BigFloat (and LittleFloat) Spreadsheet Project Notes

Robert Weaver, Saskatoon, Canada, 2018-11-03

This is a XOJO open source spreadsheet project that was submitted as part of the XOJO Summer 2018 "Just Code" Challenge. It consists of two project files: a Big Float version, and a regular "Little Float" double precision version. The Big Float version makes use of Robert Delaney's extended precision math **fpPlugin**. The fpPlugin is free and can be downloaded from Robert Delaney's Website. The Little Float version is provided for developers who would like to experiment with the spreadsheet code without having to download the fpPlugin. Except for the difference in mathematical precision, and certain omitted math functions, the programs are virtually identical. The principal difference is in the ExpressionEvaluator class. The original double precision version of the class is used in the Little Float project file, and a newer Big Float version of the class used in the Big Float project file. Each one is a nearly drop-in replacement of the other. However, the Big Float version of the ExpressionEvaluator class includes additional math functions that are available as part of the fpPlugin.

The original version that was submitted to the Just Code challenge was put together very quickly. As a result, it was very limited in functionality, and was also later found to have a bug which sometimes caused cell formulas to be calculated in the wrong order. In the month since its original submission, I've occasionally tinkered with it, fixing bugs and adding functionality. The out of order cell calculation bug has been fixed, along with a few other minor irritations. The original version allowed only one spreadsheet document to be open at a time. Now, multiple spreadsheets can be open simultaneously.

Basic Operations

A new spreadsheet document window opens automatically when you run the program. The window shows a standard spreadsheet style cell grid with a one line edit field at the very top of the window. To enter numbers, text or formulas into a cell, click on the cell to select it (or use the arrow keys) and then type the information into the edit field. (At present, there is no facility to type directly into the cell.) To enter a formula, the first character must be an equal sign, as per usual spreadsheet operation. You can type cell references directly into the formula, or click on the desired cell to have it entered automatically. For cell ranges, as used by various summary functions, click and drag across the desired range of cells. Fill Right and Fill Down functions are provided to propagate formulas across rows and columns. Relative cell references are automatically adjusted. Relative references may be converted to absolute references by preceding the row and/or column reference with a $ symbol. Anyone who has used a spreadsheet program should have no trouble figuring out the basic operations.

To set the precision to be used in the spreadsheet, use the Set Precision function in the Operations menu. This function is present in both versions of the program, but is ignored in the Little Float version (precision is always about 16 significant digits).

Numbers may be formatted by selecting a cell or range of cells and then choosing the Number Format function from the Operations menu. The format string text is the same as is used in the XOJO format function. There are some limitations to this in the Big Float version, because a custom format function had to be developed to handle Big Float number formatting. In the Big Float version, percent format is not allowed, and literal text is not allowed. Also, in the Big Float version, If the integer part of a number is too large to fit in the specified field width, it is displayed as a string of asterisks to avoid displaying invalid or misleading data.

Limitations

This was never intended to be a mega-project, and so certain compromises had to be made.

The size of the spreadsheet is currently limited to 100 rows and 26 columns.

In the current version, Cut-Copy-Paste functions are limited to the text in the top cell edit field. There is no provision at this time for multiple cell Cut-Copy-Paste, or for inserting and deleting rows or columns. Multi-cell Cut-Copy-Paste functions are not expected to be difficult to implement. Consequently, these will likely be the next features to be added as time permits.

Text data may be entered into any field, but there are no functions available for operating on text values. All operations are mathematical. If cells containing text data are referenced in formulas, the input parser will try to interpret them as numbers. In the Little Float version, the input will be truncated at the first non-numeric character ("12.34hello" will be interpreted as 12.34). In the Big Float version, invalid numbers will be interpreted as zero.

File Operations

The standard File **Open-Close-Save-SaveAs** operations are provided. Spreadsheet documents are saved as CSV text files. Information about the actual data format is provided in the Spreadsheet File Format section below. Multiple spreadsheet documents may be open at any time. Two example files are included in the zip archive: `Pi_Calc_AGM.csv` and `Pi_Calc_ArchimedesMean.csv`.

