AMC: Ein Werkzeug zum automatischen Differenzieren von Fortran-Programmen

Ralf Giering

Zusammenfassung

Automatisches Differenzieren ist eine Technik, um ein Programm zur Berechnung von Ableitungen zu generieren. Sie basiert auf der Tatsache, daß jedes noch so komplizierte Computerprogramm eine Sequenz von arithmetischen Operatoren und elementaren Funktionen ausführt. Die wiederholte Anwendung der Kettenregel des Differenzierens auf diese Operatoren ermöglicht die Berechnung von Ableitungen mit Maschinengenauigkeit. Der AMC konstruiert für ein gegebenes Fortran-Programm zur Auswertung einer Funktion ein Programm zur Berechnung ihrer Ableitungen.

1 Einleitung

Die Berechnung von Ableitungen ist Teil vieler Computerprogramme. Die automatische Generierung dieser Programmteile hat mehrere Vorteile. Sie erspart zeitaufwendiges Programmieren und ist weniger fehlerträchtig. Da die Berechnung von Ableitungen oft einen Großteil der Rechenzeit beansprucht, ist es günstig, hierfür effizienten Code einzusetzen. Oft wird die Entwicklung von entsprechenden Programmteilen vermieden, indem die Ableitungen durch Differenzenquotienten approximiert werden. Dies hat zwei Nachteile. Einerseits benötigt die Berechnung von Differenzenquotienten mehr Rechenzeit und andererseits liefert sie eben nur eine Näherung.

Ein numerisches Programm zur Auswertung einer Funktion kann betrachtet werden als eine Verkettung von vielen einzelnen Funktionen, die jeweils eine Anweisung repräsentieren. Sind diese Funktionen differenzierbar, kann die Verkettung durch systematisches Anwenden der Kettenregel differenziert werden. Das resultierende (assoziative) Produkt der einzelnen Funktionalmatrizen kann in beliebiger Reihenfolge ausgewertet werden. Zwei naheliegende Möglichkeiten sind der Vorwärtsmodus und der Rückwärtsmodus. Eine Einführung in das Differenzieren von Algorithmen wird in Kap. 2 gegeben.

Der Adjoint Model Compiler (AMC) generiert aus einem Fortran-Programm zur Auswertung einer Funktion ein Fortran-Programm zur Berechnung ihrer ersten Ableitungen. Je nach Wahl von Optionen arbeitet der generierte Code im Vorwärts- oder im Rückwärtsmodus. Die internen Abläufe werden in Kap. 4 skizziert.

Es gibt bereits mehrere Implementierungen zum automatischen Differenzieren von Algorithmen. Einen allerdings nicht mehr ganz aktuellen Überblick gibt Juedes (1991). Nur einige davon sind in der Lage ein gegebenes Fortran-Programm zu differenzieren. Die möglichst kurze Rechenzeit des erzeugten Codes ist in der Praxis ein entscheidendes Kriterium, das im wesentlichen nur die Softwarepakete ADIFOR (Bischof et al. 1994), Odyssée (Rostaing et al.

1993) und AMC (Giering 1992) erfüllen. Alle drei generieren Fortran-Code zur Berechnung der Ableitungen, wobei ADIFOR bis jetzt auf den Vorwärtsmodus beschränkt ist. Für die Berechnung von Gradienten einer skalarwertigen Funktion ist jedoch der Rückwärtsmodus meistens wesentlich effizienter. Die dem AMC zugrundeliegenden Konzepte zur Implementierung des Rückwärtsmodus werden in Kap. 3 beschrieben.

Es gibt viele Fragestellungen in der Wissenschaft, welche die Berechnung von Ableitungen erfordern. Zu den effektivsten Verfahren zur Minimierung einer sklarwertigen Funktion gehören die Gradientenverfahren. Im Zusammenhang mit der Simulation von Systemen durch numerische Modelle gibt es weitere Anwendungen. Die Assimilierung von Beobachtungsdaten in ein Simulationsmodell kann als ein Minimierungsproblem für eine skalarwertige Funktion aufgefaßt werden. Für die Sensitivitätsanalyse werden das tangentlineare und das adjungierte Modell eingesetzt. Sie berechnen das Produkt der Funktionalmatrix mit einem Vektor von rechts bzw. links. Die Hintereinanderanwendung beider Modelle auf einen Vektor ist Teil eines Algorithmus zur Bestimmung der führenden instabilen Moden. Kap. 5 gibt einen Überblick über bisherige Anwendungen des AMC in den Bereichen Ozeanographie und Meteorologie.

Die Berechnung der Helmholtzfunktion wird oftmals als Beispiel zur Demonstration und zum Vergleich von konstruiertem Code zur Berechnung von Ableitungen herangezogen. Im Anhang ist ein Unterprogramm zur Berechnung dieser Funktion und das vom AMC konstruierte Unterprogramm zur Berechnung des Gradienten im Rückwärtsmodus gezeigt.

