AD in FORTRAN

Implementation *via* Prepreprocessor

Alexey Radul[†*] Barak A. Pearlmutter[†‡] Jeffrey Mark Siskind[§]

[†]Hamilton Institute
[‡]Department of Computer Science
NUI Maynooth, Co. Kildare, Ireland

[§]School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN, USA

[*]Boston Fusion Corporation, Burlington, MA, USA
(current affiliation)

Outline of Talk

- motivation
- description by example
- implementation sketch
- take-home lessons

motivation

desiderata

AD which is:

- first class
  - integrated into language
  - can be used anywhere
  - can apply to anything
- convenient & natural
- modular
- expressive
- fast

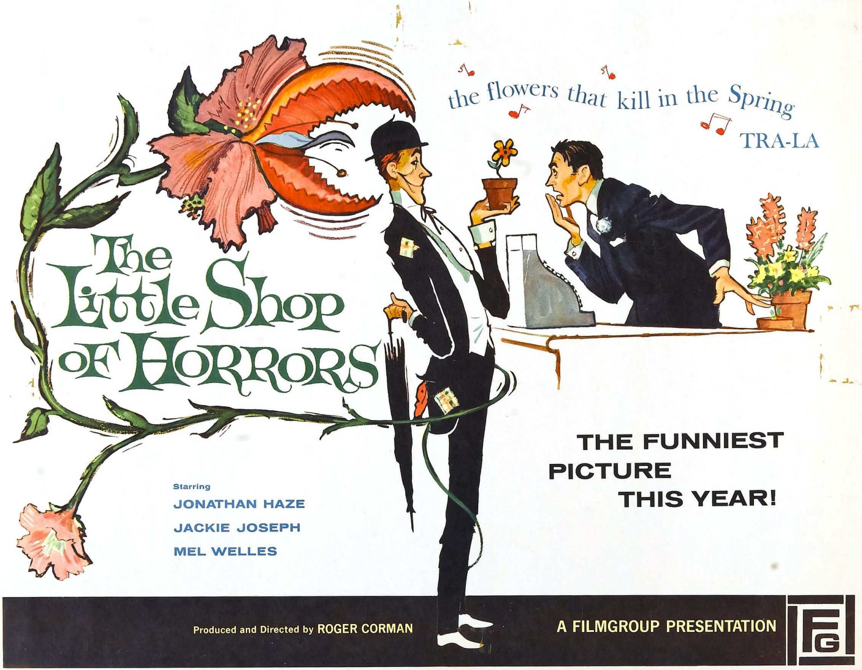

TOTALLY
COOL
PIX.COM

what we did:

add AD to FORTRAN

(implementation: leverage existing AD tools)

THE THRILLS - THE CHILLS OF WITCHCRAFT TODAY
THE City OF THE dead
CHRISTOPHER LEE · DENNIS LOTIS
BETTA ST JOHN · PATRICIA JESSEL
WITH
VALENTINE DYALL
INTRODUCING
VENETIA STEVENSON
X ADULTS ONLY
Produced by DONALD TAYLOR Directed by JOHN MOXEY
Executive Producers: SEYMOUR DORNER & MILTON SUBOTSKY
Screenplay by GEORGE BAXT Story by MILTON SUBOTSKY
A VULCAN FILM PRODUCTION
Distributed by BRITISH LION in association with BRITANNIA FILMS
PRINTED IN ENGLAND


the flowers that kill in the Spring
TRA-LA
The Little Shop of Horrors
Starring
JONATHAN HAZE
JACKIE JOSEPH
MEL WELLES
THE FUNNIEST PICTURE THIS YEAR!
Produced and Directed by ROGER CORMAN
A FILMGROUP PRESENTATION
FG

60/243

Traditional API for AD

```
      FUNCTION RAPHSN(F, FPRIME, X0, N)
      EXTERNAL F, FPRIME
      X = X0
      DO 1690 I=1,N
 1690 X = X-F(X)/FPRIME(X)
      RAPHSN = X
      END
```

Note: *caller* provides both `F` and `FPRIME`.

Manually coding `FPRIME` from `F` is often mechanical, tedious, and error prone.

