Proceedings of the APL 2001 Conference

A Dynamic APL GUI Equation Solver (ADAGE)

Stephen M. Mansour
The Carlisle Group, Inc.
544 Jefferson Ave., Scranton, PA 18510
steve@carlislegroup.com

Abstract

This paper describes ADAGE, (A Dynamic APL GUI Equation solver), which allows the user to solve a single multivariate equation when values are supplied for all but one of the variables. The application presents a graphical user interface and uses many of the newer features of Dyalog APL. These include dynamic functions and operators, recursion with tail calls, object syntax, and namespace reference array expansion. All of these features, combined with some key mathematical concepts, including function composition, implicit functions and the Secant algorithm, work together to provide a simple solution to a complex problem. Several examples from the mortgage industry will be explored.

Function Composition

We can derive a monadic function by currying one of the arguments to a dyadic function. Dyalog APL accomplishes this via the composition operator with an array operand:

```
    (2∘*)  0 1 2 3 ⍝ Powers of 2
1 2 4 8
    (*∘2) 0 1 2 3 ⍝Squaring Function
0 1 4 9
```

In effect there are two completely different monadic functions created from the power function (`*`). This concept extends to a function of n arguments, represented by a monadic function taking an n-vector as its right argument and producing a scalar result. We can derive n different functions of a single variable in a similar manner.

As an example, consider the function *pmt*, which calculates a monthly mortgage payment:

```
[0]   m←pmt X;r;t;b;D
[1]   (r t b)←X ⍝ Rate, term, balance
[2]   D←(1+r÷1200)*t
[3]   m←b×(r÷1200)×D÷D-1
    pmt 9.5 360 100000
840.8542072
```

There are three related single-variable functions, which produce the same result but take different inputs:

```
      ⍝ Payment as a function of...
      {pmt ⍵ 360 100000}9.5   ⍝ ...Rate
840.8542072
      {pmt 9.5 ⍵ 100000}360   ⍝ ...Term
840.8542072
      {pmt 9.5 360 ⍵}100000   ⍝ ...Balance
840.8542072
```

Homogenous Equations and Implicit Functions

Any function $y = f(x)$ can be converted to implicit form by the homogeneous equation $g(x,y) = y - f(x) = 0$.

Similarly, any multivariate function $y = f(x_0,\cdots,x_{n-1})$ can be converted to implicit form as follows:

$$g(x_1,\cdots,x_{n-1},y) = y - f(x_1,\cdots,x_{n-1}) = 0.$$

Then by setting $y = x_n$ we can define

$$g_k(a) = g(x_0,\cdots,x_{k-1},a,x_{k+1},\cdots,x_n).$$

Now we can solve for $x_k = g_k^{-1}(0)$ using numerical methods.

For example, *pmt* can be converted to implicit form in the following manner. Represent the homogeneous equation by the function $g(m,r,t,b) = m - pmt(r,t,b)$, whose explicit result is the error or deviation from zero. Then derive a function, e.g. $g_1(a) = g(841, a, 360, 100000)$ by fixing all parameters except the one of interest, using the *sub* operator defined below. Four separate functions can be created — one for each variable.

```
   g←{⍵[0]-pmt 1↓⍵}                ⍝ Function in implicit form
   g 841 9.5 360 100000            ⍝ Result is deviation from 0
0.1457928212
   sub←{X←⍺ ⋄ X[⍵⍵]←⍵ ⋄ ⍺⍺ X}      ⍝ Sub operator fixes all but one variable
   X←0 9.5 360 100000              ⍝ Fix rate, term, balance
   X∘(g sub 0) 841                 ⍝ Input payment= $841
0.1457928212
   X←841 0 360 100000              ⍝ Fix payment, term, balance
   X∘(g sub 1) 9.5                 ⍝ Input rate= 9.5%
0.1457928212
   X←840.81 9.5 0 100000           ⍝ Fix payment, rate, balance
   X∘(g sub 2) 360                 ⍝ Input term = 360 months
0.1457928212
   X←841 9.5 360 0                 ⍝ Fix payment, rate, term
   X∘(g sub 3) 100000              ⍝ Input balance = $100,000
0.1457928212
```

