Expanding Spreadsheet Calculations

Jozef Hvorecky

Higher Colleges of Technology
Abu Dhabi, United Arab Emirates
&
College of Management
Bratislava, Slovakia
Jozef.Hvorecky@hct.ac.ae

Ivan Trencansky

University of Economics
Bratislava, Slovakia
ivo@euba.sk

Abstract

Inserting a cell between existing ones belongs among elementary spreadsheet operations. The newly inserted cell is empty. One can specify its content by introducing a constant or a formula into it. The formula usually expresses the cell value as a function of values of its neighbors. When all the referred values are known, the formula is calculated and its result is displayed. Since then, it can be used in next calculations.

In the calculations described in this paper, single cells (or entire rows or entire columns or their combinations, depending on the aim) and their contents are inserted by macros. The values evaluated in the previous steps serve as a basis for next ones. The range of spreadsheet used for the calculation spreads automatically, giving a new esprit to the notion of "spreadsheet".

The same method can be used for many purposes. Here we apply it to drawing graphs of one- and two-dimensional functions and of fractals. The expansion generates graphs with higher and higher precision.

If spreadsheets would have an infinite size, the calculation could spread indefinitely. In reality, there are limits that are also demonstrated in our contribution.

1 Basic Notions

Spreadsheet programs are tools for (pseudo) parallel computations over large groups of numbers. The numbers are located in a ***sheet*** – a matrix with ***rows*** and ***columns***. The elements of the matrix are called ***cells***.

Theoretically, the numbers of rows and columns are both unlimited. Naturally, each *real* software product has these numbers exactly specified. As we show below this fact has an important impact on some computations.

Spreadsheet calculations use various operations. The most frequent ones are [1]:

- ***Initialization*** of cells, i.e., assigning initial values into them.

- ***Definition of relationships*** between the initialized cells and selected ones.
- ***Spreading*** the relationships throughout the sheet.

The range occupied by such calculations is usually fixed. However, there are operations allowing its expansion. The one discussed in our paper is the ***insertion***.

The application of an insertion generates a new cell in the sheet. Naturally, the specification of its position by pointing into a place in a spreadsheet is ambiguous. Programmers must also indicate whether the new cell belongs to the particular column or to the row. Only then, the existing cells are shifted in the required direction creating a vacant cell. (Note that insertions do not harm the contents of already existing cells.) In a similar way, one can insert entire columns and rows. Again, the existing cells move to the required direction and their contents remain unchanged.

As newly inserted cells are empty, their content should be specified by introducing a constant or a formula into them. As usual, the formula expresses the cell value as a function of values of its close neighbors. If the values of the referred cells are known, the formula is calculated and its result is displayed[1]. The resulting value can instantly be used in following calculations.

Below, we demonstrate drawing graphs using series of insertions. In the beginning, just a few cells are initialized so the graph does not display the function properly. To improve its shape, we use macros to insert new cells between existing ones. In each step at least one new point is inserted. As the result, the graph quality constantly improves. The process is repeated until the precision is sufficient or the capacity of the spreadsheet program is exceeded. The method can in fact be used in any spreadsheet environment. Our examples have been made in Microsoft Excel.

2 Inserting Cells

We exemplify using the insertion of individual cells on improved drawings of a one-dimensional function.

In the „standard method“ of representing one-dimensional functions, the values of the *x* variable are stored in one column, the function values in the next one. As the number of cells in both columns is identical, the discussed range always forms a rectangle with a varying number of rows and two columns. Its minimum size is 2 x 2 as we need at least two values to specify the interval in which the function is drawn.

The program starts with specifying the (two or more) initial values and the function. Then, it spreads automatically. In each stage, pairs of cells (in the same row) are inserted between any two existing rows inside the range. The following formulas are inserted into them:

1. The value of the independent variable (i.e. that one in the left column) is determined as the middle of the interval specified by its neighbors in the same column:

```
Selection.Formula = "= (r[-1]c[0] + r[1]c[0])/2"
```

2. The drawn function is then copied from its (upper or lower) neighbor:

```
Cells(i+1, 2).Select
Selection.Copy
Cells(i, 2).Select
```

[1] If the formula refers to cells with unspecified values or to cells with wrong data type, an error message is presented.