2 Differenzieren von Algorithmen

Es sei

$$\begin{array}{rcl} \mathcal{H} : I\!R^n & \rightarrow & I\!R^m \\ x & \mapsto & y \end{array} \tag{1}$$

eine Abbildung, gegeben durch einen numerischen Algorithmus, der aus den unabhängigen Variablen x die abhängigen Variablen y berechnet. Die explizite Darstellung von $\mathcal{H}$ kann sehr kompliziert sein, da der Algorithmus möglicherweise lang und komplex ist. Es ist jedoch häufig möglich, den Algorithmus in einzelne Schritte zu zerlegen, die ihrerseits eine einfache analytische Darstellung haben:

$$\begin{array}{rcll} \mathcal{H}^l : I\!R^{n_{l-1}} & \rightarrow & I\!R^{n_l} & (l = 1, \ldots, k) \\ z^{l-1} & \mapsto & z^l \ . & \end{array} \tag{2}$$

Dann ist $\mathcal{H}$ eine Verkettung dieser Funktionen $\mathcal{H}^l$

$$\mathcal{H} = \bigodot_{l=1}^{k} \mathcal{H}^l \ , \tag{3}$$

mit $n_0 = n$, $z^0 = x$, $n_k = m$ und $z^k = y$. Für eine differenzierbare Funktion $\mathcal{H}$ ist die Funktionalmatrix A an der Stelle x_0 definiert durch:

$$A_{ij}(x_0) := \left. \frac{\partial \mathcal{H}_i(x)}{\partial x_j} \right|_{x = x_0} \qquad (i = 1, \ldots, m \ ; \ j = 1, \ldots, n) \ . \tag{4}$$

Nach Anwendung der Kettenregel auf (3) folgt:

$$A(x_0) = \prod_{l=1}^{k} \left. \frac{\partial \mathcal{H}^l}{\partial z^{l-1}} \right| z^{l-1} = z_0^{l-1} \quad , \tag{5}$$

wobei

$$z_0^{l-1} \quad = \quad \left(\bigodot_{j=1}^{l-1} \mathcal{H}^j \right) (x_0) \qquad (l = 1, \ldots, m) \tag{6}$$

ein Zwischenergebnis ist. Die Auswertung des Matrixproduktes auf der rechten Seite von (5) erfordert $k-1$ Matrixmultiplikationen. Da diese Multiplikation eine assoziative Verknüpfung ist, gibt es $(k-1)!$ Möglichkeiten die Reihenfolge der Multiplikationen festzulegen. Die Rechenzeit für das Produkt hängt von der Reihenfolge der Multiplikationen ab, weil die Matritzen i.allg. unterschiedlich groß und dicht besetzt sind. Die Suche nach der Permutation mit der geringsten Rechenzeit ist ein NP-wertiges Problem[1] (Griewank und Reese 1991).

Zwei Strategien zur Auswertung heben sich besonders hervor. Im *Vorwärtsmodus* (forward mode) wird das Produkt in der gleichen Reihenfolge berechnet wie die Verkettung (3), d.h es wird zunächst $\frac{\partial \mathcal{H}^2}{\partial z^1} \cdot \frac{\partial \mathcal{H}^1}{\partial z^0}$ berechnet und dann das Ergebnis von links mit $\frac{\partial \mathcal{H}^3}{\partial z^2}$ multipliziert usw.. Bei dem anderen Extrem, dem *Rückwärtsmodus* (reverse mode), wird umgekeht verfahren, die Rechnung beginnt mit dem Produkt $\frac{\partial \mathcal{H}^k}{\partial z^{k-1}} \cdot \frac{\partial \mathcal{H}^{k-1}}{\partial z^{k-2}}$. Beide Methoden unterscheiden sich in der Größe der Matrizen, die als Zwischenergebnisse auftreten. Im ersten Fall haben diese immer n Spalten, im zweiten immer m Zeilen. Daher benötigt für $m > n$ der Vorwärtsmodus weniger Operationen als der Rückwärtsmodus und für $m < n$ ist es umgekehrt[2]. Ein wichtiger Unterschied zwischen beiden Methoden besteht in der Erfordernis von Zwischenergebnissen z^l. Im Vorwärtsmodus werden sie in der gleichen Reihenfolge benötigt, wie sie zur Berechnung der Funktion $\mathcal{H}$ anfallen. Es ist daher leicht möglich, Ableitungen und Funktion nebeneinander zu berechnen. Im Rückwärtsmodus dagegen werden die Zwischenergebnisse in der umgekehrten Reihenfolge gebraucht. Ihre Bereitstellung erfordert zusätzliche Anweisungen.

Für eine skalarwertige Funktion ($m = 1$) wird der Rückwärtsmodus meist mit weniger Operationen auskommen. Für diesen Fall nennt man den Rückwärtsmodus auch adjungierten Modus. Bei der Auswertung des Produkts (5) fallen hierbei immer nur Vektoren als Ergebnisse an.

Die skalarwertige Funktion J sei durch eine Verkettung von Funktionen $\mathcal{H}^l$ (2) gegeben:

$$J \quad = \quad \bigodot_{l=1}^{k} \mathcal{H}^l \quad , \tag{7}$$

wobei $n_k = m = 1$ ist. In der linearen Approximation von (6) hängt die Variation eines Zwischenergebnises δz^l von der Variation δx der unabhängigen Variablen x in folgender Weise ab:

$$\delta z^l \quad = \quad \frac{\partial}{\partial x} \left(\bigodot_{i=1}^{l} \mathcal{H}^i \right) (x) \Bigg|_{x \,=\, x_0} \delta x \quad . \tag{8}$$

Die Variationen aufeinanderfolgender Zwischenergebnisse sind durch

$$\delta z^l \quad = \quad \frac{\partial \mathcal{H}^l(z^{l-1})}{\partial z^{l-1}} \Bigg|_{z^{l-1} \,=\, z_0^{l-1}} \delta z^{l-1} \tag{9}$$

[1] Die Rechenzeit zur exakten Lösung eines NP-wertigen Problems steigt exponentiell mit seiner Dimension.

