

# Synthetic Calculus -- A Paradigm of Mathematical Program Synthesis

Joseph Thames, Digital Calculus Corporation

## INTRODUCTION

For more than 30 years, scientific programming has addressed applications at an *algorithmic* level of mathematical problem description, pre-reduced from the *user* level of problem declaration to the level of numerical computation. Automatic differentiation has made it possible to transcend this level to a higher order of *non-algorithmic* programming, where the description is a declarative model of the application problem to be solved. The model description remains essentially algebraic, but the numerical analysis is not explicitly described. Thus the model is a pure application description and the solution methods are modularized tools, hidden and interchangeable.

This is a new paradigm of program synthesis, which we call *synthetic calculus*. It has been the basis of several modeling languages including the one shown here, an extension of FORTRAN 77. This is the initial implementation of our new product, which we call *PowerCalculus™*. In it, the declarative description of the application model is literally called a MODEL. The core of *PowerCalculus™* is the inclusion of automatic differentiation and a family of numerical solution techniques and coordinate transformations based on it, all of which are hidden within the language. The solution techniques are called *solvers* and are invoked via language operators, such as the FIND statement, the operator for solving inverse problems.

We will discuss two major levels of software synthesis and demonstrate some example applications. First is "basic synthesis", the application of automatic differentiation to re-engineer (remodel) existing source code for optimization. Next is "advanced synthesis", the capability of general purpose non-algorithmic programming that we call *structured calculus*. This leverages the power of basic synthesis via nesting.

## BASIC SYNTHESIS

Figure 1 is a case study of re-engineering -- a simulation program for A.C. motor design that was reapplied as an optimization program and fully tested in about an

```

GLOBAL ALL
PROBLEM ACMOTOR
DIMENSION CNS(7)
DYNAMIC BOTM,BND
CALL INPUT
CALL DESIGN
PRINT *,----- INITIAL DESIGN -----
CALL OUTPUT
FIND COILTRNS: I Number of turns per coil
I SEPDIAM: I Separating diameter
I STASLOTW: I Stator slot opening width
I ROTSLOTW: I Rotor slot opening width
I AIRGAP: I Air gap
I STATOOTW: I Stator tooth width
I STABAKIR: I Stator back iron
I ROTTOOTW: I Rotor tooth width
I ROTBAKIR: I Rotor back iron
I STASLOTO: I Stator slot opening depth
I ROTSLOTO: I Rotor slot opening depth
I SLIP: I Slip
I IN DESIGN; BY THOR(TCON);
I WITH BOUNDS BND; AND LOWERS BOTM;
I HOLDING CNS; TO MAXIMIZE EFF
PRINT *,----- OPTIMIZED DESIGN -----
CALL OUTPUT
END
MODEL DESIGN
F1 = 231258.0438/COILTRNS: Q1 = SEPDIAM*PI/56
Q2 = SEPDIAM*PI/68: Q3 = SEPDIAM*PI/28
C1 = Q1/((Q1-STASLOTW) + (STASLOTW*(.7-.036*(STASLOTW/AIRGAP))))
C2 = Q2/((Q2-ROTSLOTW) + (ROTSLOTW*(.7-.036*(ROTSLOTW/AIRGAP))))
CRNOL = F1*AIRGAP*C1*C2/(8.96*COILTRNS*Q3)
Q4 = (STATOROD-STABAKIR-STABAKIR-SEPDIAM-.1)/2-.01
Q5 = (SEPDIAM + Q4 + .1)*PI/56-STATOOTW-.018: A3 = Q4*Q5: Z1 = 7000*A3
Q6 = COILTRNS*2/Z1: Z3 = -2: Z4 = 32 + Z3: Q7 = 1.28**Z3*182
Q8 = 64/1.28**Z3: Z5 = STACKLEN + STACKLEN + PI*STATOROD/14
RDC = Z5*COILTRNS**28*Q7/12000
V1 = (SEPDIAM-.06-ROTORID-ROTBKIR-ROTBKIR)/2
V2 = (SEPDIAM-.06-V1)*PI/68-ROTTOTW: A4 = V1*V2
R2 = (.0133/A4)*(STACKLEN/12000)*U1
RROT = (58*COILTRNS)**2*R2/138: R4 = RROT
G3 = 3.2*(Q4/Q3*Q5) + .03/(STASLOTW + Q5) + STASLOTO/STASLOTW
G4 = 3.2*(V1/(3*V2) + .03/(ROTSLOTW + V2) + ROTSLOTO/ROTSLOTW)
G5 = (1/C1 + 1/C2-1)**2*.26/AIRGAP
G6 = (G3 + G5*Q1)/56 + (G4 + G5*Q2)/68
XLEAK = XC*PI*COILTRNS**2*(.05 + G6*STACKLEN) I Leakage reactance
CRSTAL = 115/(SQRT(((RDC + R4)**2) + XLEAK**2)) I Stall current
CRFL = 115/(SQRT(((RDC + R4/SLIP)**2) + XLEAK**2)) I Full load current
STORQ = 1.5792*(CRSTAL**2)*R4 I Stall torque
RTORQ = 1.5792*(CRFL**2)*R4/SLIP I Running torque
P6 = RTORQ*(1-SLIP)*1.284944649: P6 = (CRSTAL**2)*(RDC + R4)
P7 = (CRFL**2)*R4/SLIP
FLUX = (1.4*F1/(STATOOTW*STACKLEN))/6450 I Flux density
B7 = (SQRT(CRFL**2 + CRNOL**2)*FLUX/CRNOL)/1.4: B2 = .13*B7**1.9
B3 = (((STATOROD**2-SEPDIAM**2)*PI/4)-56*A3)*0.28*STACKLEN
FELOSS = B3*B2 I Iron losses
CULSTA = (B7*CRNOL/FLUX)**2*RDC I Stator copper losses
EFF = .5*P6/(P7 + FELOSS/2 + CULSTA) I Efficiency - maximize this
RAC = 13225/FELOSS I A.C. resistance
XMAG = 115/CRNOL I Magnetizing reactance
CULROT = CRFL*CRFL*RROT/STASLOTW I Rotor copper losses
CNS(1) = STATOROD-.1-SEPDIAM I Stator O.D = Separation Diam +.1
CNS(2) = SEPDIAM-ROTORID +.1 I Separation Diam = Rotor LD -.1
CNS(3) = ((SEPDIAM*PI/68) -.035)-ROTTOTW I Rotor tooth geometry
CNS(4) = .5*(SEPDIAM-ROTORID) -.025-ROTBKIR I Rotor back iron geometry
CNS(5) = STORQ-60 I Stall torque = 60
CNS(6) = 18-FLUX I Flux density LE 19
CNS(7) = .05-SLIP I Slip LE 5 percent
END
    
```

