Objektorientierte Formulierung von Anfangs- und Randwertbedingungen bei elektrochemischen Simulationen

Wissenschaftliche Prüfung für das Lehramt an Gymnasien

Wissenschaftliche Arbeit im Fach Chemie

vorgelegt von

Heiko Anders

Universität Tübingen
19. Mai 2005

Ich erkläre, dass ich die Arbeit selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnung kenntlich gemacht worden sind.

Tübingen, im Mai 2005 Heiko Anders

Danksagung

Ganz herzlich bedanken möchte ich mich bei Herrn Prof. Dr. Bernd Speiser, für die Vergabe des interessanten Themas und die ausgezeichnete Betreuung während der Erstellung dieser Arbeit.

Mein Dank gilt dazu noch allen Mitgliedern des Arbeitskreises Speiser: Anna Budny, Kai Ludwig, Wolfgang Märkle, Petra Marschner, Filip Novak, Nicolas Plumeré, Elena Popa, Markus Schwarz und Carsten Tittel für ein sehr nettes Arbeitsklima in diesen sechs Monaten. Besonderer Dank gilt Kai Ludwig, der mir in vielen Fragen zu meiner Zulassungsarbeit rund um C++ geduldig zugehört und geholfen hat.

Meinen Eltern danke ich sehr für ihre Unterstützung und die Finanzierung meines Studiums.

Meiner Freundin Stefanie Wais danke ich für häufige Anregungen und ein erstes Korrekturlesen dieser Zulassungsarbeit.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Durch die stürmische Entwicklung der Computerleistungsfähigkeit der letzten Jahrzehnte, gewinnen computergesteuerte Simulationen von Naturvorgängen immer mehr an Einfluß in der heutigen naturwissenschaftlichen Forschung oder werden erst möglich gemacht. Was einst Monate an Rechenzeit — teils noch per Hand — verschlang, ist heute in Sekundenbruchteilen möglich [1].

Solche Simulationen werden meist mit der Zeit als Laufzeitvariablen durchgeführt. Anhand eines mathematischen Modells wird untersucht, wie sich ein gegebenes System von Anfangsbedingungen über die Zeit hinweg entwickelt. An bestimmten Ortspunkten werden dabei Randbedingungen definiert. Als besonders aufwendiges Beispiel sei die Wettersimulation oder die Simulation von aerodynamischem Verhalten erwähnt. Wichtig ist es, die Natur möglichst gut approximierende Modelle zu entwickeln. Auch die Anfangsbedingungen sollten so exakt als möglich gemessen oder berechnet werden.

Auch die moderne Chemie befasst sich mit aufwendigen Computersimulationen, so das Echem++-Projekt von SPEISER UND LUDWIG [2, 3] mit der Simulation elektrochemischer Prozesse. Möchte man chemische Vorgänge in einem flüssigen Gemisch verschiedener Substanzen simulieren, ist es wichtig, als Anfangs- und Randwertbedingungen die Konzentrationen dieser Substanzen sorgfältig zu messen oder zu berechnen. Sind alle chemischen Reaktionen in diesem Modell enthalten, sucht man sich diese heraus, die vor $t = 0$ stattgefunden haben sollen. Dies sind normalerweise irreversible Reaktionen und vorgelagerte Gleichgewichte — aus denen sich die Anfangskonzentrationen ergeben werden. Interessant ist zunächst die generelle Frage, ob mehrere verschiedene Anfangskonzentrationsverteilungen existieren können. Zu ihrer endgültigen Berechnung sind weitere zusätzliche Informationen nötig, die in diesem Simulationsmodell enthalten sein müssen. Die Randbedingungen für die Integration hängen von der simulierten Geometrie und der Art des durchgeführten Experiments ab [4].

So benutzen WESTAL et al. [5] Atombilanzgleichungen und Gleichgewichtskon-

stanten, WELTIN [6] einen thermodynamischen Weg über Standardbildungsenthalpien der Reaktanden und LUDWIG [2][3] löst das Problem mit Geschwindigkeitskonstanten durch Integration gewöhnlicher Differentialgleichungen.

Eine allgemeine Lösung ohne weitere Informationen als den Gleichgewichtskonstanten ist bisher in der Literatur und der Simulation wenig beachtet worden. Im Rahmen des Echem++-Projekts wird diese Art von Lösung jedoch benötigt, da in diesem Modell keine zusätzlichen Informationen wie Bildungsenthalpien oder Summenformeln der abstrakt formulierten Reaktanden "A,B,C, ..." beschrieben werden.

Viele Programmiersprachen wurden für die Simulation chemischer Prozesse verwendet. Bieniasz [7] verwendet C, WESTAL et al. Basic und SPEISER UND LUDWIG in Echem++ die Programmiersprache C++, die von Stroustrup [8] entwickelt wurde. C++ erlaubt es dem Anwender objektorientiert zu arbeiten. Die objektorientierte Programmierung stellt sicher, dass implementierte Prozeduren und Funktionen an jedes C++-Projekt ohne Änderung des Codes angegliedert werden können.

Kapitel 2

Problemstellung

Das Ziel dieser Arbeit ist es, rechnergestützt **Anfangs- und Randwertbedingungen bei elektrochemischen Simulationen** – in diesem Fall im Rahmen des Echem++-Projekts [2, 3] – effizient zu berechnen.

Dazu werden den zu simulierenden Reaktionen vorgelagerte Gleichgewichtsreaktionen angenommen. Zusammen mit den Anfangskonzentrationen zum Zeitpunkt $t = 0$ und den Gleichgewichtskonstanten dieser Reaktionen sollen die Gleichgewichtskonzentrationen aller Reaktanden nach Einstellung sämtlicher Gleichgewichte berechnet werden. Das Problem der Berechnung dieser Konzentrationen wird im folgenden als das (mehrdimensionale) **"Equilibrium-Problem"** bezeichnet. Als Einschränkung soll es gelöst werden, ohne weitere Informationen über das Reaktionssystem zu verwenden (wie Geschwindigkeitskonstanten, Standardbildungsenthalpien, Summenformeln, Massenbilanzgleichungen, ...). Nach einer Herleitung des Massenwirkungsgesetzes soll die

- **eindeutige Lösbarkeit des Equilibrium-Problems**

aus thermodynamischer und kinetischer Sicht untersucht werden. Danach sollen mathematische Modelle für das Equilibrium-Problem beschrieben werden, um daraus

- **rechnerische Methoden zur Lösung des Equilibrium-Problems**

zu entwickeln. Im Laufe der Arbeit zeigt sich, dass das Equilibrium-Problem mathematisch gesehen mit einer Nullstellensuche ein- oder mehrdimensionaler Funktionen korrespondiert. Zur Umsetzung dieser und zur Motivation des am weitesten entwickelten *Broydenverfahrens* sollen

- **numerische Verfahren zur Lösung des Equilibrium-Problems und deren objektorientierte Implementierung in C++**

durchgeführt werden. Die Wahl der modernen Programmiersprache C++ und eine objektorientierte Programmiertechnik, dient unter anderem dem Zweck, die

- **Anbindung an das Echem++-Projekt**

zur Berechnung von Anfangswerten bei der Simulation elektrochemischer Reaktionen reibungslos durchführen zu können und gleichzeitig sicherzustellen, dass eine Anbindung an andere Projekte ebenso einfach möglich ist.

Die berechneten Gleichgewichtskonzentrationen stellen die homogenen Anfangsbedingungen für die Integration der partiellen Differentialgleichungen dar, deren Lösung das Ziel einer elektrochemischen Simulation ist. Gleichzeitig werden diese Konzentrationen als Randbedingungen für Orte weit entfernt von der Elektrode benutzt, wo während der Experimente keine Konzentrationsänderungen erfolgen sollen.

Kapitel 3

Theoretische Lösung des Equilibrium-Problems

3.1 Chemisches Gleichgewicht, Reaktionslaufzahl und das Massenwirkungsgesetz

Chemische Reaktionen müssen bekanntermaßen nicht vollständig von den Edukten zu den Produkten hin ablaufen. Ein **chemisches Gleichgewicht** hat sich eingestellt, wenn Edukte und Produkte nebeneinander vorliegen, sich aber Ihre Stoffmengen nicht mehr ändern.

Die chemische Thermodynamik liefert Aussagen über die Lage dieser Gleichgewichte mit Hilfe des thermodynamisch motivierten Massenwirkungsgesetzes. Eine *kinetische Herleitung* des Massenwirkungsgesetzes wird im darauf folgenden Abschnitt diskutiert.

Für eine allgemeine Gleichgewichtsreaktion (3.1) aus n Edukten und m Produkten

$$|\nu_1|\mathrm{A_1} + ... + |\nu_n|\mathrm{A_n} \rightleftharpoons \nu_{n+1}\mathrm{A_{n+1}} + ... + \nu_{n+m}\mathrm{A_{n+m}} \tag{3.1}$$

gilt:

$$K_{c/x/p} = \prod_{i=1}^{n+m} [\mathrm{A_i}]^{\nu_i} \tag{3.2}$$

Dies ist die allgemeinste Formulierung des **Massenwirkungsgesetzes** (3.2) ([9], S. 378). Der Index c, x oder p zeigt an, dass sich die Gleichgewichtskonstante auf Konzentrationen, Molenbrüche oder Partialdrücke bezieht.

Dabei tragen die stöchiometrischen Koeffizienten der Edukte negative und die der Produkte konventionsgemäß positive Vorzeichen. Man bezeichnet K als die *Gleichgewichtskonstante.*

Die Reaktionslaufzahl ξ

Die Änderungen der Stoffmengen n_j der verbrauchten Edukte A, B verhalten sich wie

ihre stöchiometrischen Faktoren — für die Produkte (P, Q) gilt trotz der unterschiedlichen Vorzeigen der Stöchiometriekoeffizienten dasselbe ([9], S. 378) (Gleichungen 3.3, 3.4)

$$\frac{n_A(Ende) - n_A(Anfang)}{n_B(Ende) - n_B(Anfang)} = \frac{\nu_A}{\nu_B} \tag{3.3}$$

$$\frac{n_P(Ende) - n_P(Anfang)}{n_Q(Ende) - n_Q(Anfang)} = \frac{\nu_P}{\nu_Q} \tag{3.4}$$

Für einen differentiellen Umsatz gilt (3.5):

$$\frac{\mathrm{d}n_A}{\nu_A} = \frac{\mathrm{d}n_B}{\nu_B} = \frac{\mathrm{d}n_P}{\nu_P} = \frac{\mathrm{d}n_Q}{\nu_Q} \tag{3.5}$$

Beschreibt man den Fortschritt der Reaktion mit Hilfe der Stoffmengen von A, B, P, Q, d. h. mit den *Änderungen der Stoffmengen der einzelnen Reaktanden* $\mathrm{d}n_A$, $\mathrm{d}n_B$, $\mathrm{d}n_P$, $\mathrm{d}n_Q$, so käme man zu zahlenmäßig unterschiedlichen Ergebnissen aufgrund der verschiedenen stöchiometrischen Faktoren. Deshalb normiert man die Stoffmengenänderungen mit diesen und erhält eine einheitliche Zahl, die genau den Reaktionsfortgang beschreibt — die **Reaktionslaufzahl** ξ (3.6):

$$d\xi = \frac{dn_i}{\nu_i} \tag{3.6}$$

Allerdings ist die Bezeichnung „Zahl“ etwas unglücklich, da es sich hier um eine Größe der Einheit Stoffmenge handelt. Bei $d\xi = 1$ mol haben sich gerade ν_A mol A mit ν_B mol B zu ν_P mol P und ν_Q mol Q umgesetzt.

3.1.1 Chemisches Gleichgewicht — Thermodynamischer Ansatz

Im folgendem wird die allgemeine Reaktionsgleichung (3.1) betrachtet (aus [10], S. 108). Die **Gibbs Energie G** (auch freie Enthalpie) des Systems berechnet sich aus den Stoffmengen und den chemischen Potentialen der Reaktanden (3.7)

$$G = \sum_{i=0}^{n+m} n_i \mu_i \tag{3.7}$$

Die chemischen Potentiale μ_i ändern sich allerdings mit der Zeit, da sich im Verlauf einer Reaktion auch Konzentrationen ändern werden. μ_i gibt die *aktuellen* **chemischen Potentiale** zu einem bestimmten Zeitpunkt an. Dasselbe gilt für die Stoffmengen n_i. Führt man wieder die bekannte Reaktionslaufzahl ξ ein (3.6), so wird klar, dass die Gibbs-Energie G eine Funktion von ξ sein muss.
Aus der Bedingung $dG < 0$ für spontan ablaufende Prozesse folgt, dass nur dann eine

Reaktion stattfindet, wenn die Änderung der Gibbs-Funktion $\frac{\partial G}{\partial \xi}$ negativ ist. Im Minimum der Gibbs-Funktion ist die Änderung null und das **chemische Gleichgewicht** ist erreicht. Mit Gleichung (3.6) und Gleichung (3.7) gilt:

$$\mathrm{d}G = \sum_{i=1}^{n+m} \mu_i \mathrm{d}n_i = \sum_{i=1}^{n+m} \nu_i \mu_i \mathrm{d}\xi \tag{3.8}$$

und somit

$$\frac{\partial G}{\partial \xi} = \sum_{i=1}^{n+m} \nu_i \mu_i \tag{3.9}$$

Den Ausdruck (3.9) kann man mit der *molaren Reaktions-Gibbs-Energie* (oder molaren freien Reaktionsenthalpie) (3.10) identifizieren:

$$\Delta G_m = \sum_{i=1}^{n+m} \nu_i \mu_i \tag{3.10}$$

Nimmt man nun an, dass die Reaktion in flüssiger Phase abläuft (die Aktivitäten a_i der einzelnen Komponenten bleiben ausser acht, im folgenden werden näherungsweise die Konzentrationen c_i benutzt), gilt für das chemische Potential der i. Reaktionskomponente definitionsgemäß ([10], S. 109) (3.11)

$$\mu_i = \mu_i^0 + \mathrm{RT}\ \ln c_i \tag{3.11}$$

und für die molare Reaktions-Gibbs-Energie (3.14)

$$\begin{aligned} \Delta G_m &= \sum_{i=1}^{n+m} \nu_i \left(\mu_i^0 + \mathrm{RT} \ln c_i\right) \\ &= \sum_{i=1}^{n+m} \nu_i \mu_i^0 + \mathrm{RT} \sum_{i=1}^{n+m} \nu_i \ln c_i \\ &= \Delta G_m^0 + \mathrm{RT}\ \ln \prod_{i=1}^{n+m} c_i^{\nu_i} \end{aligned} \tag{3.12}$$

ΔG_m^0 ist die molare *Standard-Reaktions-Gibbs-Energie* als Summe der chemischen Standardpotentiale μ_i^0. Im Gleichgewicht ist, wie eingangs erwähnt, ΔG_m null. Es ergibt sich

$$\Delta G_m^0 = -\mathrm{RT} \ln K c \tag{3.13}$$

mit

$$K_\mathrm{C} = \prod_{i=1}^{n+m} c_i^{\nu_i} \tag{3.14}$$

K_C nennt man *Gleichgewichtskonstante* und Gleichung (3.14) (das thermodynamisch hergeleitete) *Massenwirkungsgesetz*.

3.1.2 Chemisches Gleichgewicht — Kinetischer Ansatz

Eine kinetische und allgemeine Herleitung des Massenwirkungsgesetzes erweist sich als schwierig und wenig anschaulich — aber das grundlegende Prinzip und damit der Unterschied zum thermodynamischen Ansatz und die eventuell daraus entstehenden Lösungsmöglichkeiten für das Equilibrium-Problem sollen im folgenden Abschnitt (für einen Spezialfall) verdeutlicht werden (aus [11], S. 827).

Eine chemische Reaktion muss nicht vollständig ablaufen. So existiert zu einer Hinreaktion (3.15)

$$\nu_A \mathrm{A} + \nu_B \mathrm{B} \xrightarrow{k_1} \nu_C \mathrm{C} + \nu_D \mathrm{D} \tag{3.15}$$

auch eine Rückreaktion (3.16)

$$\nu_C \mathrm{C} + \nu_D \mathrm{D} \xrightarrow{k_{-1}} \nu_A \mathrm{A} + \nu_B \mathrm{B} \tag{3.16}$$

die aus den Produkten wieder die Edukte zurückbildet. Die Reaktionen haben im allgemeinen verschiedene Geschwindigkeitskonstanten k_1 und k_{-1}. In dem angeführten Beispiel seien die Reaktionen zweiter Ordnung.
Mit der Reaktionslaufzahl ξ und den Anfangskonzentrationen $[A_0]$, $[B_0]$, $[C_0]$, $[D_0]$ lauten die *Geschwindigkeitsgleichungen* (3.17, 3.18) für Reaktionen zweiter Ordnung für die Hinreaktion

$$\frac{\mathrm{d}\xi_{\mathrm{hin}}}{\mathrm{d}t} = k_1([\mathrm{A}_0] + \nu_A\xi)([\mathrm{B}_0] + \nu_B\xi) \tag{3.17}$$

und die Rückreaktion

$$\frac{\mathrm{d}\xi_{\mathrm{rück}}}{\mathrm{d}t} = -k_{-1}([\mathrm{C}_0] + \nu_C\xi)([\mathrm{D}_0] + \nu_D\xi) \tag{3.18}$$

Für die Gleichgewichtsreaktion ergibt sich die Gesamtreaktionsgeschwindigkeit (3.19)

$$\frac{\mathrm{d}\xi}{\mathrm{d}t} = k_1([\mathrm{A}_0] + \nu_A\xi)([\mathrm{B}_0] + \nu_B\xi) - k_{-1}([\mathrm{C}_0] + \nu_C\xi)([D_0] + \nu_D\xi) \tag{3.19}$$

Im Gleichgewicht ändern sich die Konzentrationen nicht mehr, es gilt

$$\frac{\mathrm{d}\xi}{\mathrm{d}t} = 0 \tag{3.20}$$

Das heißt nicht, dass alle Reaktionen zum Stillstand kommen. Der Betrag der Geschwindigkeit der Hinreaktion ist im Gleichgewichtszustand gleich dem der Rückreaktion (3.21) — bei einer eindeutigen Reaktionslaufzahl ξ_{eq}

$$k_1([\mathrm{A}_0] + \nu_A\xi_{\mathrm{eq}})([\mathrm{B}_0] + \nu_B\xi_{\mathrm{eq}}) = k_{-1}([\mathrm{C}_0] + \nu_C\xi_{\mathrm{eq}})([\mathrm{D}_0] + \nu_D\xi_{\mathrm{eq}}) \tag{3.21}$$

Man spricht deshalb auch von einem **dynamischen Gleichgewicht**. Aus Gleichung (3.21) ergibt sich das Massenwirkungsgesetz (3.22) für diesen Fall:

$$\frac{[\mathrm{C}]_{\mathrm{eq}}[\mathrm{D}]_{\mathrm{eq}}}{[\mathrm{A}]_{\mathrm{eq}}[\mathrm{B}]_{\mathrm{eq}}} = \frac{k_1}{k_{-1}} = K_{\mathrm{c}} \tag{3.22}$$

Wie man hier sieht, kann nicht aus jedem Geschwindigkeitsgesetz das allgemeine Massenwirkungsgesetz formuliert werden, da die Reaktionsgeschwindigkeit im allgemeinen nicht immer stetig von den Konzentrationen jedes Reaktionspartners abhängig ist ([12], S.316) — beispielsweise bei Reaktionen pseudo-erster Ordnung.

Der vorgestellte kinetische Ansatz ist somit nicht hilfreich, das Equilibrium-Problem in seiner Allgemeinheit zu lösen.

3.2 Eindeutige Lösbarkeit des Problems

Bevor man die praktische Umsetzung der Lösung des Equilibrium-Problems diskutiert, ist von grundlegendem Interesse, ob tatsächlich eine Lösung existieren muss. Vorausgesetzt sei, das vorgegebene Reaktionssystem enthält keine physikalisch unmöglichen Reaktionen wie beispielsweise (3.23)

$$\mathrm{A} \rightleftharpoons \mathrm{A} + \mathrm{B} \tag{3.23}$$

und die Eingangsdaten sind vollständig (d.h. es fehlen z.B. keine Gleichgewichtskonstanten oder Anfangskonzentrationen).

Von fundamentalem Interesse ist es außerdem, ob immer *genau eine* Lösung existiert oder ob *mehrere verschiedene Gleichgewichtszusammensetzungen* mit denselben Vorgaben an stöchiometrischen Koeffizenten, Gleichgewichtskonstanten und Anfangskonzentrationen möglich sind. Im eindimensionalen Falle (eine Reaktionsgleichung, n Reaktanden) ist der Beweis hierfür recht einfach (siehe Kapitel 3.3.2). Bei einem Reaktionssystem von beliebiger Komplexität zeigt sich die Beweisführung erheblich umfangreicher.

3.2.1 Eindeutigkeit der Lösung des Equilibrium-Problems, thermodynamische Variante

WELTIN beweist sehr elegant die Existenz einer eindeutigen Lösung mit einem thermodynamischen Ansatz [6].
Hier wird ihm folgend der allgemeine Fall betrachtet: gegeben ist ein geschlossenes System aus n Reaktanden, die am Gleichgewichtszustand beteiligt sind, und alle n freien Standardbildungsenthalpien $\Delta G^0_{\mathrm{f}_i}$ der Reaktanden.

In einem Gemisch aller n Reaktanden setzt sich die Gesamtstoffmenge Y (3.24) aus den einzelnen Stoffmengen $y_0, y_1, y_2, \ldots$ zusammen, die jeweils wieder größer oder gleich null sein müssen:

$$Y = \sum_{i=0}^{n} y_i, \quad y_i \geq 0 \ \forall i \tag{3.24}$$

Bei konstantem Druck p und konstanter Temperatur T gilt für die freie Reaktionsenthalpie G (3.25)

$$G = \sum_{i=0}^{n} y_i \mu_i \tag{3.25}$$

Das chemische Potential (3.26) ist als (partielle) Ableitung der freien Enthalpie G nach einer Stoffmenge eines Reaktanden definiert ([13], S. 40) — und zwar bei konstantem Druck, konstanter Temperatur und konstanten Stoffmengen $y_k \neq y_i$:

$$\mu_i = \left(\frac{\partial G}{\partial y_i}\right)_{p,T,y_k} \tag{3.26}$$

Für das chemische Potential gilt somit (3.27)

$$\mu_i = \mu_i^0 + \mathrm{RTln}\left(\frac{y_i}{Y}\right) \tag{3.27}$$

mit Gleichung (3.25) ergibt sich für die freie Reaktionsenthalpie (3.28)

$$G = \sum_{i=0}^{n} n_i(\mu_i^0 + \mathrm{RTln}\left(\frac{n_i}{Y}\right) \tag{3.28}$$

Für die weitere Argumentation wichtig ist die zweite partielle Ableitung von G nach den einzelnen Stoffmengen n_i (bei konstanter Temperatur und konstantem Druck). Die entstehende Matrix heisst *Hessematrix* (=:M,[14], S. 286). Sie wird im folgenden berechnet.