[2] Wenn die Matrizen schwach besetzt sind und diese Eigenschaft zur Reduktion der Operationen ausgenutzt wird, kommen noch andere Kriterien zum Tragen (Bischof und Hovland 1991).

miteinander verbunden, wobei $\delta z^0 = \delta x$ ist. Die adjungierte Variation $\delta^* z^l$ eines Zwischenergebnisses z^l ist nun definiert als die Ableitung der Funktion J bezüglich des Zwischenergebnisses:

$$\delta^* z^l := \nabla_{z^l} J(z^l)\Big|_{z^l = z_0^l} \quad . \tag{10}$$

Mit der Definition des Gradienten folgt:

$$\delta J = \left\langle \delta^* z^l , \delta z^l \right\rangle \quad . \tag{11}$$

Die adjungierten Werte beschreiben also den Einfluß des zugehörigen Zwischenergebnisses auf den Funktionswert. Da (11) für alle l gilt, folgt aus (9) und der Definition des adjungierten Operators:

$$\begin{aligned} \left\langle \delta^* z^{l-1} , \delta z^{l-1} \right\rangle &= \left\langle \delta^* z^l , \delta z^l \right\rangle \\ &= \left\langle \delta^* z^l , \left(\frac{\partial \mathcal{H}^l(z^{l-1})}{\partial z^{l-1}} \right)\Big|_{z^{l-1} = z_0^{l-1}} \delta z^{l-1} \right\rangle \\ &= \left\langle \left(\frac{\partial \mathcal{H}^l(z^{l-1})}{\partial z^{l-1}} \right)^*\Big|_{z^{l-1} = z_0^{l-1}} \delta^* z^l , \delta z^{l-1} \right\rangle \quad . \end{aligned}$$

Dies gilt für alle δz^{l-1} und damit:

$$\delta^* z^{l-1} = \left(\frac{\partial \mathcal{H}^l(z^{l-1})}{\partial z^{l-1}} \right)^*\Big|_{z^{l-1} = z_0^{l-1}} \delta^* z^l \quad , \tag{12}$$

oder komponentenweise:

$$\delta^* z_i^{l-1} = \sum_{j=1}^{n_l} \frac{\partial \mathcal{H}_j^l(z^{l-1})}{\partial z_i^{l-1}}\Big|_{z^{l-1} = z_0^{l-1}} \delta^* z_j^l \qquad \text{für alle } i = 1, \ldots, n_{l-1} \ . \tag{13}$$

Der letzte Schritt liefert den Gradienten der Funktion bzgl. der unabhängigen Variablen:

$$\delta^* z^0 = \delta^* x = \nabla_x J \ . \tag{14}$$

Gleichung (13) ist die grundlegende Beziehung für einen Schritt im Rückwärtsmodus. Sollen die Ableitungen einer nicht skalarwertigen Funktion berechnet werden, so ist jedem Zwischenergebnis ein Vektor von adjungierten Variablen zugeordnet.

3 Konstruktion adjungierten Codes

Die elementaren Schritte in einem Programm sind die Anweisungen. Ein numerisches Modell ist im wesentlichen aus wenigen Typen von Anweisungen aufgebaut:

- Zuweisungen
- bedingte Anweisungen
- Wiederholungsanweisungen
- Prozeduraufrufe

- Anweisungsfolgen

Die Konstruktion tangentlinearen Codes wurde bereits mehrfach beschrieben (Bischof et al. 1994, Rostaing et al. 1993). Das im AMC implementierte Verfahren weicht nur wenig davon ab. Für die wichtigste Anweisung in einem Programm, der Zuweisung, wird die Konstruktion des adjungierten Codes im folgenden beschrieben. Eine ausführliche Darstellung auch für andere Anweisungen geben Giering und Kaminski (1996). Doch zunächst sollen einige grundlegende Konzepte vorgestellt werden; sie ermöglichen das Aufstellen von Regeln und haben sich als hilfreich für die Entwicklung und Pflege von adjungierten Modellen herausgestellt.

3.1 Grundlegende Konzepte

Korrespondenz zwischen Variablen und adjungierten Variablen In einem Programm halten Variablen die Zwischenergebnisse des vorangehenden Abschnitts. Diese Variablen können jedoch beliebig oft mit weiteren Ergebnissen überschrieben werden. Es ist daher möglich, daß zwei Zwischenergebnisse z_i^l und z_j^p $(l \neq p)$ in der gleichen Variablen aber an verschiedenen Abschnitten der Programmabfolge gespeichert werden.

Der adjungierte Code berechnet die zu den Zwischenergebnisse z_i^l gehörigen adjungierten Werte $\delta^* z_i^l$. Es ist ausreichend, zu jeder Variablen im Code genau eine adjungierte Variable im adjungierten Code zu definieren. Sie hält die jeweiligen adjungierten Werte der Zwischenergebnisse der Variablen.

Unterscheidung zwischen aktiven und passiven Variablen Variablen, die von den unabhängigen Variablen abhängen und Einfluß auf die abhängigen Variablen haben, werden aktive Variablen genannt. Alle anderen Variablen sind passive Variablen. Nur Anweisungen, die aktive Variablen berechnen, müssen adjungiert werden.

