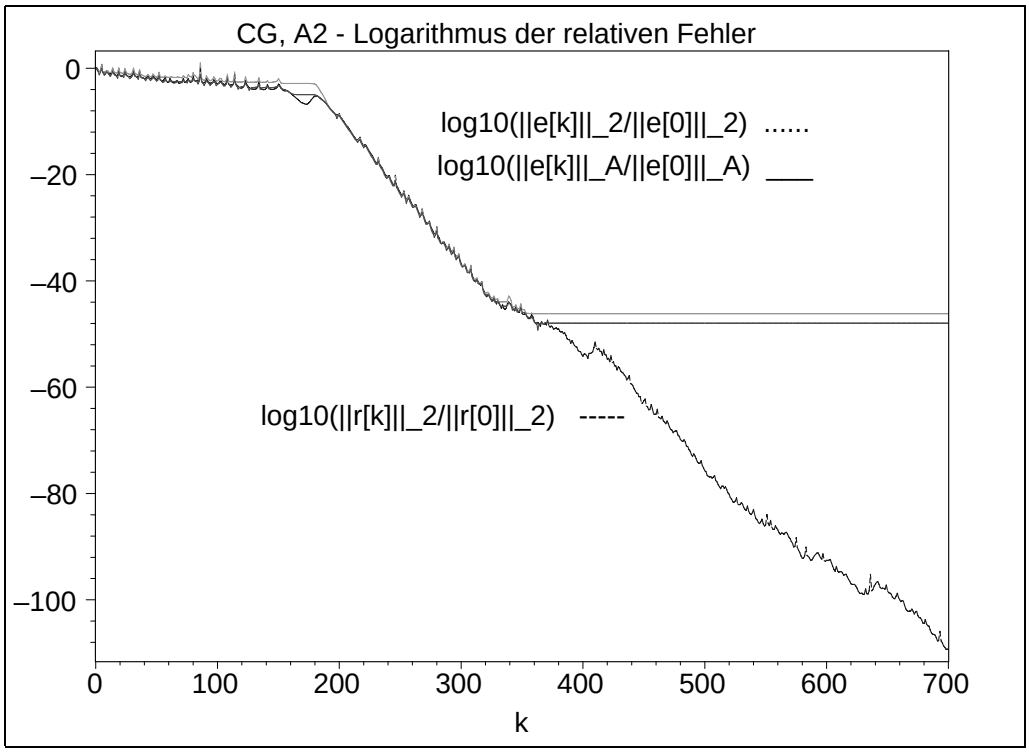


Abb. 6.128 Datei *ccga2cm36.ps*, Matrix $A2(625, 625)$, Shift $c = -1.0059$,
CG: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

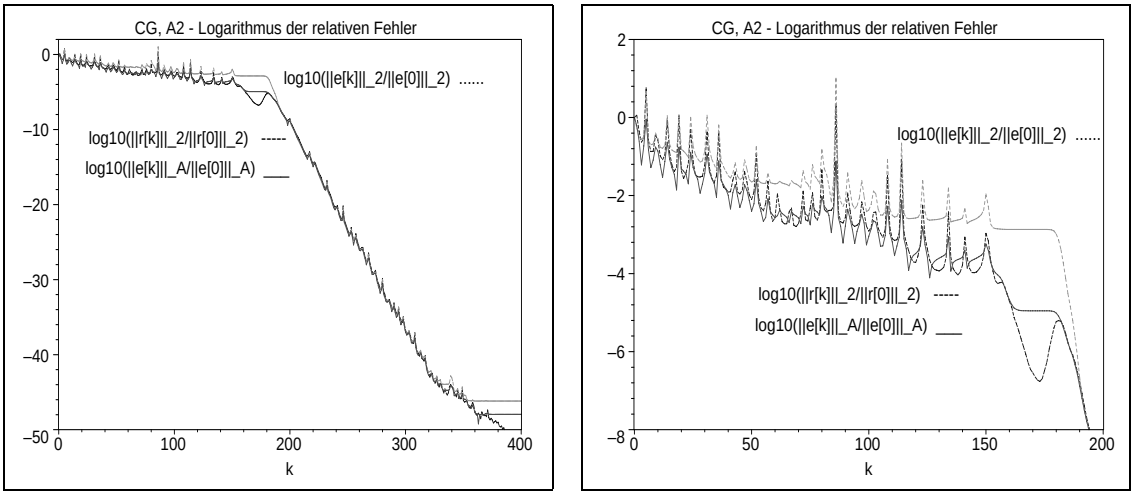


Abb. 6.129 Dateien *ccga2cm36a.ps*, *ccga2cm36aa.ps*, Matrix $A2$, $c = -1.0059$,
CG: Verlauf der 3 relativen Fehler für $k = 0(1)400$ bzw. $k = 0(1)200$

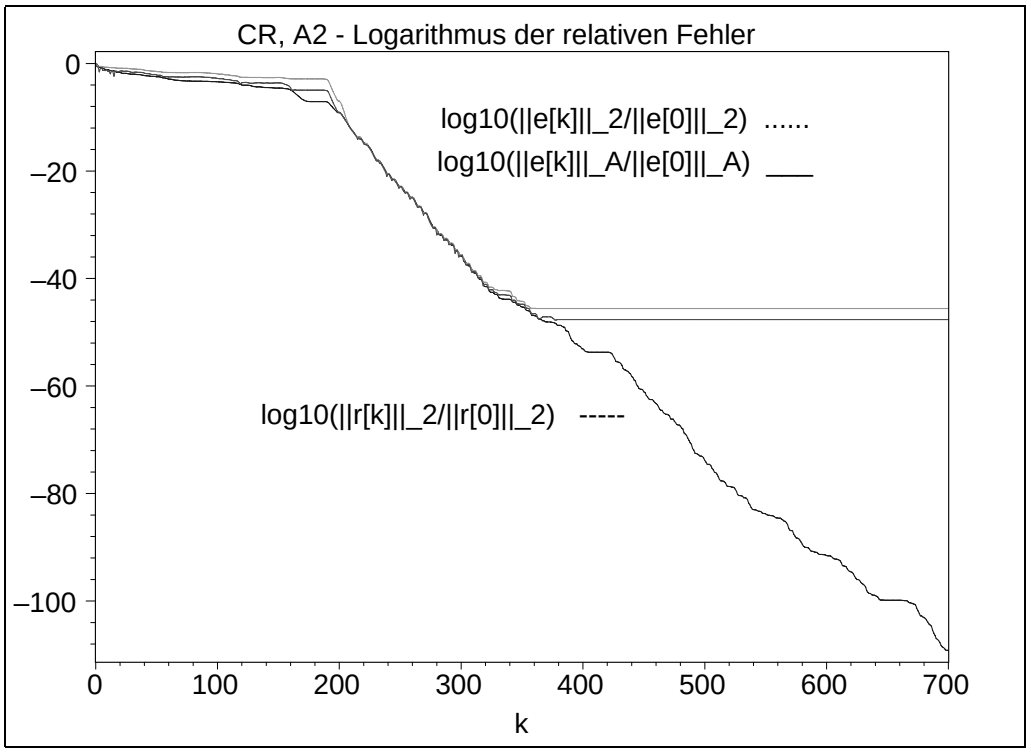


Abb. 6.130 Datei *ccra2cm36.ps*, Matrix $A2(625, 625)$, Shift $c = -1.0059$, CR: Verlauf der 3 relativen Fehler für die Iterationen $k = 0(1)700$

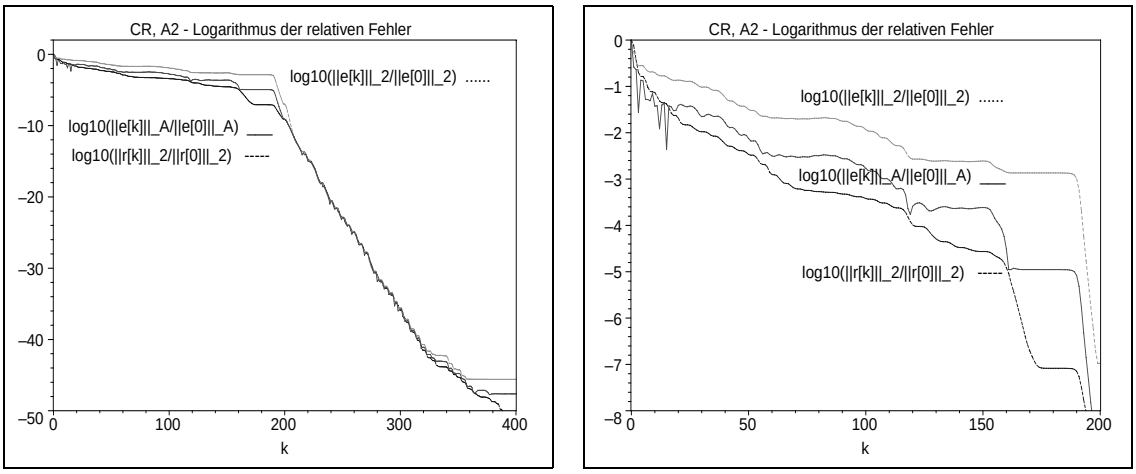


Abb. 6.131 Dateien *ccra2cm36a.ps*, *ccra2cm36aa.ps*, Matrix $A2$, $c = -1.0059$, CR: Verlauf der 3 relativen Fehler für $k = 0(1)400$ bzw. $k = 0(1)200$

Beide AV CG und CR brechen nicht vorzeitig ab und brauchen für eine gute Genauigkeit etwas mehr als $n/2$ Iterationen. Eine sichtbare Verbesserung tritt nach ungefähr $n/3$ Iterationen auf. Das Verhalten ist qualitativ ähnlich zu dem bei $c = -1,\ -2$. Man bemerke im CG die deutliche Überlagerung von Oszillation und "Plateau-Situation". Auch im CR sieht man die "Plateau-Situation" und den monotonen, aber wenig glatten Verlauf der Fehler $\|r^{(k)}\|_2$ bzw. $\|e^{(k)}\|_2$ sowie die Unregelmäßigkeiten bei $\|e^{(k)}\|_A$.

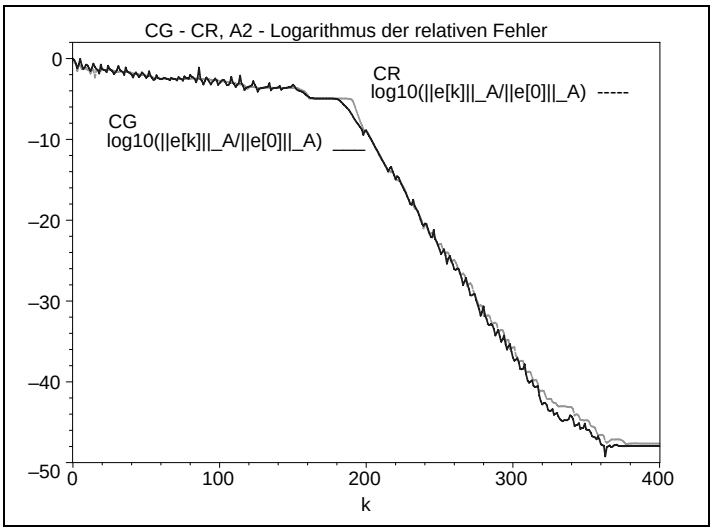


Abb. 6.132
Datei $ccgra2cm361.ps$
Matrix $A2(625, 625)$,
$c = -1.0059$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ k = 0(1)400$

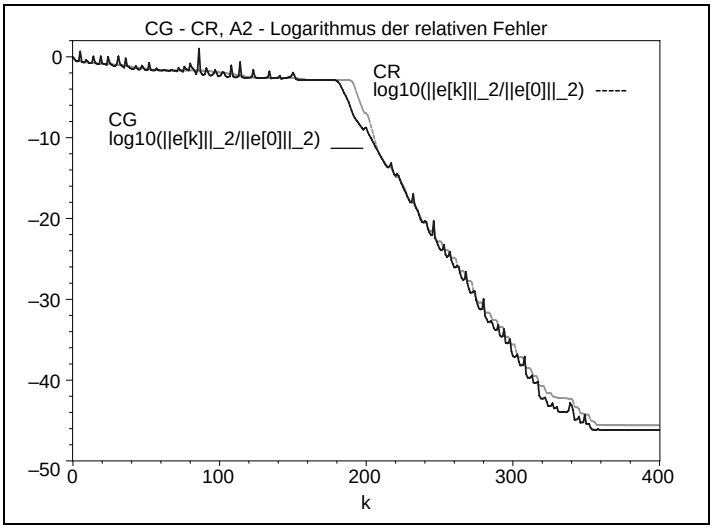


Abb. 6.133
Datei $ccgra2cm362.ps$
Matrix $A2(625, 625)$,
$c = -1.0059$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|e^{(k)}\|_2}{\|e^{(0)}\|_2}),\ k = 0(1)400$

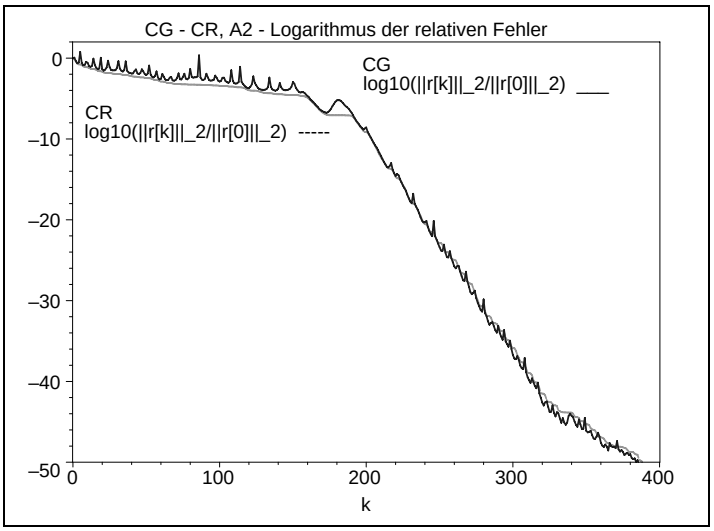


Abb. 6.134
Datei $ccgra2cm363.ps$
Matrix $A2(625, 625)$,
$c = -1.0059$,
CG versus CR:
Verlauf des
relativen Fehlers
$\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0(1)400$

Kapitel 7

Einfluss von Rundungsfehlern bei Implementation von CG und CR

Wir illustrieren den Einfluss von Rundungsfehlern bei der Implementation der Verfahren CG und CR im Zusammenhang mit ihrer theoretischen Endlichkeit sowie mit der Notation des Formelapparates und seiner numerischen Auswertung. Im Verhalten beider AV bemerkt man eine so genannte "Fehlererinnerung", besonders auffällig und typisch bei der Berechnung der Residuumsnorm $\|r^{(k)}\|_2$.
Dieses Verhalten der AV testen wir an fünf Beispielen mit Koeffizientenmatrizen $A = A^T > 0$, $A = A^T$ diagonal und indefinit, sowie $A = A^T$ indefinit.

7.1 CG mit Fehlererinnerung

Zunächst verweisen wir auf die Implementierung des CG in den Versionen 1 (3.96) und 2 (3.97) im Teil I. Dabei werden wir bei der Berechnung der Schrittzahl α sowie bei der Bestimmung des Residuums r verschiedene Varianten realisieren.

Einfache Darstellung des CG

$x^{(0)}$ Startvektor, $r^{(0)} = b - Ax^{(0)}$ Residuum, $p^{(0)} = r^{(0)}$ Suchrichtung

$k = 0, 1, 2, ...$

$$
\begin{aligned}
x^{(k+1)} &= x^{(k)} + \alpha_k p^{(k)}, \quad \text{wobei} \quad \alpha_k = \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}} = \frac{\|r^{(k)}\|_2^2}{\|p^{(k)}\|_A^2} > 0 \\
r^{(k+1)} &= b - Ax^{(k+1)} \qquad \text{(explizite Berechnung)} \\
&= r^{(k)} - \alpha_k A p^{(k)} \qquad \text{(rekursive Berechnung)} \\
p^{(k+1)} &= r^{(k+1)} + \beta_k p^{(k)}, \quad \text{wobei} \quad \beta_k = -\frac{p^{(k)T} A r^{(k+1)}}{p^{(k)T} A p^{(k)}} = \frac{\|r^{(k+1)}\|_2^2}{\|r^{(k)}\|_2^2} > 0
\end{aligned}
$$

In jedem Iterationsschritt ist nur eine Matrix-Vektor-Multiplikation $Ap^{(k)}$ zu machen.

Weitere Problemgrößen, die ebenfalls in die Betrachtungen und zu Testzwecken einbezogen werden, verursachen meistens zusätzlichen Berechnungsaufwand.

Funktional

$$\begin{aligned} Q(x) &= \frac{1}{2}\Big(\|e(x)\|_A^2 - x^{*T}b\Big) = \frac{1}{2}x^TAx - x^Tb \text{ im Algorithmus} \\ Q(x^*) &= -\frac{1}{2}x^{*T}b \le 0 \end{aligned}$$

Fehler

$$\begin{aligned} e(x) &= x^* - x \\ \|e(x)\|_A &= \|x^* - x\|_A = \sqrt{(x^*-x)^TA(x^*-x)} = \sqrt{2Q(x) + x^{*T}b} \\ \tilde{e}(x) &= \sqrt{(x^*-x)^TA(x^*-x)}, \ \ |\tilde{e}(x)| \text{ im Algorithmus} \end{aligned}$$

Residuum

$$\begin{aligned} r(x) &= b - Ax \\ \|r(x)\|_2 &= \|b - Ax\|_2 = \sqrt{r(x)^Tr(x)} \\ \|r(x)\|_2^2 &= r(x)^Tr(x) \text{ im Algorithmus} \end{aligned}$$

Kontrolle, ob mit Beendigung des Algorithmus für $e^{(k)} = x^* - x^{(k)}$ gelten

$$\begin{aligned} \|e^{(k)}\|_2 &\approx \|e^{(k)}\|_A \\ \|e^{(k)}\|_2 &= \|A^{-1}r^{(k)}\|_2 \end{aligned}$$

Falls $A = A^T > 0$ ist, gelten

$$\begin{aligned} Q(x^{(0)}) \ge Q(x^{(1)}) \ge Q(x^{(2)}) \ge ... &\to Q(x^*) \le 0, \\ \|e^{(0)}\|_A \ge \|e^{(1)}\|_A \ge \|e^{(2)}\|_A \ge ... &\to 0, \\ \|e^{(0)}\|_2 \ge \|e^{(1)}\|_2 \ge \|e^{(2)}\|_2 \ge ... &\to 0. \end{aligned}$$

Aber die Folge $\|r^{(k)}\|_2$ ist i. Allg. nicht monoton fallend.
Das ist durchaus ein gewisser Nachteil, weil ja $\|r^{(k)}\|_2$ für das Abbruchkriterium bzw. die Iterationssteuerung eingesetzt wird und nicht $e^{(k)}$ genommen werden kann.

Die Vektoren
– Residua und Abstiegsrichtungen $r^{(k)}$ sind orthogonal, d. h. $r^{(k)} \perp r^{(j)}$ für $k \ne j$,
– Suchrichtungen $p^{(k)}$ sind A-orthogonal (A-konjugiert),
– und es gilt

$$\text{span}\{r^{(0)}, r^{(1)}, ..., r^{(n-1)}\} = \text{span}\{p^{(0)}, p^{(1)}, ..., p^{(n-1)}\} = \text{span}\{r^{(0)}, Ar^{(0)}, ..., A^{n-1}r^{(0)}\}.$$

Algorithmus für die Implementierung

ε Toleranz für die Residuumnorm $\|r^{(k)}\|_2$
maxiter maximale Iterationsanzahl

$k = 0$, $x^{(0)}$ Startvektor, Berechnung von $Q(x^{(0)})$
$r^{(0)} = b - Ax^{(0)}$ Residuum
$p^{(0)} = r^{(0)}$ Suchrichtung, $lp = p^{(0)}$ letzte Suchrichtung

$$\alpha_0 = \begin{cases} \|r^{(0)}\|_2^2, & \text{falls } varianta = 1 \\ \sqrt{r^{(0)T} r^{(0)}}^2, & \text{falls } varianta = 2 \\ r^{(0)T} r^{(0)}, & \text{falls } varianta = 3 \\ \sum\limits_{i=1}^{n} |r_i^{(0)}|^2, & \text{falls } varianta = 4 \end{cases}$$

Speichern von $Q(x^{(0)}), |\tilde{e}(x^{(0)})|, \|e^{(0)}\|_2, \sqrt{\alpha_0}$ in `Qv[1]`, `epsv[1]`, `epsv2[1]`, `rv[1]`
evtl. Ausgabe und/oder Fileausgabe von $0, x^{(0)}, Q(x^{(0)}), r^{(0)}, \alpha_0$

while $(\sqrt{\alpha_k} > \varepsilon)$ & $(k < maxiter)$ **do**

$$\begin{aligned} v &= Ap^{(k)} \\ d &= v^T p^{(k)} = p^{(k)T} A p^{(k)} \\ &\text{falls } d = 0, \text{ dann Abbruch mit Ergebnis } x^{(k)}, k \\ \alpha &= \alpha_k / d \\ x^{(k+1)} &= x^{(k)} + \alpha p^{(k)} \\ r^{(k+1)} &= \begin{cases} r^{(k)} - \alpha v = r^{(k)} - \alpha A p^{(k)}, & \text{falls } variantr = 1 \\ b - Ax^{(k+1)}, & \text{falls } variantr = 2 \end{cases} \\ \alpha_{k+1} &= \begin{cases} \|r^{(k+1)}\|_2^2, & \text{falls } varianta = 1 \\ \sqrt{r^{(k+1)T} r^{(k+1)}}^2, & \text{falls } varianta = 2 \\ r^{(k+1)T} r^{(k+1)}, & \text{falls } varianta = 3 \\ \sum\limits_{i=1}^{n} |r_i^{(k+1)}|^2, & \text{falls } varianta = 4 \end{cases} \\ \beta &= \alpha_{k+1} / \alpha_k \\ p^{(k+1)} &= r^{(k+1)} + \beta p^{(k)}, \quad lp = p^{(k+1)} \text{ letzte Suchrichtung} \\ k &= k + 1 \end{aligned}$$

Berechnung von $Q(x^{(k)})$
evtl. Ausgabe und/oder Fileausgabe von $k, x^{(k)}, Q(x^{(k)}), p^{(k)}, r^{(k)}, \alpha_k$
Ausgabe und evtl. Fileausgabe von $k, \sqrt{\alpha_k}$
Speichern von $Q(x^{(k)}), |\tilde{e}(x^{(k)})|, \|e(x^{(k)})\|_2, \sqrt{\alpha_k}$ in
`Qv[k + 1]`, `epsv[k + 1]`, `epsv2[k + 1]`, `rv[k + 1]`

end while

Ausgabe von $\|x^* - x^{(k)}\|_2$, $\|A^{-1} r^{(k)}\|_2$ zwecks Vergleich mit $\|e^{(k)}\|_A, \|e^{(k)}\|_2$
$x^{(k)}, k$ Ergebnisse nach der Schleife

Algorithmus für Implementierung, weitgehend ohne Indizierung

ε Toleranz für Residuumnorm $\|r\|_2$
maxiter maximale Iterationsanzahl

$k = 0$, $x = x0$, $x0$ Startvektor, Berechnung von $Q(x)$
$r = b - Ax$ Residuum
$p = r$ Suchrichtung, $lp = p$ letzte Suchrichtung

$$\alpha_{old} = \begin{cases} \|r\|_2^2 & \text{falls } varianta = 1 \\ \sqrt{r^T r}^2, & \text{falls } varianta = 2 \\ r^T r, & \text{falls } varianta = 3 \\ \sum_{i=1}^{n} |r_i|^2, & \text{falls } varianta = 4 \end{cases}$$

Speichern von $Q(x), |\tilde{e}(x)|, \|e(x)\|_2, \sqrt{\alpha_{old}}$ in `Qv[1]`, `epsv[1]`, `epsv2[1]`, `rv[1]`
evtl. Ausgabe und/oder Fileausgabe von $0, x, Q(x), r, \alpha_{old}$

while $(\sqrt{\alpha_{old}} > \varepsilon)$ & $(k < maxiter)$ **do**

$v = Ap$
$d = v^T p = p^T Ap$
falls $d = 0$, dann Abbruch mit Ergebnis x, k
$\alpha = \alpha_{old}/d$
$x = x + \alpha p$

$$r = \begin{cases} r - \alpha v = r - \alpha Ap, & \text{falls } variantr = 1 \\ b - Ax, & \text{falls } variantr = 2 \end{cases}$$

$$\alpha_{new} = \begin{cases} \|r\|_2^2, & \text{falls } varianta = 1 \\ \sqrt{r^T r}^2, & \text{falls } varianta = 2 \\ r^T r, & \text{falls } varianta = 3 \\ \sum_{i=1}^{n} |r_i|^2, & \text{falls } varianta = 4 \end{cases}$$

$\beta = \alpha_{new}/\alpha_{old}$
$p = r + \beta p$, $lp = p$ letzte Suchrichtung
$k = k + 1$
$\alpha_{old} = \alpha_{new}$

Berechnung von $Q(x)$
evtl. Ausgabe und/oder Fileausgabe von $k, x, Q(x), p, r, \alpha_{old}$
Ausgabe und evtl. Fileausgabe von $k, \sqrt{\alpha_{old}}$
Speichern von $Q(x), |\tilde{e}(x)|, \|e(x)\|_2, \sqrt{\alpha_{old}}$ in
`Qv[k + 1]`, `epsv[k + 1]`, `epsv2[k + 1]`, `rv[k + 1]`

end while

Ausgabe von $\|x^* - x\|_2$, $\|A^{-1}r\|_2$ zwecks Vergleich mit $\|e(x)\|_A, \|e(x)\|_2$
x, k Ergebnisse nach der Schleife

Variantenauswahl

Dies betrifft die Berechnung des Residuums r in 2 Varianten

1. **rekursiv** (`recursive, update`) mit $r = r - \alpha Ap,\ \alpha = \dfrac{(r,r)}{(Ap,p)} = \dfrac{r^T r}{p^T Ap}$,
2. **wirklich** (`realy, true`) mit $r = b - Ax$,

sowie 4 Varianten der Berechnung des Zählers $\alpha_k = r^T r$ in der Schrittzahl α

1. $\alpha_k = \|r\|_2^2$,
2. $\alpha_k = \left(\sqrt{r^T r}\right)^2$,
3. $\alpha_k = r^T r$,
4. $\alpha_k = \sum_{i=1}^{n} |r_i|^2$.

In den Berechnungen mit Maple zeigt sich, dass bezüglich der α-Varianten gilt

$$1. \approx 2., \quad 3. \approx 4,$$

so dass in weiteren Rechnungen dann nur mit den α-Varianten 1 und 3 gearbeitet wird. Außerdem erweist sich die Wurzelberechnung in Maple als nicht besonders gut implementiert.

In Maple werden sowohl die exakte Rechnung (symbolische Berechnung in der Rationalarithmetik), wo erwartunggemäß keine Unterschiede in den Varianten auftreten, sowie die numerische Rechnung bei 4 Genauigkeiten mit der Gleitpunktarithmetik von 10, 16, 20, 25 Dezimalstellen (`Digits:=...`) ausgeführt.

Das CG wird in Maple durch die folgende Prozedur implementiert.

```
> cg1:=proc(n::posint, A::matrix, b::vector, x0::vector,
           maxiter::posint, eeps::numeric,
           varianta::posint,  variantr::posint,
           aus::name, fileaus::name)
  local  k,i,x,p,r,v,xh,alpha,alphaold,alphanew,beta,Q,fh,fh1,fh2,h,d;
  global Qv,epsv,epsv2,rv,Ainv,xs,lp,file1;

  fh2:=‘%.16e‘;         # Ausgabeformate einstellen
  fh1:=‘%+.10e‘;
  fh :=fh1;
  for i from 2 to n do
   fh:=cat(fh,‘ ‘,fh1);
  end do;

  k:=0:
```

```
x:=evalf(evalm(x0)):   # x:=evalm(x0):
Q:=0.5*evalm(transpose(x)&*A&*x)-evalm(transpose(x)&*b):
r:=evalm(b-A&*x):
p:=evalm(r):  lp:=evalm(p):

if varianta=1 then alphaold:=norm(r,frobenius)^2;
 elif varianta=2 then alphaold:=sqrt(evalm(transpose(r)&*r))^2;
 elif varianta=3 then alphaold:=evalm(transpose(r)&*r);
 elif varianta=4 then i:='i': alphaold:=sum(abs(r[i])^2,i=1..n);
 else
end if;

Qv[1]:=Q;
h:=evalm(transpose(xs)&*b);  xh:=evalm(xs-x);
epsv[1]:=abs(sqrt(2.0*Q+h));
epsv2[1]:=sqrt(evalm(transpose(xh)&*xh));
rv[1]:=evalf(sqrt(alphaold));

if aus=ja then
 fprintf(default,`\n`):
 fprintf(default,`Schritt k = %g\n`,k);
 fprintf(default,`Startvektor            x =
                 [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(default,`Funktionswert       Q(x) = `||fh2||`\n`,Q);
 fprintf(default,`Residuum/Suchr.  r =b-Ax =
                 [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(default,`Anfangsfehlerquadrat r'r = `||fh2||`\n\n`,alphaold);
end if;
if fileaus=ja then
 fprintf(file1,`\n`):
 fprintf(file1,`Schritt k = %g\n`,k);
 fprintf(file1,`Startvektor            x =
                 [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(file1,`Funktionswert       Q(x) = `||fh2||`\n`,Q);
 fprintf(file1,`Residuum/Suchr.  r =b-Ax =
                 [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(file1,`Anfangsfehlerquadrat r'r = `||fh2||`\n\n`,alphaold);
end if;

while (sqrt(1.0*alphaold)>eeps) and (k<maxiter) do
                                # absoluter Fehler mit ||r||
 v:=evalm(A&*p);
 d:=evalm(transpose(v)&*p);
 if d=0 then
  lprint(`Abbruch wegen Nenner p'Ap=0`):
  RETURN(x,k);
 end if;

 alpha:=alphaold/d;
 x:=evalm(x+alpha*p);
 if variantr=1 then r:=evalm(r-alpha*v); else r:=evalm(b-A&*x); end if;
 if varianta=1 then alphanew:=norm(r,frobenius)^2;
  elif varianta=2 then alphanew:=sqrt(evalm(transpose(r)&*r))^2;
  elif varianta=3 then alphanew:=evalm(transpose(r)&*r);
```

```
 elif varianta=4 then i:='i': alphanew:=sum(abs(r[i])^2,i=1..n);
 else
end if;

beta:=alphanew/alphaold;
p:=evalm(r+beta*p);  lp:=evalm(p);
k:=k+1;
alphaold:=alphanew;

Q:=0.5*evalm(transpose(x)&*A&*x)-evalm(transpose(x)&*b);

if aus=ja then
 fprintf(default,`\n`);
 fprintf(default,`Schritt k = %g\n`,k);
 fprintf(default,`Iterationsvektor      x =
                 [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(default,`Funktionswert      Q(x) = `||fh2||`\n`,Q);
 fprintf(default,`Suchrichtung          p =
                 [`||fh||`]\n`,seq(p[i],i=1..n));
 fprintf(default,`Residuum       r = b-Ax =
                 [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(default,`Fehlernormquadrat   r'r = `||fh2||`\n`,alphaold);
end if;
if fileaus=ja then
 fprintf(file1,`\n`);
 fprintf(file1,`Schritt k = %g\n`,k);
 fprintf(file1,`Iterationsvektor      x =
               [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(file1,`Funktionswert      Q(x) = `||fh2||`\n`,Q);
 fprintf(file1,`Suchrichtung          p =
               [`||fh||`]\n`,seq(p[i],i=1..n));
 fprintf(file1,`Residuum       r = b-Ax =
               [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(file1,`Fehlernormquadrat   r'r = `||fh2||`\n`,alphaold);
end if;
fprintf(default,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sqrt(alphaold));
if fileaus=ja then
 fprintf(file1,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sqrt(alphaold));
end if;

Qv[k+1]:=Q;
xh:=evalm(xs-x);
epsv[k+1]:=abs(sqrt(evalm(transpose(xh)&*A&*xh)));
           # epsv[k+1]:=abs(sqrt(2.0*Q+h));
epsv2[k+1]:=sqrt(evalm(transpose(xh)&*xh));
rv[k+1]:=evalf(sqrt(alphaold));
end do:

lprint(`||xs-x||_2    = `,evalf(norm(xs-x,frobenius))):
lprint(`||A^(-1)r||_2 = `,evalf(norm(Ainv&*r,frobenius))):
lprint(` `):

[x,k];
end:
```

7.1.1 Beispiel 1

Im Beispiel 1 machen wir die Ausführungen etwas dataillierter, während in den nachfolgenden wir uns auf die grafische Darstellung der Ergebnisse beschränken.

```
# Beispiel 1, A=A'>0

 n:=8:
 A:=matrix(n,n,
    [[ 168,   24,  338,  27,  27,   53,  -7,  80],
     [  24,  178,  169,  72,  53, -103,  17,  80]
     [ 338,  169, 1177, 192, -62, -108, -48, 180],
     [  27,   72,  192, 125,   2,  -24,  36, 180],
     [  27,   53,  -62,   2, 222,   70,  46, 100],
     [  53, -103, -108, -24,  70,  178,  34, 100],
     [  -7,   17,  -48,  36,  46,   34,  34, 100],
     [  80,   80,  180, 180, 100,  100, 100, 400]]):

 b:=vector(n,[ 229, 129, 790, -214, 106, -276, -216, -600]):

# Loesung
 xs:=vector(n,[1,-1,1,-1,2,-2,2,-2]):
```

EW von $\lambda(A) = \{0.006\,305, 1.359\,344, 5.859\,472, 44.417\,725, 199.730\,179, 282.983\,844,$
$548.279\,552, 1\,399.363\,575\}$,

Kondition mittels Spektralnorm $\kappa(A) = 221\,911.791....$

Für das spezielle LGS berechnen wir in den verschiedenen Varianten die absoluten Fehler

$$\|e^{(k)}\|_A, \quad \|e^{(k)}\|_2, \quad \|r^{(k)}\|_2, \quad k = 0, 1, 2, ...,$$

die man als Ergebnisvektoren `epsv[1..maxiter+1]`, `epsv2[1..maxiter+1]` bzw. `rv[1..maxiter+1]` der obigen Prozedur `cg1` erhält. Die Werte $\|e^{(k)}\|_A$ bilden eine monoton abnehmende Folge, was für $\|r^{(k)}\|_2$ i. Allg. und speziell bei numerischen Rechnungen über n Iterationen hinaus, nicht zutreffen muss.

Dann folgen grafische Vergleiche der relativen Fehler des CG, die aus Gründen der besseren Übersicht als Funktionen $\log_{10}()$ in Abhängigkeit von der Iterationszahl k dargestellt werden, also

$$f(k) \;=\; \log_{10}\left(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}\right),\ \log_{10}\left(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}\right),\ k = 0, 1, 2, ... \ . \tag{7.1}$$

Zunächst schauen wir auf eine exakte Rechnung in der Rationalarithmetik von Maple, die bei $x^{(0)} = (1, 0, ..., 0)^T$ und der Dimension $n = 8$ des Problems ohne Weiteres machbar ist und wo spätestens $\|e^{(n)}\|_A = 0$ sein muss.

```
CG : Iterationsverlauf bei exakter Rechnung
 i         Q(x[i])=          ||e[i]||_A=||xs-x[i]||_A   ||e[i]||_2=             ||r[i]||_2=
     (||e[i]||^2_A-xs'b)/2     =sqrt(2Q(x[i])+xs'b)     ||xs-x[i]||_2           ||b-Ax[i]||_2
 ------------------------------------------------------------------------------------------------
 0  -1.4500000000000000e+02  4.8435524153249340e+01  4.3588989435406740e+00  9.4746714982631460e+02
 1  -9.8368605228089200e+02  2.5857840115489440e+01  3.6560995780917200e+00  5.2812573464063930e+02
 2  -1.1459168350799170e+03  1.8551720401088610e+01  3.4384336294029440e+00  2.9031119293215690e+02
 3  -1.2813654838675770e+03  8.5597331888818880e+00  3.1543128197346300e+00  5.8409543263193730e+01
 4  -1.3150994801622860e+03  2.4085347569483770e+00  2.8805665078096420e+00  3.0634487373454350e+01
 5  -1.3168735117533320e+03  1.5009918365320540e+00  2.8737370089577910e+00  2.0801202738379160e+00
 6  -1.3175615355950960e+03  9.3644477135973120e-01  2.6938938285905780e+00  1.4582848191466750e+00
 7  -1.3179788185392140e+03  2.0582254874512040e-01  2.5918966487041220e+00  1.6345905778557270e-02
 8  -1.3180000000000000e+03  0.0000000000000000e+00  0.0000000000000000e+00  0.0000000000000000e+00
```

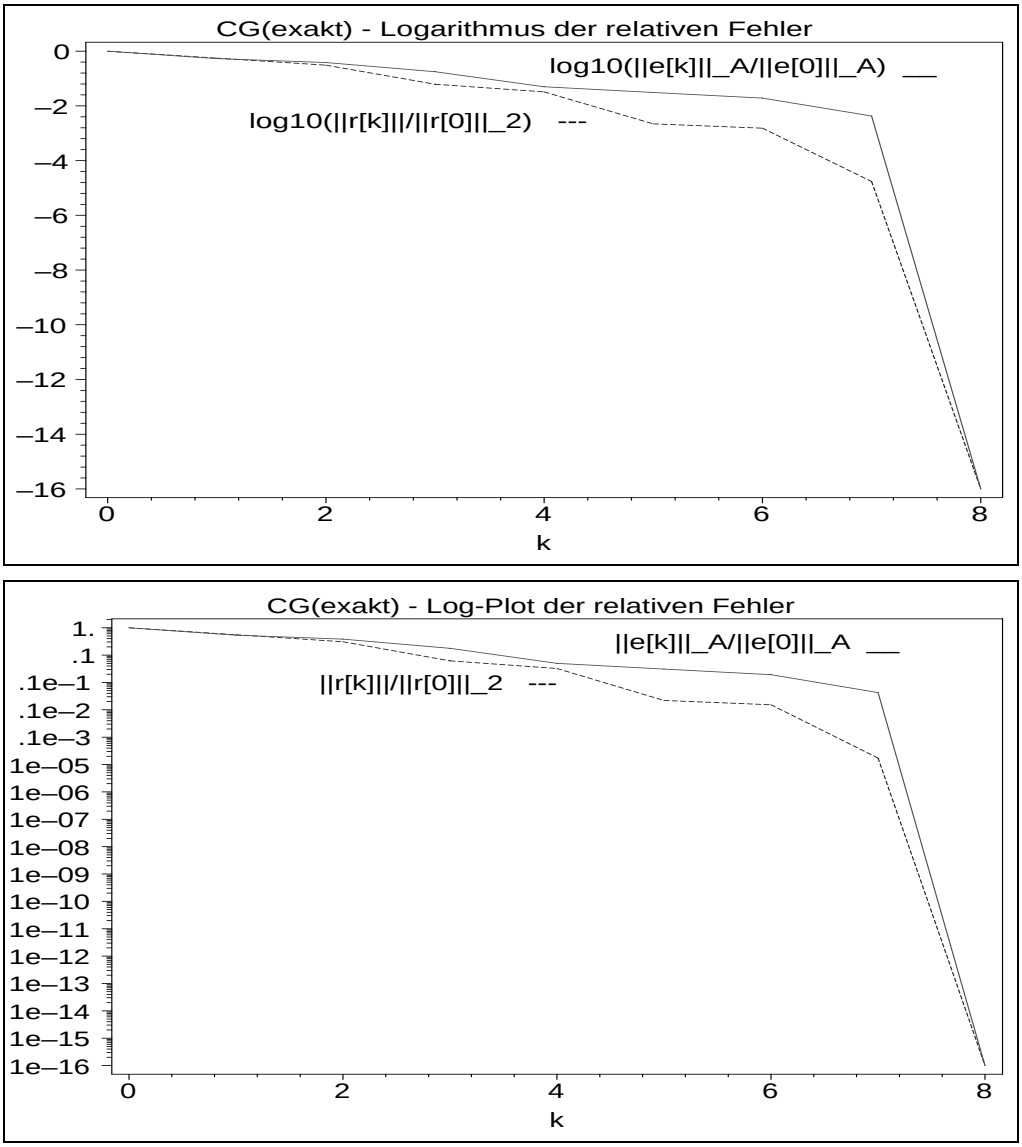


Abb. 7.1 Dateien *cg*1_01.*ps*, *cg*1_02.*ps*, Matrix $A(8,8)$ zu Beispiel 1, CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik), Verlauf der relativen Fehler $\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}, \frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2},\ k = 0(1)8$, als Logarithmus bzw. im Log-Plot

Unser Hauptaugenmerk liegt nun auf der numerischen Bestimmung des Residuums $r(x)$ mit den beiden schon genannten Varianten `recursive` und `realy`.

Die folgenden Abbildungen zeigen den Verlauf jedes der genannten relativen Fehler von CG in der Gegenüberstellung der beiden r-Varianten. Es zeigt sich, das die wirkliche Berechnung des Residuums ab einer bestimmten Iterationsanzahl zu keiner Verkleinerung dieses Wertes mehr führt. Bei der rekursiven Berechnung des Residuums tritt diese Stagnation gar nicht ein. Diesen Effekt wollen wir als **Fehlererinnerung** bezeichnen. Er ist auch ein Hinweis darauf, das im Iterationsprozess jeweils nach einer gewissen Anzahl von Schritten die Formel $r(x) = b - Ax$ auszuwerten ist, auch wenn das eine zusätzliche Matrix-Vektor-Multiplikation kostet.

Bei der Berechnung von $e(x)$ "steht" oft der Fehler bei geringer Genauigkeit in der Variante `realy`, für höhere Genauigkeiten weichen die r-Varianten nicht wesentlich voneinander ab und stagnieren in der Nähe der vorgegebenen Gleitpunktarithmetik.

