PCOMP: A Modeling Language for Nonlinear Programs with Automatic Differentiation

Klaus Schittkowski
Department of Mathematics
University of Bayreuth

Purpose: automatic differentiation
PCOMP syntax and program structure
case study: data fitting in dynamical systems (EASY-FIT)

Design Goals

1. modeling nonlinear functions or dynamical equations for technical and scientific applications in engineering and natural sciences

2. Fortran-similar syntax

3. flexible program organization in form of a Fortran tool box without fixed attachment to domain-specific modeling system

4. direct interpretation of given code for function and derivative computations

5. generation of efficient Fortran code for function and gradient evaluations

6. extendable by user-provided symbols and functions

Purpose: Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. Compute $\nabla f(x)$

Function evaluation: Let f_i defined on $\mathbb{R}^{n_i}$, $i = n+1, \ldots, m$, be a sequence of elementary functions for f, $m \geq n$ with index sets $J_i \subset \{1, \ldots, i-1\}$, $|J_i| = n_i$, $i = n+1, \ldots, m$, such that a function value $f(x)$ is evaluated according to the following program:

$$\begin{aligned} &\text{For} \quad i = n+1, \ldots, m \text{ let} \\ &\qquad x_i = f_i(x_k, k \in J_i) \quad . \\ &\text{Let} \quad f(x) = x_m \quad . \end{aligned}$$

See also: ADIFOR (Bischof et al.,1992), ADOL-C (Griewank et al., 1991)

Example:

$$f(x_1, x_2) = (x_1 + x_2)x_1 + \sin(x_2)$$

Function and gradient evaluation:

$$\begin{aligned}
x_3 &= f_3(x_1, x_2) = x_1 + x_2 \ , & J_3 &= \{1, 2\} \ , \\
x_4 &= f_4(x_3, x_1) = x_3 x_1 \ , & J_4 &= \{3, 1\} \ , \\
x_5 &= f_5(x_2) = \sin(x_2) \ , & J_5 &= \{2\} \ , \\
x_6 &= f_6(x_4, x_5) = x_4 + x_5 \ , & J_6 &= \{4, 5\} \ .
\end{aligned}$$

Result: $f(x_1, x_2) = x_6$

Forward mode: Compute derivatives within forward loop by chain rule

For $i = 1, ..., n$ let

$$\nabla x_i = e_i$$

For $i = n+1, ..., m$ let

$$x_i = f_i(x_k, k \in J_i),$$

$$\nabla x_i = \sum_{j \in J_i} \frac{\partial f_i(x_k, k \in J_i)}{\partial x_j} \nabla x_j$$

Let $f(x) = x_m$,

$$\nabla f(x) = \nabla x_m$$

Reverse mode: Compute intermediate variables $x_{n+1}, \ldots, x_m$, then perform reverse sweep

For $i = n+1, \ldots, m$ let

$$x_i = f_i(x_k, k \in J_i) \ , \quad y_i = 0 \ .$$

Let $f(x) = x_m$,

$$y_i = 0 \ , \ i = 1, \ldots, n \ , \quad y_m = 1 \ .$$

For $i = m, m-1, \ldots, n+1$ let

$$y_j = y_j + \frac{\partial f_i(x_k, k \in J_i)}{\partial x_j} \, y_i \text{ for all } j \in J_i \ .$$

Let $f(x) = x_m$, $\nabla f(x) = (y_1, \ldots, y_n)^T$.

Theorem (Griewank, 1989):

The work ratio of the reverse mode, i.e., the relative calculation time of one function and one gradient evaluation subject to one function evaluation alone, is bounded by 5.