Spreadsheet Functions

The functions which are available for use are all those included in the ExpressionEvaluator class. The Big Float version of the ExpressionEvaluator also includes all of the functions available in the fpPlugin which take Big Float arguments are return Big Float results.

All operations take floating point arguments and return floating point results. Note that logical operations are **not** bitwise. For logical operations, zero is interpreted as False, and non-zero values are interpreted as True. Operations which return boolean values will return 0 for False, and 1 for True.

The complete list of operators and functions are on the following pages.

Math Operators

```
+, -, *, /, \, ^, mod
```

Note that the backslash is the integer divide operator (as per XOJO standard).

Logical Operators

```
XOR
AND
OR
NOT
```

Comparison Operators

```
==, <>, <, >, <=, >=
```

Note that the equals comparison operator is == rather than XOJO's usual =. This choice was made in order to simplify distinguishing the comparison operator from the assignment operator.

Special Constants

```
EulerGamma      - Euler–Mascheroni constant γ (0.5772156649...)
Pi              - ratio of circumference to diameter of circle π (3.14159265...)
```

Logical Functions

```
If              - If 1st argument is true, returns 2nd argument, otherwise returns 3rd
                  argument (same as XOJO If() function)
```

Summary Functions

These functions take a cell range as argument (e.g., "B7:D9" which specifies the range comprising cells B7, B8, B9, C7, C8, C9, D7, D8, D9). Ranges must be contiguous; discontinuous ranges are not allowed.

```
Sum             - Sum of all values in the cell range
Average         - Average of all values in the cell range
Product         - Product of all values in the cell range
StDev           - Standard deviation of all values in the cell range
RMS             - Root Mean Square of all values in the cell range
Maximum         - *Highest (most positive) value in the cell range
Minimum         - *Lowest (most negative) value in the cell range
```

* Note that the Maximum and Minimum functions take a cell range, while the Max and Min functions (see below) take exactly two non-range cell arguments (e.g., "A6, B5").

Math Functions

These are the same as XOJO functions of same name, except where noted otherwise.

```
abs
acos
acosh          - *hyperbolic arccos function
asin
asinh          - *hyperbolic arcsine function
atan
atan2
atanh          - *hyperbolic arctangent function
besselI        - *Bessel I Function
besselJ        - *Bessel J Function
besselJzero    - *Bessel J zero Function
besselK        - *Bessel K Function
besselY        - *Bessel Y Function
besselYzero    - *Bessel Y zero Function
beta           - *Beta function
ceil
cos
cosh           - *hyperbolic cosine function
ElipE          - Complete Elliptic Integral function E
ElipK          - Complete Elliptic Integral function K
ElipKE         - Combined Complete Elliptic Integral function K minus E
erf            - *Error function
erfc           - *Erfc function
exp
expint         - *Exponential Integral Function
factorial
floor
fresnelC       - *FresnelC function
fresnelS       - *FresnelS function
gamma          - *Gamma function
ibeta          - *Incomplete Beta Function
int            - truncate to integer value (BigFloat version calls fpTrunc function)
kummerm        - *Kummer's confluent hypergeometric Function
lnfactorial    - *Log Factorial Function
log
max
min
round          - *rounds value to nearest integer
sin
sinh           - *hyperbolic sine function
sphBesselJ     - *Spherical Bessel J Function
sphBesselY     - *Spherical Bessel Y Function
sqrt
tan
tanh           - *hyperbolic tangent function
```

* Asterisk indicates functions available only in the BigFloat version. Refer to Robert Delaney's Website and fpPlugin documentation for further information on these functions. Except as noted, the fpPlugin functions that are called have the same name, but with the prefix "fp." For example spreadsheet function fresnelC calls fpPlugin function fpFresnelC.

Spreadsheet File Format

The spreadsheet document files are RFC4180 compliant CSV text. The files written by the Big Float program can be read by the Little Float version, and vice versa, although going from Big Float to Little Float can result in loss of precision of data values.