In der Grafik zum CG sind bei Gegenüberstellung der zwei r-Varianten die Fehler gekennzeichnet mit den Linienarten

$\log_{10}(\|r^{(k)}\|_2/\|r^{(0)}\|_2)$	solid line, bold	`_____ recursive`
$\log_{10}(\|r^{(k)}\|_2/\|r^{(0)}\|_2)$	dash line	`- - - - - realy`

genauso für die den Fehler $\log_{10}(\|e^{(k)}\|_A/\|e^{(0)}\|_A)$.

Die 4 Genauigkeiten der numerischen Rechnung mit der Gleitpunktarithmetik bei 10, 16, 20, 25 Dezimalstellen werden in Maple mittels `Digits:=...` realisiert.

Wir testen dafür alle vier α-Varianten bei Gegenüberstellung der zwei r-Varianten. Die erste Abbildung zeigen wir etwas vergrößert, werden jedoch dann jeweils eine komplette α-Variante zusammen darstellen.

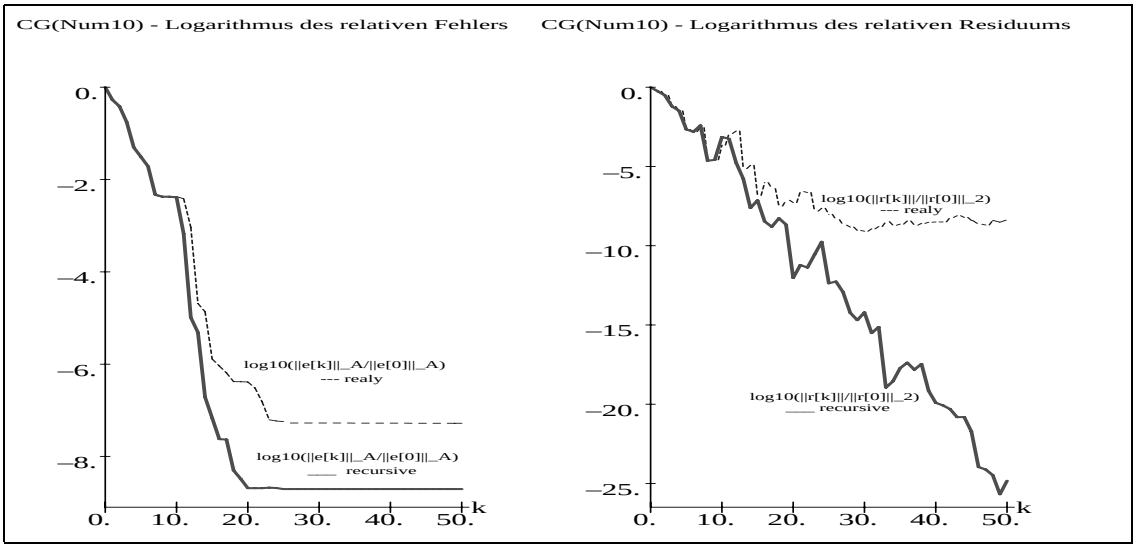


Abb. 7.2 Datei *cg1va110.ps*, Matrix $A(8,8)$ zu Beispiel 1,
CG: numerische Rechnung, Digits=10, α-Variante 1
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der zwei r-Varianten `recursive` und `realy`

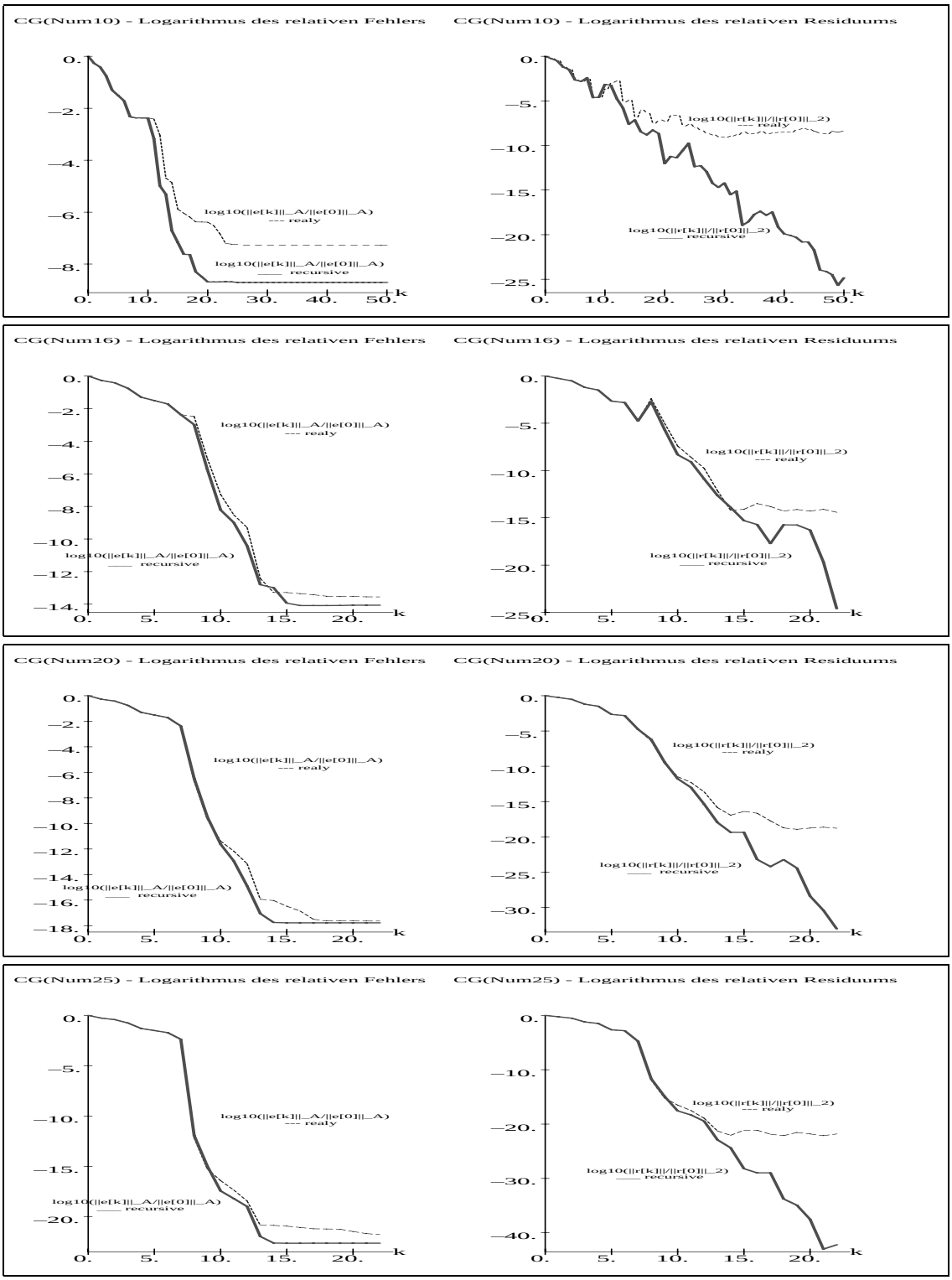


Abb. 7.3 Dateien $cg1va110.ps, cg1va116.ps, cg1va120.ps, cg1va125.ps$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

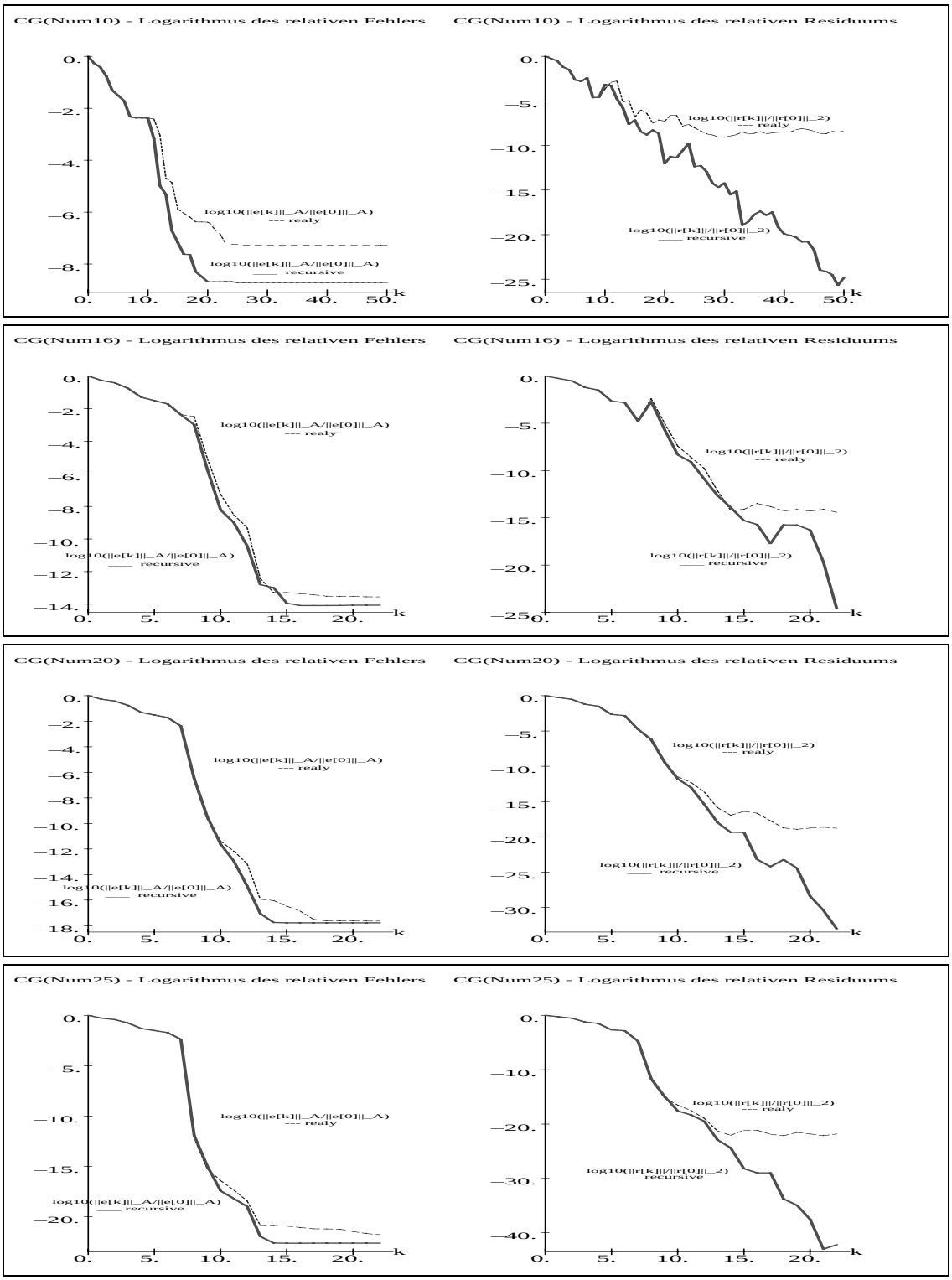


Abb. 7.4 Dateien $cg1va210.ps, cg1va216.ps, cg1va220.ps, cg1va225.ps$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 2,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

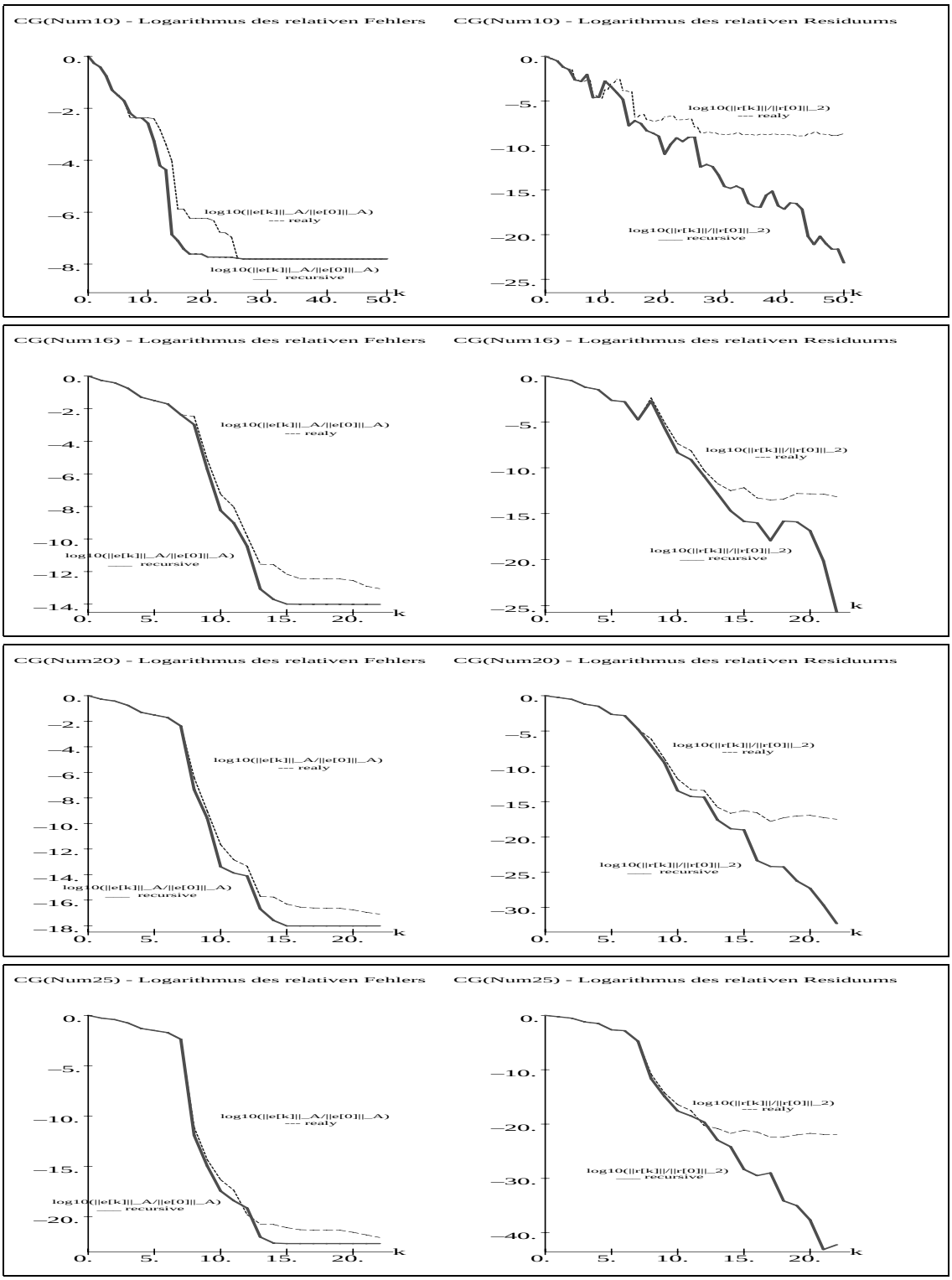


Abb. 7.5 Dateien $cg1va310.ps, cg1va316.ps, cg1va320.ps, cg1va325.ps$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

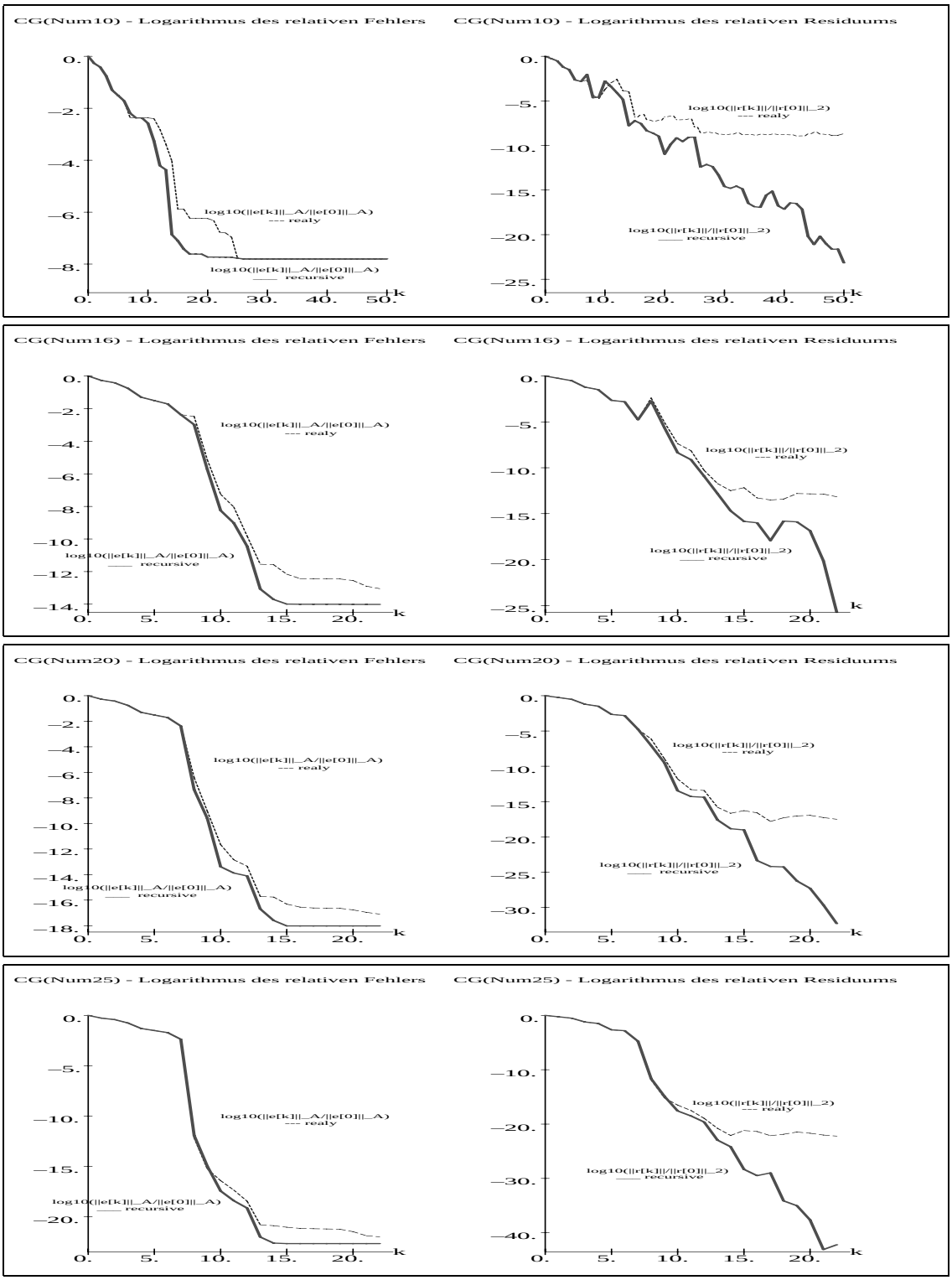


Abb. 7.6 Dateien $cg1va410.ps, cg1va416.ps, cg1va420.ps, cg1va425.ps$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 4,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.1.2 Beispiel 2

```
# Beispiel 2, A=A'>0

 n:=6:
 A:=matrix(n,n,
   [[ 71,  -1,  23, -36,  70,   60],
    [ -1, 161, 102,  33,  40, -120],
    [ 23, 102, 135,  18, 110,  -60],
    [-36,  33,  18,  35, -22,  -60],
    [ 70,  40, 110, -22, 148,   24],
    [ 60,-120, -60, -60,  24,  144]]):

 b:=vector(n,[187, 194, 360, -45, 432, 0 ]):

# Loesung
 xs:=vector(n,[1,-1,1,2,-2,2]):
```

EW von $\lambda(A) = \{0.182\,596, 4.793\,186, 6.700\,107, 46.718\,766, 263.902\,849, 371.702\,493\}$, Kondition mittels Spektralnorm $\kappa(A) = 2\,035.647....$

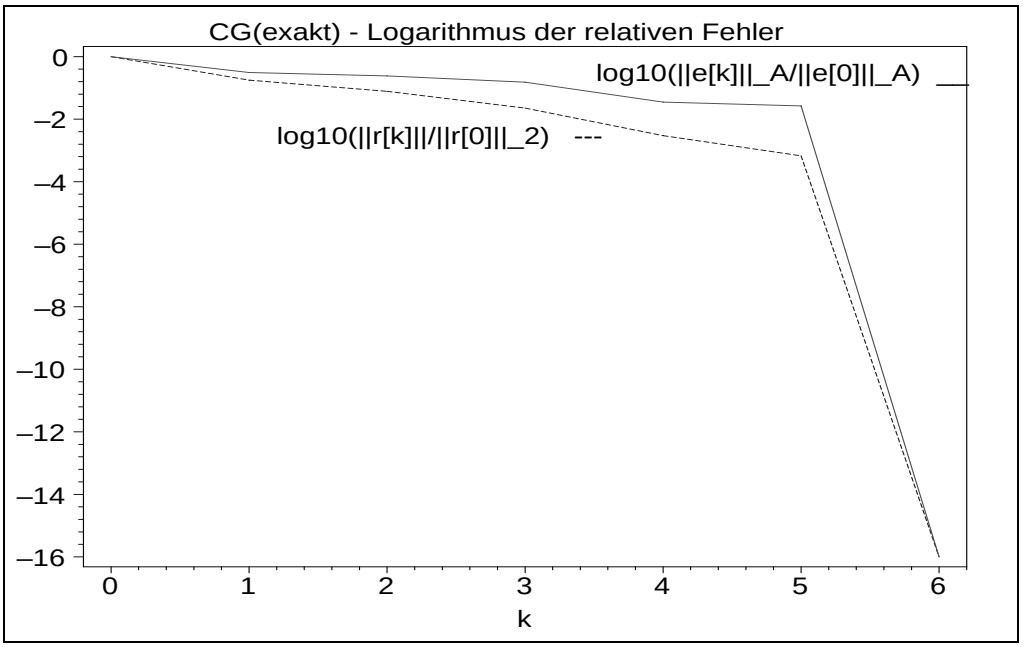


Abb. 7.7 Datei *cg2_01.ps*, Matrix $A(6,6)$ zu Beispiel 2,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0(1)6$

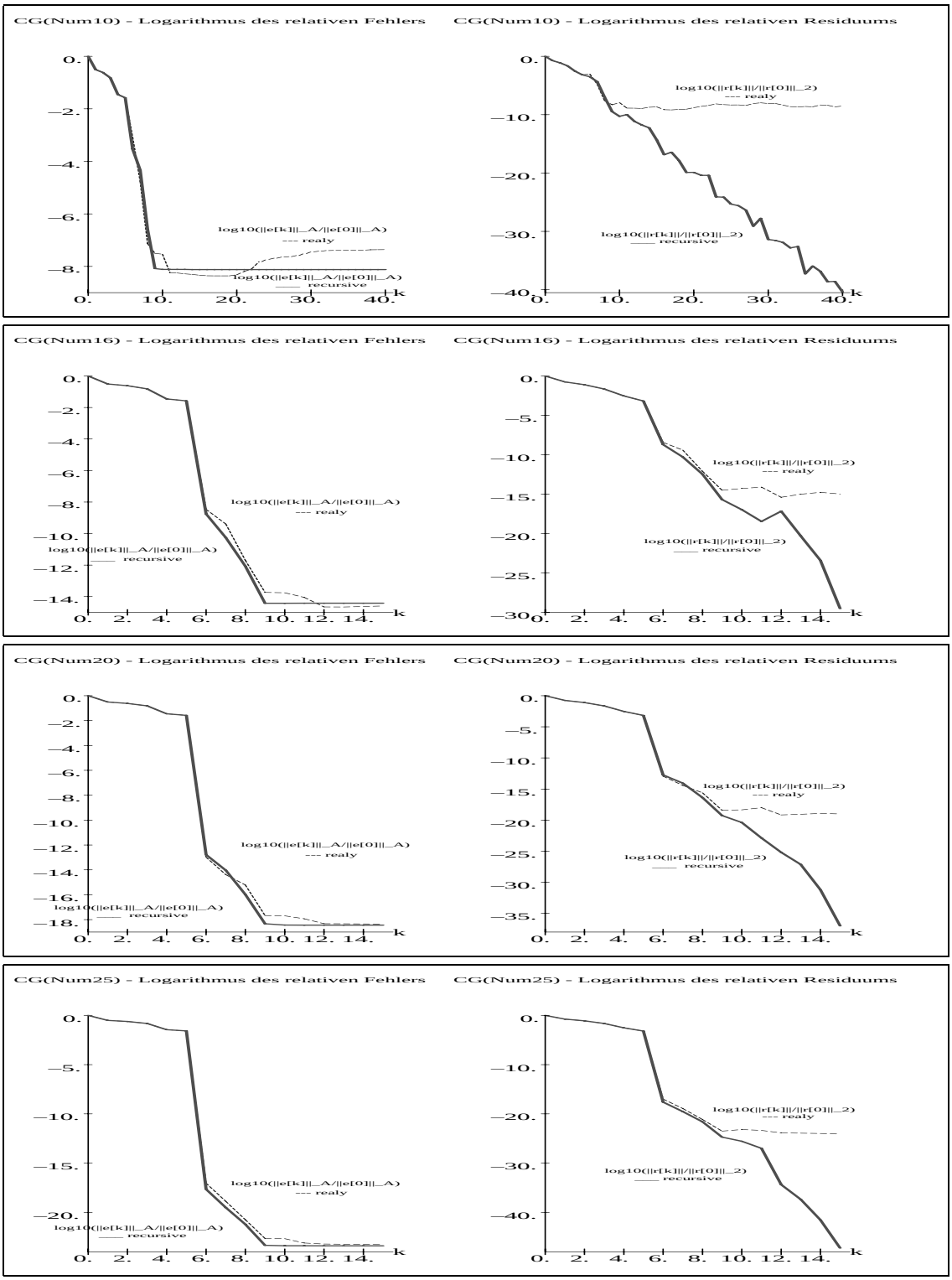


Abb. 7.8 Dateien $cg2va110.ps, cg2va116.ps, cg2va120.ps, cg2va125.ps$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

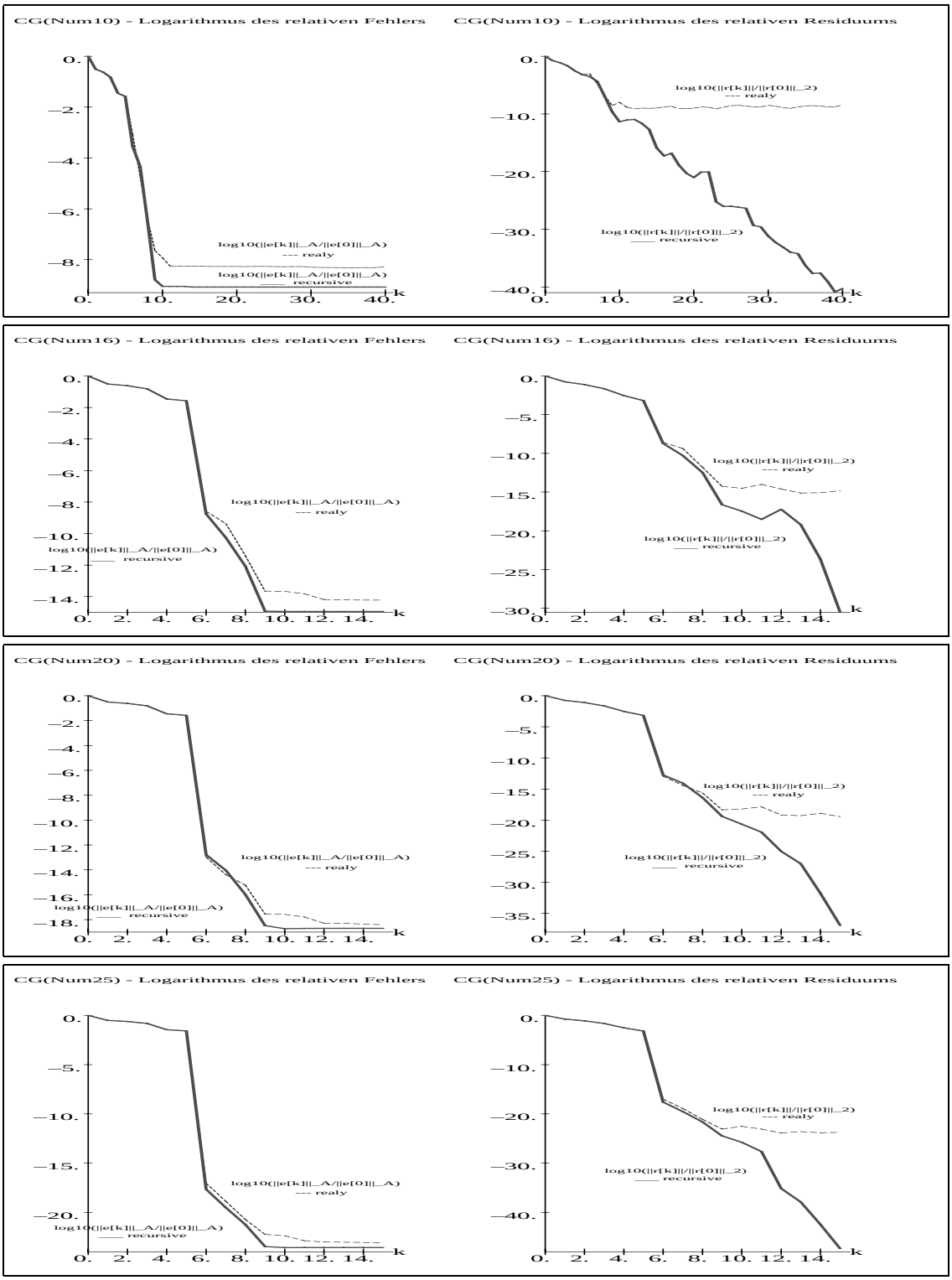


Abb. 7.9 Dateien *cg2va310.ps*, *cg2va316.ps*, *cg2va320.ps*, *cg2va325.ps*, CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.1.3 Beispiel 3

```
# Beispiel 3  A=A' diagonal und indefinit
# Van der Vorst Matrix   A = diag(-9-c,-7-c,-5-c,...,187-c,189-c)
 n:=100:
# Bsp. 3.1
 c:=0:
 A0:=evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
 b0:=vector(n,[ seq(-11+2*i-c,i=1..n)]):
# Loesung
 xs0:=vector(n,[1$n]):  # 1,1,1,....
# Bsp. 3.2
 c:=97/100:
 As:=evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
 bs:=vector(n,[ seq(-11+2*i-c,i=1..n)]):
# Loesung
 xss:=vector(n,[1$n]):  # 1,1,1,....
```

Die EW einer Diagonalmatrix sind die Diagonalelemente selber.
Kondition mittels Spektralnorm $\kappa(A_0) = 189,\ \kappa(A_s) = 6\,267.666....$

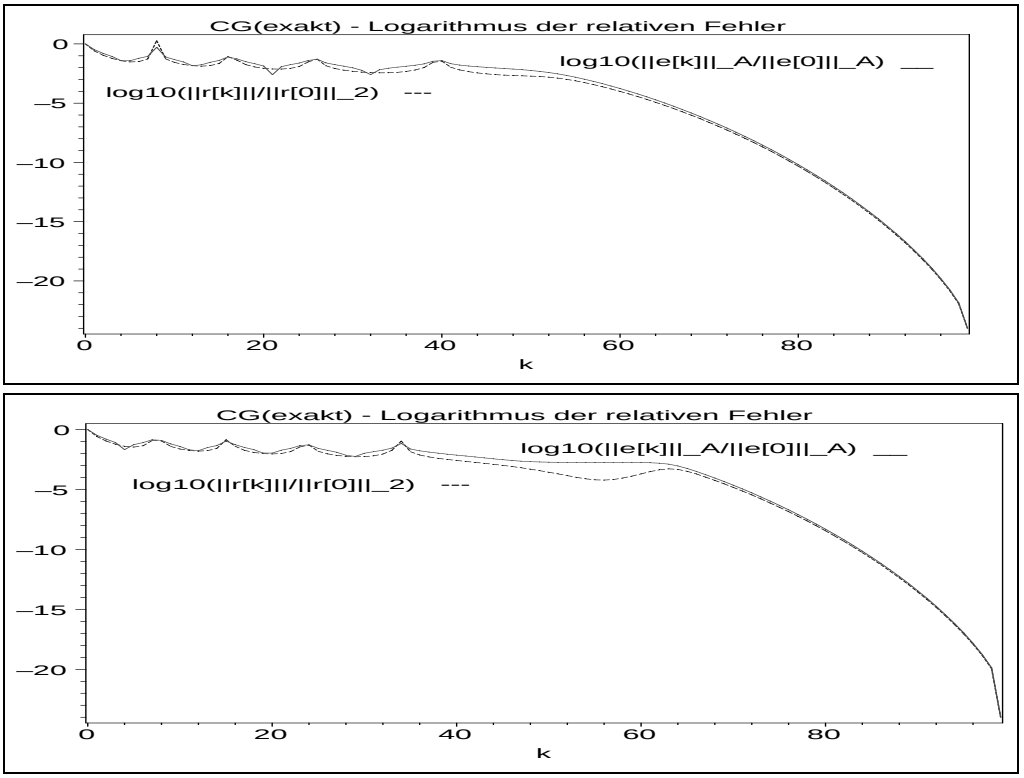


Abb. 7.10 Datei *cg3a0_01.ps*, *cg3as_01*, Matrix $A_{=,s}(100, 100)$ zu Beispiel 3, CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik), Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ..., 100$

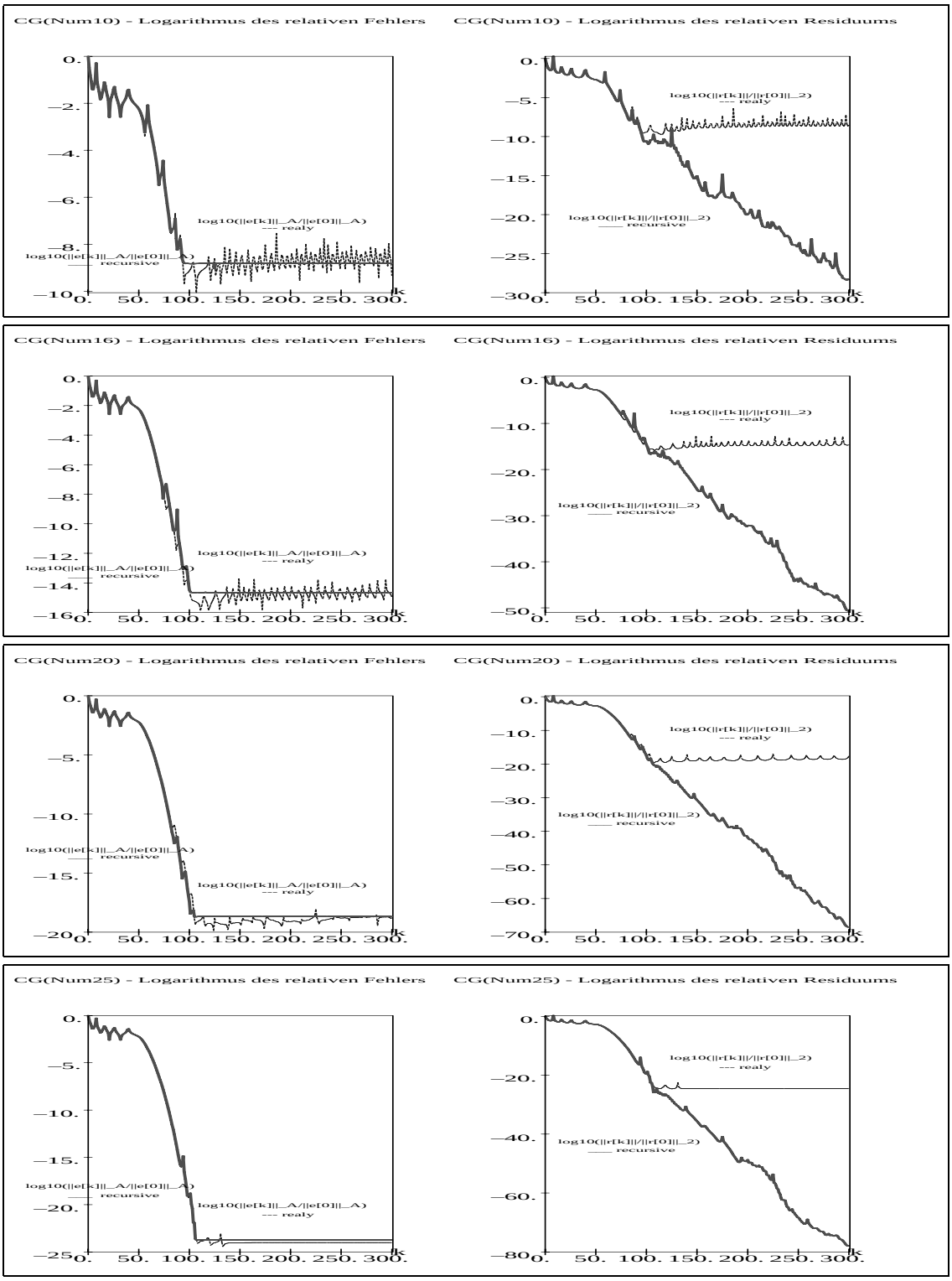


Abb. 7.11 Dateien $cg3a0110.ps, cg3a0116.ps, cg3a0120.ps, cg3a0125.ps$, A_0
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

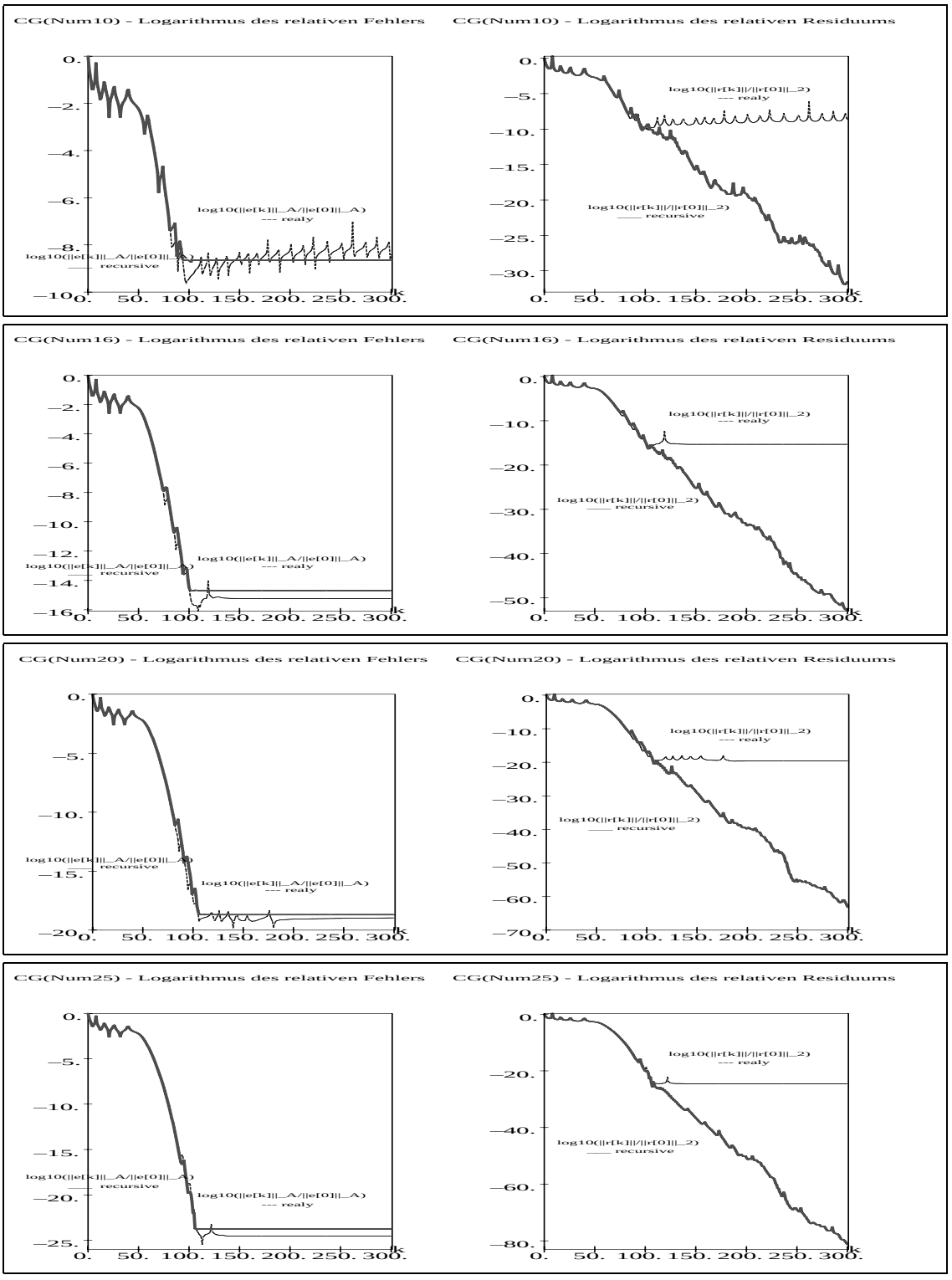


Abb. 7.12 Dateien $cg3a0310.ps, cg3a0316.ps, cg3a0320.ps, cg3a0325.ps$, A_0, CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