Automatic differentiation (Speelpenning, 1980; Wengert, 1964) eliminates that, but the *caller* of `RAPHSN` still needs to provide `FPRIME`, perhaps also arranging for it to be generated automatically from `F`.

allow *callee derives*

FARFEL: add AD block constructs to FORTRAN

Compute derivative `PHIPRM` of `PHI` wrt `SIGMA` by forward AD:

```
      ADF(TANGENT(SIGMA)=1.0)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*
      END ADF(PHIPRM=TANGENT(PHI))
```

FARFEL: add AD block constructs to FORTRAN

Compute derivative `PHIPRM` of `PHI` wrt `SIGMA` by forward AD:

```
      ADF(TANGENT(SIGMA)=1.0)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*
      END ADF(PHIPRM=TANGENT(PHI))
```

Same, by reverse AD:

```
      ADR(COTANGENT(PHI)=1.0)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*
      END ADR(PHIPRM=COTANGENT(SIGMA))
```

FARFEL: add AD block constructs to FORTRAN

Compute derivative `PHIPRM` of `PHI` wrt `SIGMA` by forward AD:

```
ADF(TANGENT(SIGMA)=1.0)
PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*
END ADF(PHIPRM=TANGENT(PHI))
```

Same, by reverse AD:

```
ADR(COTANGENT(PHI)=1.0)
PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*
END ADR(PHIPRM=COTANGENT(SIGMA))
```

Syntactically similar to the Naumann and Riehme (2005) NAGWARE 95 extensions, but more (a) general, (b) expressive, (c) available, (d) performant, (e) examples to follow.

```
      SUBROUTINE GRAD(F, X, N, DX)
      EXTERNAL F
      DIMENSION X(N), DX(N)
      DO 1492 I=1,N
      ADF((TANGENT(X(J))=0.0, J=1,N), TANGENT(X(I))=1.0)
      Y = F(X, N)
 1492 END ADF(DX(I)=TANGENT(Y))
      END
```

```
      SUBROUTINE GRAD(F, X, N, DX)
      EXTERNAL F
      DIMENSION X(N), DX(N)
      ADR(COTANGENT(Y)=1.0)
      Y = F(X, N)
      END ADR((DX(I)=COTANGENT(X(I)), I=1,N))
      END
```

- callee derives
- same API for both versions
- implied **DO** syntax for arrays
- no restrictions, **EXTERNAL** parameters allowed

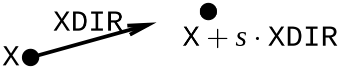
```
XSTAR = ARGMAX(F, X0)
```

```
C       CHECKPOINT REVERSE F->G.
C       BOTH F AND G ARE 1ST ARG IN, 2ND ARG OUT
        CALL F(X, Y)
        ADR((COTANGENT(Z(I))=DZ(I), I=1,NZ))
        CALL G(Y, Z)
        END ADR((DY(I)=COTANGENT(Y(I)), I=1,NY))
        ADR((COTANGENT(Y(I))=DY(I), I=1,NY))
        CALL F(X, Y)
        END ADR((DX(I)=COTANGENT(X(I)), I=1,NX))
```

FARFEL allows nested subprograms

Allowing nested definitions is synergistic with first-class AD.

```
C     MAXIMIZE F ALONG THE LINE PARALLEL TO XDIR THROUGH X
      SUBROUTINE LINMAX(F, X, XDIR, LENX, N, XOUT)
      EXTERNAL F
      DIMENSION X(LENX), XDIR(LENX), XOUT(LENX)
        FUNCTION ALINE(DIST)
        DIMENSION Y(50)
        DO 2012 I=1,LENX
 2012   Y(I) = X(I)+DIST*XDIR(I)
        ALINE = F(Y, LENX)
        END
      BESTD = ARGMAX(ALINE, 0.0, N)
      DO 2013 I=1,LENX
 2013 XOUT(I) = X(I)+BESTD*XDIR(I)
      END
```



optimise scalar $a(s) = f(\vec{x} + s \cdot \vec{dir})$

encapsulation

modularity

nesting for modularity?

nesting for expressivity!

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player B's strategy is b.

Player A's return is $A(a, b)$.
Player B's return is $B(a, b)$.

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a,b)$.

Player B's strategy is b.
Player B's return is $B(a,b)$.

Equilibria must satisfy:

$$a^* = \operatorname*{argmax}_a A(a, b^*) \qquad b^* = \operatorname*{argmax}_b B(a^*, b)$$

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.

Player A's return is $A(a,b)$.