Figure 1. A.C. Motor Design Optimization Model

hour. The original program had been used for trial & error design calculations by a west coast manufacturer. To synthesize this mature engineering model into a design optimization program, all that was necessary was to introduce a FIND statement defining a constrained optimization problem with 12 unknown parameters and 7 constraints. The solver involved in this case is a nonlinear programming algorithm called THOR. Figure 2 (part of the output) is a report automatically generated by THOR, summarizing the iterations. This shows the initial guesses of the parameters and the final results.

```

---THOR SUMMARY, INVOKED AT ACMOTOR[20] FOR MODEL DESIGN ---

CONVERGENCE CONDITION AFTER 35 ITERATIONS
UNKNOWN CONVERGED
OBJECTIVE CRITERION UNSATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ----- [INITIAL]      35
UNKNOWN
COILTRNS      2.900000E+01  3.214302E+01
SEPDIAM      4.000000E+00  5.010000E+00
STASLOTW     8.000000E-02  8.570807E-02
ROTSLOTW     3.500000E-02  3.500000E-02
AIRGAP       8.000000E-03  8.000000E-03
STATOOTW     1.350000E-01  1.687119E-01
STABAKIR     1.500000E-01  3.855060E-01
ROTTOOTW     1.100000E-01  8.885808E-02
ROTBKIR      4.700000E-01  8.700004E-01
STASLOTO     3.500000E-02  2.500000E-02
ROTSLOTO     1.500000E-02  0.000000E+00
SLIP         3.000000E-02  5.000000E-02
OBJECTIVE
EFF          4.020854E-01  7.312523E-01
INEQUALITY CONSTRAINTS
CNS ( 1)    1.010000E+00  0.000000E+00
CNS ( 2)    1.225000E+00  2.235000E+00
CNS ( 3)    3.712132E-02  1.042519E-01
CNS ( 4)    8.750000E-02  7.248863E-02
CNS ( 5)   -7.558836E+00  8.415480E-04
CNS ( 6)    1.258833E+01  1.437190E+01
CNS ( 7)    2.000000E-02  0.000000E+00
---END OF LOOP SUMMARY

```

Figure 2. A.C. Motor Output Report

This case study illustrates the economic benefits of synthetic calculus for reapplying existing application software in a higher-productivity mode. The existing engineering simulation model was used "as is". It was automatically *elevated* to optimization by the hidden differential arithmetic. There was no mathematical analysis or algorithm design required. Because this was a re-engineering of an existing program, no debugging was necessary, and since the optimization tools are interchangeable, different algorithms could be substituted to verify correctness.

As this example illustrates, the benefits of multiple-parameter optimization in practical engineering calculation can often be dramatic. Because of the large number of parameters, something on the order of a billion to a trillion cases would have been necessary to achieve this result by parameterizing the original program. The bottom line benefit is dramatically reduced

cost, higher engineering productivity, and immediate results to meet tight schedules.

The next two examples illustrate another form of the FIND statement for solving implicit equation systems with a solver called AJAX. The first (Figure 3) is the solution of seven simultaneous nonlinear equations representing a classical chemical equilibrium problem. The model equations simply define a vector of equality constraints to be matched to zero. The icon symbolizes this FIND...MATCH program structure to solve implicit algebraic equations (IAE) as a *primitive* of program synthesis. In Figure 4 we see the characteristic summary report generated by the FIND statement, listing the unknowns and constraints throughout the iterative solution process.

```

PROBLEM REACT(10000,1000,1000)
DIMENSION X(7),F(7)
DATA X/5.0,0.0,0.0,5.0,0.0,5.2,0/
FIND X; IN MOLES(X,F); BY AJAX; TO MATCH F
END

MODEL MOLES(X,F)
DIMENSION X(7),F(7)
F(1) = X(1)/2 + X(2) + X(3)/2 - X(6)/X(7)
F(2) = X(3) + X(4) + 2*X(5) - 2/X(7)
F(3) = X(1) + X(2) + X(5) - 1/X(7)
F(4) = -28837*X(1) - 139069*X(2) - 78213*X(3) + 18927*X(4)
      + 8427*X(5) + 13482/X(7) - 10690*X(6)/X(7)
F(5) = X(1) + X(2) + X(3) + X(4) + X(5) - 1
F(6) = 400*X(1)*X(4)**3 - 1.7837E5*X(3)*X(5)
F(7) = X(1)*X(3) - 2.6058*X(2)*X(4)
END

```

IAE  
▽  
Icon

Figure 3. Chemical Equilibrium Problem

```

--- AJAX SUMMARY, INVOKED AT REACT[4] FOR MODEL MOLES ---

CONVERGENCE CONDITION AFTER 10 ITERATIONS
UNKNOWN CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER --- [INITIAL]      9      10
UNKNOWN
X ( 1)    5.000000E-01  3.228708E-01  3.228708E-01
X ( 2)    0.000000E+00  9.223544E-03  9.223544E-03
X ( 3)    0.000000E+00  4.601709E-02  4.601709E-02
X ( 4)    5.000000E-01  8.181717E-01  8.181717E-01
X ( 5)    0.000000E+00  3.716851E-03  3.716851E-03
X ( 6)    5.000000E-01  5.767154E-01  5.767154E-01
X ( 7)    2.000000E+00  2.977863E+00  2.977863E+00
CONSTRAINTS
F ( 1)    0.000000E+00  -1.383578E-08  -1.819538E-13
F ( 2)   -5.000000E-01  -9.598910E-08  -1.390108E-11
F ( 3)    0.000000E+00  -4.798495E-08  1.418820E-11
F ( 4)   -8.815000E+02  4.995033E-04  2.473498E-07
F ( 5)    0.000000E+00  0.000000E+00  1.079581E-11
F ( 6)    2.500000E+01  8.865512E-07  1.952332E-08
F ( 7)    0.000000E+00  -1.060134E-11  8.205562E-12
---END OF LOOP SUMMARY

```

Figure 4. Chemical Equilibrium Problem Output

The second IAE problem (Figure 5) is an admittance circuit fitting problem involving an overdetermined system of 21 equations in 3 unknowns. In this case the FIND statement implements a Newton-Gauss method minimizing the Euclidean norm of the constraint vector R, relating the residuals between computed and measured admittance. The output graph (Figure 6)

```

GRAPHICS BOTH
PROBLEM CIRCUIT (5000,1000,1000)
COMMON/PARAMS/EL,ELS,CS
DIMENSION F(21),Y(21),R(21),W(21),YC(21)
DATA Y/1.273,1.278,1.382,1.804,1.708,1.950,2.148,2.297,
~ 2.503,2.883,3.305,4.005,5.077,8.069,14.84,39.47,
~ -15.08,-9.708,-7.388,-5.288,-3.907/
EL = 1.1E-10 : ELS = -1.1E-10 : CS = 1.1E-12
DO 10 I = 1,21
  F(I) = 2850 + 50*I
  W(I) = 6.28318E8*F(I)
10 CONTINUE
@AXES('CIRFIT',F,Y,YC,W)
FIND EL,ELS,CS; IN FIT(F,Y,R,W,YC); BY AJAX; TO MATCH R
@FINISH('CIRFIT',F,YC,W)
END

MODEL FIT(F,Y,R,W,YC)
COMMON/PARAMS/EL,ELS,CS
DIMENSION F(*),Y(*),R(*),W(*),YC(*)
DO 10 I = 1,21
  YC(I) = -1/(W(I)*EL - 1/(W(I)*CS - 1/(W(I)*ELS)))
  R(I) = 1/YC(I) - 1/Y(I)
10 CONTINUE
END

```



Figure 5. Admittance Circuit Fitting Problem

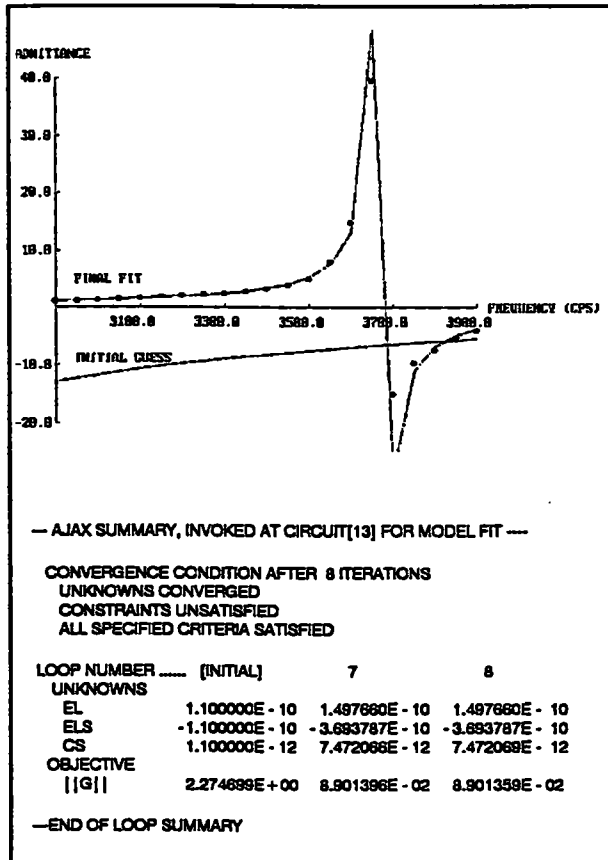


Figure 6. Admittance Circuit Fit

illustrates a benefit of using a problem-equation approach to nonlinear curve fitting, as opposed to a general polynomial approach. The equations, being phenomenological, faithfully reproduce the problem singularities, allowing accurate fitting. In the printed output we see the familiar summary print but with the Euclidean norm shown as the objective function.

