

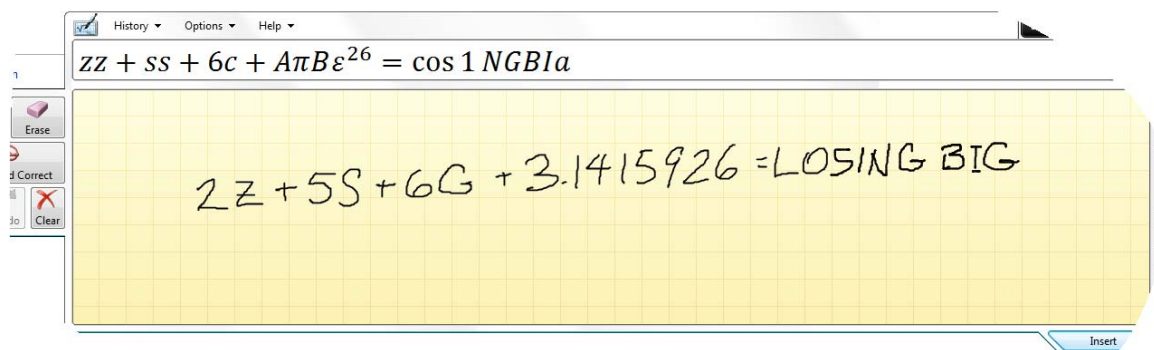
Using the Math Input Panel on Windows 7

Richard Fateman March 11, 2010

Experiments using an accessory graphics tablet (Wacom Cintiq). These are screen clips from interactions in which the “recognized result” is in the small panel in the upper left. The large panel with handwriting in it is my attempt to write some math. It is not particularly comfortable to write on a smooth hard plastic surface with a slippery pen. I accommodated to writing on a horizontal surface with feedback coming from a vertical display, though it is important to be aligned properly. Writing and viewing on the same surface, as in a tablet PC, is a different and not necessarily improved experience. Your hand and stylus obscure a critical piece of the display surface. If some elaborate visual feedback is designed at the stylus point, it may not be visible. (Forget using “change the cursor” feedback).

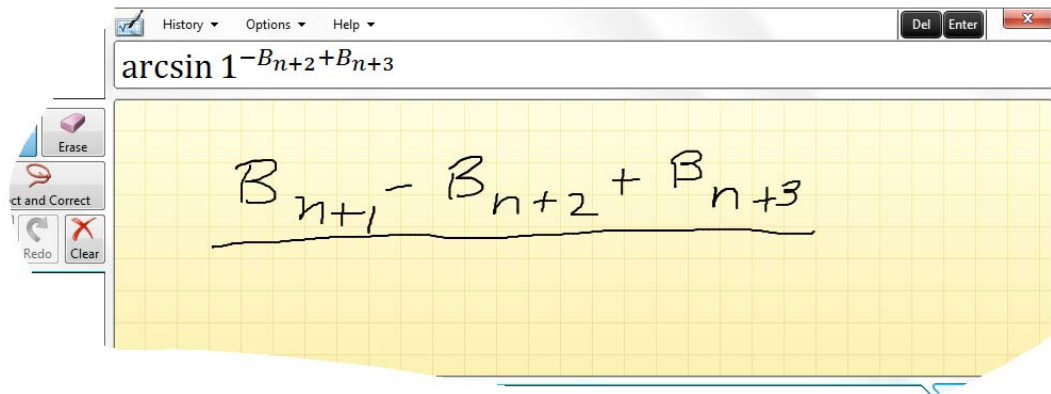
There are positive aspects to using a Tablet PC vs an accessory tablet, and so my experience may be less favorable. The Tablet PC has better integration with the sensor and “inking”, including (probably) a higher sampling rate, higher resolution in time, position, and pressure. The abstraction of “ink strokes” may provide more information in such an environment. The tablet PC also has a larger active surface than the Wacom tablet; the one I used was 5 inches wide by 4 inches high.

Here is an example of stuff that I thought would confuse the math input panel, e.g. 2 vs z, 5 vs s and 6 vs G, but I was surprised by the inability of the math input panel to do a better job with numbers.