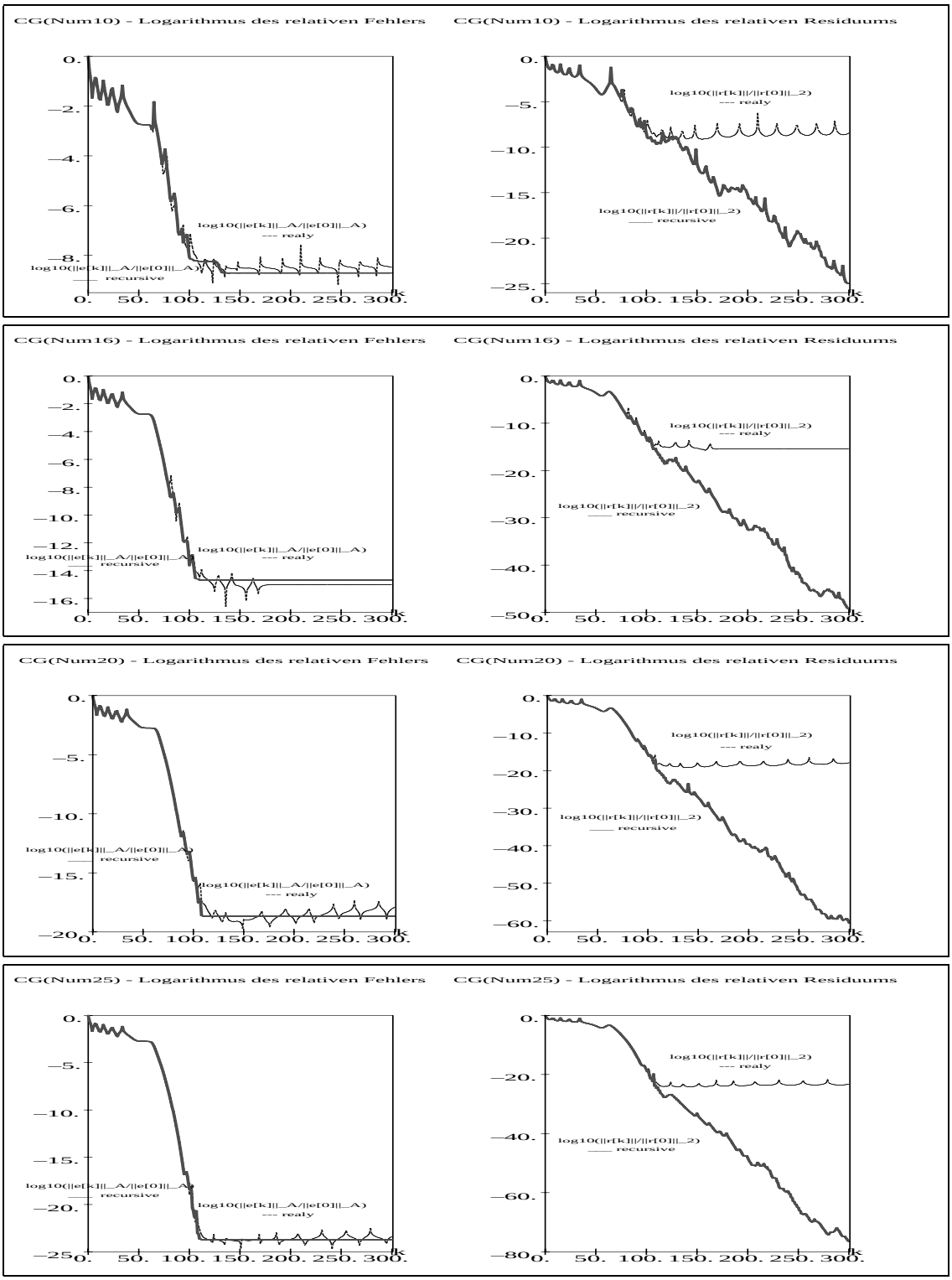


Abb. 7.13 Dateien $cg3as110.ps, cg3as116.ps, cg3as120.ps, cg3as125.ps$, A_s, CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

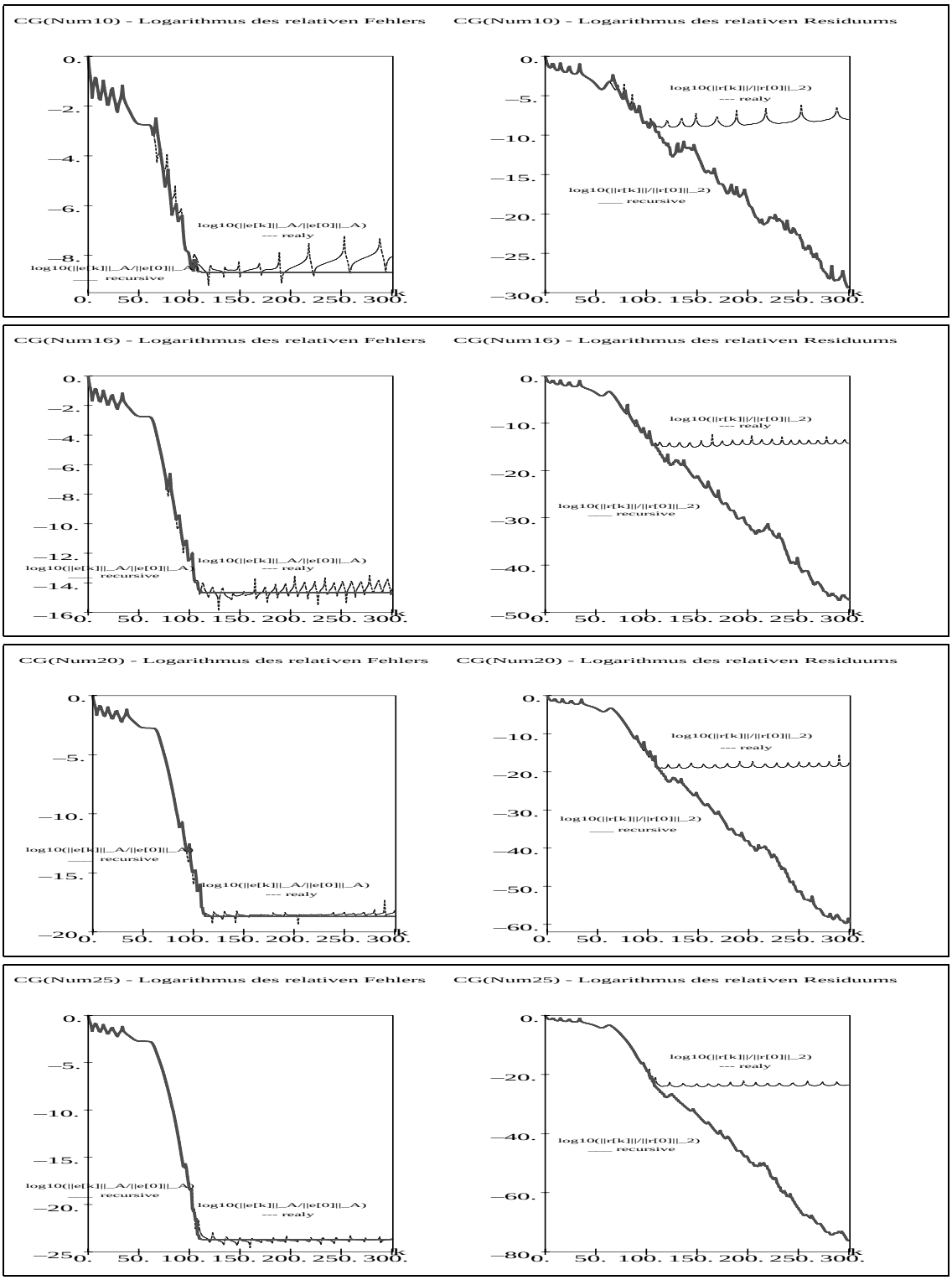
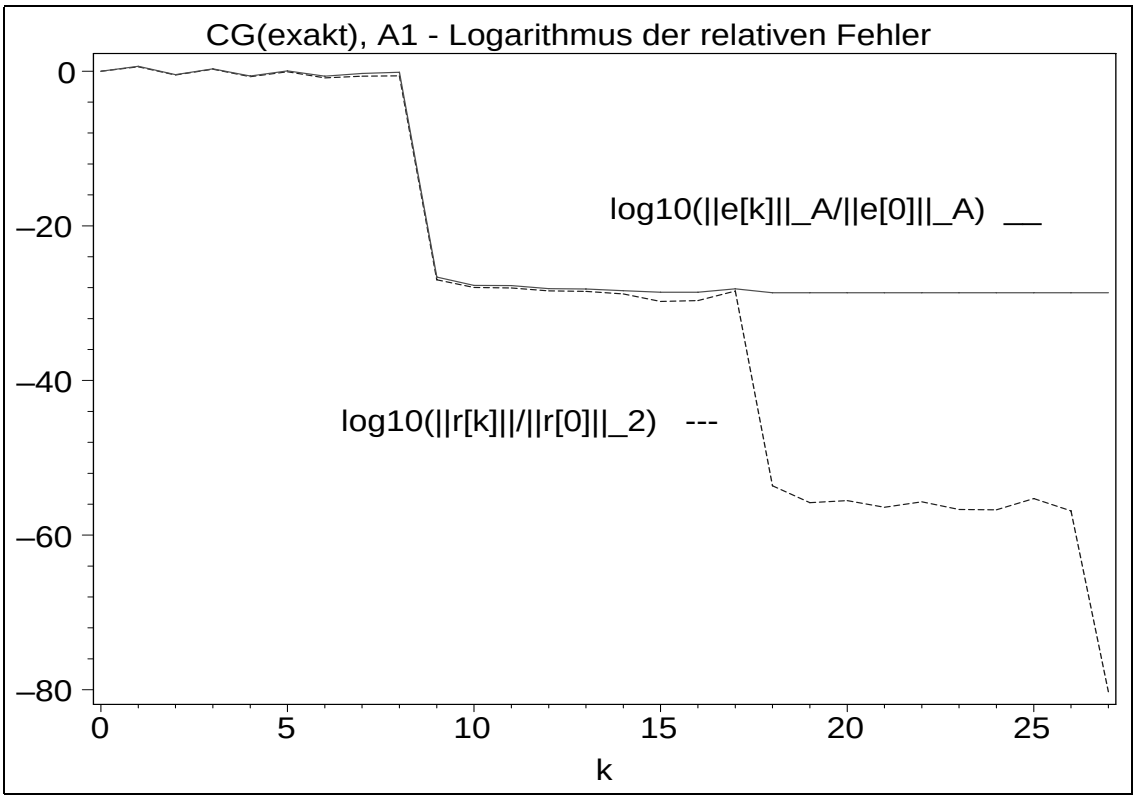
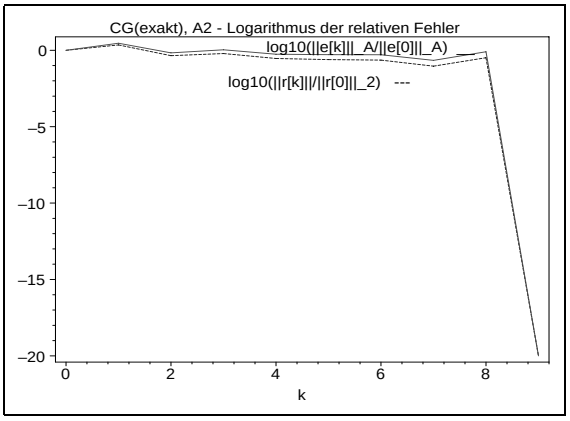


Abb. 7.14 Dateien $cg3as310.ps, cg3as316.ps, cg3as320.ps, cg3as325.ps$, A_s, CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.1.4 Beispiel 4

```
# Beispiel 4, A=A'  diagonal, A>0 oder indefinit
# 3 Diagonalmatrizen mit speziellen Spektren (Verteilung der
#   Diagonalelemente  unter Verwendung eines Shift c

 n := 10:   # 20, 50, 100, 200  # gerade
 c := 0:    # 1+1E-6            # Shift

# Bsp. 4.1,   A1 mit  EW im Intervall [-1+c,1+c],
#             dichter am Rand, weniger dicht in der Mitte
# sehr langsame symbolische Rechnung ersetzen durch
# hochgenaue Float-Rechnung
 A1 := evalm(diag(seq(evalf(sin(Pi/2*(-1+2*(i-1)/(n-1)))),i=1..n))
       +c*diag(1$n)):
 b1 := vector(n,[seq(evalf(sin(Pi/2*(-1+2*(i-1)/(n-1))))+c,i=1..n)]):
# Loesung
 xs1:=vector(n,[1$n]):  # 1,1,1,....
-----------------------------------------------------------------------

# Bsp. 4.2,   A2 mit gleichverteilten EW im Intervall [-1+c,1+c],
 m := n/2:
 A2 := evalm(diag(seq(-1+2*(i-1)/(n-1),i=1..n))+c*diag(1$n)):
 b2 := vector(n,[ seq(-1+2*(i-1)/(n-1)+c,i=1..n)]):
# Loesung
 xs2:=vector(n,[1$n]):  # 1,1,1,....
-----------------------------------------------------------------------

# Bsp. 4.3,   A3 mit  EW im Intervall [-1+c,1+c],
#             verteilt in 2 schmalen Streifen nahe den Raendern
 d := 1/10:   # Streifenbreite des Teilspektrums
 m := n/2:
 A3 := diag(0$n):
 for i from 1 to m do
   h := -1+d*(i-1)/(m-1);
   A3[i,i] := h+c;           # Spektrum in 2 kleinen randnahen
   A3[n+1-i,n+1-i] := -h+c;  # Teilintervallen/Streifen
 end do:
 A3 := evalm(A3):
 b3 := vector(n,[seq(A3[i,i],i=1..n)]):
# Loesung
 xs3:=vector(n,[1$n]):  # 1,1,1,....
```





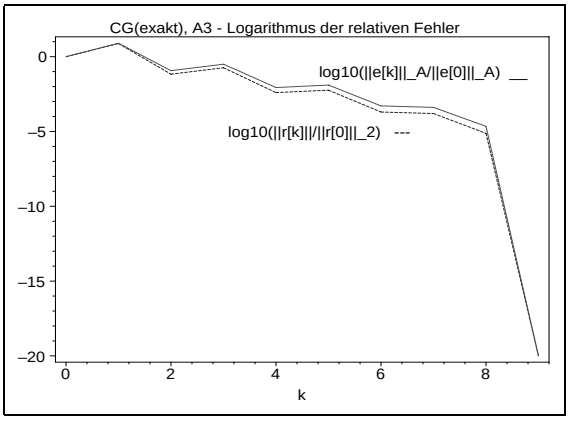
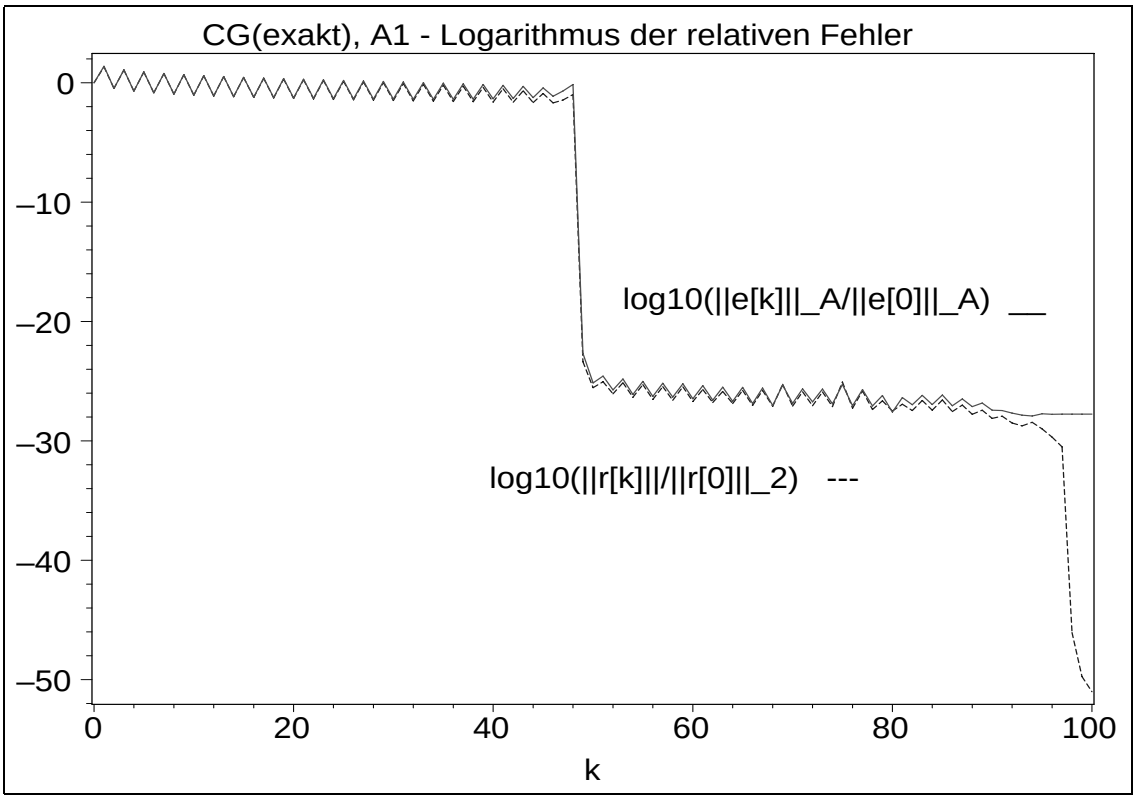
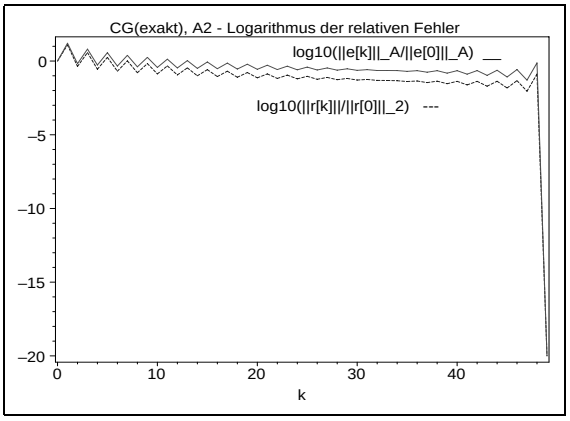


Abb. 7.15
Dateien
$cg4a110.ps,\ cg4a210.ps,\ cg4a310.ps,$
Matrizen $A_{1,2,3}(10,10),\ c=0,$
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k=0,1,...$





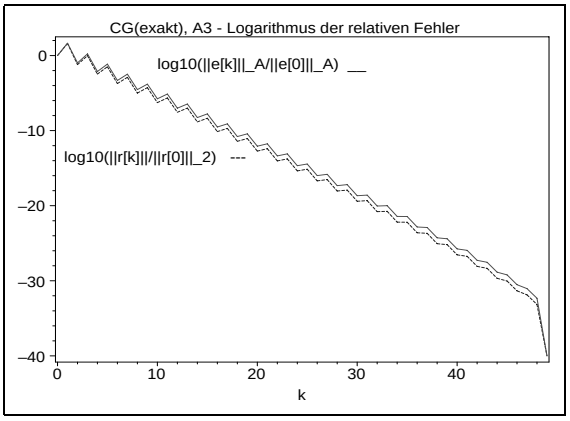
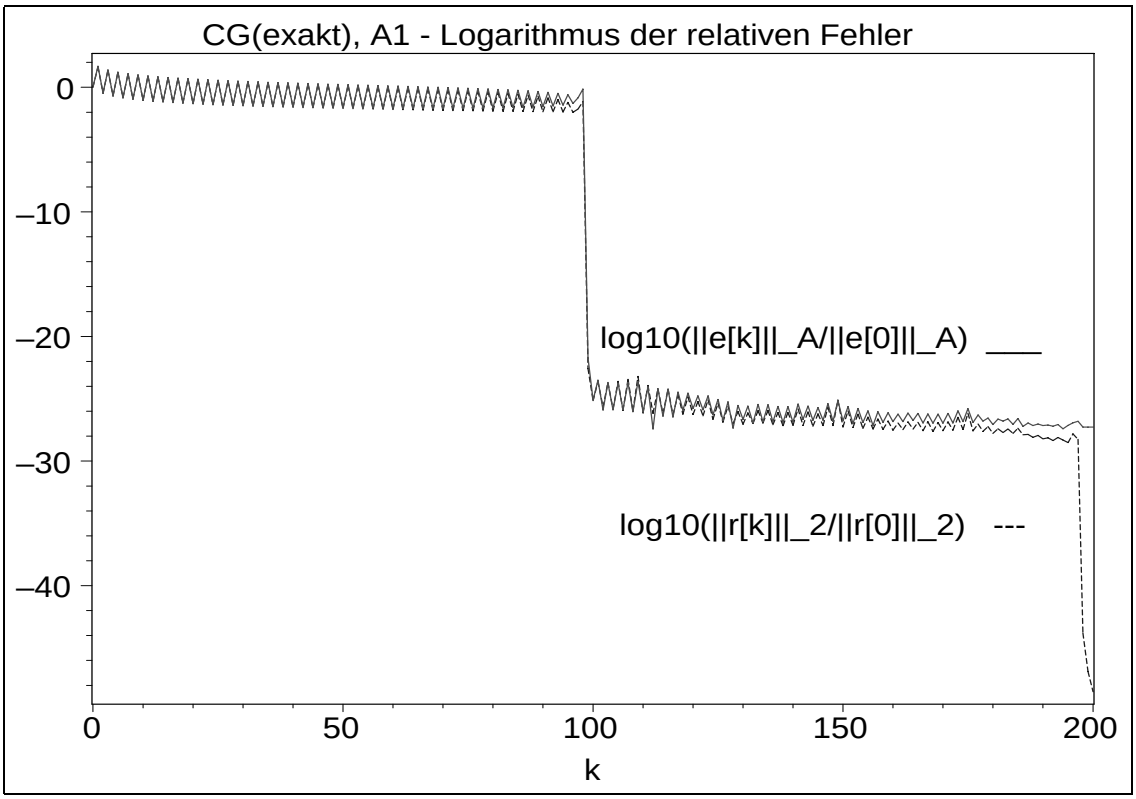
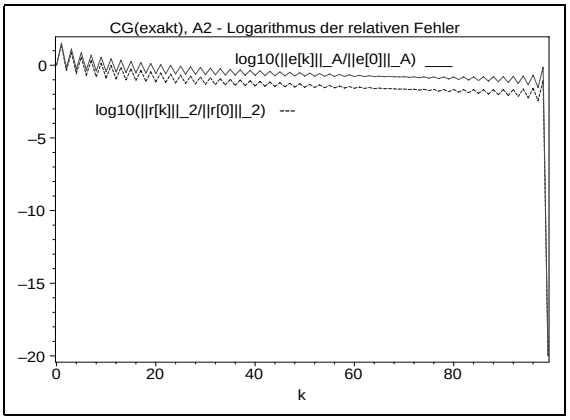


Abb. 7.16
Dateien
$cg4a150.ps,\ cg4a250.ps,\ cg4a350.ps,$
Matrizen $A_{1,2,3}(50,50),\ c=0,$
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k=0,1,...$





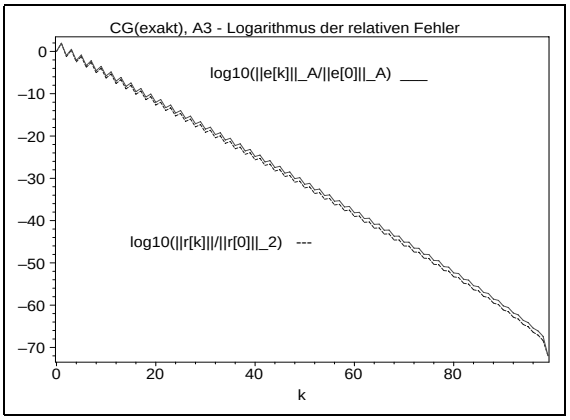
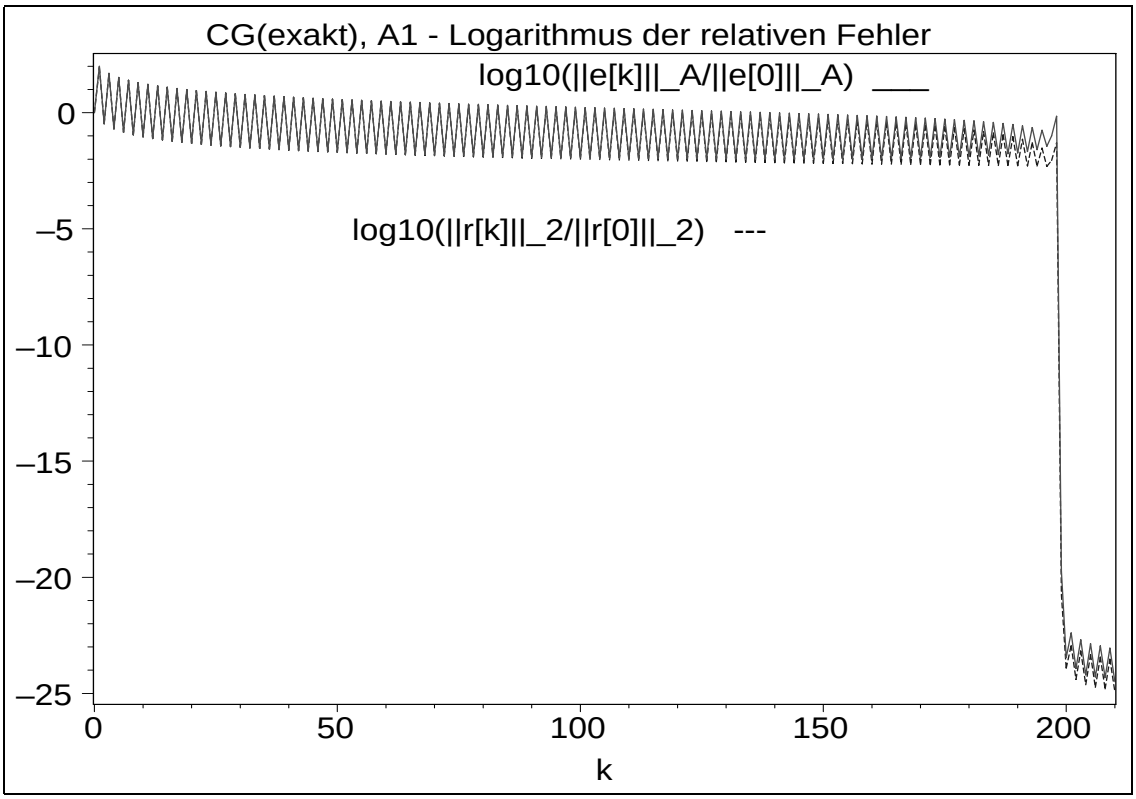
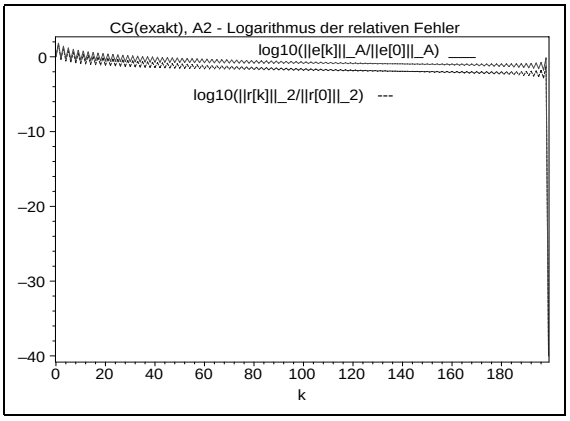


Abb. 7.17
Dateien
$cg4a11h.ps$, $cg4a21h.ps$, $cg4a31h.ps$,
Matrizen $A_{1,2,3}(100,100)$, $c = 0$,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





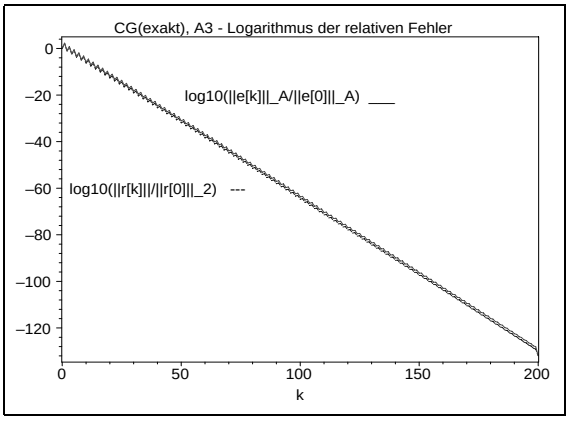
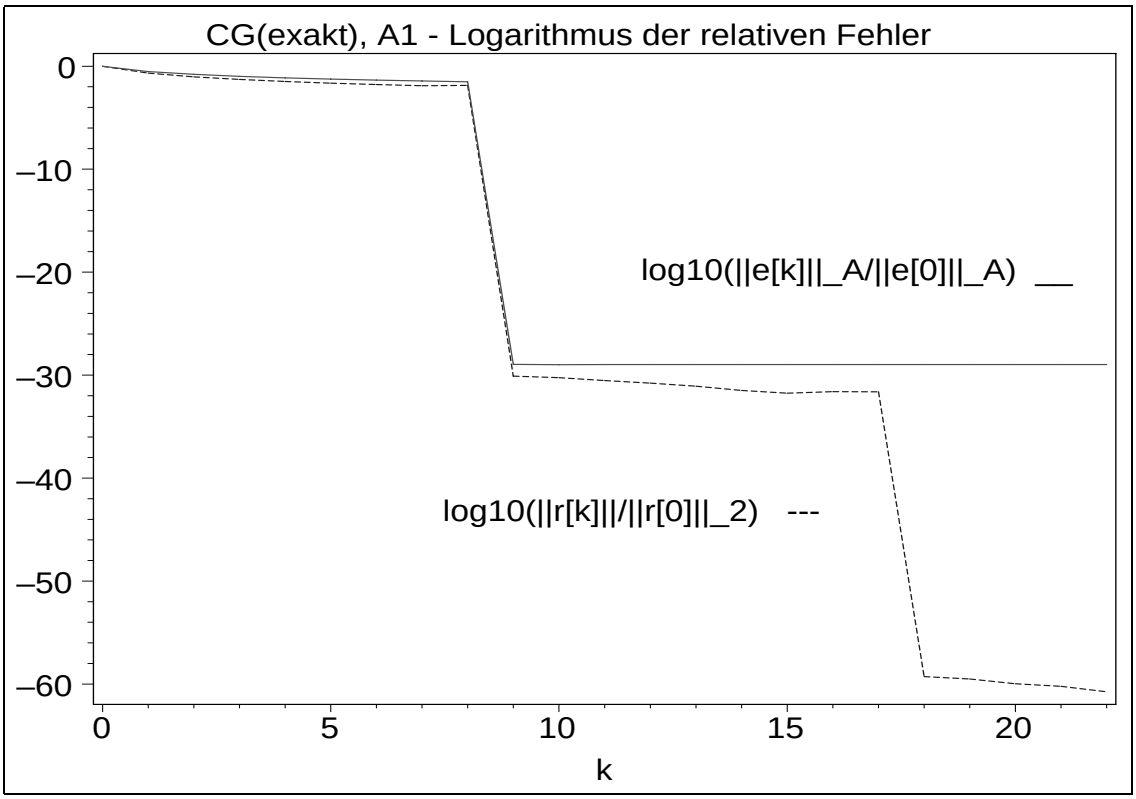
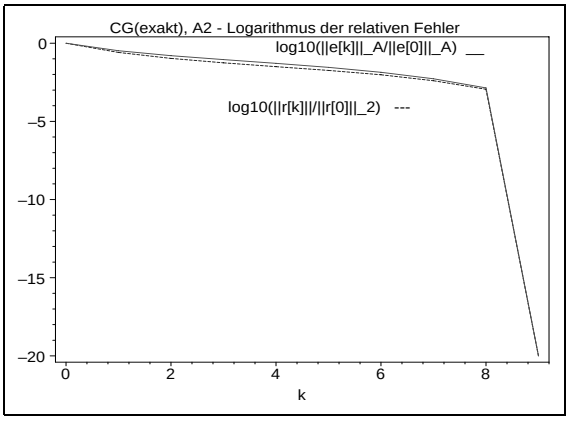


Abb. 7.18
Dateien
$cg4a12h.ps$, $cg4a22h.ps$, $cg4a32h.ps$,
Matrizen $A_{1,2,3}(200, 200)$, $c = 0$,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





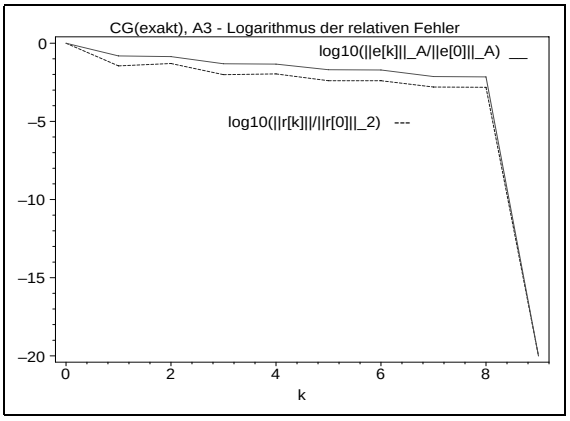
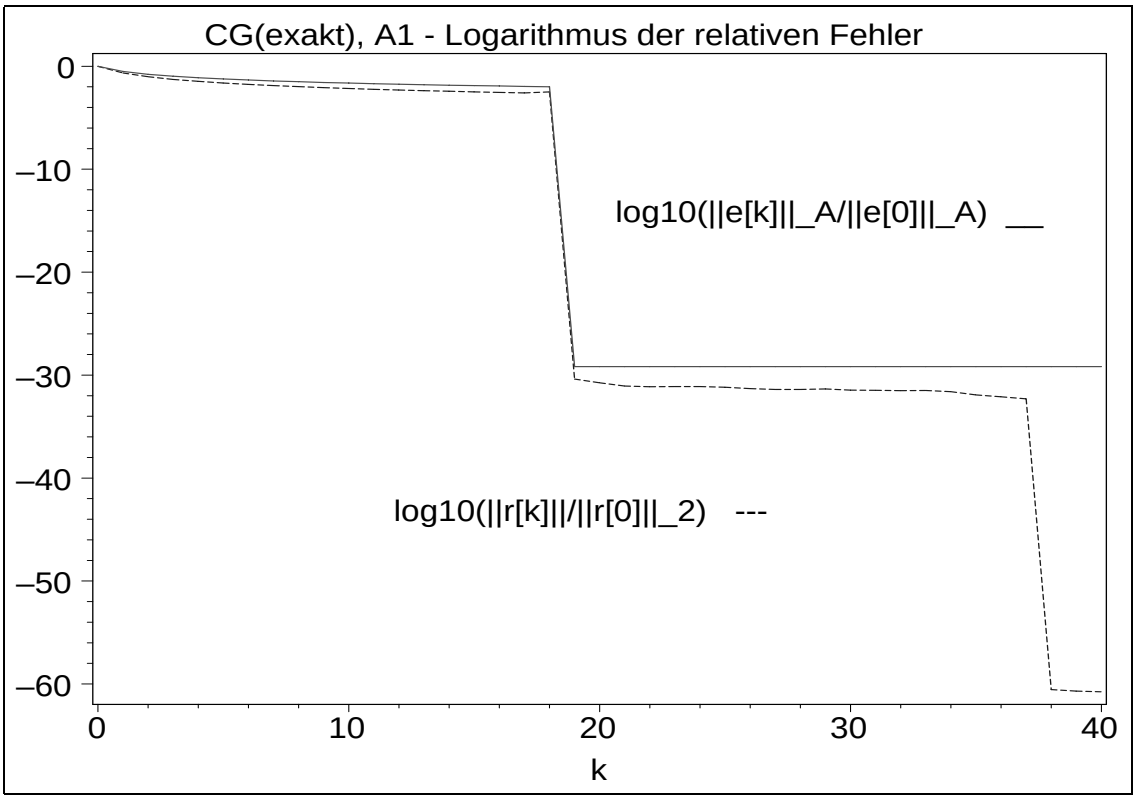
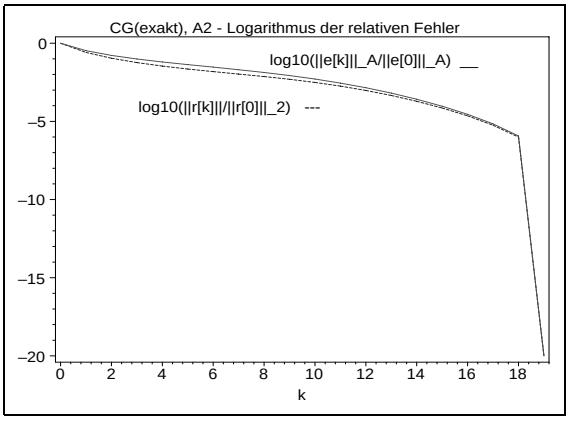


Abb. 7.19
Dateien
$cg4a110c.ps$, $cg4a210c.ps$, $cg4a310c.ps$,
Matrizen $A_{1,2,3}(10,10),\ c = 1 + 10^{-6}$,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...$





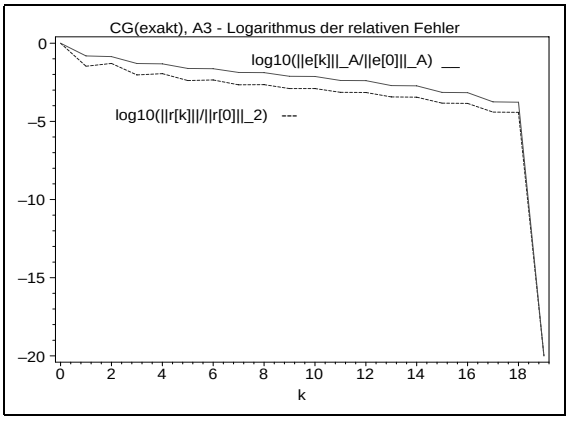
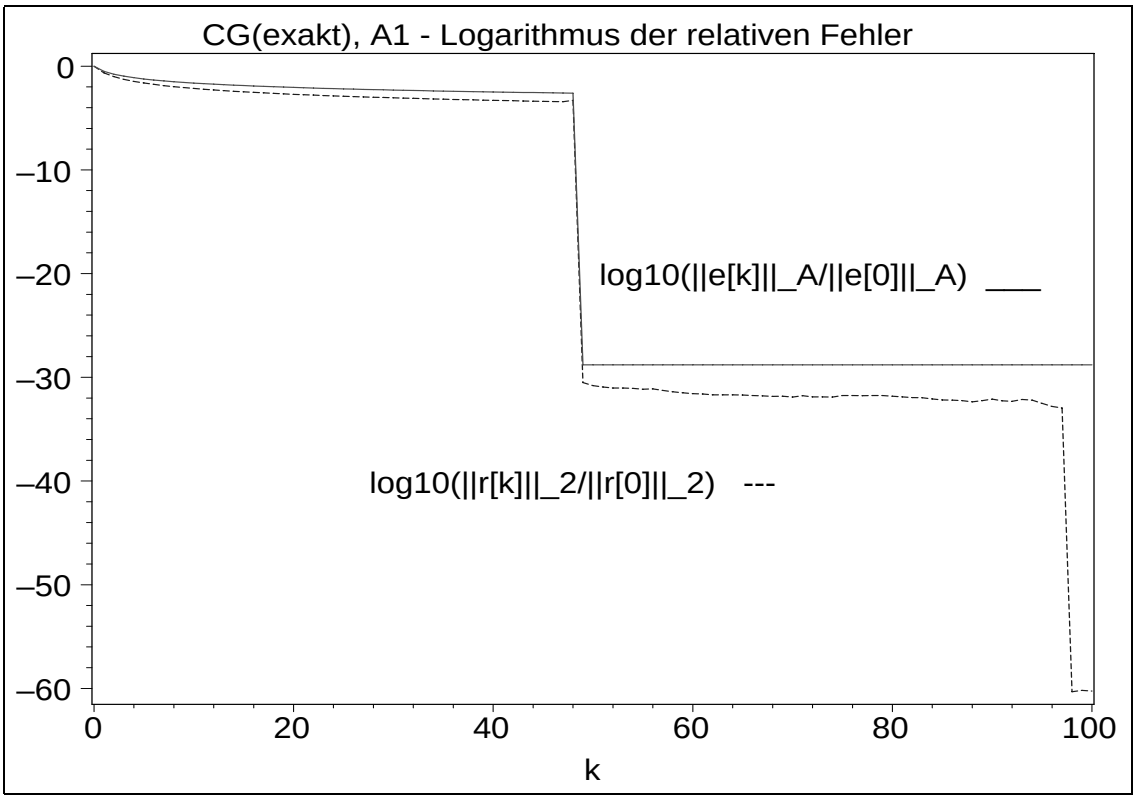
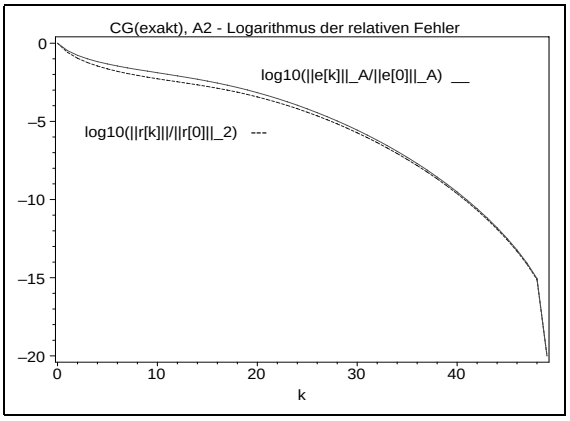


Abb. 7.20
Dateien
$cg4a120c.ps$, $cg4a220c.ps$, $cg4a320c.ps$,
Matrizen $A_{1,2,3}(20,20)$, $c = 1 + 10^{-6}$,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