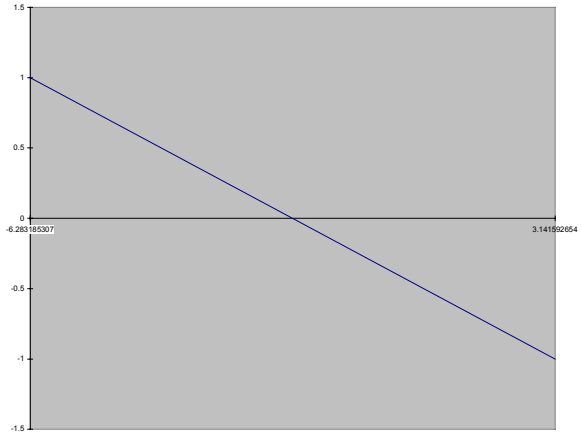

Figure 1 a. *The cos(x) function defined in 2 points*

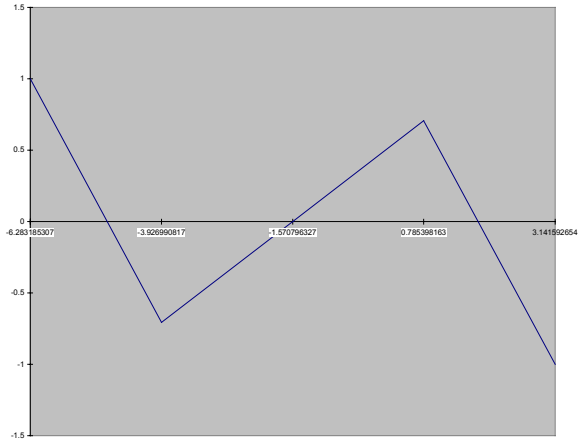

Figure 1 b. *The same function defined in 5 points*

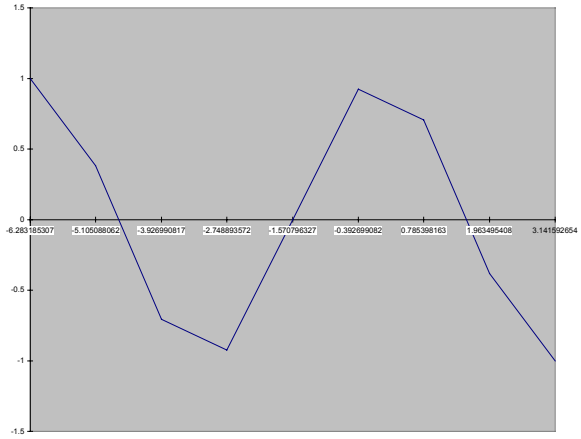

Figure 1 c. *The same function defined in 9 points*

```
ActiveSheet.Paste
```

(Here the formula is copied from the lower neighbor).

Note that each cell must be individually selected (as a 1 x 1 range) before the formula insertion.

The order of the insertions is important, because all values must be specified before their use:

- The values of the neighboring cells in the first column allow computing the middle one.

- When this new value is known, its functional value can safely be calculated.

The cells are always shifted down. To make the loop simple and efficient, we start inserting from the bottom of the current range. The Visual Basic loop has the form

```
for i = No_Of_Existing_Rows to 2 step -1
  Range(Cells(i,1), Cells(i,2)).Select
  Selection.Insert shift:=xlDown
  Cells(i, 1).Select
  Selection.FormulaR1C1 = "=(r[-1]c[0] + r[+1]c[0])/2"
  Cells(i+1, 2).Select
  Selection.Copy
  Cells(i, 2).Select
  ActiveSheet.Paste
  No_Of_Existing_Rows = No_Of_Existing_Rows + 1
next i
```

Inside the loop, the new lines are inserted before the active (i.e., *i*-th) row, because the value of the independent variable can only be computed when its preceding and following values are known. Thus, the last insertion is done for the second row. Each loop increases the number of existing rows by one[2]. When starting with 2 rows, their number grows as

$$2, 3, 5, 9, 17, ..., 2^n + 1, ...$$

where *n* is the number of complete executions of the above loop. Selected first steps of the approximation of cos(x) in the interval $(-2\pi, \pi)$ are shown in Figure 1.

3 Inserting Columns and Rows

A similar approach can be used for drawing two-dimensional functions. The main difference is that we now insert rows and columns instead of individual cells.

