Das Konzept

Vektorfunktion in **C/C++**: $F : \mathbb{R}^n \to \mathbb{R}^m : x \mapsto y = F(x)$

$\Downarrow$ Operatorüberladen (C++)

Interne Darstellung von F ($\equiv$*tape*)

$\Downarrow$ Interpretation $\Downarrow$

Vorwärtsmodus

$$x(t) = \sum_{j=0}^{d} x_j t^j$$

$$\Downarrow$$

$$y(t) = \sum_{j=0}^{d} y_j t^j + O\left(t^{d+1}\right)$$

$\Rightarrow$ **Richtungsableitungen**

Rückwärtsmodus

$$y_j = y_j\left(x_0, x_1, \ldots, x_j\right)$$

$$\Downarrow$$

$$\frac{\partial y_j}{\partial x_i} = \frac{\partial y_{j-i}}{\partial x_0} = A_{j-i}\left(x_0, x_1, \ldots, x_{j-i}\right)$$

$\Rightarrow$ **Gradienten (Adjungierte)**

$$\begin{aligned}
y_0 &= F(x_0) \\
y_1 &= F'(x_0)\, x_1 \\
y_2 &= F'(x_0)\, x_2 + \frac{1}{2} F''(x_0)\, x_1 x_1 \\
y_3 &= F'(x_0)\, x_3 + F''(x_0)\, x_1 x_2 + \frac{1}{6} F'''(x_0)\, x_1 x_1 x_1 \\
&\ldots
\end{aligned}$$

$$\begin{aligned}
\frac{\partial y_0}{\partial x_0} = \frac{\partial y_1}{\partial x_1} = \frac{\partial y_2}{\partial x_2} = \frac{\partial y_3}{\partial x_3} &= A_0 = F'(x_0) \\
\frac{\partial y_1}{\partial x_0} = \frac{\partial y_2}{\partial x_1} = \frac{\partial y_3}{\partial x_2} &= A_1 = F''(x_0)\, x_1 \\
\frac{\partial y_2}{\partial x_0} = \frac{\partial y_3}{\partial x_1} &= A_2 = F''(x_0)\, x_2 + \frac{1}{2} F'''(x_0)\, x_1 x_1 \\
\frac{\partial y_3}{\partial x_0} &= A_3 = F''(x_0)\, x_3 + F'''(x_0)\, x_1 x_2 + \frac{1}{6} F^{(4)}(x_0)\, x_1 x_1 x_1 \\
&\ldots
\end{aligned}$$

Anwendung

Konzept Operatorüberladen ⇒ Quelltexterweiterungen

- Einbinden der ADOL-C-Header-Datei `adolc.h`
- „Aktivieren" aller involvierten Variablen (`adouble`); ggf. Einführen zusätzlicher aktiver Variablen und Prozeduren
- Markieren des aktiven Bereiches (`trace_on`/`trace_off`)
- Festlegen der unabhängigen und abhängigen Variablen (`<<=`/`>>=`)
- Beschreiben der eigentlichen Differentiationsaufgabe(n)
- Übersetzen und Linken mit der ADOL-C Bibliothek `libad.a`

Beispiel:

```
#include "adolc.h"                      // Einbinden des ADOL-C Headers
...
adouble foo ( adouble x )               // aktive Funktion
{ adouble tmp;
  tmp = log(x);
  return 3.0*tmp*tmp + 2.0;
}
...
int main (int argc, char* argv[])       // Hauptprogramm (oder andere Prozedur)
{ ...
  double   x[2],  y;
  adouble ax[2], ay;                    // Deklaration der aktiven Variablen
  x[0]=0.3; x[1]=2.3;
  trace_on(1);                          // Beginn des aktiven Bereiches
    ax[0]<<=x[0]; ax[1]<<=x[1];         // Markieren unabhängiger Variablen
    ay=ax[0]*sin(ax[1])+ foo(ax[1]);    // aktive Funktionsauswertung
    ay>>=y;                             // Markieren abhängiger Variablen
  trace_off();                          // Ende des aktiven Bereiches
  ...
  double g[2];
  gradient(1,2,x,g);                    // Anwendung eines ADOL-C-Treibers
  ...                                   // zur Gradientenberechnung
  x[0]+=0.1; x[1]+=0.3;
  gradient(1,2,x,g);                    // Anwendung auf andere Argumente
  ...
}
```