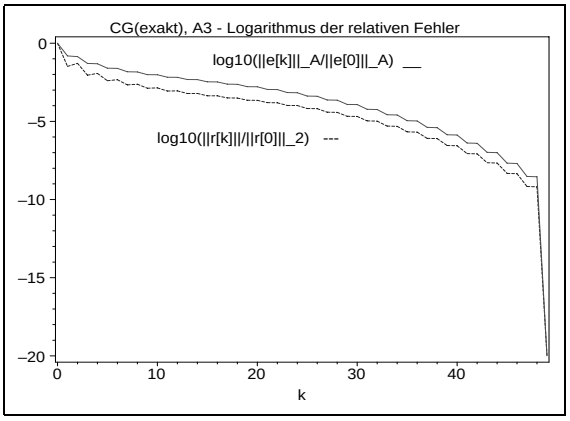
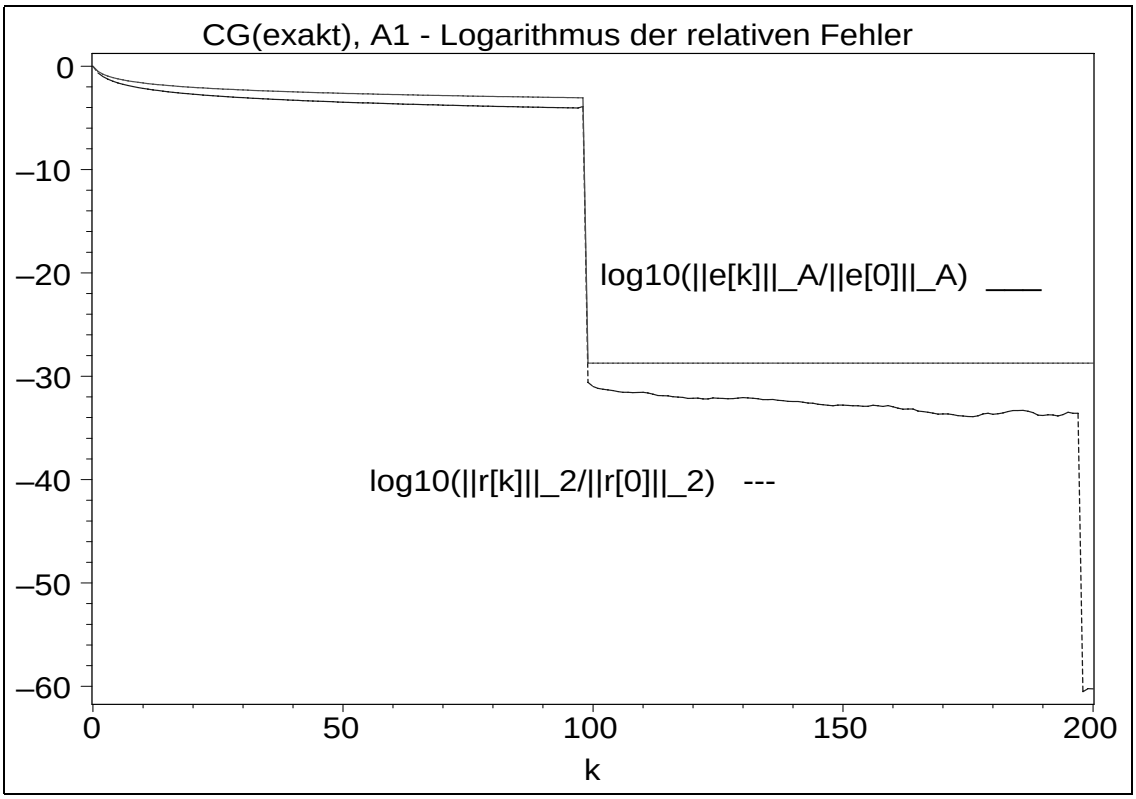
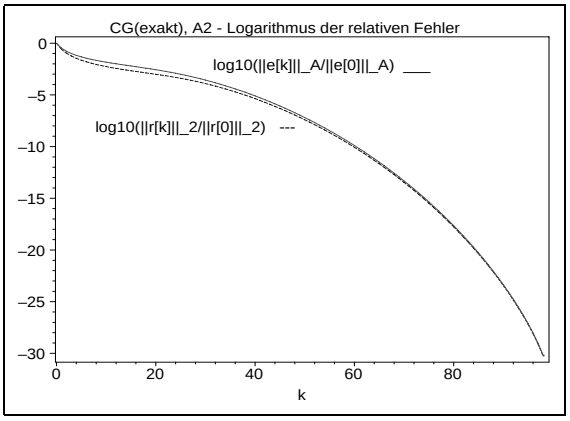


Abb. 7.21
Dateien
$cg4a150c.ps$, $cg4a250c.ps$, $cg4a350c.ps$,
Matrizen $A_{1,2,3}(50,50),\ c = 1 + 10^{-6}$,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





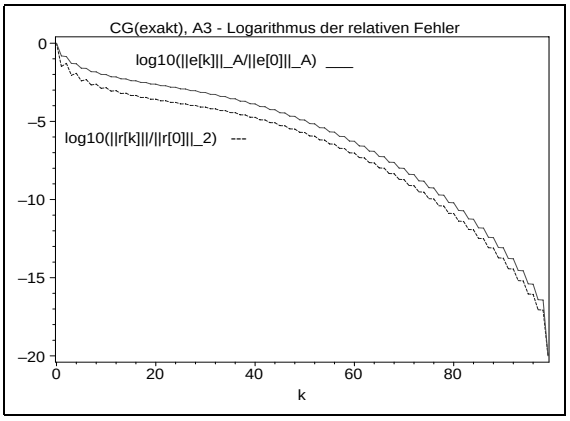


Abb. 7.22
Dateien
$cg4a11hc.ps$, $cg4a21hc.ps$, $cg4a31hc.ps$,
Matrizen $A_{1,2,3}(100,100)$, $c = 1+10^{-6}$,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$

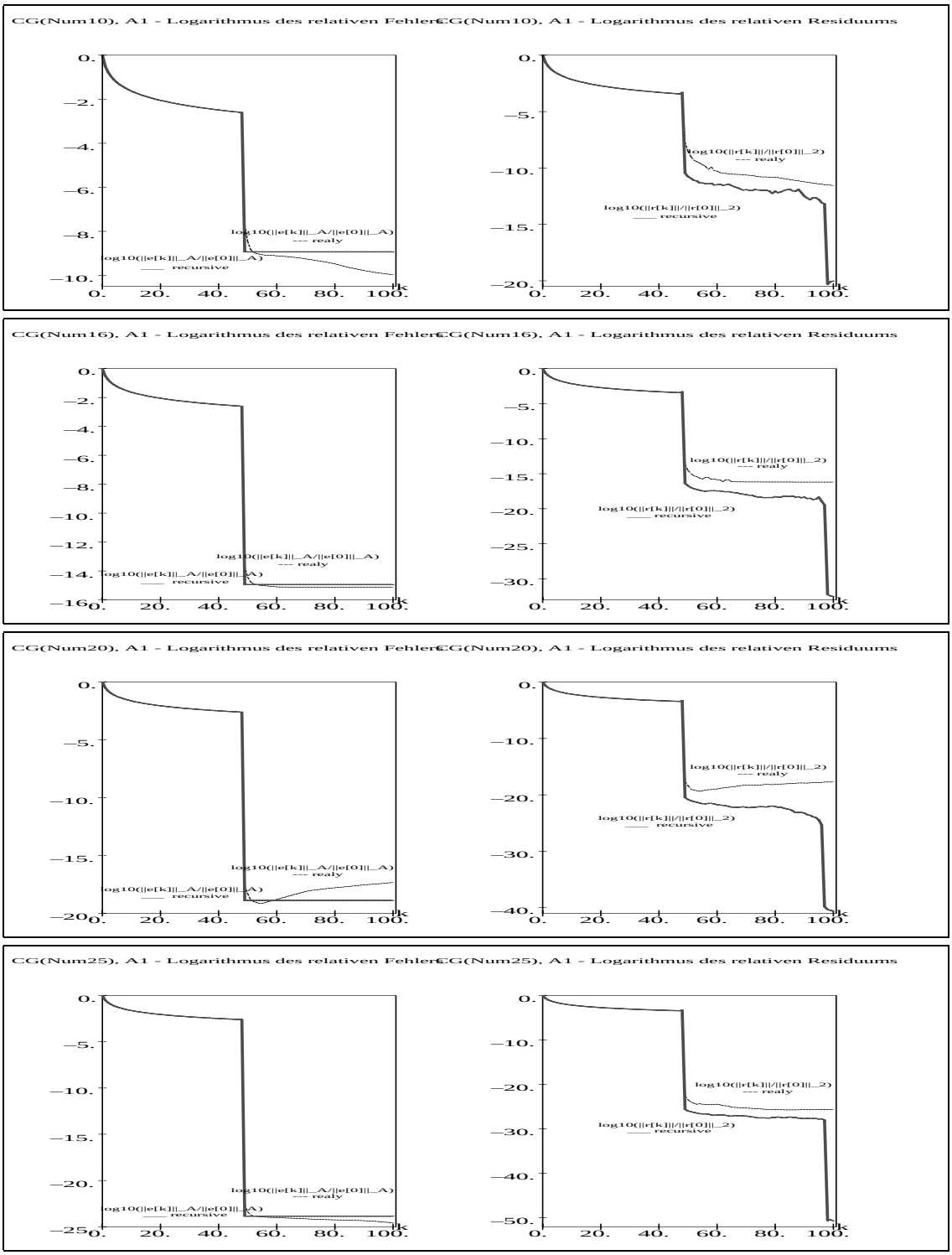


Abb. 7.23 Dateien $cg4a1110.ps, cg4a1116.ps, cg4a1120.ps, cg4a1125.ps$,
$A_1(50,50)$, $c = 1 + 10^{-6}$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

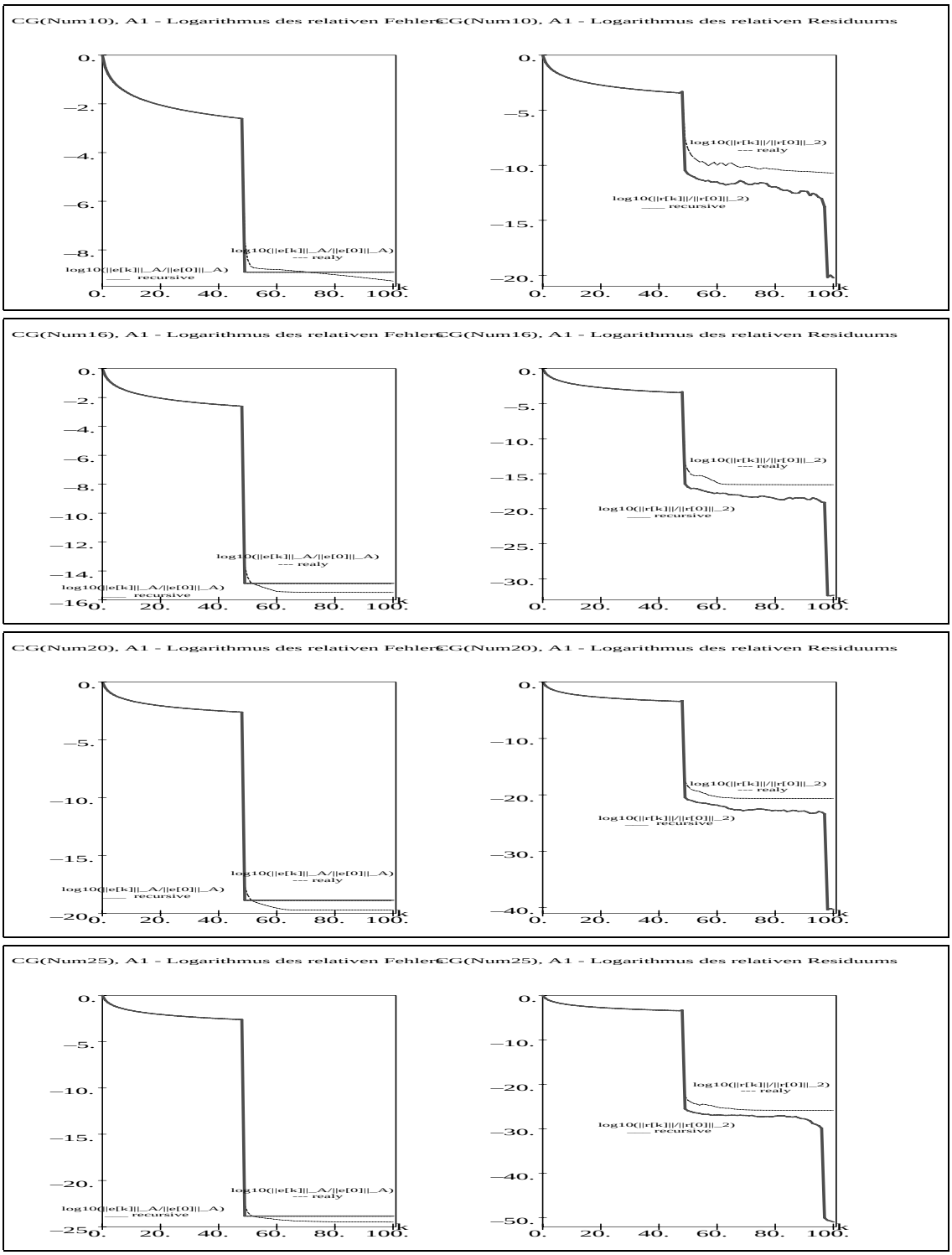


Abb. 7.24 Dateien $cg4a1310.ps, cg4a1316.ps, cg4a1320.ps, cg4a1325.ps$,
$A_1(50,50)$, $c = 1 + 10^{-6}$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

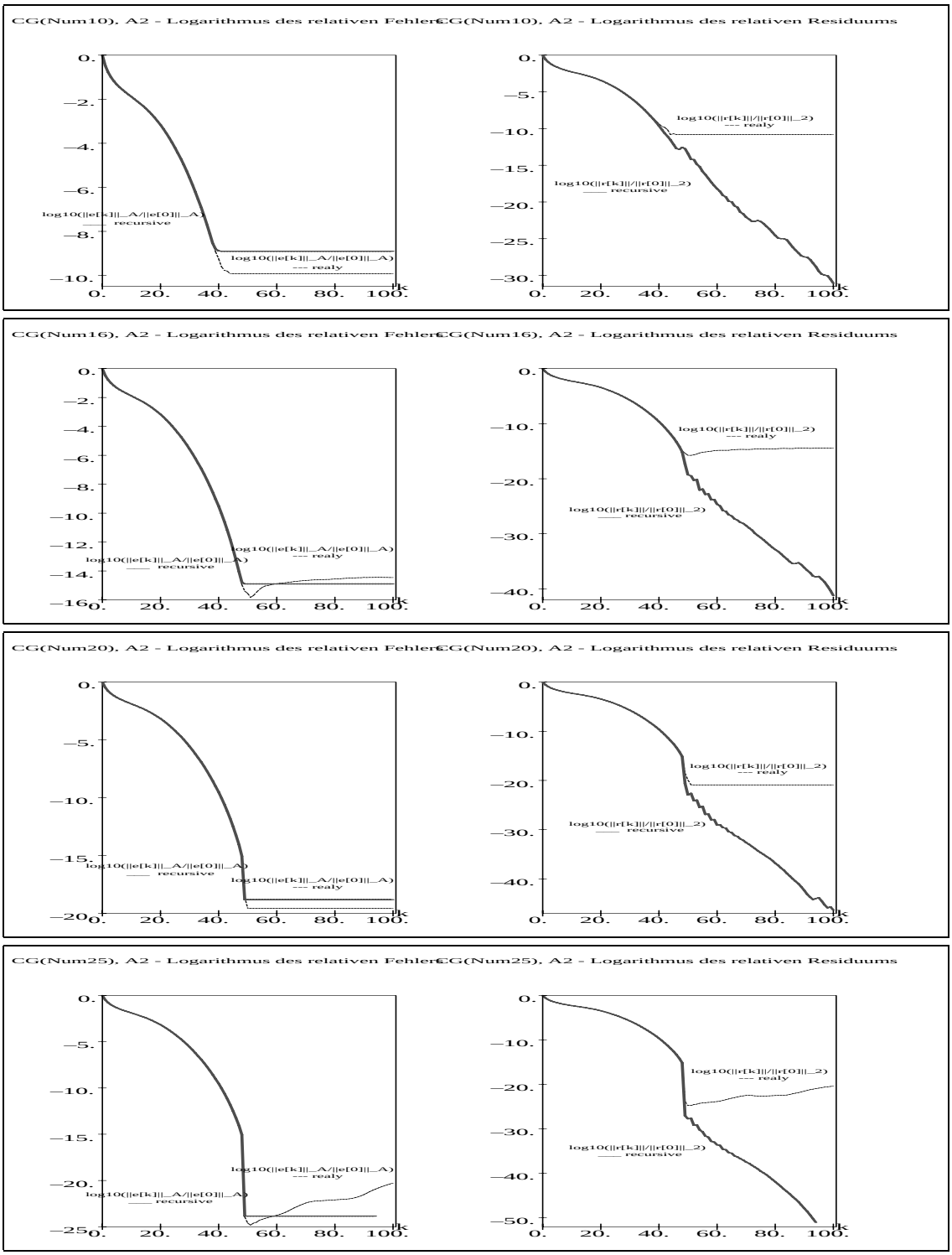


Abb. 7.25 Dateien $cg4a2110.ps, cg4a2116.ps, cg4a2120.ps, cg4a2125.ps$,
$A_2(50,50)$, $c = 1 + 10^{-6}$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

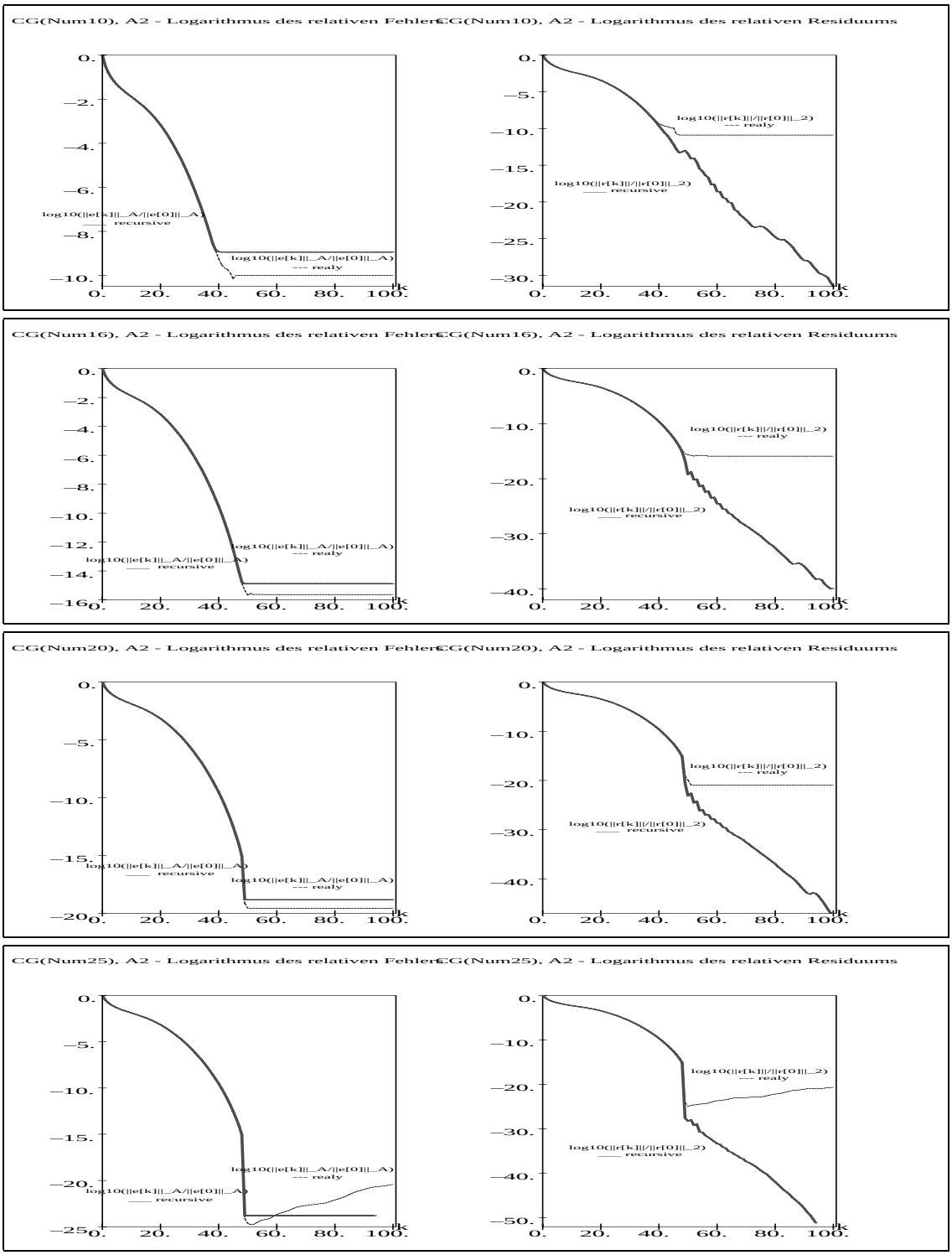


Abb. 7.26 Dateien $cg4a2310.ps, cg4a2316.ps, cg4a2320.ps, cg4a2325.ps$,
$A_2(50,50)$, $c = 1 + 10^{-6}$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

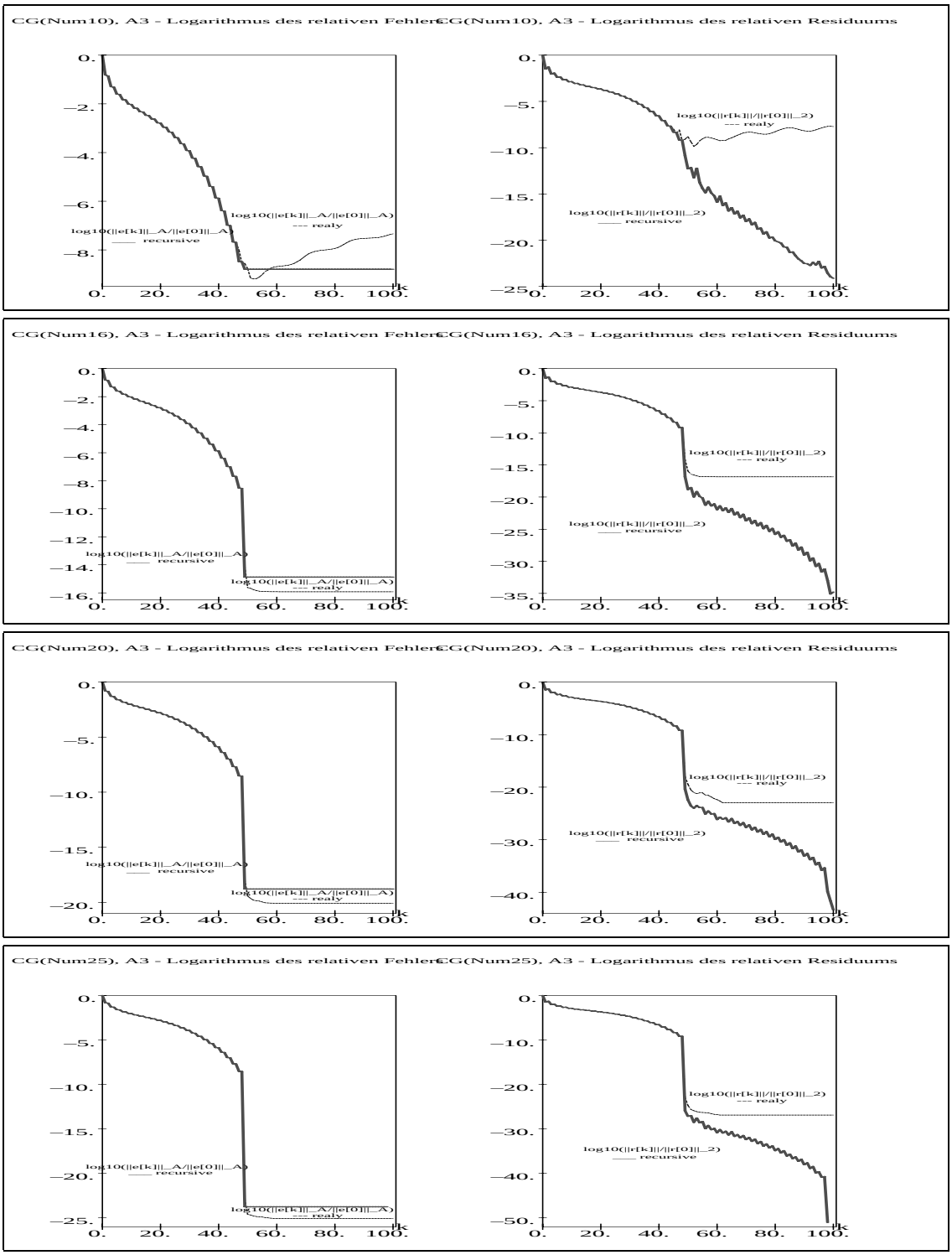


Abb. 7.27 Dateien $cg4a3110.ps, cg4a3116.ps, cg4a3120.ps, cg4a3125.ps$,
$A_3(50,50)$, $c = 1 + 10^{-6}$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

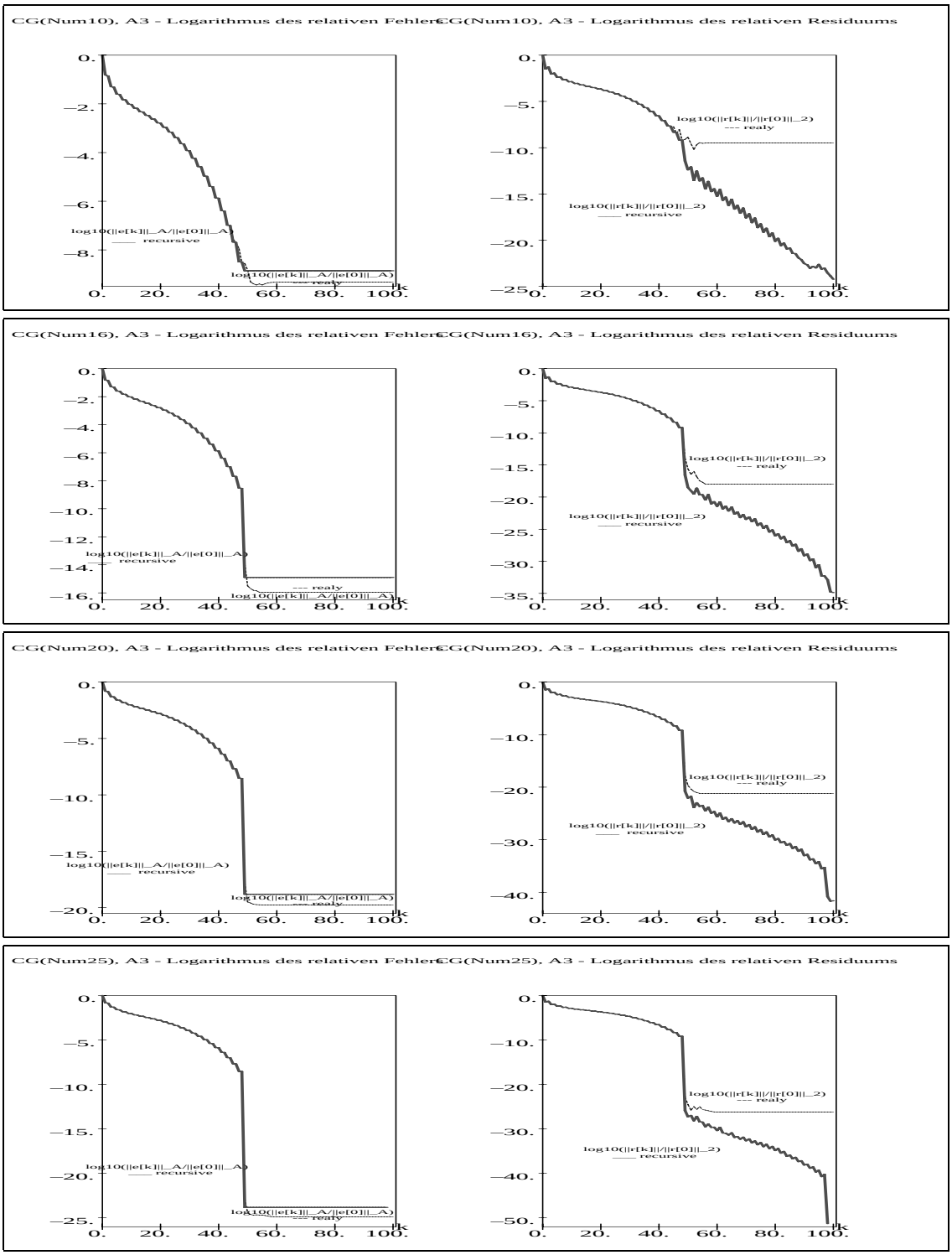


Abb. 7.28 Dateien *cg4a3310.ps*, *cg4a3316.ps*, *cg4a3320.ps*, *cg4a3325.ps*,
$A_3(50,50)$, $c = 1 + 10^{-6}$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.1.5 Beispiel 5

```
# Beispiel 5, A=A'>0 mit Tridiagonalstruktur
# Bsp. 5.1, A1 Tridiagonalmatrizen mit spezieller Struktur und Shift c
# 1D-Laplace-Operator auf aequidistantem Gitter,
# 3-Punkte-Differenzenstern
 c:=0:
 n:=100:
 A1:=band([-1,2,-1],n):
 A1:=evalm(A1+c*diag(1$n));
 b1:=vector(n,[1+c,c$(n-2),1+c]);
# Loesung
 xs1:=vector(n,[1$n]):  # 1,1,1,....
-----------------------------------------------------------------------

# Bsp. 5.2, A2 Blocktridiagonalmatrizen mit spezieller Struktur und
# Shift c
# 2D-Laplace-Operator auf quadratischem Gitter,
# 5-Punkte-Differenzenstern
# Generierung, Dauer: ca 40 sec bei n1=25, PC 800MHz
 c:=0:
 n1:=10:   # 25:
 n:=n1^2:
 sta:=time():
 T:=band([-1,4,-1],n1):
 A2:=diag(T$n1):
 i:='i':
 for i from 1 to n-n1 do
  A2[i,i+n1]:=-1;
  A2[i+n1,i]:=-1;
 end do:
 A2:=evalm(A2):
 print(time()-sta);
 A2:=evalm(A2+c*diag(1$n)):

# Multiplikation, Dauer: ca 70 sec bei n1=25, n=625, PC 800MHz
# Zeiteinsparung durch viele Nullen in der Matrix
 xs2:=evalm(vector(n,[1$n])):
 sta:=time():
 b2:=evalm(A2&*xs2)   # rechte Seite
 print(time()-sta);

# Loesung
 xs2:=vector(n,[1$n]):
```

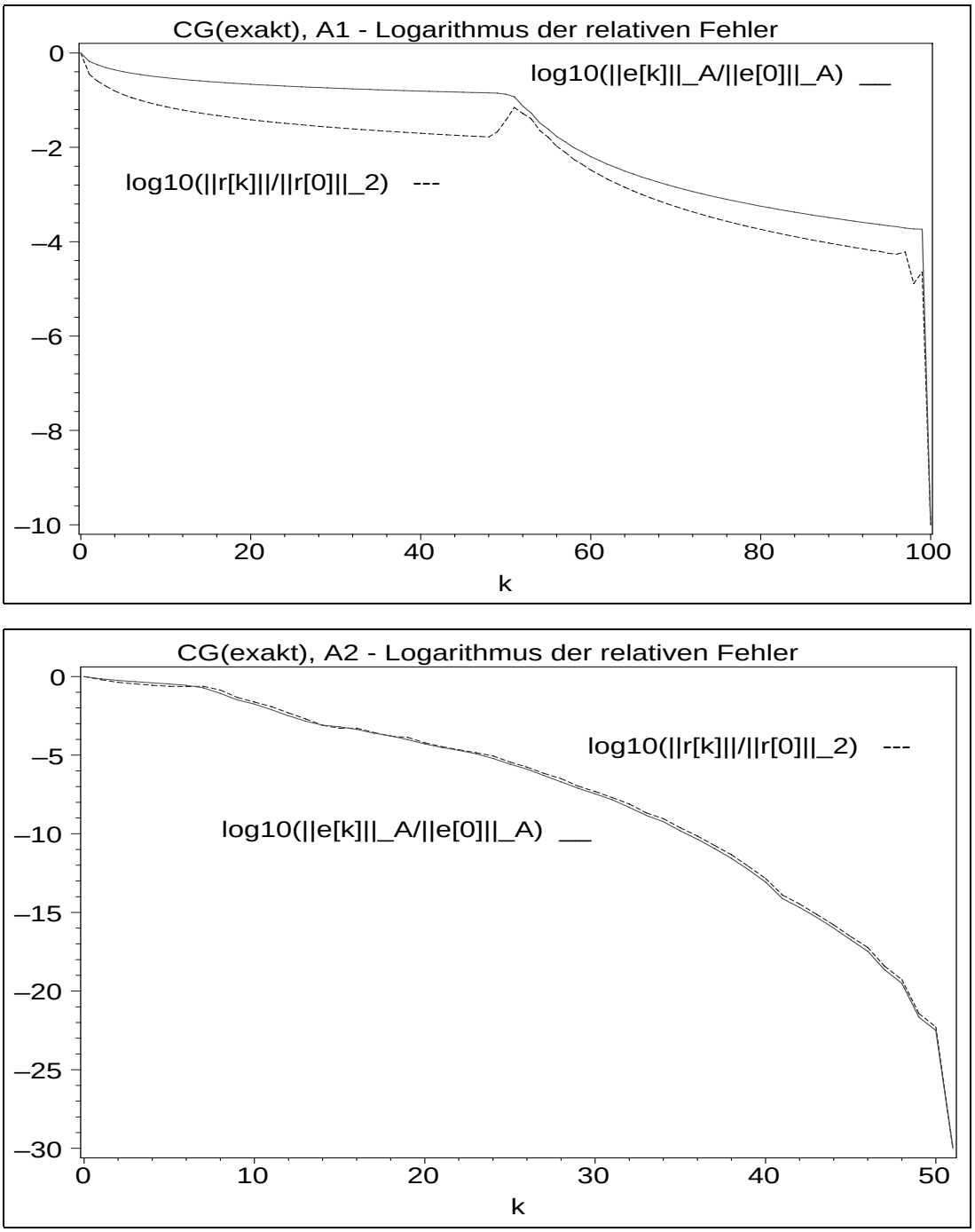


Abb. 7.29 Dateien *cg5a1_01.ps*, *cg5a2_01.ps*,
Matrix $A_{1,2}(100, 100)$, $c = 0$ zu Beispiel 5,
CG: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$

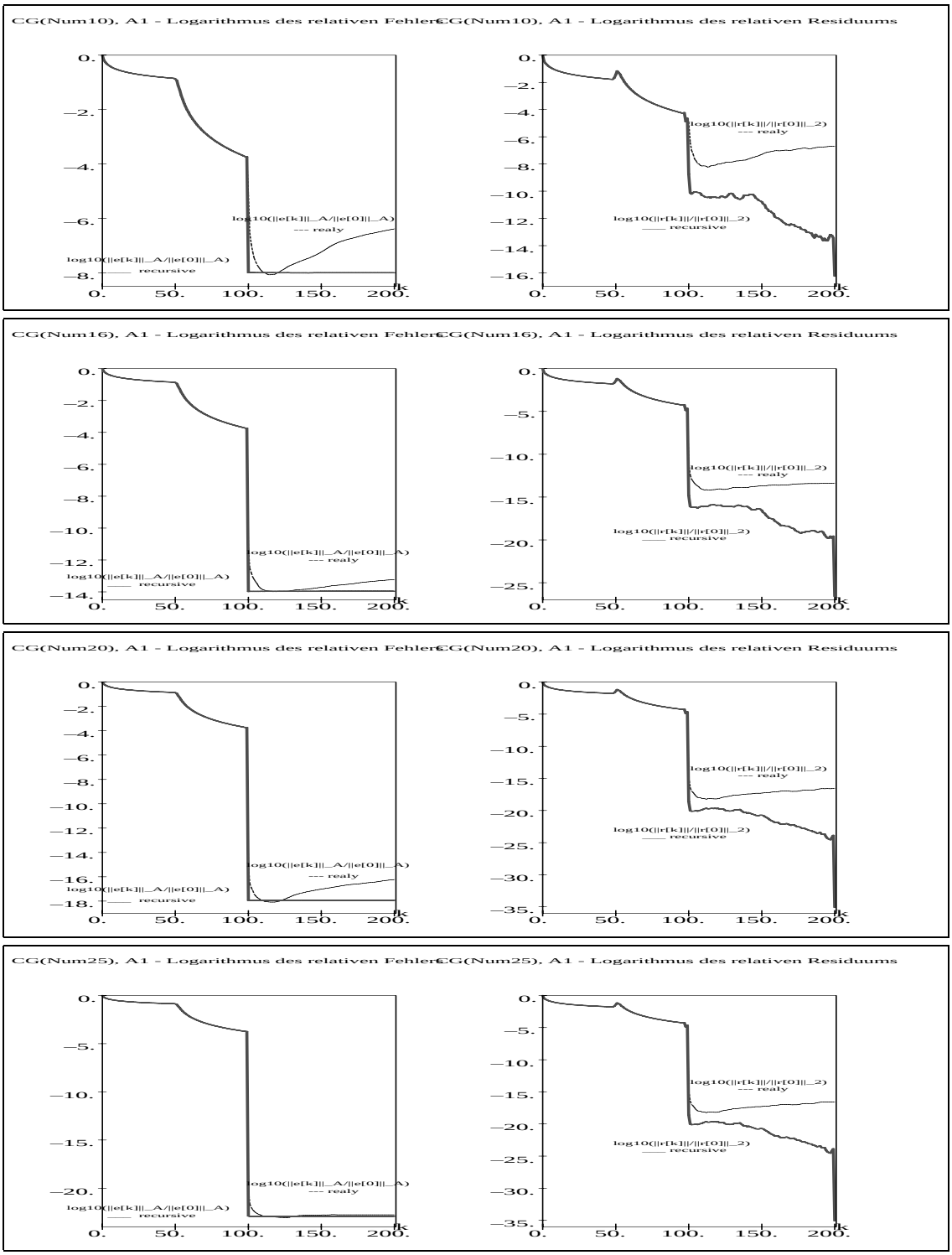


Abb. 7.30 Dateien $cg5a1110.ps, cg5a1116.ps, cg5a1120.ps, cg5a1125.ps$,
$A_1(100, 100)$, $c = 0$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

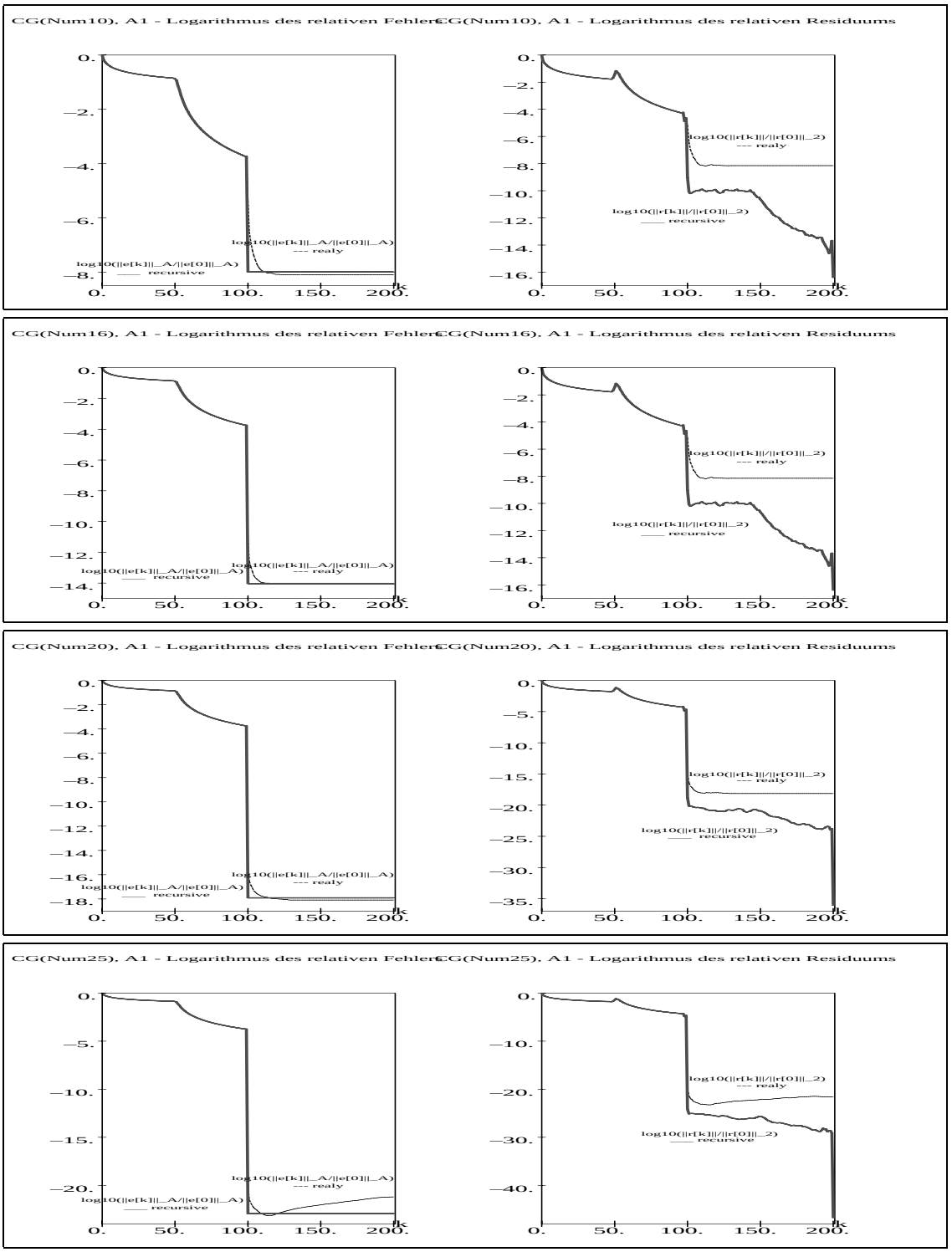


Abb. 7.31 Dateien $cg5a1310.ps, cg5a1316.ps, cg5a1320.ps, cg5a1325.ps$,
$A_1(100, 100)$, $c = 0$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

