[4.10] Faster plots of integral functions

Suppose you want to plot an integral function with an independent variable *x*. Such a function might be in the form

$$\int_{X_0}^{X} f(t)dt$$

where x_0 is a constant. For example, the sine integral function is defined as

$$Si(x) = \int_0^x \frac{\sin(t)}{t} dt$$

You should always first try to find a symbolic expression for the integral; in this case, there is no closed form solution. The most obvious way to plot the function is to define it as a function in the Y= Editor, for example,

```
y1=nInt(sin(t)/t,t,Ø,x)
```

This certainly works, but it is predictably slow, because the integral must be evaluated for each plotted point. For a HW2 92+ with AMS 2.05, it takes about 15 minutes to plot the function with ZoomFit, over a range of 0 to 15 with a plot resolution of one pixel. This is about 3.78 seconds/point. For a HW1 TI89, the time is about 5.41 seconds/point. The efficiency is even worse if we want to plot the function over a range that does not include x_0. In other words, we want to plot the function over some range x_l to x_h, and $x_l > x_0$. Then, the integral from x_0 to x_l is recalculated for every plotted point, but is really the same for each point.

We can considerably reduce the plotting time by recognizing that the integral at each point is just the sum of the integral over the previous plotted points, plus the integral over the interval of the current point. The plotting time is also shortened by generating a lists of x-coordinates and corresponding function values, then plotting with a data plot instead of a function plot. This program, *plotintg(),* implements these ideas:

```
plotintg(fname,xØ,xl,xh,pn,res)
Prgm
©(f(x) or "f(x)",lower ∫ limit,xlow,xhigh,plot#,xres) plot ∫(f(x))
©leaves global variables xx and yy
©11decØØ/dburkett@infinet.com

local dxx,k,p
© dxx is the x-distance between the plotted points
© k is a counter in the loops
© p is (screen width - 2)

©Delete any existing xx and yy variables
delvar xx,yy

©Find x-step and make x-list
getconfg()[dim(getconfg())-1Ø]-2→p     ©Get 'p' for 89 or 92+
res*((xh-xl)/p)→dxx                    ©dxx set by screen width & resolution
seq(k*dxx+xl,k,Ø,floor(p/res))→xx      ©Use floor() so we never try to plot
                                       ©beyond xh

©Find integral at 1st point
©Note that the calculation method depends on whether fname was passed as an
©expression or a string
if gettype(fname)="STR" then
```

```
  when(x0=xl,0,nint(expr(fname),x,x0,xl))→yy[1]
else
  when(x0=xl,0,nint(fname,x,x0,xl))→yy[1]
endif

©Find integral at remaining points
© For each plotted point, find the integral between the current value of x and
© the previous value of x, then add the previous integral to find the total

©Handle case where fname is passed as a string
if gettype(fname)="STR" then
  for k,2,dim(xx)
    nint(expr(fname),x,xx[k-1],xx[k])+yy[k-1]→yy[k]
  endfor

©Handle case where fname is passed as an expression
else
  for k,2,dim(xx)
    nint(fname,x,xx[k-1],xx[k])+yy[k-1]→yy[k]
  endfor
endif

©Set graph window limits
xl→xmin
xh→xmax
min(yy)→ymin
max(yy)→ymax

©Create plot; display graph
fnoff                    ©Turn off all function plots
plotsoff                 ©Turn off all data plots
newplot pn,2,xx,yy,,,,5  ©Generate xyline plot with dot symbols
dispg                    ©Display the plot

EndPrgm
```

You should run *plotintg()* in Approx mode, and the Graph Mode must be set to FUNCTION. This same program works with either the 89 or the 92+. If you have a global variable called *x*, delete it before running *plotintg()*.

These are the program arguments:

fname	The name of the function to be plotted, as an expression or string. The plot is much faster if *fname* is an expression, but some functions can't be passed as expressions. The independent variable must be *x* May also be an expression in *x*
x0	The lower integration limit
xl	The minimum value of *x* to plot You must set xl < xh
xh	The maximum value of *x* to plot You must set xh > xl
pn	The number of the data plot to use. If you have existing data plots in the Y= Editor that you don't want destroyed by the program, choose an unused plot number.
res	The plot resolution in pixels, If *res* = 1, then the function is plotted at each pixel. If *res* = 3, then the function is plotted at each third pixel. *res* must be greater than or equal to 1. There is no upper limit, above resolutions of 20 the plot is very coarse.