## ADVANCED SYNTHESIS

Advanced synthesis (structured calculus) involves the nesting of the basic synthesis structures to address complex inverse problems. It employs the simplicity of hierarchic description for ease of comprehension and the synergistic power of combined algorithms at the same time. The three types of basic synthesis primitives are used a program forming alphabet. These synthesis elements are shown in Figure 7a in formal notation, in characteristic program structure, and with an associated icon for representation in graphical user interfaces. Figure 7b illustrates the constrained form of these operations. The two lower primitives encompass the general field of mathematical programming and the general field of continuous systems simulation. As we will show, synthetic calculus unifies these technologies into a single paradigm. Advanced synthesis is mainly concerned with nesting simulation problems within mathematical programming problems.

Advanced synthesis is enabled by hidden transformations which propagate differentiation. This includes automatic differentiation of integration algorithms and differential coordinate transformations for nesting inverse problems. These transformations are invoked automatically by the solvers at nesting interfaces.

(a) Synthesis Elements - Simultaneous Equations Systems		
Element Alphabet	STRUCTURE	ICON
<b>IAE</b> Implicit Algebraic Equations: $g(x) = 0$	FIND - MATCH MODEL	
<b>UPO</b> Unconstrained Parameter Optimization: $f(x) = \min$	FIND - MINIMIZE MODEL	
<b>ODE</b> Ordinary Differential Equations: $y' = f(x), y_0 = C$	INTEGRATE MODEL	
(b) Limits & Constraints on Synthesis Elements		
Element Alphabet	STRUCTURE	ICON
<b>IAEL</b> IAE with Limits $g(x) = 0, a \leq x \leq b$	FIND - UPPER - LOWER TO MATCH MODEL	
<b>CPO</b> Constrained Parameter Optimization: $f(x) = \min, g(x) = 0, h(x) \geq 0$ LP, NLP, IP, MIP (Mathematical Programming Domain)	FIND - MATCHING - HOLDING TO MINIMIZE MODEL	
<b>LODE</b> - Limited ODEs: $y' = f(x), a \leq y \leq b, y_0 = C$ (Continuous Systems Simulation Domain)	INTEGRATE - UPPER LOWER MODEL	

Figure 7. Program Forming "Alphabets"

## Nested Program Structure

Figure 8 introduces a few examples to illustrate nesting of the synthesis primitives to form composite program structures, symbolized by combined icons.

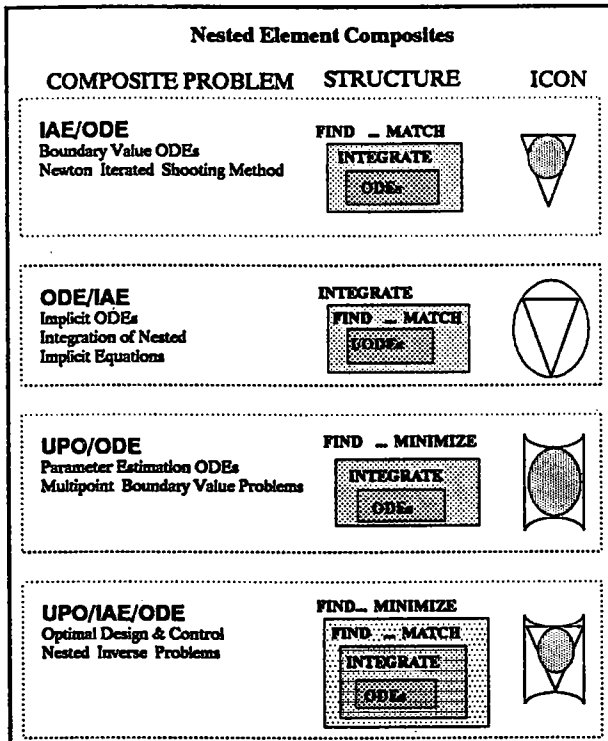


Figure 8. Structured Element Combinations

**Pilot Ejection as a Boundary-Value Problem** - The first example (Figure 9) illustrates the use of the IAE/ODE structure to solve a boundary-value problem. This involves the nesting of differential equations (the INTEGRATE primitive) inside a system of implicit equations (the FIND...MATCH primitive).

This is the well-known Pilot Ejection problem often used to illustrate simulation languages, but in this case it is solved in a loop using altitude as a parameter, and the aircraft speed and seat trajectory time are solved as unknowns by the FIND statement to match boundary condition constraints GX and GY which specify minimum clearance for safe ejection. The outer solver invoked is AJAX.

The differential equations reside in the model MOTION. The initial value problem to solve these equations is defined and initialized by the INITIATE statement. This statement connects the solver ISIS to the equations, identifying the rates to be integrated, the states that are the output of integration, the independent variable, the integration step size and the upper limit of integration. The INTEGRATE state-

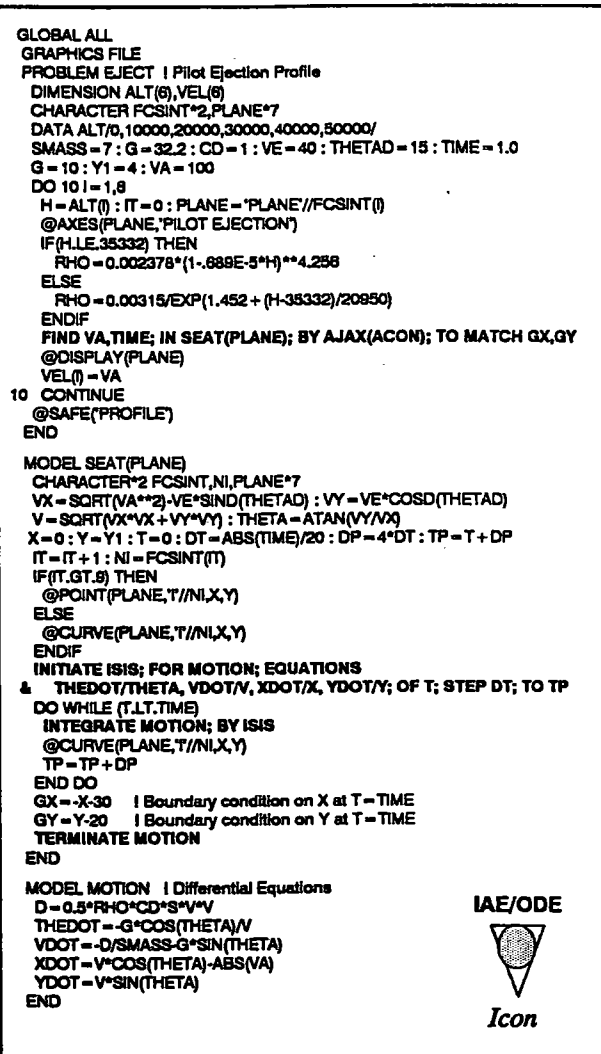


Figure 9. Pilot Ejection Boundary-Value Problem

ment then generates the trajectory using the solver ISIS. In this case the integration is piecemeal in order to plot trajectories.