	A	B	C
1		y_1	y_2
2	x_1	$F(x_1, y_1)$	$F(x_1, y_2)$
3	x_2	$F(x_2, y_1)$	$F(x_2, y_2)$

Figure 2 *Positioning first four function values*

The calculation starts in a 3 x 3 range. The first row and the first column are constantly occupied by the values of independent variables *x* and *y*. So, the active (expanding) area used for the function F(x, y) goes from the row 2 to *No_Of_Existing_Rows* + 1 and from the column 2 to *No_Of_Existing_Columns* + 1. Figure 2 shows the location of the values in the simplest case. The factual numbers of existing rows and columns of the expansion area are both equal to two.

The expansion is executed in a similar way:

---

[2] The change of the *No_Of_Existing_Rows* variable (the last command of the loop body) does not affect the number of the loop repetitions. Its value is used in calculating the loop control variable only once – during its initialization. So, it can be safely changed inside the loop and used again when the loop is entered for the next time.

```
for i = No_Of_Existing_Rows + 1 to 3 step -1
      Rows(i).Select
      Selection.Insert shift:=xlDown
      Cells(i, 1).Select
      Selection.FormulaR1C1 = "=(r[-1]c[0] + r[+1]c[0])/2"

      Cells(2, 2).Select
      Selection.Copy
      Range(Cells(i, 2), Cells(i, No_Of_Existing_Columns + 1)).Select
      ActiveSheet.Paste

      No_Of_Existing_Rows  = No_Of_Existing_Rows + 1
next i
for j = No_Of_Existing_Columns + 1 to 3 step -1
      Columns(j).Select
      Selection.Insert shift:=xlRight
      Cells(1, j).Select
      Selection.FormulaR1C1 = "=(r[0]c[-1] + r[0]c[+1])/2"

      Cells(2, 2).Select
      Selection.Copy
      Range(Cells(2, j), Cells(No_Of_Existing_Rows + 1, j)).Select
      ActiveSheet.Paste

      No_Of_Existing_Columns = No_Of_Existing_Columns + 1
next j
```

The both sides of the expanded area grow with the same speed

$$2, 3, 5, 9, 17, ..., 2^n + 1, ...$$

So, the consecutive areas occupy

$$2^2, 3^2, 5^2, 9^2, 17^2, ..., (2^n + 1)^2, ...$$

sheet cells. The first four approximations of the graph of the cos(x).sin(x) function for $x \in [-\pi, \pi]$ and $y \in [-\pi, \pi]$ are shown in Figure 3.

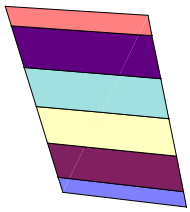

Figure 3 a. *The* cos(x)sin(y) *function defined in 4 points*

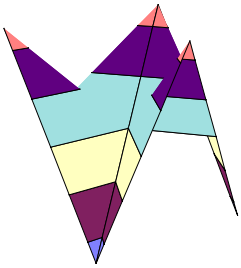

Figure 3 b. *The* cos(x)sin(y) *function defined in 9 points*

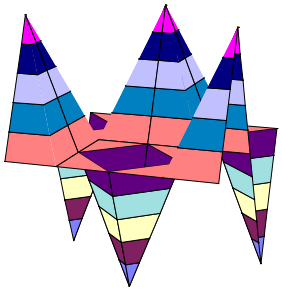

Figure 3c *The* cos(x)sin(y) *function defined in 25 points*

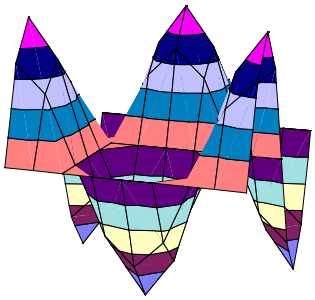

Figure 3d *The* cos(x)sin(y) *function defined in 49 points*

4 Complex expansions

The previous expansions have been simple:

1. The formulas inserted to the new cells are identical.
2. Standard methods of graph presentation have been used: A vector holds the functional values of a one-dimensional function, a rectangular array is used for a two-dimensional one.
3. The insertion points are evenly distributed – always in the middle of the interval.

As a result of this simplicity, identical approximations can be used for each (and any) one- and two-dimensional function in any specified interval. Now we demonstrate a more complex example. We will create fractal curves consisting of linear sections.