Lokalität Lokalität ist gegeben, wenn die adjungierten Anweisungen innerhalb des adjungierten Codes diejenige Reihenfolge haben, die durch die Reihenfolge der Anweisungen des Codes festgelegt ist. Auch wenn dies Rechenzeit einsparen könnte, werden Anweisungen nicht zusammengefaßt. Es besteht somit eine direkte Korrespondenz zwischen Anweisungen und den zugehörigen adjungierten Anweisungen. Als direkte Konsequenz wird jedem Unterprogramm, welches aktive Variablen berechnet, genau ein adjungiertes Unterprogramm zugeordnet.

Modularität Jeder Teil des Codes soll weitgehend unabhängig von anderen Teilen adjungiert werden. Nun sind aber viele adjungierte Anweisungen von Zwischenergebnissen, d.h. von Werten der Variablen des ursprünglichen Codes abhängig. Die Variablen mögen in Indexausdrücken, Schleifengrenzen oder Bedingungen auftreten. Es können aber auch aktive Variablen in nichtlinearen Ausdrücken auf der rechten Seite von Zuweisungen sein. Ihre Werte werden für die Berechnung adjungierter Werte benötigt und müssen mit denjenigen im ursprünglichen Code übereinstimmen. Solche Variablen werden im folgenden *Bedarfsvariablen* genannt.

Modularität ist gegeben, wenn alle Bedarfsvariablen einer adjungierten Anweisung zu Beginn genau den Wert haben, den sie auch im Code vor der entsprechenden Anweisung hatten. Denn unter dieser Voraussetzung kann der adjun-

gierte Code einer Anweisung unabhängig von den vorhergehenden und nachfolgenden Anweisungen konstruiert werden. Ebenso ermöglicht es das Ableiten von einfachen Regeln. Um Modularität zu garantieren müssen eventuell zusätzliche Anweisungen im adjungierten Code eingefügt werden.

Übersichtliche Nomenklatur Es ist zur besseren Lesbarkeit des adjungierten Codes empfehlenswert, eine einprägsame Konvention zur adjungierten Namensgebung einzuhalten. Alle adjungierten Namen können sich z.B. aus Kennbuchstaben `AD` und dem Namen aus dem Code zusammensetzen. Der AMC verfährt nach dieser Konvention und stellt sicher, daß keine Kollisionen mit bereits vergebenen Namen auftreten. Alle Bedarfsvariablen bekommen im adjungierten Code den Namen, den sie auch im Code haben. Anweisungen zur erneuten Berechnung dieser Variablen im adjungierten Code können somit direkt aus dem Code übernommen werden. Auch dies erhöht die Lesbarkeit des adjungierten Codes.

3.2 Adjungierte Zuweisungen

Eine Zuweisung wird nur adjungiert, wenn sie den Wert einer aktiven Variablen verändert. Die zu berechnende aktive Variable und alle aktiven Variablen des Ausdrucks auf der rechten Seite, ausgenommen Variablen zur Berechnung von Indizes, bilden den Ein- und Ausgabevektor. Dabei sind mehrdimensionale Variablen mit gleichem Namen aber unterschiedlichen Indizierungen, als verschiedene Variablen zu behandeln, falls sie auf verschiedene Addressen verweisen. Der AMC vergleicht dazu die einzelnen Indexausdrücke. Für eine in allen Fällen richtige Entscheidung müßte jedoch eine Analyse der Indexausdrücke unter Berücksichtigung etwaiger äußerer Schleifen durchgeführt werden.

Die Zuweisung kann als ein Operator aufgefaßt werden, der auf den Eingangsvektor wirkt. Für eine allgemeine Zuweisung:

$$y \leftarrow f(x_1, .., x_n, y) \tag{15}$$

besteht dieser Vektor aus den Variablen $x_1, \ldots, x_n$ und y. Es sei f ein Ausdruck, der die Variablen $x_1, \ldots, x_n$ und möglicherweise auch die Variable der linken Seite y enthält.

Die tangentlineare Zuweisung dazu ist:

$$\delta y \leftarrow \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\, \delta x_i \; + \; \frac{\partial f}{\partial y}\, \delta y \;, \tag{16}$$

dabei sind δy und δx_i die entsprechenden Variationen. Ihre Koeffizienten bilden die erste Zeile der Funktionalmatrix des Operators. Alle anderen Zeilen bestehen aus Nullen, nur die Diagonale ist mit Einsen besetzt. Die adjungierte Matrix hiervon ist die transponierte Matrix. Aus letzterer lassen sich die adjungierten Zuweisungen ableiten:

$$\begin{aligned} \delta^* x_i &\leftarrow \delta^* x_i + \delta^* y \frac{\delta f}{\delta x_i} \qquad (i = 1, \ldots, n) \\ \delta^* y &\leftarrow \delta^* y \frac{\delta f}{\delta y} \;, \end{aligned}$$

wobei $\delta^* x_i$ und $\delta^* y$ die korrespondierenden adjungierten Variablen sind. Die Zuweisung an die adjungierte Variable $\delta^* y$ muß immer als letztes erfolgen, denn ihr alter Wert wird für die anderen adjungierten Zuweisungen benötigt.