Figure 10a-b shows the iteration of trajectories for two parameterized altitudes to converge on the point above the rudder defined as minimum clearance for safe ejection. The aircraft speed computed at each altitude is the maximum speed that meets this criteria. Safe speed increases with altitude, and fewer iterations are required because the previous result is a good guess.

This process is a Newton iterated shooting method where the numerical integration algorithm is being differentiated as it is executed, producing partial derivatives of boundary conditions with respect to initial conditions. This process is not possible with symbolic differentiation methods because no formula of the integral function exists. It exemplifies why automatic differentiation is a programming breakthrough.

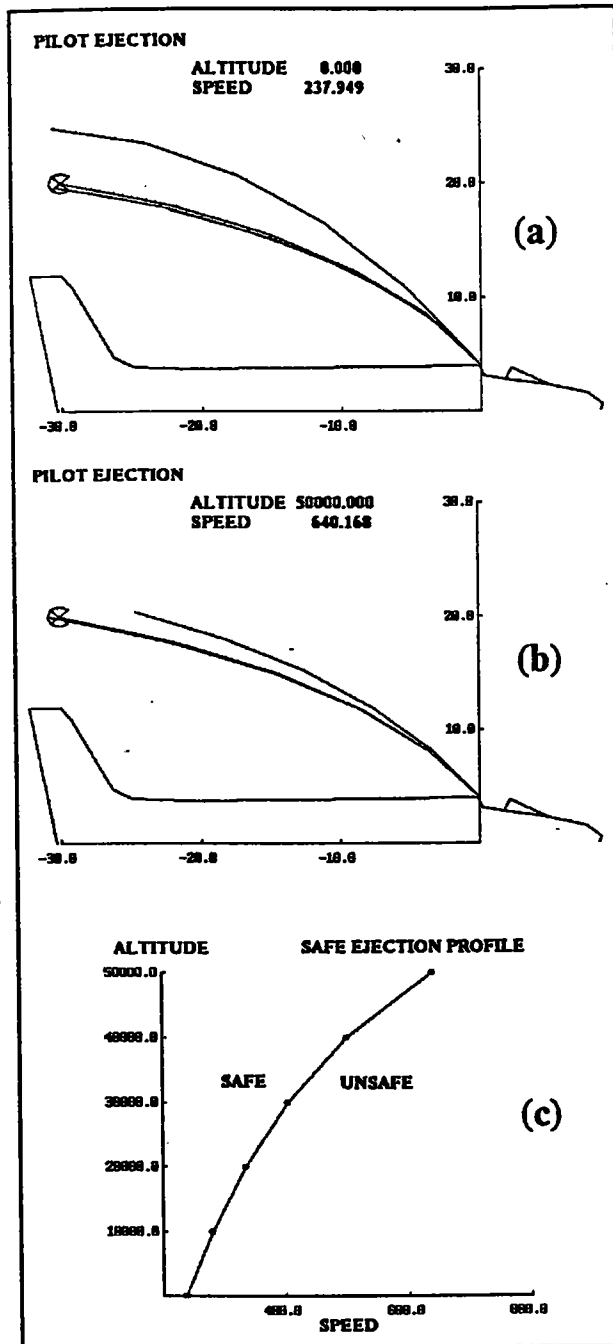


Figure 10. Pilot Ejection Iterations and Safe Profile

The final result (Figure 10c) is a plot of the safe ejection profile, created in a single run. Using the old trial & error simulation approach, this would have required 30 to 60 runs.

**Implicit Differential Equations** - The second example, illustrates the ODE/IAE structure, used to solve a pair of implicit differential equations. This involves nesting implicit equations (the FIND...MATCH primitive) inside a differential equations model. In the program (Figure 11), the integration model is identified in the

```

PROBLEM IMPDES
COMMON XDOT,X,YDOT,Y,T
X=14000:Y=7000          | Initial Conditions
XDOT=-50:YDOT=-25      | Initial Rate Guesses
T=0:DT=.25:TP=.5
@AXES('PLOT')
INITIATE JANUS; FOR IDEQ; EQUATIONS
* XDOT/X, YDOT/Y; OF T; STEP DT; TO TF;
PRINT *, TIME XDOT X YDOT Y
TF=TP
DO WHILE (TF.LE.50)
INTEGRATE IDEQ; BY JANUS
PRINT '(E15.0)',T,XDOT,X,YDOT,Y
@CURVES('PLOT')
TF=TF+TP
END DO
@SHOW('PLOT')
END

MODEL IDEQ              | Implicit Differential Equations
COMMON XDOT,X,YDOT,Y,T
FIND XDOT,YDOT; IN IRATE(GX,GY); BY AJAX(ACON); TO MATCH GX,GY
END

MODEL IRATE(GX,GY)     | Implicit Rate Equations
COMMON XDOT,X,YDOT,Y,T
GX=XDOT+3.2*SQRT(1-(XDOT+YDOT)*EXP(-T/50)
& *(1.15+57.5/(20000+X+Y)))
& *(1+.1*EXP(-T/10)*SIN(1.5708*T))
GY=YDOT+1.58*SQRT(1-(XDOT+YDOT)*EXP(-T/50)
& *(1.15+38.2/(20000+X+Y)))
& *(1+.1*EXP(-T/10)*SIN(1.5708*T))
END

CONTROLLER ACON(AJAX)
SUMMARY=0
END

```

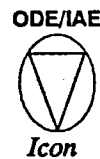


Figure 11. Implicit Differential Equations

INITIATE statement, as IDEQ. This model contains a FIND statement which solves the model IRATE for the implicit rate variables XDOT and YDOT to match the equality constraints GX and GY to zero. Thus Newton's method is used during each integration step to implicitly determine the rates used in the integration algorithm.