Die ersten partiellen Ableitungen von G nach den einzelnen Stoffmengen y_i (3.29) lauten:

$$\left(\frac{\partial G}{\partial y_i}\right) = \mu_i = \mu_i^0 + \mathrm{RTln}\left(\frac{n_i}{Y}\right) \tag{3.29}$$

Die zweiten partiellen Ableitungen von G nach den verschiedenen Stoffmengen y_i (3.30, 3.31) lauten:

$$\left(\frac{\partial^2 G}{\partial y_i^2}\right) = \mathrm{RT}\left(\frac{1}{y_i} - \frac{1}{Y}\right) \tag{3.30}$$

und

$$\frac{\partial^2 G}{\partial y_j \partial y_i} = -\frac{\mathrm{RT}}{Y} \tag{3.31}$$

Die sich daraus ergebende Hessematrix hat somit folgende Gestalt:

$$M := \begin{pmatrix} \frac{\partial^2 G}{\partial y_1^2} & \frac{\partial^2 G}{\partial y_2 \partial y_1} & \cdots & \frac{\partial^2 G}{\partial y_n \partial y_1} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 G}{\partial y_1 \partial y_n} & \frac{\partial^2 G}{\partial y_2 \partial y_n} & \cdots & \frac{\partial^2 G}{\partial y_n^2}) \end{pmatrix}$$

$$= \begin{pmatrix} \mathrm{RT}(\frac{1}{y_1} - \frac{1}{Y}) & -\frac{\mathrm{RT}}{Y} & \cdots & -\frac{\mathrm{RT}}{Y} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ -\frac{\mathrm{RT}}{Y} & -\frac{\mathrm{RT}}{Y} & \cdots & RT(\frac{1}{y_n} - \frac{1}{Y}) \end{pmatrix}$$

Sind zwei oder mehr Reaktanden beteiligt, gilt $M_{ij} > 0$ (mit nur einem Reaktand wäre die zweite Ableitung 0). M hat einige interessante Eigenschaften:

$$\det(M) = 0 \tag{3.32}$$

Gleichung (3.32) besagt, dass M singulär ist, wie sich einfach nachrechnen läßt. Außerdem sind alle Hauptminoren[1] größer 0 ([16], S. 514). Aus der Linearen Algebra folgt: A ist *positiv semidefinit* (3.33). Das heißt:

$$x^T Ax \geq 0 \quad \forall x \tag{3.33}$$

Aus der Linearen Algebra weiß man[17]: *M hat genau einen Eigenwert von 0 und $n-1$ positive Eigenwerte.*
Der Eigenvektor, der zum Eigenwert 0 gehört, ist proportional zum Vektor

$$y := (y_0, y_1, y_2, y_3, ...) \tag{3.34}$$

Dies ergibt sich folgendermaßen:
Es gilt

$$G(ay) = aG(y) \quad a \in \Re. \tag{3.35}$$

Multipliziert man y mit einem beliebigem Skalar a, ergibt sich dasselbe Ergebnis, wie wenn G mit a multipliziert wird (3.35). Anschaulich würde sich eine Gerade ergeben. Somit muss die Kurvenkrümmung — gleichbedeutend mit der zweiten Ableitung — in Richtung von y gleich null sein. Es gilt somit $My = 0$, d.h. y ist wirklich Eigenvektor zum Eigenwert 0.

Mit der positiven Semidefinitheit von M ergibt sich ein für die folgende Argumentation grundlegender Zusammenhang:

In jeder Richtung $r \neq y$, ist die Kurvenkrümmung ($=r^T Mr$) von G positiv!

In einem geschlossenen System gelten für die Änderung der Zusammensetzung des Systems y^s zu $y^s + \xi \cdot r$ die üblichen Massen- und Ladungserhaltungsgesetze. Die Komponenten von r sind stöchiometrische Koeffizienten chemischer Reaktionen. Es sei erinnert, dass diese konventionsgemäß für Edukte negativ und für Produkte positiv sind. ξ sei die Reaktionslaufzahl relativ zur Startkomposition y^s der Reaktionsrichtung r.
ξ ist begrenzt (da Stoffmengen nicht negativ sein können) durch (3.36, 3.37)

$$\xi_{\min} = \max_i \frac{y_i^s}{r_i} \quad \forall r_i < 0 \tag{3.36}$$

[1] Sei A eine $n \times n$-Matrix, die Determinante der Untermatrix A_k, in der jede Spalte und jede Zeile mit Index größer k gestrichen ist, heisst Hauptminor von A.

$$\xi_{\max} = \min_i \frac{y_i^s}{r_i} \quad \forall_i > 0 \tag{3.37}$$

Somit bilden diese Gleichungen die linearen Nebenbedingungen des Equilibrium-Problems im betrachteten Fall.

Das chemische Gleichgewicht

Im chemischen Gleichgewicht nimmt die freie Reaktionsenthalpie G ein Minimum ein. Rein mathematisch folgt, dass die Ableitung von G in jeder Reaktionsrichtung r in diesem Minimum null sein muss. Es gilt somit im Gleichgewicht

$$0 = \sum_{i=0}^{n} r_i^k \mu_i, \quad k = 1, 2, ..., n_\mathrm{r} \tag{3.38}$$

n_r (3.38) ist hierbei die Anzahl an unabhängigen Reaktionen des Systems. Da das System sich nun im Gleichgewicht befindet, ist Gleichung (3.38) äquivalent zu:

$$\Delta G^k = 0 \tag{3.39}$$

Das Problem der Berechnung der Gleichgewichtszusammensetzung ist mathematisch in diesem Fall ein nichtlineares Minimierungsproblem von G mit linearen Nebenbedingungen. Die globalen Minima von Gleichung (3.38) müssen genau die Nullstellen von der Funktion G sein. Dies folgt aus der positiven Semidefinitheit von M und der Tatsache, dass jede Richtung r nicht zu y proportional sein kann, da jedes r mindestens einen positiven und einen negativen Eintrag haben muss (eine Reaktion ganz ohne Produkt oder Edukt ist nicht möglich).

Die linearen Nebenbedingungen setzen sich aus der Ausgangszusammensetzung y^0 und den Massen- und Ladungsbilanzen zusammen. Gemischzusammensetzungen, die diese Nebenbedingungen erfüllen, heissen im folgenden *"erlaubte" Zusammensetzungen.* Aus der Linearität dieser Nebenbedingungen folgt, dass man zwei beliebige erlaubte Zusammensetzungen direkt miteinander durch einen Reaktionsweg r verbinden kann, wobei die Gemischzusammensetzung entlang dieses Weges stets innerhalb der Nebenbedinungen bleibt und somit "erlaubt" ist. Man nennt dies eine *konvexe* Lösungsmenge ([15], S. 157).

Nach diesen Vorbemerkungen folgt der eigentliche Beweis der eindeutigen Lösbarkeit des Equilibrium-Problems nach WELTIN:

Für jede chemische Reaktion sind die freie Reaktionsenthalpie G und ihre erste Ableitung hinsichtlich der Reaktionslaufzahl ξ stetige Funktionen innerhalb des chemisch erlaubten Intervalls (alle Stoffmengenanteile sind positiv).

Nimmt man nun an, es existieren (mindestens) *zwei* Minima y^1 und y^2 der Funktion G, dann muss — wegen der Stetigkeit von G und der ersten Ableitung — zwischen

den Minima y^1 und y^2 mindestens eine erlaubte Zusammensetzung als Maximum liegen. In diesem Maximum wäre die Krümmung, die zweite Ableitung, in allen Richtungen r negativ — sonst wäre an dieser Stelle kein Maximum, da die notwendige Bedingung für Extremwerte ($\nabla f(x) = 0$) ([14], 278) nicht erfüllt ist. Dies ist allerdings nicht möglich, da M positiv semidefinit ist. Daraus folgt, dass die Annahme falsch ist, kein Maximum existiert und ergo nur *ein* Minimum existieren kann. Es kann nur *eine* Gleichgewichtskomposition existieren und **das Equilibrium-Problem ist eindeutig lösbar.**

Bereits an dieser Stelle muß gesagt werden: Will man das Equilibrium-Problem lösen auf diese Weise, ist ein globales Minimum von Gleichung (3.25) zu errechnen. Leider hat man es hier mit einem hochgradig nichtlinearen System zu tun, welches nur sehr schwierig global zu lösen ist. Betrachtet man nur ΔG^0, bekommt man ein lineares Optimierungsproblem mit linearen Nebenbedingungen (3.40), welches mit der bewährten Simplexmethode ([18], S. 262) zu lösen wäre

$$\Delta G^0 = \sum_{i=1}^{n} n_i \mu_i^0 \longrightarrow \min \tag{3.40}$$

Allerdings sind für diesen Ansatz zusätzliche Informationen nötig, beispielsweise sämtliche Standardbildungsenthalpien der einzelnen Reaktanden, die im untersuchten allgemeinen und abstrakten Fall dieser Zulassungsarbeit nicht gegeben sind.

3.2.2 Eindeutigkeit der Lösung des Equilibrium-Problems, kinetische Variante

Auch mittels Argumentation durch die chemische Reaktionskinetik und mathematischer Aussagen über gewöhnliche Differentialgleichungen kann die Eindeutigkeit des Equilibrium-Problems bewiesen werden. Nach einer persönlichen Mitteilung von RUDOLPH [19]:

Wir nehmen an, n Reaktionspartner $\mathrm{A}_0, \mathrm{A}_1, ..., \mathrm{A}_n$ sind an dem Reaktionssystem beteiligt mit ihren Anfangskonzentrationen zum Zeitpunkt $t = 0$ $(a_0, a_1, ..., a_{n-1})$.

Nach diesem Zeitpunkt kommen die einzelnen Reaktionen zwischen den Reaktanden in Gange, die für jede Spezies durch eine gewöhnliche Differentialgleichung erster Ordnung beschrieben werden kann. Vereinfacht dargestellt erhält man folgendes System (3.41)

$$\frac{\mathrm{d}[A_i]}{dt} = K \cdot C \qquad i = 0, ..., n-1 \tag{3.41}$$

$K \cdot C$ besteht aus einer Kombination aus Geschwindigkeitskonstanten und Konzentrationen einzelner Reaktionspartner, zu denen der Reaktand A_i abreagiert oder die an der Bildung von A_i beteiligt sind. Wie $K \cdot C$ genau aussieht, ist im folgenden nicht von Relevanz. Die Lösung des Konzentration-Zeit-Verlaufs jedes Reaktanden erhält man

durch exakte Integration des Systems von n gewöhnlichen Differentialgleichungen erster Ordnung (3.41). Dabei werden n Integrationskonstanten erzeugt, welche durch die n Anfangskonzentrationen eindeutig bestimmt werden. Allerdings ist es — zumindest theoretisch — möglich, dass keine Lösung existiert. Wir gehen deshalb davon aus, dass sich immer ein Gleichgewicht einstellen wird, was durchaus nach einem beliebigen Zeitraum als vernünftig anzunehmen ist. Nach dem Satz von Picard-Lindelöf zur Existenz und Eindeutigkeit der Lösung gewöhnlicher Differentialgleichungen ([33], S. 68) gilt

Satz von Picard-Lindelöf, qualitative Fassung: Ist D eine offene Teilmenge des $\Re^{1+N}$ und ist $f : D \longrightarrow \Re^N$ stetig und bezüglich x Lipschitz-stetig, so besitzt jedes der Anfangswertprobleme

$$f'(x) = f(t, x), \quad x(t_0) = x_0, \quad (t_0, x_0) \in D$$

eine eindeutig bestimmte (lokale) Lösung.

Die Funktionen (3.41) sind trivialerweise stetig und auf einer offenen Teilmenge definiert. Außerdem sind sie beschränkt, da jede Konzentration endlich sein muss. Beschränkte und stetige Funktionen sind stets Lipschitz-stetig. Somit sind die Voraussetzungen des Satzes von Picard-Lindelöf erfüllt und eine Lösung immer eindeutig.

RUDOLPH argumentiert nun, dass es somit zu jedem Zeitpunkt (insbesondere ab dem Zeitpunkt der Gleichgewichtseinstellung) immer nur *eine einzige mögliche Konzentrationsverteilung* geben kann und diese daher immer die physikalisch Richtige sein muß und zu positiven Konzentrationen führen, "wenn man akzeptiert, dass die formalkinetische mathematische Beschreibung den Sachverhalt physikalisch korrekt wiederspiegelt" [19].

Es muss gesagt werden, dass die Lösung des Equilibrium-Problems durch diese Methode auch möglich ist. Dieses *kinetische* Lösungsverfahren hätte allerdings völlig neue prinzipielle mathematische Hintergründe und besitzt wie das thermodynamische Lösungsverfahren dieser Arbeit Vor- und Nachteile. Hauptunterschied ist es, dass man zusätzliche Informationen über die einzelnen Geschwindigkeitskonstanten der Hin- und Rückreaktion jedes Gleichgewichts benötigt. Im folgenden wird ein Lösungsverfahren ohne diese Zusatzinformationen entwickelt.

3.3 Methoden zur Lösung des Equilibrium-Problems

3.3.1 Ist eine Linearisierung möglich?

Sehr hilfreich wäre es, das Equilibrium-Problem auf ein einfach zu behandelndes lineares Gleichungssystem zu reduzieren [7]. Durch Logarithmieren der einzelnen Massenwir-

kungsgleichungen ist dies möglich. Da man immer weniger (linear unabhängige) Gleichungen hat als Reaktanden, erhält man ein **unterbestimmtes lineares Gleichungssystem.** Auf diese Weise ist das Equilibrium-Problem nicht eindeutig lösbar, weshalb hier nicht weiter auf diese Methode eingegangen wird. Allerdings können Abhängigkeiten der einzelnen Gleichgewichtskonzentrationen untereinander berechnet werden, die vereinzelt in der Anwendung von Nutzen sein können [7].

3.3.2 Eindimensionaler Fall, n Reaktanden

Das einfachste denkbare Reaktionssystem mit einer beliebigen Zahl an Reaktanden (3.42) ist ein *eindimensionales* System mit nur einer Reaktionsgleichung [2], wie beispielsweise WELTIN zeigt [20]:

$$|\nu_A|\mathrm{A} + |\nu_B|\mathrm{B} + ... \rightleftharpoons |\nu_P|\mathrm{P} + |\nu_Q|\mathrm{Q} + ... \tag{3.42}$$

Für (3.42) gilt die Gleichgewichtskonstante K_C und die positiven Anfangskonzentrationen a_0, b_0, ...

Für das zu untersuchende eindimensionale Equilibrium-Problem definiert man sich eine Reaktionslaufzahl ξ als Variable. Zu jedem Zeitpunkt der Reaktion gilt für die Edukte

$$\begin{aligned} c(\mathrm{A}) &= a_0 - |\nu_A|\xi && (3.43)\\ c(\mathrm{B}) &= b_0 - |\nu_B|\xi && (3.44)\\ &... \end{aligned}$$

und für die Produkte

$$\begin{aligned} c(\mathrm{P}) &= p_0 + |\nu_P|\xi && (3.45)\\ c(\mathrm{Q}) &= q_0 + |\nu_Q|\xi && (3.46)\\ &... \end{aligned}$$

Stellt man nun das Massenwirkungsgesetz (3.47) auf, erhält man eine nichtlineare Gleichung mit einer Unbekannten ξ (mit 3.43-3.46):

$$(p_0 + |\nu_P|\xi)^{|\nu_P|}(q_0 + |\nu_Q|\xi)^{|\nu_Q|}..(a_0 - |\nu_A|\xi)^{-|\nu_A|}(b_0 - |\nu_B|\xi)^{-|\nu_B|}... - K_\mathrm{C} = 0 \tag{3.47}$$

Um eine eindeutige Lösung zu erhalten, muss der mögliche Bereich von ξ eingeschränkt werden unter den physikalisch einzig sinnvollen Nebenbedingungen (3.48):

$$c(\mathrm{A}) > 0; c(\mathrm{B}) > 0; ..; c(\mathrm{P}) > 0; c(\mathrm{Q}) > 0; .. \tag{3.48}$$

[2] *m-dimensional* bedeutet hier: m Reaktionen, beliebig viele Reaktanden

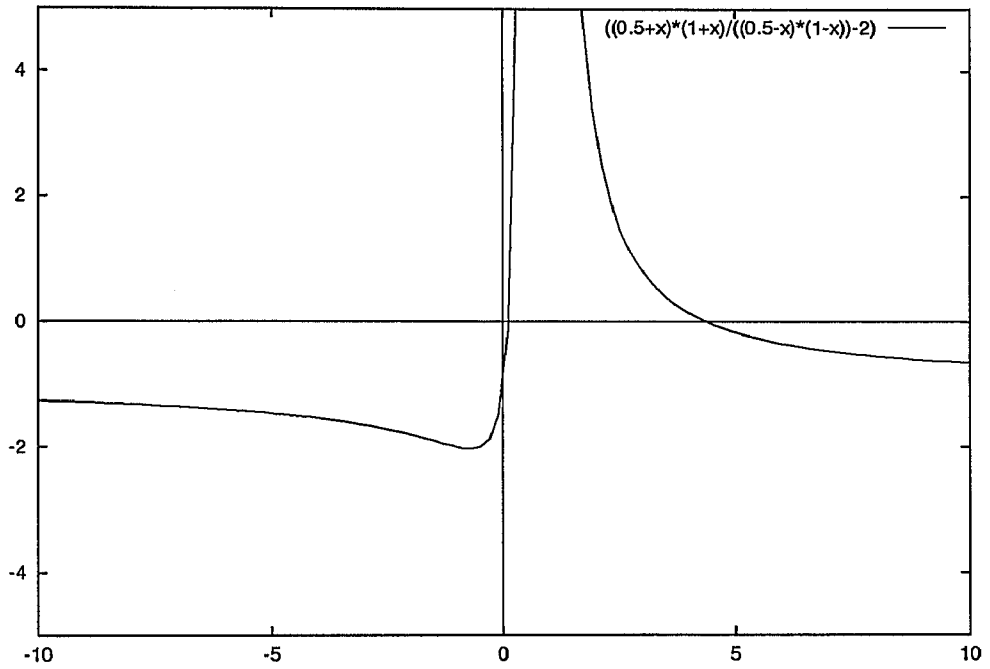

Abbildung 3.1: *"Equilibrium-Funktion" zum System (3.51, 3.52) — wie die Abbildung zeigt, existieren zwei Nullstellen. Nur eine davon erfüllt die Nebenbedingungen.*

Dabei ist das maximale ξ natürlicherweise

$$\xi_{\max} = \min(\frac{a_0}{|\nu_A|}, \frac{b_0}{|\nu_B|}, ...) \tag{3.49}$$

da sonst eine negative Konzentration eines Edukts auftreten würde. Für das minimale ξ gilt:

$$\xi_{\min} = \max(-\frac{p_0}{|\nu_P|}, -\frac{q_0}{|\nu_Q|}, ...) \tag{3.50}$$

sonst bekäme man mindestens eine negative Produktkonzentration.
Da die nichtlineare Gleichung (3.47) überall auf dem erlaubten Intervall (3.49, 3.50) stetig ist und streng monoton fällt (der Zähler wird mit steigendem ξ immer kleiner, der Nenner immer größer), existiert genau eine Lösung im fraglichen Bereich und kann mit mehreren, zur Verfügung stehenden numerischen Mitteln gelöst werden (z. B. Bisektionsverfahren). Die so berechnete Reaktionslaufzahl ξ wird in die Gleichungen (3.43 – 3.46) eingesetzt und die Gleichgewichtskonzentrationen berechnet.

Zur Veranschaulichung dient beispielhaft Abbildung 3.1 des folgenden Equilibrium-Problems (3.51, 3.52):

$$\mathrm{A} + \mathrm{B} \rightleftharpoons \mathrm{C} + \mathrm{D} \tag{3.51}$$

$$a_0 = c_0 = 1; \quad b_0 = d_0 = 0.5; \quad K_0 = 2 \tag{3.52}$$

Anschaulich wird, dass zwar zwei Nullstellen existieren, aber nur eine der Nebenbedingung $-0,5 \leq \xi \leq 0,5$ gerecht wird.

3.3.3 Zwei Gleichungen, ein gemeinsamer Reaktionspartner

Ein Reaktionsystem aus 2 Gleichungen mit genau einem gemeinsamen Reaktionspartner lässt sich ähnlich wie im obigen Fall (als erste Verallgemeinerung) lösen (aus [21] nach WELTIN). Eindrucksvoll einfach ist hier wieder *die Eindeutigkeit der Lösung einzusehen.* Gegeben ist folgendes System (3.53, 3.54)

$$\mathrm{A} + b\mathrm{B} + ... \rightleftharpoons p\mathrm{P} + q\mathrm{Q} + ... \tag{3.53}$$

$$\mathrm{A} + c\mathrm{C} + ... \rightleftharpoons u\mathrm{U} + v\mathrm{V} + ... \tag{3.54}$$

Mit den Gleichgewichtskonstanten K_1 und K_2 und den Anfangskonzentrationen a_0, b_0, c_0, ...

Die Stöchiometriekoeffizienten wurden aus rechnerischen Gründen mit dem Kehrwert des Stöchiometriekoeffizienten des gemeinsamen Reaktanden A normiert.

Es sei nun z eine der Reaktionslaufzahl des Gesamtprozesses entsprechende Zahl. Sinnvollerweise setzt man z gleich der Konzentration der gemeinsamen Komponente A. Des weiteren seien x und y die Reaktionslaufzahlen der ersten und zweiten Reaktion. Damit ergeben sich folgende Konzentrationsgleichungen(3.55-3.61):

$$c(\mathrm{A}) = a_0 - x - y =: z \tag{3.55}$$

$$c(\mathrm{B}) = b_0 - bx \tag{3.56}$$

$$c(\mathrm{C}) = c_0 - cy \tag{3.57}$$

$$\cdots$$

$$c(\mathrm{P}) = p_0 + px \tag{3.58}$$

$$c(\mathrm{Q}) = Q_0 + qx \tag{3.59}$$

$$c(\mathrm{U}) = u_0 + uy \tag{3.60}$$

$$c(\mathrm{V}) = v_0 + vy \tag{3.61}$$

$$\cdots$$

Mit der Aufstellung der zwei Massenwirkungsgesetze und einer einfachen Umformung $(\cdot z)$ ergibt sich im Gleichgewicht:

$$(p_0 + px)^p (q_0 + qx)^p (b_0 - bx)^{-b} \cdot ... \cdot = zK_1 \tag{3.62}$$

$$(u_0 + uy)^u (v_0 + vy)^v (c_0 - cy)^{-c} \cdot ... \cdot = zK_2 \tag{3.63}$$

Man kann diese Gleichungen (3.62, 3.63) als Massenwirkungsgesetze von Reaktionen mit der *effektiven Gleichgewichtskonstante* zK_1 bzw. zK_2 verstehen. Als dritte Gleichung gilt:

$$z + x + y = a_0 \tag{3.64}$$

Diese drei Gleichungen (3.62-3.64) bestimmen das Gleichgewicht mit den drei Variablen x, y und z. Um sie möglichst effizient zu lösen, löst man besser die Hilfsfunktion (3.65)

$$f(z) = z + x(z) + y(z) \tag{3.65}$$

$x(z)$ und $y(z)$ sind die Lösungen von (3.62) und (3.63). Da $f(z)$ nur noch von der Variablen z abhängt, läßt sich das Problem auf eine einfache Nullstellensuche in einer Variablen reduzieren.

