PROGRAMMING THE PALM TUNGSTEN
SECOND IN A SERIES

Edwin G. Wiggins
Webb Institute

Introduction

The first article in this series[1] describes the creation of a simple applet for the Palm Tungsten in the programming language PocketC. That applet was a Palm Database (pdb) file. It would run only within the PocketC environment.

When an applet is intended for wide distribution, it is better to create a stand-alone version of the program. Such a version can be executed by users who do not have PocketC.

This article describes a new applet, one that finds roots of a polynomial by means of a successive approximation algorithm. It also demonstrates the process for creating stand-alone applets, called prc files.

Mathematical Background

The algorithm used in this applet is commonly referred to as the Newton-Raphson method. A good discussion of this method can be found at

http://mathworld.wolfram.com/NewtonsMethod.html.

The essential idea is to make an initial guess at the root, draw a tangent to the function at this point, and extrapolate until the tangent crosses the x axis. This intersection is taken as a new guess at the root, and the process is repeated. In symbolic terms, the process is written as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Since, by definition, a root is a point where the value of the function is zero, success in finding the root is measured by the value of the function. It should be noted that there are two potential problems here. The first is the possibility that the algorithm may arrive at a point where the derivative is zero. It is important to intercept this possibility before the applet attempts to divide by this derivative.

The second problem is more subtle. Some functions are very flat near a root. Thus the value of the function may be very close to zero at a significant distance away from the root. In this case, the algorithm may report a root that is not accurate. For example, the polynomial $x^4+4x^3+6x^2+4x+1$, which is $(x+1)^4$, has a quadruple root at x=-1. The function is very flat near this root. At x=-1.05, the value of the polynomial is 6.25×10^{-6}.

Some polynomials of second order and higher have no real roots. An example is x^2+1. This algorithm finds only real roots, so it will find no roots for such functions. When a polynomial has more than one real root, the initial guess will determine which real root is found. Although the algorithm is likely to find the real root nearest the initial guess, this is not always the case.

The Applet in PocketC

As in the previous article, the program code is presented in segments below. Each segment is discussed in turn. The code itself is heavily commented to make it almost self-explanatory.

Following several lines of comments (anything preceded with a double slash), the code begins with the commands required to create a stand-alone applet.

```
// Polyroots
// Finds roots of a polynomial up to 4th order
// 2/11/05

@cid "pol!";                // registration code
@ver "1.0";                 // version number
@category "Engineering";    // Palm applet category where icon should appear
@name "PolyRoot";           // program name to appear under icon
@dbname "Polyroots";        // name of pdb file
@licon1 "polybig.bmp";      // large monochrome icon
@licon8 "polybig.bmp";      // large color icon
@sicon1 "polysmall.bmp";    // small monochrome icon
```

These lines need careful explanation. Line 1 (@cid "pol!";) contains the four character registration code for the program. This code may be obtained at no cost from the PalmOne developer site. Line 2 identifies the version of the program as assigned by the program's creator.

Palm PDAs come with several predefined categories (System, Main, Utilities …), and the user is free to create other categories that serve useful purposes. Assigning applets to particular categories can simplify the process of locating them. The author has created a category called Engineering. The command (@category "Engineering";) in line 3 specifies that this applet should be assigned to that category. When the main screen is displayed, the user may select a category from the drop-down list at the upper right. The default choice (All) displays all icons. If the Engineering category is selected, only programs assigned to that category will have their icons displayed.

Lines 4 and 5 contain the name to be displayed under the icon and the name of the pdb file from which the prc file was created. The next three lines contain references to the icons for the program. Two icons are mandatory: licon1 and sicon1. These are monochrome icons (indicated by the 1). Large icons must be bitmap files that are 32 pixels square. Small icons must be bitmap files that are 15 pixels wide by 9 pixels high. The command @licon8 "polybig.bmp"; points to a color icon. This command is optional, but color icons are much more attractive than monochrome ones. When both a monochrome and a color icon are defined, my Palm Tungsten automatically chooses to display the color icon.