Grundlegende Differentiationsroutinen (C/C++)

Vorwärtsmodus

`zos_forward(tag,m,n,keep,x[n],y[m])`

- skalarer Vorwärtsmodus 0. Ordnung: berechnet $y = F(x)$
- $0 \leq$ `keep` ≤ 1; `keep` $= 1:$ Vorbereitung für `fos_reverse` oder `fov_reverse`

`fos_forward(tag,m,n,keep,x0[n],x1[n],y0[m],y1[m])`

- skalarer Vorwärtsmodus 1. Ordnung: berechnet $y_0 = F(x_0)$, $y_1 = F'(x_0)\, x_1$
- $0 \leq$ `keep` ≤ 2; `keep` $= \begin{cases} 1: & \text{Vorbereitung für } \texttt{fos\_reverse} \text{ oder } \texttt{fov\_reverse} \\ 2: & \text{Vorbereitung für } \texttt{hos\_reverse} \text{ oder } \texttt{hov\_reverse} \end{cases}$

`fov_forward(tag,m,n,p,x[n],X[n][p],y[m],Y[m][p])`

- vektorieller Vorwärtsmodus 1. Ordnung: berechnet $y = F(x)$, $Y = F'(x)\, X$

`hos_forward(tag,m,n,d,keep,x[n],X[n][d],y[m],Y[m][d])`

- skalarer Vorwärtsmodus d-ter Ordnung: berechnet $y_0 = F(x_0)$, $y_1 = F'(x_0)\, x_1$ usw., wobei $x = x_0$, $X = [x_1, x_2, \ldots, x_d]$ und $y = y_0$, $Y = [y_1, y_2, \ldots, y_d]$
- $0 \leq$ `keep` $\leq d{+}1$; `keep` $\begin{cases} = 1: & \text{Vorbereitung für } \texttt{fos\_reverse} \text{ oder } \texttt{fov\_reverse} \\ > 1: & \text{Vorbereitung für } \texttt{hos\_reverse} \text{ oder } \texttt{hov\_reverse} \end{cases}$

`hov_forward(tag,m,n,d,p,x[n],X[n][p][d],y[m],Y[m][p][d])`

- vektorieller Vorwärtsmodus d-ter Ordnung: berechnet $y_0 = F(x_0)$, $Y_1 = F'(x_0)\, X_1$ usw., wobei $x = x_0$, $X = [X_1, X_2, \ldots, X_d]$ und $y = y_0$, $Y = [Y_1, Y_2, \ldots, Y_d]$

Rückwärtsmodus

`fos_reverse(tag,m,n,u[m],z[n])`

- skalarer Rückwärtsmodus 1. Ordnung: berechnet $z^T = u^T F'(x_0)$
- nach Aufruf von `zos_forward`, `fos_forward` oder `hos_forward` mit `keep` $= 1$

`fov_reverse(tag,m,n,q,U[q][m],Z[q][n])`

- vektorieller Rückwärtsmodus 1. Ordnung: berechnet $Z = UF'(x_0)$
- nach Aufruf von `zos_forward`, `fos_forward` oder `hos_forward` mit `keep` $= 1$

`hos_reverse(tag,m,n,d,u[m],Z[n][d+1])`

- skalarer Rückwärtsmodus d-ter Ordnung: berechnet die Adjungierten $z_0^T = u^T F'(x_0) = u^T A_0$, $z_1^T = u^T F''(x_0)\, x_1 = u^T A_1$ usw., wobei $Z = [z_0, z_1, \ldots, z_d]$
- nach Aufruf von `fos_forward` oder `hos_forward` mit `keep` $= d + 1 > 1$