Player B's strategy is b.

Player B's return is $B(a,b)$.

Equilibria must satisfy:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a, b^*) \qquad b^* = \underset{b}{\operatorname{argmax}}\, B(a^*, b)$$

Find by solving:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a, \underset{b}{\operatorname{argmax}}\, B(a^*, b))$$

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.

Player A's return is $A(a, b)$.

Player B's strategy is b.

Player B's return is $B(a, b)$.

Equilibria must satisfy:

$$a^* = \operatorname*{argmax}_a A(a, b^*) \qquad b^* = \operatorname*{argmax}_b B(a^*, b)$$

Find by solving:

$$a^* = \operatorname*{argmax}_a A(a, \operatorname*{argmax}_b B(a^*, b))$$

nesting

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a, b)$.

Player B's strategy is b.
Player B's return is $B(a, b)$.

Equilibria must satisfy:

$$a^* = \operatorname*{argmax}_a A(a, b^*) \qquad b^* = \operatorname*{argmax}_b B(a^*, b)$$

Find by solving:

$$a^* = \operatorname*{argmax}_a A(a, \operatorname*{argmax}_b B(a^*, b))$$

nesting

aka

$$\operatorname*{root}_{a^*} \operatorname*{argmax}_a A(a, \operatorname*{argmax}_b B(a^*, b)) - a^*$$

more nesting

$$\operatorname*{root}_{a^*}\ \operatorname*{argmax}_{a} A(a, \operatorname*{argmax}_{b} B(a^*, b)) - a^*$$

Five levels of nesting. (One from `ROOT`, two from each `ARGMAX`.)

```
C     ASTAR & BSTAR: GUESSES IN, OPTIMIZED VALUES OUT
      SUBROUTINE EQLBRM(BIGA, BIGB, ASTAR, BSTAR, N)
      EXTERNAL BIGA, BIGB
        FUNCTION F(ASTAR)
          FUNCTION G(A)
            FUNCTION H(B)
            H = BIGB(ASTAR, B)
            END
          BSTAR = ARGMAX(H, BSTAR, N)
          G = BIGA(A, BSTAR)
          END
        F = ARGMAX(G, ASTAR, N)-ASTAR
        END
      ASTAR = ROOT(F, ASTAR, N)
      END
```

```
      FUNCTION ARGMAX(F, X0, N)
        FUNCTION FPRIME(X)
        FPRIME = DERIV1(F, X)
        END
      ARGMAX = ROOT(FPRIME, X0, N)
      END
```

```
      FUNCTION ROOT(F, X0, N)
      X = X0
      DO 1669 I=1,N
      CALL DERIV2(F, X, Y, YPRIME)
 1669 X = X-Y/YPRIME
      ROOT = X
      END
```

```
      FUNCTION DERIV1(F, X)
      EXTERNAL F
      ADF(X)
      Y = F(X)
      END ADF(DERIV1 = TANGENT(Y))
      END
```

```
      SUBROUTINE DERIV2(F, X, Y, YPRIME)
      EXTERNAL F
      ADF(X)
      Y = F(X)
      END ADF(YPRIME = TANGENT(Y))
      END
```

implementation


Nuclear Bunker
CLOTHES MUST BE REMOVED WHEN FINISHED
CLOTHES MAY BE REMOVED BY WAITING CUSTOMERS OR MANAGEMENT
Curry Prevention Services

Possible Implementation Strategies:

- native (integrated into compiler)
- preprocessor (generate FORTRAN)
- prepreprocessor (generate input to AD tools)

Possible Implementation Strategies:

- native (integrated into compiler)
- preprocessor (generate FORTRAN)
- prepreprocessor (generate input to AD tools)

FARFALLEN implementation of FARFEL:

- move contents of AD blocks into (nested) subroutines
- closure-convert nested subprograms to top level
- specialize away **EXTERNAL** constructs
- target TAPENADE or ADIFOR


灌木林步行徑
若非與經驗步行者同行，
請勿步過警告牌。
등산길 경고문
등산에 관한 전문지식이나 경험이 적으신
분들은 이 표지판 뒤로 넘어가지 말아 주시길
바랍니다.
ブッシュ・ウォーキング(山歩き)
コース
ブッシュ・ウォーキング(山歩き)の熟練者
以外は、これより先には行かないで
下さい。
THIS SIGN IS TO PREVENT FOREIGN
TOURISTS FROM GETTING LOST

