Comments about

SMatth doesn't have true local variables, so it's usual put some decoration for avoid that issue. Maple have the sentence local ( ) and Mathematica have Block[ ] and Module[ ] for provide that functionality. For example

$f(x) := k \cdot x + 1$

$MySum(f(1), x, n) :=$
$s := 0$
$\text{for } k \in [1..n]$
$\quad s := s + f(x)$
s

$MySum\#(f\#(1), x\#, n\#) :=$
$s\# := 0$
$\text{for } k\# \in [1..n\#]$
$\quad s\# := s\# + f\#(x\#)$
$s\#$

$MySum(f(x), 2, 3) = 15$

$MySum\#(f(x), 2, 3) = 3 \cdot (1 + 2 \cdot k)$

Comments about feval

SMath have the equivalent of matlab function handler @ indicating the "signature" of the function, that is the number of paraters that it have. That enable to define f, f(x), f(x,y) as different things. But have a lot of issues handling 3 cases

Functions with if

Functions with functions defined in plugins

Functions that call functions

You can use Unknowns(f) for check if procedures like solve, rotts, diff, int and others goes to work or not. For example

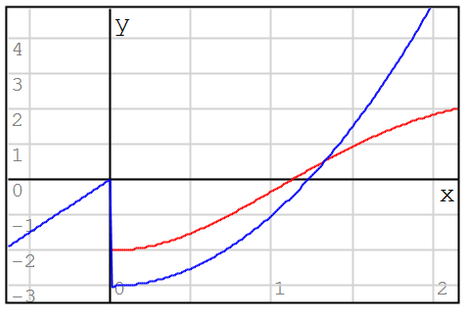
$f(x) := \text{if } x > 0$
$\quad 2 \cdot x^2 - 3$
else
$\quad 3 \cdot x$

$g(x) := 5 \cdot Bessel(x, 4)$

$Unknowns(f(x)) =$ ∎

$Unknowns(g(x)) = [x]$

lastError = "x - not defined."



$f(x)$
$g(x)$

Using this feval you can handle them, or more or less something like a workaround. Also, feval is like feval's old matlab, before the introduction of functions handlers.

$feval(\#fs\#, \#x\#) := str2num(concat(num2str(\#fs\#), "(", num2str(\#x\#), ")"))$

$feval(f, 2) = 5$

$feval("g", 2) = 1.8206$

Comments about Bis

In SMath if you write a procedure and in its body you change the value of one of its parameters, then you change that variable globally. Actually, that's a great SMath feature. For example

$f(x, k) :=$
$k := 2$
$k \cdot x$

$k := 3$ $f(1, k) = 2$ $k = 2$

Following Bisection takes as arguments the string of an univariate function name, the bounds of some real interval where the function have a sign change and two epsilons: one for check if the function value is enough small, εy, and another for check if the interval where you are working is enough small.

About εy, notice that in numerical procedures the assertion a = b have not too much sense, it is just a casuality. Better is ask if abs(a-b) < ε for some ε value. In the examples I use εy = 0, this is, check if for casuallity f(x) = 0, and try to find and interval if not.

About εx notice that its value determine the number of iterations, given by $N = \log_2\left(\frac{b-a}{\varepsilon}\right)$

Finally, the returned value c indicates that f have a zero in the interval $c - \varepsilon x \le c \le c + \varepsilon x$

```
Bis(f#, ao#, bo#, εy#, εx#):=
    "Avoid parameter modification"
    [a# b#]:=[ao# bo#]
    "Initialize"
    [ya# yb#]:=[feval(f#, a#) feval(f#, b#)]
    if sign(ya#)=sign(yb#)
        "Check interval"
        error("Bad interval")
    else if |ya#|≤εy#·UnitsOf(ya#)
        "Check if one bound the solution"
        c#:=a#
    else if |yb#|≤εy#·UnitsOf(yb#)
        "Check the other bound"
        c#:=b#
    else
        "Number of iterations"
        N#:=1+round(ln(|(b#-a#)/UnitsOf(b#-a#)|)-ln(εx#))/ln(2), 0)
        for iter#∈[1..N#]
            "Find the middle point"
            [c#:=eval(0.5·(a#+b#)) yc#:=feval(f#, c#)]
            if |yc#|≤εy#·UnitsOf(yc#)
                "The function value is enough small"
                break
            else
                if sign(yb#)·sign(yc#)>0
                    "The solution is in the right interval"
                    [b# yb#]:=[c# yc#]
                else
                    "The solution is in the left interval"
                    [a# ya#]:=[c# yc#]
    c#
```

```
ndiff_1(f#, a#, δx#):=
    (feval(f#, a#+δx#·UnitsOf(a#))-feval(f#, a#))/(δx#·UnitsOf(a#))
```

Usual values $\varepsilon y := 0$ $\varepsilon x := 10^{-15}$ $\delta x := 10^{-7}$

function $F(x) := x \cdot \text{BesselJ}(1, x) - 1.01 \cdot \text{BesselJ}(0, x)$

derivative $f(x) := \left| ndiff_1(F, x, \delta x) \right.$

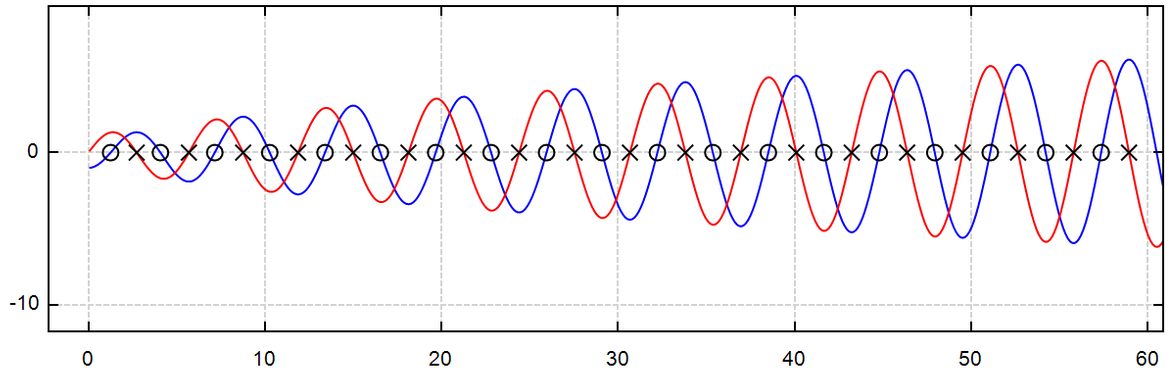
interval $[a\ b] := [0\ 60]$

sub intervals $N := 100$ $\quad X := a + \frac{b-a}{N} \cdot [0..N]$ $\quad n := 0$ $\quad m := 0$

Scan each subinterval for a root, and store it in a vector, detecting where there are a change of sign

for $k \in [1..N]$
  if $sign(F(X_k)) \cdot sign(F(X_{k+1})) \le 0$
    $xo_{n := n+1} := Bis(F, X_k, X_{k+1}, \varepsilon y, \varepsilon x)$

for $k \in [1..N]$
  if $sign(f(X_k)) \cdot sign(f(X_{k+1})) \le 0$
    $uo_{m := m+1} := Bis(f, X_k, X_{k+1}, \varepsilon y, \varepsilon x)$



$F(x)$
$f(x)$
$augment(xo, \overrightarrow{F(xo)}, "o")$
$augment(uo, \overrightarrow{f(uo)}, "x")$

Alvaro

$appVersion(4) = "1.0.8253.4763"$