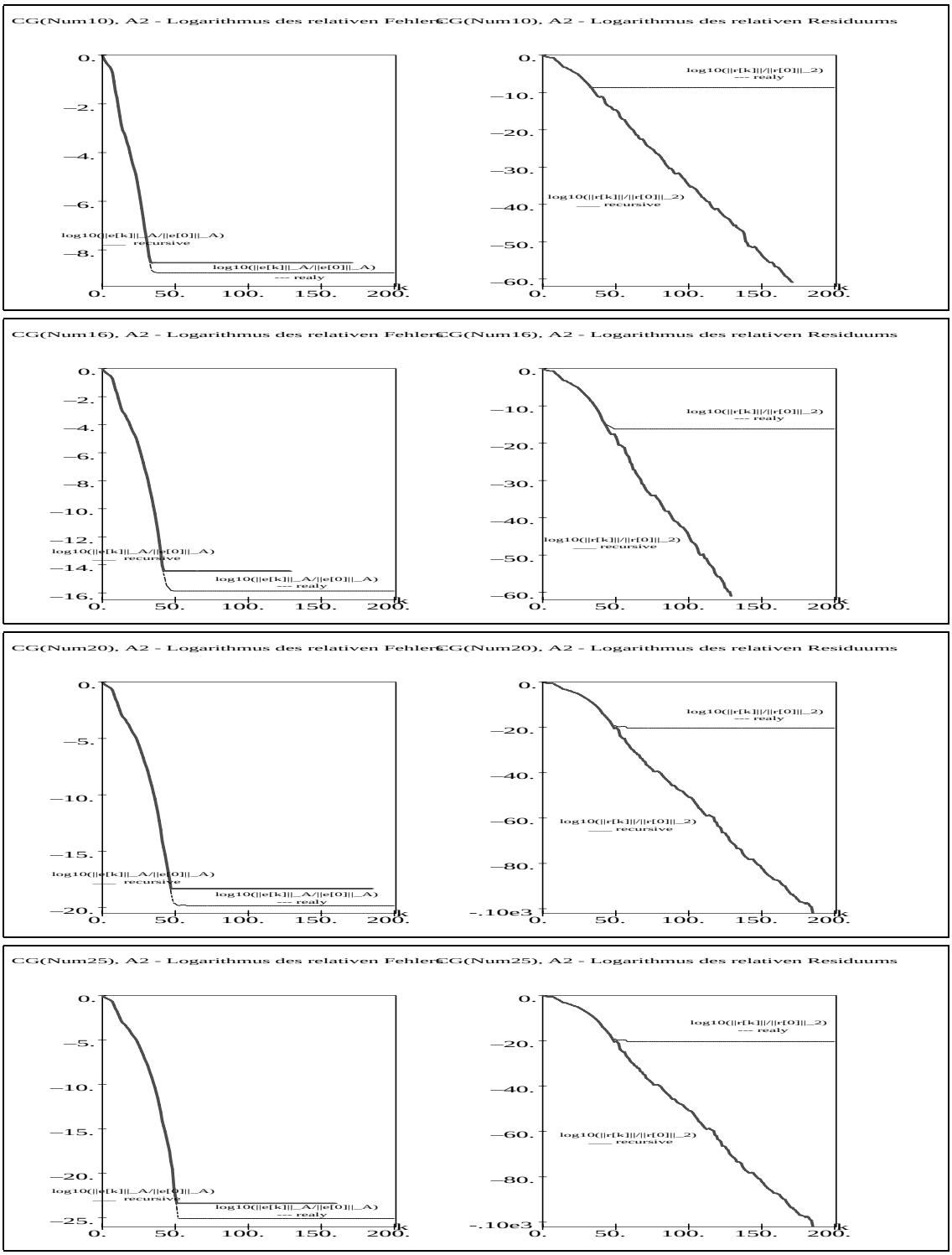


Abb. 7.32 Dateien $cg5a2110.ps, cg5a2116.ps, cg5a2120.ps, cg5a2125.ps$,
$A_2(100, 100)$, $c = 0$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

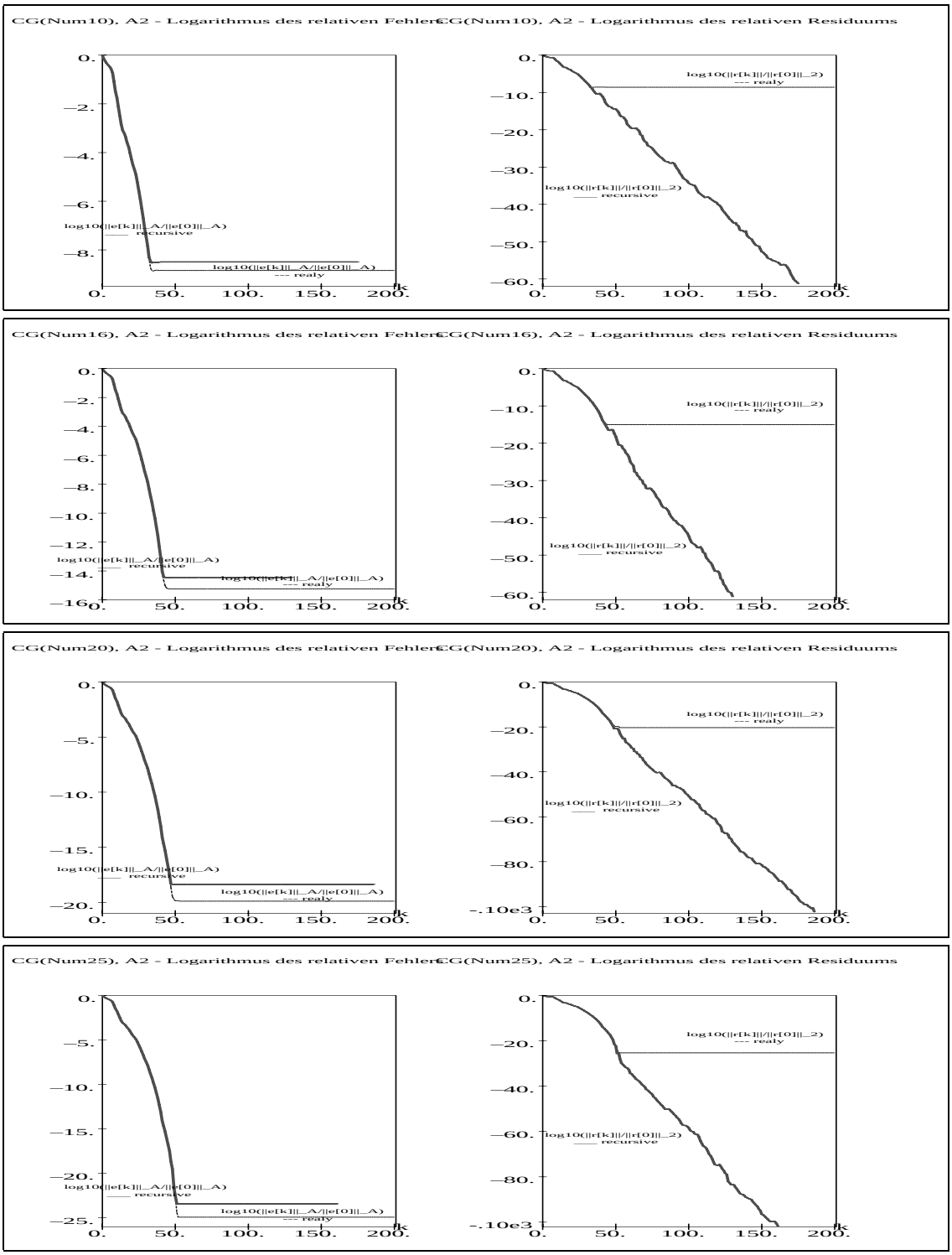


Abb. 7.33 Dateien $cg5a2310.ps, cg5a2316.ps, cg5a2320.ps, cg5a2325.ps$,
$A_2(100,100)$, $c = 0$,
CG: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.2 CR mit Fehlererinnerung

Zunächst verweisen wir auf die Implementierung des CR in Teil II, Abschnitt 4.3. Dabei werden wir wie beim CG bei der Berechnung der Schrittzahl α sowie bei der Bestimmung des Residuums r verschiedene Varianten realisieren.

Einfache Darstellung des CR

$x^{(0)}$ Startvektor, $r^{(0)} = b - Ax^{(0)}$ Residuum, $p^{(0)} = r^{(0)}$ Suchrichtung

$k = 0, 1, 2, ...$

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \alpha_k p^{(k)}, \\ &\quad \text{wobei } \alpha_k = \frac{r^{(k)T} A r^{(k)}}{(Ap^{(k)})^T A p^{(k)}} = \frac{\|r^{(k)}\|_A^2}{\|Ap^{(k)}\|_2^2} > 0 \\ r^{(k+1)} &= b - Ax^{(k+1)} \qquad \text{(explizite Berechnung)} \\ &= r^{(k)} - \alpha_k A p^{(k)} \qquad \text{(rekursive Berechnung)} \\ p^{(k+1)} &= r^{(k+1)} + \beta_k p^{(k)}, \\ &\quad \text{wobei } \beta_k = \frac{r^{(k+1)T} A r^{(k+1)}}{r^{(k)T} A r^{(k)}} = \frac{\|r^{(k+1)}\|_A^2}{\|r^{(k)}\|_A^2} > 0 \end{aligned}$$

In jedem Iterationsschritt ist bei rekursiver Berechnung von $r^{(k+1)}$ nur eine Matrix-Vektor-Multiplikation zu machen, da $Ap^{(k)}$ und $Ar^{(k)}$ auseinander abgeleitet werden können.

Weitere Problemgrößen, die ebenfalls in die Betrachtungen und zu Testzwecken einbezogen werden, verursachen meistens zusätzlicher Berechnungsaufwand.

Funktionale

$$\begin{aligned} Q(x) &= \frac{1}{2}\Big(\|e(x)\|_A^2 - x^{*T} b\Big) = \frac{1}{2} x^T A x - x^T b \text{ im Algorithmus} \\ Q(x^*) &= -\frac{1}{2} x^{*T} b \le 0 \\ R(x) &= \frac{1}{2}\Big(\|r(x)\|_2^2 - \|b\|_2^2\Big) = \frac{1}{2}(Ax)^T Ax - (Ax)^T b \text{ im Algorithmus} \\ R(x^*) &= -\frac{1}{2}\|b\|_2^2 \le 0 \end{aligned}$$

Fehler

$$\begin{aligned} e(x) &= x^* - x \\ \|e(x)\|_A &= \|x^* - x\|_A = \sqrt{(x^* - x)^T A (x^* - x)} = \sqrt{2Q(x) + x^{*T} b} \\ \tilde{e}(x) &= \sqrt{(x^* - x)^T A (x^* - x)}, \quad |\tilde{e}(x)| \text{ im Algorithmus} \end{aligned}$$

Residuum

$$\begin{aligned} r(x) &= b - Ax \\ \|r(x)\|_2 &= \|b - Ax\|_2 = \sqrt{r(x)^T r(x)} = \sqrt{2R(x) + b^T b} \\ \|r(x)\|_2^2 &= r(x)^T r(x) \text{ im Algorithmus} \end{aligned}$$

Kontrolle, ob mit Beendigung des Algorithmus für $e^{(k)} = x^* - x^{(k)}$ gelten

$$\begin{aligned} \|e^{(k)}\|_2 &\approx \|e^{(k)}\|_A \\ \|e^{(k)}\|_2 &= \|A^{-1} r^{(k)}\|_2 \end{aligned}$$

Falls $A = A^T$ ist, gilt

$$\|r^{(0)}\|_2 \geq \|r^{(1)}\|_2 \geq \|r^{(2)}\|_2 \geq ... \quad \to 0,$$

Das ist ein Vorteil, weil $\|r^{(k)}\|_2$ für das Abbruchkriterium/Iterationssteuerung eingesetzt wird.
Falls $A = A^T > 0$ ist, gilt ebenfalls

$$\|e^{(0)}\|_2 \geq \|e^{(1)}\|_2 \geq \|e^{(2)}\|_2 \geq ... \quad \to 0$$

und meistens auch

$$\|e^{(0)}\|_A \geq \|e^{(1)}\|_A \geq \|e^{(2)}\|_A \geq ... \quad \to 0$$

Letztere Ungleichungskette ist oft auch erfüllt für die Matrix $A = A^T$ indefinit.

Die Vektoren
– Residuum und Abstiegsrichtung $r^{(k)}$ sind A-orthogonal (A-konjugiert), d.h.
$Ar^{(k)} \perp r^{(j)},\ k \neq j,$
– Suchrichtungen $p^{(k)}$ sind i. Allg. nicht A-orthogonal, aber $Ap^{(k+1)} \perp Ap^{(k)}$,
– und es gilt

$$\text{span}\{r^{(0)}, r^{(1)}, ..., r^{(n-1)}\} = \text{span}\{p^{(0)}, p^{(1)}, ..., p^{(n-1)}\} = \text{span}\{r^{(0)}, Ar^{(0)}, ..., A^{n-1} r^{(0)}\}.$$

Nachfolgend die beiden Algorithmen:

Algorithmus für die Implementierung mit Indizierung

Algorithmus für die Implementierung, weitgehend ohne Indizierung

mit:

ε Toleranz für die Residuumnorm $\|r^{(k)}\|_2$ bzw. $\|r\|_2$
maxiter maximale Iterationsanzahl

$k = 0,$ $x^{(0)}$ Startvektor, Berechnung von $Q(x^{(0)}), R(x^{(0)})$

$r^{(0)} = b - Ax^{(0)}$ Residuum, $t^{(0)} = Ar^{(0)}$

$\alpha_r = t^{(0)T} r^{(0)} = r^{(0)T} A r^{(0)}$

$p^{(0)} = r^{(0)}$ Suchrichtung, $lp = p^{(0)}$ letzte Suchrichtung

$s^{(0)} = Ap^{(0)} = t^{(0)}$

$$\alpha_s = \begin{cases} \|s^{(0)}\|_2^2, & \text{falls } varianta = 1 \\ \sqrt{s^{(0)T} s^{(0)}}^2, & \text{falls } varianta = 2 \\ s^{(0)T} s^{(0)}, & \text{falls } varianta = 3 \\ \sum_{i=1}^{n} |s_i^{(0)}|^2, & \text{falls } varianta = 4 \end{cases}$$

$\gamma_0 = r^{(0)T} r^{(0)} = \|r^{(0)}\|_2^2$

Speichern von $Q(x^{(0)}), R(x^{(0)}), |\tilde{e}(x^{(0)})|, \|e^{(0)}\|_2, \sqrt{\gamma_0}$ in

$\texttt{Qv[1]}, \texttt{Rv[1]}, \texttt{epsv[1]}, \texttt{epsv2[1]}, \texttt{rv[1]}$

evtl. Ausgabe und/oder Fileausgabe von $0, x^{(0)}, Q(x^{(0)}), R(x^{(0)}), r^{(0)}, \gamma_0$

while $(\sqrt{\gamma_k} > \varepsilon)$ & $(k < maxiter)$ **do**

falls $\alpha_r = 0,$ dann Abbruch mit Ergebnis $x^{(k)}, k$

$$\begin{aligned} \alpha &= \alpha_r / \alpha_s \\ x^{(k+1)} &= x^{(k)} + \alpha p^{(k)} \\ r^{(k+1)} &= \begin{cases} r^{(k)} - \alpha s^{(k)} = r^{(k)} - \alpha A p^{(k)}, & \text{falls } variantr = 1 \\ b - Ax^{(k+1)}, & \text{falls } variantr = 2 \end{cases} \\ \gamma_{k+1} &= r^{(k+1)T} r^{(k+1)} = \|r^{(k+1)}\|_2^2 \\ \beta_r &= 1/\alpha_r \\ t^{(k+1)} &= Ar^{(k+1)} \\ \alpha_r &= t^{(k+1)T} r^{(k+1)} = r^{(k+1)T} A r^{(k+1)}, \quad \beta = \alpha_r \beta_r = \frac{r^{(k+1)T} A r^{(k+1)}}{r^{(k)T} A r^{(k)}} \\ p^{(k+1)} &= r^{(k+1)} + \beta p^{(k)}, \quad lp = p^{(k+1)} \text{ letzte Suchrichtung} \\ s^{(k+1)} &= Ap^{(k+1)} = t^{(k+1)} + \beta s^{(k)} \\ k &= k + 1 \end{aligned}$$

$$\alpha_s = \begin{cases} \|s^{(k)}\|_2^2, & \text{falls } varianta = 1 \\ \sqrt{s^{(k)T} s^{(k)}}^2, & \text{falls } varianta = 2 \\ s^{(k)T} s^{(k)}, & \text{falls } varianta = 3 \\ \sum_{i=1}^{n} |s_i^{(k)}|^2, & \text{falls } varianta = 4 \end{cases}$$

Berechnung von $Q(x^{(k)}),\ R(x^{(k)})$

evtl. Ausgabe und/oder Fileausgabe von $k, x^{(k)}, Q(x^{(k)}), R(x^{(k)}), p^{(k)}, r^{(k)}, \gamma_k$

Ausgabe und evtl. Fileausgabe von $k, \sqrt{\gamma_k}$

Speichern von $Q(x^{(k)}), R(x^{(k)}), |\tilde{e}(x^{(k)})|, \|e(x^{(k)})\|_2, \sqrt{\gamma_k}$ in

$\texttt{Qv[k+1]}, \texttt{Rv[k+1]}, \texttt{epsv[k+1]}, \texttt{epsv2[k+1]}, \texttt{rv[k+1]}$

end while

Ausgabe von $\|x^* - x^{(k)}\|_2,\ \|A^{-1} r^{(k)}\|_2$ zwecks Vergleich mit $\|e^{(k)}\|_A, \|e^{(k)}\|_2$

$x^{(k)}, k$ Ergebnisse nach der Schleife

$k = 0$, $x = x0$, $x0$ Startvektor, Berechnung von $Q(x), R(x)$

$r = b - Ax$ Residuum, $t = Ar$

$\alpha_r = t^T r = r^T Ar$

$p = r$ Suchrichtung, $lp = p$ letzte Suchrichtung

$s = Ap = t$

$$\alpha_s = \begin{cases} \|s\|_2^2 & \text{falls } varianta = 1 \\ \sqrt{s^T s}^2, & \text{falls } varianta = 2 \\ s^T s, & \text{falls } varianta = 3 \\ \sum_{i=1}^{n} |s_i|^2, & \text{falls } varianta = 4 \end{cases}$$

$\gamma = r^T r = \|r\|_2^2$

Speichern von $Q(x), R(x), |\tilde{e}(x)|, \|e(x)\|_2, \sqrt{\gamma}$ in
$\texttt{Qv[1]}, \texttt{Rv[1]}, \texttt{epsv[1]}, \texttt{epsv2[1]}, \texttt{rv[1]}$

evtl. Ausgabe und/oder Fileausgabe von $0, x, Q(x), R(x), r, \gamma$

while $(\sqrt{\gamma} > \varepsilon)$ & $(k < maxiter)$ **do**

falls $\alpha_r = 0$, dann Abbruch mit Ergebnis x, k

$$\begin{aligned} \alpha &= \alpha_r/\alpha_s \\ x &= x + \alpha p \\ r &= \begin{cases} r - \alpha s = r - \alpha Ap, & \text{falls } variantr = 1 \\ b - Ax, & \text{falls } variantr = 2 \end{cases} \\ \gamma &= r^T r = \|r\|_2^2 \\ \beta_r &= 1/\alpha_r \\ t &= Ar \\ \alpha_r &= t^T r = r^T Ar, \quad \beta = \alpha_r \beta_r \\ p &= r + \beta p, \quad lp = p \text{ letzte Suchrichtung} \\ s &= Ap = t + \beta s \\ k &= k + 1 \\ \alpha_s &= \begin{cases} \|s\|_2^2, & \text{falls } varianta = 1 \\ \sqrt{s^T s}^2, & \text{falls } varianta = 2 \\ s^T s, & \text{falls } varianta = 3 \\ \sum_{i=1}^{n} |s_i|^2, & \text{falls } varianta = 4 \end{cases} \end{aligned}$$

Berechnung von $Q(x)$, $R(x)$

evtl. Ausgabe und/oder Fileausgabe von $k, x, Q(x), R(x), p, r, \gamma$

Ausgabe und evtl. Fileausgabe von $k, \sqrt{\gamma}$

Speichern von $Q(x), R(x), |\tilde{e}(x)|, \|e(x)\|_2, \sqrt{\gamma}$ in
$\texttt{Qv[k+1]}, \texttt{Rv[k+1]}, \texttt{epsv[k+1]}, \texttt{epsv2[k+1]}, \texttt{rv[k+1]}$

end while

Ausgabe von $\|x^* - x\|_2$, $\|A^{-1}r\|_2$ zwecks Vergleich mit $\|e(x)\|_A, \|e(x)\|_2$

x, k Ergebnisse nach der Schleife

Variantenauswahl

Das betrifft die Berechnung des Residuums r in 2 Varianten

1. **rekursiv** (`recursive, update`) mit $r = r - \alpha Ap,\ \alpha = \dfrac{(Ar,r)}{(Ap,Ap)} = \dfrac{r^T Ar}{(Ap)^T Ap}$,
2. **wirklich** (`realy, true`) mit $r = b - Ax$,

und 4 Varianten der Berechnung des Nenners $\alpha_s = s^T s,\ s = Ap$, in der Schrittzahl α

1. $\alpha_s = \|s\|_2^2$,
2. $\alpha_s = \left(\sqrt{s^T s}\right)^2$,
3. $\alpha_s = s^T s$,
4. $\alpha_s = \sum_{i=1}^{n} |s_i|^2$.

In den Berechnungen mit Maple zeigt sich, dass bezüglich der α-Varianten gilt

1. $\approx$ 2., 3. $\approx$ 4,

so dass in weiteren Rechnungen dann nur mit den α-Varianten 1 und 3 gearbeitet wird. Außerdem erweist sich die Wurzelberechnung in Maple als nicht besonders gut implementiert.

In Maple werden sowohl die exakte Rechnung (symbolische Berechnung in der Rationalarithmetik), wo erwartunggemäß keine Unterschiede in den Varianten auftreten, sowie die numerische Rechnung bei 4 Genauigkeiten mit der Gleitpunktarithmetik von 10, 16, 20, 25 Dezimalstellen (`Digits:=...`) ausgeführt.

Das CR wird in Maple durch die folgende Prozedur implementiert.

```
cr1:=proc(n::posint, A::matrix, b::vector, x0::vector,
          maxiter::posint, eeps::numeric,
          varianta::posint,  variantr::posint,
          aus::name, fileaus::name)

  local  k,i,x,xh,p,r,r2,sr2,v,t,s,alpha,alphar,alphas,beta,betar,
         Q,R,fh,fh1,fh2,h;
  global Qv,Rv,epsv,epsv2,rv,Ainv,xs,lp,file1;

  fh2:=‘%.16e‘;          # Ausgabeformate einstellen
  fh1:=‘%+.10e‘;
  fh :=fh1;
  for i from 2 to n do
   fh:=cat(fh,‘ ‘,fh1);
  end do;
```

```
k:=0:
x:=evalf(evalm(x0)):  # x:=evalm(x0):
v:=evalm(A&*x):
Q:=0.5*evalm(transpose(x)&*v)-evalm(transpose(x)&*b):
R:=0.5*evalm(transpose(v)&*v)-evalm(transpose(v)&*b):
r:=evalm(b-v):
t:=evalm(A&*r):
alphar:=evalm(transpose(t)&*r):
p:=evalm(r): lp:=evalm(p):
s:=evalm(t):
if varianta=1 then alphas:=norm(s,frobenius)^2;
 elif varianta=2 then alphas:=sqrt(evalm(transpose(s)&*s))^2;
 elif varianta=3 then alphas:=evalm(transpose(s)&*s);
 elif varianta=4 then i:='i': alphas:=sum(abs(s[i])^2,i=1..n);
 else
end if;

Qv[1]:=Q;
Rv[1]:=R;
h:=evalm(transpose(xs)&*b);  xh:=evalm(xs-x);
epsv[1]:=abs(sqrt(2.0*Q+h));
epsv2[1]:=sqrt(evalm(transpose(xh)&*xh));
r2:=evalm(transpose(r)&*r);
sr2:=evalf(sqrt(r2));
rv[1]:=sr2;
fprintf(default,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sr2);
if fileaus=ja then
 fprintf(file1,`k = %3d,  ||r||_2 = `||fh2||`\n`,k,sr2);
end if;

if aus=ja then
 fprintf(default,`\n`):
 fprintf(default,`Schritt k = %g\n`,k);
 fprintf(default,`Startvektor            x =
                 [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(default,`Funktionswert       Q(x) = `||fh2||`\n`,Q);
 fprintf(default,`Funktionswert       R(x) = `||fh2||`\n`,R);
 fprintf(default,`Residuum/Suchr.  r =b-Ax =
                 [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(default,`Anfangsfehlerquadrat r'r = `||fh2||`\n\n`,r2);
end if;

if fileaus=ja then
 fprintf(file1,`\n`):
 fprintf(file1,`Schritt k = %g\n`,k);
 fprintf(file1,`Startvektor            x =
               [`||fh||`]\n`,seq(x[i],i=1..n));
 fprintf(file1,`Funktionswert       Q(x) = `||fh2||`\n`,Q);
 fprintf(file1,`Funktionswert       R(x) = `||fh2||`\n`,R);
 fprintf(file1,`Residuum/Suchr.  r =b-Ax =
               [`||fh||`]\n`,seq(r[i],i=1..n));
 fprintf(file1,`Anfangsfehlerquadrat r'r = `||fh2||`\n\n`,r2);
end if;
```

```
while (sr2>eeps) and (k<maxiter) do
 if alphar=0 then
  lprint(‘Abbruch wegen Nenner r’Ar=0‘):
  RETURN(x,k);
 end if;

 alpha:=alphar/alphas;
 x:=evalm(x+alpha*p);
 if variantr=1 then r:=evalm(r-alpha*s); else r:=evalm(b-A&*x); end if;
 r2:=evalm(transpose(r)&*r);
 sr2:=evalf(sqrt(r2));
 betar:=1/alphar;
 t:=evalm(A&*r):
 alphar:=evalm(transpose(t)&*r);
 beta:=alphar*betar;
 p:=evalm(r+beta*p);  lp:=evalm(p);
 s:=evalm(t+beta*s);
 k:=k+1;

 if varianta=1 then alphas:=norm(s,frobenius)^2;
  elif varianta=2 then alphas:=sqrt(evalm(transpose(s)&*s))^2;
  elif varianta=3 then alphas:=evalm(transpose(s)&*s);
  elif varianta=4 then i:=’i’: alphas:=sum(abs(s[i])^2,i=1..n);
  else
 end if;

 v:=evalm(A&*x):
 Q:=0.5*evalm(transpose(x)&*v)-evalm(transpose(x)&*b):
 R:=0.5*evalm(transpose(v)&*v)-evalm(transpose(v)&*b):

 if aus=ja then
  fprintf(default,‘\n‘);
  fprintf(default,‘Schritt k = %g\n‘,k);
  fprintf(default,‘Iterationsvektor      x =
                  [‘||fh||‘]\n‘,seq(x[i],i=1..n));
  fprintf(default,‘Funktionswert      Q(x) = ‘||fh2||‘\n‘,Q);
  fprintf(default,‘Funktionswert      R(x) = ‘||fh2||‘\n‘,R);
  fprintf(default,‘Suchrichtung          p =
                  [‘||fh||‘]\n‘,seq(p[i],i=1..n));
  fprintf(default,‘Residuum       r = b-Ax =
                  [‘||fh||‘]\n‘,seq(r[i],i=1..n));
  fprintf(default,‘Fehlernormquadrat   r’r = ‘||fh2||‘\n‘,r2);
 end if;
 if fileaus=ja then
  fprintf(file1,‘\n‘);
  fprintf(file1,‘Schritt k = %g\n‘,k);
  fprintf(file1,‘Iterationsvektor      x =
                [‘||fh||‘]\n‘,seq(x[i],i=1..n));
  fprintf(file1,‘Funktionswert      Q(x) = ‘||fh2||‘\n‘,Q);
  fprintf(file1,‘Funktionswert      R(x) = ‘||fh2||‘\n‘,R);
  fprintf(file1,‘Suchrichtung          p =
                [‘||fh||‘]\n‘,seq(p[i],i=1..n));
  fprintf(file1,‘Residuum       r = b-Ax =
                [‘||fh||‘]\n‘,seq(r[i],i=1..n));
```

```
 fprintf(file1,‘Fehlernormquadrat   r’r = ‘||fh2||‘\n‘,r2);
end if;

fprintf(default,‘k = %3d,  ||r||_2 = ‘||fh2||‘\n‘,k,sr2);
if fileaus=ja then
 fprintf(file1,‘k = %3d,  ||r||_2 = ‘||fh2||‘\n‘,k,sr2);
end if;

Qv[k+1]:=Q;
Rv[k+1]:=R;
xh:=evalm(xs-x);
v:=evalm(A&*xh):
epsv[k+1]:=abs(sqrt(evalm(transpose(xh)&*v)));
           # epsv[k+1]:=abs(sqrt(2.0*Q+h));
epsv2[k+1]:=sqrt(evalm(transpose(xh)&*xh));
rv[k+1]:=sr2;
end do:

lprint(‘||xs-x||_2    = ‘,evalf(norm(xs-x,frobenius))):
lprint(‘||A^(-1)r||_2 = ‘,evalf(norm(Ainv&*r,frobenius))):
lprint(‘ ‘):

[x,k];
end:
```

7.2.1 Beispiel 1

Im Beispiel 1 machen wir die Ausführungen etwas dataillierter, während in den nachfolgenden wir uns auf die grafische Darstellung der Ergebnisse beschränken.

```
# Beispiel 1, A=A'>0

 n:=8:
 A:=matrix(n,n,
    [[ 168,   24,  338,  27,  27,   53,  -7,  80],
     [  24,  178,  169,  72,  53, -103,  17,  80]
     [ 338,  169, 1177, 192, -62, -108, -48, 180],
     [  27,   72,  192, 125,   2,  -24,  36, 180],
     [  27,   53,  -62,   2, 222,   70,  46, 100],
     [  53, -103, -108, -24,  70,  178,  34, 100],
     [  -7,   17,  -48,  36,  46,   34,  34, 100],
     [  80,   80,  180, 180, 100,  100, 100, 400]]):

 b:=vector(n,[ 229, 129, 790, -214, 106, -276, -216, -600]):

# Loesung
 xs:=vector(n,[1,-1,1,-1,2,-2,2,-2]):
```

EW von $\lambda(A) = \{0.006\,305, 1.359\,344, 5.859\,472, 44.417\,725, 199.730\,179, 282.983\,844, 548.279\,552, 1\,399.363\,575\}$,

Kondition mittels Spektralnorm $\kappa(A) = 221\,911.791....$

Für das spezielle LGS berechnen wir in den verschiedenen Varianten die absoluten Fehler

$$\|r^{(k)}\|_2, \quad \|e^{(k)}\|_A, \quad \|e^{(k)}\|_2, \quad k = 0, 1, 2, ...,$$

die man als Ergebnisvektoren `rv[1..maxiter+1]`, `epsv[1..maxiter+1]` bzw. `epsv2[1..maxiter+1]` der obigen Prozedur `cr1` erhält. Die Werte $\|r^{(k)}\|_2$ bilden eine monoton abnehmende Folge, was auch für $\|e^{(k)}\|_2$ gilt, aber für $\|e^{(k)}\|_A$ i. Allg. und speziell bei numerischen Rechnungen über n Iterationen hinaus, nicht zutreffen muss.

Dann folgen grafische Vergleiche der relativen Fehler des CR, die aus Gründen der besseren Übersicht als Funktionen $\log_{10}()$ in Abhängigkeit von der Iterationszahl k dargestellt werden, also

$$f(k) \;=\; \log_{10}\left(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}\right),\ \log_{10}\left(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}\right),\ k = 0, 1, 2, \ldots\ . \tag{7.2}$$

Zunächst schauen wir auf eine exakte Rechnung in der Rationalarithmetik von Maple, die bei $x^{(0)} = (1, 0, ..., 0)^T$ und der Dimension $n = 8$ des Problems ohne Weiteres machbar ist und wo spätestens $\|r^{(n)}\|_2 = 0$ sein muss.

```
CR : Iterationsverlauf bei exakter Rechnung
i         Q(x[i])=         ||e[i]||_A=   ||e[i]||_2=          R(x[i])=              ||r[i]||_2=
    (||e[i]||^2_A-xs'b)/2  ||xs-x[i]||_A ||xs-x[i]||_2  (||r[i]||^2_2-b'b)/2        ||b-Ax[i]||_2
-------------------------------------------------------------------------------------------------------
0 -1.4500000000000000e+02  4.843552e+01  4.358899e+00  -1.6767600000000000e+05  9.4746714982631460e+02
1 -9.3655760976449890e+02  2.762037e+01  3.759722e+00  -5.1012335956705510e+05  4.6130172432572860e+02
2 -1.1290668013410820e+03  1.943879e+01  3.504881e+00  -5.8633776951151860e+05  2.4570401090939350e+02
3 -1.2809297380043410e+03  8.610489e+00  3.161297e+00  -6.1490840702123780e+05  5.6825926807435110e+01
4 -1.3133668731345050e+03  3.044052e+00  2.902845e+00  -6.1615942648028840e+05  2.6965664082737400e+01
5 -1.3168733890526200e+03  1.501074e+00  2.873782e+00  -6.1652084934731680e+05  2.0739588630652940e+00
6 -1.3174862152795890e+03  1.013691e+00  2.736956e+00  -6.1652228848201940e+05  1.1929107094578960e+00
7 -1.3179788185218540e+03  2.058226e-01  2.591899e+00  -6.1652299986643070e+05  1.6344371444408400e-02
8 -1.3180000000000000e+03  0.000000e+00  0.000000e+00  -6.1652300000000000e+05  0.0000000000000000e+00
```

Es gibt im Iterationsverlauf des CR keine gravierenden wertemäßigen Unterschiede zum CG, was in der Gegenüberstellung der Ergebnistableaus deutlich wird.

```
CG : Iterationsverlauf bei exakter Rechnung
 i         Q(x[i])=          ||e[i]||_A=||xs-x[i]||_A   ||e[i]||_2=            ||r[i]||_2=
     (||e[i]||^2_A-xs'b)/2     =sqrt(2Q(x[i])+xs'b)     ||xs-x[i]||_2          ||b-Ax[i]||_2
 ------------------------------------------------------------------------------------------------------
 0  -1.4500000000000000e+02  4.8435524153249340e+01  4.3588989435406740e+00  9.4746714982631460e+02
 1  -9.8368605228089200e+02  2.5857840115489440e+01  3.6560995780917200e+00  5.2812573464063930e+02
 2  -1.1459168350799170e+03  1.8551720401088610e+01  3.4384336294029440e+00  2.9031119293215690e+02
 3  -1.2813654838675770e+03  8.5597331888818880e+00  3.1543128197346300e+00  5.8409543263193730e+01
 4  -1.3150994801622860e+03  2.4085347569483770e+00  2.8805665078096420e+00  3.0634487373454350e+01
 5  -1.3168735117533320e+03  1.5009918365320540e+00  2.8737370089577910e+00  2.0801202738379160e+00
 6  -1.3175615355950960e+03  9.3644477135973120e-01  2.6938938285905780e+00  1.4582848191466750e+00
 7  -1.3179788185392140e+03  2.0582254874512040e-01  2.5918966487041220e+00  1.6345905778557270e-02
 8  -1.3180000000000000e+03  0.0000000000000000e+00  0.0000000000000000e+00  0.0000000000000000e+00
```

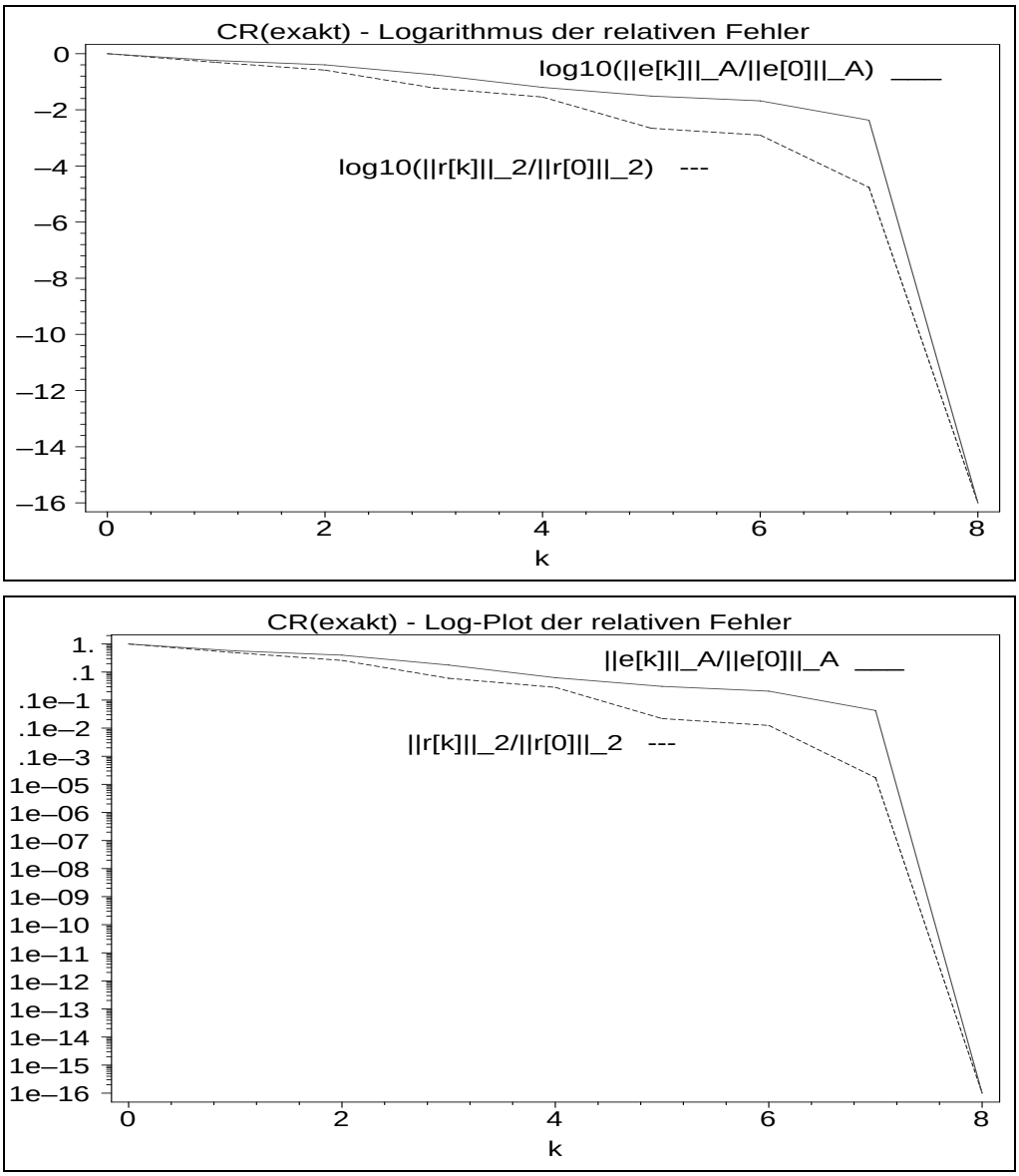


Abb. 7.34 Dateien *cr*1_01.*ps*, *cr*1_02.*ps*, Matrix $A(8,8)$ zu Beispiel 1, CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik), Verlauf der relativen Fehler $\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}, \frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}$, $k = 0(1)8$, als Logarithmus bzw. im Log-Plot

Unser Hauptaugenmerk liegt wiederum auf der numerischen Bestimmung des Residuums $r(x)$ in seinen beiden schon genannten Varianten `recursive` und `realy`.

Bezüglich der **Fehlererinnerung** und Iterationsverlauf treffen im CR die gleichen Aussagen zu wie beim CG im Abschnitt 9.1.1. Sie ist typisch für den Fehler $\|r^{(k)}\|_2$. Es zeigt sich, das die wirkliche Berechnung des Residuums ab einer bestimmten Iterationsanzahl zu keiner Verkleinerung dieses Wertes mehr führt. Bei der rekursiven Berechnung des Residuums tritt diese Stagnation gar nicht ein.

Bei der Berechnung von $e(x)$ "steht" meistens auch der Fehler bei geringer Genauigkeit in der Variante `realy`, für höhere Genauigkeiten weichen die beiden r-Varianten nicht wesentlich voneinander ab und stagnieren in der Nähe der vorgegebenen Gleitpunktarithmetik.

In der Grafik zum CR sind bei Gegenüberstellung der zwei r-Varianten die Fehler gekennzeichnet mit den Linienarten

$\log_{10}(\|r^{(k)}\|_2/\|r^{(0)}\|_2)$	solid line, bold	`recursive`
$\log_{10}(\|r^{(k)}\|_2/\|r^{(0)}\|_2)$	dash line	`realy`

genauso für die den Fehler $\log_{10}(\|e^{(k)}\|_A/\|e^{(0)}\|_A)$, um einen Vergleich zum Abstiegsverhalten der relevanten Größe $\|e^{(k)}\|_A$ des CG zu haben.

Die 4 Genauigkeiten der numerischen Rechnung mit der Gleitpunktarithmetik bei 10, 16, 20, 25 Dezimalstellen werden in Maple mittels `Digits:=...` realisiert.
Wir testen dafür die Varianten 1 ($\approx$ 2) und 3 ($\approx$ 4) von den vier α-Varianten bei Gegenüberstellung der zwei r-Varianten.
Die erste Abbildung zeigen wir etwas vergrößert, werden jedoch dann jeweils eine komplette α-Variante zusammen darstellen.