Example: Helmholtz energy function

$$f(x) = RT \sum_{i=1}^{n} x_i \log\left(\frac{x_i}{1 - b^T x}\right) - \frac{x^T A x}{\sqrt{8} b^T x} \log\left(\frac{1 + (1 + \sqrt{2}) b^T x}{1 + (1 - \sqrt{2}) b^T x}\right)$$

Numerical results:

n	W_{rev}	W_{fwd}	W_{num}
25	3.03	4.04	26.01
50	3.13	6.63	51.01
100	3.76	11.88	100.99
200	3.33	22.47	200.89
400	3.70	45.77	400.88
800	3.49	81.59	802.56

Block structure:

* PARAMETER
* INDEX
* INTEGER CONSTANT
* VARIABLE
* LININT <identifier>
* MACRO <identifier>
* END
* SET OF INDICES
* REAL CONSTANT
* TABLE <identifier>
* CONINT <identifier>
* SPLINE <identifier>
* FUNCTION <identifier>

PCOMP Syntax (continued)

Format: Similar to Fortran

- arithmetic expressions:

```
A = 2.0*X**2 - 1.2E-5*X*(1 - Y)
```

- intrinsic functions:

```
ABS, SIN, COS, TAN, ASIN, ACOS, ATAN, SINH, COSH
TANH, ASINH, ACOSH, ATANH, EXP, LOG, LOG10, SQRT
```

- sums and products over index sets:

```
*     FUNCTION f
      f = 100*PROD(x(i)**a(i), i IN inda)
```

Control statements:

```
    IF ⟨condition⟩ THEN
        ⟨statements⟩
    ENDIF
    IF ⟨condition⟩ THEN
        ⟨statements⟩
    ELSE
        ⟨statements⟩
    ENDIF
    GOTO ⟨label⟩
⟨label⟩ CONTINUE
```

Example:

```
*      PARAMETER
       n = 100
*      SET OF INDICES
       index = 1..n
*      REAL CONSTANT
       R = 8.314
       T = 273
       A(I,J) = 1/(i+j-1), i IN index, j IN index
       b(i) = 0.00001, i IN index
*      VARIABLE
       x(i), i IN index
*      FUNCTION f
       bx = SUM(b(i)*x(i), i IN index)
       xAx = SUM(x(i)*SUM(A(i,j)*x(j), j IN index), i IN index)
       f = R*T*SUM(x(I)*LOG(x(i)/(1 - bx)),i IN index)
      /    - xAx*LOG((1 + (1+SQRT(2))*bx)/(1 + (1-SQRT(2))*bx))/(SQRT(8)*bx)
```

Fortran subroutines:

1. Parser: Analyze source code, generation of intermediate code based on formal grammar, parser generated by *yacc* (SYMINP)

2. Interpreter: Compute function, gradient, and Hessian values by interpreting intermediate code in forward mode (SYMFUN, SYMGRA, SYMHES)

3. Code generator: Generate Fortran subroutines for function and gradient evaluation (XFUN, XGRA) in reverse mode

Important: New formal expressions and external functions can be attached!

Applications:

- simulation and control of a tubular reactor given in form of a distributed system (Birk et al., 1999), also differentiation of high-order Runge-Kutta formula
- optimal control of ordinary and one dimensional partial differential equations (PDECON, Blatt and Schittkowski, 2000)
- interactive nonlinear programming including multicriteria, least squares, min-max, and L_1 optimization (EASY-OPT, Schittkowski, 1999)
- data fitting in dynamical systems (EASY-FIT, Schittkowski,2002)

Case Study: EASY-FIT

Purpose: Interactive data fitting in

- explicit model functions
- steady-state systems
- Laplace transformations of differential equations
- ordinary differential equations
- differential algebraic equations
- one-dimensional partial differential equations
- one-dimensional partial differential algebraic equations

Least squares problem: Only ODE considered

$$
\begin{aligned}
& \min \textstyle\sum_{i=1}^{n_t} \sum_{k=1}^{n_m} w_i^k (h_k(x, y(x, t_i), t_i) - y_i^k)^2 \\
& g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \\
x \in \mathbb{R}^n : \quad & g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \\
& x_l \leq x \leq x_u
\end{aligned}
$$

where $y(x,t)$ solution of system of ordinary differential equations

$$
\dot{y} = F(x, y, t) \ , \quad y(0) = y^0(x)
$$

Case Study: EASY-FIT (continued)