The Secant Algorithm

One could use Newton's method to solve $x_k = g_k^{-1}(0)$. However, this involves evaluating $g_k'(a)$ for each k. A slightly less efficient but easier solution is to apply the Secant Method [1]. This method solves for x_k by choosing two initial approximations, a_0 and a_1, then iteratively solves the following equation:

$$a_n = a_{n-1} - \frac{g_k(a_{n-1})(a_{n-1} - a_{n-2})}{g_k(a_{n-1}) - g_k(a_{n-2})} \qquad (0.1)$$

When $|a_n - a_{n-1}| < \varepsilon$ stop and choose $x_k = a_n$ as the solution.

The dynamic operator *SecAlg* can be used for this purpose. Notice that the operator is recursive and uses a tail call:

```
[0]   SecAlg←{
[1]   (P0 Q0 P1 Q1)←⍵      ⍝ Unpack right arg
[2]   TOL←⎕CT×(|P0)⌈|P1   ⍝ Set tolerance
[3]   P←P1-Q1×(P1-P0)÷Q1-Q0   ⍝ Apply
algorithm
[4]   P0←P1 ⋄ Q0←Q1 ⋄ Q1←⍺⍺ P1←P ⍝ Reassign
variables
[5]   (TOL∧.>|P-P0)∨⍺≤1:P0 Q0 P1 Q1 ⍝ Check
tolerance
[6]   (⍺-1)∇ P0 Q0 P1 Q1          ⍝ Continue
[7]   }
```

```
   g1←X∘(g sub 1)  ⍝ Create rate function
   (P0 P1)←9 10     ⍝ Two approximations
   (Q0 Q1)←g1¨P0 P1 ⍝ Apply function to
approximations
   50 g1 SecAlg P0 Q0 P1 Q1 ⍝ Secant Algorithm
9.501998177 ¯3.069544618E¯12 9.501998177
9.094947018E¯13
```

The result is a four-item vector containing the final two iterations and the results of applying the implicit function to those two values. The first item of the result as the solution; the interest rate satisfying the homogeneous equation is approximately 9.502%

Creating a GUI

A GUI front end for the mortgage calculations requires a form, which contains a "calculate" button and an edit box for each variable. The user can then enter a value in each edit box. When he presses a button, the system will recalculate the correct value of the corresponding variable to satisfy the equation.

The function *SOLVE* (Figure 1) takes the name of an implicit function as input. (See Figure 2: the implicit function *mtg*). A corresponding *data dictionary* provides a description for the input; the data format and default values. (See Figure 3: the variable *DD_mtg*). It is a matrix containing a row for each input variable to the implict function. *SOLVE* creates an array of namespaces corresponding to the edit boxes; each item corresponds to a row in the data dictionary. Namespace reference array expansion and object syntax permits direct assignment of properties to each edit box:

```
ED.FormatString←DD[;2] ⍝ ED is a namespace array
ED.Value←DD[;3]        ⍝ DD is the data dictionary
```

The default column serves a dual purpose, first to supply a default value to appear in the edit box to save typing. Secondly, it is used as the first initial value a_0 when the corresponding calc button is pressed. Before pressing the button, the user should change it to another value, which will serve as the second initial value a_1. If the user doesn't change it, the system will automatically create a unique second initial value.

To run ADAGE, the user simply enters: *SOLVE* '*mtg*' and the Equation Solver dialog box appears (Figure 4).

When the user presses a button, the callback function *RECALC* (Figure 6) receives the name of the button object, then uses the Secant Method to calculate the appropriate value and changes the value property of the appropriate edit box.