AD block contents to nested subroutines

```
      PROGRAM MAIN
      SIGMA = 1.0
      PI = 3.14159
      X = 1.0
      XBAR = 1.0
      ADF(TANGENT(SIGMA) = 1.0)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*((X-XBAR)/SIGMA)**2)
      END ADF(PHIPRM = TANGENT(PHI))
      PRINT *, PHIPRM
      END
```

$\Downarrow$

```
      PROGRAM MAIN
      SIGMA = 1.0
      PI = 3.14159
      X = 1.0
      XBAR = 1.0
        SUBROUTINE ADF1(PHI, SIGMA)
        PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*((X-XBAR)/SIGMA)**2)
        END
      SIGMA_G1 = 1.0
      ADF CALL ADF1(PHI, PHI_G1, SIGMA, SIGMA_G1)
      PHIPRM = PHI_G1
      PRINT *, PHIPRM
      END
```

Closure Conversion

```
      PROGRAM MAIN
      SIGMA = 1.0
      PI = 3.14159
      X = 1.0
      XBAR = 1.0
        SUBROUTINE ADF1(PHI, SIGMA)
        PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*((X-XBAR)/SIGMA)**2)
        END
      SIGMA_G1 = 1.0
      ADF CALL ADF1(PHI, PHI_G1, SIGMA, SIGMA_G1)
      PHIPRM = PHI_G1
      PRINT *, PHIPRM
      END
```

$\Downarrow$

```
      SUBROUTINE MAIN_ADF1(PHI, SIGMA, PI, X, XBAR)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*((X-XBAR)/SIGMA)**2)
      END

      PROGRAM MAIN
      SIGMA = 1.0
      PI = 3.14159
      X = 1.0
      XBAR = 1.0
      SIGMA_G1 = 1.0
      ADF CALL MAIN_ADF1(PHI, PHI_G1, SIGMA, SIGMA_G1, PI, X,
     + XBAR)
      PHIPRM = PHI_G1
      PRINT *, PHIPRM
      END
```

Specialization

```
      SUBROUTINE QUAD(F, A, B, STEP, OUT)
      EXTERNAL F
      OUT = 0.0
      DO 5 X = A,B,STEP
 5    OUT = OUT+F(X)
      END

      ...
      CALL QUAD(EXP, 0.0, 5.0, 0.1, Z)
      CALL QUAD(SQRT, 1.0, 6.0, 0.1, X)
```

```
      SUBROUTINE QUAD_SQRT(A, B, STEP, OUT)
      OUT = 0.0
      DO 5 X = A,B,STEP
 5    OUT = OUT+SQRT(X)
      END

      SUBROUTINE QUAD_EXP(A, B, STEP, OUT)
      OUT = 0.0
      DO 5 X = A,B,STEP
 5    OUT = OUT+EXP(X)
      END

      ...
      CALL QUAD_EXP(0.0, 5.0, 0.1, Z)
      CALL QUAD_SQRT(1.0, 6.0, 0.1, X)
```

Calling Conventions

```
      SUBROUTINE MAIN_ADF1(PHI, SIGMA, PI, X, XBAR)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*((X-XBAR)/SIGMA)**2)
      END

      PROGRAM MAIN
      SIGMA = 1.0
      PI = 3.14159
      X = 1.0
      XBAR = 1.0
      SIGMA_G1 = 1.0
      ADF CALL MAIN_ADF1(PHI, PHI_G1, SIGMA, SIGMA_G1, PI, X,
     + XBAR)
      PHIPRM = PHI_G1
      PRINT *, PHIPRM
      END
```

$$\Downarrow$$

```
      SUBROUTINE MAIN_ADF1(PHI, SIGMA, PI, X, XBAR)
      PHI = 1/SQRT(2*PI*SIGMA**2)*EXP(-0.5*((X-XBAR)/SIGMA)**2)
      END

      PROGRAM MAIN
      SIGMA = 1.0
      PI = 3.14159
      X = 1.0
      XBAR = 1.0
      SIGMA_G1 = 1.0
      CALL MAIN_ADF1_G1(PHI, PHI_G1, SIGMA, SIGMA_G1, PI, X,
     + XBAR)
      PHIPRM = PHI_G1
      PRINT *, PHIPRM
      END
```

