0.1 Algorithms evaluation

Based on our experience and with the information gleaned from the interview, we decided to implement at least the following transcendental functions: x^y, e^x, $sin(x)$ and $cos(x)$, $tan(x)$, $ln(x)$ and $sqrt(x)$. Currently, optional functions are $arcsin(x)$, $arccos(x)$, $arctan(x)$. In order to develop solutions for our calculator without help from Java's math library, we would have to do research on numerical methods to compute these functions. We would have to find the sites with mathematical references such as Wolfram-Alpha, Wikipedia and YouTube in order to find solutions that would be feasible for us.

A few of these functions ($sin(x)$, $cos(x)$, $tan(x)$, e^x, and the $arc*(x)$ functions could be approximated relatively easily by Taylor polynomials. Series lend themselves naturally to iterative methods (ie. simple 'for' loop) so that is what these algorithms are.

For example, here is the algorithm for e^x:

$result \leftarrow 0$
$precision \leftarrow p$ (arbitraty precision integer)
$iterator \leftarrow 0$

while $iterator < precision$ **do**
  $result \leftarrow result + \frac{x^i}{i!}$
  $i \leftarrow i + 1$

return $result$

This algorithm does not pass test for a wide range of values of e^x. Currently, we are debugging this. This is an extremely simple algorithm compared to the other but they mostly take this form. The complexity of this particular algorithm is constant, however, it calls the x^i function which we have implemented hamfistedly as x multiplied by itself i times. We intend to optimize the power function in later iteration to take advantage of the many optimizations possible (ie. bitshifting, etc.).

The algorithm for $ln(x)$ is:

if $result \leq 0$
  $return$ **error**

$precision \leftarrow p$ (arbitraty precision integer)
$iterator \leftarrow 0$

while $iterator < precision$ **do**
  $result \leftarrow result + \frac{1}{(2i+1)} * \frac{(x-1)}{(x+1)}^{(2i+1)}$
  $i \leftarrow i + 1$

return $(2 * result)$

The $ln(x)$ algorithm is of a similar vein as e^x and of a similar complexity. Tests have been successful up to about x = 870. We are still debugging this. We think it has something to do with our algorithms being centered around 0 and the solution becoming less and less precise as we increase or decrease x away from 0. We will definitely have this sorted out for iteration 2.