Da $f(z)$ durchaus mehrere Nullstellen haben kann und nur eine physikalisch sinnvoll ist (siehe auch Abschnitt 3.2), ist es notwendig, den Definitionsbereich vorher einzuschränken, was hier sehr gut möglich ist.

Aus der trivialen Nebenbedingung, dass alle Konzentrationen zu jeder Zeit, d. h. bei jeder gültigen Reaktionslaufzahl, grösser gleich null sein müssen, gilt für die Funktion $x(z)$ der ersten Reaktion:

$$x_l := \max\left(\frac{-p_0}{p}, \frac{-q_0}{q}, \ldots\right) \leq x \leq \min\left(\frac{b_0}{b}, \ldots\right) =: x_h \tag{3.66}$$

und für $y(z)$ gilt analog:

$$y_l := \max\left(\frac{-u_0}{u}, \frac{-v_0}{v}, \ldots\right) \leq x \leq \min\left(\frac{c_0}{c}, \ldots\right) =: y_h \tag{3.67}$$

Zwei wichtige Eigenschaften folgen nun für $x(z)$ und $y(z)$ aus den Gleichungen (3.62) und (3.63): erhöht man z, erhöht sich die effektive Gleichgewichtskonstante zK_1 bzw. zK_2. Das Gleichgewicht verschiebt sich in Richtung der Produkte — die Reaktion schreitet weiter fort, was gleichbedeutend mit einer Erhöhung der Reaktionslaufzahl $x(z)$ bzw. $y(z)$ ist. $x(z)$ und $y(z)$ sind folglich streng monoton steigend auf ihrem Definitionsbereich (3.66, 3.67).
Für die Grenzen der Gesamtreaktionslaufzahl z gilt:

$$z_l = a_0 - x_h - y_h \tag{3.68}$$

oder

$$z_l = 0 \tag{3.69}$$

falls z_l kleiner null sein sollte (was zu einer negativen Konzentration von A führen würde).

$$z_h = a_0 - x_l - y_l \tag{3.70}$$

Setzt man z_l und z_l in Gleichung (3.65) ein, ergibt sich:

$$f(z_l) < 0 \tag{3.71}$$

$$f(z_h) > 0 \tag{3.72}$$

Zwischen diesen Grenzen ist $f(z)$ (als Summe von $x(z)$ und $y(z)$) streng monoton steigend und stetig. Nach dem Zwischenwertsatz ([14], S. 94) existiert für $f(z)$ genau eine, somit eindeutige, Nullstelle z_{eq} im erlaubten Intervall.
Die Berechnung von z_{eq} kann zum Beispiel mit dem Bisektionsverfahren (Kapitel 4.1.1) effizient durchgeführt werden. Mit Gleichungen (3.55-3.61) ergeben sich die gewünschten Gleichgewichtskonzentrationen — als **eindeutige Lösung des vorgestellten Equilibrium-Problems.**

Die dargestellte Methode kann um eine beliebige Anzahl an Reaktionen erweitert werden — als einzige Bedingung bleibt, dass *genau ein Reaktionspartner in allen Gleichgewichten gleich sein muss.* Für diesen eher speziellen Fall stellt die Methode von WELTIN [21] zwar ein sehr effizientes Verfahren dar. Da aber dieser Fall in der Praxis nur selten auftauchen dürfte und somit wegen fehlender Allgemeingültigkeit als Problemlösung nicht in Frage kommt, wird die C++-Implementierung nicht weiter beschrieben.

3.3.4 Allgemeine Lösung des Equilibrium-Problems

Es wurden in den vorhergehenden Abschnitten einige Spezialfälle des Equilibrium-Problems untersucht. Wünschenswert ist allerdings eine allgemeine Lösung, die dazu noch in akzeptabler Rechenzeit (erstrebenswert wäre weniger als 5 s — mit heutigen Computerstandards) berechnet werden kann.

Formulierung der Eingangsdaten und Eigenschaften des Systems

Gegeben sind als Eingangsdaten:

- eine $m \times n$ *Stöchiometriematrix* H eines beliebigen Reaktionssystems aus m Reaktionen und n Reaktanden. Ihre Elemente H_{ij} stellen Stöchiometriekoeffizienten dar. Die Zeilen entsprechen den Gleichgewichtsreaktionen, die Spalten den beteiligten Reaktionspartnern:

$$\begin{pmatrix} a_{00} & a_{01} & \cdots & \cdots & a_{1(n-1)} \\ a_{10} & a_{11} & \cdots & \cdots & a_{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{(m-1)0} & a_{(m-1)1} & \cdots & \cdots & a_{(m-1)(n-1)} \end{pmatrix} =: H$$

  Konventionsgemäß haben die Koeffizienten der Edukte negative und die der Produkte positive Vorzeichen.

- n Anfangskonzentrationen $[A_0]_0, [A_1]_0, [A_2]_0, ...$ der Reaktanden $A_0, A_1, A_2, ...$ (das Verfahren funktioniert auch, falls Anfangspartialdrücke oder -molenbrüche als Eingangsdaten gegeben sind — im folgenden wird nur K_C betrachtet)
- m Gleichgewichtskonstanten $K_0, K_1, ..., K_{m-1}$ der Gleichgewichtsreaktionen

Zu berechnen sind die Gleichgewichtskonzentrationen $[A_0]_{eq}, [A_1]_{eq}, ...$ Die Lösung kann mit oder ohne Hilfe sogenannter *Bilanzgleichungen* (Atombilanzen, Standardreaktionsenthalpien, Massenerhaltungsgleichungen, ...) gesucht werden.
Für das Reaktionssystem müssen **einige Eigenschaften** gelten:

- es dürfen keine unmöglichen Reaktionen im Reaktionssystem enthalten sein
- jede Zeile der Stöchiometriematrix muss einer Gleichgewichtsreaktion entsprechen. Man nimmt an, dass irreversible Reaktionen zu diesem Zeitpunkt bereits abgelaufen sind und die Anfangskonzentrationen dementsprechend abgeändert sind. Außerdem dürfen im Laufe der Einstellung des Gleichgewichts keine irreversiblen Reaktionen erneut ermöglicht werden. Irreversible Reaktionen werden von dem zu beschreibenden Verfahren ignoriert.
- es wird angenommen, dass sich alle Gleichgewichte vollständig einstellen können — formal heißt das, dass zum Berechnungszeitpunkt unendlich viel Zeit vergangen ist.

Allgemeine Lösung ohne Bilanzgleichungen

Man bekommt ohne Bilanzgleichungen (Atombilanzen, Massenbilanzen, Reaktionsenthalpien, ...) dasselbe zu lösende Gleichungssystem wie mit Bilanzgleichungen. Die Nebenbedingungen sind allerdings verschieden — wie in den folgenden zwei Abschnitten aufgezeigt werden soll.

Für unsere Lösung sollen keine Bilanzgleichungen verwendet werden — da in dem nun angenommenen allgemeinen Ansatz die Reaktanden abstrakt mit A_i bezeichnet werden und prinzipiell keine weiteren Informationen als mögliche zusätzliche Eingangsdaten über ihre chemische Zusammensetzung, ihre molare Masse, ihre Standardbildungsenthalpien, usw. gegeben sind.
Da im allgemeinen Fall eine beliebige Anzahl von Reaktionen im System enthalten sein kann, führt man für jede i. Reaktion eine eigene Reaktionslaufzahl ξ_i $(i = 0, ..., m-1)$ ein.

Zu jedem Zeitpunkt t (insbesondere nach Einstellung des chemischen Gleichge-

wichts) der Gesamtreaktion gilt für die Konzentrationen der Reaktanden (3.73)

$$[\mathrm{A}_j]_t = [\mathrm{A}_j]_0 + \sum_{i=0}^{m-1} a_{ij} \cdot \xi_i \tag{3.73}$$

Formuliert man nun die einzelnen Massenwirkungsgesetze (3.74-3.76), erhält man folgendes nichtlineares Gleichungssystem:

$$\left(\prod_{j=0}^{n-1} \left[[\mathrm{A}_j]_0 + \sum_{i=0}^{m-1} a_{ij} \cdot \xi_i \right]^{a_{0j}} \right) - K_0 = 0 \tag{3.74}$$

$$\left(\prod_{j=0}^{n-1} \left[[\mathrm{A}_j]_0 + \sum_{i=0}^{m-1} a_{ij} \cdot \xi_i \right]^{a_{1j}} \right) - K_1 = 0 \tag{3.75}$$

$$\ldots$$

$$\left(\prod_{j=0}^{n-1} \left[[\mathrm{A}_j]_0 + \sum_{i=0}^{m-1} a_{ij} \cdot \xi_i \right]^{a_{mj}} \right) - K_{m-1} = 0 \tag{3.76}$$

unter den (linearen) Nebenbedingungen (3.77) (da physikalisch nur positive Konzentrationen erlaubt sind):

$$[\mathrm{A}_j]_t = [\mathrm{A}_j]_0 + \sum_{i=0}^{m-1} a_{ij} \cdot \xi_i \geq 0 \quad j = 0, ..., n-1 \tag{3.77}$$

Berechnet man nun den Lösungsvektor $\xi = (\xi_0, \xi_1, ..., \xi_{m-1})$ aus Reaktionslaufzahlen, kann man mit Gleichung (3.77) die Gleichgewichtskonzentrationen ausrechnen.

Ein **direktes, allgemeines und dazu mathematisch exaktes Lösungsverfahren** kann für ein Gleichungssystem dieser Komplexität nicht existieren (zur Diskussion dieses grundlegenden numerischen Problems siehe Abschnitt 4.2).

Allerdings bleibt man nicht völlig chancenlos. Zur Lösung nichtlinearer Gleichungssysteme stehen mächtige iterative Verfahren wie das Newton-Raphson-Verfahren mit Schrittweitensteuerung und das Broydenverfahren zur Verfügung (siehe Kapitel 4.2.2-4.2.4).

Das Hauptproblem allerdings ist, dass man bei einem beliebigen Startvektor im allgemeinen im Vorneherein nicht weiß, ob:

- die Iteration zur Konvergenz führen muss
- numerische Instabilitäten auftreten werden
- die berechnete Lösung auch die physikalisch sinnvolle Lösung sein wird, also die Nebenbedingungen (3.77) erfüllt
- der Startwert nahe der richtigen Lösung liegt oder vielleicht sehr weit entfernt

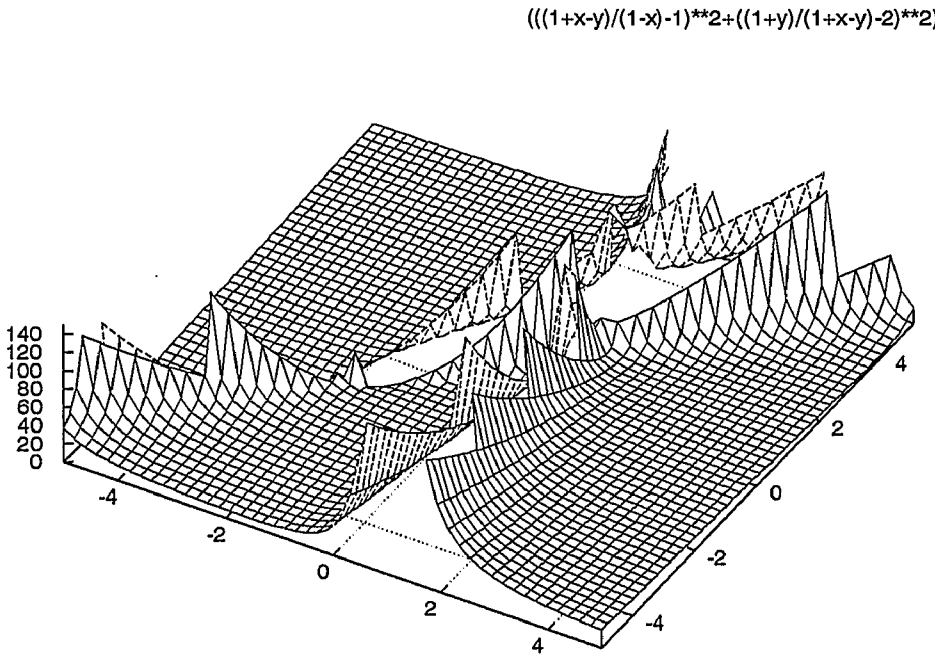

Abbildung 3.2: *Die Nullstellenumgebung des zweidimensionalen Equilibrium-Problems (3.78-3.81). Deutlich wird die Komplexität einer Nullstellensuche. (Grundebene ist die (x,y)-Ebene. Die dritte Achse ist die f(x,y)-Achse). Die Nullstelle befindet sich in (0.25/0.5).*

Abbildung (3.2) illustriert die Komplexität der zu untersuchenden Funktionen des Equilibrium-Problems für einen einfachen zweidimensionalen und gekoppelten Fall (3.78-3.81)

$$A \rightleftharpoons B \tag{3.78}$$

$$B \rightleftharpoons C \tag{3.79}$$

$$a_0 = b_0 = c_0 = 1 \tag{3.80}$$

$$K_0 = 1,\ K_1 = 2 \tag{3.81}$$

Um diese zweidimensionale Funktion darstellen zu können, wurde die zwei Einträge des Funktionsvektors quadriert und addiert. Eine Nullstelle dieser neuen Funktion muss eine Nullstelle beider Funktionen sein.
Es wird anschaulich klar, dass selbst bei diesem einfachen Reaktionssystem nicht jeder Startwert einer iterativen Nullstellensuche in das richtige Gebiet der einzigen, richtigen Nullstelle führen wird.

Zumindest kann dieses Problem programmiertechnisch durch eine Lösungsrouti-

ne teilgelöst werden, die sinnvolle Startwerte durch einen Zufallsalgorithmus erzeugt (siehe Abschnitt 5.1.3) und stochastisch gesehen nach einigen Versuchen mit großer Wahrscheinlichkeit einen guten Startwert erzeugt. Nachteilig bleibt allerdings, dass so keine endliche Rechenzeit garantiert werden kann. Ein vernünftiges Abbruchkriterium muss vorher definiert werden. Möglichkeiten und Probleme in der Anwendung neuartiger, sogenannter *„genetischer"* Algorithmen werden in Abschnitt 5.1.4 andiskutiert.

Allgemeine Lösung mit Bilanzgleichungen

Um das Problem **der fehlenden Bilanzgleichungen** in dem hier untersuchten Fall als Nebenbedingungen genauer zu erläutern, wird kurz etwas von der abstrakten Allgemeinheit weggegangen und angenommen, *die chemischen Formeln der Reaktanden seien bekannt.* Dann ist es möglich — bei m Reaktionen — einige Atombilanzgleichungen als mathematische Nebenbedingungen zu formulieren.
Beispiel: Das Gleichgewichtsreaktionssystem (3.82, 3.83)

$$CH_4 + H_2O \rightleftharpoons CO + 3H_2 \tag{3.82}$$

$$CH_4 + 2H_2O \rightleftharpoons CO_2 + 4H_2 \tag{3.83}$$

hat folgende zusätzliche Nebenbedingungen (3.84-3.86) durch Aufstellung der einzelnen Atombilanzen

$$4[CH_4] + 2[H_2O] + 2[H_2] = n_H \tag{3.84}$$

$$[CH_4] + [CO] + [CO_2] = n_C \tag{3.85}$$

$$[H_2O] + [CO] + 2[CO_2] = n_O \tag{3.86}$$

Diese Gleichungen gelten zu jeder Zeit des Reaktionsfortgangs. Die Atombilanzkonstanten n_H, n_C, n_O können einfach aus den Anfangskonzentrationen berechnet werden.

Da man jetzt drei zusätzliche Gleichungen zur Verfügung hat, die das Equilibrium-Problem ergänzen, kann man die Methode der **Lagrangeschen Multiplikatoren** [22] einsetzen, um die Reaktionslaufzahlen und schließlich die Gleichgewichtskonzentrationen zu berechnen.

Hat man dagegen ein abstraktes System — nachteiligerweise ohne chemische Formeln der Reaktionspartner, ohne Standardbildungsenthalpien oder Molmassen:

$$A_0 + A_1 \rightleftharpoons A_2 + 3A_3 \tag{3.87}$$

$$A_0 + 2A_1 \rightleftharpoons A_4 + 4A_3 \tag{3.88}$$

so kann man keine zusätzlichen Gleichungen wie (3.75-3.77) aufstellen, die als sinnvolle Nebenbedingungen dienen könnten. Es bleibt als Nebenbedingung nur noch die Tatsache, dass Konzentrationen nicht negativ sein können. Mathematisch bleiben lediglich

die linearen *Un*gleichungen (3.89)

$$[\mathrm{A}_i] > 0 \quad i = 0, ..., m \tag{3.89}$$

Somit kommt die Methode der Lagrangeschen Multiplikatoren nicht mehr in Betracht, bei welchem notwendigerweise Gleichungen als Nebenbedingungen benötigt werden.

Kapitel 4

Numerische Lösungsverfahren zur Nullstellenbestimmung

Im folgenden Kapitel werden Verfahren vorgestellt, die die numerische Berechnung von Nullstellen eindimensionaler und mehrdimensionaler reeller Funktionen ermöglichen. Wie in Kapitel 2 erläutert, benötigt man diese Algorithmen, um das Equilibrium-Problem computergestützt zu lösen.

Eingangs werden einfache eindimensionale Verfahren erläutert: das Bisektionsverfahren und das eindimensionale Newton-Raphson-Verfahren, letzteres motiviert das mehrdimensionale Newton-Raphsonverfahren und macht die Herleitung einsichtiger. Eindimensionale Verfahren reichen für eine allgemeine Lösung allerdings nicht aus.

Das mehrdimensionale Newton-Raphson-Verfahren wird darauf mit einer Schrittweitensteuerung optimiert. Auf diesem "gedämpften Newtonverfahren" baut das am weitesten fortgeschrittene **Broydenverfahren** auf. Das Broydenverfahren ist schließlich das Verfahren der Wahl, um das allgemeine Equilibrium-Problem zu lösen.

4.1 Eindimensionale Verfahren

Zur Lösung des eindimensionalen Equilibrium-Problems mit nur einer Gleichung bieten sich zahlreiche Verfahren zur Nullstellensuche an: das Bisektions- und Newtonverfahren wird im folgenden besprochen. Beide sind für das Verständnis mehrdimensionaler Methoden gut geeignet. WELTIN [21] beispielsweise verwendet das Bisektionsverfahren.

4.1.1 Das Bisektionsverfahren

Ein sehr robustes und elementares Verfahren zur Lösung von nichtlinearen Gleichungen in einer Variablen ist das *Bisektionsverfahren* (auch Intervallschachtelung) (für weitere Details siehe FAIRES, BURDEN [23], S. 30-36).

Das Bisektionsverfahren konvergiert *immer* gegen eine Nullstelle, sofern die zu

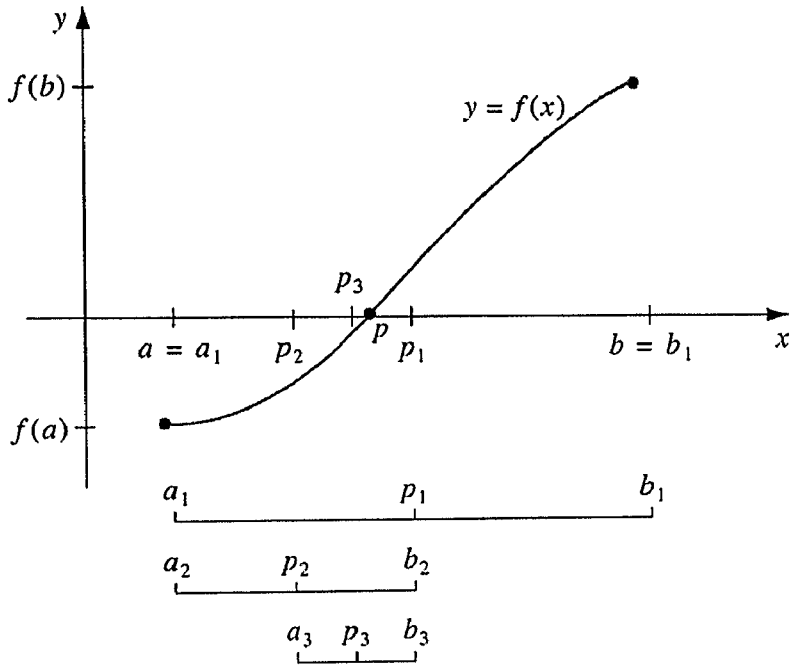

Abbildung 4.1: *Das Bisektionsverfahren in den ersten Schritten (aus [23])*

untersuchende Funktion $f : [a, b] \rightarrow \Re$ folgendes erfüllt:
- f ist stetig auf dem Intervall $[a, b]$
- a und b haben unterschiedliche Vorzeichen

Das Vorgehen ist recht einfach (siehe Abbildung (4.1)): p sei der Mittelpunkt von [a,b]. Nun wertet man Funktion in p aus und untersucht das Vorzeichen. Nun vertauscht man a oder b mit p, je nachdem, ob $f(a)$ oder $f(b)$ dasselbe Vorzeichen wie $f(p)$ hat. Formalisiert heisst das:

Ein Intervall $[a_{i+1}, b_{i+1}]$, das eine Approximation einer Wurzel von $f(x) = 0$ enthält, wird aus einem Intervall $[a_i, b_i]$, das die Wurzel enthält, konstruiert

$$p_{i+1} = a_i + \frac{b_i - a_i}{2}$$

Dann gilt $a_{i+1} = a_i$ und $b_{i+1} = p_{i+1}$, wenn $f(a_i) \cdot f(p_{i+1}) < 0$ ist (d.h. unterschiedliche Vorzeichen haben), und andernfalls $a_{i+1} = p_{i+1}$ und $b_{i+1} = b_i$ (aus [23], S. 31).

Die Nullstellengenauigkeit beträgt nach n Iterationen:

$$\epsilon = \frac{b - a}{2^n} \tag{4.1}$$

Trotz seiner konzeptionellen Klarheit hat das Verfahren erhebliche Nachteile. Es konvergiert sehr langsam (im Vergleich z.B. zum Newton- und Sekantenverfahren) und es ist außerdem für mehrdimensionale Probleme nicht einsetzbar.

Zur Lösung eindimensionaler Equilibrium-Probleme oder zur Lösung im Falle genau eines gemeinsamen Reaktionspartners bei beliebig vielen Reaktionen (Kapitel 3.3.2 und 3.3.3), stellt es aber ein sehr robustes und verständliches Verfahren dar WELTIN [21].

4.1.2 Das Newton-Verfahren und grafische Interpretation

Die eindimensionale Nullstellensuche durch das Newton-Verfahren besitzt eine geometrisch anschauliche Darstellung (aus [23] S. 42-47) (siehe Abbildung 4.2). So wird die Nullstelle einer den Kurvenverlauf von f approximierenden Geraden gesucht und auf diese Weise die Lösung von f(x)=0 approximiert. Auf ein mehrdimensionales Newton-Raphson-Verfahren wird später eingegangen.

Die Gerade, die das Kurvenbild einer Funktion in einem Punkt am besten approximiert, ist die Tangente in diesem Punkt.