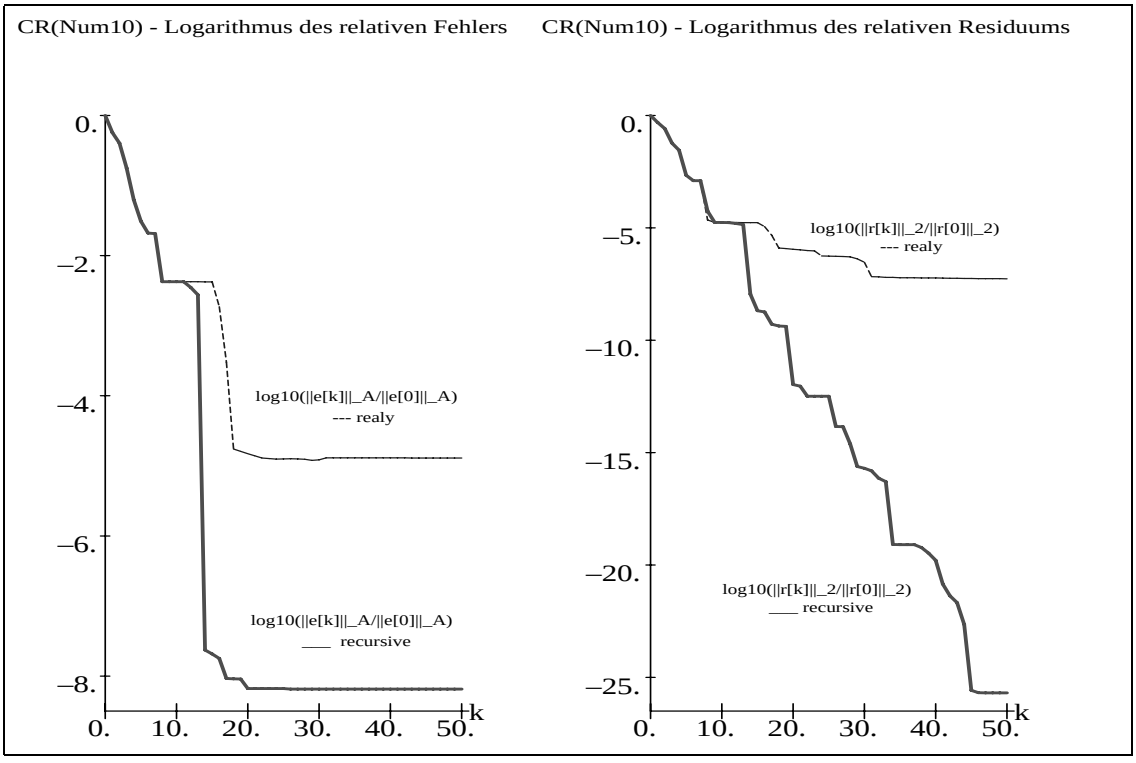


Abb. 7.35 Datei *cr1va110.ps*, Matrix $A(8,8)$ zu Beispiel 1,
CR: numerische Rechnung, Digits=10, α-Variante 1
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

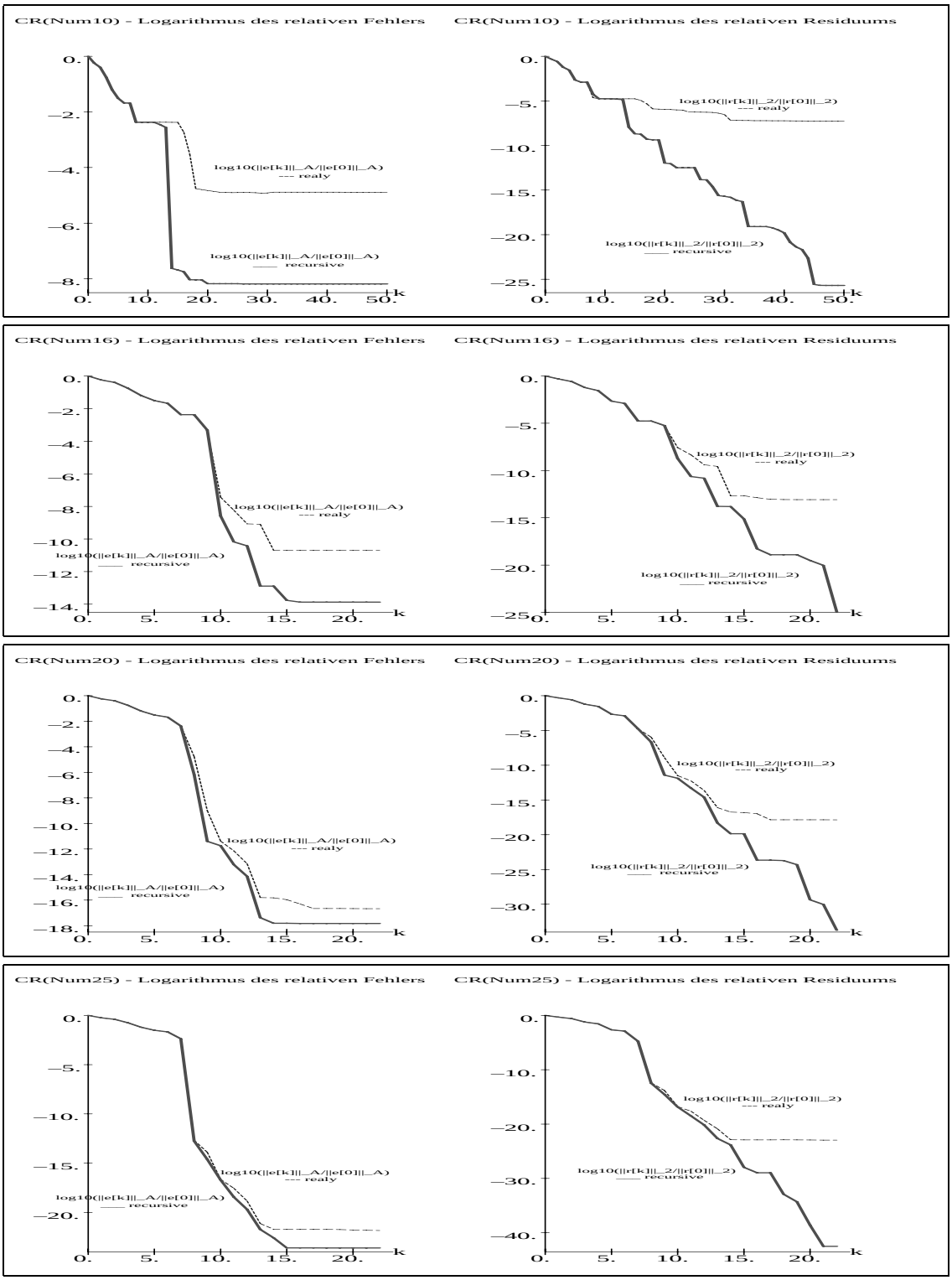


Abb. 7.36 Dateien $cr1va110.ps, cr1va116.ps, cr1va120.ps, cr1va125.ps$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1 $\approx$ V. 2,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

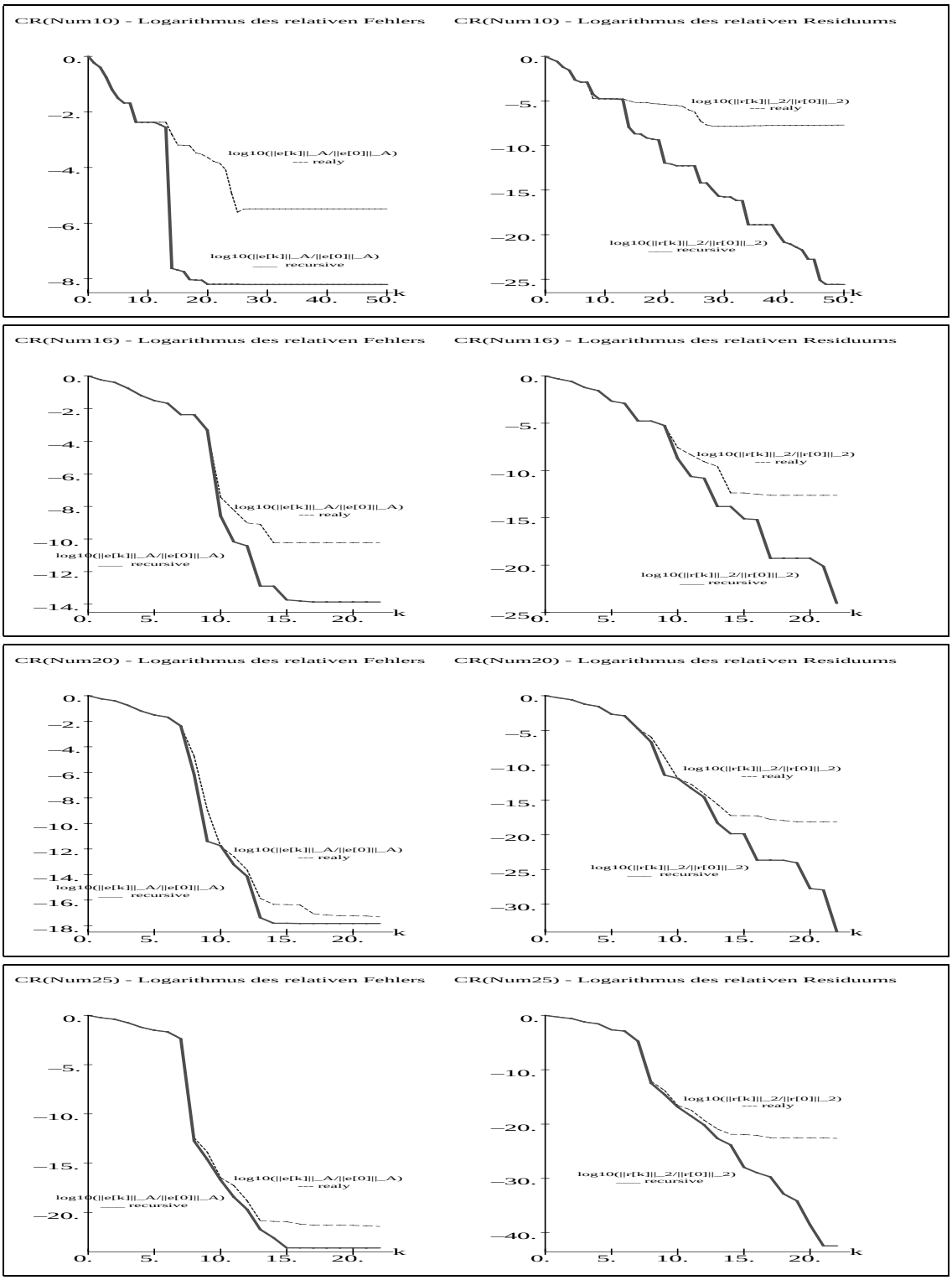


Abb. 7.37 Dateien $cr1va310.ps, cr1va316.ps, cr1va320.ps, cr1va325.ps$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3 $\approx$ V. 4,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.2.2 Beispiel 2

```
# Beispiel 2, A=A'>0

 n:=6:
 A:=matrix(n,n,
   [[ 71,  -1,  23, -36,  70,   60],
    [ -1, 161, 102,  33,  40, -120],
    [ 23, 102, 135,  18, 110,  -60],
    [-36,  33,  18,  35, -22,  -60],
    [ 70,  40, 110, -22, 148,   24],
    [ 60,-120, -60, -60,  24,  144]]):

 b:=vector(n,[187, 194, 360, -45, 432, 0 ]):

# Loesung
 xs:=vector(n,[1,-1,1,2,-2,2]):
```

EW von $\lambda(A) = \{0.182\,596, 4.793\,186, 6.700\,107, 46.718\,766, 263.902\,849, 371.702\,493\}$, Kondition mittels Spektralnorm $\kappa(A) = 2\,035.647....$

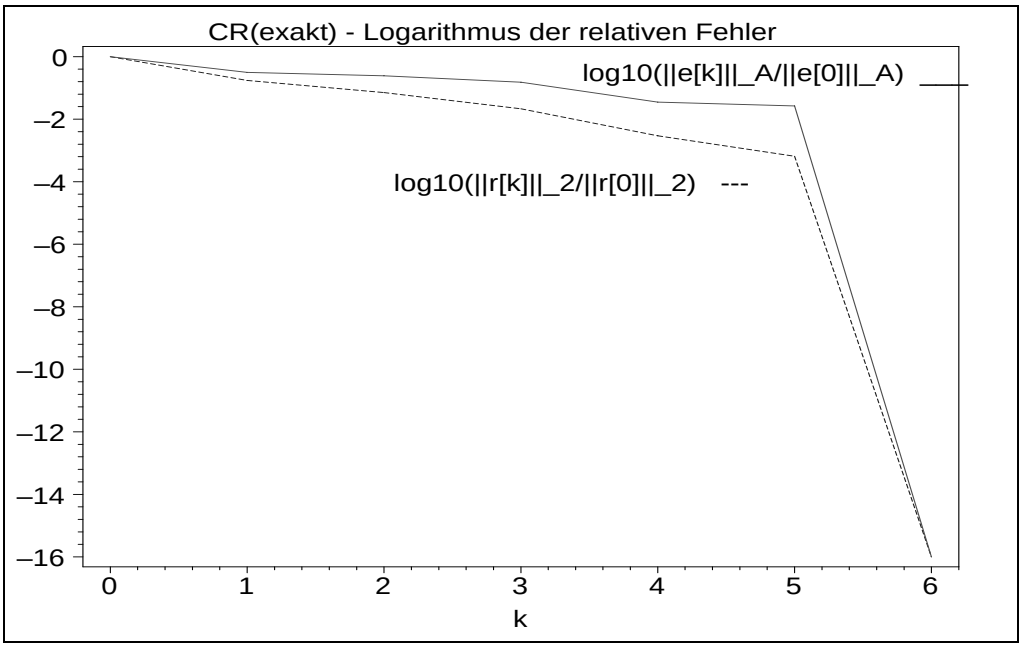


Abb. 7.38 Datei *cr2_01.ps*, Matrix $A(6,6)$ zu Beispiel 2,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0(1)6$

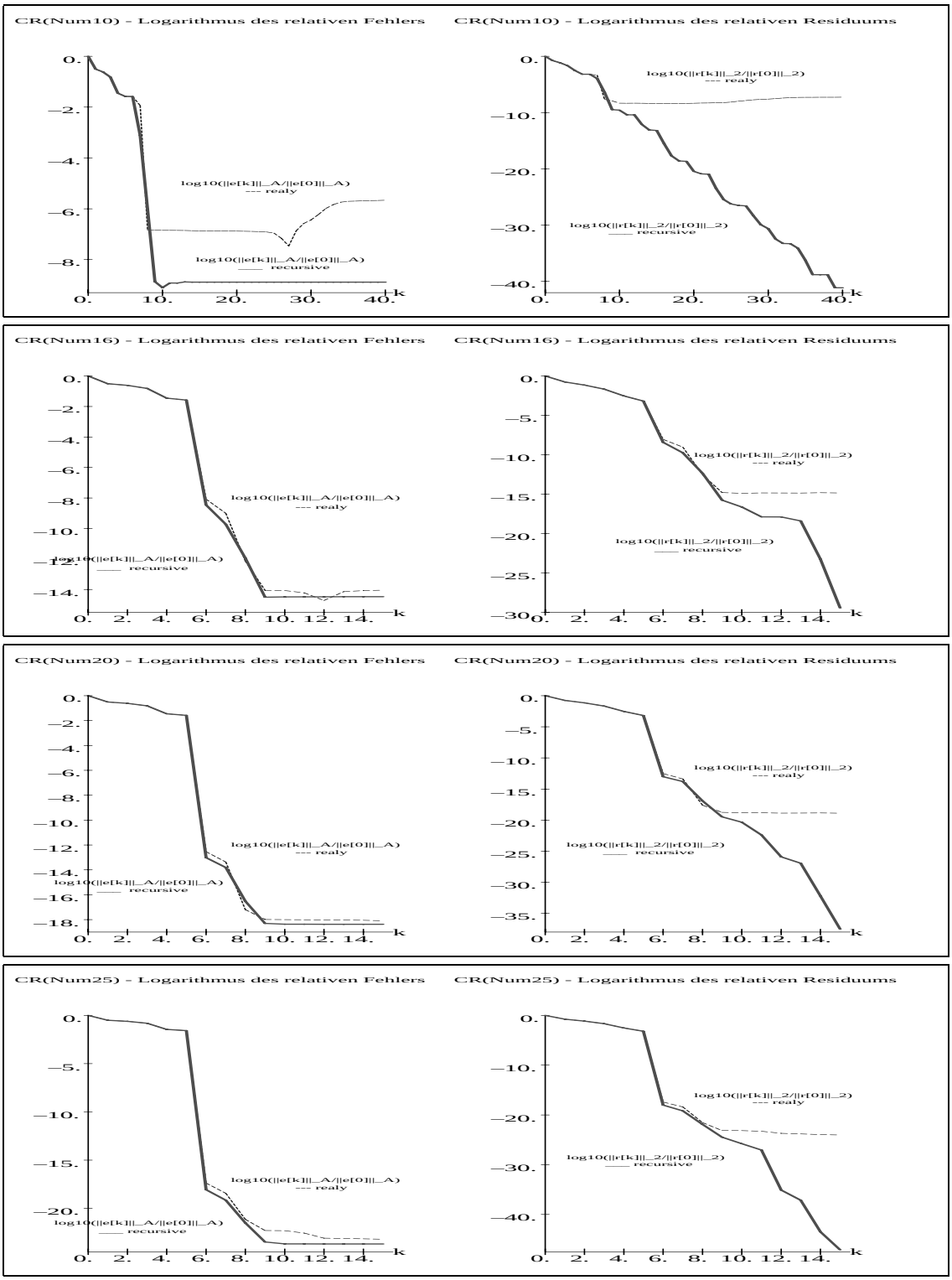


Abb. 7.39 Dateien $cr2va110.ps, cr2va116.ps, cr2va120.ps, cr2va125.ps$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

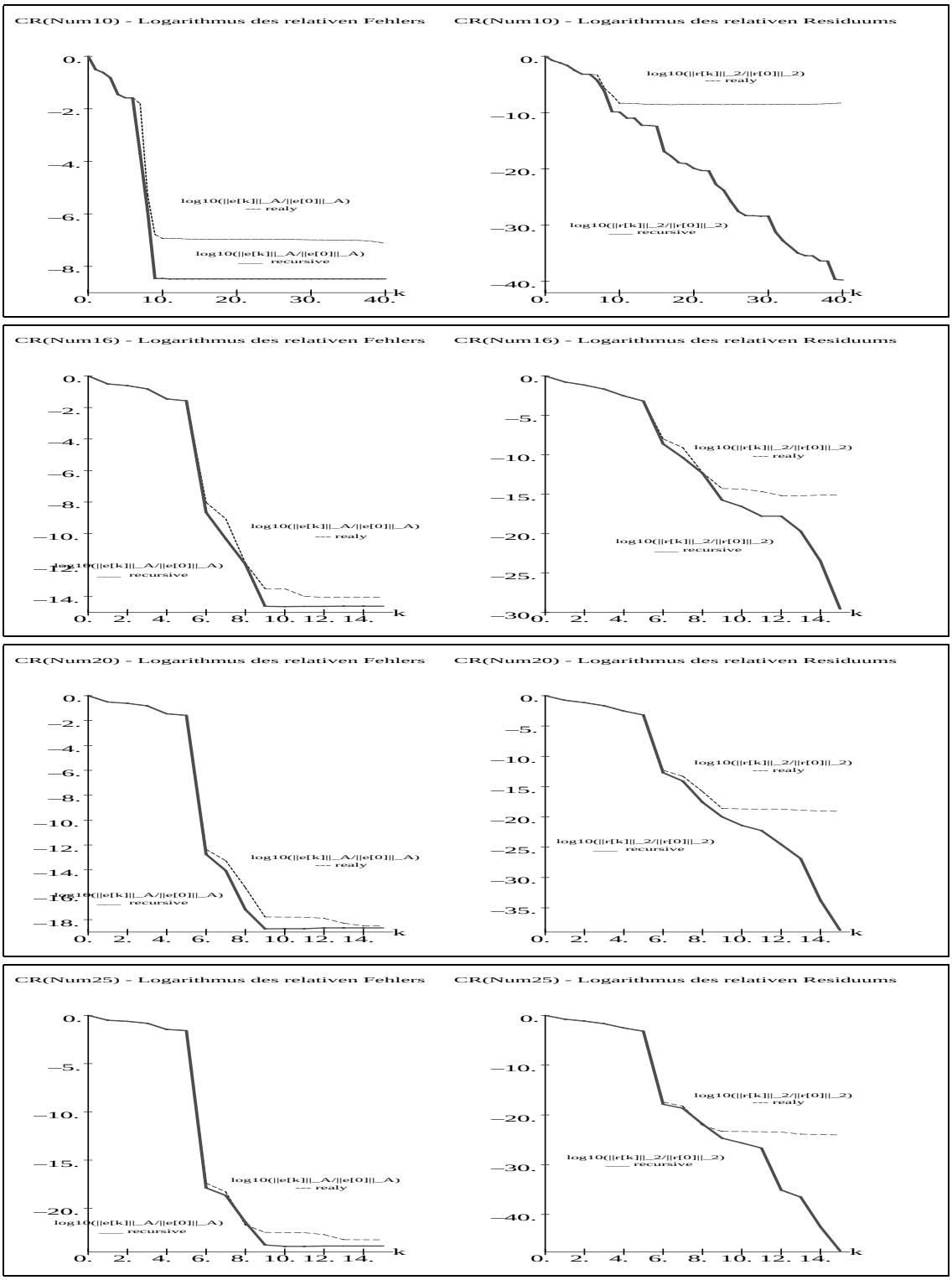


Abb. 7.40 Dateien $cr2va310.ps, cr2va316.ps, cr2va320.ps, cr2va325.ps$, $A(6,6)$, CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.2.3 Beispiel 3

```
# Beispiel 3  A=A' diagonal und indefinit # Van der Vorst Matrix
 A = diag(-9-c,-7-c,-5-c,...,187-c,189-c)
 n:=100:
# Bsp. 3.1
 c:=0:
 A0:=evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
 b0:=vector(n,[ seq(-11+2*i-c,i=1..n)]):
# Loesung
 xs0:=vector(n,[1$n]):  # 1,1,1,....
# Bsp. 3.2
 c:=97/100:
 As:=evalm(diag(seq(-11+2*i,i=1..n))-c*diag(1$n)):
 bs:=vector(n,[ seq(-11+2*i-c,i=1..n)]):
# Loesung
 xss:=vector(n,[1$n]):  # 1,1,1,....
```

Die EW einer Diagonalmatrix sind die Diagonalelemente selber.
Kondition mittels Spektralnorm $\kappa(A0) = 189,\ \kappa(As) = 6\,267.666....$

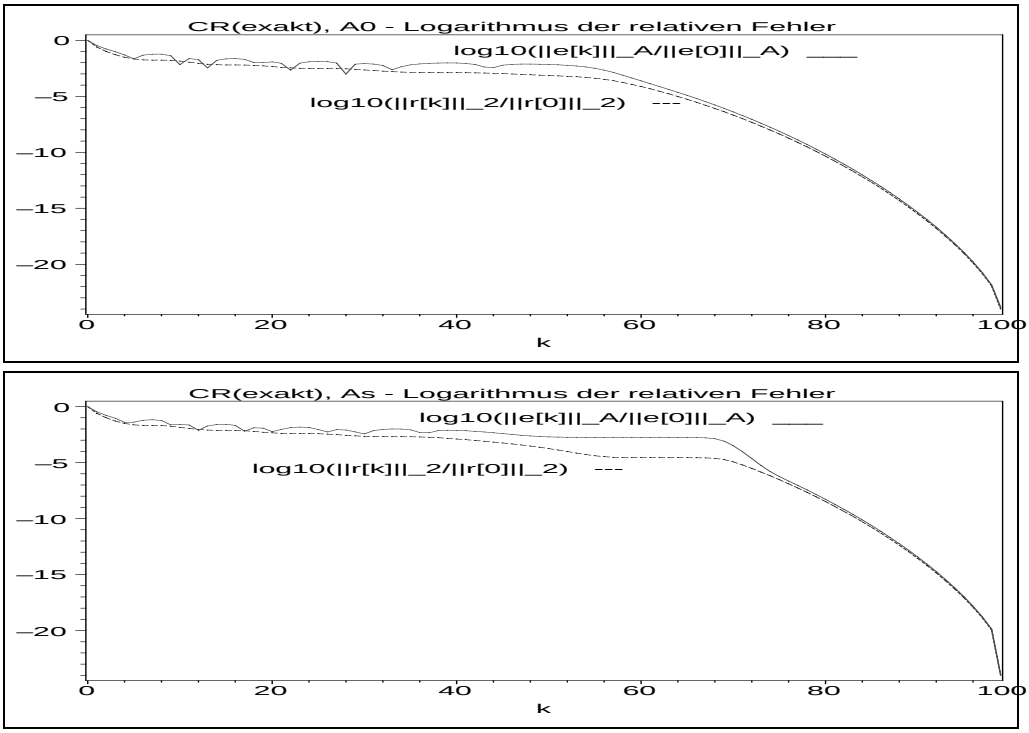


Abb. 7.41 Datei *cr*3*a*0_01.*ps*, *cr*3*as*_01, Matrix $A_{0,s}(100, 100)$ zu Beispiel 3, CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik), Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...100$

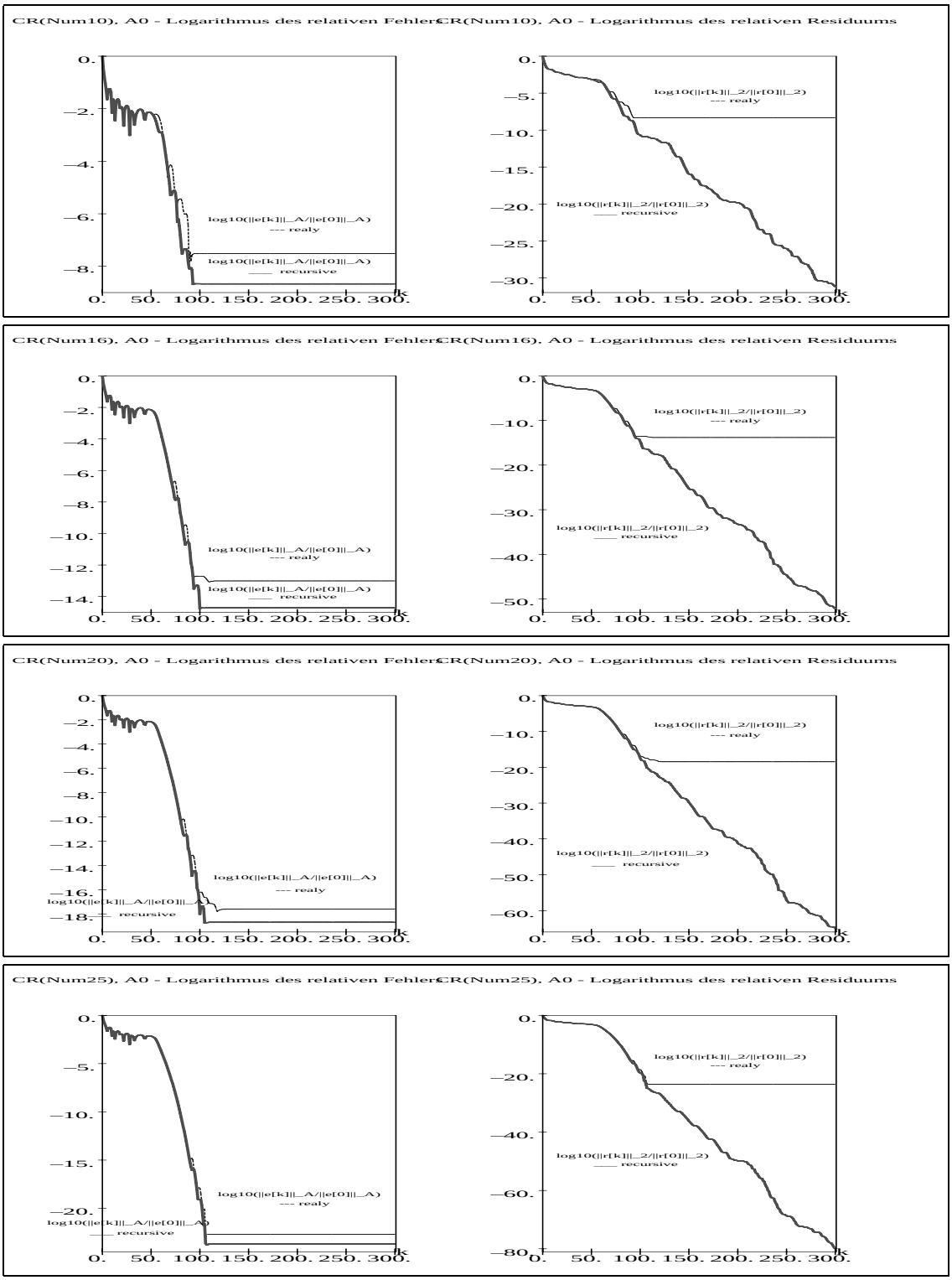


Abb. 7.42 Dateien $cr3a0110.ps, cr3a0116.ps, cr3a0120.ps, cr3a0125.ps$, A_0, CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

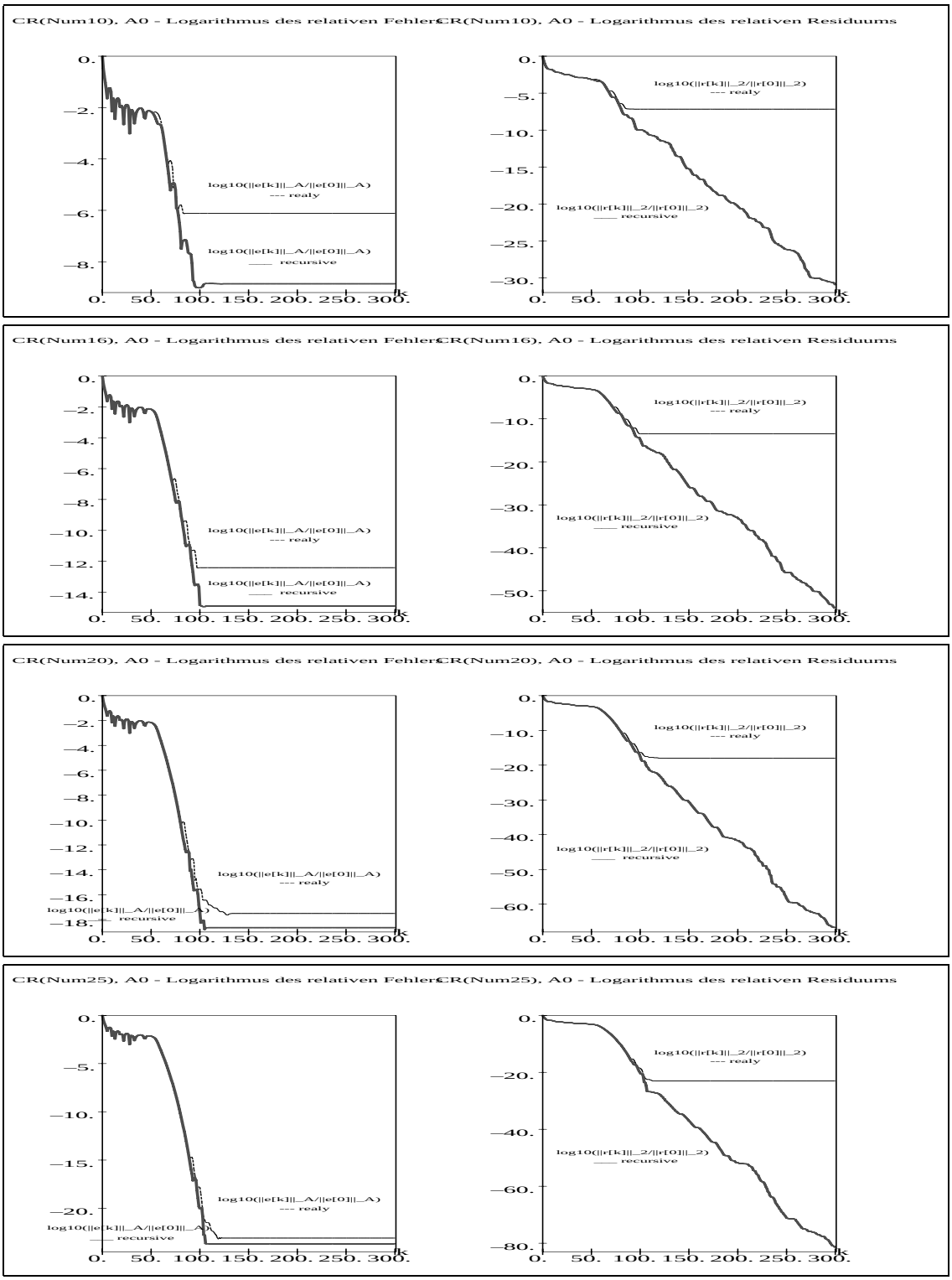


Abb. 7.43 Dateien $cr3a0310.ps, cr3a0316.ps, cr3a0320.ps, cr3a0325.ps$, A_0, CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

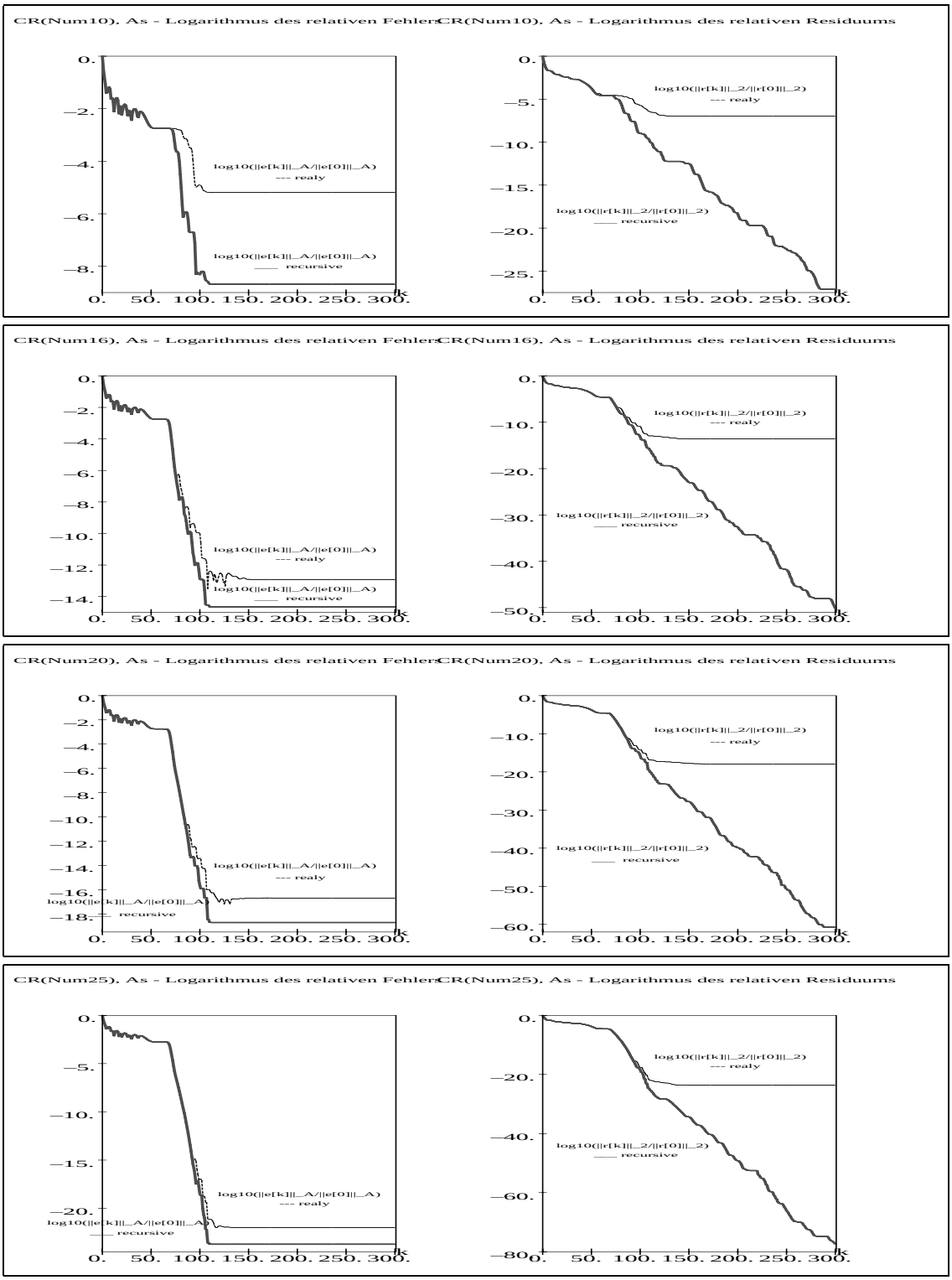


Abb. 7.44 Dateien $cr3as110.ps, cr3as116.ps, cr3as120.ps, cr3as125.ps$, A_s, CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

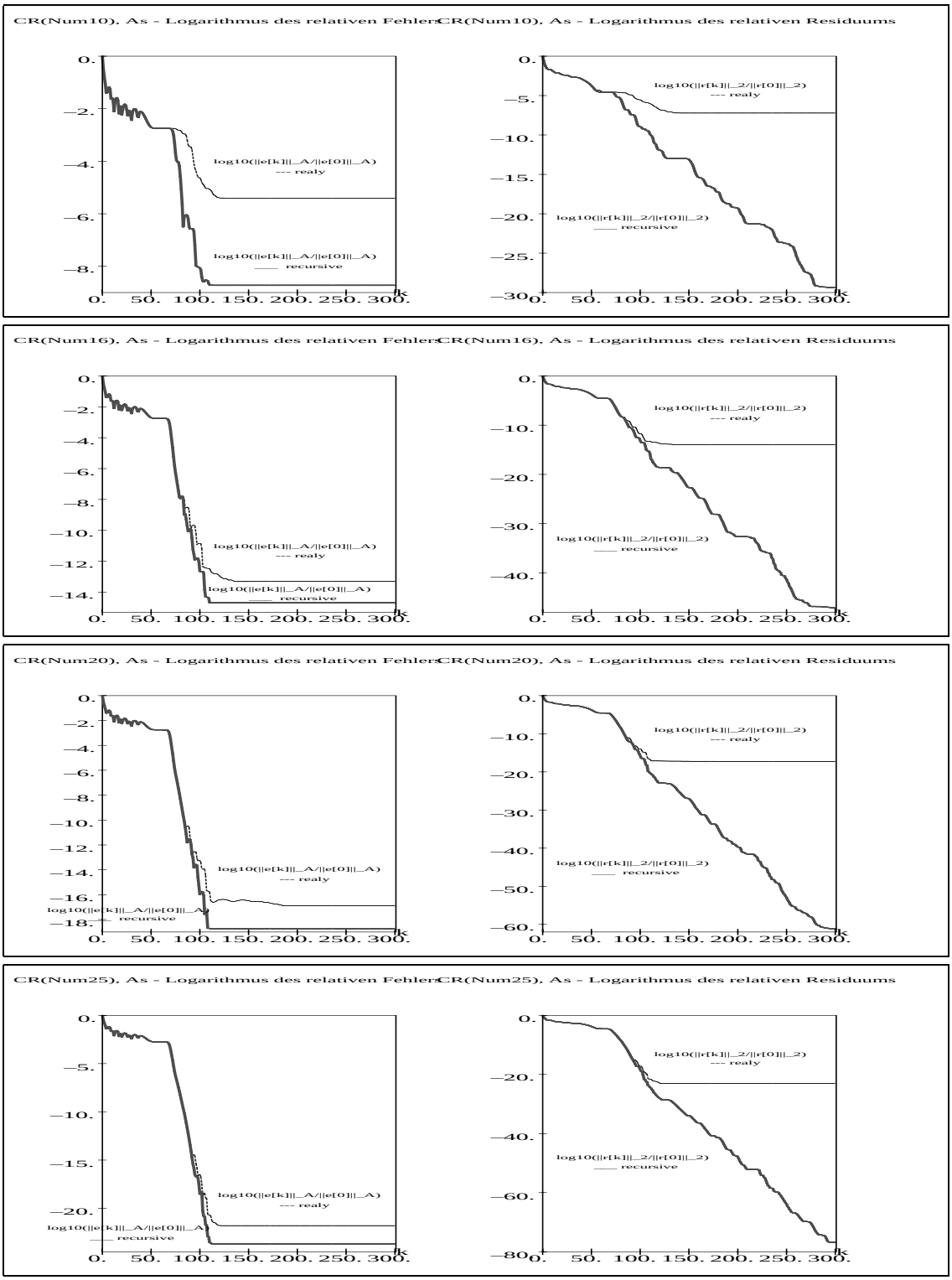
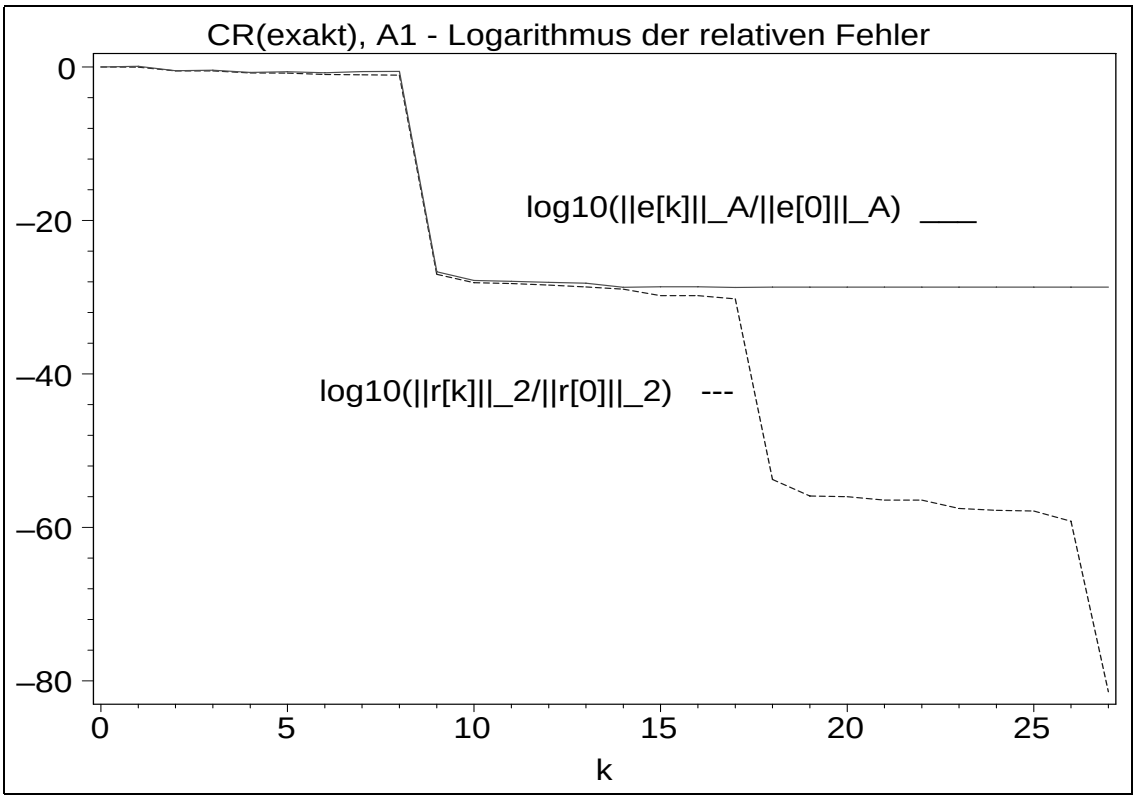
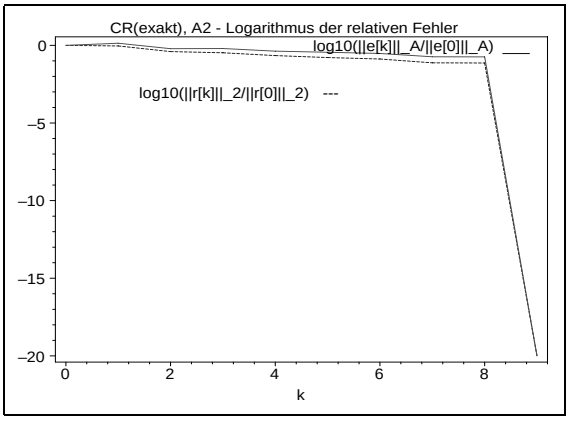