A Complex Problem: Pricing Mortgage-Backed Securities

Mortgage-backed securities are callable bonds that pay both principal and interest as opposed to traditional bonds, which pay only interest until maturity. There are many factors involved in pricing these securities. These include yield, prepayment rate, coupon rate, servicing fees, maturity, and the delay (days to first payment). If we assume that the prepayment rate is constant, we can use the following formula (from [2]):

$$g(v,y,p,c,s,m,d)=v-100\left(\frac{1+y}{1+yd}\right)\left[\frac{(1+c)^m(p+c-s)z_1+(s-p(1+c))z_2}{(1+c)^m-1}\right]=0 \quad (0.2)$$

$$z_1=\frac{1-\left(\frac{1-p}{1+y}\right)^m}{y+p}$$

$$z_2=\begin{cases}\dfrac{1-\left[\dfrac{(1-p)(1-c)}{1+y}\right]^m}{y+p+pc-c} & (1-p)(1-c)\neq(1+y)\\[2ex] \dfrac{m}{1+y} & (1-p)(1-c)=(1+y)\end{cases}$$

Although the formula is fairly complex, it is very straightforward to code in APL. (See Figure 7: The implicit function *price*) One can provide any six inputs and solve for the remaining one using our application, the function price and the data dictionary, *DD_price*:

```
DD_price
 YLD  Yield:             F7.3    10
 CPR  Prepayment Rate:   F7.3    30
 CPN  Coupon:            F7.3    10
 SVC  Service Rate:      F7.3     0.5
 TRM  Term:              I3     360
 DEL  Delay:             I3      30
 PRI  Price:             F7.3   100
```

```
   SOLVE 'price'
```

The corresponding dialog box appears in Figure 5.

While price and yield are the most commonly requested information, we could also calculate the required prepayment rate necessary to satisfy a particular price and yield. Or we could determine the interest rate that would satisfy a particular price, yield and prepayment speed. Notice that Newton's Method would require calculating partial derivatives of equation (0.2) with respect to each of the variables. However, using the secant method, we can avoid all of that.

Notice that the price function is in effect a scalar function; it consists of only scalar primitive functions; there are no operators or non-scalar primitives in the code. All components of the right argument must be conformable; that is, they must all have the same shape or be scalars. Notice also that there are no *:if* control structures to handle the exception case when $(1-p)(1-c)=1+y$ because an *:if* statement cannot handle an array unless it is inside a loop. Lines 17 and 18 of the *price* function handle the exception using scalar functions. In fact, the function behaves like a scalar utility (See [3]). Although the application is set up to calculate a single value, it can easily be adapted to solve for arrays. The domain of the function price includes not only simple 7-element vectors, but also nested 7-element vectors whose elements may also be arrays, e.g.

```
    price (7.5 7.75) 10 (6.98 6.8) .5
360 30 (101 102)
6.186713626 9.270884727
```

Conclusion

Building a user interface to solve equations is not difficult in APL. In the workspace ADAGE there are only two defined functions and one defined operator in the basic application:

SOLVE – Cover function; Creates the GUI interface

RECALC – Callback function.

SecAlg – Operator used to solve equation using secant method.

An additional operator *sub* is localized in the *RECALC* function as a dynamic operator.

For a user application, one needs only supply an implicit function and a corresponding data dictionary. For the two examples in this paper:

mtg, price – Implicit functions

DD_mtg, DD_price – Data Dictionaries

ADAGE is easily adaptable to inputting variable names instead of values and solving for many different values simultaneously. A production version of the pricing model is currently being implemented for use in the mortgage industry.

References

[1] Burden and Faires, *Numerical Analysis*, Fourth Edition, PWS-Kent, 1989

[2] Fabozzi, *The Handbook of Mortgage Backed Securities*, Revised Edition, Probus Publishing, 1988