Except for a couple of special case record types, each record (row) in the file holds the information of one spreadsheet cell. The field designations are as follows:

```
row number, column number, data value, formula, number format string
```

A typical cell data record looks like this:

```
3,1,0.85355339059327376220042218105 2,(B3+C3)/2,"###.00000000000000000000000000000000"
```

Note that the column is identified here by number, not letter. Row and Column numbers are zero based so that the top left cell is 0,0. Thus, in this example, 3,1 refers to cell **B4**. The cell value field, formula field and format field will be enclosed in quotes if the values contain special characters such as embedded commas or quotes. If the cell has no formula assigned, then the formula field is blank. If the cell does have a formula, then the content of the value field is the last calculated value of the formula.

The special records contain data that relate to the overall document or a complete row or complete column. These records have a row number value of -1 to indicate it is a special record type, followed by an arbitrary column number value (typically 0). The third field contains an identifier indicating the specific record type. Currently, there are only two such record types: calculation precision, and column width.

The calculation precision specification record identifier is **$Precision**. The following field is the precision value. A typical precision specification record looks like this:

```
-1,0,$Precision,75
```

In this case it specifies that the spreadsheet calculations are to have a precision of 75 decimal digits. The Little Float version of the spreadsheet program reads this record, but ignores the value, because Little Float precision is always 16 digits.

For the column width specification record, the identifier is **$Cwidth**. The remaining fields in the record contain the column widths, in order, for all of the columns beginning with the first column. A typical column width specification record looks like this:

```
-1,0,$Cwidth,34,115,120,108,320,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75,75
```

Other special record types may be added in the future.

The zip archive of this project includes two example spreadsheet files: `Pi_Calc_AGM.csv` and `Pi_Calc_ArchimedesMean.csv`. Each one demonstrates a different method of calculating the fundamental constant π. Since these are text files, they can be opened in a simple text editor in order to see how the file data is formatted.

Program Code Notes

Part of the motivation for this project was to demonstrate the original ExpressionEvaluator class and the newer Big Float version of the ExpressionEvaluator class, and to give both versions a serious test as part of a reasonably large project. These classes had been previously tested only in fairly simple projects. Another goal of this project was to gain some experience in dealing with complex linked structures.

The spreadsheet project consists of three main components:
- The ExpressionEvaluator class which handles the parsing and evaluation of cell formulas;
- A subclassed listbox which has been customized to behave as a standard spreadsheet cell grid with regards to cell selection and navigation;
- A set of methods and properties, which are part of the main document window, which handle some of the spreadsheet maintenance functions and manage cell formula dependencies (in the form of a linked data structure) to ensure proper cell calculation order.

It was anticipated that some shortcomings would become apparent in the ExpressionEvaluator classes in this larger project, and this was indeed the case—though nothing insurmountable. In the process, several changes were made to the two versions of the ExpressionEvaluator class to make them more easily interchangeable. For example, public functions were added for converting between numeric values and their string representations, so that external code would not need to know the internal details of the numeric data types. Also for improved compatibility, a few methods unique to the Big Float version were added back to the original version, even though they may not perform any useful function in the original version (e.g., SetPrecision and GetPrecision).

A significant difference between the double type numeric value and the Big Float value is in exception handling. With the double type, an invalid operation such as divide by zero will simply result in a NaN (not a number) which propagates along subsequent intermediate calculations and is eventually displayed as "NaN" in the spreadsheet grid. With the Big Float type, NaNs are not generated. Invalid Big Float calculations result in an exception being generated. In order to maintain consistent operation between the two ExpressionEvaluator versions, the Big Float version includes an exception handler and an additional boolean array property varNaN() which tracks invalid results of intermediate calculations, without requiring any additional exception handling. When displaying the final result of a calculation, the ExpressionEvaluator number-to-string conversion methods will return "NaN" if any intermediate calculation result was invalid.