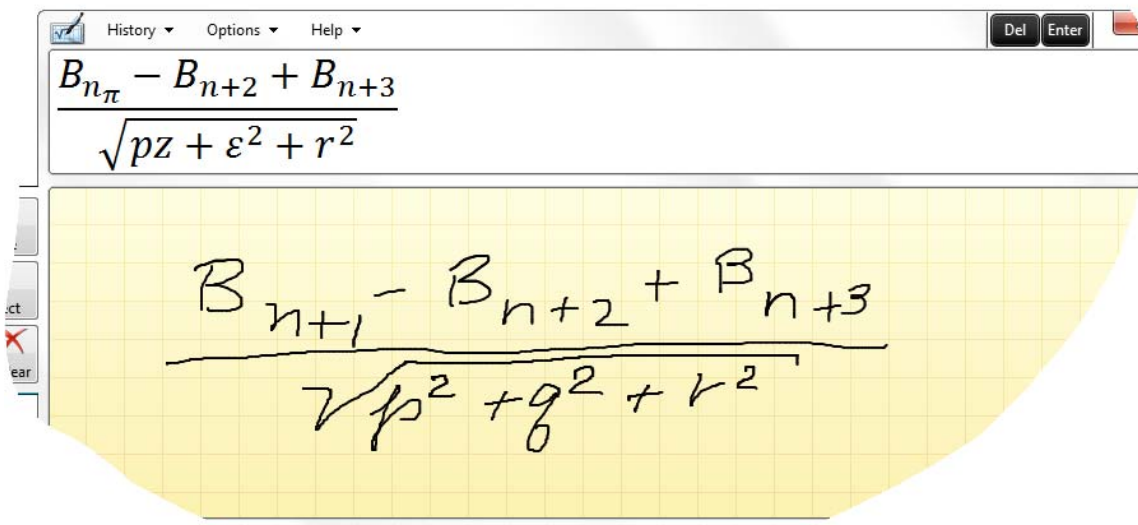


While it is not apparent from these static displays, the MIP apparently re-parses the whole expression after each stroke (or collection of strokes); incomplete or unsyntactic input lead it astray, and it issues warnings that you should complete the expression.

In the following two screen clips you can see what happens when we first write a horizontal line; it vastly improves when we place something in the denominator:



Now put something in the denominator.



One of my favorite examples is the ambiguity of $|<$ vs the letter K. See it here. Of course getting k vs K right is tough too.

A screenshot of a handwriting recognition software interface. The top window displays the recognized text: $tr ue = 1 < 2 + k_2 + k^{3+y^4} \frac{\eta}{2}$. Below this is a yellow grid area where the original handwritten text is visible: $tr ue = 1 < 2 + k_2 + k^3 + y^4 \frac{\eta}{2}$. The interface includes a toolbar on the left with buttons for Erase, Select and Correct, Undo, Redo, and Clear. The top window has a menu bar with History, Options, and Help, and buttons for Del, Enter, and a close button.

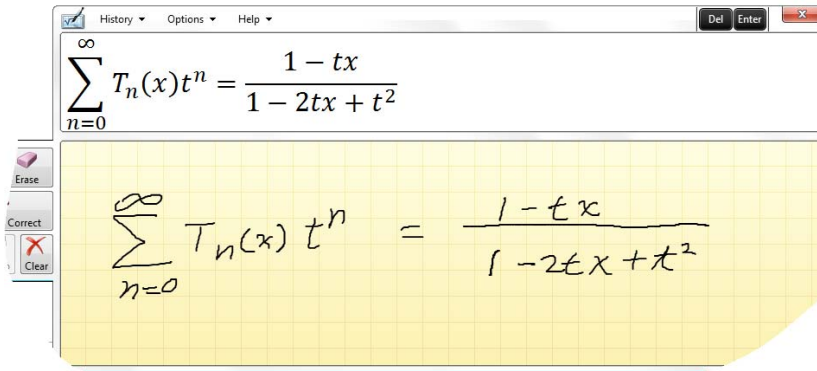
To get this next expression to come out right at all we had to place the subscript on the T so that it was not below the bar of the T. It seems that (my?) tendency to “kern” certain letters, like capital T will confuse the recognizer to problems. For that matter, lower-case t looks rather like a +.