Der AMC differenziert die Ausdrücke $\frac{\delta f}{\delta y}$ und $\frac{\delta f}{\delta x_i}$ symbolisch und vereinfacht sie. Zur Ausführung der adjungierten Zuweisungen müssen alle Variablen, die in den vereinfachten Ausdrücken auftreten, die Werte besitzen, die sie auch in dem Moment vor Ausführung der ursprünglichen Zuweisung hatten; sie sind Bedarfsvariablen dieser adjungierten Zuweisungen.

4 AMC

Der AMC ist ein Komandozeilenprogramm. Das zu differenziernde Unterprogramm und die unabhängigen und abhängigen Variablen werden durch Optionen spezifiziert. Je nach Optionen generiert der AMC adjungierte oder tangentlineare Unterprogramme. Außerdem werden gegebenenfalls Unterprogramme erstellt, die nur mit zusätzlichen Anweisungen zur Speicherung von Bedarfsvariablen versehen sind. Die Namen der Dateien, in welche die konstruierten Unterprogramme geschrieben werden, generiert der AMC aus den angegebenen Namen der Quellprogramme.

Nach Aufruf des AMC liest ein Scanner die Quellprogramme und führt eine lexikalische Analyse durch Abb. 1. Er erzeugt eine Folge von Grundsymbolen, die vom Parser einer syntaktischen Analyse unterzogen wird. Dabei werden Fehler erkannt und für alle Unterprogramme ein abstrakter Syntaxbaum aufgebaut. Eine anschließende semantische Analyse überprüft die wesentlichen Kontextbedingungen für Fortran-Programme und warnt den Benutzer falls nicht implementierte Sprachelemente auftreten.

Der AMC führt anhand der abstrakten Syntaxbäume der Unterprogramme eine Datenabhängigkeitsanalyse durch. Ausgehend von dem zu differenzierenden Unterprogramm werden alle Variablen bestimmt, deren Wert von denjenigen der unabhängigen Variablen abhängt. Desweiteren werden alle Variablen bestimmt, deren Wert Einfluß auf die Werte der abhängigen Variablen hat. Die Schnittmenge beider Variablenmengen enthält die aktiven Variablen. Unterprogramme, die aktive Variablen berechnen, werden gekennzeichnet.

Beginnend mit den Unterprogrammen auf unterster Ebene des Programmflußgraphen werden nacheinander die adjungierten Unterprogramme zu allen aktiven Unterprogrammen konstriert. Diese 'bottom-up'-Methode stellt sicher, daß zu einem Aufruf eines Unterprogramms die adjungierte Anweisung generiert werden kann, weil die Bedarfsvariablen des aufgerufenen adjungierten Unterprogramms bekannt sind. Für die Konstruktion tangentlinearer Unterprogramme wird ebenso verfahren, obwohl hier die Reihenfolge unbedeutend ist.

Die Erstellung eines einzelnen tangentlinearen Unterprogramms ist unterteilt in die Generierung der Argumentenliste, der Anweisungen und der Deklarationen. Für ein adjungiertes Unterprogramm wird zunächst die adjungierte Sequenz von Anweisungen konstruiert, denn erst danach sind ihre Bedarfsvariablen bekannt, und die Deklaration sowie die Argumentenliste kann erstellt werden. Vor den adjungierten Anweisungen erzeugt der AMC Code zur Initialisierung aller lokalen adjungierten Variablen. Die globalen adjungierten Variablen werden in einem eigens dazu generierten Unterprogramm initialisiert.

Abschließend schreibt der AMC auf Basis der abstrakten Syntaxbäume den Fortran-Code der adjungierten oder tangentlinearen Unterprogramme in die entsprechenden Dateien.

Der AMC akzeptiert verschiedene Direktiven. Sie übermitteln Informationen zur Konstruktion des tangentlinearen und adjungierten Codes. So können beispielsweise die Stellen im Programm festgelegt werden, an denen bestimmte

Bedarfsvariablen gespeichert werden sollen. Andere Direktiven geben Informationen über den Datenfluß innerhalb von Unterprogrammen, die nicht im Quellcode vorliegen, z.B. weil sie in Bibliotheken eingebunden sind.

5 Anwendungen des AMC

Das Large-Scale-Geostrophic (LSG) Modell der globalen ozeanischen Zirkulation wird zusammen mit seinem adjungierten Modell bereits intensiv zur Datenassimilation eingesetzt. Das vom AMC konstruierte adjungierte Modell zu dem Hamburg-Ocean-Primitive-Equation (HOPE) Modell soll neben der Datenassimilation für Sensitivitätsanalysen benutzt werden. Gekoppelt an ein statistisches Modell der Atmosphäre wird das Modell zur Simulation und Vorhersage von El-Niño verwendet. Die Bestimmung der führenden instabilen Moden des gekoppelten Modells könnten zum Aufspüren der Ursachen dieses Phänomens dienen. Das zu einem Modell zur Simulation der Ausbreitung von Meeresoberflächenwellen (Wave-Model, WAM) adjungierte Modell wird zur Optimierung der Parametrisierungen von Prozessen mithilfe von Beobachtungsdaten eingesetzt.