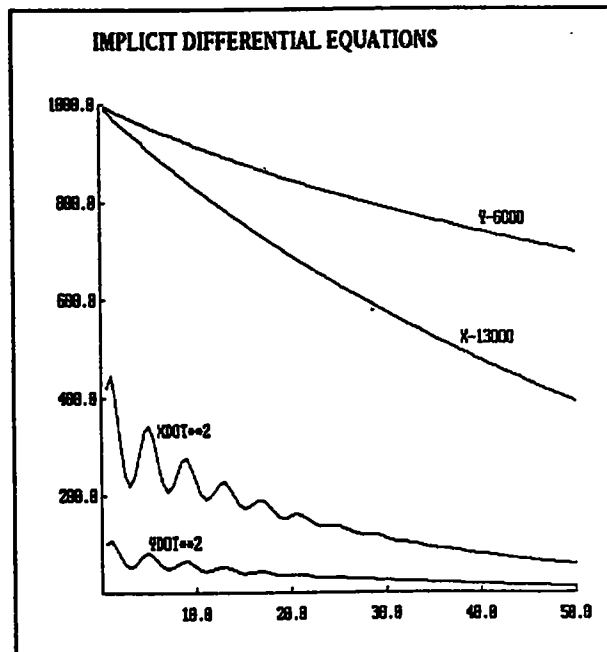


Figure 12. Implicit Differential Equations Output

**Multipoint Boundary Value Parameter Estimation** - The next example (Figure 13) illustrates the UPO/ODE structure used to solve a multipoint boundary-value problem determining the parameters of a set of differential equations. These equations govern the kinetics of a chemical reaction having four chemical concentrations, A,B,C, and D. It is an inverse problem to match experimental data for the concentrations A and C over time, determining the parameters P1 and P2 by least-squares curve fitting. The unconstrained optimizer HERA is used to minimize the least-squares error. This error is accumulated during the reaction when data were observed. Figure 14 shows the concentration curves for the initial and final iterations.

```

GLOBAL ALL
GRAPHICS FILE
PROBLEM CHEMPARE          | Chemical Kinetics Parameter Estimation
DIMENSION TM(6),CM(6),AM(6)
DATA TM/10,20,30,40,50,60/      | Time points for measurements
DATA CM/0.419,0.563,0.629,0.666,0.699,0.708/  | C measurements
DATA AM/0.483,0.281,0.181,0.134,0.097,0.068/  | A measurements
DATA NM,A0,B0,C0,D0,DT/8,1.0,1.03,0.0,0.0,2.0/ | Initial Conditions.
DATA B1,B2/0.015,0.015/        | Optimization step bounds
P1 = 0.01; P2 = 0.06           | Parameter starting guesses
INITIATE ATHENA; FOR REACTION;
& EQUATIONS DADT/A,DBDT/B,DCDT/C,DDDT/D;
& OF T; STEP DT; TO TF
FIND P1,P2; IN CURFIT;
& BY HERA(SET); WITH BOUNDS B1,B2; TO MINIMIZE ERROR
END

MODEL CURFIT
CHARACTER*2 FCSINT,NI
DATA IT/0 : NI = FCSINT(IT)
@AXES(ITER//NI,ITERATION '//NI)
A = A0; B = B0; C = C0; D = D0; T = 0; TF = 0; ERROR = 0; I = 1
DO WHILE (I.LE.NM)
  TF = TF + DT
  INTEGRATE REACTION; BY ATHENA
  @CURVES(ITER//NI)
  IF (TF.EQ.TM(I)) THEN
    ERROR = ERROR + (AM(I)-A)**2 + (CM(I)-C)**2      | Cumulative error
    @MEASURED(ITER//NI,AM(I),CM(I),TM(I))          | Plot measurement
    I = I + 1
  ENDIF
END DO
@SHOW(ITER//NI)      | Show graph for this iteration
IT = IT + 1
END

MODEL REACTION | Rates of reaction differential equations
DCDT = P1*A*B
DADT = -(DCDT + (.01 + P2)*A)
DBDT = -(DCDT + .05*B*D)
DDDT = DBDT-DADT
END

CONTROLLER SET(HERA)
DELTA = 1E-2; DETAIL = 1; ADJUST = 2
END
  
```



Figure 13. Chemical Parameter Estimation Problem

Figure 15 shows the final summary report of the iterations produced by the FIND statement. This is followed by a detailed iteration report. These detailed reports show the Hessian matrix and the gradient vector at the current point. Next are the eigenvalues and eigenvectors of the Hessian, and the computed step along each eigenvector. In the first iteration both steps were bounded, and the second was an anti-Newton step. This detailed history of the iterations is often useful in understanding pathological surfaces.

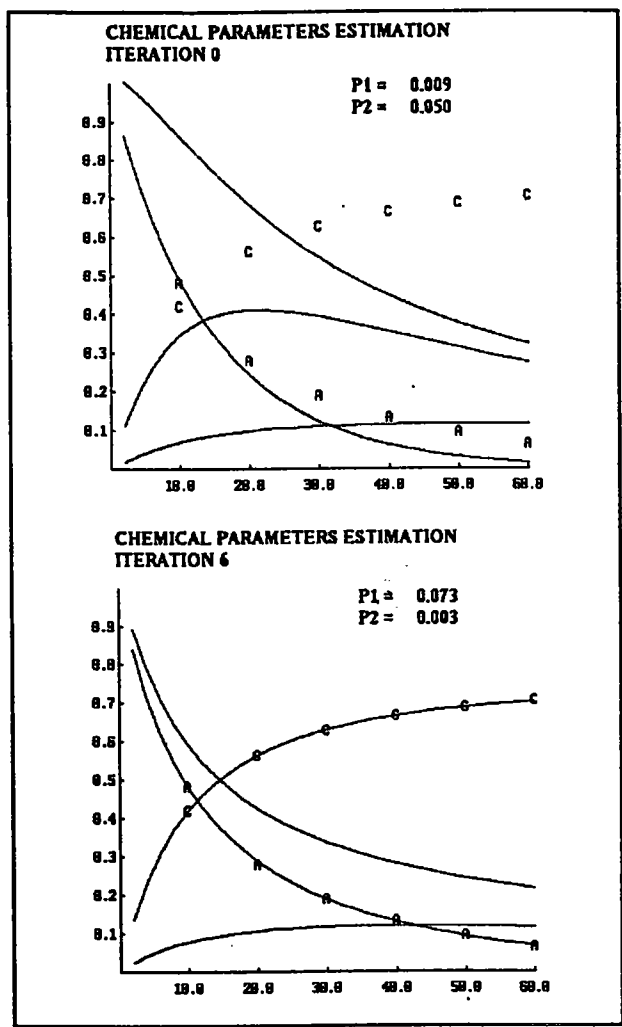


Figure 14. Initial & Final Fit Iterations

```

--- HERA SUMMARY, INVOKED AT CHEMPARE[12] FOR MODEL CURFIT ---

CONVERGENCE CONDITION AFTER 6 ITERATIONS
UNKNOWN S CONVERGED
OBJECTIVE CRITERION UNSATISFIED
ALL SPECIFIED CRITERIA SATISFIED
LOOP NUMBER .....[INITIAL]          5          6
UNKNOWN S
P1          1.000000E - 02  7.227140E - 02  7.315935E - 02
P2          5.000000E - 02  3.450409E - 03  3.565576E - 03
OBJECTIVE
ERROR       1.597781E + 00  1.967029E - 04  1.275779E - 04
---END OF LOOP SUMMARY

==== SCROLL PAGE 1
--- HERA ITERATION 0 INVOKED AT CHEMPARE[12] FOR MODEL CURFIT ---
OBJECTIVE [F] 1.597781E + 00
INDEPENDENT VARIABLES [X] 1.000000E - 02  5.000000E - 02
HESSIAN MATRIX [D2F/DXD] (1) (2)
(1) 2.577044E + 03  4.739828E + 02
(2) 4.739828E + 02  -1.903211E + 01
GRADIENT VECTOR [DF/DX] -5.820254E + 01  8.241599E + 00
EIGENVALUES OF HESSIAN MATRIX 5.986969E - 01  -2.314423E - 02
MATRIX OF EIGENVECTORS (1) (2)
(1) 8.847170E - 01  -1.741823E - 01
(2) 1.741823E - 01  8.847170E - 01
DELTA-X [I = BOUNDED, I = ANTI-NEWTON]
0.173832E - 01 | -0.121583E - 01
1.738319E + 00 | 2.431864E - 01
CONVERGENCE CONDITION AFTER 0 ITERATIONS
OBJECTIVE CRITERION UNSATISFIED
SPECIFIED CRITERIA UNSATISFIED
  
```

Figure 15. Summary & Detailed Reports

**Optimal Design & Control** - The next example (Figures 16 & 17) illustrates nesting of inverse problems (UPE/IAE/ODE structure). Model TWOPT solves a two-point boundary-value problem using the Newton solver AJAX to find the initial condition of the control variable Y to match its boundary condition to zero. Then it computes the objective function for the outer optimization process to find the design parameter A appearing in the objective function and in the ODEs.

```

GLOBAL ALL
PROBLEM OPTDES
  FIND A; IN TWOPT; BY HERA;
  TO MINIMIZE OBJ
END

MODEL TWOPT | Two point boundary value problem
Y0 = .7 | Guess for initial condition
FIND Y0; IN TRAJ; BY AJAX; TO MATCH Y1
OBJ = Z/2 + A**2/2
END

MODEL TRAJ | Initial value problem
Y = Y0; X = 1; Z = 0; T = 0 | Initial conditions
DT = 0.25; TF = 1
INITIATE ISIS; FOR DIFF;
-- EQUATIONS ZDOT/Z, XDOT/X, YDOT/Y;
-- OF T; STEP DT; TO TF
INTEGRATE DIFF; BY ISIS
Y1 = Y | Boundary condition
TERMINATE DIFF
END

MODEL DIFF
ZDOT = X**2 + Y**2 | Objective differential equation
XDOT = -A*X + Y | State differential equation
YDOT = X + A*Y | Control differential equation
END
  
```

UPO/IAE/ODE



Icon

Figure 16. Optimal Design & Control Problem

Upon convergence of the inner FIND statement (in model TWOPT), differential coordinate transformation takes place to transform all the (first order) derivatives with respect to Y0 into (first & second order) derivatives with respect to the design parameter A. This also differentiates Y0 with respect to A, making it into a dependent variable.