The full EQLBRM example

```
      FUNCTION GMBIGB(A, B)
      PRICE = 20-0.1*B-0.0999*A
      COSTS = B*(10.005-0.05*B)
      GMBIGB = B*PRICE-COSTS
      END

      FUNCTION EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, B)
      EQLBRM_GMBIGA_GMBIGB_F_G_H = GMBIGB(ASTAR, B)
      END

      SUBROUTINE DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H_ADF(ASTAR, X, Y)
      Y = EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, X)
      END

      FUNCTION DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, X)
      X_G1 = 1.0
      ASTAR_G1 = 0.0
      Y_G1 = 0.0
      CALL DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H_ADF_G1(ASTAR, ASTAR_G1, X,
     +X_G1, Y, Y_G1)
      DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H = Y_G1
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME = DERIV1_EQLBRM_GMBIGA_GM
     +BIGB_F_G_H(ASTAR, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME_ADF(AST
     +AR, X, Y)
      Y = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR,
     +X, Y, YPRIME)
      X_G2 = 1.0
      ASTAR_G2 = 0.0
      Y_G2 = 0.0
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME_ADF_G2(ASTAR,
     + ASTAR_G2, X, X_G2, Y, Y_G2)
      YPRIME = Y_G2
      END

      FUNCTION ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X0,
     +N)
      X = X0
      DO 1669 I = 1, N
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X, Y,
     +YPRIME)
 1669 X = X-Y/YPRIME
      ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME = X
      END

      FUNCTION GMBIGA(A, B)
      PRICE = 20-0.1*A-0.1*B
      COSTS = A*(10-0.05*A)
      GMBIGA = A*PRICE-COSTS
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, X0, N)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H = ROOT_ARGMAX_EQLBRM_GMBIGA_GMBI
     +GB_F_G_H_FPRIME(ASTAR, X0, N)
      END

      FUNCTION EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N, A)
      BSTAR = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, BSTAR, N)
      EQLBRM_GMBIGA_GMBIGB_F_G = GMBIGA(A, BSTAR)
      END

      SUBROUTINE DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_ADF(ASTAR, BSTAR, N, X,
     + Y)
      Y = EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N, X)
      END

      FUNCTION DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N, X)
      X_G3 = 1.0
      ASTAR_G3 = 0.0
      BSTAR_G3 = 0.0
      N_G3 = 0.0
      Y_G3 = 0.0
      CALL DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_ADF_G3(ASTAR, ASTAR_G3, BSTAR
     +, BSTAR_G3, N, N_G3, X, X_G3, Y, Y_G3)
      DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G = Y_G3
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR, N, X
     +)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME = DERIV1_EQLBRM_GMBIGA_GMBI
     +GB_F_G(ASTAR, BSTAR, N, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME_ADF(ASTAR
     +, BSTAR, N, X, Y)
      Y = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR, N, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BS
     +TAR, N, X, Y, YPRIME)
      X_G4 = 1.0
      ASTAR_G4 = 0.0
      BSTAR_G4 = 0.0
      N_G4 = 0.0
      Y_G4 = 0.0
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME_ADF_G4(ASTAR, A
     +STAR_G4, BSTAR, BSTAR_G4, N, N_G4, X, X_G4, Y, Y_G4)
      YPRIME = Y_G4
      END

      FUNCTION ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR,
     + N)
      X = ASTAR
      DO 1669 I = 1, N
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR, N
     +, X, Y, YPRIME)
 1669 X = X-Y/YPRIME
      ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME = X
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G = ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB
     +_F_G_FPRIME(ASTAR, BSTAR, N)
      END

      FUNCTION EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, ASTAR)
      EQLBRM_GMBIGA_GMBIGB_F = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BS
     +TAR, N)-ASTAR
      END

      SUBROUTINE DERIV2_EQLBRM_GMBIGA_GMBIGB_F_ADF(BSTAR, N, X, Y)
      Y = EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X)
      END

      SUBROUTINE DERIV2_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X, Y, YPRIME)
      X_G5 = 1.0
      BSTAR_G5 = 0.0
      N_G5 = 0.0
      Y_G5 = 0.0
      CALL DERIV2_EQLBRM_GMBIGA_GMBIGB_F_ADF_G5(BSTAR, BSTAR_G5, N, N_G5
     +, X, X_G5, Y, Y_G5)
      YPRIME = Y_G5
      END

      FUNCTION ROOT_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X0)
      X = X0
      DO 1669 I = 1, N
      CALL DERIV2_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X, Y, YPRIME)
 1669 X = X-Y/YPRIME
      ROOT_EQLBRM_GMBIGA_GMBIGB_F = X
      END

C     ASTAR & BSTAR: GUESSES IN, OPTIMIZED VALUES OUT
      SUBROUTINE EQLBRM_GMBIGA_GMBIGB(ASTAR, BSTAR, N)
      ASTAR = ROOT_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, ASTAR)
      END

      PROGRAM MAIN
      READ *, ASTAR
      READ *, BSTAR
      READ *, N
      CALL EQLBRM_GMBIGA_GMBIGB(ASTAR, BSTAR, N)
      PRINT *, ASTAR, BSTAR
      END
```

```
#! /bin/bash

tapenade -root deriv1_2_adf2 -d -o eqlbrm42      \
    -diffvarname "_g2" -difffuncname "_g2"       \
    eqlbrm42.f

tapenade -root deriv2_1_2_adf4 -d -o eqlbrm42    \
    -diffvarname "_g4" -difffuncname "_g4"       \
    eqlbrm42{,_g2}.f

tapenade -root deriv1_1_adf1 -d -o eqlbrm42      \
    -diffvarname "_g1" -difffuncname "_g1"       \
    eqlbrm42{,_g2,_g4}.f

tapenade -root deriv2_1_1_adf3 -d -o eqlbrm42    \
    -diffvarname "_g3" -difffuncname "_g3"       \
    eqlbrm42{,_g2,_g4,_g1}.f

tapenade -root deriv2_2_adf5 -d -o eqlbrm42      \
    -diffvarname "_g5" -difffuncname "_g5"       \
    eqlbrm42{,_g2,_g4,_g1,_g3}.f
```

