IMPLICIT DIFFERENTIATION ON THE TI-92+ CALCULATOR AS AN ILLUSTRATION OF SOME POWERFUL PROGRAMMING FEATURES

By John Alexopoulos,
Kent State University, Stark Campus

Introduction

The TI-92+ calculator among other things offers its users a powerful Computer Algebra System (CAS) and 3-D graphing capabilities. One of the not so well known features of the TI-92+ is its pseudocode-like, natural programming language. By focusing on the problem of implicit differentiation, we provide an example on how programming the TI-92+ could enhance the teaching of introductory calculus and computer science. Since the calculator lacks the built-in commands to perform implicit differentiation, programming such a task is a meaningful exercise in symbolic computation for the novice programmer possessing some knowledge of calculus.

We present two approaches to this problem. The first approach requires that the programmer is familiar with only the most elementary mathematical tools. The program provides us with the opportunity to explore some of the TI-92+'s powerful string commands as well as its unusual ability to convert expressions to strings *and vice-versa.* The second approach requires familiarity with multivariate calculus. This approach rewards the programmer with much greater efficiency and generality. We believe that both approaches are accessible to the novice programmer with the first approach requiring more programming skills than the second. As it is often the case, the use of advanced mathematical tools can considerably simplify many tasks.

1. Implicit Differentiation, using elementary tools.

We begin by stating the problem:
Let f be a function of two variables, say x and y, implicitly defining y as a function of x by the equation $f(x, y) = 0$. We need to find the derivative of y with respect to x, in terms of x and y. That is, compute dy/dx as a function of x and y [1, pp. 146-150].

Keeping in mind that $y=g(x)$ for some function g the procedure involves solving the equation df/dx=0 for dy/dx. Implementing this procedure as a function on the TI-92+ presents the programmer with a variety of technical difficulties:

1. The TI-92+ will regard y as a constant when the expression df/dx is evaluated. In order to circumvent this problem, the programmer must substitute $g(x)$ (or something similar) in place of y.
2. Once the expression df/dx is evaluated, we need to replace $g(x)$ and dy/dx with y and δ respectively and then solve for δ. The difficulty at this stage lies with the machine's inability to substitute variables in place of whole expressions from within a program. Our way of dealing with this problem is to exploit the TI-92+'s ability to convert expressions to strings and vice-versa: A locally user-defined function *replace*() allows

the replacement of all occurrences of a given string within another string with a third string. The availability of powerful string commands makes this task relatively easy.

3. Once d*f*/d*x* is converted back to an expression, we need to solve the equation d*f*/d*x*=0 for δ. We found the various build-in solve commands slow and unreliable. In order to improve the program's performance, we exploit the fact that the expression d*f*/d*x* is linear on δ. Thus $\delta = \dfrac{-(df/dx)|_{\delta=0}}{[df/dx-(df/dx)|_{\delta=0}]|_{\delta=1}}$.

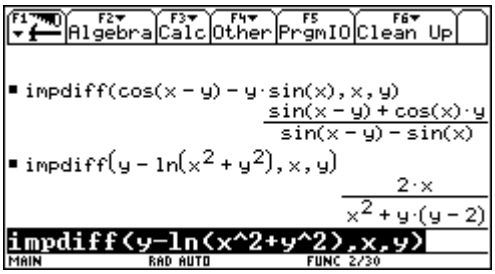
Next we show an implementation of implicit differentiation as a function on the TI-92+. The code is natural and offers much more than what was intended to. The TI-92+'s powerful ability to apply an operator across a list or a matrix can be exploited even in user defined functions. The ordinary arithmetic operators can be applied across lists and matrices, but in the case of matrices, * means *matrix* multiplication and / means multiplication by the inverse of a matrix. The use of the operators .+, .-, .* and ./ in place of ordinary addition, subtraction, multiplication and division, ensures that the machine will perform these operations coordinate-wise when lists or matrices are present in the input. As a result, our implicit differentiation function is capable of finding the derivatives of functions, lists of functions or even matrices of functions defined implicitly. All this at no additional cost to the programmer in terms of handling different data types. The code and the pictures of the screen that follow illustrate this:

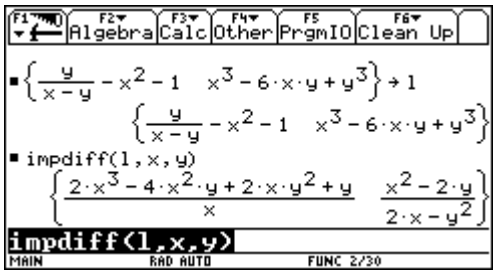
```
impdiff(f,x,y)
Func

¨ This function computes dy/dx from the implicit relation f(x,y)=0.

Local  replace,f,g,d

Define  replace(str1,str2,str3)=Func

    ¨ This function replaces every occurrence of str2 in str1 with
      str3.

    Local  l1,l2,n,leftstr,str1
    dim(str2)»l2
    ""»leftstr
    inString(str1,str2)»n
    While  n≠0
      dim(str1)»l1
      leftstr&left(str1,n-1)&str3»leftstr
      right(str1,l1-n-l2+1)»str1
      inString(str1,str2)»n
    EndWhile
    Return  leftstr&str1
    EndFunc

¨ Find df/dx assuming that y=g(x).

¶(f|y=g(x),x)»d

¨ Replace all occurrences of ¶(g(x),x) and g(x) with … and y
```

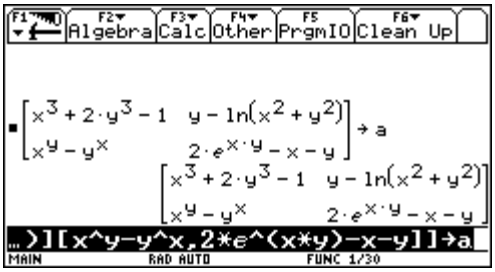
respectively.

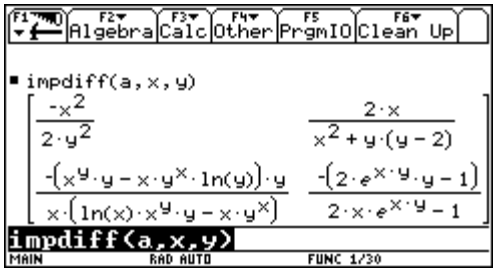
```
string(d)»d
replace(d,"¶(g("&string(x)&"),"&string(x)&")","…")»d
replace(d,"g("&string(x)&")",string(y))»d
expr(d)»d

¨ Solve for dy/dx

ªd|…=0»f
d+f|…=1»g
Return  f ./ g
EndFunc
```

The following pictures illustrate a few sample calls for the preceding function:









2. Implicit Differentiation, using the Implicit Function Theorem.

From a programming point of view the problem can be substantially simplified, if one brings to bear tools from multivariate calculus such as *the Implicit Function Theorem*. Indeed if *f* is a function of *x* and *y*, implicitly defining *y* as a function of *x* by the equation $f(x, y) = 0$ then $\frac{dy}{dx} = -\frac{\partial f}{\partial x} \Big/ \frac{\partial f}{\partial y}$ [1, p. 799].

In general, if $f^{(n-1)}$ is the $(n\text{-}1)$th derivative of y with respect to x, in terms of both x and y, then by the generalized chain rule [1, p.797] we have that

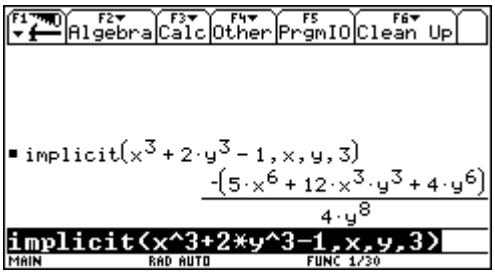
$$f^{(n)} = \frac{d^n y}{dx^n} = \frac{df^{(n-1)}}{dx} = \frac{\partial f^{(n-1)}}{\partial x} + \frac{\partial f^{(n-1)}}{\partial y} \cdot \frac{dy}{dx}.$$

Next we show a slightly more general implicit differentiation function on the TI-92+. This function computes *n*th order implicit derivatives. The code is very simple and

natural. It too offers much more than what was intended by allowing lists and matrices as input parameters:

```
implicit(f,x,y,n)
Func

¨ This function computes the nth derivative of y with respect to x where
  y is implicitly defined as a function of x by the relation f(x,y)=0.

Local f,i,d
a¶(f,x) ./ (¶(f,y))»d
d»f
For i,2,n
  ¶(f,x) .+ d .* ¶(f,y)»f
EndFor
Return f
EndFunc
```

As you can see, the use of the Implicit Function Theorem makes the programming of an implicit differentiation function on the TI-92+ almost trivial. A picture of the screen featuring a sample call for this function follows:



References

[1] J. Stewart, **Calculus**, third edition, Brooks/Cole, 1995

jalexopoulos@stark.kent.edu

Kent State University, Stark Campus
6000 Frank Avenue NW
Canton, OH 44720