Figure 17 illustrates the hierarchy of solvers and model contexts in this program, designating the interface be-

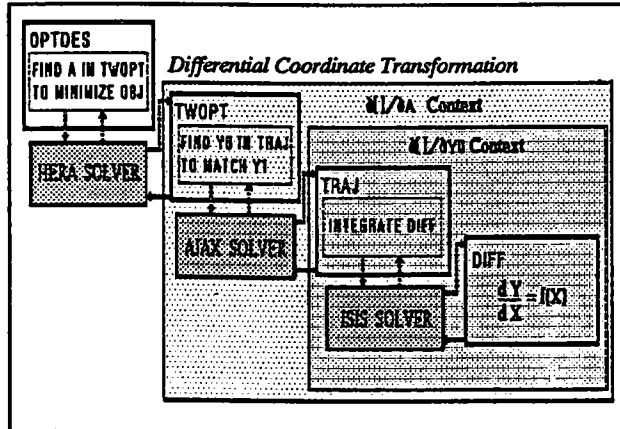


Figure 17. Optimal Design & Control Hierarchy

tween the two nested inverse problems where the coordinate transformation occurs. This transformation is an essential part of the hierarchic programming apparatus, since it enables the differentiation of implicit functions, thereby eliminating the need for restructuring the problem for solution purposes. It is invoked by the inner inverse solver (AJAX in this case) upon detecting that an outer derivative context exists.

Figure 18 shows summary reports from AJAX, each corresponding to an iteration of the optimizer HERA, followed by a summary print from HERA.

```

--AJAX SUMMARY, INVOKED AT TWOPT(8) FOR MODEL TRAJCT --
CONVERGENCE CONDITION AFTER 1 ITERATIONS
UNKNOWN CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED
LOOP NUMBER ---- [INITIAL] 1

UNKNOWN
Y0 0.700000E+00 -0.761580E+00
CONSTRAINTS
Y1 0.225529E+01 -0.183664E-15
--END OF LOOP SUMMARY

--AJAX SUMMARY, INVOKED AT TWOPT(8) FOR MODEL TRAJCT --
CONVERGENCE CONDITION AFTER 1 ITERATIONS
UNKNOWN CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED
LOOP NUMBER ---- [INITIAL] 1

UNKNOWN
Y0 0.700000E+00 -0.842454E+00
CONSTRAINTS
Y1 0.247563E+01 0.602491E-15
--END OF LOOP SUMMARY

--AJAX SUMMARY, INVOKED AT TWOPT(8) FOR MODEL TRAJCT --
CONVERGENCE CONDITION AFTER 1 ITERATIONS
UNKNOWN CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED
LOOP NUMBER ---- [INITIAL] 1

UNKNOWN
Y0 0.700000E+00 -0.840252E+00
CONSTRAINTS
Y1 0.248091E+01 -0.174685E-15
--END OF LOOP SUMMARY

--AJAX SUMMARY, INVOKED AT TWOPT(8) FOR MODEL TRAJCT --
CONVERGENCE CONDITION AFTER 1 ITERATIONS
UNKNOWN CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED
LOOP NUMBER ---- [INITIAL] 1

UNKNOWN
Y0 0.700000E+00 -0.840251E+00
CONSTRAINTS
Y1 0.248091E+01 -0.234730E-15
--END OF LOOP SUMMARY

--HERA SUMMARY, INVOKED AT OPTDES(3) FOR MODEL TWOPT --
CONVERGENCE CONDITION AFTER 3 ITERATIONS
UNKNOWN CONVERGED
OBJECTIVE CRITERION SATISFIED
ALL SPECIFIED CRITERIA SATISFIED
LOOP NUMBER ---- [INITIAL] 1 2

UNKNOWN
ALPHA 0.000000E+00 0.228183E+00 0.232900E+00
OBJECTIVE
OBJ 0.380810E+00 0.347279E+00 0.347265E+00
LOOP NUMBER ---- [INITIAL] 3

UNKNOWN
ALPHA 0.000000E+00 0.232902E+00
OBJECTIVE
OBJ 0.380810E+00 0.347265E+00
--END OF LOOP SUMMARY

ELAPSED TIME = 5.27 SECONDS
  
```

Figure 18. Optimal Design & Control Output

**Wing Design Optimization** - The final example (Figure 18) illustrates the CPO/IAE/ODE structure, using constrained optimization to solve for 12 design parameters subject to an equality constraint and an inequality constraint. This program determines an optimal wing design in the sense that under prescribed total weight constraints, and upper and lower bounds on flexural rigidity (EI) the values of EI for NEI wing sections are determined, along with the length ALPHA of each section, so as to maximize a normalized lift to weight ratio. The wing is modeled as a non-uniform cantilever beam. Its cross-section may be non-uniform in material properties (characterized by elastic modulus E) and/or in geometry (characterized by moment of inertia I). Because the aerodynamics are coupled with the structure, it is necessary to know the wing deflection to determine the lift, but the deflection itself depends upon the lift. This is reflected in the differential equation boundary-value problem used to model the wing structure.


The nested boundary-value problem is solved by the FIND statement in model FLEX, the equation integrated is a second-order differential equation (statement number 20 in BEAMODE). The INITIATE statement in model BEAMIVP handles this as an equivalent system of first order ODEs because of the repeated occurrence of the the state variable DY0DX as the rate variable of a second (hidden) ODE.

The solution of definite integrals via Simpsons rule occurs in the WINGLIFT formula in model FLEX, and in the differential equation formula in model BEAMODE. The integrand in both cases is the function model FORCE, containing an IF statement to switch between the lift integral and the moment integral formulas. IF statements also appear in the Lagrange interpolation function model TERP, called by FORCE. The appearance of IF statements in procedures undergoing differentiation (see also Figure 13) is a novel capability of automatic differentiation. Since differentiation is dynamic rather than static, there is no concern that the formula differentiated is changed from instance to instance.