Some features: Only for systems of ODEs

- alternative norms (L_1, L_∞)
- additional independent model parameter
- switching points, also variable ones
- shooting method
- dynamic constraints
- 463 ODE test examples (among 1,000 in total)

See also: Schittkowski K. (2002): *Numerical Data Fitting in Dynamical Systems - A Practical introduction with Applications and Software*, Kluwer

Automatic differentiation:

1. Partial differentiation of $h_k(x, y(x,t), t)$ subject to x and y

2. Differentiation of $y(x,t)$ subject to x by sensitivity equations, internal numerical differentiation, ..., requiring partial derivatives of $F(x,y,t)$ subject to x and y and derivative of $y^0(x)$

3. Implicit solvers for stiff ODEs require partial derivatives of $F(x,y,t)$ subject to y

4. Partial differentiation of constraints

Case Study: EASY-FIT (continued)

Example: Cooling of a crystallization process

Constants:

```
*     REAL CONSTANT
      mh2o = 1659.8
      rhos = 2.11
      kv = 1
      ka = 6
      ls = 0.2
      g = 1.32
      b = 1.78
*     SPLINE Te
       0.0          32.0400
       0.35000      32.0190
       0.60000      31.9953
       ...
       80.8600      28.9252
       81.1100      28.9297
```

Variables: Partial derivatives subject to subsets

```
C
C-------------------------------------------------
C
C   - Independent variables in the following order:
C     1. parameters to be estimated (x)
C     2. variables identifying solution of ordinary
C        differential equations (y)
C     3. concentration variable, if exists (c)
C     4. time variable (t)
C
*     VARIABLE
      kg, kb, m, mue0a, mue1a, mue2a, mue3a,  t
```

Differential equations:

```
*     FUNCTION dm_t
      cs = 0.1286 + 5.88E-3*Te(t) + 1.721E-4*Te(t)**2
      S = (m/mh2o - cs)/ cs
      IF (S.GT.0) THEN
        xG = kg*S**g
      ELSE
        xG = 0
      ENDIF
      dm_t = -3*kv*rhos*xG*mue2a
C
*     FUNCTION dmue0a_t
      IF (S.GT.0) THEN
        xB = EXP(kb)*mue3a*S**b
      ELSE
        xB = 0
      ENDIF
      dmue0a_t = xB
      ......
```

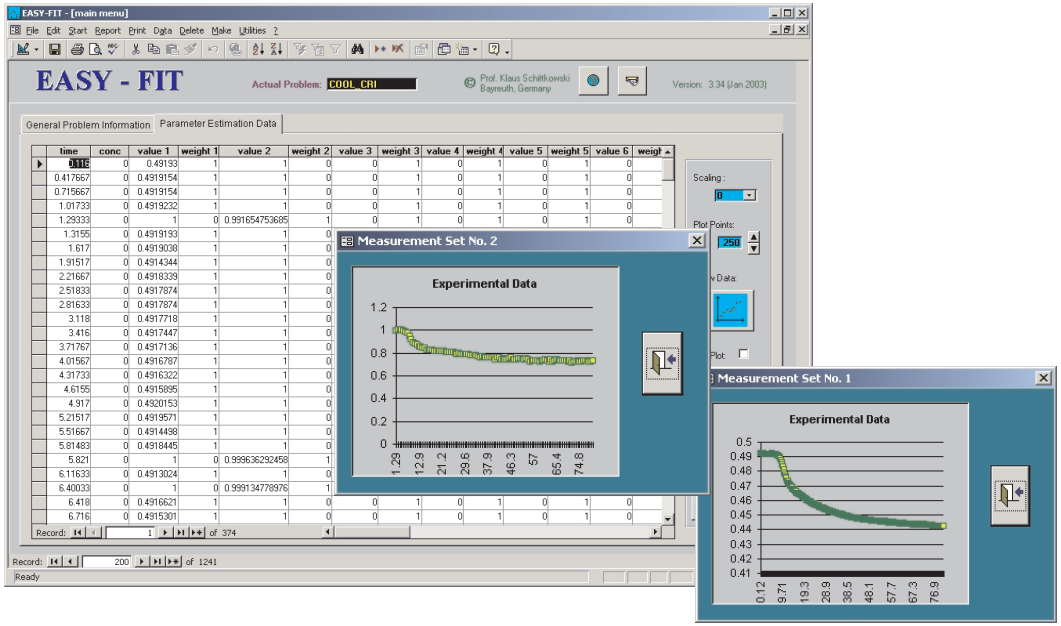
Initial values:

```
C
*     FUNCTION m0
      m0 = 0.4922*mh2o
C
*     FUNCTION mue0a0
      mue0a0 = 0.811*mh2o
C
*     FUNCTION mue1a0
      mue1a0 = 1.59E-2*mh2o
C
*     FUNCTION mue2a0
      mue2a0 = 3.11E-4*mh2o
C
*     FUNCTION mue3a0
      mue3a0 = 6.1E-6*mh2o
```