Abb. 7.45 Dateien $cr3as310.ps, cr3as316.ps, cr3as320.ps, cr3as325.ps$, A_s, CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3, Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$ in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.2.4 Beispiel 4

```
# Beispiel 4, A=A'  diagonal, A>0 oder indefinit
# 3 Diagonalmatrizen mit speziellen Spektren (Verteilung der
#   Diagonalelemente) unter Verwendung eines Shift c

 n := 10:   # 20, 50, 100, 200  # gerade
 c := 0:    # 1+1E-6            # Shift

# Bsp. 4.1,  A1 mit  EW im Intervall [-1+c,1+c],
# dichter am Rand, weniger dicht in der Mitte
# sehr langsame symbolische Rechnung ersetzen durch hochgenaue
# Float-Rechnung
 A1 := evalm(diag(seq(evalf(sin(Pi/2*(-1+2*(i-1)/(n-1)))),i=1..n))
             +c*diag(1$n)):
 b1 := vector(n,[seq(evalf(sin(Pi/2*(-1+2*(i-1)/(n-1))))+c,i=1..n)]):
# Loesung
 xs1:=vector(n,[1$n]):  # 1,1,1,....
-----------------------------------------------------------------------

# Bsp. 4.2,   A2 mit gleichverteilten EW im Intervall [-1+c,1+c],
 m := n/2:
 A2 := evalm(diag(seq(-1+2*(i-1)/(n-1),i=1..n))+c*diag(1$n)):
 b2 := vector(n,[ seq(-1+2*(i-1)/(n-1)+c,i=1..n)]):
# Loesung
 xs2:=vector(n,[1$n]):  # 1,1,1,....
-----------------------------------------------------------------------

# Bsp. 4.3,  A3 mit  EW im Intervall [-1+c,1+c],
# verteilt in 2 schmalen Streifen nahe den Raendern
 d := 1/10:   # Streifenbreite des Teilspektrums
 m := n/2:
 A3 := diag(0$n):
 for i from 1 to m do
   h := -1+d*(i-1)/(m-1);
   A3[i,i] := h+c;           # Spektrum in 2 kleinen randnahen
   A3[n+1-i,n+1-i] := -h+c;  # Teilintervallen/Streifen
 end do:
 A3 := evalm(A3):
 b3 := vector(n,[seq(A3[i,i],i=1..n)]):
# Loesung
 xs3:=vector(n,[1$n]):  # 1,1,1,....
```





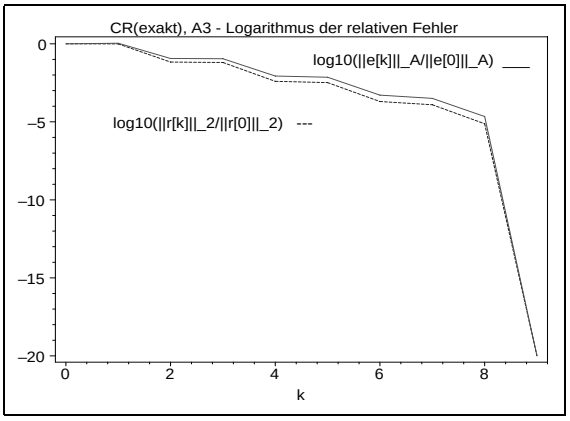
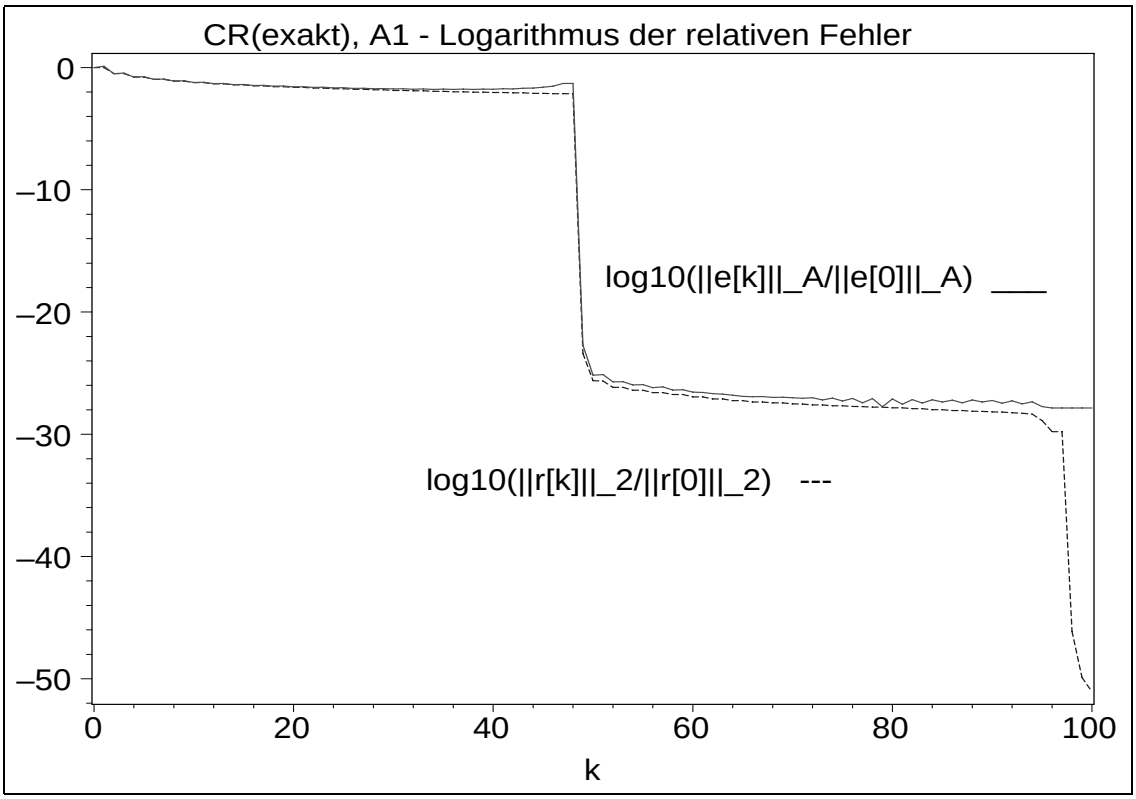
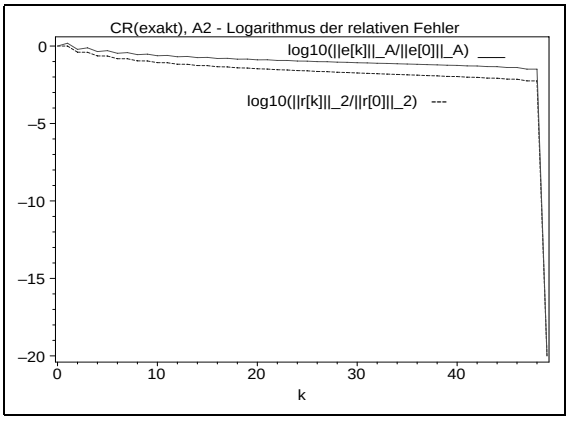


Abb. 7.46
Dateien
$cr4a110.ps,\ cr4a210.ps,\ cr4a310.ps,$
Matrizen $A_{1,2,3}(10,10),\ c=0,$
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k=0,1,...$





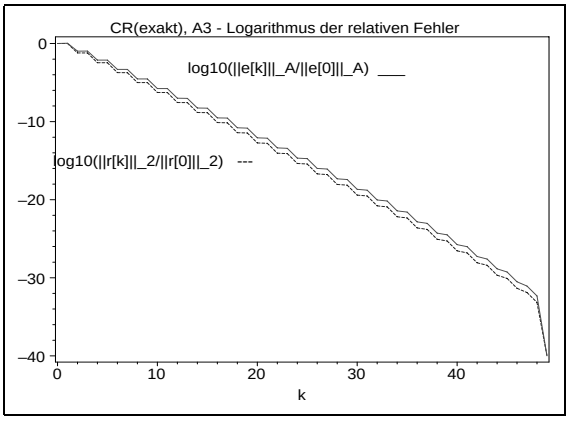
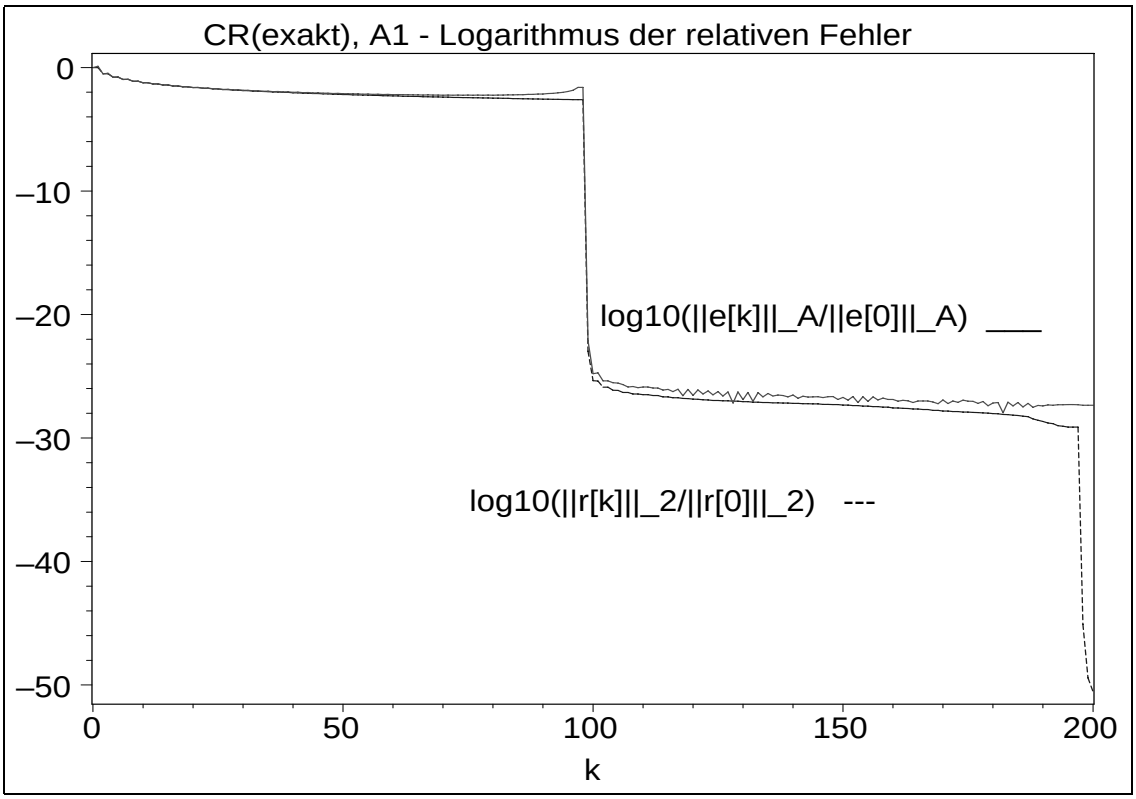
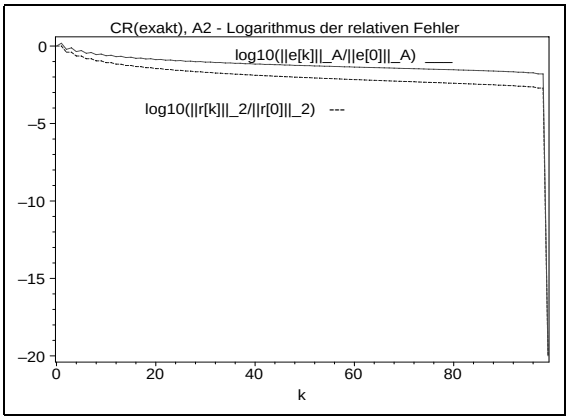


Abb. 7.47
Dateien
$cr4a150.ps,\ cr4a250.ps,\ cr4a350.ps,$
Matrizen $A_{1,2,3}(50,50),\ c=0,$
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}),\ \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k=0,1,...$





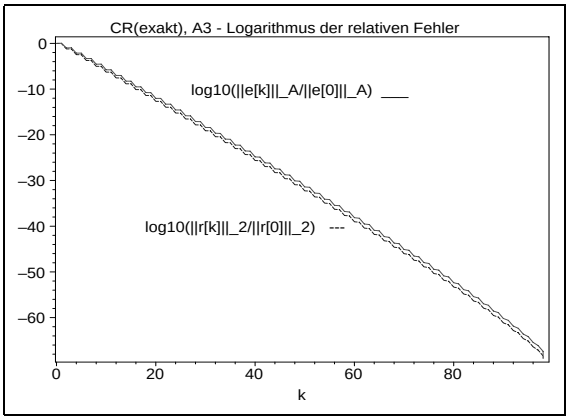
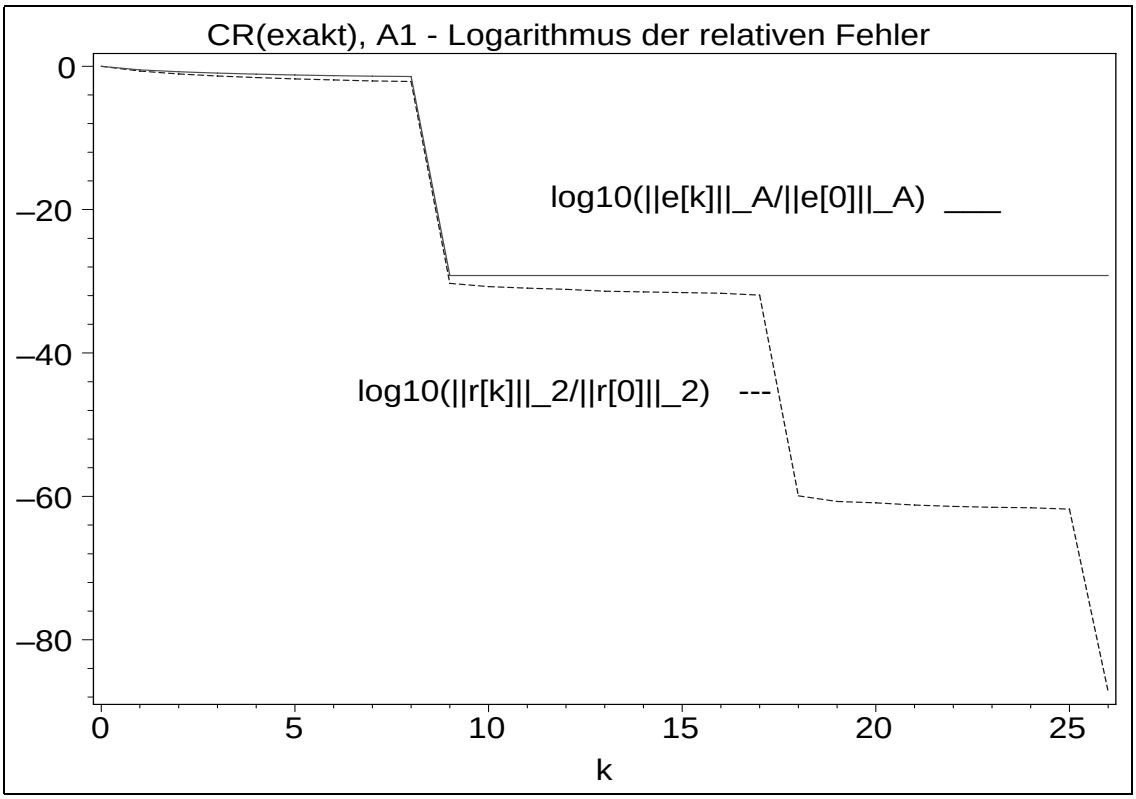
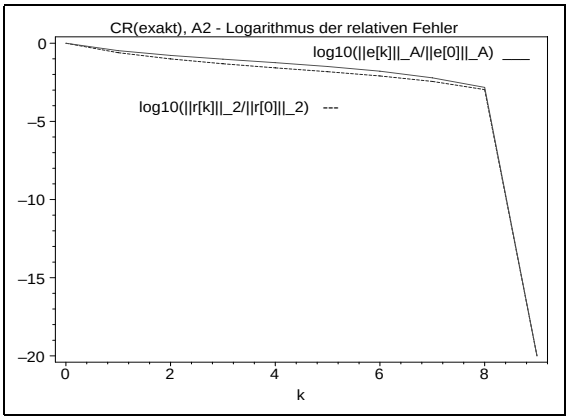


Abb. 7.48
Dateien
$cr4a11h.ps$, $cr4a21h.ps$, $cr4a31h.ps$,
Matrizen $A_{1,2,3}(100, 100)$, $c = 0$,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





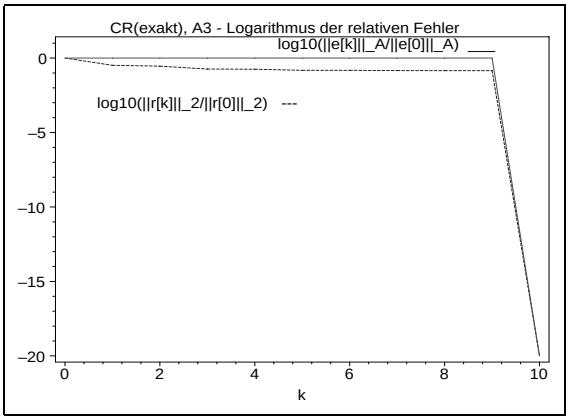
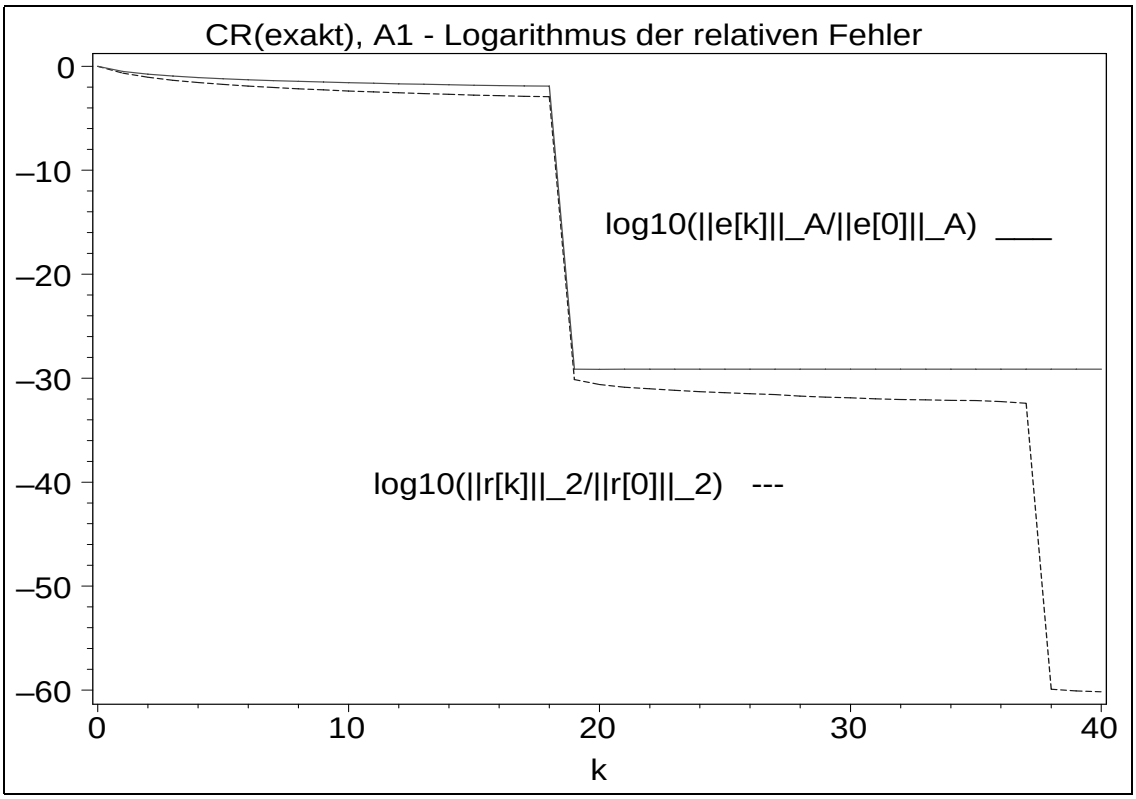
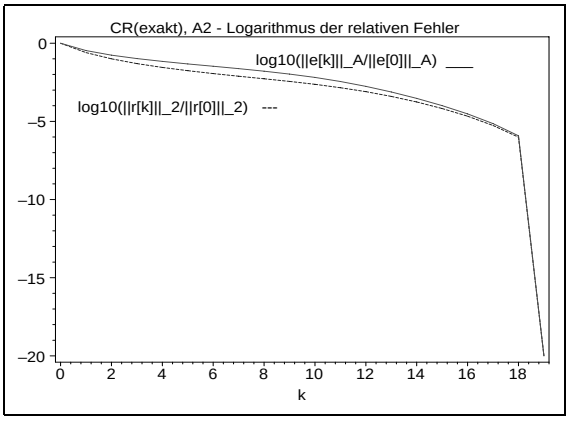


Abb. 7.49
Dateien
$cr4a110c.ps$, $cr4a210c.ps$, $cr4a310c.ps$,
Matrizen $A_{1,2,3}(10,10)$, $c = 1 + 10^{-6}$,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





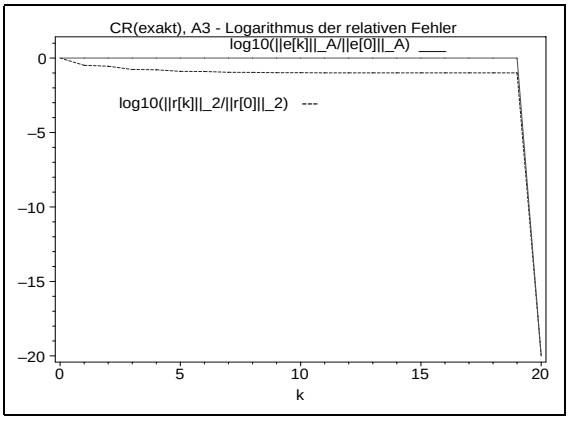
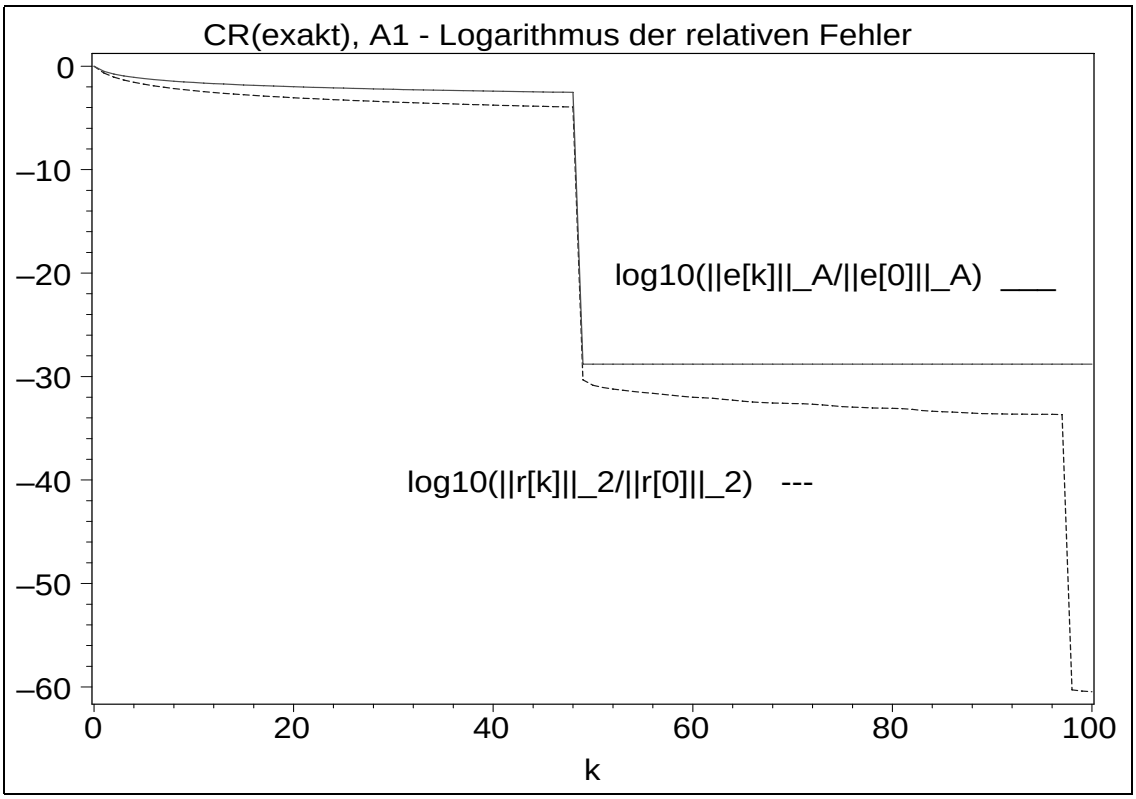
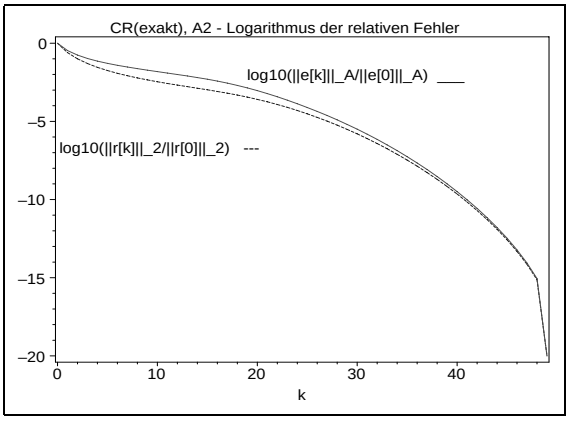


Abb. 7.50
Dateien
$cr4a120c.ps$, $cr4a220c.ps$, $cr4a320c.ps$,
Matrizen $A_{1,2,3}(20,20),\ c = 1 + 10^{-6}$,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2}),\ k = 0, 1, ...$





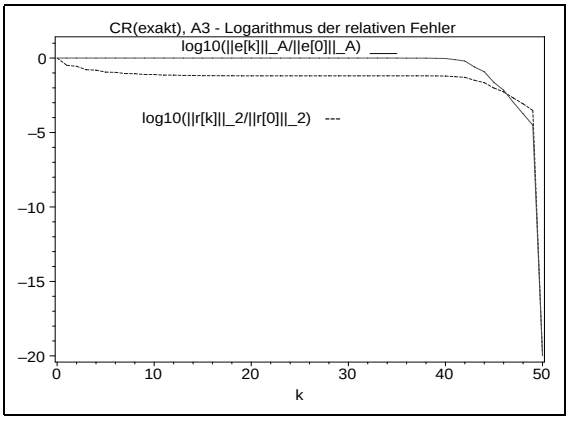
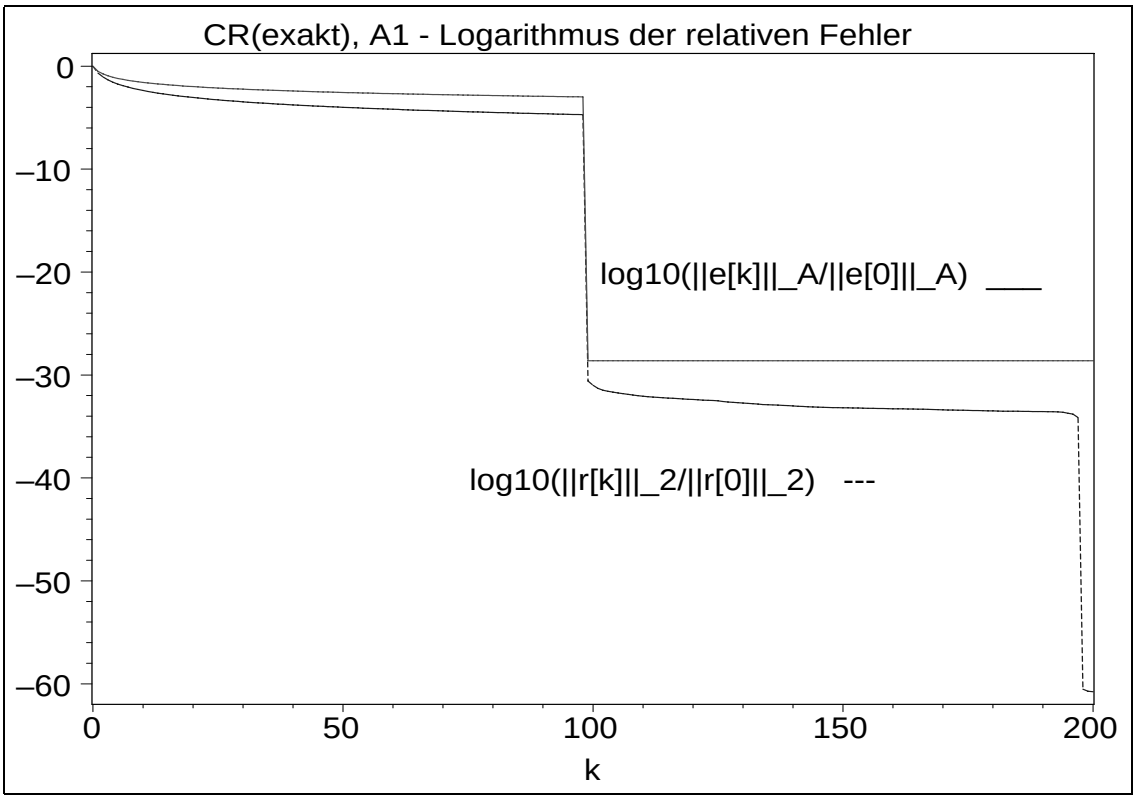
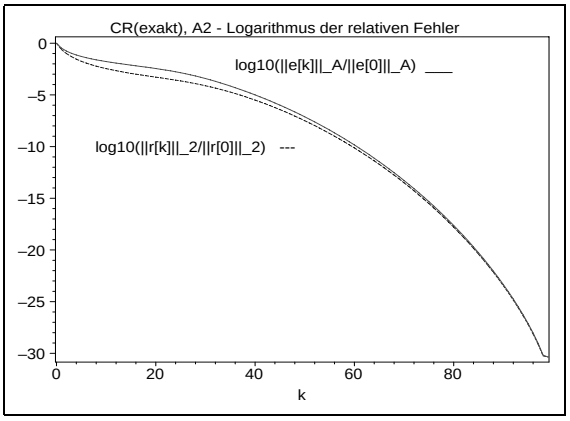


Abb. 7.51
Dateien
$cr4a150c.ps$, $cr4a250c.ps$, $cr4a350c.ps$,
Matrizen $A_{1,2,3}(50,50)$, $c = 1 + 10^{-6}$,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$





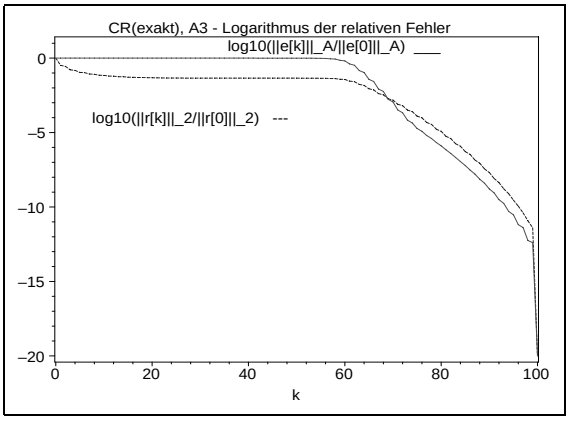


Abb. 7.52
Dateien
$cr4a11hc.ps$, $cr4a21hc.ps$, $cr4a31hc.ps$,
Matrizen $A_{1,2,3}(100, 100)$, $c = 1 + 10^{-6}$,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler
$\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A})$, $\log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$

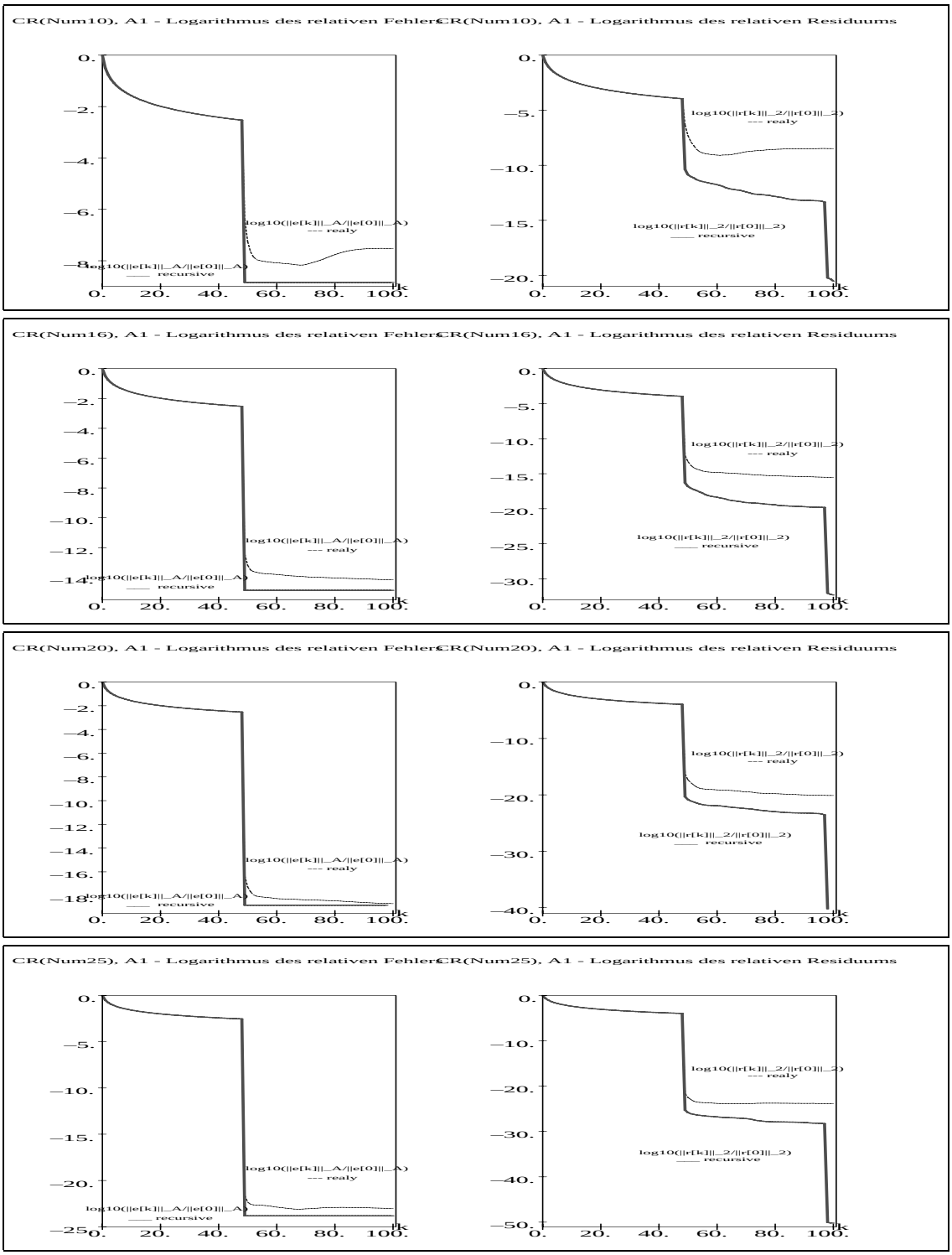


Abb. 7.53 Dateien $cr4a1110.ps, cr4a1116.ps, cr4a1120.ps, cr4a1125.ps$,
$A_1(50,50)$, $c = 1 + 10^{-6}$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

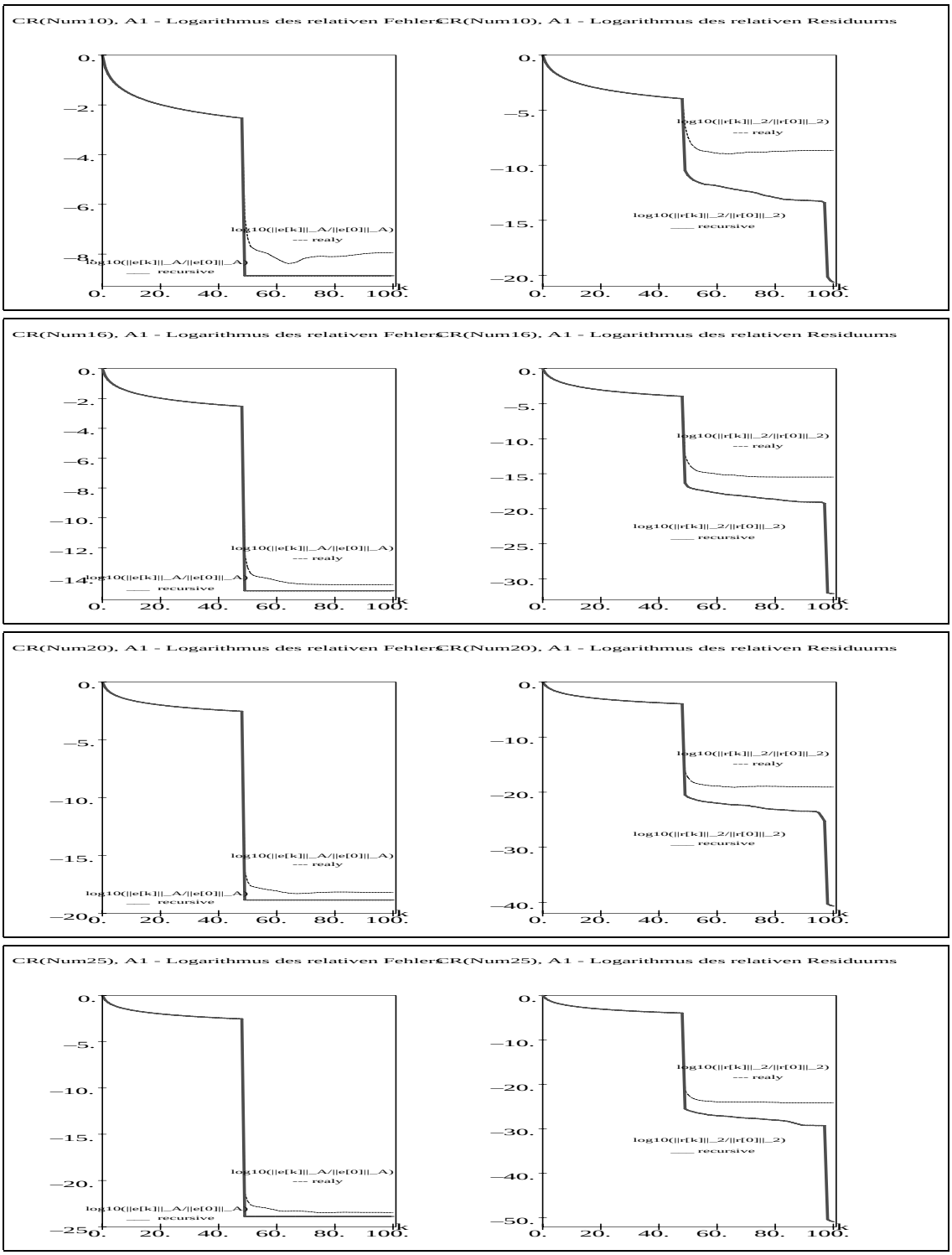


Abb. 7.54 Dateien $cr4a1310.ps, cr4a1316.ps, cr4a1320.ps, cr4a1325.ps$,
$A_1(50,50)$, $c = 1 + 10^{-6}$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