The biggest challenge in developing this software was in handling the dependencies of the cell formulas. To that end, a class **ssNode** was created. Each instance of ssNode represents the contents of one spreadsheet cell which may be either a numeric value or a formula. (Cells which contain numeric or text data that is not referenced by any other cell do not have ssNodes assigned, because they are not required for unreferenced data.) The ssNode class includes two properties nPred() and nSucc() which are links to predecessor nodes and successor nodes respectively. These form a doubly linked nonlinear list which allows keeping track of correct solve order. nPred() and nSucc() are arrays, because each node may have more than one predecessor, and more than one successor. Consequently, the structure of a node list is not necessarily linear or a tree. Branches may diverge and then converge again, in both directions. This results in some interesting issues for efficiently solving some sets of formulas.

An ideal example demonstrating the problem is the AGM Pi calculation. To calculate the AGM (arithmetic-geometric mean), we start with two cells in row 1: **A1** and **B1** each containing a starting value. In the row below this pair is another pair. In row 2, cell **A2** has a formula calculating the arithmetic mean of the pair of cells above:

```
A2 = (A1+B1)/2
```

Cell **B2** has a formula calculating the geometric mean of the pair of cells above:

```
B2 = sqrt(A1*B1)
```

These formula cells are then duplicated downwards, so that the formulas in each row calculate the arithmetic and geometric means of the pair of cells in the row immediately above. The arithmetic and geometric means quickly converge to the same value which is called the arithmetic-geometric mean. This seems simple enough, but if the spreadsheet calculation is done simply by starting at the first calculation cell in row two, and then following the downward links, calculating each node that is encountered, it will be found that, because each formula cell has two successor links, it appears that the number of calculations will double with each row, even though each row has only two calculation cells. This results in multiple recalculations of each cell, and grows exponentially with each additional row, so that just a few rows of these simple calculations could easily take hours to calculate. It is therefore necessary to detect converging branches and avoid redundant recalculations. It was pure luck that the AGM example was chosen for testing, because the exponential growth problem might not otherwise have been discovered until much later.

To resolve this issue, the spreadsheet document maintains an integer property **calcSerialNumber**, which is initialized to zero when the document is created or when an existing file is opened. Its value is incremented whenever the spreadsheet (or part of the spreadsheet) must be recalculated. In addition, each ssNode has an integer property **cSerNo** which defaults to zero when the node is created or when the file is opened. When the content of a cell is changed, the cell's dependent nodes must then be recalculated. The calcSerialNumber value is incremented, and then the node links are traversed downwards, with the formulas recalculated as follows:

First, the node's cSerNo value is compared to calcSerialNumber value. If the values are not equal, then the cell is calculated, and cSerNo is set to the value of calcSerialNumber. However, if it is found that the value of cSerNo is already equal to the value of calcSerialNumber, then this node has already been calculated, and this branch of node traversal can be aborted at this point. If the node has not been previously calculated, then the successor node(s) links can be followed to continue the dependent calculations. This ensures that the branches are pruned as necessary. An additional constraint is that the calculations must also be done in a certain order to ensure that, before a node is calculated, all of its predecessors have already been calculated. This is facilitated with an additional integer property in the ssNode called **solveOrder**. There is a single top node of the spreadsheet which is a parent to all other nodes, and has a solveOrder value of zero. Each subsequent descendent node has a solveOrder value of one greater than the maximum solveOrder values of all of the node's immediate predecessors. When traversing down the branches, the solveOrder values of the immediate successor nodes are examined to determine which branch to take and when to suspend a branch traversal if necessary.

Admittedly, this description of the calculation sequence is very cursory. This is primarily because the logic is currently under review, and due for imminent revision to improve calculation efficiency.

To aid in debugging there is a separate window called dbugWindow containing a text area where diagnostic messages are logged during various operations. To use it, simply set it's visible property to True. When this is done, a button labelled "Nodes" will also appear in the document windows. Clicking the Nodes button will cause the current node structure information to be sent to the debug window.