Fitting criteria:

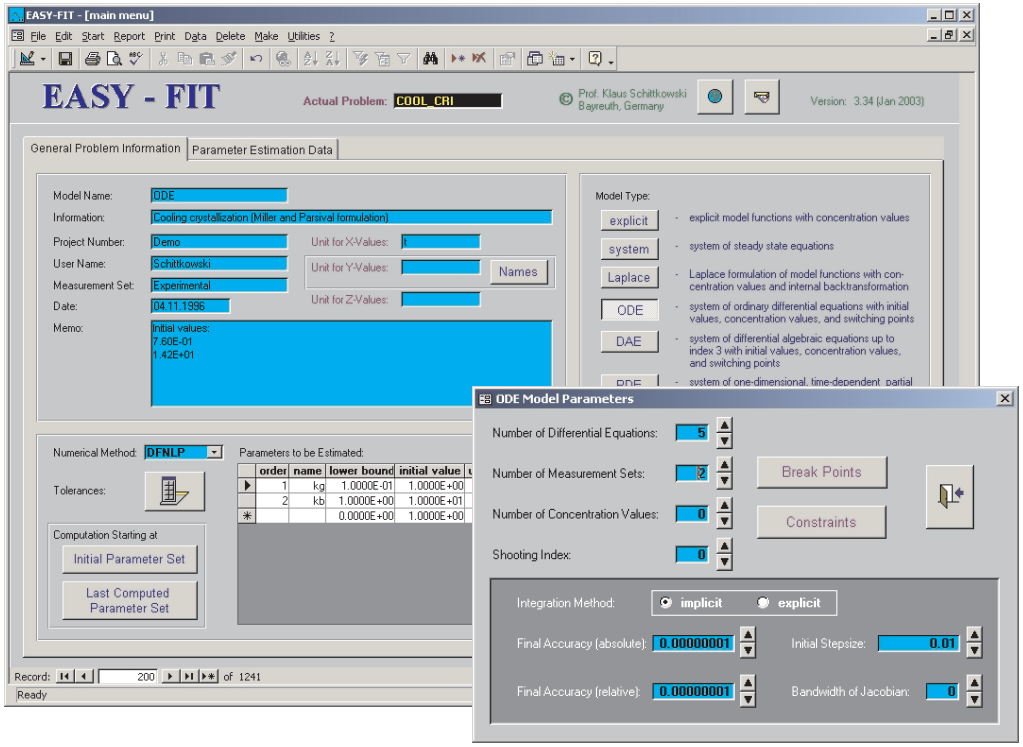
```
C
*     FUNCTION FIT1
      FIT1 = m/mh2o
c
*     FUNCTION FIT2
      X = m/mh2o
      rho28 = 0.4214*X + 1.0236
      rho32 = 0.5433*X + 0.9667
      rhof  = (rho32 - rho28)/4.0*(Te(t) - 28) + rho28
      Vges  = (m + mh2o)/rhof + kv*mue3a
      convert = mh2o/Vges
      FIT2 = EXP(-ka/2*ls*convert*mue2a/mh2o)
```

Step 1: Create a new problem in the database, insert some information strings and in particular experimental data