Note that you can pass the integrand *fname* as an expression or a string. This flexibility is provided so that expressions plot quickly, but the program still works for integrands that cannot be passed as expressions. This is a limitation of TI BASIC. In general, functions that contain conditional expressions cannot be passed as expressions. If you are not sure that your function can be passed as an expression, just try it - you will quickly get an error message.

For example, to plot a function called *fun1()* from x = 0 to x = 15, use this call:

```
plotintg("fun1(x)",Ø,15,1,2)
```

which will use plot number 1 and a resolution of 2 pixels. To plot the sine integral shown above, you could use this call:

```
plotintg(sin(x)/x,Ø,15,1,1)
```

which plots the sine integral from x=0 to x=15, using data plot number 1 and a resolution of 1.

plotintg() has these side effects:

- All functions and plots in the Y= Editor are turned off.
- The global variables *xx* and *yy* are created and not deleted. If you have any variables called *xx* or *yy*, they will be overwritten.
- The data plot remains selected in the Y= Editor.
- The plot Window variables are changed and not restored.

The program listing above includes comments which are not included in the actual source file. Remove the comments if you are typing the program instead of using GraphLink.

A design goal for the program was never to attempt to plot beyond the maximum x-value specified by *xh*. This goal is desirable because the function might not even be defined beyond *xh*, or it might be increasing so rapidly that the resulting y-axis scaling would degrade the plot resolution. This is prevented by using the *floor()* function to calculate the number of points plotted. The side effect of this approach is that the function might not be plotted all the way to *xh* if the resolution is not set to 1. This is not catastrophic, in fact, the built-in function and data plotters do the same thing.

The time savings with *plotintg()* can be significant, as shown below. The numbers in parenthesis show the plot time in seconds/point. For the *plotintg()* results, execution times are shown both for passing the integrand as a string (STR) and as an expression (EXP).

Integrand function	TI89 HW1 built-in plotter	TI89 HW1 plotintg()	TI92+ HW2 built-in plotter	TI92+ HW2 plotintg()
sin(x)/x x0 = xl = 0 xh = 15 res = 1	14:25 (5.41)	STR: 3:59 (1.50) EXP: 3:27 (1.30)	15:07 (3.78)	STR: 4:21 (1.09) EXP: 2:34 (0.64)
ln(x)/sin(x) x0 = xl = 0.1 xh = 3.1 res = 1	Not finished in 35 minutes, "Questionable accuracy' shown	STR: 5:14 (1.97) EXP: 3:58 (1.50)	Not finished in 30 minutes, 'Questionable accuracy' shown	STR: 5:38 (1.41) EXP: 3:01 (0.76)
1 x0 = xl = 0 xh = 1 res = 1	0:42 (0.26)	STR: 1:27 (0.55) EXP: 1:16 (0.48)	0:43 (0.18)	STR: 1:42 (0.43) EXP: 1:32 (0.38)
x x0 = xl = 0 xh = 1 res = 1	0:47 (0.30)	STR: 1:42 (0.64) EXP: 1:21 (0.51)	0:73 (0.31)	STR: 1:58 (0.49) EXP: 1:36 (0.40)

For the first two functions in the table, *plotintg()* is much faster than the built-in plotter. However, for the last two functions, the built-in plotter is faster. Perhaps if the 89/92+ AMS can find a symbolic solution, it uses that solution to evaluate the integral, which would be much faster. Again, always first try to evaluate the integral symbolically - plotting the symbolic solution will always be faster than evaluating the integral.

plotintg() does no error checking. Results may be unpredictable if invalid arguments are used.