Icons can be created with Microsoft Paint, but it is awkward to work with very small images there. Even when a 32x32 icon is zoomed up to the maximum size (800%) it is too small for easy work. The author uses IconCooleditor to produce the large color icons. This program insists on square icons, so it works fine for 32x32, but it does not work for the small (15x9) icons. Those are done with Microsoft Paint. Since only the large color icon appears on the Palm Tungsten screen, the quality of the small icon is not critical.

Next, all variables that will appear in the code are declared.

```
main() {

// Declare variables

    // Coefficients of the polynomial, beginning with
    the constant term
    float a;  // constant term
    float b;  // coefficient of x
    float c;  // coefficient of x^2
    float d;  // coefficient of x^3
    float e;  // coefficient of x^4
    int order;  // the order of the polynomial

    float function;     // The value of the  polynomial
    float derivfunction;     // The value of its
    derivative

    float x;          // The current estimate of the root
```

```
float x0; // Initial guess of root
float dx; // Change in x

int sentry;      // "sentry" is used to prevent
 infinite loops
```

Then the input values are obtained from the user. First, the applet asks for the order of the polynomial and checks to make sure it is four or less.

```
// Get input
clear();

title("Polyroots");
order= (int) gets ( "  Enter polynomial
 order [1 to 4]");  // order of polynomial

// error trap in case user enters an order
  higher than 4

if (order >4) {
puts ("\n\n" + "Order MUST be <= 4");
// polynomial orders greater than 4 are
not permitted
wait();
exit();
} // end error trap
```

The "if" statements in the code below make sure that the user is only asked for coefficients that are appropriate to the order of the polynomial. If a lower order term is not present, the user should enter zero. However, the program assumes zero values for all variables unless other values are explicitly entered.

```
// ask for coefficients of polynomial

a= (float) gets("Constant term? ");
b= (float) gets("Coefficient of x? ");

if (order > 1) {
c= (float) gets("Coefficient of x^2? ");
} // end if

if (order > 2) {
d= (float) gets("Coefficient of x^3? ");
} // end if

if (order > 3) {
e= (float) gets("Coefficient of x^4? ");
} // end if
```

Finally, the user is asked for an initial estimate of the root. If there are multiple roots, this choice will determine which one is found. The value of the initial guess is preserved as x0, and that value is included in the output.

```
x= (float) gets("Estimate of root? ");
// Get the initial estimate of root
x0=x;  // remember the initial estimate
```

The sentry variable is set equal to 1 in the code below, and the function value is set to 10.

A "while" structure is used to test for convergence and to limit the number of iterations. Because PocketC appears to lack an absolute value function, the convergence test is done by squaring the function value and taking the square root. The convergence tolerance (1e-5 below) must be chosen carefully. If it is too small, the internal accuracy of the machine itself may be insufficient to give convergence. As noted above the polynomial $x^4+4x^3+6x^2+4x+1$ is expected to present a problem in this regard. With this convergence tolerance, the applet reports a root at -0.9440 with a function value of $9.835x10^{-6}$. Unfortunately, smaller tolerances result in failure to converge.

The test for remaining inside the "while" structure has two parts. Logical AND is represented by &&. Thus the loop will continue until either the function value falls below $1x10^{-5}$ or the number of iterations reaches 100.

```
// Perform calculations

sentry=1; // set initial value of "sentry"
function=10; // set initial value of
"function"

while (sqrt(function*function) > 1e-5
 && sentry <= 100) {  // iterate until root
 is found or 100 iterations have been
 performed
```

Both the function and its derivative are evaluated at the current value of x.

```
function=a+b*(x)+c*(x*x)+d*(x*x*x)+e
      *(x*x*x*x);  // evaluate the function
      //note that this is  a+b*x+c*x^2+d*x^3+e*x^4

      derivfunction=b+2*c*(x)+3*d*(x*x)+4*
      e*(x*x*x);   // evaluate its derivative
       //note that this is  b+2*c*x+3*d*x^2+4*e*x^3
```