This example also illustrates several language features that aid in simplifying programs. The GLOBAL ALL declaration avoids the chore of declaring the scope of the variables in each procedure. It causes all program variables to be accumulated in a common block /GLOBALS/. The DYNAMIC declaration defines variables to be used as free storage arrays, which may be ALLOTEd and PURGEd at run time. Dynamic arrays are objects containing rank and dimension information which may be accessed by the array operations of the language, such as the array assignment

```

GLOBAL ALL          I Wing Design Optimization
PROBLEM WING (40000,5000,2000)
DYNAMIC XS,Y,ALPHA,ALPHAL,ALPHAU,DELALP,EIS,EIL,EIU,DELEI
@NODIAG(1006): CALL SETUP
FIND EIS,ALPHA; I Flexural rigidity vector, length fraction vector
~ IN FLEX; BY THOR(TCON);
~ WITH BOUNDS DELEI,DELALP;
~ WITH LOWERS EIL,ALPHAL;
~ WITH UPPERS EIU,ALPHAU;
~ HOLDING WEIGHT; MATCHING ALTOTL;
~ TO MAXIMIZE WINGLIFT I LM/Weight
PRINT *; SOLUTION TO ODE-S'
TABULATE XS,Y
END

                                CPO/IAE/ODE
                                
                                Icon

PROCEDURE SETUP
PI=3.141592654; O=0; WL=1; A=6; EPSLN=0.1; DY0=0.01; DYLDX=0;
H=0.05; NEI=8; WTMAX=7; NPTS=(WL-O)/H+1
ALLOT ALPHA(NEI),ALPHAL(NEI),ALPHAU(NEI),DELALP(NEI)
ALLOT EIS(NEI),EIL(NEI),EIU(NEI),DELEI(NEI); ALLOT XS(NPTS),Y(NPTS)
<ALPHA>=(0.2); <ALPHAL>=(0.1); <ALPHAU>=(1.0)
<DELALP>=(0.0333); <DELEI>=(0.333); <EIS>=DATA(6,5,4,3,2)
<EIL>=DATA(3,2,2,5,1,1); <EIU>=DATA(10,7,5,4,4)
DO 10 I=1,NPTS
  XS(I)=O+(I-1)*H; Y(I)=EPSLN*SIN(PI*XS(I)/2)
10 CONTINUE
TABULATE EIL,EIU,ALPHAL
END

MODEL FLEX
FIND DY0; IN BEAMIVP; I Beam equation boundary value problem
~ BY AJAX(KNOB); TO MATCH YPA
EIAVG=0
DO 10 K=1,NEI
10 EIAVG=EIAVG+ALPHA(K)*EIS(K) I Avg. flexural rigidity
LFLG=1; WINGLIFT=#INTEGSI(FORCE,O,WL,1D-4)/EIAVG
LFLG=0; WEIGHT=WTMAX-EIAVG I Weight (inequality) constraint
ALTOTL=ARRAYSUM(ALPHA)-1 I Total length (equality) constraint
ROWPRINT EIAVG,WINGLIFT,WEIGHT,ALTOTL
TABULATE EIS,ALPHA
END

MODEL BEAMIVP          I Beam equation initial value problem
X=O; Y0=0; DY0DX=DY0; XF=H; Y(1)=Y0
INITIATE ATHENA(CNTRL); FOR BEAMODE; I Integrate beam equation
~ EQUATIONS D2Y0DX2/DY0DX,DY0DX/Y0; OF X; STEP H; TO XF
DO 10 I=2,NPTS
  INTEGRATE BEAMODE; BY ATHENA
  Y(I)=Y0; XF=XF+H
10 CONTINUE
TERMINATE BEAMODE
YPA=DY0DX-DYLDX I Boundary constraint - must be zero
END

MODEL BEAMODE I Cantilever beam differential equation
SUM=0
DO 10 J=1,NEI
  IF(X.GE.SUM*WLANDX.LT.(ALPHA(J)+SUM)*WL) THEN
    EI=EIS(J); GOTO 20
  ENDF
  SUM=SUM+ALPHA(J)
10 CONTINUE
20 D2Y0DX2=#INTEGSI(FORCE,O,X,4)/EI*(1+DY0DX**2)**1.5
END

FMODEL FORCE(F) I Quadrature integrand - lift or moment
IF(LFLG.GT.0) THEN I Calculate lift integrand
  FORCE=A*COS(PI*F/(1.95*WL))-EPSLN*TERP(F)
ELSE I Calculate moment integrand
  FORCE=R*(A*COS(PI*F/(1.95*WL))-EPSLN*TERP(F))
ENDIF
END

FMODEL TERP(Z) I Lagrange Interpolation of wing formula
IF(Z.EQ.XS(1)) THEN
  TERP=Y(1)
ELSEIF(Z.EQ.XS(NPTS)) THEN
  TERP=Y(NPTS)
ELSEIF(Z.LT.XS(2)) THEN
  TERP=POLY(XS(1),XS(2),XS(3),Y(1),Y(2),Y(3),Z)
ELSE
  DO 10 I=2,NPTS-1
    IF(Z.GE.XS(I)) GOTO 20
10 CONTINUE
  TERP=POLY(XS(I-1),XS(I),XS(I+1),Y(I-1),Y(I),Y(I+1),Z)
ENDIF
END

```

Figure 18. Wing Design Optimization Problem



—THOR SUMMARY, INVOKED AT WING(8) FOR MODEL FLEX —  
 CONVERGENCE CONDITION AFTER 5 ITERATIONS  
 UNKNOWN CONVERGED  
 OBJECTIVE CRITERION SATISFIED  
 ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER	[INITIAL]	1	2
UNKNOWN			
EIS ( 1)		6.000000E+00	5.334000E+00
EIS ( 2)		5.000000E+00	4.334000E+00
EIS ( 3)		4.000000E+00	3.334000E+00
EIS ( 4)		3.000000E+00	2.334000E+00
EIS ( 5)		2.000000E+00	1.334000E+00
ALPHA ( 1)		2.000000E-01	1.750000E-01
ALPHA ( 2)		2.000000E-01	1.700000E-01
ALPHA ( 3)		2.000000E-01	1.825000E-01
ALPHA ( 4)		2.000000E-01	2.258000E-01
ALPHA ( 5)		2.000000E-01	2.886000E-01
OBJECTIVE			
WINGLIFT		9.278185E-01	1.187818E+00
INEQUALITY CONSTRAINTS			
WEIGHT		3.000000E+00	3.805100E+00
EQUALITY CONSTRAINTS			
ALTOTL		0.000000E+00	0.000000E+00

LOOP NUMBER	[INITIAL]	3	4
UNKNOWN			
EIS ( 1)		8.000000E+00	4.002000E+00
EIS ( 2)		5.000000E+00	3.002000E+00
EIS ( 3)		4.000000E+00	2.500000E+00
EIS ( 4)		3.000000E+00	1.002000E+00
EIS ( 5)		2.000000E+00	1.000000E+00
ALPHA ( 1)		2.000000E-01	1.000000E-01
ALPHA ( 2)		2.000000E-01	1.000000E-01
ALPHA ( 3)		2.000000E-01	1.000000E-01
ALPHA ( 4)		2.000000E-01	3.002000E-01
ALPHA ( 5)		2.000000E-01	3.898000E-01
OBJECTIVE			
WINGLIFT		9.278185E-01	2.239101E+00
INEQUALITY CONSTRAINTS			
WEIGHT		3.000000E+00	5.348000E+00
EQUALITY CONSTRAINTS			
ALTOTL		0.000000E+00	0.000000E+00

LOOP NUMBER	[INITIAL]	5
UNKNOWN		
EIS ( 1)		8.000000E+00
EIS ( 2)		5.000000E+00
EIS ( 3)		4.000000E+00
EIS ( 4)		3.000000E+00
EIS ( 5)		2.000000E+00
ALPHA ( 1)		2.000000E-01
ALPHA ( 2)		2.000000E-01
ALPHA ( 3)		2.000000E-01
ALPHA ( 4)		2.000000E-01
ALPHA ( 5)		2.000000E-01
OBJECTIVE		
WINGLIFT		9.278185E-01
INEQUALITY CONSTRAINTS		
WEIGHT		3.000000E+00
EQUALITY CONSTRAINTS		
ALTOTL		0.000000E+00

—END OF LOOP SUMMARY

SOLUTION TO ODE-S

XS	Y
1. 0.000000E+00	0.000000E+00
2. 5.000000E-02	2.452448E-02
3. 1.000000E-01	4.804388E-02
4. 1.500000E-01	7.353753E-02
5. 2.000000E-01	9.797026E-02
6. 2.500000E-01	1.223121E-01
7. 3.000000E-01	1.465227E-01
8. 3.500000E-01	1.703469E-01
9. 4.000000E-01	1.835454E-01
10. 4.500000E-01	2.158355E-01
11. 5.000000E-01	2.373134E-01
12. 5.500000E-01	2.574548E-01
13. 6.000000E-01	2.781171E-01
14. 6.500000E-01	2.930404E-01
15. 7.000000E-01	3.078527E-01
16. 7.500000E-01	3.205759E-01
17. 8.000000E-01	3.308368E-01
18. 8.500000E-01	3.378819E-01
19. 9.000000E-01	3.420975E-01
20. 9.500000E-01	3.431318E-01

Figure 19. Wing Optimization Final Output

operations appearing in the SETUP procedure, and the TABULATE statement.