`hov_reverse(tag,m,n,d,q,U[q][m],Z[q][n][d+1],nz[q][n])`

- vektorieller Rückwärtsmodus d-ter Ordnung: berechnet die Adjungierten $Z_0 = UF'(x_0) = UA_0$, $Z_1 = UF''(x_0)\, x_1 = UA_1$ usw., wobei $Z = [Z_0, Z_1, \ldots, Z_d]$
- nach Aufruf von `fos_forward` oder `hos_forward` mit `keep` $= d + 1 > 1$
- optionale Dünnbesetztheitsstruktur `nz` ($\Rightarrow$ Handbuch)

Beispiel:

```
...
double x[n], y[m], **I, **J;
I=myallocI2(m);                    // Allokation der Identitätsmatrix
J=myalloc2(m,n);                   // Allokation der Jacobi-Matrix
...
initialize(x);                     // Setzen des Startwertes von x
...
zos_forward(ftag,m,n,1,x,y);        // Berechnen der Jacobimatrix im
fos_reverse(ftag,m,n,m,I,J);       // Rückwärtsmodus des AD
...
```

Spezielle C++-Schnittstellen

Vorwärtsmodus

`forward(tag,m,n,d,keep,X[n][d+1],Y[m][d+1])` `forward(tag,m=1,n,d,keep,X[n][d+1],Y[d+1])`	hos, fos, zos hos, fos, zos
`forward(tag,m,n,d=0,keep,x[n],y[m])` `forward(tag,m,n,keep,x[n],y[m])`	zos zos
`forward(tag,m,n,p,x[n],X[n][p],y[m],Y[m][p])`	fov
`forward(tag,m,n,d,p,x[n],X[n][p][d],` `y[m],Y[m][p][d])`	hov

Rückwärtsmodus

`reverse(tag,m,n,d,u[m],Z[n][d+1])` `reverse(tag,m=1,n,d,u,Z[n][d+1])`	hos hos
`reverse(tag,m,n,d=0,u[m],z[n])` `reverse(tag,m=1,n,d=0,u,z[n])`	fos fos

`reverse(tag,m,n,d,q,U[q][m],Z[q][n][d+1],nz[q][n])`
`reverse(tag,m=1,n,d,q,U[q],Z[q][n][d+1],nz[q][n])`
`reverse(tag,m=1,n,d,Z[m][n][d+1],nz[m][n])` $(U = I_m)$
	hov
	hov
	hov

`reverse(tag,m,n,d=0,q,U[q][m],Z[q][n])`
`reverse(tag,m,n,q,U[q][m],Z[q][n]`
`reverse(tag,m=1,n,d=0,q,U[q],Z[q][n])`
	fov
	fov
	fov

Treiber für die Optimierung und nichtlineare Gleichungen (C/C++)

$$\min_x f(x), \qquad f : \mathbb{R}^n \to \mathbb{R}$$
$$F(x) = 0_m, \qquad F : \mathbb{R}^n \to \mathbb{R}^m$$

`function(tag,m,n,x[n],y[m])`	$F(x)$
`gradient(tag,n,x[n],g[n])` `hessian(tag,n,x[n],H[n][n])`	$\nabla f(x)$ $\nabla^2 f(x)$

`jacobian(tag,m,n,x[n],J[m][n])`
`vec_jac(tag,m,n,repeat?,x[n],u[m],z[n])`
`jac_vec(tag,m,n,x[n],v[n],z[m])`
	$F'(x)$
	$u^T F'(x)$
	$F'(x)\, v$

`hess_vec(tag,n,x[n],v[n],z[n])` `lagra_hess_vec(tag,m,n,x[n],v[n],u[m],h[n])`	$\nabla^2 f(x)\, v$ $u^T F''(x)\, v$
`jac_solv(tag,n,x[n],b[n],sparse?,mode?)`	$F'(x)\, w = b$

Beispiel: Lösung von $F(x) = 0$ mit Newton-Methode

```
...
double x[n], r[n];
int i;
...
initialize(x);                     // Setzen des Startwertes x
...
function(ftag,n,n,x,r);            // Berechnen des Residuums r
while (norm(r) > EPSILON)          // Abbruch bei kleinem Residuum
{ jac_solv(ftag,n,x,r,0,2);        // Berechnung von r := F'(x)^-1 * r
  for (i=0; i<n; i++)              // Aufdatieren von x
    x[i] -= r[i];
  function(ftag,n,n,x,r);          // Berechnen des Residuums r
}
...
```