Here is an error trap in case the value of the derivative is close to zero. In that case the derivative is arbitrarily set equal to 0.5 to avoid a divide by zero error.

```
if ( pow(derivfunction,2) < 1e-6) { //
  error trap in case derivative is near 0

derivfunction=0.5;  // if derivative is
   zero, set it to this value

} // end divide by zero if
```

A new value of x is calculated in accordance with the Newton-Raphson procedure, and the sentry variable is incremented by one.

```
dx = function/derivfunction;

x = x - dx;  // new estimate of the root
sentry=sentry+1;  // increment the sentry
variable


} // end function while
```

The final section of the code displays the output. First, the polynomial itself is displayed so that the user can verify that the correct polynomial is being analyzed. The "if" statements ensure that only appropriate terms are displayed.

```
// Display output

    // Display polynomial

    puts("The polynomial is" + "\n");

if (order==4) { // fourth order polynomial
puts(format(a,0) + " + " + format(b,0) + "x + " +
format(c,0) + "x^2 + " + format(d,0) + "x^3 + " +
format(e,0) + "x^4");
}  // end if
if (order==3) { // third order polynomial
puts(format(a,0) + " + " + format(b,0) + "x + " +
format(c,0) + "x^2 + " + format(d,0) + "x^3");
}  // end if

if (order==2) { // second order polynomial
puts(format(a,0) + " + " + format(b,0) + "x + " +
format(c,0) + "x^2");
}  // end if

if (order==1) { // first order polynomial
puts(format(a,0) + " + " + format(b,0) + "x");
}  // end if
```

If convergence was achieved in 100 iterations or fewer, the approximate value of the root and the corresponding value of the polynomial are displayed. If the order of the polynomial is greater than 1, the user is reminded that there may be other roots.

```
// If convergence occurred display value of the
      root and final value of polynomial

if (sentry <= 100) {  // the while was terminated
      by convergence not by sentry

    puts("\n\n" + "approximate root= " +
     format(x,4)+ "\n\n" + "polynomial
     value= " + function);

    if ( order > 1 ) {
        puts ( "\n\n" + "Initial guess " +
          format(x0,0) + "\n\n" + "Other
          guesses may yield" + "\n" + "other
          real root(s).");

    } // end if
```

When convergence does not occur in 100 iterations, the last value of x and the corresponding values of the function and its derivative are displayed, and the user is informed that there was no convergence within 100 iterations.

```
    } else {  // if while was terminated by sentry give final values of function and
                 derivative

              puts ("\n\n" + "x = " + format(x,4) + "\n" + "function = " + function +
                   "\n" + "derivative = " + derivfunction);
              puts ("\n\n" + "No solution after " + sentry + " iterations." + "\n\n" +
                    "There may be no real roots" + "\n" +"Program terminated.");


     } // end if

    wait();
    exit();

} // End main
```

Conclusion

This article demonstrates the commands required in PocketC to produce a stand-alone (prc) file. It further illustrates the use of a "While" structure and the use of logical AND. In addition to its tutorial value, this applet may be a useful tool for engineering students. The resulting prc file will be made available for download at no charge. A later article in this series will give details of where to find the files.

References

1. Wiggins, Edwin G., "Programming the Palm Tungsten," Computers in Education Journal, October-December 2005, Vol. XV No.4 pages 53-57.

Biographical Information

Edwin G. Wiggins holds BS, MS, and Ph.D. degrees in chemical, nuclear, and mechanical engineering respectively from Purdue University. He is the Mandell and Lester Rosenblatt Professor of Marine Engineering at Webb Institute in Glen Cove, NY. Ed is a past chairman of the New York Metropolitan Section of the Society of Naval Architects and Marine Engineers (SNAME) and a past regional vice president of SNAME. A Centennial Medallion and a Distinguished Service Award recognize his service to SNAME. As a representative of SNAME, Ed Wiggins serves on the Engineering Accreditation Commission of the Accreditation Board for Engineering and Technology.