TM2 ist ein globales dreidimensionales numerisches Modell des atmosphärischen Transports. Durch Lösung der Kontinuitätsgleichung simuliert es die zeitliche Entwicklung der Konzentrationsverteilung eines beliebigen Spurenstoffes. Dazu müssen die Anfangskonzentrationen und der zeitliche Verlauf der Quellen und Senken vorgegeben werden. Die Frage nach demjenigen zeitlichen Verlauf der Quellfelder, der am besten durch das Modell auf eine Zeitreihe von Beobachtungen der Konzentrationen abgebildet wird, stellt ein inverses Problem dar. Wie oben beschrieben kann nach Konstruktion eines Programmteils zur Berechnung des Gradienten durch den AMC ein Gradientenverfahren zur Optimierung der Quellfelder eingesetzt werden. Auf diese Weise wurde anhand von mehrjährigen Beobachtungen der Kohlendioxidkonzentration an 27 Meßstationen der Jahresgang der Kohlendioxidflüsse von der Landbiosphäre in die Atmosphäre angepaßt. Ferner wurde das adjungierte Modell von TM2 für eine Sensitivitätsstudie eingesetzt. Abb. 2 zeigt den Einfluß der Quellstärke in jeder bodennahen Gitterzelle am 1. November auf den Mittelwert der Konzentration im Dezember an der Meßstation Mauna Loa auf Hawaii[3].

6 Diskussion und Ausblick

Die Entwicklung des AMC ist noch nicht abgeschlossen. Doch bereits jetzt erspart die automatische Erzeugung von adjungierten Modellen sehr viel Arbeit. Typischerweise wird mehr als ein Jahr benötigt, um ein adjungiertes Zirkulationsmodell zu erstellen. Diese Zeit verkürzt sich nun auf wenige Wochen.

Die Weiterentwicklung des AMC umfaßt Verbesserungen und Erweiterungen. Die Datenabhängigkeitsanalyse des AMC muß vielfach zurückhaltende Annahmen über Schleifenvermittelte Abhängigkeiten machen. Unter Berücksichtigung von Feldindexausdrücken sollen in Zukunft, zumindest wenn nur Ausdrücke auftreten, die linear von den Schleifenvariablen abhängen, sichere Entscheidungen getroffen werden. Die geplante automatische Lösung auftretender Konflikte würde die meisten noch notwendigen Änderungen der Quell-

[3]Mauna Loa, Hawaii, 3397m, 19°32′N, 155°35′W

programme, darunter das Einfügen von Direktiven zur Speicherung von Bedarfsvariablen, vermeiden.

Eine zukünftige Erweiterung kann die automatische Entscheidung zwischen Wiederberechnung und Speicherung von Bedarfsvariablen sein. Es wäre ebenso möglich, eine (sub)optimale Entscheidung zwischen Vorwärts- und Rückwärtsmodus für die einzelnen Progammteile zu treffen.

Programme zur Berechnung von höheren Ableitungen können durch Anwendung des AMC auf ein generiertes Programm zur Berechnung von Richtungsableitungen erzeugt werden. Ihre direkte Konstruktion hätte jedoch einige Vorteile.

Bisher akzeptiert der AMC lediglich eine Untermenge der Sprachelemente von Fortran-77. Neben der Implementierung des vollen Sprachumfangs ist eine Erweiterung um Elemente der Sprachen Fortran-90, High-Performance-Fortran (HPF) und CMFortran geplant.

Für die Zukunft wäre es wünschenswert, für die Konstruktion von Programmen zur Berechnung von Ableitungen eine 'black box' zur Verfügung zu haben. Sie würde ein Standardwerkzeug sein, wie das symbolische Differenzieren und die automatische Vektorisierung und Parallelisierung von Programmen.