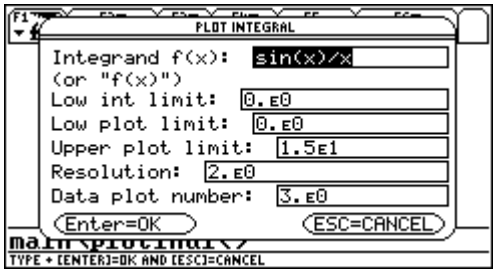
The program shown below, *plotinui()*, provides a user interface to *plotintg()*. It simply prompts for the arguments, does some simple validation of the user input, and calls *plotintg()*.

```
plotinui()
Prgm
©Interface for plotintg() integral plotter
©Calls plotintg() in same folder
©12dec00/dburkett@infinet.com

local umode,et
©umode saves the user's modes, to be restored on exit
©et is the error dialog box title

©Initialize
©Set the error dialog box title
"ERROR"→et
©Save the user's modes
getmode("0")→umode
©Set modes to Function plot, full screen, and Approx
setmode({"1","1","8","1","14","3"})

©Initialize settings if nonexistent
©The settings are saved as global variables, for defaults each time the program
©is executed. If igrand does not exist, the program assumes none of the
©parameters exist, and they are initialized to valid values.
```

```
©These global variables are created in the current folder, and are NOT deleted.

if gettype(igrand)="NONE" then
 x→igrand    ©integrand
 Ø→lil       ©lower integration limit
 Ø→lpl       ©lower plot limit
 1→upl       ©upper plot limit
 1→res       ©plot resolution
 3→dpn       ©data plot number
endif

©Main loop
lbl l1

©Convert parameters to strings for dialog box
string(igrand)→igrand
string(lil)→lil
string(lpl)→lpl
string(upl)→upl
string(res)→res
string(dpn)→dpn

©Prompt for all parameters
dialog
 title "PLOT INTEGRAL"
 request "Integrand f(x)",igrand
 text "(or ""f(x)"")"
 request "Low int limit",lil
 request "Low plot limit",lpl
 request "Upper plot limit",upl
 request "Resolution",res
 request "Data plot number",dpn
enddlog

©Convert parameters from strings to expressions
expr(igrand)→igrand
expr(lil)→lil
expr(lpl)→lpl
expr(upl)→upl
expr(res)→res
expr(dpn)→dpn

©Give user the chance to quit here
if ok=Ø:goto l2

©Validate parameters:
©lower plot limit must be less than upper plot limit,
©resolution must be an integer greater than zero,
©data plot number must be an integer from 1 to 99
©If parameter is invalid, display error message and return to input dialog box

©Validate upper & lower plot limits
if lpl≥upl then
 dialog
  title et
  text "Lower plot limit"
  text "must be less than"
  text "upper plot limit"
 enddlog
 goto l1

©Validate plot resolution
```

```
elseif res≤0 or fpart(res)≠0 then
 dialog
  title et
  text "Resolution must be an"
  text "integer greater than zero"
 enddlog
 goto l1

©Validate data plot number
elseif dpn≤0 or dpn>99 or fpart(dpn)≠0 then
 dialog
  title et
  text "Data plot number must be"
  text "an integer 1 to 99"
 enddlog
 goto l1
endif

©Plot the integral
plotintg(igrand,lil,lpl,upl,dpn,res)
pause
goto l1

©Exit
lbl l2
setmode(umode)      ©Restore modes

EndPrgm
```

This program works on both the 89 and 92+. The input dialog box looks like this on the 92+:

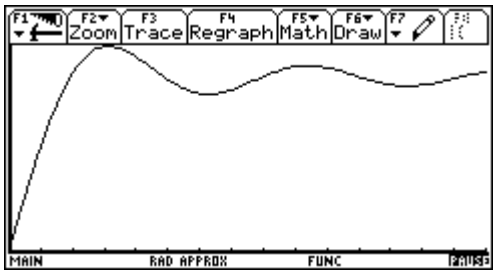


Note that all the input parameters can be entered, as expected, and that the integrand can be entered as an expression or a string. The program saves the user's modes and restores them when the program is done. Press [ESC] from this dialog box to exit the program, or press [ENTER] to plot the integral. The plot is shown until you press [ENTER], then the input dialog box is displayed again. This lets you make more than one plot without restarting the program.

The program creates several global variables, so that you do not need to re-enter the parameters to repeat a plot. Therefore, the global variables are *not* deleted. These variables are

```
igrand, lil, lpl, upl, res, dpn
```

Finally, if you're curious as to what the sine integral looks like over the range of zero to 15:


F1
F2
Zoom
F3
Trace
F4
Regraph
F5
Math
F6
Draw
F7
MAIN
RAD APPROX
FUNC
PAUSE