bugs

“issues”

BUGS BUGS BUGS! (Just a taste of the yummy bug juice.)

(a) Deliberate errors on corner cases complicates automated use.
(E.g., refusal to process routine with no active inputs or no body.)

(b) Inconsistent or incomplete or unpredictable name transformation.
- `DIFFSIZES.INC`, `ND`, `NBDIRS`, `NBDIRSMAX`, etc.
- ADIFOR prefixing interacts poorly with **`IMPLICIT`** declarations.
- One-character ADIFOR prefix limit constitutes a limited resource.

(c) Activity declarations are not respected.
- Only sometimes.
- By both ADIFOR and TAPENADE.
  - In different ways.
  - That vary by release.
  - (sometimes they determine an input inactive but if you *ask* for it to be inactive it will be made active.)
- Need to know in order to call.
- Must parse tool output to determine.

(d) Reverse mode sometimes does not generate primal result.
- Must parse tool output to determine.
- Workarounds can interact catastrophically with impure primal.

(e) No way to make cotangent inputs dependent upon primal result.

(f) Many nesting issues, e.g., ADIFOR by default generates singularity check code which cannot itself be transformed, must set `AD_EXCEPTION_FLAVOR=performance`

EQLBRM example in VLAD

Same "EQLBRM" example in VLAD

$$
\begin{aligned}
&\textsc{eqlbrm}(A, B, a_0, b_0, n) \triangleq \\
&\quad \textbf{let } a^* = \textsc{root}_{a^*}(\textsc{argmax}_a(A(a, (\textsc{argmax}_b(B(a^*, b), b_0, n))), a_0, n) - a^*, a_0, n) \\
&\qquad\; b^* = \textsc{argmax}_b(B(a^*, b), b_0, n) \\
&\quad \textbf{in } \langle a^*, b^* \rangle \\
&\textsc{argmax}(f, x_0, n) \triangleq \textbf{let } f'(x) \triangleq \textsc{deriv1}(f, x) \textbf{ in } \textsc{root}(f', x_0, n) \\
&\textsc{root}(f, x, n) \triangleq \\
&\quad \textbf{if } n = 0 \textbf{ then } x \textbf{ else let } \langle y, y' \rangle = \textsc{deriv2}(f, x) \textbf{ in } \textsc{root}(f, x - \frac{y}{y'}, n - 1) \\
&\textsc{deriv1}(f, x) \triangleq \textbf{let } \langle x, \acute{y} \rangle = \overrightarrow{\mathcal{J}}(f, x, 1) \textbf{ in } \acute{y} \\
&\textsc{deriv2}(f, x) \triangleq \overrightarrow{\mathcal{J}}(f, x, 1)
\end{aligned}
$$

The $\overrightarrow{\mathcal{J}}$ symbol is a differential-geometric pushforward operator, analogous to the FARFEL **ADF** block construct.