[3] Mansour, S. How to Write an APL Utility Function, *APL97 Conference Proceedings*, Quote Quad Vol 28 No. 4

```
 ∇SOLVE
[0]    SOLVE FN;DD;F;SZ;FT;BC;EC;R;EV;ED;C
[1]   ⍝∇RUN the application
[2]   ⍝⍵ Function Name: e.g. price (assumes data dictionary DD_price)
[3]   ⍝Create basic form
[4]    'FORM'⎕WC'Form' 'Inverse Calculator'('Coord' 'Pixel')
[5]    'FORM.PARMS'⎕WC'Group'('Function: ',FN)(8 16)(312 416)
[6]    'FORM.PARMS.Label1'⎕WC'Label' 'Press button to calculate:'(24 8)
[7]    'FORM.PARMS.Label2'⎕WC'Label' 'Enter Values:'(24 162)
[8]    C←1⊃⍴DD←⍎'DD_',FN                     ⍝ Data dictionary, # of cols
[9]    F←'FORM.PARMS'                         ⍝ Parent
[10]   SZ←24 128 ⋄ BC←32 ⋄ EC←162             ⍝ Size, Button col, Edit col
[11]   R←56+32×⍳↑⍴DD                          ⍝ Rows
[12]   EV←'Event'('Select' 'RECALC'FN)        ⍝ Event
[13]   FT←'FieldType' 'Numeric'               ⍝ ED.FieldType←⊂'Numeric'
[14]   R{(F,'.',(0⊃⍵),'_B')⎕WC'Button'(1⊃⍵)(⍺ BC)SZ EV}¨⊂[1]DD
[15]   R{(F,'.',⍵,'_E')⎕WC'Edit' ''(⍺ EC)SZ FT}¨DD[;0]
[16]   ED←{⍎F,'.',⍵,'_E'}¨DD[;0]              ⍝ Edit Objects
[17]   :If 2<C ⋄ ED.FormatString←DD[;2] ⋄ :EndIf
[18]   :If 3<C ⋄ ED.Value←DD[;3] ⋄ :EndIf ⍝ Update properties from DD
[19]   ⎕DQ'.'                                 ⍝ Put up form
```

Figure 1: Main function `SOLVE`

```
[0]    Z←mtg Y;N;R;B0;BN;YN;F;B;C;M;OK;I;L;P;A;RB
[1]   ⍝∊ Financial
[2]   ⍝∇ Mortgage function representing homogeneous equation
[3]   ⍝⍺ DATE:     Pay-thru date
[4]   ⍝⍺ DAYCOUNT: 'ACT/ACT' 'ACT/360' '30/360'(default)
[5]   ⍝⍵ RATE:  Should be in percentage form
[6]   ⍝⍵ TERM:  In months
[7]   ⍝⍵ PV:    Present Value (Default=$1.00)
[8]   ⍝⍵ FV:    Future Value (Default=0)
[9]   ⍝⍵ TYPE:  1=Arrears, 0=Advance (default)
[10]  ⍝⍎ mtg RATE TERM PV PMT
[11]  ⍝⍎ Examples:     mtg 360 9.5 100000 670
[12]   (R N B0 M)←Y              ⍝ Unpack right argument
[13]   YN←N*⍨Y←1+R÷1200          ⍝ Calculate monthly, lifetime amort factor
[14]   Z←P←(B0×YN)×(Y-1)÷1-YN    ⍝ Calculate monthly payment
[15]   Z←M+Z÷(-N)*R=0            ⍝ Calculate when rate=0
```

Figure 2: Implicit function `mtg`

Variable	Description	Format	Default
R	Rate	F7.3	10
T	Term	I3	360
B0	Balance	P<$>CF16.2	100000
M	Payment	P<$>CF16.2	900