A screenshot of a handwriting recognition software interface. The top window displays the recognized text: $T_{nn}^{+T} - 1 = T_{n+1}$. Below this is a yellow grid area where the original handwritten text is visible: $T_n + T_{n-1} = T_{n+1}$. The interface includes a toolbar on the left with buttons for Write, Erase, Select and Correct, Undo, Redo, and Clear. The top window has a menu bar with History, Options, and Help, and buttons for Del, Enter, and a close button.

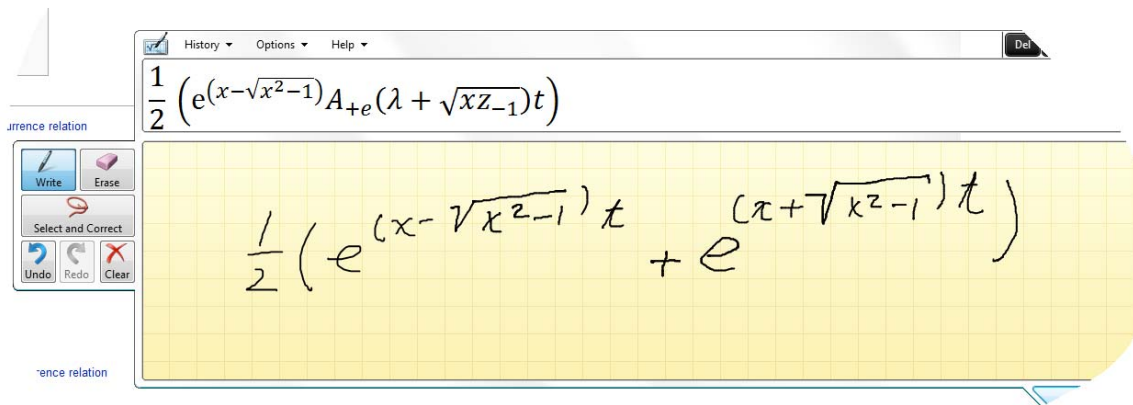
$$I_i = 2x_{n-1}^T - T_{n-2}$$

The image shows a digital whiteboard interface. At the top left, the equation $I_i = 2x_{n-1}^T - T_{n-2}$ is displayed. Below it is a toolbar with buttons for 'Write', 'Erase', 'Select and Correct', 'Undo', 'Redo', and 'Clear'. The main area of the whiteboard is a yellow grid with the handwritten equation $T_n = 2x_{n-1}^T - T_{n-2}$ written in black ink. Above the grid is a header bar with 'History', 'Options', and 'Help' menus, and a 'Del' button. The equation in the header bar is $n^T = 2x_{n-1}^T - T_{n-2}$.

Finally, an example in which everything works as we would like it to. Unfortunately, it required many erasures and corrections, and a fairly careful adherence to a base line. Writing a program capable of separating “touching” symbols as the 2t below, may not be as hard as you might think, since there are “temporal” spaces too. I erased the “2” and rewrote it later, so these letters are definitely separate ink objects or strokes.



One of our first examples, and we could not get it to work better, even after many corrections.



Conclusion: you can provide neat demos with the MIP, and undoubtedly with care, forethought, and training you can use it more effectively. That is, you can improve its success by careful choice of characters, neatness in drawing the characters, and knowledge of the available syntax. Yet it seems unlikely to be a good technology for an untrained (even if mathematically adept) person to try entering math equations.

The user interface for corrections is difficult to use, though the design of an improved system is undoubtedly a challenge. How to select, correct, replace pieces? How to nudge a selection to a proper place, size, slant?

While there is a history mechanism to allow you to use previous expressions from the same execution of the program, there is no way of (re) introducing expressions from other programs.

The output of the recognition is available as MathML, placed on the OS clipboard and so can be pasted in various programs such as Microsoft Word, willing to accept MathML.

(disappointingly, the emacs editor did not accept these objects, though they should look like text.)