Angenommen, p_0 sei eine Startnäherung der Nullstelle p der Gleichung $f(x) = 0$. Weiter muss die Differenzierbarkeit in einer Umgebung von p_0 angenommen werden. Die Tangentengleichung (4.2) lautet

$$y - f(p_0) = f'(p_0)(x - p_0) \tag{4.2}$$

Schneidet man die Tangente nun mit der x-Achse ($y = 0$), erhält man die erste Näherung p_1 von p (4.3)

$$0 - f(p_0) = f'(p_0)(p_1 - p_0) \quad \Leftrightarrow \quad p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} \tag{4.3}$$

daraus folgt:

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} \tag{4.4}$$

vorausgesetzt, dass $f'(p_0) \neq 0$ ist. Das führt auf die Iterationsvorschrift des Newton-Verfahrens (4.5).

Newton-Verfahrens:

$$p_{i+1} = p_i - \frac{f(p_i)}{f'(p_i)} \tag{4.5}$$

mit einem geeigneten Startwert p_0

Befindet sich allerdings der Startpunkt p_0 weit von der Nullstelle entfernt, kann das Newton-Rahpson-Verfahren auch sehr ungenaue oder unbedeutende iterative Näherungen liefern. In welche der eventuell sehr zahlreichen Nullstellen das Verfahren konvergiert, kann zusätzlich nicht allgemein vorhergesagt werden.

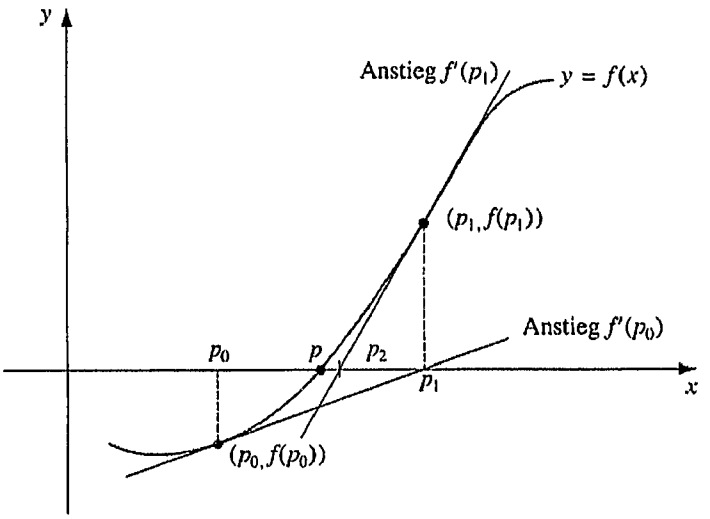

Abbildung 4.2: *Das eindimensionale Newton-Verfahren in den ersten beiden Schritten (aus [23])*

Konvergenz des Newton-Verfahrens

Die Konvergenz des Newton-Verfahrens ist im Vergleich zu anderen Verfahren (z.B. Bisektionsverfahren) sehr gut:

Angenommen, p sei die Lösung von $f(x) = 0$ und f'' existiere auf einem Intervall, das sowohl p als auch die Näherungslösung p_i enthält. Entwickeln von f in sein Taylorpolynom zweiter Ordnung (4.6) ([15], S. 282) in p_i und Auswerten in $x = p$ ergibt:

$$0 = f(p) \approx f(p_i) + f'(p_i)(p - p_i) + \frac{f''(\xi)}{2}(p - p_i)^2, \tag{4.6}$$

wobei ξ zwischen p_i und p liegt. Folglich gilt für $f'(p_i) \neq 0$

$$p - p_i + \frac{f(p_i)}{f'(p_i)} = -\frac{f''(\xi)}{2f'(p_i)}(p - p_i)^2 \tag{4.7}$$

Da

$$p_{i+1} = p_i - \frac{f(p_i)}{f'(p_i)} \tag{4.8}$$

gilt, folgt daraus

$$p - p_{i+1} = -\frac{f''(\xi)}{2f'(p_i)}(p - p_i)^2 \tag{4.9}$$

Ist nun die zweite Ableitung von f auf einem Intervall über p durch eine obere Schranke $M \in \Re$ beschränkt und p_i liegt innerhalb dieses Intervalls, dann gilt (4.10)

$$|p - p_{i+1}| \leq \frac{M}{2|f'(p_i)|}|p - p_i|^2. \tag{4.10}$$

Somit beträgt der Fehler der $(i+1)$.-Approximation $|p - p_{i+1}|$ ungefähr das Quadrat der i. Approximation. Es ergibt sich **quadratische Konvergenz.**

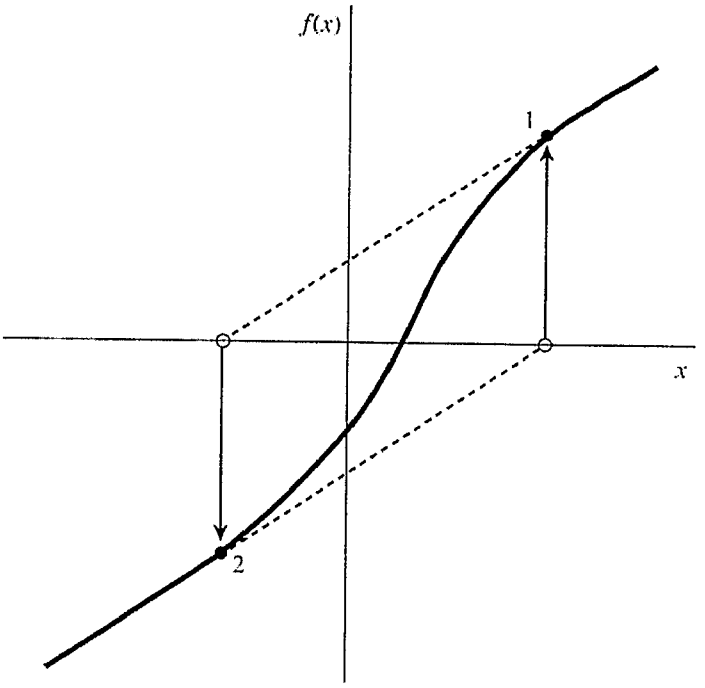

Abbildung 4.3: *Im gezeigten Fall kommt es zu einer Endlosschleife in der Newton-Iteration. Hier muss das Verfahren unterbrochen und ein neuer Startwert verwendet werden (aus [24]).*

Das Newton-Verfahren zeigt sich als gutes Verfahren zur Nullstellensuche, da es lokal um die Nullstelle schnell konvergiert.

Nachteilig ist allerdings, dass das Verfahren nicht bei jedem Startwert konvergent ist. So kann man im Laufe der Iteration ungünstigerweise in einem lokalen Minimum landen oder es kann eine Endlosschleife auftreten (siehe Abbildungen (4.3-4.4)). Deshalb ist es in der praktischen Anwendung nötig, die Iteration nach einer vorher definierten, maximalen Rechenzeit mit einem anderen Startwert zu wiederholen.

4.2 Mehrdimensionale Verfahren

Im folgenden Abschnitt wird diskutiert, wie das nichtlineare System (4.11)

$$F_i(x_0, x_1, x_2, ..., x_{N-1}) = 0 \tag{4.11}$$

N-dimensionaler Funktionen mit den Variablen $x_i, i = 0, 1, ..., N-1$ iterativ zu lösen ist.

4.2.1 Problematik mehrdimensonaler Verfahren

Eine mehrdimensionale Nullstellensuche gilt als "sehr aufwendiges Problem in der heutigen Numerik" ([25], S. 299).

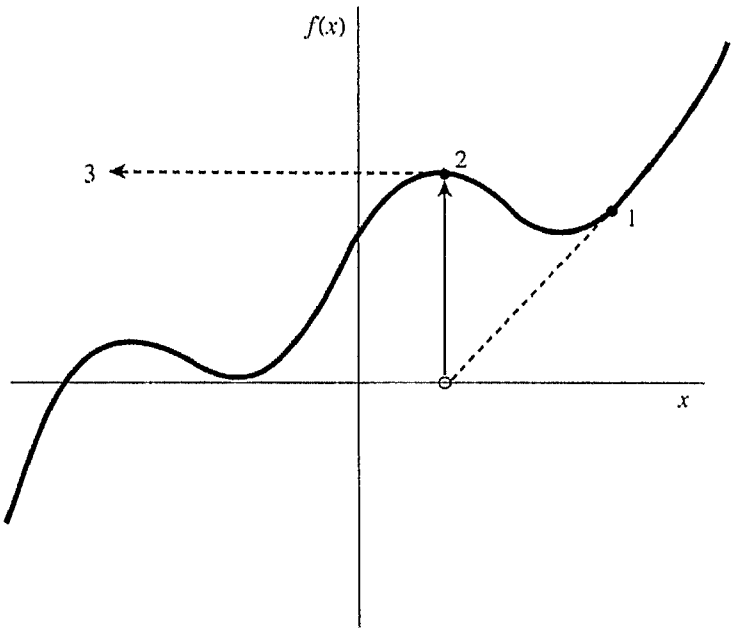

Abbildung 4.4: *Abbruch des Newtonverfahrens in einem lokalen Minimum (aus [24]).*

„There are no good, general methods for solving systems of more than one nonlinear equation. Furthermore there never will be any good, general methods." ([24], S. 383)

Zu den in Kapitel 4.1.2 erläuterten Problemen ergeben sich im mehrdimensionalen Fall weitere Schwierigkeiten für den Anwender.
Als Beispiel diene ein zweidimensionales, nichtlineares System (4.12, 4.13)

$$f(x,y) = 0 \tag{4.12}$$

$$g(x,y) = 0 \tag{4.13}$$

Die Funktionen f und g sind grundsätzlich zwei völlig verschiedene Funktionen, jede von ihnen hat eine Nulllinie, die die (x,y)-Ebene in Gebiete negativer und positiver Funktionswerte teilt. Die gesuchte Lösung liegt in den gemeinsamen Punkten dieser Nulllinien (sofern mindestens eine existiert) (siehe Abbildung 4.4 ([24], S. 384)). Leider haben die Funktionen f und g im allgemeinen *nichts* miteinander zu tun — das heisst, sie müssen weder gleiche Eigenschaften wie Differenzierbarkeit, Integrierbarkeit, Stetigkeit noch Variablen gemeinsam haben. Aus der alleinigen Sicht von f oder g gelten keine besonderen Eigenschaften in eventuellen gemeinsamen Punkten. Um alle Lösungen dieser nichtlinearen Gleichungen zu finden, müsste man die gesamten Nulllinien von f und g untersuchen!

Für Probleme von allgemein n. Dimension, müsste man gemeinsame Punkte von n $(n-1)$-dimensionalen Hyperflächen, die im allgemeinen nichts miteinander zu tun haben, berechnen. Ohne weitere Einsichten ist dies eine meist unlösbare Aufgabe.

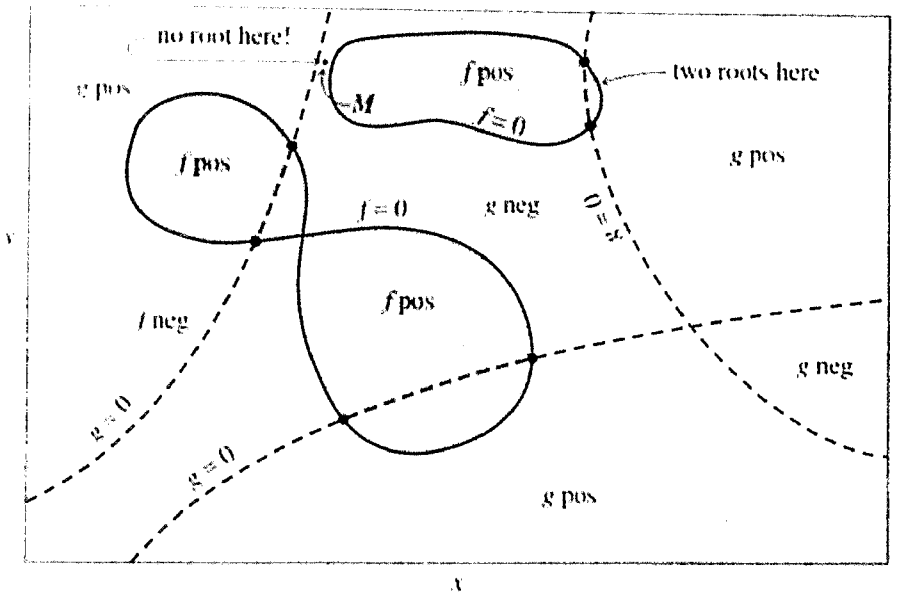

Abbildung 4.5: *Ein klassisches, zweidimensionales Nullstellenproblem. Graphen und Nulllinien zweier unabhängiger, nichtlinearer Funktionen f (durchgezogene Linie) und g (gestrichelte Linie). Lösungen liegen in den hervorgehobenen Punkten, deren Lage und Anzahl im allgemeinen Fall im vorneherein nicht bekannt sind (aus [24]).*

Man benötigt folglich weitere Informationen — etwa über die Eindeutigkeit der Lösung und eine Eingrenzung des Lösungsraums ([24], S. 384). Verfahren, die versuchen, diese Probleme zu lösen, werden in den folgenden Abschnitten vorgestellt.

4.2.2 Das mehrdimensionale Newton-Raphson-Verfahren

Das mehrdimensionale Newton-Raphson-Verfahren konvergiert wie das eindimensionale Verfahren sehr schnell gegen eine Nullstelle. Der Nachteil, dass ein guter Startwert vorgegeben werden muss, bleibt erhalten — Konvergenz kann nur lokal garantiert werden. Eine auf globale Konvergenz erweiterte Variante wird im nächsten Abschnitt vorgestellt.
Die folgende Darstellung leht sich ([24], S. 385) an. Gegeben ist das zu untersuchende m-dimensionale System (4.14)

$$F_i(x_0, x_1, x_2, ..., x_{n-1}) = 0 \quad i = 0, ..., m-1 \tag{4.14}$$

und es sei im folgenden x der Vektor aller x_i und F der Vektor aller m Funktionen F_i. Die Grundidee ist, das nichtlineare Problem durch ein (einfacher zu handhabendes)

lineares Problem zu approximieren. Dazu entwickelt man F_i in seine Taylorreihe (4.15)

$$F_i(x + \delta x) \approx F_i(x) + \sum \frac{\partial F_i}{\partial x_j} \delta x_j \tag{4.15}$$

In Matrixschreibweise bedeutet das (4.16)

$$F(x + \delta x) = F(x) + J \cdot \delta x \tag{4.16}$$

wobei J die *Jacobi-Matrix* mit den partiellen Ableitungen von F darstellt mit den Einträgen

$$J_{ij} = \frac{\partial F_i}{\partial x_j} \tag{4.17}$$

Setzt man nun $F(x + \delta x) = 0$, so erhält man ein *lineares Gleichungssystem*, für dessen Lösung zahlreiche ausgereifte Algorithmen zur Verfügung stehen, beispielsweise das Gauss-Verfahren ([25], S. 94). Aus dem folgendem System (4.18) berechnet man den *Korrekturvektor* δx:

$$F(x + \delta x) = 0 \quad \Leftrightarrow \quad J \cdot \delta x = -F \tag{4.18}$$

Der Korrekturvektor δx wird zum bisherigen Lösungsvektor x addiert

$$x_{\text{neu}} = x_{\text{alt}} + \delta x \tag{4.19}$$

und der Prozess iterativ bis zur Konvergenz fortgesetzt. Vereinfacht hat man folgende Iterationsvorschrift:

Mehrdimensionales Newton-Raphson-Verfahren:

$$x_{i+1} = x_i - J^{-1}(x_i) F(x_i) \tag{4.20}$$

mit einem geeigneten Startvektor x_0

Der Prozess konvergiert wie im eindimensionalen Fall lokal quadratisch. Leider bleiben die Nachteile des eindimensionalen Falls erhalten: es können immer noch Endlosschleifen auftreten, die Ableitung (hier die Jacobi-Matrix) kann null (singulär im Matrix-Fall) werden und die Konvergenz ist nur auf eine kleine Umgebung um die Nullstelle begrenzt ([26], S. 88).

Für die Anwendung unerlässlich ist eine deutliche Vergrösserung des Konvergenzbereichs. Mit dieser Problematik befasst sich der nächste Abschnitt.

4.2.3 Newton-Raphson-Verfahren mit Schrittweitensteuerung

Im folgenden soll ein Verfahren entwickelt werden, das von fast jedem Startpunkt aus in eine Lösung des mehrdimensionalen, nichtlinearen Nullstellenproblems $F(x) = 0$ führt

([24], S. 387). Sinnvoll wäre ein Verfahren, das in jedem Iterationsschritt der Nullstelle auch garantiert näher kommt.

Das folgende Verfahren ([24], S. 388) kombiniert die schnelle Konvergenz des Newton-Raphson-Verfahrens mit einer Strategie der Schrittweitensteuerung.
Betrachtet wird der *Newton-Schritt* (4.21, 4.22)

$$x_{i+1} = x_i + \delta x \tag{4.21}$$

mit

$$\delta x = -J^{-1} \cdot F \tag{4.22}$$

Nun muss dieser Schritt bewertet werden. Sinnvoll wäre, zu verlangen, dass

$$F(x_{i+1})^2 \leq F(x_i)^2, \tag{4.23}$$

gelten muss, da eine Nullstelle von F stets ein Minimum von F^2 ist. Man versucht also im folgenden die Funktion (4.24)

$$f = F \cdot F \tag{4.24}$$

zu minimieren. Allerdings kann man immer noch in lokalen Minima von f landen, die keine Lösung von F darstellen. Eine einfache Minimierungsstrategie wird nicht ausreichen. Es muss versucht werden, globale Minima zu finden. Diese Strategie unterscheidet sich im Wesentlichen dadurch von einer gewöhnlichen Minimierungsstrategie, dass die notwendige Bedingung $\nabla f(x) = 0$ für Extremwerte nicht ausreichen kann, um globale Minima zu finden.

Die Strategie der Nullstellensuche durch das Newton-Raphson-Verfahren beruht daruaf, dass idealerweise der Newton-Schritt automatisch eine *Abstiegsrichtung* für f ist, denn es gilt

$$\nabla f \cdot \delta x = (F \cdot J) \cdot (-J^{-1} \cdot F = -F^2 < 0 \tag{4.25}$$

∇f (4.25) ist der *Gradient* der Funktion f und entspricht dem Steigungsvektor (genauer: der Gradient zeigt in die Richtung des steilsten Anstiegs) von f — der Gradient ist folgendermäßen definiert (4.26):

$$\nabla f = \left(\frac{\mathrm{d}f}{\mathrm{d}x_1}, \frac{\mathrm{d}f}{\mathrm{d}x_2}, ..., \frac{\mathrm{d}f}{\mathrm{d}x_n} \right) \tag{4.26}$$

Nachdem die Richtung, in die die Iteration verlaufen soll, berechnet wurde, bleibt die Schrittweite zu wählen. Hier verfolgt man eine einfache Strategie:

- zuerst versucht man den kompletten Newton-Schritt
- nun prüft man, ob dieser Schritt f wirklich verkleinert hat (vgl. (4.24.))

- wenn nicht, geht man entlang der Newton-Richtung zurück (*sog. backtracking*), bis eine akzeptable Schrittweite gefunden ist. Da die Newton-Richtung immer eine Abstiegsrichtung ist, wird man stets eine akzeptable Schrittweite finden.
- ist der Schritt akzeptabel, berechnet man dort den Gradienten ∇F, eine neue Abstiegsrichtung und iteriert weiter. In einer näheren Umgebung um die Nullstelle erreicht man schließlich quadratische Konvergenz.

Das Verfahren schlägt fehl, falls die Iteration im Laufe des Verfahrens in einem lokalen Minimum landet. In diesem Fall muss man einen neuen Startvektor bestimmen und das Verfahren wiederholen.

In der Praxis wird mit dem Newton-Raphson-Verfahren mit Schrittweitensteuereung eine globale Konvergenz erreicht.

Das Newtonverfahren ist ein sehr gutes Verfahren zur Nullstellensuche. Ein Geschwindigkeitsnachteil ist, dass mehrmals die Jacobimatrix berechnet werden muss. Oft steht diese nicht zur Verfügung. Nachteilig ist auch, dass die Kalkulation der Jacobimatrix viele Funktionsauswertungen benötigt. Falls die Funktionsauswertung aufwenig sein sollte, kann dies das Newtonverfahren in der Praxis deutlich verlangsamen. Sinnvoll wäre es in diesem Fall, die Jacobi-Matrix "billiger" zu approximieren. Der nächste Abschnitt befasst sich mit der Realisierung dieser Approximation.

4.2.4 Das Broydenverfahren — ein multidimensionales Sekantenverfahren

Im eindimensionalen Fall kann die Tangente und damit die Ableitung durch eine Sekante approximiert werden. Im folgenden soll ein mehrdimensionales, dem Newtonverfahren angelehntes, Sekantenverfahren oder auch *Quasi-Newtonverfahren* entwickelt werden — das sogenannte Broydenverfahren. Es benötigt deutlich weniger Funktionsauswertungen als das bisher beschriebene Newton-Raphson-Verfahren, bei etwa gleichen Konvergenzeigenschaften ([24], S. 393).

Es sei nun B eine Approximation an die Jacobi-Matrix J von F. Für den Korrekturvektor $\delta x = x_{i+1} - x_i$ im *i-ten Quasi-Newton-Schritt* gilt:

$$B_i \cdot \delta x = -F_i \tag{4.27}$$

Die Quasi-Newton-Bedingung (4.28) für die Approximationsmatrix B_i (4.27) lautet

$$B_{i+1} \cdot \delta x_i = \delta F_i \tag{4.28}$$

mit $\delta F_i = F_{i+1} - F_i$. Dies ist eine Verallgemeinerung der eindimensionalen Sekantenapproximation an die Ableitung, $\partial F / \partial x$. Allerdings legt Gleichung (4.29) B_{i+1} nicht

eindeutig fest.
In der Praxis wurden viele Möglichkeiten entwickelt, B_{i+1} möglichst sinnvoll zu berechnen. Eine gute Möglichkeit stellt das **Broydenverfahren** dar ([24], S. 393). Es gilt für B_{i+1}[1]:

$$B_{i+1} = B_i + \frac{\delta F_i - B_i \cdot \delta x_i) \otimes \delta x_i}{\delta x_i \cdot \delta x_i} \tag{4.29}$$

Diese Formel basiert auf der Idee, B_{i+1} so zu berechnen und damit B_i minimalst so zu veränderen, dass gerade noch die Sekantenbedingung (4.27) eingehalten wird. Dass (4.29) Gleichung (4.27) erfüllt, ist einfach nachzurechnen.

Für die praktische Berechnung von $\delta x_{i+1} = -B^{-1} \cdot F$ ist es nötig, B_{i+1} zu invertieren. Dies geschieht durch eine QR-Zerlegung von B_{i+1} per Householdertransformation ([26], S. 73). Der Vorteil gegenüber der gebräuchlichen LR-Zerlegung ist es, dass man auf Grund der speziellen Form der Matrix B_{i+1} ihre QR-Zerlegung effizient als Update der QR-Zerlegung von B_i berechnen kann (für Details siehe ([24], S. 101-104)).