Figure 3: Sample data dictionary `DD_mtg`

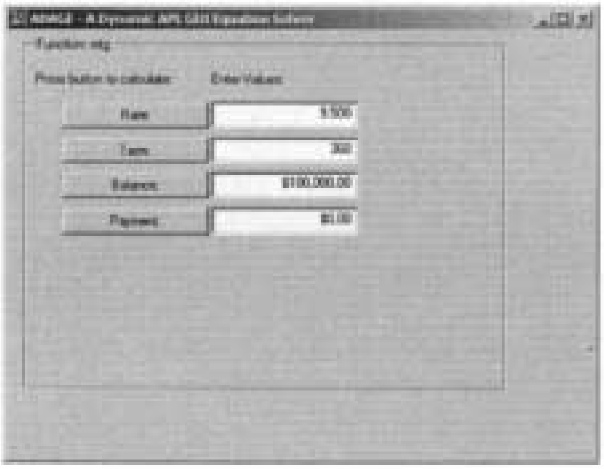

Figure 4: ADAGE dialog box

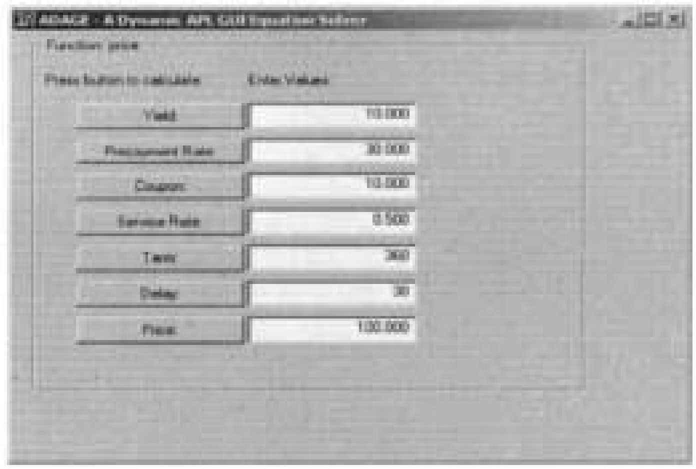

Figure 5: ADAGE dialog box for `price`

```
     ∇RECALC
   FN RECALC X;BUT;G;GK;NM;GRP;EDITS;K;P;sub
  ⍝∊ Written by Steve Mansour
  ⍝∇ Recalculate parameters
  ⍝⍵ Button, Event
   BUT←↑X ⋄ G←⍎FN                           ⍝ Get full name of button
   NM←¯2↓(1+⌈/('.'=BUT)/⍳⍴BUT)↓BUT          ⍝ Get parameter name
   GRP←(⍎BUT).##                            ⍝ Find parent
   EDITS←⍎¨'Edit'⎕WN⍕GRP                    ⍝ Find all edit fields
   X←EDITS.Value                            ⍝ Get values from screen
   K←DD[;0]⍳⊂NM                             ⍝ Find position of button pressed
   sub←{X←⍺ ⋄ X[⍵⍵]←⍵ ⋄ ⍺⍺ X}                ⍝ Sub operator
   P←P+0.=/P←DD[K;3],K⊃X                    ⍝ Set Initial values
   GK←X∘(G sub K)                           ⍝ Implicit function
   X[K]←2⊃50(GK SecAlg),P,[0.5]GK¨P         ⍝ Apply Secant Algorithm
   EDITS.Value←X                            ⍝ Assign values back to screen
 ∇
```

Figure 6: Callback function `RECALC`

```
   Z←price X;V;y;p;c;s;m;d;z1;z2;q;Y;P;C;S;D;⎕DIV
  ⍝∊ Steve Mansour
  ⍝∇ Price at Loan Level
  ⍝∇ Model based on Fabozzi, "The Handbook of mortgage Backed securities"
  ⍝∇ Revised Edition, 1988 -- cash Flow yield pp. 302-303
  ⍝⍵ Yield, Prepayment Rate, Coupon, Servicing Fees, Term, Delay, Price
  ⍝← Error
  ⍝⍎       price  7.5 10 8 .25 360 30 100
  ⍝⍎ ¯1.284836795
   (Y P C S m D V)←X ⋄ ⎕DIV←1          ⍝ Unpack right argument
   c←C÷1200                            ⍝ Convert coupon rate to monthly
   y←Y÷1200                            ⍝ Convert yield to monthly
   s←S÷1200                            ⍝ Convert servicing Rate to monthly
   d←D÷30                              ⍝ Convert delay to days over 30
   p←1-(1-P÷100)*÷12                   ⍝ Convert prepayment speed to monthly
   z1←(1-((1-p)÷1+y)*m)÷y+p            ⍝
   q←y+p+(p×c)-c                       ⍝ Denominator
   z2←(1-((1-p)×(1+c)÷1+y)*m)÷q        ⍝ (1-p)(1+c)≠(1+y)
   z2←z2+(q=0)×m÷1+y                   ⍝ (1-p)(1+c)=(1+y)
   Z←z1×(p+c-s)×(1+c)*m                ⍝
   Z←Z+z2×s-p×1+c                      ⍝
   Z←Z+((1+c)*m)-1                     ⍝
   Z←V-Z×100×(1+y)÷1+d×y                ⍝
```

Figure 7: Implicit function: `price`