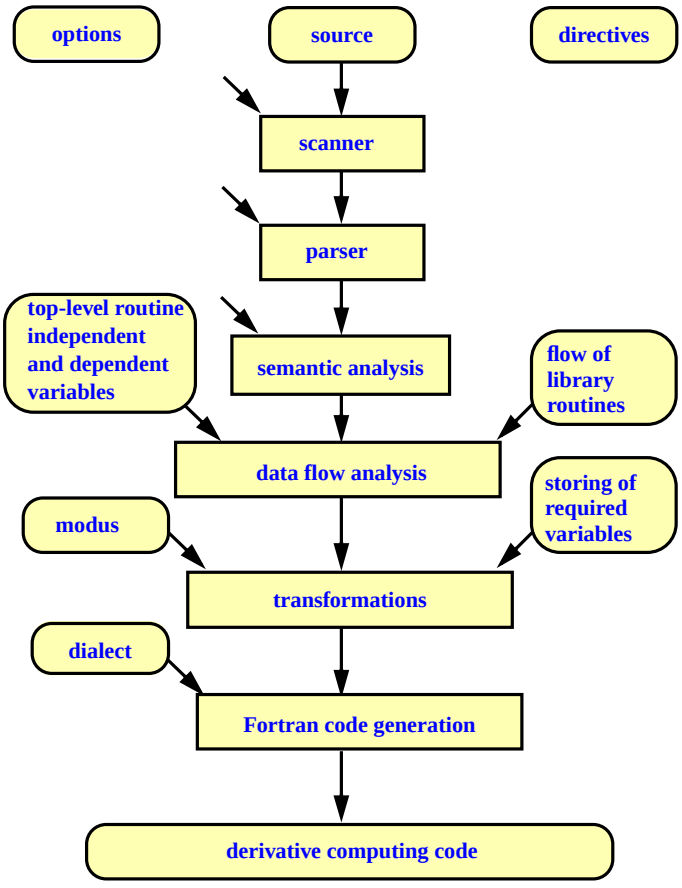

Abbildung 1: Schematischer Aufbau des AMC

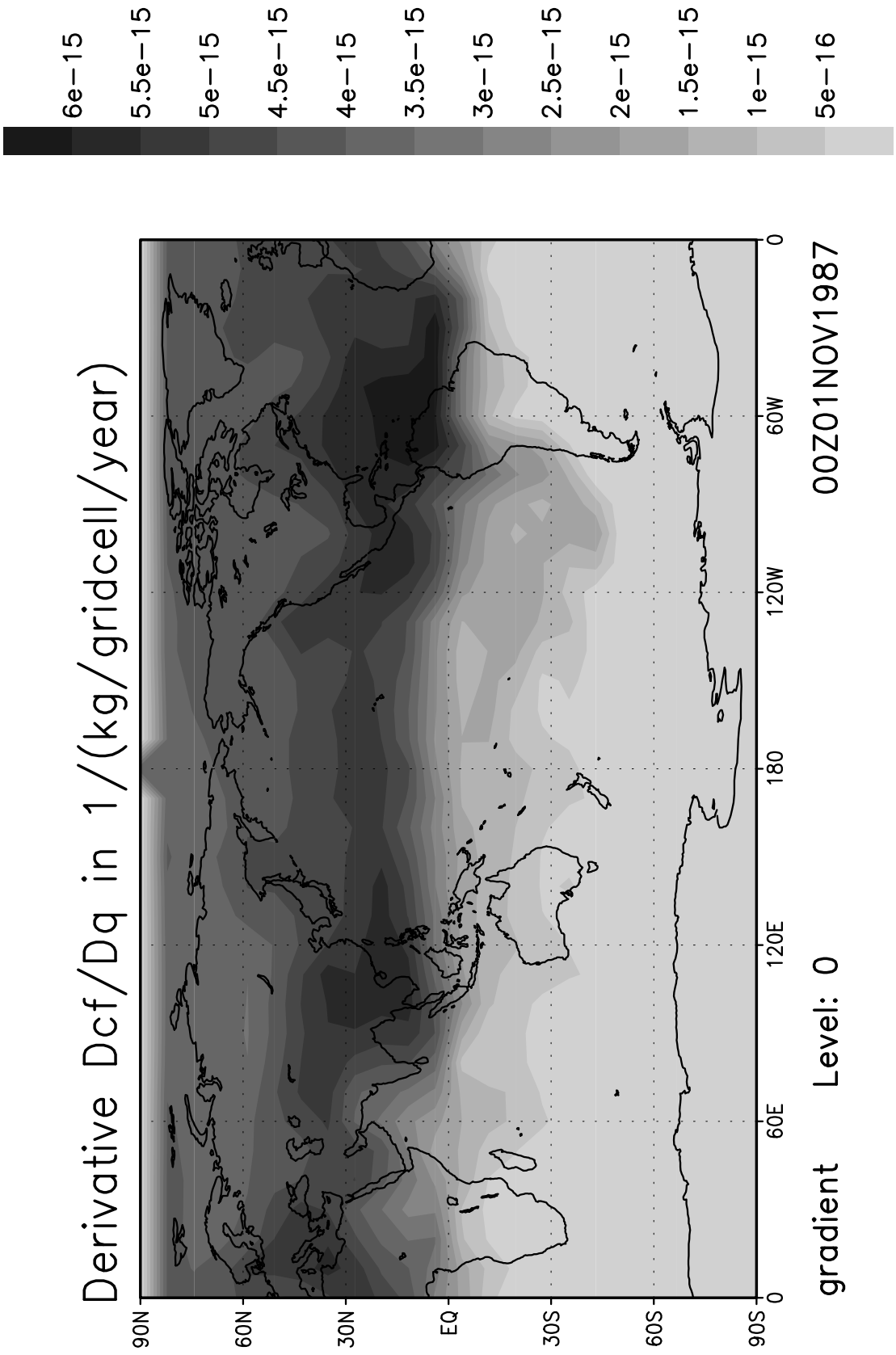

Abbildung 2: Einfluß der Quellstärke in jeder bodennahen Gitterzelle am 1. November auf den Mittelwert der Konzentration im Dezember an der Meßstation Mauna Loa

Literatur

Bischof, C., Carle, A., Khademi, P. und Maurer, A. (1994), The ADIFOR 2.0 System for the Automatic Differentiation of Fortran 77 Programs., Argonne Preprint ANL-MCS-P981-1194, Mathematics and Computer Science Division,Argonne National Laboratory, 9700 South Cass Ave., Argonne.

Bischof, C. und Hovland, P. (1991), Using ADIFOR to compute dense and sparse jacobians, Technical report ANL/MCS-TM-158, Mathematics and Computer Science Division,Argonne National Laboratory, 9700 South Cass Ave., Argonne.

Giering, R. (1992), *Adjoint Model Compiler, users manual*, Max-Planck-Institut für Meteorologie.

Giering, R. und Kaminski, T. (1996), Recipes for Adjoint Code Construction, Technical Report 212, Max-Planck-Institut für Meteorologie.

Griewank, A. und Reese, S. (1991), On the calculation of Jacobian matrices by the Markowitz rule, *in* A. Griewank und G. Corliess, eds, 'Automatic Differentiation of Algorithms: Theory, Implementation and Application', SIAM, Philadelphia, Penn, pp. 191–201.