Um das Broyden Verfahren zu starten, braucht man jetzt nur noch eine erste Startapproximation B_0 von J_0. Am einfachsten ist es, für B_0 die Einheitsmatrix zu verwenden und mit einigen Iterationsschritten zu verbessern.

Das Broydenverfahren konvergiert beinahe so schnell wie das Newton-Raphson-Verfahren und kann auch mit einer Schrittweitensteuerung analog zu Kapitel (4.2.3) verknüpft werden, um den Konvergenzbereich zu erweitern. Es braucht, wie eingangs erwähnt, weniger Funktionsauswertungen als das Newton-Raphson-Verfahren, womit es zur Lösung des allgemeinen Equilibrium-Problems das Verfahren der Wahl darstellt. Die Auswertung der Equilibrium-Funktionen nämlich als ziemlich aufwendig angesehen.

Das Broydenverfahren versagt allerdings, falls B im Laufe der Iteration singulär oder fast singulär werden sollte, da dann der Korrekturvektor δx nicht mehr berechnet werden kann. Die einfachste Lösung ist es in diesem (nicht seltenen) Fall, einen neuen Startvektor zu wählen und die Iteration zu wiederholen.

Abschließend soll nicht unerwähnt bleiben, dass auch einige, andere (allerdings nicht notwenig schnellere) Algorithmen existieren, die dieses Problem lösen können (zum Beispiel erweiterte Minimierungsalgorithmen [28]).

[1] für eine mathematische Herleitung siehe [27]

Kapitel 5

Objektorientierte Realisierung in C++

Als zentrales mehrdimensionales Verfahren wurde das Broydenverfahren mit Schrittweitensteuerung zur Lösung des Equilibrium-Problems wegen seiner Allgemeingültigkeit realisiert.

5.1 Implementierung der allgemeinen Lösung in C++

Im folgenden soll das bereits bekannte "Equilibrium-Gleichungssystem" (Kapitel 3) (5.1-5.3)

$$\left(\prod_{j=0}^{n-1}\left[[\mathrm{A}_j]_0+\sum_{i=0}^{m-1}a_{ij}\cdot\xi_i\right]^{a_{0j}}\right)-K_0 \;=\; 0 \tag{5.1}$$

$$\left(\prod_{j=0}^{n-1}\left[[\mathrm{A}_j]_0+\sum_{i=0}^{m-1}a_{ij}\cdot\xi_i\right]^{a_{1j}}\right)-K_1 \;=\; 0 \tag{5.2}$$

$$\cdots$$

$$\left(\prod_{j=0}^{n-1}\left[[\mathrm{A}_j]_0+\sum_{i=0}^{m-1}a_{ij}\cdot\xi_i\right]^{a_{mj}}\right)-K_{m-1} \;=\; 0 \tag{5.3}$$

unter den Nebenbedingungen (5.4)

$$[\mathrm{A}_j]_t=[\mathrm{A}_j]_0+\sum_{i=0}^{m-1}a_{ij}\cdot\xi_i\geq 0 \quad j=0,...,n-1 \tag{5.4}$$

computergestützt gelöst werden. Als schnellstes und allgemeines Verfahren in der Anwendung auf das Equilibrium-Problem hat sich das Broydenverfahren erwiesen, wie in Kapitel 4 dargestellt wurde.
Dieses Kapitel befasst sich mit der objektorientierten *C++*-Implementierung

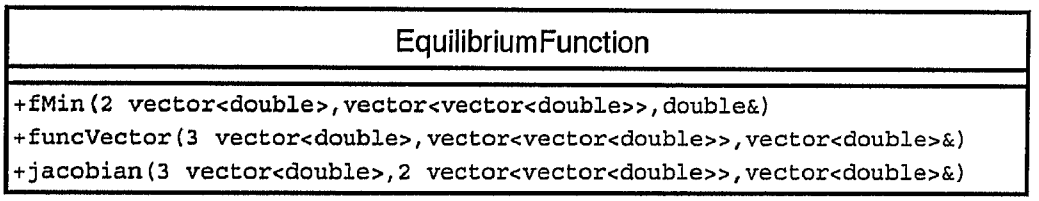

Abbildung 5.1: *Die Klasse EquilibriumFunction zur Berechnung von Funktionsauswertungen und Jacobi-Matrizen*

- der **EquilibriumFunction-Klasse** mit Methoden der Funktionsgenerierung des allgemeinen Equilibrium-Problems (5.1-5.4), dessen Jacobi-Matrix und weiterer wichtiger Hilfsmethoden
- der **Broyden-Klasse** mit der Lösung der Equilibrium-Funktionen mit dem Broydenverfahren, der Schrittweitensteuerung, der Solve-Methode, die unter anderem die Nebenbedingungen des Equilibrium-Problems betrachtet und der initial-Methode, die Matrixzerlegungen durchführt und die endgültige Lösung berechnet

Dabei werden ausgewählte, zentrale Methoden und Klassen erklärt. Der Programmcode befindet sich im Anhang und online unter http://echempp.sourceforge.net

5.1.1 Die Klasse EquilibriumFunction

Eine Übersicht der Klasse EquilibriumFunction findet sich in Abbildung (5.1). **Die Methode fMin**

Die Methode `void fMin()` berechnet den Funktionswert, der für das schrittweitengesteuerte Broydenverfahren wichtigen Hilfsfunktion f (vgl. Gleichung (4.25)):

$$f = F \cdot F = \sum_i F_i(x)^2 \tag{5.5}$$

Diese Funktion wird im Broydenverfahren minimiert. Ihre globalen Minima — ihre Nullstellen in somit — sind auch Nullstellen des "Equilibrium-Funktionsvektors" — der mit der Methode `funcvector` berechnet wird (vgl. Gleichung (5.1-5.3).

Die Eingangsdaten an `fMin` sind die Stöchiometriematrix A, der Vektor der Anfangskonzentrationen, der Vektor der Gleichgewichtskonstanten und der momentane Reaktionslaufzahlvektor x. Rückgabewert ist ein Skalar — f.

Die Methode funcVector

Die grundlegende Methode `void funcVector()` der Klasse EquilibriumFunction berechnet mit Hilfe der Eingangsdaten (Stöchiometriematrix A, Vektor der Anfangskonzentrationen initialconc, Vektor der Gleichgewichtskonstanten und als Variable den

Reaktionslaufzahlvektor x) den Funktionsvektor $F(x)$ (hier: fvec) des Equilibrium-Funktions-Systems (5.1-5.3). Aufgrund der häufigen Anwendung und der einsichtigen Implementierung der `funcVector`-Methode, ist hier der C++-Code abgedruckt:

```
void funcVector (const vector<vector<double> >& A,
 const vector<double>& initialconc,
 const vector<double>& Kc,
 vector<double>& fvec)

{
 double value;
 double basisvalue;
 fvec.clear();
 for (unsigned int reactionnumber=0;reactionnumber<A.size();++reactionnumber){
   value = 1;
   for (unsigned int i = 0; i < A[0].size(); ++i)  {
      basisvalue = initialconc[i];
      for (unsigned int j = 0; j < A.size(); ++j) {
              basisvalue += x[j] * A[j][i];
              }
      if (A[reaction][i] != 0.0)
       value *= pow(basisvalue, A[reactionnumber][i]);
      }
      fvec.push_back(value - Kc[reactionnumber]);
    }
}
```

Die Methode jacobian

Die Methode `void jacobian()` berechnet die Jacobi Matrix J (5.6) des Equilibrium-Funktionssystems $F(x)$ zum Zeitpunkt des Reaktionslaufzahlvektors x

$$J_{ij} = \frac{\partial F_i}{\partial x_j} \tag{5.6}$$

Da $F(x)$ aus beliebigen gebrochenrationalen Funktionen besteht, ist eine direkte Kalkulation von J nicht möglich. Die Methode `jacobian` benutzt eine *Finite-Differenzen-Methode*, um die Jacobi-Matrix zu approximieren. Die wesentliche Idee ist es, die einzelnen Tangenten-Steigungen bzw. partiellen Ableitungen durch geeignete Sekantensteigungen (bzw. durch einen Quotient *finiter Differenzen*) (5.7) zu ersetzen, die einfach zu berechnen sind:

$$\frac{\partial F(x)}{\partial x_j} \approx \frac{F(x) - F(x_\epsilon)}{\epsilon} \qquad \text{mit} \qquad x_\epsilon = (x_0, ..., x_j + \epsilon, ...)^T \tag{5.7}$$

Dabei wurde ϵ möglichst klein gesetzt (hier $\epsilon = 10^{-8}$).
Die Eingangsdaten an `jacobian` sind die Stöchiometriematrix A, der Vektor der Anfangskonzentrationen, der Vektor der Gleichgewichtskonstanten und der momentane Reaktionslaufzahlvektor x. Ausgabewert ist die Jacobimatrix J.

Broyden
+linesearch(..) +broydn(vector<double>&, vector<vector<double>>,2 vector<double>,bool) +solve(vector<vector<double>>, 2 vector<double>,vector<double>&) +initial(vector<vector<double>>,2 vector<double>,vector<double>&) -qrdcmp(vector<vector<double>>,vector<double>&,vector<double>&,bool) -qrupdt(vector<vector<double>>,vector<vector<double>>&,2 vector<double>&) -rsolv(vector<vector<double>>,vector<double>&,vector<double>&) -rotate(2 vector<vector<double>>&,int, 2 double)

Abbildung 5.2: *Die wichtige Broyden-Klasse*

5.1.2 Die Hauptklasse Broyden

Im folgenden werden zunächst die privaten Memberfunktionen `qrdcmp`, `qrudpt`, `rsolv` und `rotate` der Klasse `Broyden` vorgestellt, mit denen die benötigten Matrizenalgorithmen aus der Linearen Algebra durchgeführt werden können (aus [24]).

Die zentralen public Methoden der Broyden-Klasse sind die `linesearch`, `broydn`, `solve` und `initial` Methoden. `linesearch` steuert die Schrittweitenlänge, in `broydn` ist das Broydenverfahren realisiert und `solve` generiert unter anderem dynamisch Startwerte, kontrolliert die Nebenbedingungen und berechnet Gleichgewichtskonzentrationen. `initial` zerlegt die Eingangsmatrix in Subsysteme und löst diese mit Hilfe von `solve`.

Die Methode qrdcmp

Die Methode `void qrdcmp()` (QR-decomposition) berechnet die *QR-Zerlegung* der Eingabematrix A ([25], S. 131)

$$A = QR \tag{5.8}$$

Q ist eine orthogonale Matrix (d.h. es gilt: $Q^{-1} = Q^T$) und R eine rechte obere Dreiecksmatrix. Zur Berechnung der QR-Zerlegung werden *Householder-Transformationen* verwendet, die Spiegelungen an $(m-1)$-dimensionalen Hyperebenen darstellen. Durch geeignete Wahl dieser Spiegelungen wird die Matrix A auf obere Dreiecksform gebracht. Die Gesamtheit dieser Spiegelungen, die für sich genommen auch orthogonale Abbildungen darstellen, wird in Q gespeichert. Für eine genaue mathematisch-numerische Erläuterung des Verfahrens, siehe ([29], S. 283).

Die Methode rsolve

Die Methode `void rsolve()` löst das lineare Gleichungssystem $Rx = b$, wobei R wieder eine obere Dreiecksmatrix ist.

Die Methoden qrupdt und rotate

Einige numerische Algorithmen (beispielsweise das Broydenverfahren) benötigen die Lösung einer ganzen Reihe von linearen Gleichungssystemen, mit der Eigenschaft, dass sich jede Matrix nur etwas von ihrer Vorgängermatrix unterscheidet ([24], S. 101-104).

$$QR \longmapsto Q'R' \tag{5.9}$$

Um Rechenzeit zu sparen, berechnen die Methoden `void qrupdt()` und `rotate()` aus einer vorhergehenden QR Zerlegung von A ein Update der QR-Zerlegung $Q'R'$ (5.9) einer leicht veränderten Matrix A'. Der Algorithmus besteht aus zwei Teilen: `qrupdt` bedient sich der Methode `rotate` und transformiert die Eingangsmatrix in obere Hessebergform mittels *Jacobi-Rotationen* ([25], S. 281). Danach wird mittels weiterer Jacobi-Transformationen die obere Hessebergform in obere Dreickecksform R' gebracht. Q' ist dann einfach das Produkt aller Jacobi-Transformationen.

Die Methode linesearch

Die Methode `void linesearch()` wird für die Schrittweitensteuerung der broydn-Methode benötigt (Abschnitt 4.2.3):
Wenn man sich nicht sehr nahe am Minimum der Funktion $f = F \cdot F$ befindet, kann es sein, dass der volle Newton-Schritt den Funktionswert gar nicht kleiner machen wird ([24], S. 393). Man weiß, dass zu Anfang eine Abstiegsrichtung (=: p) hinsichtlich f benutzt wird. Da allerdings f quadratisch ist, muss nicht jede Schrittlänge zu einem Abstieg führen. Das Ziel ist es jetzt, von dem Ausgangspunkt x_{alt} zwar entlang der Richtung p zu einem neuen Punkt x_{neu} zu gelangen, aber nicht notwendigerweise den gesamten Schritt auszuführen.

$$x_{\text{neu}} = x_{alt} + \lambda p \quad 0 < \lambda \leq 1 \tag{5.10}$$

Jetzt ist λ (5.10) so zu wählen, dass $f(x_{\text{alt}} + \lambda p)$ signifikant abgenommen hat. Um λ zu bestimmen, gibt es verschiedene Strategien in der Literatur [18, 24]. Die hier verwendete Routine wird in ([24], S. 389) beschrieben.

Die Methode `linesearch` berechnet ausgehend von x_{alt} mit Hilfe des Gradienten und Funktionswertes in x_{alt} entlang einer vorgegebenen Newton-Richtung p eine optimale Schrittweite λ und daraus $x_{\text{neu}} = x_{\text{alt}} + \lambda p$. Danach wird $f(x_{\text{neu}})$ berechnet.

Die Methode broydn

Die `void broydn()` Methode ist der zentrale Algorithmus zur Lösung des Equilibrium-Problems. Sie bedient sich aller bisher beschriebenen Methoden. Ein Fliessdiagramm ist in Abbildung 5.3 dargestellt.

In der `broydn` Methode ist das Broydenverfahren zur Lösung nichtlinearer Gleichungssysteme (mit Schrittweitensteuerung zur Globalisierung der Konvergenz) implementiert, welches das Equilibrium-Funktionensystem (5.1-5.3) lösen kann. Die `broydn` Methode berechnet von einem gegebenen Startvektor (aus Reaktionslaufzahlen) x_0 iterativ neue Näherungsvektoren x_i zur Nullstellenbestimmung von $F(x)$. Die benötigten Jacobi-Matrizen (ausser im ersten Schritt) und die daraus folgenden Kalkulationen werden — wie in Kapitel (4.2.4) beschrieben — durch Broyden's Verfahren angenähert. In jedem Schritt ruft `broydn` die `linesearch` Methode auf, um die Schrittweiten zu berechnen. Dadurch wird eine globale Konvergenz erreicht. Leider kann es zu Verfahrensabbrüchen kommen, wenn

- die Jacobi-Matrix im ersten Schritt singulär ist, das heißt, dass man zufällig in einem lokalen Minimum gestartet ist
- `linesearch` nicht in einer vorgegebenen Anzahl an Durchläufen eine signifikante Abnahme von $f = F \cdot F$ feststellen kann
- die `broydn` Methode zu langsam konvergiert und in einer vorgegebenen Anzahl an Iterationen keine Approximation an die Nullstelle gefunden hat.

In diesem Fall gibt `broydn` ein *Fehlerflag* (einen boolschen Wert) als *true* zurück, der anzeigt, dass ein Problem aufgetreten ist – was in der Praxis durchaus nicht selten vorkommt.

Mit der `broydn` Methode und der equilibriumFunction-Klasse kann also das Equilibrium-Problem in den meisten Fällen gelöst werden. Leider kommt es häufig vor, dass die gefundene Nullstelle die Nebenbedingungen des Equilibrium-Problems (5.4) nicht erfüllt oder die `broydn` Methode ein Fehlerflag setzen muss. Tritt einer dieser Fälle auf, ist es am einfachsten, einen neuen Startwert zu benutzen und die Nullstellensuche zu wiederholen (siehe auch Fliessdiagramm(Abb. 5.3)).

Die Methode solve

Die `void solve()` Methode generiert unter anderem Startwerte und wird im nächsten Abschnitt vorgestellt.

5.1.3 Die Methode initial in der Broyden-Klasse

Die Methode `void initial()` löst das Equilibrium-Problem mit Hilfe des erläuterten allgemeinen Lösungs-Verfahrens unter Anwendung der `solve`-Methode. `initial` zerlegt die Eingabematrix in Subsysteme (siehe Abschnitt 5.2.2) und löst diese mit der `solve` Methode (siehe Fliessdiagramm 5.4). Anschließend setzt `initial` die berechneten Gleichgewichtskonzentrationen in der richtigen Reihenfolge zu einem einzelnen

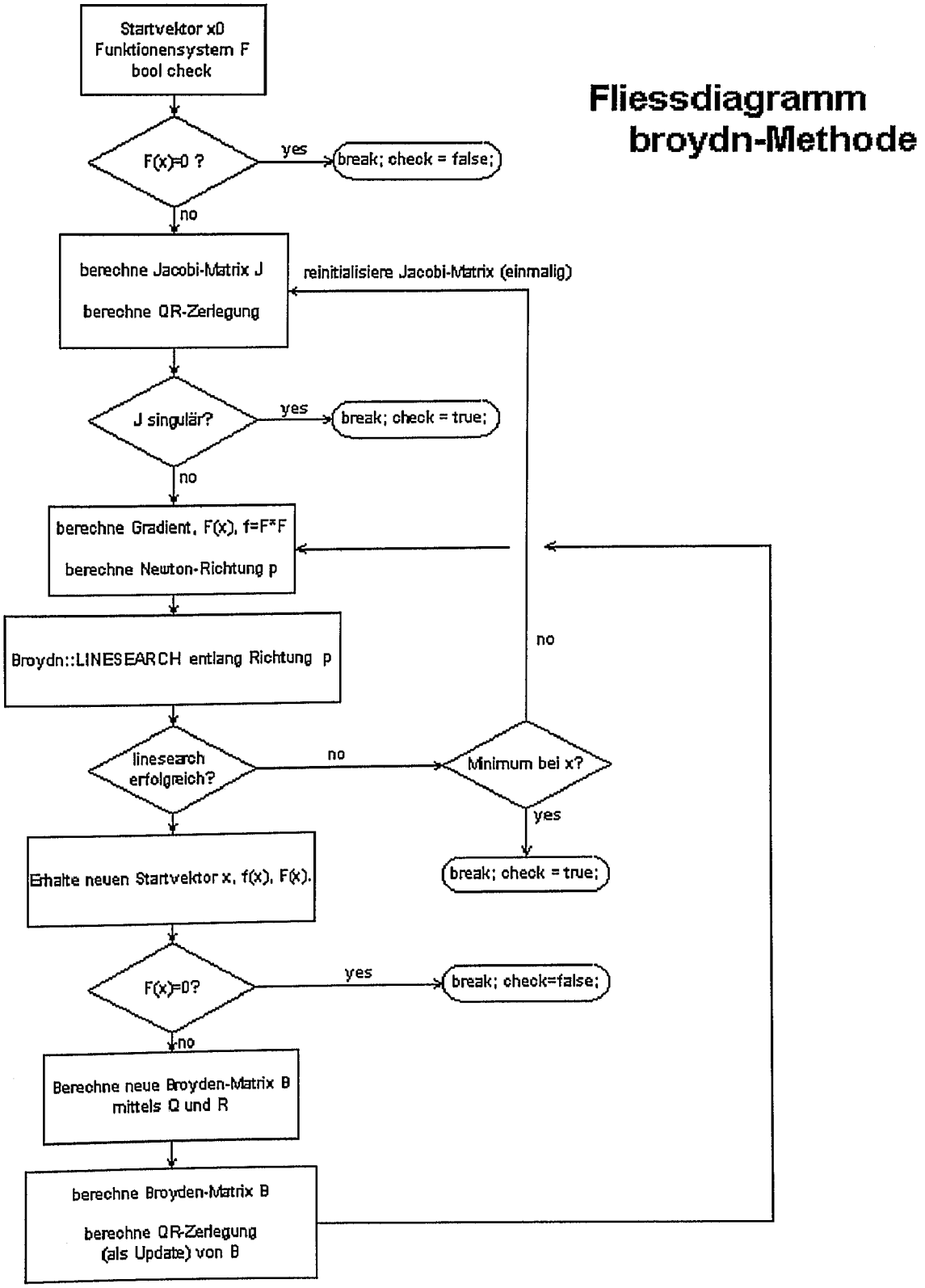

Abbildung 5.3: *Fliessdiagramm der zentralen Methode broydn*

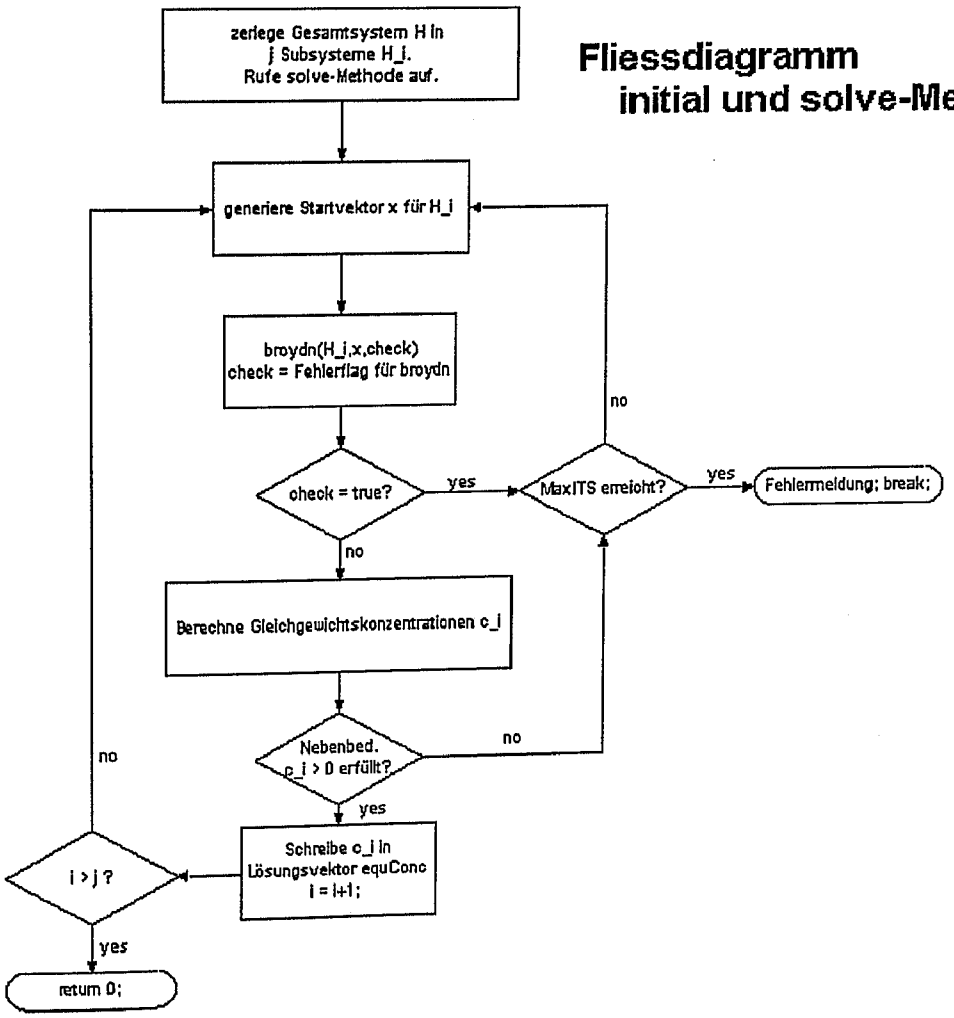

Abbildung 5.4: *Fliessdiagramm der Methode solve*

Lösungsvektor zusammen.