There is an analogous $\overleftarrow{\mathcal{J}}$ pullback operator that corresponds to the **ADR** block construct.

For details on the language and its implementation see Pearlmutter and Siskind (2008a,b); Siskind and Pearlmutter (2008).

performance

Printed in England

Stafford & Co. Ltd, Nottingham and London

Farfallen		
Tapenade	Adifor	Stalingrad
6.97	8.92	5.83

AVAILABLE NOW!

lessons

seeking PhD students / postdocs

Support

This work was supported, in part, by Science Foundation Ireland grant 09/IN.1/I2637, National Science Foundation grant CCF-0438806, the Naval Research Laboratory under Contract Number N00173-10-1-G023, and the Army Research Laboratory accomplished under Cooperative Agreement Number W911NF-10-2-0060. Any views, opinions, findings, conclusions, or recommendations contained or expressed in this document or material are those of the author(s) and do not necessarily reflect or represent the views or official policies, either expressed or implied, of SFI, NSF, NRL, the Office of Naval Research, ARL, or the Irish or U.S. Governments.

References

U. Naumann and J. Riehme. A differentiation-enabled Fortran 95 compiler. *ACM Transactions on Mathematical Software*, 31(4):458–474, 2005. URL http://doi.acm.org/10.1145/1114268.1114270.

B. A. Pearlmutter and J. M. Siskind. Using programming language theory to make automatic differentiation sound and efficient. In C. H. Bischof, H. M. Bücker, P. D. Hovland, U. Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, volume 64 of *Lecture Notes in Computational Science and Engineering*, pages 79–90. Springer, Berlin, 2008a. ISBN 978-3-540-68935-5. doi: 10.1007/978-3-540-68942-3_8.

B. A. Pearlmutter and J. M. Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. on Programming Languages and Systems*, 30(2): 1–36, Mar. 2008b. doi: 10.1145/1330017.1330018.

J. M. Siskind and B. A. Pearlmutter. Using polyvariant union-free flow analysis to compile a higher-order functional-programming language with a first-class derivative operator to efficient Fortran-like code. Technical Report TR-ECE-08-01, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, jan 2008. URL http://docs.lib.purdue.edu/ecetr/367.

B. Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, January 1980.

R. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7 (8):463–464, 1964.

FAQs

FORTRAN 77

We would *love* to extend FARFALLEN from FORTRAN 77 to FORTRAN 95 or FORTRAN 2008 *aka* ISO/IEC 1539-1:2010.

no

yes

GAME EXAMPLE DISCLAIMER

The EXISTENCE OR UNIQUENESS OF AN EQUILIBRIUM IS NOT IN GENERAL GUARANTEED, but our particular A and B have a unique equilibrium.

COORDINATE DESCENT (alternating optimization of a^* and b^*) would require less nesting, but HAS INFERIOR CONVERGENCE PROPERTIES.

Although this example involves AD THROUGH ITERATIVE PROCESSES, we do not address that issue in this work: it IS BEYOND THE SCOPE OF THIS PAPER, and used here only in a benign fashion, for vividness.

On our concrete objective functions THESE CONVERGE RAPIDLY, so for clarity we SKIP the clutter of CONVERGENCE DETECTION.


CRAWLING, SLIMY THINGS
TERROR-BENT ON
DESTROYING THE WORLD!
the
Brain
Eaters
Starring EDWIN NELSON · JOANNA LEE · ALAN FROST · Produced by EDWIN NELSON · Directed by BRUNO VESOTA · Story & Screenplay by GORDON URQUHART · AN AMERICAN INTERNATIONAL PICTURE
58/419

FARFEL *vs* Naumann and Riehme (2005)

FARFEL constructs and features that the NAGWARE 95 AD extensions could not, to our knowledge, handle:

- **EXTERNAL** arguments
- nested internal definitions (lexically scoped)
- derivatives of derivatives (nesting)

This would impact all of the examples in this talk.