Juedes, D. (1991), A taxonomy of automatic differentiation tools, *in* A. Griewank und G. Corliess, eds, 'Automatic Differentiation of Algorithms: Theory, Implementation and Application', SIAM, Philadelphia, Penn, pp. 315–329.

Rostaing, N., Dalmas, S. und Galligo, A. (1993), 'Automatic differentiation in Odyssée', *Tellus* pp. 558–568.

A Beispiel

Zur Erzeugung des adjungierten Unterprogramms wurde `x` als unabhängige Variable und `fc` als abhängige Variable angegeben. Der Deklarationsteil des adjungierten Unterprogramms wurde nachträglich editiert, damit der Code auf eine Seite paßt.

```
      subroutine helmholtz( n, x, rt, fc )

      implicit none

c--------------------------------------
c arguments
c--------------------------------------
      integer n
      real x(n), rt, fc

c--------------------------------------
c constants
c--------------------------------------
      integer nmax
      parameter( nmax = 40 )

c--------------------------------------
c global variables
c--------------------------------------
      real a, b
      common /matrix/ a(nmax,nmax), b(nmax)

c--------------------------------------
c local variables
c--------------------------------------
      real hh, sum1, sum2
      real btx, xax, ss, t1
      integer i, j

c--------------------------------------
c btx = b^T * x
c--------------------------------------
      btx = 0.0
      do i = 1, n
         btx = btx + b(i)*x(i)
      end do

c--------------------------------------
c xax = x^T A x
c--------------------------------------
      xax = 0.0
      do i = 1, n
         ss = 0.0
         do j = 1, n
            ss = ss + a(i,j)*x(j)
         end do
         xax = xax + x(i)*ss
      end do

c-----------------------------------------------
c SUM_{i=1}^{n} x_i ln( x_i/(1 - b^T x) )
c-----------------------------------------------
      t1 = 0.0
      do i = 1, n
         hh = alog( x(i)/(1.0-btx) )
         t1 = t1 + x(i) * hh
      end do

c--------------------------------------
c    x^T Ax        1+(1+sqrt(2))b^T x
c ------------ ln( ------------------ )
c sqrt(8)b^T x     1+(1-sqrt(2))b^T x
c--------------------------------------
      sum1 = (1.0 + sqrt(2.0))*btx
```

```
sum2 = (1.0 - sqrt(2.0))*btx
fc = rt*t1 - (xax/(sqrt(8.0)*btx))*alog((1.0+sum1)/(1.0+sum2))
end
```

```
      subroutine adhelmholtz( n, x, rt, adx, adfc )

      implicit none
      integer nmax
      parameter ( nmax = 40 )

      common /matrix/ a, b
      real a(nmax,nmax), b(nmax)

      real adbtx, adhh, adss, adsum1, adsum2, adt1, adxax, btx, hh
      integer n, i, ip1, j
      real adfc, adx(n), rt, x(n), ss, sum1, sum2, xax

      adbtx = 0
      adhh = 0
      adss = 0
      adsum1 = 0
      adsum2 = 0
      adt1 = 0
CDIR$ IVDEP
      do ip1 = 1, n
        adx(ip1) = 0
      end do
      adxax = 0
      btx = 0.
      do i = 1, n
        btx = btx+b(i)*x(i)
      end do
      xax = 0.
      do i = 1, n
        ss = 0.
        do j = 1, n
          ss = ss+a(i,j)*x(j)
        end do
        xax = xax+x(i)*ss
      end do
      sum1 = (1.+sqrt(2.))*btx
      sum2 = (1.-sqrt(2.))*btx
      adbtx = adbtx+adfc*xax*sqrt(8.)/(btx*btx*sqrt(8.)*sqrt(8.))*alog((
     $1.+sum1)/(1.+sum2))
      adsum1 = adsum1-adfc*xax/(btx*sqrt(8.))*1./((1.+sum1)/(1.+sum2))/(
     $1.+sum2)
      adsum2 = adsum2+adfc*xax/(btx*sqrt(8.))*1./((1.+sum1)/(1.+sum2))*(
     $1.+sum1)/((1.+sum2)*(1.+sum2))
      adt1 = adt1+adfc*rt
      adxax = adxax-adfc/(btx*sqrt(8.))*alog((1.+sum1)/(1.+sum2))
      adfc = 0
      adbtx = adbtx+adsum2*(1.-sqrt(2.))
      adsum2 = 0
      adbtx = adbtx+adsum1*(1.+sqrt(2.))
      adsum1 = 0
      do i = n, 1, -1
        hh = alog(x(i)/(1.-btx))
        adhh = adhh+adt1*x(i)
        adx(i) = adx(i)+adt1*hh
        adbtx = adbtx+adhh*1./(x(i)/(1.-btx))*x(i)/((1.-btx)*(1.-btx))
        adx(i) = adx(i)+adhh*1./(x(i)/(1.-btx))/(1.-btx)
        adhh = 0
      end do
      do i = n, 1, -1
        ss = 0.
        do j = 1, n
          ss = ss+a(i,j)*x(j)
        end do
        adss = adss+adxax*x(i)
        adx(i) = adx(i)+adxax*ss
        do j = n, 1, -1
          adx(j) = adx(j)+adss*a(i,j)
        end do
        adss = 0
      end do
      do i = n, 1, -1
        adx(i) = adx(i)+adbtx*b(i)
```

```
end do

end
```