5.1.4 Die Methode solve in der Broyden-Klasse

Die Methode `void solve()` stellt die Lösungsroutine des Equilibrium-Problems dar. Sie erfüllt hierbei mehrere Aufgaben (siehe Fliessdiagramm Abb. 5.4):

- **generiert Startwerte**, ruft die `broydn`-Methode auf und kontrolliert diese.
- überprüft Einhaltung der **Nebenbedingungen** des Equilibrium-Problems
- berechnet schließlich die **Gleichgewichtskonzentrationen** als Lösungen des Equilibrium-Problems

Rückgrat dieser Methode ist die Generierung von Startwerten, die im folgenden Abschnitt beschrieben wird.

Dynamische Generierung von Startwerten

Es ist notwendig, die `broydn` Methode mit immer neuen Startwerten zu versorgen, bis fehlerfrei eine Nullstelle gefunden wurde, die dazu noch die Nebenbedingungen des Equilibrium-Problems erfüllt.

Die `solve` Methode generiert auf folgende Weise zufällige Startwerte zur Lösung des Equilibrium-Problems:

1. Der Maximalwert $Y_{\max}$ der einzelnen Reaktionslaufzahlen wird betragsmäßig errechnet als Summe aller Anfangskonzentrationen, ansonsten kann nicht ausgeschlossen werden, dass negative Konzentrationen auftreten.

2. Die Grenze Y wird auf einen kleinen Wert gesetzt (z.B. 0.1).

3. Es wird ein zufälliger Startwert generiert. Dieser wird so erzeugt, dass jeder Vektoreintrag im Intervall $[-Y, +Y]$ liegt. Wie Tests (siehe auch Abschnitt 5.4) gezeigt haben, ist es von Vorteil, zuerst viele Startwerte um den Nullvektor zu generieren.

4. Dann wird überprüft, ob `broydn` fehlerfrei durchgelaufen ist. Ist das Fehlerflag false, werden die Nebenbedingungen abgefragt. Dazu werden mit den Gleichungen (5.4) die Gleichgewichtskonzentrationen berechnet und geprüft, ob diese positiv sind. Ist dies erfüllt, wird `solve` verlassen.

5. Falls `broydn` nicht fehlerfrei abgelaufen ist oder die Nebenbedingungen nicht erfüllt sind, wird Y um einen kleinen Wert erhöht und ein neuer Startwert generiert. Dies wird wiederholt, bis $Y_{\max}$ erreicht ist.

6. Der beschriebene Algorithmus wird mehrfach ausgeführt, bis zu einer Maximalanzahl an Durchläufen. Besser wäre es, nach einer vorzugebenden Rechenzeit (z.B. 3 s) den Algorithmus abzubrechen und eine Fehlermeldung auszugeben. Dem Anwender würde es in diesem Fall überlassen bleiben, es noch einmal zu versuchen.

Berechnung der Gleichgewichtskonzentrationen

Ist der Lösungsvektor aus Reaktionslaufzahlen gefunden, berechnet `solve` mit Gleichung (5.4) die Gleichgewichtskonzentrationen als Lösung des Equilibrium-Problems.

5.1.5 Möglichkeit eines genetischen Solvers

Die Generierung der Startwerte erfolgt alleine aufgrund von Erfahrungswerten und Zufallsalgorithmen. Mit neuartigen **"genetischen Algorithmen"** könnte man diese Generierung qualitativ steuern.

Viele genetische Algorithmen beruhen auf folgender Grundidee (die der natürlichen Evolution und Genetik entlehnt ist) [30]:

- beispielsweise 100 "Kreaturen" (hier Startwerte x_0) **mutieren** zufällig (es wird in etwa ein Zufallsvektor von bestimmter Maximalgröße addiert)
- jeweils zwei Kreaturen (Startwerte) **paaren** sich und bringen eine neue Kreatur hervor, mit Eigenschaften der beiden Startkreaturen. Im Beispiel des Equilibrium-Problems wäre die Mittelwertbildung der beiden Startwerte ein möglicher Paarungsvorgang.
- von den nun vorhandenen 150 Kreaturen "**sterben**" die 50, die am wenigsten "fit" sind.
- die verbleibenden 100 Kreaturen mutieren wieder und der Prozess beginnt von vorn. Damit werden die so generierten Startvektoren in jedem Schritt besser.

Leider war es für das Equilibrium-Problem nicht möglich, eine qualitative Bewertung der Startwerte zu implementieren — d.h. eine geeignete Fitnessfunktion aufzustellen. Dies liegt daran, dass man *a priori* nicht entscheiden kann, ob ein Startwert zur Konvergenz in die richtige Nullstelle führt. Führt der Startwert im nachhinein zu einer falschen Nullstelle, weiß man nur, dass der Startwert schlecht war — ist nicht einsichtig *wie "schlecht"* er wirklich ist. Die Startwerte hängen in keiner systematischen Weise (stetig, monoton) von den Lösungen ab.

Allerdings könnte ein genetischer Algorithmus das Verfahren der Wahl für zukünftige numerische Lösungen des Equilibrium-Problems zu sein.

5.2 Algorithmus-Beschleunigung

Die computergestützte Lösung des Equilibrium-Problems zeigt sich als sehr aufwendig. Von Vorteil ist es somit, das Problem schon vor dem Beginn der Berechnungen möglichst einfach zu gestalten, um den Rechenaufwand nicht unnötig groß werden zu lassen.

5.2.1 Eliminierung linearer Abhängigkeiten

In einem beliebigen Reaktionssystem kann es vorkommen, dass einzelne Reaktionen direkt aus einer Kombination anderer Reaktionen beschrieben werden können. In einfachen Fällen ist dies direkt aus dem Reaktionsschema ersichtlich. Dazu folgendes Beispiel ([13], S. 170) — die Methanspaltung bei 900K (5.11-5.13)

$$CH_4 + H_2O \rightleftharpoons CO + 3H_2 \tag{5.11}$$

$$CO + H_2O \rightleftharpoons CO_2 + H_2 \tag{5.12}$$

$$CH_4 + 2H_20 \rightleftharpoons CO_2 + 4H_2 \tag{5.13}$$

Hier ist offensichtlich die Reaktionsgleichung der dritten Reaktion gerade die Summe der ersten beiden Reaktionen. Somit kann die letzte Reaktion weggelassen werden. Die Matrixdarstellung dieses Systems

$$H = \begin{pmatrix} -1 & -1 & 1 & 3 & 0 \\ 0 & -1 & -1 & 1 & 1 \\ -1 & -2 & 0 & 4 & 1 \end{pmatrix}$$

zeigt, dass die dritte Zeile eine *Linearkombination* der ersten beiden Zeilen und folglich **linear abhängig** ist.

In beliebig komplexen Reaktionssystemen sucht man somit in der Stöchiometriematrix nach Zeilen, die linear abhängig von den anderen Zeilen (=Reaktionen) sind. Dies kann ohne viel Aufwand durch Matrixtechniken überprüft werden (z. B. durch QR-Zerlegung [23] S. 399). Diese Zeilen werden dann für die folgenden Berechnungen gestrichen. Die Dimension des Equilibrium-Problems wird so um die Anzahl der linear abhängigen Zeilen reduziert und der Lösungsalgorithmus kann dementsprechend beschleunigt ablaufen.

Die praktische Umsetzung dieser Beschleunigung wurde nicht implementiert, da diese bereits im Ecco-Compiler [31] realisiert ist.

5.2.2 Aufteilung in Subsysteme

Nach der Entfernung eventuell vorhandener linear abhängigen Zeilen, besteht eine weitere Vereinfachung darin, die Stöchiometriematrix in mehrere einzelne Matrixteile, bzw. das Reaktionssystem des Equilibrium-Problems in — unabhängig voneinander dem Gleichgewicht zustrebende — "Subsysteme", zu unterteilen und diese getrennt zu behandeln. Wenn dies möglich ist, würde das eine bedeutende Beschleunigung des Verfahrens erlauben.
Zum Beispiel ließe sich das System (5.14-5.16)

$$A \rightleftharpoons B \tag{5.14}$$

$$B \rightleftharpoons C \tag{5.15}$$

$$D \rightleftharpoons E \tag{5.16}$$

in zwei Teile zerlegen und zwar in ein Subsystem aus Gleichung (5.14) mit (5.15) und ein System aus (5.16). Diese ließen sich dann getrennt behandeln. Offensichtlich ist es deulich effizienter, nacheinander ein zweidimensionales und ein dreidimensionales Nullstellenproblem zu lösen, als ein einzelnes fünfdimensionales Problem.

Die Aufteilung in Subsysteme ist nur dann ohne Verfälschung des Ergebnisses möglich, wenn die einzelnen Teile nichts mehr miteinander zu tun haben — das ist der Fall, wenn sie keine gemeinsamen Spezies haben. Die Aufteilung besteht darin, dass man einzelne *Reaktionen* (d.h. Zeilen) zusammenfasst. Würde man Reaktionen zerstückeln (bzw. die Zeilen der Stöchiometriematrix), bekäme man Probleme mit der Gleichgewichtskonstanten, die nicht nur für einzelne Reaktionsteile gelten kann, sondern stets für die Gesamtreaktion gilt.

Die Methode `initial` in der broyden-Klasse erfüllt unter anderem diese Aufgabe. Die entstandenen Blöcke (oder *Subsysteme*) werden getrennt behandelt und der Gleichgewichtsvektor am Schluss der Methode richtig zusammengesetzt. Dies verlangsamt die Methode im Falle nur eines unabhängigen Blocks nicht merklich. Sind allerdings Möglichkeiten zur unabhängigen Zerlegung des Systems vorhanden, beschleunigt dies das Verfahren enorm (siehe Abschnitt 5.4.1).

5.2.3 Präiterative Startwertbestimmung

Eine logisch erscheinende, zusätzliche Erweiterung des Solvers wäre es, nicht um den Durchschnitt an erlaubten Reaktionslaufzahlen (den Nullvektor) Startvektoren zu generieren, sondern um einen berechneten Vektor, der vorhandene Informationen mitberücksichtigt. Dieser würde vor dem Beginn der dynamischen Startwertgenerierung (*d.h. präiterativ*) kalkuliert.

Numerisch wenig aufwendig ist es, jede Reaktionsgleichung einzeln zu betrachten und dieses Subsystem mit der `solve`-Methode zu lösen. Man erhält so einen Startvektor $x \neq 0$ von Reaktionslaufzahlen und kann diesen statt dem Nullvektor in die `solve`-Methode einbauen. Einige Tausend Testdurchläufe haben allerdings gezeigt, dass dies zu keiner Beschleunigung des Gesamtalgorithmus im allgemeinen führt. Es wurde im Gegenteil sogar eine signifikante Verlangsamung beobachtet. Als Testdaten wurden beispielswiese verwendet:

- Stöchiometriematrix

$$H = \begin{pmatrix} -1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 2 \end{pmatrix}$$

- Gleichgewichtskonstanten: $K_c = (1, 2, 3, 4)^T$
- Anfangskonzentrationen: $c^0 = (1, 2, 3, 4, 5, 6, 0)^T$

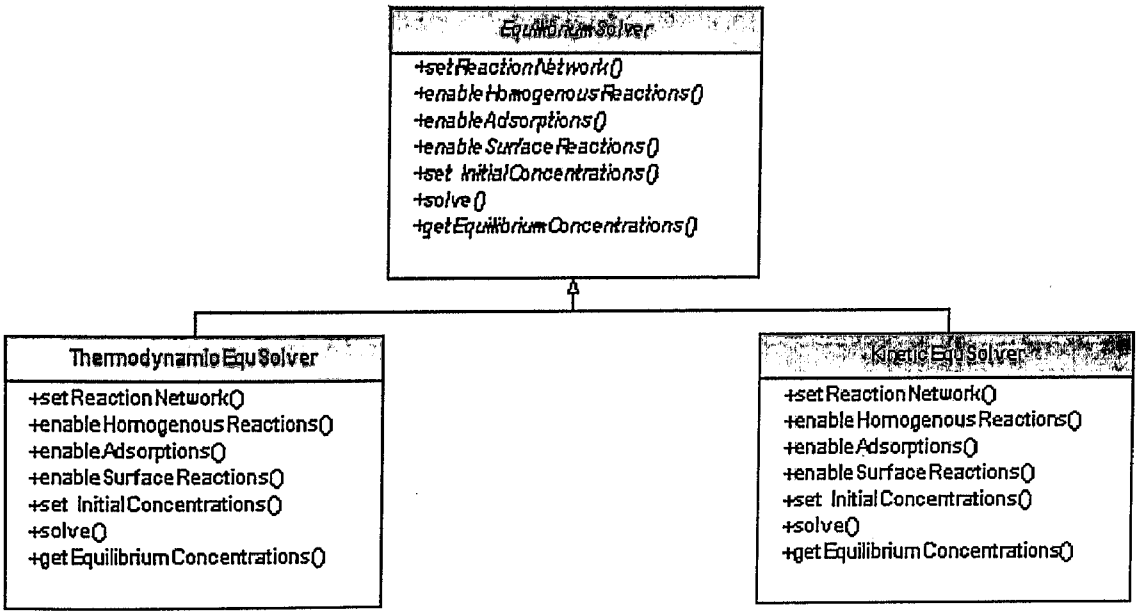

Abbildung 5.5: *Klassenhierarchie zur Lösung des Equilibrium-Problems. Virtuelle Klassen und Methoden sind kursiv dargestellt.*

Die Geschwindigkeit nahm in diesem Fall um den Faktor 10 ab — was vermutlich von der Wahl der Testdaten abhängt. Auf jeden Fall wurde eine generelle Algorithmusbeschleunigung durch eine präiterative Startwertbestimmung widerlegt und daher wurde diese Methode nicht weiter verfolgt.

5.3 Einbindung in das Echem++-Projekt

Das Echem++-Projekt versucht mit rechnergestützten Simulationen die Auswertung elektrochemischer Vorgänge zu unterstützen. Das Open-Source Projekt **Echem++** [31] findet sich unter http://echempp.sourceforge.net. Dort findet sich auch der vollständige Sourcecode der vorgestellten Klassen `EquilibriumFunction` und `Broyden` zur allgemeinen Lösung des Equilibrium-Problem wieder. Das folgende Kapitel beschreibt die Klassenhierarchie der Equilibrium-Problem-Löser und die Anbindung des beschriebenen Lösungsverfahrens an das Echem++-Projekt.

5.3.1 Die EquilibriumSolver-Klasse als Schnittstelle zu Ecco

Der Ecco-Compiler [2, 3] liefert die benötigten Eingangsdaten für die nun vorzustellende virtuelle Klasse "EquilibriumSolver". Diese Klasse löst das Equilibrium-Problem und gibt die berechneten Gleichgewichtslösungen an den Ecco-Compiler zurück: Wie man anhand der dargestellten Klassenhierarchie (Abb. (5.5)) der Klasse `EquilibriumSolver` sieht, kann die Lösung des Equilibrium-Problems nach Wahl des Anwenders entwe-

der durch die vorgestellte thermodynamische Methode mit Hilfe der (noch zu beschreibenden) `ThermodynamicEquSolver`-Klasse oder durch eine kinetische Methode (implementiert von LUDWIG [6]) mit Hilfe der `KineticEquSolver`-Klasse gelöst werden. Durch die objektorientierte C++-Progammiertechnik des *Polymorphismus*[8] der `EquilibriumSolver`-Klasse ist diese Wahl benutzerfreundlich für andere Programmierer umzusetzen.

Die kinetische Methode beruht auf der Lösung eines Systems gewöhnlicher Differentialgleichungen (vgl. Abschnitt 3.2.2). Dabei werden Geschwindigkeitskonstanten der Hin- und Rückreaktionen aller Gleichgewichte benötigt, womit mehr Informationen für diesen Solver zur Verfügung stehen müssen, als die `ThermodynamicEquSolver`-Klasse benötigt. Prinzipiell hat jede der beiden Lösungsmethoden ihre Vor- und Nachteile (siehe Abschnitt 5.4.3). Es bleibt dem Anwender überlassen, für welche Methode er sich entscheidet.

5.3.2 Die Klasse ThermodynamicEquSolver

Die Klasse `ThermodynamicEquSolver` stellt die Schnittstelle des hier vorgestellten Projekts an das Echem++-Projekt dar. Im folgenden werden die Methoden dieser Klasse erläutert.

Die Methoden setReactionNetwork und -enable Methoden

Die Methode `setReactionNetwork` importiert die vollständige über den Ecco-Compiler eingegebene Stöchiometriematrix in die ThermodynamicEquSolver-Klasse (eventuelle linear abhängige Zeilen (=Reaktionen) wurden bereits eliminiert).

Allerdings enthält diese Stöchiometriematrix nicht nur die Koeffizienten der benötigten vorgelagerten Gleichgewichte, sondern betrachtet auch Reaktionen, die nichts mit der Einstellung der Anfangsbedingungen zu tun haben. *enableHomogenousReactions*, *enableAdsorptions* und *enableSurfaceReactions* mit einem boolschen Übergabewert erlauben es, gezielt einzelne Reaktionsklassen zu eliminieren. So werden die Zeilen der Adsorptionsreaktionen mit *enableAdsorptions* und der Oberflächenreaktionen mit *enableSurfaceReactions* in der Stöchiometriematrix gestrichen.

Die Methode getInitialConcentrations

Die Methode `getInitialConcentrations` setzt den benötigten Vektor der Anfangskonzentrationen der Reaktionspartner und den der Gleichgewichtskonstanten der betrachteten Reaktionen.

Die Methode getEquilibriumConcentrations

Die Methode `getEquilibriumConcentrations` übergibt den Vektor der Gleichgewichtskonzentrationen an den Ecco++-Compiler zur Verwendung als Anfangs-(oder Randwert-)konzentrationen zur Simulation elektrochemischer Reaktionen (siehe Screenshots Abb. (5.4.3)).

5.4 Bewertung, Tests und Screenshots

5.4.1 Testsysteme und Leistung des Verfahrens

In diesem Abschnitt werden einige Testreaktionssysteme vorgestellt und deren Lösung mit dem ThermodynamicEquSolver diskutiert. Die Rechenzeiten ergeben sich als Mittelwert von 1.000 Testdurchläufen mit einem Pentium III, 650 MHz unter dem Linux-Betriebssystem.

Testsystem I

Einfachstes Gleichgewichtssystem ist ein eindimensionales System, d.h. es besteht nur aus einer Reaktion. Das folgende Beispielreaktionssystem besteht aus 14 Reaktanden, um zu demonstrieren, dass die Anzahl der Reaktanden durchaus groß sein darf — *eindimensional, 14 Reaktanden* (5.17-5.18)

$$H = \begin{pmatrix} -1 & -2 & -3 & -2 & -2 & -4 & -5 & -1 & 1 & 2 & 2 & 2 & 3 & 4 \end{pmatrix} \tag{5.17}$$

$$K_C = 2 \tag{5.18}$$

$$[A]_0 = (1,1,1,1,1,1,1,1,1,1,1,1,1,1)^T \tag{5.19}$$

Der ThermodynamicEquSolver lieferte das unten stehende Ergebnis. Durch Einsetzen in das Massenwirkungsgesetz ergibt sich K_C — die Berechnung der Gleichgewichtskonzentrationen ist also korrekt durchgeführt worden.

$$[A_0]_{eq} = 0.9932; \quad [A_1]_{eq} = 0.9865; \quad [A_2]_{eq} = 0.9797; \quad [A_3]_{eq} = 0.9865;$$

$$[A_4]_{eq} = 0.9865; \quad [A_5]_{eq} = 0.9729; \quad [A_6]_{eq} = 0.9662; \quad [A_7]_{eq} = 0.9932;$$

$$[A_8]_{eq} = 1.007; \quad [A_9]_{eq} = 1.014; \quad [A_{10}]_{eq} = 1.014; \quad [A_{11}]_{eq} = 1.014;$$

$$[A_{12}]_{eq} = 1,020; \quad [A_{13}]_{eq} = 1,027;$$

Die Rechenzeit betrug durchschnittlich **0.0031 s**. Man sieht, dass trotz der 14 einzelnen Konzentrationen, die zu berechnen sind, der Rechenaufwand vergleichsweise klein bleibt. Wie sich im folgenden Beispiel zeigt, beeinflußt die Anzahl der Reaktionen (=: *die Dimension des Systems*) die Kalkulationsgeschwindigkeit deutlich stärker.

Testsystem II

Es folgt ein komplizierteres System (5.20-5.22), an dem vor allem der Geschwindigkeitsvorteil durch Einteilung in Subsysteme demonstriert wird. Die erste und dritte Reaktion haben mit der zweiten und vierten Reaktion keine Reaktanden (d.h. Spalteneinträge $\neq 0$ in H) gemeinsam. Das System zerfällt folglich in zwei Subsysteme.
4-dimensional, 7 Reaktanden, 2 Subsysteme.
Stöchiometriematrix

$$H = \begin{pmatrix} -1 & 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 2 \end{pmatrix} \tag{5.20}$$

Daraus ergebende unabhängige Subsysteme — System I (Zeile 1 und 3)

$$\begin{aligned} \mathrm{A} &\rightleftharpoons \mathrm{B{+}2D} \\ \mathrm{3D} &\rightleftharpoons \mathrm{2B} \end{aligned}$$

und System II (Zeile 2 und 4)

$$\begin{aligned} \mathrm{C} &\rightleftharpoons \mathrm{E} \\ \mathrm{C{+}F} &\rightleftharpoons \mathrm{2G} \end{aligned}$$

Mit den Reaktionsparametern

$$K_C = (1, 2, 3, 4)^T \tag{5.21}$$

$$[A]_0 = (1, 2, 3, 4, 5, 6, 0)^T \tag{5.22}$$

Der ThermodynamicEquSolver liefert folgende Lösung:

$$\begin{aligned} [A_0]\mathrm{eq} &= 1.6949; \quad [A_1]\mathrm{eq} = 1.71603; \quad [A_2]\mathrm{eq} = 0.993823; \quad [A_3]\mathrm{eq} = 1.91724; \\ [A_4]\mathrm{eq} &= 3.83447; \quad [A_5]\mathrm{eq} = 2.75171; \quad [A_6]\mathrm{eq} = 6.49658; \end{aligned}$$

Setzt man die berechneten Werte in das Massenwirkungsgesetz ein, ergeben sich tatsächlich die Gleichgewichtskonstanten 1, 2, 3 und 4. Die Rechenzeit betrug durchschnittlich **0.014** s.