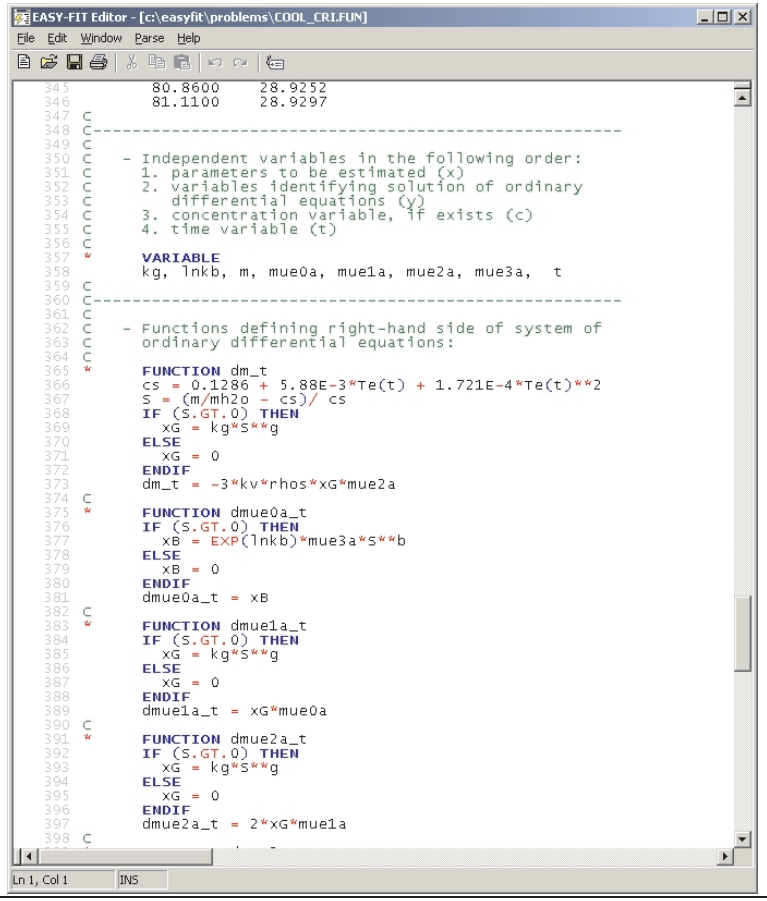
Case Study: EASY-FIT (continued)

Step 2: Choose type of dynamical model (ODE, PDE, ...), define model structure, and set discretization parameters

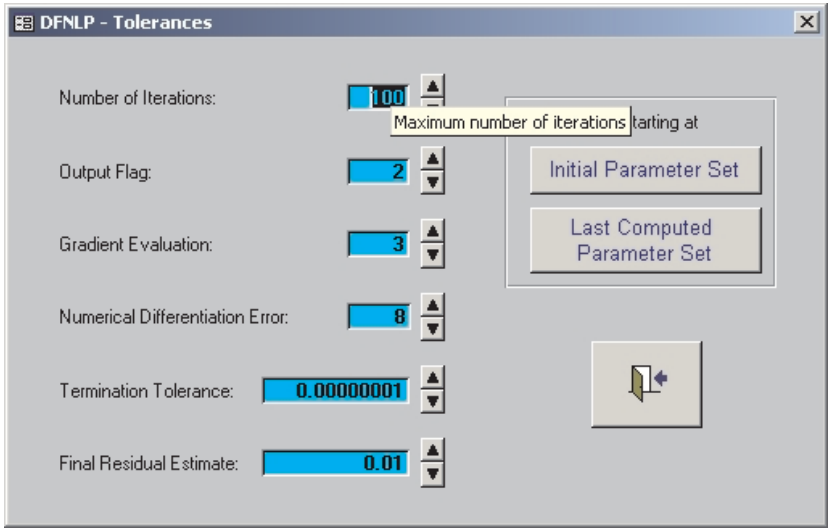


Case Study: EASY-FIT (continued)

Step 3: Implement model equations and check correct syntax



Step 4: Define parameters to be estimated, select least squares solver, set termination tolerances, and start data fitting run



Step 5: Check report, especially parameter values and residuals