The final output of this problem is shown in Figure 19, containing a THOR iteration summary report followed by the output from the TABULATE statement describing the solution of the ODEs.

### Programming Reduced to Modeling

Synthetic calculus renders algorithmic high-level languages into non-algorithmic very-high-level languages by formulating problems according to intrinsic mathematical decompositions:

- **Model/algorithm** - The migration of partial differentiation from model definition to automatic arithmetic permits the separation of models from algorithms, enabling algorithms to become part of the environment rather than the program;
- **Explicit/implicit** - Problems are decomposed according to whether the unknowns are dependent variables (explicit) or independent variables (implicit) and implicit solution operators are provided that enable the separated implicit sub-problems to be stated explicitly;
- **Prediction/Control** - The separation and nesting of prediction processes within control processes is a necessary method of avoiding paradoxical problem statements in mathematical modeling, and decomposes the problem description into a nested hierarchy of separate mathematical problems, each functioning as a complementary part of a dual prediction-control process.

Each level of the control/prediction hierarchy is a complete system-level problem (e.g. simultaneous equations). The exchange of functional dependence for prediction occurs visibly through the exchange of variables between the levels. The exchange of differential dependence for control occurs invisibly through underlying hidden differential propagation mechanisms.

The meaning of non-algorithmic programming is that models used to describe the problem unit functions are expressed as 'open-loop' predictive procedures containing only explicit non-iterative formulas. This obviates the convoluted blending of numerical solution methods with the problem formulation, thereby preserving the engineering coherence of the model. Except in rare cases there is no need for mathematical reduction from the original engineering synthesis of a problem, and the problem can be evolved through run experience. Except for the generic solver operation statements, which interface the hidden algorithms, the model becomes the entire program.

## CONCLUSION

*First Principles to Solutions* - In summary, our goal has been to achieve a level of program synthesis that enables modelers to go from first principles to solutions in a single description. This means elimination of low level algorithmic description, and directly addressing inverse problems, since that is the form in which the scientific method presents problems to us.

The new paradigm has the elegance of a minimum set of primitives which can be combined via nesting to provide multiplicative leverage in solving complex problems, especially when implicitly formulated.

Synthetic calculus provides a hidden calculus basis for standard programming languages, purifying them of the algorithmics that has obscured their use as modeling languages. This enables the *declarative* expression of problems that has been the goal of AI languages like PROLOG, but in a common mathematical genre familiar and comfortable to scientific and engineering work.

*The CASE for Optimization* - Automatic differentiation provides the enabling technology for the use of optimization in the dual role as: (1) the primary solution means for inverse problems and (2) a realizable goal of engineering design. Synthetic calculus is the CASE paradigm that automatically *re-models* existing software for optimization (basic synthesis) and enables its reuse in new-purpose applications (advanced synthesis). This is achieved in a low technology process employing source code largely *as is* with little or no mathematical analysis, algorithm design, or debugging.

In the realm of new applications, synthetic calculus provides the leverage of 90 percent prebuilt software plus the solution power of higher mathematics. Its focus is one of rapid prototyping to provide engineering results as immediately for new technology applications as for staple applications having the benefit of existing software tools, such as historical CAD applications.

The product described above is a combination of standard FORTRAN 77 and our product software, *PowerCalculus*<sup>™</sup>. This product consists of a precompiler and a large runtime library. It can be readily installed in any 32-bit or 64-bit FORTRAN environment. Our current versions are available on Cray, DEC VAX computers, and will soon be available on IBM mainframes. We can provide custom Client-Server installations, and are planning to offer ADA versions in 1992.

## BIBLIOGRAPHY

1. Adamson, D.S. and Winant, C.W.: "A SLANG Simulation of an Initially Strong Shock Wave Downstream of an Infinite Area Change", Proc. Conf. on Applic. of Continuous-System Simulation Languages (June 1969) pp. 231-240.
2. Baur, W. and Strassen, V.: "The Complexity of Partial Derivatives", Theoretical Computer Science 22 (1983), North Holland Publishing Co.
3. Bellman, R.E., and Kalaba, R.E. Quasilinearization and Non-linear Boundary Value Problems, American Elsevier Publishing Co. New York, 1965
4. Griewank, A.: "On Automatic Differentiation", Preprint MCS-P10-1088, Argonne National Laboratory, October 1988.
5. Iri, M.: "Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors - Complexity and Practicality", Japan Journal of Applied Mathematics 1 (1984), 223-252
6. Iri, M. and Kubota, K.: "Methods of Fast Automatic Differentiation and Applications", RMI 87-02, Dept. of Mathematical Engineering, University of Tokyo, Nov. 1986.
7. Jerrell, M.: "Automatic Differentiation Using Almost Any Language", SIGNUM Newsletter, January 1989.
8. Kagiwada, H., Kalaba, R., Rosakho, N., and Springarn, K.: Numerical Derivatives and Nonlinear Analysis, Plenum Press, New York, 1986
9. Kedem, G.: "Automatic Differentiation of Computer Programs", ACM TOMS, June 1980
10. Krinsky, B. and Thames, J.M.: "The Structure of Synthetic Calculus", Proceedings of the Int'l Workshop on High-Level Computer Architecture, Los Angeles, 1984.
11. Neidinger, R.D.: "Automatic Differentiation and APL", College Mathematics Journal, Vol. 20, No. 3, May 1989.
12. Pfeiffer, F.: "Automatic Differentiation in PROSE", SIGNUM Newsletter Nov. 22, 1987
13. Rall, L.B.: "The Arithmetic of Differentiation", Mathematics Magazine, Vol. 59, No. 5, December 1986
14. Rall, L.B.: "Differentiation in PASCAL-SC: Type Gradient", ACM TOMS, June 1984
15. Rall, L.B.: "Differentiation and the Generation of Taylor Coefficients in PASCAL-SC". In a New Approach to Scientific Computation, U.W. Kulisch and W.L. Miranker, Eds., Academic Press, New York, 1983
16. Rall, L.B.: Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science, Vol. 120,k, Springer-Verlag, New York, 1981
17. Thames, J.M.: "FORTRAN CALCULUS: A New Implementation of Synthetic Calculus", Digital Calculus Corporation, December, 1989.
18. Thames, J.M.: "The Evolution of Synthetic Calculus: A Mathematical Technology for Advanced Architecture", Proceedings of the Int'l Workshop on High-Level Language computer Architecture Fort Lauderdale, Florida, 1982
20. Thames, J.M.: "Computing in Calculus", Research/Development 26,5 (May 1975)
21. Thames, J.M.: "SLANG, A Problem-Solving Language for Continuous-Model Simulation and Optimization", Proceedings, ACM 24th National Conf. (Dec. 1969)
22. PROSE - A General Purpose Higher Level Language, Calculus Applications Guide, Control Data Corp. Cybernet Services, Pub. No. 84000170 Rev. A (Jan. 1977)
23. PROSE - A General Purpose Higher Level Language, Calculus Reference Manual, Control Data Corp. Cybernet Services, Pub. No. 84003200 Rev. B (Jan 1977)