Ohne eine Aufteilung in zwei zweidimensionale Subsysteme, erhöht sich die durchschnittliche Rechenzeit auf **0.213** s.

Wird zusätzlich die dynamische Startwertgenerierung nicht verwendet, sondern Zufallsstartwerte innerhalb des erlaubten Bereichs generiert, erhöhte sich die durchschnittliche Rechenzeit auf **0.96** s. Die Verwendung dieser beiden Beschleunigungsverfahren zeigte sich somit als sinnvoll.

Das beschriebene Verfahren hat sich als leistungsfähig erwiesen. So löst es die in der Anwendung meist auftretenden nicht zu hochdimensionalen Systeme in kurzer Rechenzeit (< 5s). Limitiert wird dabei die Leistung vor allem durch:

1. große Stöchiometriekoeffizienten (z.B. > 4), da diese die Funktionsauswertungen der equilibriumFunction-Klasse verlangsamen, die sehr häufig benötigt werden. Das Broydenverfahren steuert diesem Effekt zwar entgegen, doch ist eine signifikante Verlangsamung zu spüren. Auch steigt die Anzahl der Nullstellen des Equilibrium-Problems und somit die Anzahl der Tests, die die `solve`-Methode durchschnittlich durchführen muss, da der Algorithmus häufiger eine Nullstelle berechnen wird, die die geforderten Nebenbedingungen nicht erfüllt.

2. eine zu große Dimension des Problems (z.B. > 7)

3. die Anzahl der Reaktanden. Allerdings kommt es hier vorallem auf die Summe der Stöchiometriekoeffizienten der einzelnen Reaktanden an. Zum Beispiel wirkt sich ein $+8A_9$ weniger stark aus, als wenn $+A_9 + A_{10}$ hinzukommt — was die Funktionsauswertung weniger stark verlangsamen wird.

4. die vorgegebene Anzahl an Nichtnull-Anfangskonzentrationen. Je weniger Anfangskonzentrationen ungleich null vorgegeben werden, desto langsamer ist der Algorithmus. Vermutlich liegen die Nullstellen der Equilibrium-Funktion zu nahe aneinander, so dass die stochastische Wahrscheinlichkeit, die *richtige* Nullstelle zu berechnen, abnimmt.

5. die Grösse der Gleichgewichtskonstanten. Sind diese sehr groß (> 1000), verlangsamt sich der Algorithmus. Die Ursache ist wahrscheinlich derselbe Effekt wie in (4).

5.4.2 Vergleich mit dem KineticEquSolver und Berechnungsgrenzen der Solver

Der KineticEquSolver (LUDWIG [31]) berechnet die Lösung des Equilibrium-Problems mit Hilfe von Differentialgleichungen. Hat der Anwender des Ecco-Compilers die Geschwindigkeitskonstanten der Hin- und Rückreaktion eingegeben, kann er sich zwischen dem KineticEquSolver und dem ThermodynamicEquSolver entscheiden. Im folgenden werden Beispiele vorgestellt, in denen jeweils ein Solver zu keinem Ergebnis kommen wird.

Testsystem III

$$\mathrm{A} \rightleftharpoons \mathrm{B} \tag{5.23}$$

$$2\mathrm{B} \rightleftharpoons \mathrm{C{+}D} \tag{5.24}$$

$$K_\mathrm{C} = (1,3)^T, \quad [A]_\mathrm{init} = (0.00001, 0, 0, 0)^T, \quad k_{\mathrm{f}_1} = 10^4, \quad k_{\mathrm{f}_2} = 10^{-4} \tag{5.25}$$

k_{f_1} und k_{f_2} sind die Geschwindigkeitskonstanten der Hinreaktionen der ersten bzw. zweiten Reaktion. Der KineticEquSolver liefert für das System (5.26-5.28) kein Ergebnis. Der Grund sind die sehr unterschiedlichen Geschwindigkeitskonstanten, was das numerische Problem der "steifen Differentialgleichungen" hervorruft [32]. Diese versucht der KineticEquSolver erfolglos zu lösen. Der ThermodynamicEquSolver löst das System problemlos:

$$[A_0]_\mathrm{eq} = [A_1]_\mathrm{eq} = 1.8301 \cdot 10^{-5}; \quad [A_2]_\mathrm{eq} = [A_3]_\mathrm{eq} = 3.1699 \cdot 10^{-5};$$

Testsystem IV

$$\mathrm{A} \rightleftharpoons \mathrm{B} \tag{5.26}$$

$$2\mathrm{B} \rightleftharpoons \mathrm{C{+}D} \tag{5.27}$$

$$\mathrm{D} \rightleftharpoons \mathrm{E} \tag{5.28}$$

$$\mathrm{E} \rightleftharpoons 2\mathrm{F} \tag{5.29}$$

$$\mathrm{F} \rightleftharpoons \mathrm{G} \tag{5.30}$$

$$K_\mathrm{C} = (1,2,3,4,5)^T, \quad [A]_\mathrm{init} = (1,0,0,0,0,0,0)^T, \tag{5.31}$$

Mit geeigneten vorgegebenen Geschwindigkeitskonstanten (die sich nicht zu stark von einander unterscheiden) kann der KineticEquSolver das System 5.26-5.31) lösen. Der ThermodynamicEquSolver löst dieses Equilibrium-Problem nicht in akzeptabler Rechenzeit ($<$ 5s) — auf Grund der hohen Dimension des Systems und der Tatsache, dass nur eine Anfangskonzentration ungleich null vorgegeben wird. Werden allerdings drei oder mehr Anfangskonzentrationen ungleich null vorgegeben, kommt der Solver stets zu einer Lösung. Vermutet wird, dass durch die mangelnden Nichtnull-Anfangskonzentration die Nullstellen der Equilibrium-Funktionen zu nahe aneinander liegen. Dadurch würde die Wahrscheinlichkeit stark sinken, dass der Algorithmus in der richtigen Nullstelle terminiert.

Zusammengefaßt heißt das, dass der ThermodynamicEquSolver verwendet werden sollte, wenn keine Geschwindigkeitskonstanten vorgegeben werden können, die Dimension des Equilibrium-Problems nicht zu hoch ist, die Geschwindigkeitskonstanten sehr hohe Unterschiede aufweisen oder der KineticEquSolver durch andere Gründe kein Ergebnis liefern sollte. Der KineticEquSolver löst besser höherdimensionale Systeme, falls wenig Nichtnull-Anfangskonzentrationen vorgegeben sind. Seine Geschwindigkeit ist allerdings im allgemeinen niedriger und die Anzahl der Reaktanden fällt stärker ins Gewicht. In der Anwendungspraxis sollte der Benutzer beide Solver-Varianten selbst auszuprobieren und sich individuell für einen Solver entscheiden.

5.4.3 Screenshots

Wie in Abschnitt 5.3 erwähnt, kann sich der Anwender des Ecco-Compilers entscheiden, welche Lösungsmethode zur Berechnung der benötigten Anfangswerte er benutzen will. Bisher stehen ein kinetischer Solver (von LUDWIG [31]) und der hier vorgestellte thermodynamische Solver zur Verfügung. Im Graphical User Interface (GUI) der Echem++-Software ist diese Vorabwahl möglich, wie der erste Screenshot (Abb. 5.6) zeigt. Im unteren Fenster werden die Anfangskonzentrationen angegeben. Der zweite Screenshot (Abb. 5.7) zeigt die Eingabe der Gleichgewichtskonstanten und ggf. der Geschwindigkeitskonstanten in einem Beispielsystem. Der dritte Screenshot (Abb. 5.8) zeigt die Darstellung der kalkulierten Lösung des vorgelagerten Gleichgewichtssystems durch den ThermodynamicEquSolver.

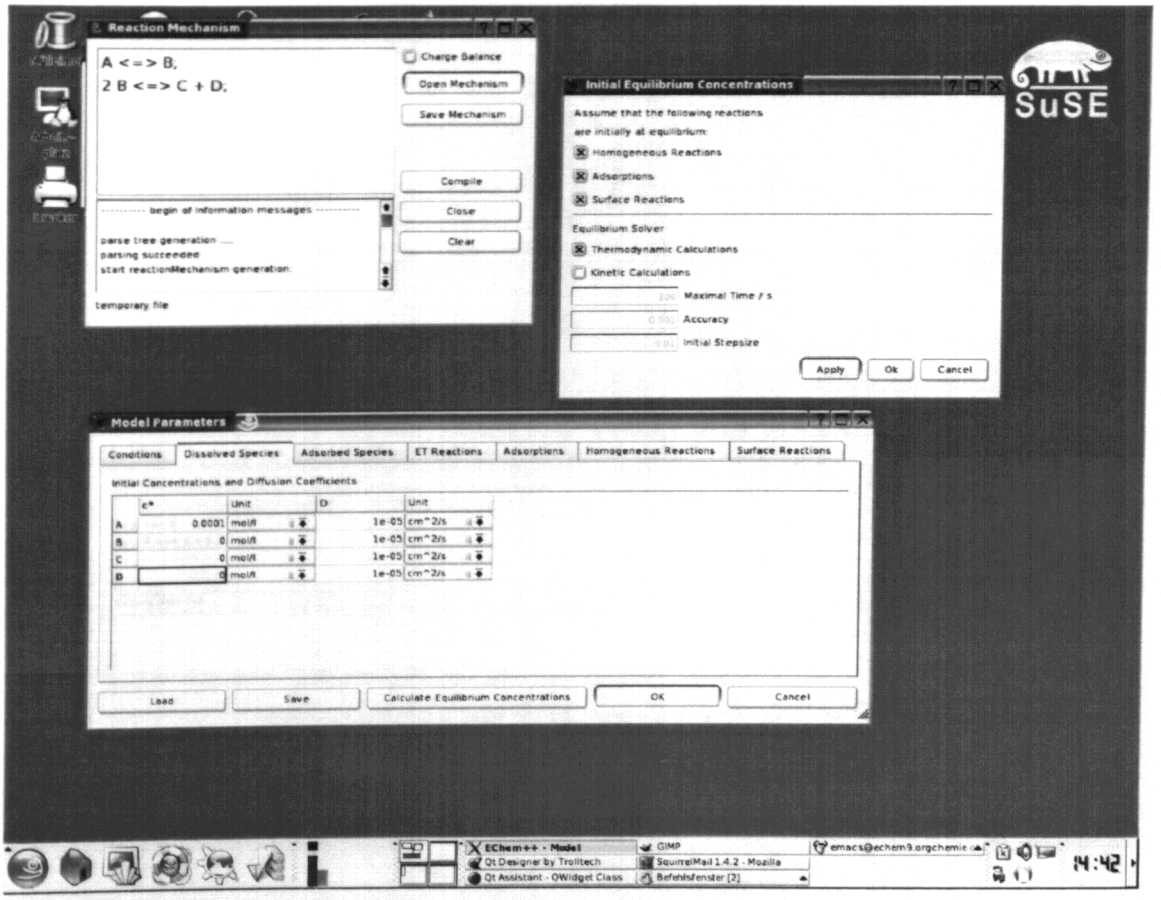

Abbildung 5.6: *Wahl des Equilibrium-Solvers (oberes rechtes Fenster)*

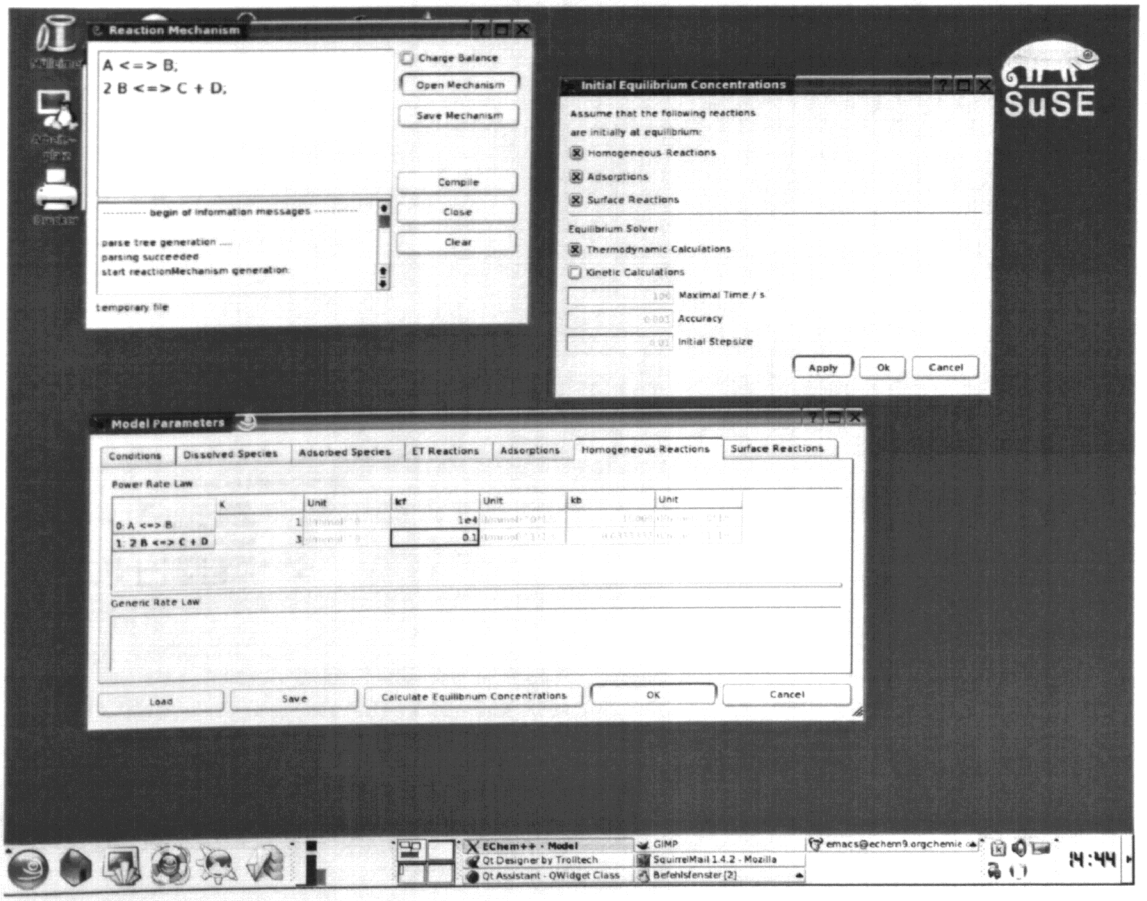

Abbildung 5.7: *Wahl der Reaktionsparameter (unteres Fenster)*

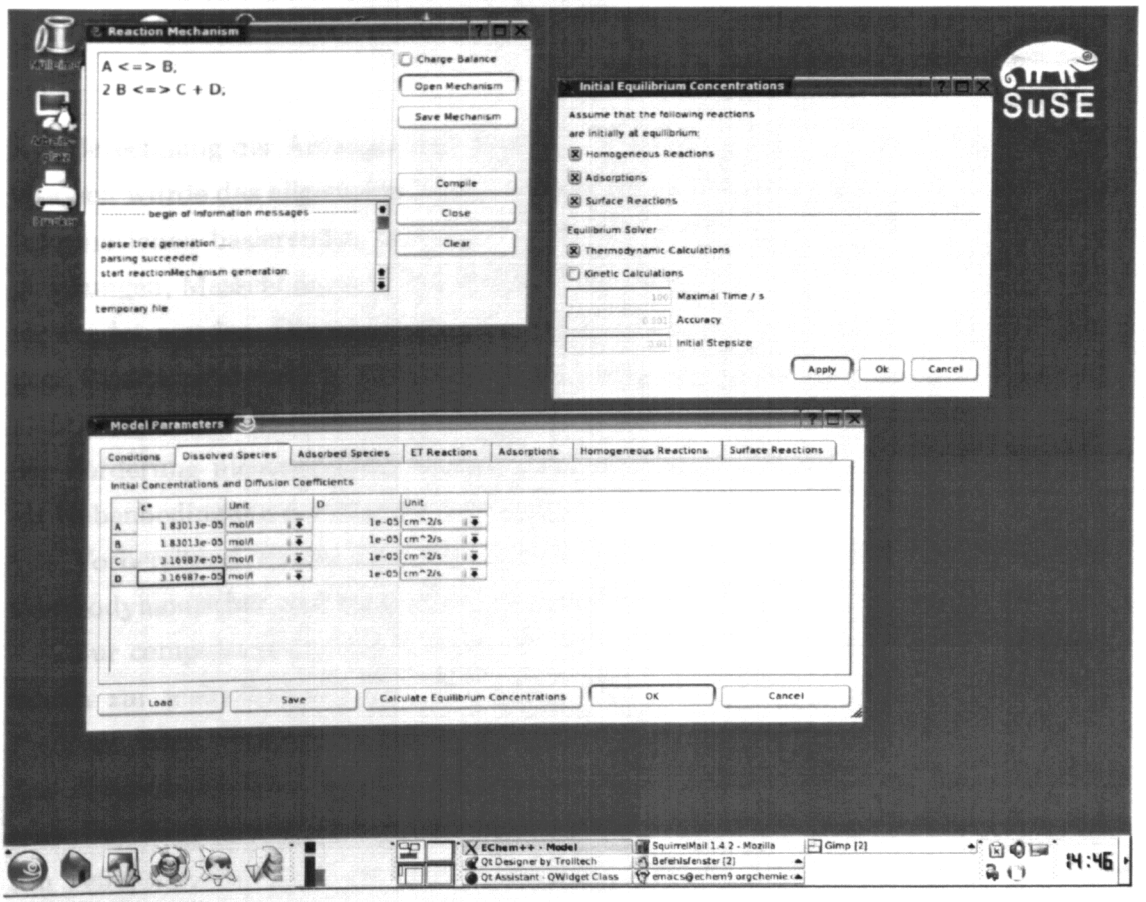

Abbildung 5.8: *Darstellung der Ergebnisse (unteres Fenster)*

Kapitel 6

Zusammenfassung

Zur Berechnung der Anfangs- und Randwertbedingungen bei elektrochemischen Simulationen wurde das allgemeine Equilibrium-Problem in ein nur auf thermodynamischen Informationen basierendes, mathematisches Modell übersetzt, ohne dass Atombilanzgleichungen, Massenbilanzen, Standardbildungsenthalpien oder ähnliche Informationen verwendet wurden. Die wesentliche Idee dieses Modells ist es, jeder Reaktion eine eigene Reaktionslaufzahl ξ_i zuzuordnen. Das Modell stellt dann ein mehrdimensionales, nichtlineares Gleichungssystem hoher Ordnung dar — die "Equilibriumfunktion". Aus der Forderung nichtnegativer Konzentrationen folgt ein lineares Ungleichungssystem als Nebenbedingung der Equilibriumfunktion.

Vorbereitend wurde die eindeutige Lösbarkeit des Equilibrium-Problems mittels thermodynamischer und kinetischer Argumentation bewiesen.

Zur computergestützten Lösung des Systems mussten numerische Näherungsverfahren zur mehrdimensionalen Nullstellensuche herangezogen werden, da für dieses Problem keine direkten Verfahren existieren. Als mächtigstes Verfahren erwies sich das *Broydenverfahren*, eine Verallgemeinerung des eindimensionalen Sekantenverfahrens. Das Broydenverfahren steigert die Geschwindigkeit des Algorithmus, da es deutlich weniger Funktionsauswertungen benötigt, als das in Konkurrenz stehende Newton-Raphson-Verfahren.

Mittels einer Schrittweitensteuerung wurde der Konvergenzbereich des Verfahrens stark vergrößert. Gemeinsam mit einer dynamischen Generierung von Startwerten und speziellen Beschleunigungsmethoden kann in den meisten Fällen der Anwendung eine akzeptable Rechenzeit garantiert werden. Das beschriebene Verfahren wurde dann objektorientiert in die moderne Programmiersprache C++ übersetzt.

Die Projektanbindung geschah mit der Entwicklung der virtuellen Klasse EquilibriumSolver als Schnittstelle zum Ecco-Compiler des Echem++-Projekts. Die davon abgeleiteten Klassen KineticEquSolver und ThermodynamicEquSolver lösen das

Equilibrium-Problem nach Wahl des Anwenders, wobei in letzterer letztlich das in dieser Arbeit beschriebene Verfahren implementiert ist. Es unterscheidet sich von dem Verfahren der Klasse KineticEquSolver im Wesentlichen dadurch, dass keine Geschwindigkeitskonstanten mehr vorgegeben werden müssen und Lösungen von Differentialgleichungen unnötig sind.