Treiber für gewöhnliche Differentialgleichungen (C/C++)

$$x'(t) = y(t) = F(x(t)), \qquad x(0) = x_0$$

```
forodec(tag,n,tau,dold,d,X[n][d+1])
```

- rekursive Berechnung der $x_{d_{old}+1}, \dots, x_d$ aus $x_0, \dots, x_{d_{old}}$ (mittels $x_{i+1} = \frac{1}{1+i} y_i$) im Vorwärtsmodus
- Anwendung mit $d_{old} = 0$ liefert abgebrochene Taylorreihe $\sum_0^d x_j t^j$ am Punkt x_0

```
hov_reverse(tag,n,n,d-1,n,I[n][n],A[n][n][d],nz[n][n])
```

- Berechnung der partiellen Ableitungen $A_j = \frac{\partial y_j}{\partial x_0}$ $(j = 0, \dots, d)$ im Rückwärtsmodus
- nach Aufruf von `forodec` mit Grad d; optionale Dünnbesetztheitsstruktur `nz` ($\Rightarrow$ Handbuch)

```
accodec(n,tau,d-1,A[n][n][d],B[n][n][d],nz[n][n])
```

- Akkumulation der totalen Ableitungen $B_j = \frac{dx_j}{dx_0}$ $(j = 0, \dots, d)$ aus den partiellen Ableitungen $A_j = \frac{\partial y_j}{\partial x_0}$ $(j = 0, \dots, d)$
- nach Aufruf von `hov_reverse`, optionale Dünnbesetztheitsstruktur `nz` ($\Rightarrow$ Handbuch)

C++: spezielle C++-Schnittstellen befinden sich in der Datei `SRC/DRIVERS/odedrivers.h`

Beispiel:

```
...
double x[n], **I, **X, ***A, ***B;
I=myallocI2(n);                          // Allokation der Identitätsmatrix
X=myalloc2(n,5);                         // Allokation der Matrix X
A=myalloc3(n,n,4); B=myalloc3(n,n,4);    // Allokation der Tensoren A und B
...
initialize(X);                           // Setzen des Startwertes x0
...
forodec(ftag,n,1.0,0,4,X);               // Berechne x1,...,x4
hov_reverse(ftag,n,n,3,n,I,A,NULL);      // Berechne A0,...,A3
accodec(ftag,n,1.0,3,A,B,NULL);          // Akkumuliere B0,...,B3
...
```

ADOL-C bietet

- grundlegende Differentiationsroutinen (`forward`/`reverse`)
- „Easy-to-use"-Treiber für die
  - Lösung von Optimierungsproblemen und nichtlinearen Gleichungssystemen
  - Integration gewöhnlicher Differentialgleichungen
  - Berechnung von Tensoren höherer Ordnung ($\Rightarrow$ Handbuch)
- Ableitungen impliziter und inverser Funktionen ($\Rightarrow$ Handbuch)
- Abhängigkeitsanalyse, vorwärts und rückwärts ($\Rightarrow$ Handbuch)

Geplante Erweiterungen

- Ausnutzung der Dünnbesetztheit bei der Kompression von Jacobimatrizen (Newsam-Ramsdell, Curtis-Powell-Reid)
- „Easy-to-use"-Treiber für differential-algebraische Gleichungen
- Integration von Checkpointing-Routinen
- effiziente Verfahren zur Untersuchung/Verifikation der Dünnbesetztheitsstruktur von Jacobimatrizen
- interne Optimierung zur Reduktion des Speicherbedarfs beim Rückwärtsmodus

Kontaktadresse/Quellen

- E-mail: `adol-c@math.tu-dresden.de`
- WWW: `http://www.math.tu-dresden.de/~adol-c`
- ftp: `ftp.math.tu-dresden.de/pub/ADOLC`