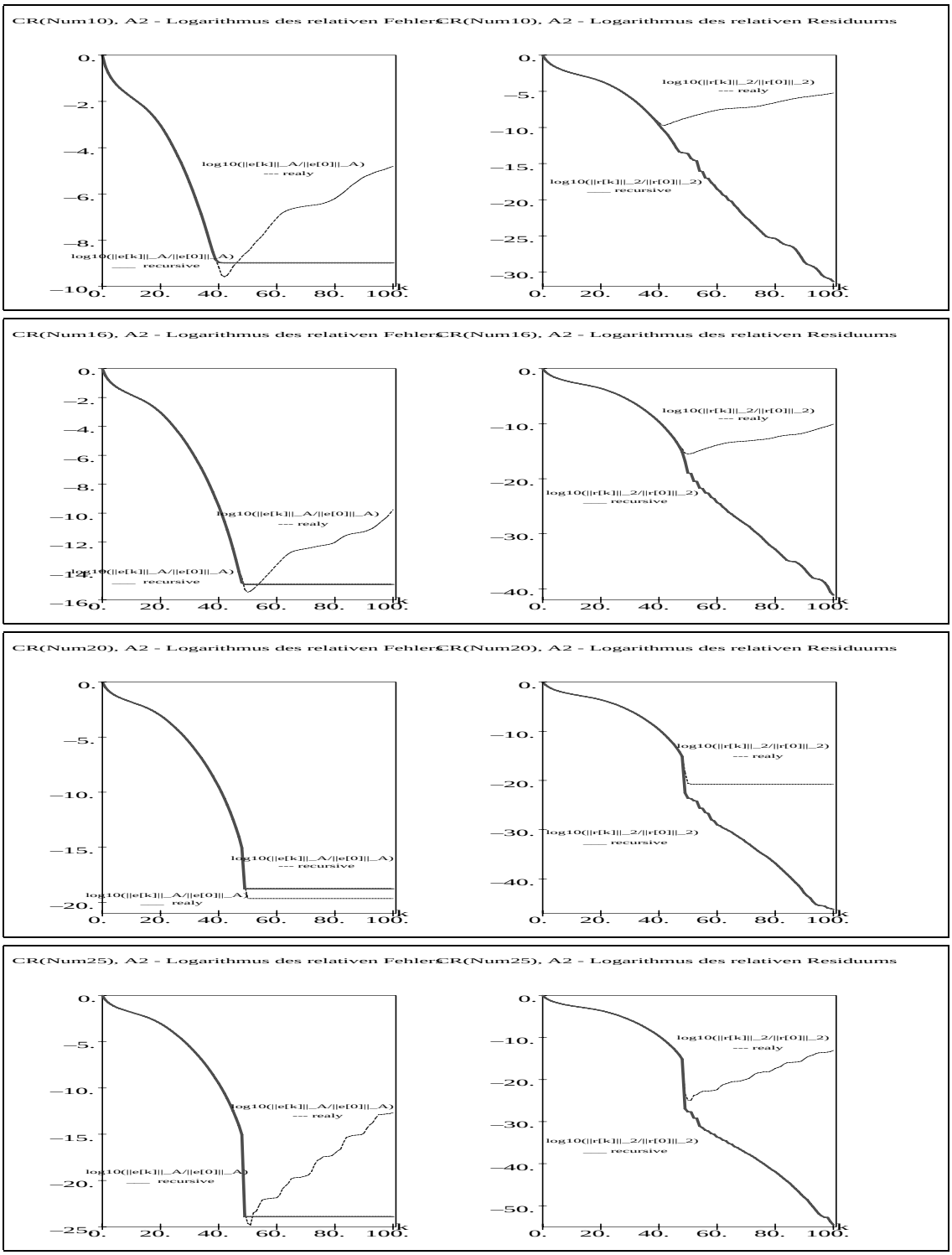


Abb. 7.55 Dateien $cr4a2110.ps, cr4a2116.ps, cr4a2120.ps, cr4a2125.ps$,
$A_2(50,50)$, $c = 1 + 10^{-6}$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

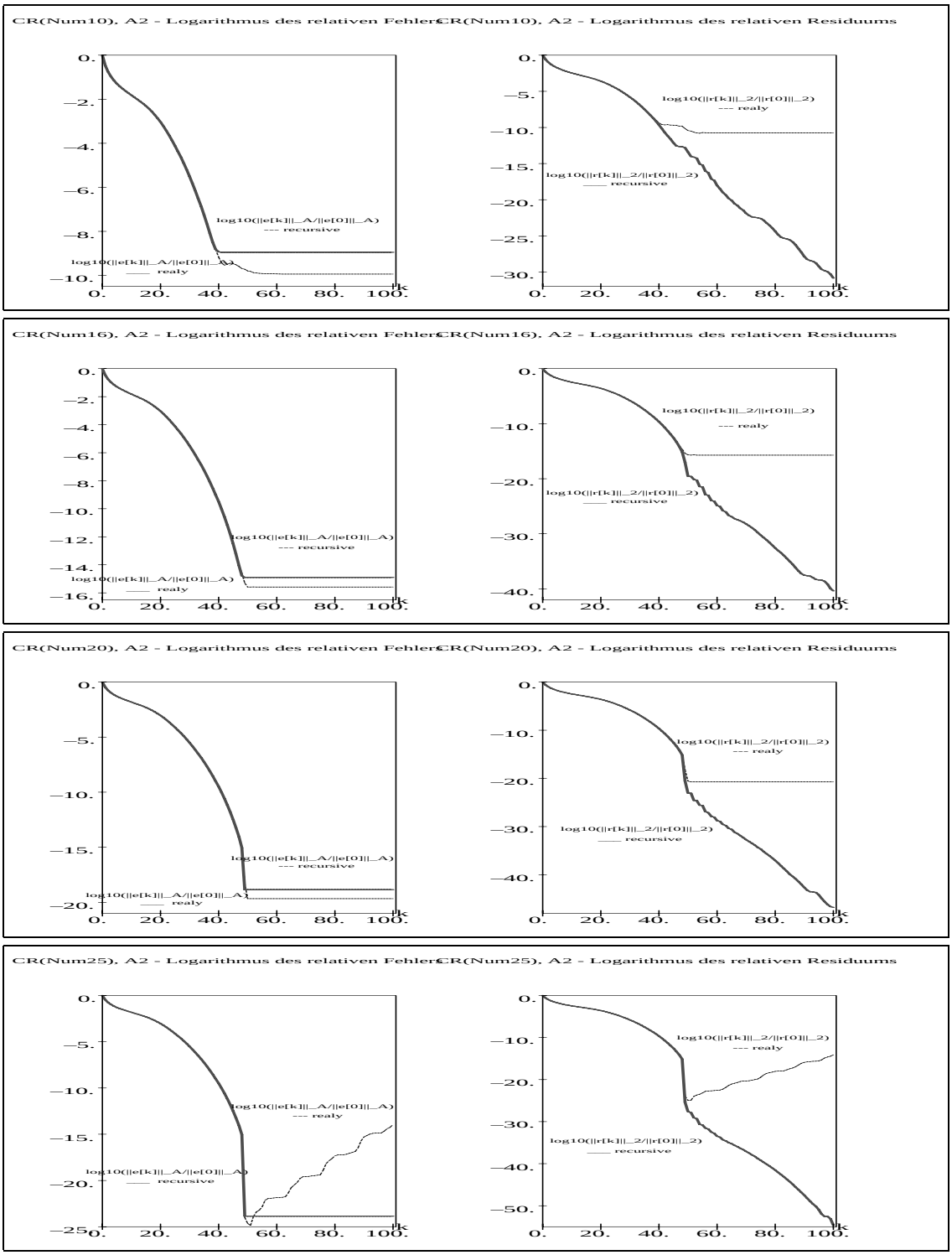


Abb. 7.56 Dateien $cr4a2310.ps, cr4a2316.ps, cr4a2320.ps, cr4a2325.ps$,
$A_2(50,50)$, $c = 1 + 10^{-6}$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

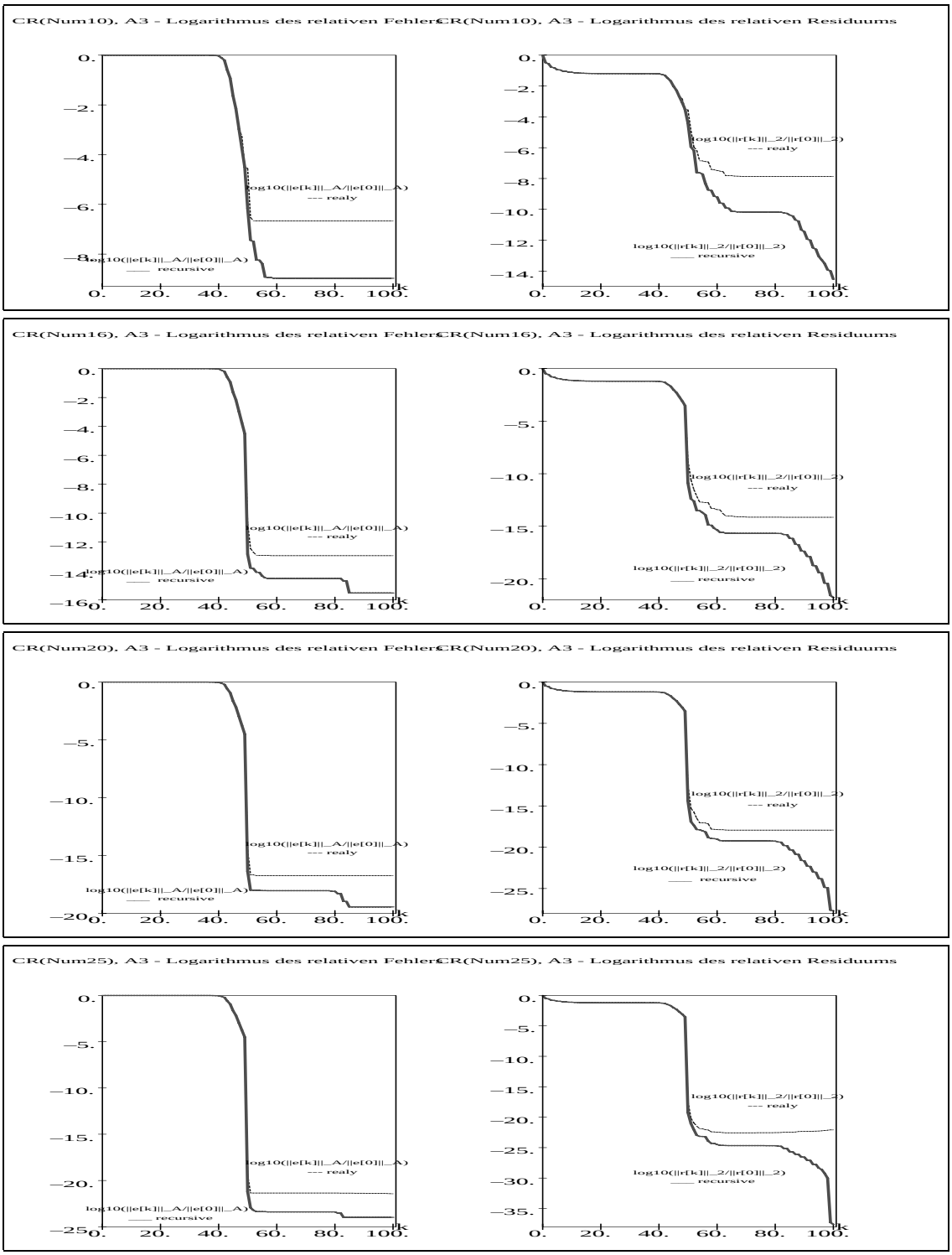


Abb. 7.57 Dateien $cr4a3110.ps, cr4a3116.ps, cr4a3120.ps, cr4a3125.ps$,
$A_3(50,50)$, $c = 1 + 10^{-6}$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

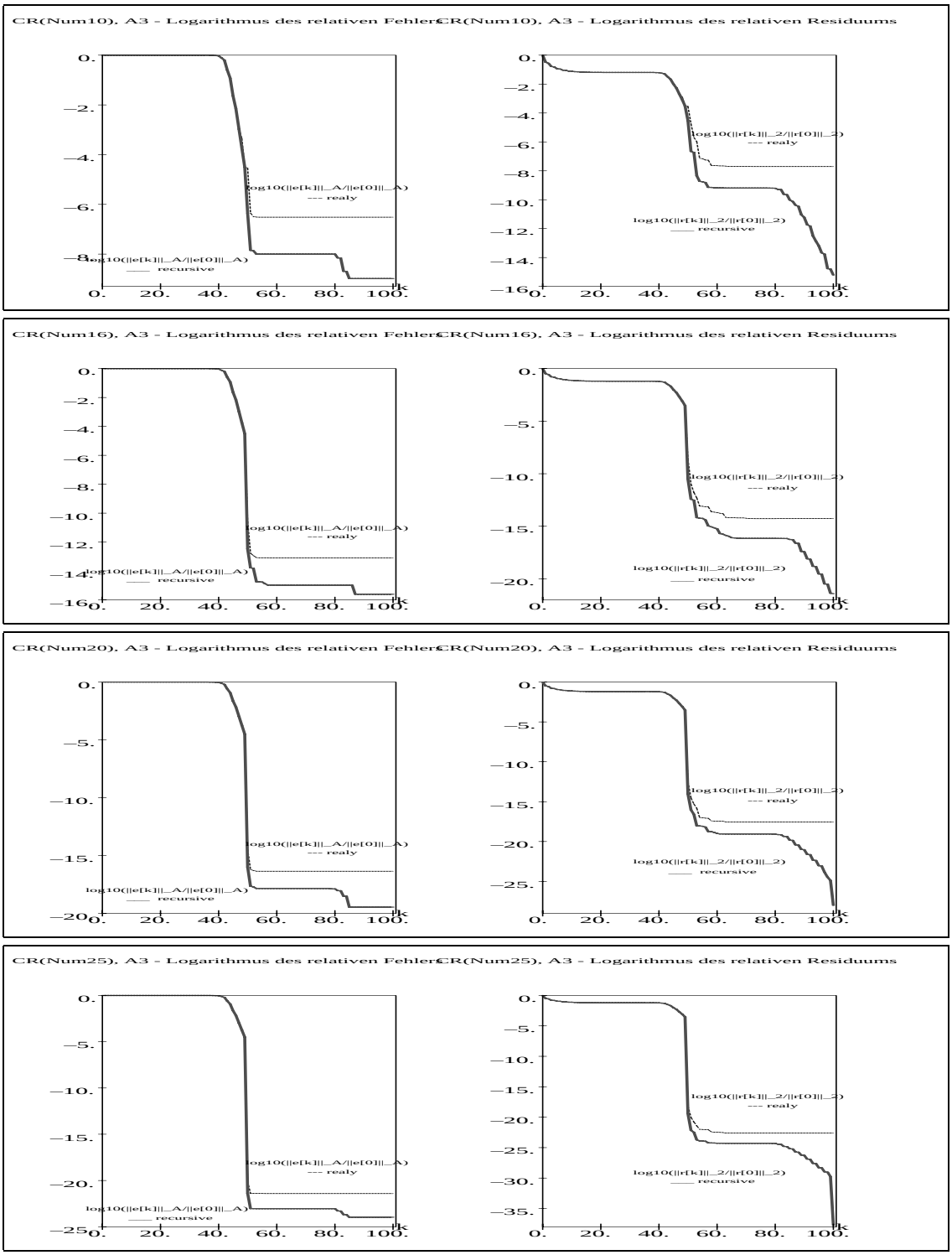


Abb. 7.58 Dateien $cr4a3310.ps, cr4a3316.ps, cr4a3320.ps, cr4a3325.ps$,
$A_3(50,50)$, $c = 1 + 10^{-6}$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

7.2.5 Beispiel 5

```
# Beispiel 5, A=A'>0 mit Tridiagonalstruktur

# Bsp. 5.1, A1 Tridiagonalmatrizen mit spezieller Struktur und Shift c
# 1D-Laplace-Operator auf aequidistantem Gitter
# 3-Punkte-Differenzenstern

 c:=0:
 n:=100:
 A1:=band([-1,2,-1],n):
 A1:=evalm(A1+c*diag(1$n));
 b1:=vector(n,[1+c,c$(n-2),1+c]);
# Loesung
 xs1:=vector(n,[1$n]):  # 1,1,1,....
-------------------------------------------------------------------------

# Bsp. 5.2, A2 Blocktridiagonalmatrizen mit spezieller Struktur und Shift c
# 2D-Laplace-Operator auf quadratischem Gitter
# 5-Punkte-Differenzenstern
# Generierung, Dauer: ca 40 sec bei n1=25, PC 800MHz

 c:=0:
 n1:=10:   # 25:
 n:=n1^2:
 sta:=time():
 T:=band([-1,4,-1],n1):
 A2:=diag(T$n1):
 i:='i':
 for i from 1 to n-n1 do
  A2[i,i+n1]:=-1;
  A2[i+n1,i]:=-1;
 end do:
 A2:=evalm(A2):
 print(time()-sta);
 A2:=evalm(A2+c*diag(1$n)):

# Multiplikation, Dauer: ca 70 sec bei n1=25, n=625, PC 800MHz
# Zeiteinsparung durch viele Nullen in der Matrix
 xs2:=evalm(vector(n,[1$n])):
 sta:=time():
 b2:=evalm(A2&*xs2)   # rechte Seite
 print(time()-sta);

# Loesung
 xs2:=vector(n,[1$n]):
```

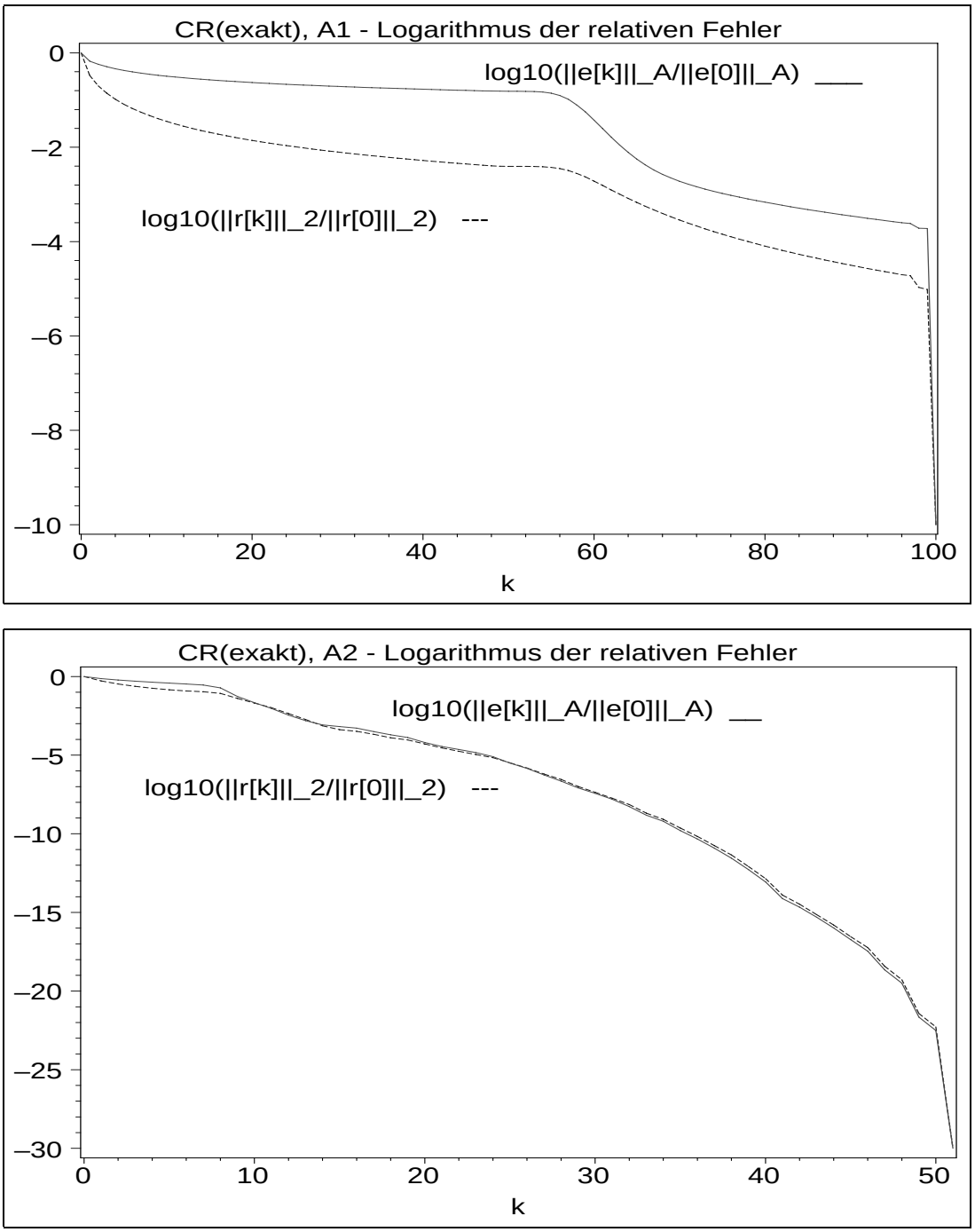


Abb. 7.59 Dateien *cr5a1_01.ps*, *cr5a2_01.ps*,
Matrix $A_{1,2}(100, 100)$, $c = 0$ zu Beispiel 5,
CR: exakte Rechnung (symbolische Rechnung mit Rationalarithmetik),
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...$

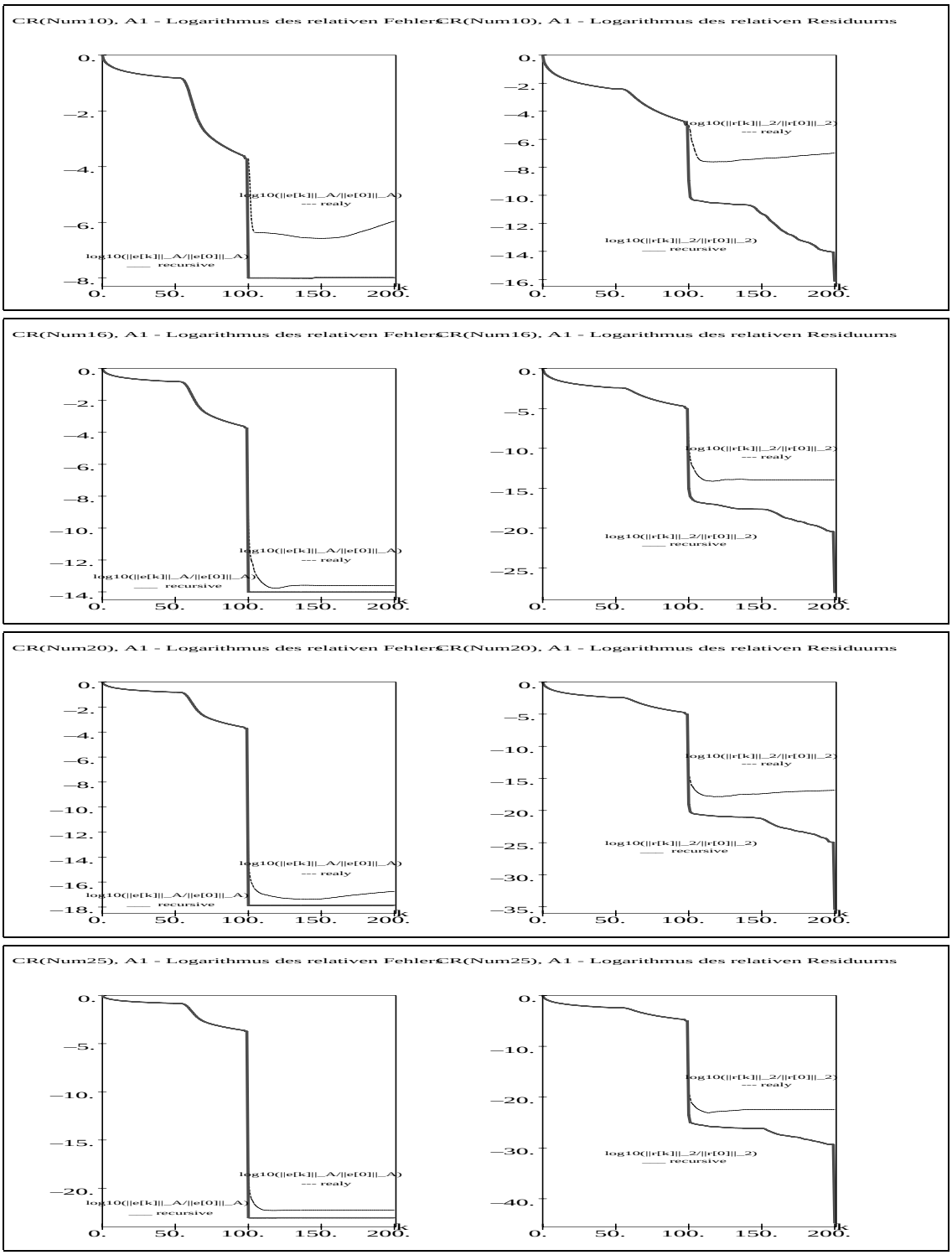


Abb. 7.60 Dateien $cr5a1110.ps, cr5a1116.ps, cr5a1120.ps, cr5a1125.ps$,
$A_1(100, 100)$, $c = 0$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

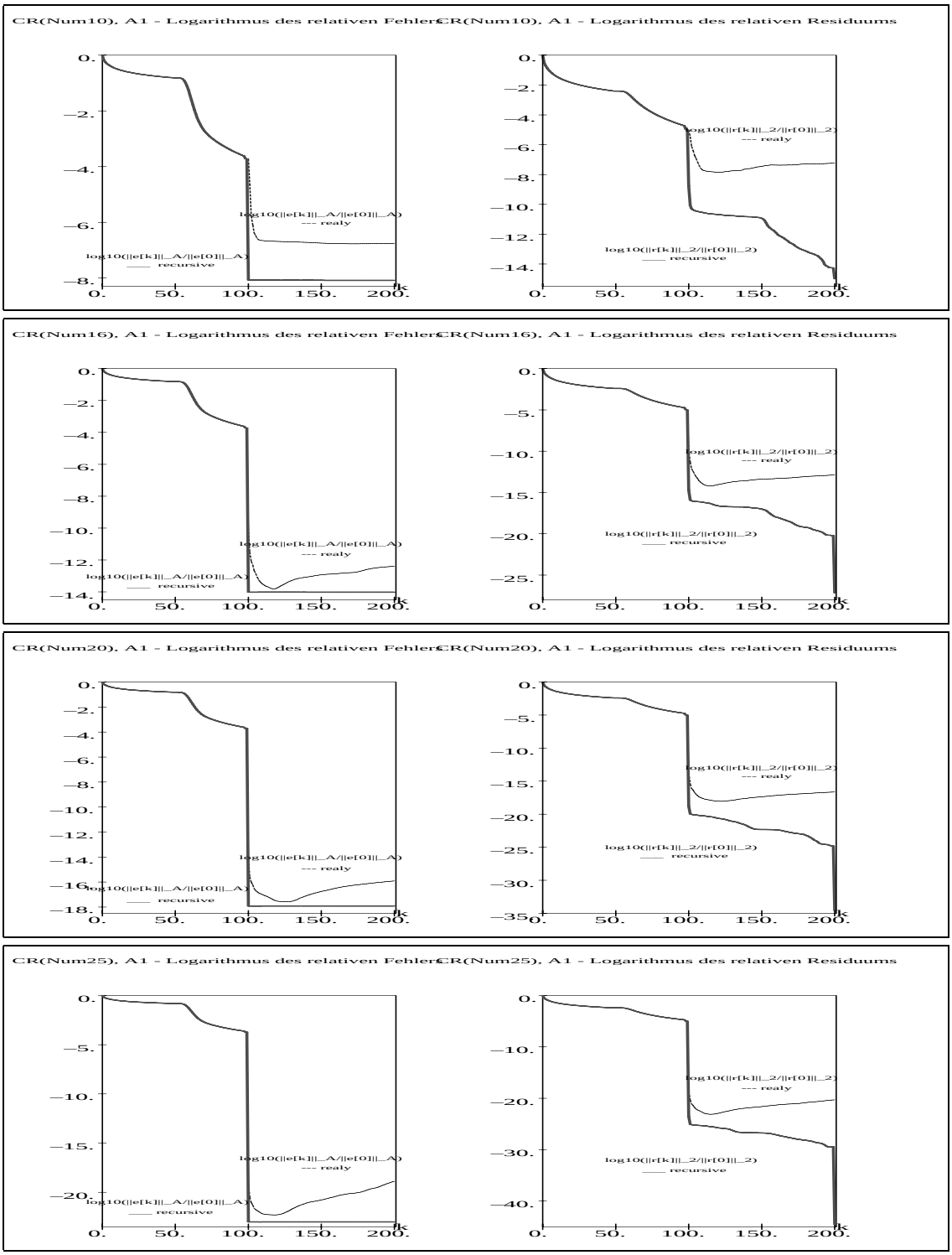


Abb. 7.61 Dateien $cr5a1310.ps, cr5a1316.ps, cr5a1320.ps, cr5a1325.ps$,
$A_1(100, 100)$, $c = 0$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

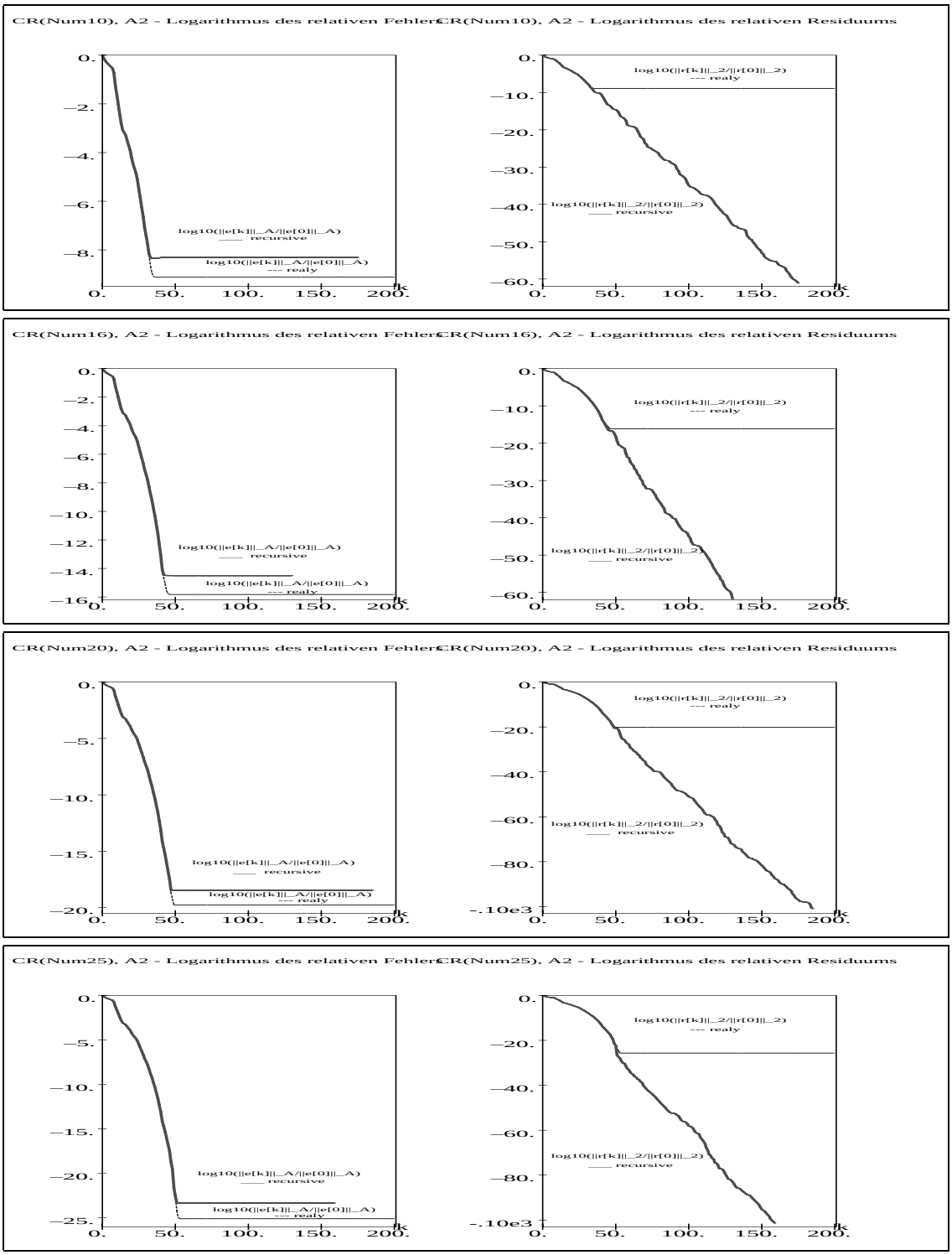


Abb. 7.62 Dateien $cr5a2110.ps, cr5a2116.ps, cr5a2120.ps, cr5a2125.ps$,
$A_2(100, 100)$, $c = 0$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 1,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

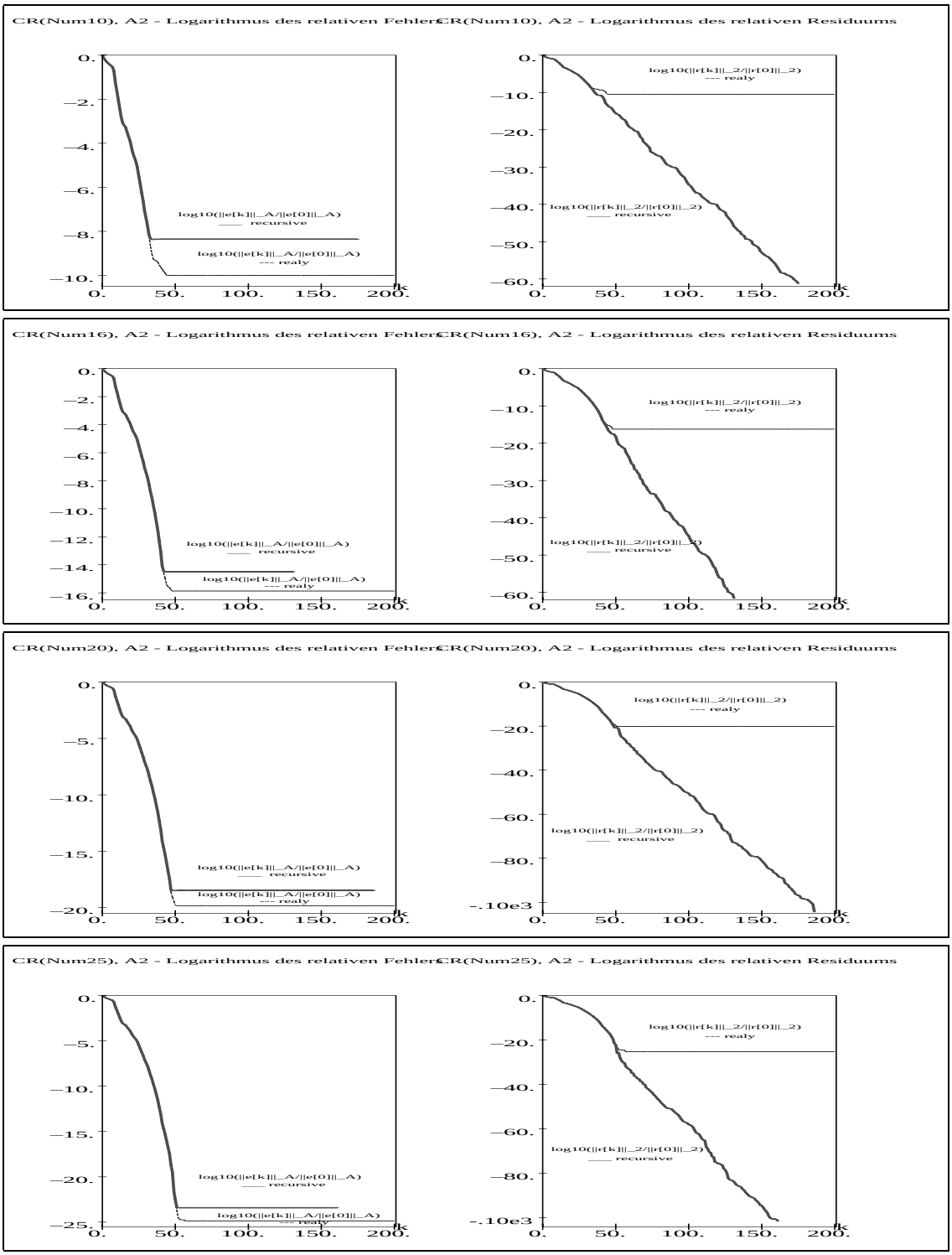


Abb. 7.63 Dateien $cr5a2310.ps, cr5a2316.ps, cr5a2320.ps, cr5a2325.ps$,
$A_2(100,100)$, $c = 0$,
CR: numerische Rechnung, Digits=10,16,20,25, α-Variante 3,
Verlauf der relativen Fehler $\log_{10}(\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A}), \log_{10}(\frac{\|r^{(k)}\|_2}{\|r^{(0)}\|_2})$, $k = 0, 1, ...,$
in Gegenüberstellung der 2 r-Varianten `recursive` und `realy`

Literaturverzeichnis

[1] Axelsson, O.: *Iterative Solution Methods*. Cambridge University Press Cambridge 1994, 1996.

[2] Beresin, I. S. und N. P. Shidkow: *Numerische Methoden*. Bd. 1,2. DVW Berlin 1970, 1971.

[3] Brezinski, C.: *Projection Methods for Systems of Equations*. Studies in Computational Mathematics. Elsevier Amsterdam 1997.

[4] Deuflhard, P. und H. Hohmann: *Numerische Mathematik*. 1: Eine algorithmisch orientierte Einführung. 3. überarbeitete und erweiterte Auflage, Lehrbuch. Walter de Gruyter Berlin 2002.

[5] Faddejew, D. K. und W. N. Faddejewa: *Numerische Methoden der linearen Algebra*. Math. für Naturwiss. und Technik, Bd. 10. DVW Berlin 1973.

[6] Fischer, B.: *Polynomial Based Iteration Methods for Symmetric Linear Systems*. Advances in Numerical Mathematics. Wiley-Teubner Stuttgart 1996.

[7] Greenbaum, A.: *Iterative Methods for Solving Linear Systems*. SIAM Philadelphia 1997.

[8] Hackbusch, W.: *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. Leitfäden der angewandten Mathematik und Mechanik Band 69. B. G. Teubner Stuttgart 1991, 1993.

[9] Hämmerlin G. und K.-H. Hoffmann: *Numerische Mathematik*. Grundwissen Mathematik 7. Springer-Verlag Berlin 1991.

[10] Hermann, M.: *Numerische Mathematik*. R. Oldenbourg Verlag München 2001.

[11] Hestenes, J. R. und E. Stiefel: *Methods of conjugate gradients for solving linear systems*. Journ. Res. Nat. Bur. Stand. 49 (1952) 409-436.

[12] Kelley, C. T.: *Iterative Methods for Linear and Nonlinear Equations*. Frontiers in Applied Mathematics. SIAM Philadelphia 1995.

[13] Kiełbasiński, A. und H. Schwetlick: *Numerische lineare Algebra*. DVW Berlin 1988.

[14] Maess, G.: *Vorlesungen über Numerische Mathematik I, II*. Akademie-Verlag Berlin 1984, 1988.

[15] Meister, A.: *Numerik linearer Gleichungssysteme*. Ein Einführung in moderne Verfahren. Vieweg Braunschweig 1999.

[16] Meurant, G.: *Computer Solution of Large Linear Systems*. Studies in Mathematics and Its Applications, Vol 28. Elsevier Science B. V. 1999.

[17] Neundorf, W.: *Numerische Mathematik*. Vorlesungen, Übungen, Algorithmen und Programme. Shaker Verlag Aachen 2002.

[18] Neundorf, W.: *Grundlagen der numerischen linearen Algebra*. Preprint No. M 04/04 IfMath der TU Ilmenau, Februar 2004.

[19] Opfer, G.: *Numerische Mathematik für Anfänger*. Vieweg Studium Grundkurs Mathematik Wiesbaden 1993, 3. überarbeitete und erw. Auflage 2001.

[20] Plato, R.: *Numerische Mathematik kompakt*. Grundlagenwissen für Studium und Praxis. Vieweg Wiesbaden 2000.

[21] Quarteroni, A., R. Sacco und F. Saleri: *Numerische Mathematik*. Band 1, 2. Springer-Verlag Berlin 2002.

[22] Ralston, A.: *A First Course in Numerical Analysis*. McGraw-Hill New York 1965.

[23] Roos, H.-G. und H. Schwetlick: *Numerische Mathematik*. Das Grundwissen für jedermann. B. G. Teubner Stuttgart 1999.

[24] Saad, Y.: *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company Boston 1995.

[25] Schwarz, H. R.: *Numerische Mathematik*. B. G. Teubner Stuttgart 1988.

[26] Schwarz, H. R., H. Rutishauser und E. Stiefel: *Numerik symmetrischer Matrizen*. B. G. Teubner Stuttgart 1972.

[27] Stoer, J. und R. Bulirsch: *Einführung in die Numerische Mathematik II*. Heidelberger Taschenbücher 114. Springer-Verlag Berlin 1990.

[28] Trefethen, L. N. und D. Bau: *Numerical Linear Algebra*. SIAM Philadelphia 1997.

[29] Überhuber, C.: *Computer-Numerik 1,2*. Springer-Verlag Berlin 1995.

[30] Van der Vorst, H. A.: *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press Cambridge 2003.

[31] Kanzow, Ch.: *Numerik linearer Gleichungssysteme*. Direkte und iterative Verfahren. Springer-Verlag Berlin Heidelberg 2005.

[32] Watkins, David S.: *Fundamentals of Matrix Computations*. 2nd Ed. Pure and Applied Mathematics. A Wiley-Interscience Series of Texts, Monographs, and Tracts. A John Wiley & Sons, Inc., Publication, New York 2002.

[33] Neundorf, W.: *Zu Orthogonalsystemen von Polynomen und ihrer Rekursion*. Preprint No. M 08/05 IfMath der TU Ilmenau, Mai 2005.

Anschrift:

Dr. rer. nat. habil. Werner Neundorf
Technische Universität Ilmenau, Institut für Mathematik
PF 10 05 65
D - 98684 Ilmenau

E-mail : werner.neundorf@tu-ilmenau.de
Homepage : http://www.mathematik.tu-ilmenau.de/∼neundorf/index_de.html