Kapitel 7

Anhang

Sourcecode

Der Sourcecode steht unter der Internetadresse:

http://echempp.sourceforge.net

auch online zur Verfügung.
Die Klasse EquilibriumFunction (Teile des Codes aus [24]):

```
void
EquilibriumFunction::fMin( std::vector<double>& x,
   const std::vector<std::vector<double> >& A,
   const std::vector<double> initialconc,
   const std::vector<double> Kc,
   double& f)
{
   double value;
   double basisvalue;
   f = 0;
   for (unsigned int reacno = 0; reacno < A.size(); ++reacno) {
     value = 1;
     for (unsigned int i = 0; i < A[0].size(); ++i)  {
       if (A[reacno][i] != 0.0) {
        basisvalue = initialconc[i];
        for (unsigned int j = 0; j < A.size(); ++j) {
                basisvalue += x[j] * A[j][i];
                }
        value *= pow(basisvalue, A[reacno][i]);
       }
     }
        value = value - Kc[reacno];
        f += value * value;
    }
    f *= 0.5;
}

void
EquilibriumFunction::funcVector( std::vector<double>& x,
  const std::vector<std::vector<double> >& A,
  const std::vector<double>& initialconc,
  const std::vector<double>& Kc,
```

```
  std::vector<double>& fvec)
{
   double value;
   double basisvalue;
   fvec.clear();
   for (unsigned int reacno = 0; reacno < A.size(); ++reacno) {
     value = 1;
     for (unsigned int i = 0; i < A[0].size(); ++i)  {
        basisvalue = initialconc[i];
        for (unsigned int j = 0; j < A.size(); ++j) {
                basisvalue += x[j] * A[j][i];
                }
        if (A[reacno][i] != 0.0) value *= pow(basisvalue, A[reacno][i]);
        }
        fvec.push_back(value - Kc[reacno]);
    }
}

void
EquilibriumFunction::jacobian( std::vector<double>& x,
       std::vector<double>& fvec,
       std::vector<std::vector<double> >& fjac,
       const std::vector<std::vector<double> >& A,
       const std::vector<double> initialconc,
       const std::vector<double> Kc)

{
   double value, basisvalue;
   const double EPS = 1.0e-8;
   double h, temp;
   unsigned int n = x.size();
   std::vector<double> f, helpvec;
   fjac.clear();
   for (unsigned int j = 0; j < n; ++j) {
        temp = x[j];
        h = EPS*fabs(temp);
        if (h == 0.0) h = EPS;
        x[j] = temp+h;
        h = x[j] - temp;
      for (unsigned int reacno = 0; reacno < A.size(); ++reacno) {
             value = 1;
             for (unsigned int i = 0; i < A[0].size(); ++i)  {
                    basisvalue = initialconc[i];
                    for (unsigned int k = 0; k < A.size(); ++k) {
                         basisvalue += x[k] * A[k][i];
                            }
               if (A[reacno][i] != 0.0)
                value *= pow (basisvalue, A[reacno][i]);
               }
               f.push_back(value - Kc[reacno]);
        }
        x[j] = temp;
        for (unsigned int i = 0; i < n; ++i)
                helpvec.push_back((f[i]-fvec[i])/h);
        fjac.push_back(helpvec);
        helpvec.clear();
        f.clear();
```

```
        }
    }
}
```

Die Klasse Broyden (Teile des Codes aus [24]):

```
void
Broyden::linesearch( std::vector<double>& xold,
 const double fold,
 std::vector<double>& g,
 std::vector<double>&p,
 std::vector<double>& x,
 double& f,
 const double stpmax,
 bool& check,
 const std::vector<std::vector<double> >& A,
 const std::vector<double>& initialconc,
 const std::vector<double>& Kc)

{
  bool debug = false;
  const double ALF=1.0e-4;
  const double TOLX = std::numeric_limits<double>::epsilon();
  int i;
  double a, alam, alam2, alamin, b, disc, f2;
  alam2 = 0.0;
  f2 = 0.0;
  double help;
  double rhs1, rhs2, slope, sum, temp, test, tmplam;
  EquilibriumFunction MaxIt;

  int n=xold.size();
  check = false;
  sum = 0.0;
  for (i = 0; i < n; ++i) sum += p[i]*p[i];
  sum = sqrt(sum);
  if (sum > stpmax)
        for (i = 0; i < n; ++i) p[i] *= stpmax/sum;
  slope = 0.0;
  for (i = 0; i < n; ++i) slope += g[i]*p[i];
  if (slope >= 0.0) if (debug) std::cout << "Roundoff problem in linesearch";
  test = 0.0;
  for (i = 0; i < n; ++i) {
        this->max(fabs(xold[i]),1.0,help);
        temp = fabs(p[i])/(help);
        if (temp > test) test = temp;
        }
  alamin = TOLX/test;
  alam = 1.0;
  for (unsigned int j = 0; j < 500; ++j) {
        for (i = 0; i < n; ++i) x[i] = xold[i]+alam*p[i];
        MaxIt.fMin(x,A,initialconc,Kc,f);
        if (alam < alamin) {
                for (i = 0; i < n; ++i) x[i] = xold[i];
                check = true;
                return;
                }
          else if (f <= fold+ALF*alam*slope) return;
```

```
        else {
                if (alam == 1.0) tmplam = -slope/(2.0*(f-fold-slope));
                else {
                        rhs1 = f-fold-alam*slope;
                        rhs2 = f2-fold-alam2*slope;
                        a = (rhs1 / (alam*alam)-rhs2/(alam2*alam2))/(alam-alam2);
                        b = (-alam2*rhs1/(alam*alam)+alam*rhs2/(alam2*alam2))/(alam-alam2);
                        if (a == 0.0) tmplam = -slope/(2.0*b);
                     else {
                                disc = b*b-3.0*a*slope;
                                if (disc < 0.0) tmplam = 0.5 * alam;
                                else if (b <= 0.0) tmplam = (-b+sqrt(disc))/(3.0*a);
                                else tmplam = -slope/(b+sqrt(disc));
                          }
                     if (tmplam > 0.5*alam) tmplam = 0.5 * alam;
                     }
        }
        alam2 = alam;
        f2 = f;
        this->max(tmplam, 0.1*alam, help);
        alam = help;
  }
}


void
Broyden::broydn( std::vector<double>& x,
 const std::vector<std::vector<double> >& A,
 const std::vector<double>& initialconc,
 const std::vector<double>& Kc,
 bool& check)
{
  bool debug = false;
  const int MAXITS = 200;
  const double EPS = std::numeric_limits<double>::epsilon();
  const double TOLF = 1.0e-8, STPMX = 100.0, TOLMIN = 1.0e-12;
  const double TOLX = EPS;
  bool restrt, sing, skip;
  int i, its, j, k;
  double den, f, fold, stpmax, sum, temp, test, help;
  int n = x.size();
  std::vector<std::vector<double> > qt(n,n), r(n,n);
  std::vector<double> c(n), d(n), fvcold(n), g(n), p(n), s(n), t(n), w(n), xold(n), fvec;
  EquilibriumFunction Calc;
  Calc.fMin(x, A, initialconc, Kc, f);
  Calc.funcVector(x, A, initialconc, Kc, fvec);
  test = 0.0;
  for (i = 0; i < n; ++i) if (fabs(fvec[i]) > test) test = fabs(fvec[i]);
  if (test < 0.01 * TOLF) {
        check = false;
        return;
       }
  sum = 0.0;
  for (i = 0; i < n; ++i) sum += x[i]*x[i];
  this->max(sqrt(sum),double(n),help);
  stpmax = STPMX*help;
  restrt = true;
```

```
for (its = 1; its <= MAXITS; ++its) {
       if (restrt) {
              Calc.jacobian(x, fvec, r, A, initialconc, Kc);
              this->qrdcmp(r, c, d, sing);
              if (sing) if (debug) std::cout << "singular jacobian in broydn";
              for (i = 0; i < n; ++i) {
                     for (j = 0; j < n; ++j) qt[i][j] = 0.0;
                     qt[i][i] = 1.0;
                     }
              for (k = 0; k < n-1; ++k) {
                     if (c[k] != 0.0) {
                            for (j = 0; j < n; ++j) {
                                   sum = 0.0;
                                   for (i = k; i < n; ++i)
                                          sum += r[i][k]*qt[i][j];
                                   sum /= c[k];
                                   for (i = k; i < n; ++i)
                                         qt[i][j] -= sum*r[i][k];
                                   }
                            }
                     }
              for (i = 0; i < n; ++i) {
                     r[i][i] = d[i];
                     for (j = 0; j < i; ++j) r[i][j] = 0.0;
                     }
              }

       else {
              for (i = 0; i < n; ++i) s[i] = x[i] - xold[i];
              for (i = 0; i < n; ++i) {
                     sum = 0.0;
                     for (j = i; j < n; ++j) sum += r[i][j] * s[j];
                     t[i] = sum;
              }
              skip = true;
              for (i = 0; i < n; ++i) {
                     sum = 0.0;
                     for (j = 0; j < n; ++j) sum += qt[j][i]*t[j];
                     w[i] = fvec[i]-fvcold[i]-sum;
                     if (fabs(w[i]) >= EPS * (fabs(fvec[i])+fabs(fvcold[i]))) skip = false;
                     else w[i] = 0.0;
                     }
              if (!skip) {
                     for (i = 0; i < n; ++i) {
                            sum = 0.0;
                            for (j = 0; j < n; ++j) sum += qt[i][j]*w[j];
                            t[i] = sum;
                           }
                     den = 0.0;
                     for (i = 0; i < n; ++i) den += s[i]*s[i];
                     for (i = 0; i < n; ++i) s[i] /= den;
                     this->qrupdt(r, qt, t, s);
                     for (i = 0; i < n; ++i) {
                            if (r[i][i] == 0.0) if (debug) std::cout << "r singular";
                            d[i] = r[i][i];
                            }
                     }
```

```
                }
        for (i = 0; i < n; ++i) {
                sum = 0.0;
                for (j = 0; j < n; j++) sum += qt[i][j]*fvec[j];
                p[i] = -sum;
                }
        for (i = n-1; i >= 0; --i) {
                sum = 0.0;
                for (j = 0; j <= i; ++j) sum -= r[j][i]*p[j];
                g[i] = sum;
                }
        for (i = 0; i < n; ++i) {
                xold[i] = x[i];
                fvcold[i] = fvec[i];
                }
        fold = f;
        this->rsolv (r,d,p);
        linesearch (xold, fold, g, p, x, f, stpmax, check, A, initialconc, Kc);
        Calc.funcVector(x, A, initialconc, Kc, fvec);
        test = 0.0;
        for (i = 0; i < n; ++i)
                if (fabs(fvec[i]) > test) test = fabs(fvec[i]);
        if (test < TOLF) {
                check = false;
                return;
                }
        if (check) {
                if (restrt) return;
                else {
                        test = 0.0;
                        this->max(f,0.5*n,den);
                        for (i = 0; i < n; ++i) {
                                this->max(fabs(x[i]),1.0,help);
                                temp = fabs(g[i])*help/den;
                                if (temp > test) test = temp;
                                }
                        if (test < TOLMIN) {
                                return;
                                }
                        else restrt = true;
                        }
                }
        else {
                restrt = false;
                test = 0.0;
                for (i = 0; i < n; ++i) {
                        this->max(fabs(x[i]),1.0,help);
                        temp = (fabs(x[i]-xold[i]))/help;
                        if (temp > test) test = temp;
                        }
                if (test < TOLX) return;
        }
  }
  if (debug) std::cout << "MAXITS exceeded in broydn";
  return;
}
```

```
void
Broyden::solve( const std::vector<std::vector<double> >& A,
 const std::vector<double>& initialconc,
 const std::vector<double>& Kc,
 std::vector<double>& equconc)
{
  bool debug = false;
  bool startcheck = false;
  bool check = false;
  bool root;
  const double tolerance = 1e-08;
  double XbracketMAX = 0;
  const unsigned int MAXrestarts = 100;
  double dynamic;
  double Xbracket;
  double help;
  std::vector<double> x;
  std::vector<double> testvec;
  srand(time(NULL));          // initializes rand()
  EquilibriumFunction test;

  for (unsigned int i = 0; i < A[0].size(); ++i)
       XbracketMAX += initialconc[i];
  for (unsigned int restarts = 0; restarts < MAXrestarts; ++restarts) {
    dynamic = XbracketMAX * (MAXrestarts - restarts);
       for (double loop = dynamic; loop > 0; --loop) {
              Xbracket = XbracketMAX - (XbracketMAX * loop) / dynamic + 0.1;
              root = true;
              x.clear();
              for (unsigned int i = 0; i < A.size(); ++i) {
                     x.push_back(rand()%100000 / (100000.0 / (2 * Xbracket)) - Xbracket);
                     }
              startcheck = true;
              for (unsigned k = 0; k < A[0].size(); ++k) {
                                   help = 0;
                                   for (unsigned l = 0; l < testvec.size(); ++l)
                                          help += x[l] * A[l][k];
                                   help += initialconc[k];
                                   if (help < 0.0) {
                                          startcheck = false;
                                          k = A[0].size();
                                          }
                                   }
             if (startcheck) {
                     broydn(x, A, initialconc, Kc, check);
                     if (check == false) {
                            test.funcVector(x, A, initialconc, Kc, testvec);
                            for (unsigned int j = 0; j < testvec.size(); ++ j) {
                                   if (fabs(testvec[j]) > tolerance) {
                                                 root = false;
                                                 j = testvec.size();
                                                 }
                                   }
                          if (root) {
                                 equconc.clear();
```

```
                                        for (unsigned k = 0; k < A[0].size(); ++k) {
                                                help = 0;
                                                for (unsigned l = 0; l < testvec.size(); ++l)
                                                      help += x[l] * A[l][k];
                                                help += initialconc[k];
                                                if (help > 0.0) equconc.push_back(help);
                                                else {  root = false;
                                                if (debug) std::cout << "ROOT NOT ACCEPTED";
                                                        k = A[0].size();
                                                }
                                                }
                                        if (root) return;
                                        }
                                }
                        }
                }
        }
    }
  equconc.clear();
  for (unsigned k = 0; k < A[0].size(); ++k) {
        equconc.push_back(0.0);
        }
}


void Broyden::Initial(const std::vector<std::vector<double> >& A,
                      const std::vector<double>& initconc,
                      const std::vector<double>& Kc,
                      std::vector<double>& equconc)

{
   std::vector<double> partialEqu;
   std::vector<unsigned int> blockROWindex;
   std::vector<bool> usedrow;
   std::vector<unsigned int> blocksize, blockCOLindex;
   std::vector<double> helpvec, equhelp, inithelp, Kchelp;
   std::vector<std::vector<double> > helpmat, helpmat2;
   unsigned int reaction, species, speciesnow;
   unsigned int rows = A.size();
   unsigned int cols = A[0].size();
   unsigned int blockcount = 1;
   unsigned int h = 0;

   equconc.assign(cols,0);
   usedrow.assign(rows,true);
   blockCOLindex.assign(cols,0);
   blockROWindex.push_back(0);
   blocksize.push_back(1);
   helpmat.push_back(A[0]);
   Kchelp.clear();
   Kchelp.push_back(Kc[0]);

   for (species = 0; species < cols; ++species)
        if (A[0][species] != 0) blockCOLindex[species] = blockcount;
   for (unsigned int blockcount = 1; blockcount <= rows; ++blockcount) {

   for (unsigned int tests = 1; tests <= rows; ++tests) {
```

```
  for (unsigned int rownow = 1; rownow < rows; ++rownow) {
   for (unsigned int loop = 1; loop < rows; ++loop) {
    for (reaction = 1; reaction < rows; ++reaction) {
     if (usedrow[reaction]) {
      for (species = 0; species < cols; ++species) {
        if ((A[reaction][species]) != 0 && (blockCOLindex[species] == blockcount)) {
          usedrow[reaction] = false;
          helpmat.push_back(A[reaction]);
          Kchelp.push_back(Kc[reaction]);
          blockROWindex.push_back(reaction);
          for (speciesnow = 0; speciesnow < cols; ++speciesnow) {
              if (A[reaction][speciesnow] != 0) blockCOLindex[speciesnow] = blockcount;
          }
          species = cols;
        }
      }
     }
    }
   }
  }
 }
 for (unsigned int x = 0; x < helpmat.size(); ++x) {
      helpvec.clear();
      for (unsigned int y = 0; y < cols; ++y)
              if (blockCOLindex[y] == blockcount) {
                      helpvec.push_back(helpmat[x][y]);
              }
      if (helpvec.size() != 0) helpmat2.push_back(helpvec);
      }
 for (unsigned int y = 0; y < cols; ++y)
              if (blockCOLindex[y] == blockcount) {
                      inithelp.push_back(initconc[y]);
              }
 this->solve(helpmat2, inithelp, Kchelp, equhelp);
 h = 0;
 for (unsigned int y = 0; y < cols; ++y)
              if (blockCOLindex[y] == blockcount) {
                      equconc[y]=equhelp[h];
                      h += 1;
              }
 Kchelp.clear();
 helpmat2.clear();
 inithelp.clear();
 for (unsigned int x = 1; x < rows; ++x)
      if (usedrow[x]) {
                      blockROWindex.push_back(x);
                      helpmat.clear();
                      helpmat.push_back(A[x]);
                      Kchelp.push_back(Kc[x]);
                      usedrow[x]=false;
                      for (species = 0; species < cols; ++species)
                       if (A[x][species] != 0) blockCOLindex[species] = blockcount + 1;
                      x = rows;
                      }
 if (blockROWindex.size() == rows) blockcount = rows + 1;
}
for (unsigned int y = 0; y < cols; ++y) if (equconc[y] == 0) equconc[y] = initconc[y];
```

```
}


double
Broyden::sign(const double& a, const double& b)
{
  return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);
}


void
Broyden::qrdcmp( std::vector<std::vector<double> >& a,
      std::vector<double>& c,
      std::vector<double>& d,
      bool& sing)
{
  int i, j, k;
  double scale, sigma, sum, tau, help;
  int n = a.size();
  EquilibriumFunction Calc;
  sing = false;

  for (k = 0; k < n-1; ++k) {
    scale = 0.0;
    for (i = k; i < n; ++i) {
        this->max(scale, fabs(a[i][k]), help);
        scale = help;
        }
    if (scale == 0.0) {
        sing = true;
        c[k]=d[k]=0.0;
        }
    else {
        for (i = k; i < n; ++i) a[i][k] /= scale;
        sum = 0.0;
        for (i = k; i < n; ++i) sum += a[i][k] * a[i][k];
        sigma = sign(sqrt(sum), a[k][k]);
        a[k][k] += sigma;
        c[k] = sigma * a[k][k];
        d[k] = -scale*sigma;
        for (j = k+1; j < n; ++j) {
                sum = 0.0;
                for (i = k; i < n; ++i) sum += a[i][k]*a[i][j];
                tau = sum / c[k];
                for (i = k; i < n; ++i) a[i][j] -= tau * a[i][k];
                }
        }
    }
  d[n-1] = a[n-1][n-1];
  if (d[n-1] == 0.0) sing = true;
}

void
Broyden::rotate( std::vector<std::vector<double> >& r,
      std::vector<std::vector<double> >& qt,
      const int i,
      const double a,
```

```
        const double b )
{
  int j;
  double c, fact, s, w, y;
  int n = r.size();
  if (a == 0.0) {
        c = 0.0;
        s = (b >= 0.0 ? 1.0 : - 1.0);
  }
  else if (fabs(a) > fabs(b)) {
        fact = b/a;
        c = sign(1.0/sqrt(1.0+(fact*fact)),a);
        s = fact * c;
        }
        else {
                fact = a/b;
                s = sign(1.0/sqrt(1.0 + (fact*fact)),b);
                c = fact*s;
        }
  for (j = i; j < n; ++j) {
        y = r[i][j];
        w = r[i+1][j];
        r[i][j] = c*y-s*w;
        r[i+1][j] = s*y+c*w;
        }
  for (j = 0; j < n; ++j) {
        y = qt[i][j];
        w = qt[i+1][j];
        qt[i][j] = c*y-s*w;
        qt[i+1][j] = s*y+c*w;
        }
}

void
Broyden::qrupdt( std::vector<std::vector<double> >& r,
      std::vector<std::vector<double> >& qt,
      std::vector<double>& u,
      std::vector<double>& v )
{
  int i, k;
  int n = u.size();
  for (k = n-1; k >= 0; --k)    // find nonzero element
       if (u[k] != 0.0) break;
  if (k < 0) k = 0;

  for (i = k-1; i >= 0; --i) {
       rotate(r, qt, i, u[i], -u[i+1]);
       if (u[i] == 0.0)
               u[i] = fabs(u[i+1]);
       else if (fabs(u[i]) > fabs(u[i+1]))
               u[i] = fabs(u[i])*sqrt(1.0+u[i+1]*u[i+1]/(u[i]*u[i]));
               else u[i] = fabs(u[i+1])*sqrt(1.0+u[i]*u[i]/(u[i+1]*u[i+1]));
       }
  for (i = 0; i < n; ++i) r[0][i] += u[0]*v[i];
  for (i = 0; i < k; ++i)
       rotate(r, qt, i, r[i][i], -r[i+1][i]);
}
```

```
void
Broyden::rsolv( std::vector<std::vector<double> >& a,
     std::vector<double>& d,
     std::vector<double>& b )
{
  int i, j;
  double sum;
  int n = a.size();
  b[n-1] /= d[n-1];
  for (i = n-2; i >= 0; --i) {
        sum = 0.0;
        for (j = i+1; j < n; ++j) sum += a[i][j] * b[j];
        b[i] = (b[i] - sum) / d[i];
        }
}

void
Broyden::max(double x, double y, double& z)

{
  if (x >= y) z = x;
  else z = y;
}
}
```

Literaturverzeichnis

[1] D. Britz, Digital Simulation in Electrochemistry, 2. Auflage, Springer-Verlag, 1988

[2] K. Ludwig, L. Rajendran und B. Speiser, Echem++ - An Object-Oriented Problem Solving Environment for Electrochemistry Part 1. A C++ class collection for electrochemical excitation functions, J. Electroanal. Chem., 2004, 568, 203

[3] K. Ludwig und B. Speiser, Echem++ — An Object-Oriented Problem Solving Environment for Electrochemistry Part 2. The kinetic facilities of Ecco — A Compiler for (Electro-)Chemistry. J. Chem. Inf. Comput. Sci., 2004, 44, 2051

[4] B. Speiser in A. J. Bord und I. Rubinstein, Electroanal. Chem., 1996, 19, 1-108

[5] E. Westall, J. Zachary und F. Morel, MineQL — A Computer Program for the Calculation of the Chemical Equilibrium Composition of Aqueous Systems, Oregon State University 1998

[6] E. Weltin, Are the Equilibrium Compositions Uniquely Determined by the Initial Compositions? — Property of the Gibbs Free Energy Function, J. Chem. Educ., 1995, 72, 508

[7] L. Bieniasz, Automatic derivation of the governing equations that describe a transient electrochemical experiment, given a reaction mechanism of arbitrary complexity, J. Electroanal. Chem., 1996, 406, 33

[8] B. Stroustrup, The C++ Programming Language, Addison-Wesley 1997

[9] G. Wedler, Lehrbuch der Physikalischen Chemie, 4. Auflage, Wiley Weinheim 1997

[10] C. Czeslik, H. Seemann, R. Winter — Basiswissen Physikalische Chemie, Teubner-Verlag Stuttgart 2001

[11] P. Atkins, Physikalische Chemie, 2. Auflage, Wiley Weinheim 1996

[12] W. Moore, Grundlagen der physikalischen Chemie, De Gruyter Berlin 1990

[13] H. Weingärtner, Chemische Thermodynamik, Teubner-Verlag Stuttgart 2003

[14] M. Wolff, O. Gloor und C. Richard, Analysis Alive, Birkhäuser-Verlag Berlin 1998

[15] K. Königsberger, Analysis 1, 6. Auflage, Springer Berlin 003

[16] H. Anton, Lineare Algebra, Spektrum-Verlag Heidelberg 1998

[17] V. Batyrev, Skriptum zur Vorlesung Lineare Algebra I, Universität Tüebingen 1998

[18] J. Stoer, Numerische Mathematik 1, 8. Auflage, Springer Berlin 1999

[19] M. Rudolph, Email an B. Speiser vom 19. April 2005

[20] E. Weltin, A numerical method to calculate equilibrium concentrations for single-equation systems, J. Chem. Educ., 1991, 68, 468

[21] E. Weltin, Calculating equilibrium concentrations: Competing, couples, and catalytic reactions, J. Chem. Educ., 1992, 69, 393

[22] O. Forster, Analysis 2, 5. Auflage, Wiley Weinheim 1984

[23] J. Faires und R. Burden, Numerische Methoden, Spektrum-Verlag Heidelberg 1994

[24] W. Press, S. Teukolsky, W. Vetterling und B. Flannery, Numerical Recipes in C++, The Art of Scientific Computing, 2. Auflage Cambridge 2002

[25] W. Oevel, Einführung in die Numerische Mathematik, Spektrum-Verlag Heidelberg 1996

[26] R. Plato, Numerische Mathematik kompakt, Vieweg-Verlag Braunschweig 2000

[27] D. Broyden, Math. of Comp., 1965, 19, 577

[28] D. Dennis jr. und R. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, R.B. 1996

[29] H. Schwarz, Numerische Mathematik, Teubner-Verlag Stuttgart 1993

[30] J. Heistermann, Genetische Algorithmen, Theorie und Praxis evolutionärer Optimierung, Teubner-Verlag Stuttgart 1994

[31] Das Echem++ Projekt findet sich unter: http://echempp.sourceforge.net

[32] H. Yserentant, Skriptum zur Vorlesung Numerische Mathematik I, Universität Tübingen 2001

[33] B. Aulbach, Gewöhnliche Differentialgleichungen, Spektrum-Verlag Heidelberg 1997