The number of points inserted between existing pairs is one of fractal's properties. Also, the formulas describing individual linear sections may be different. As a result, the fractal descriptions differ from case to case. On the other hand, there is one important similarity with the previous cases: As we are drawing curves, we use their „vector representation" and insert groups of cells.

In the below example we create Pythagoras Tree – a fractal based on his well/known theorem. The fractal starts with a square. In the first expansion a right-angle triangle is placed on its top side with two smaller squares on its sides containing the right angle – Figure 4a. This implies that:

a) The new points of the curve are not inserted between each existing pair. Only the one side of each square is expanded.
b) An entire pattern (with eight new points) is then inserted.

To simplify the insertion we keep the „shape of the fractal" in a separate sheet named *Pattern*. Each of the ending points of its linear sections is encoded as a triple: its *x* and *y* coordinates plus the information whether or not to insert new points before it[3].

The critical section of the program has the form of the nested loop. The outer loop expands the existing fractal („point by point"). If a new complete element of the fractal is to be inserted, the inner loop executes that. The lengths of its sides are then calculated from the pattern and positions of the ending point of the line that is replaced by the fractal.

```
for j = No_Of_Exist_Cells to 2 step -1
 if Cells(j, 3) <> -1 then
  Range(Cells(j, 1), Cells(j + Fractal_Size - 1, 3)).Select
  Selection.Insert shift:=xlDown
  for i = 1 to Fractal_Size
   Cells(j + i - 1, 1).Select
   Selection.FormulaR1C1 = "=r[-" + CStr(i) + "]c+(r[" +
    CStr(Fractal_Size - i + 1) + "]c-r[-" + CStr(i) + "]c)*"
    + Pattern + "!r" + CStr(i) + "c1-(r[" + CStr(Fractal_Size - i + 1)
    + "]c[1]-r[-" + CStr(i) + "]c[1])*" + Pattern + "!r" + CStr(i) +
    "c2"
   Cells(j + i - 1, 2).Select
   Selection.FormulaR1C1 = "=r[-" + CStr(i) + "]c+(r[" +
```

[3] For reading the program is necessary to know that zero means „to insert", minus one „not to insert".

```
      CStr(Fractal_Size - i + 1) + "]c-r[-" + CStr(i) + "]c)*"
      + Pattern + "!r" + CStr(i) + "c1+(r[" + CStr(Fractal_Size - i + 1)
      + "]c[-1]-r[-" + CStr(i) + "]c[-1])*" + Pattern + "!r" + CStr(i) +
      "c2"
     Cells(j + i - 1, 3).Select
     Selection.FormulaR1C1 = "=" + Pattern + "!r" + CStr(i) + "c3"
   next i
   Cells(j + Fractal_Size, 3).Select
   Selection.FormulaR1C1 = "=" + Pattern + "!r" + CStr(i) + "c3"
   No_Of_Exist_Cells = No_Of_Exist_Cells + Fractal_Size
  end if
next j
```

The number of rows occupied by the fractals can be determined in the following way:

a) Five points (occupying five rows) define the original square.

b) During each expansion we add two squares at the top side of each square that is not covered by a triangle. Each modification requires computing of four of its points (the fifth is identical with one of existing ones). Because all existing lines remain and some new are added, the number of points forming the curve grows

   5

   $5 + (2 \times 4) = 5 + 2^3$

   $5 + (2 \times 4) + (2^2 \times 4) = 5 + 2^3 + 2^4$

   $5 + (2 \times 4) + (2^2 \times 4) + (2^3 \times 4) = 5 + 2^3 + 2^4 + 2^5$

   …

   $5 + 2^3 + 2^4 + 2^5 + \ldots + 2^{n+2} = 5 + 2^3 \times (1 + 2 + 2^2 + 2^3 + \ldots + 2^{n-2}) =$

   $= 5 + 2^3 \times (2^{n-1} - 1) \approx 2^{n+2}$

Discussion

In each of the above cases, the growth of the area used for the calculation is exponential. It implies that the range in which the calculation is executed expands extremely quickly. As the professional spreadsheet products contain much less columns than rows in their sheets (e.g., Microsoft Excel has 256 columns and 16 384 rows [2]), the width of the sheet is quickly exceeded. As one can derive from the formula in the end of Section 2, no more than seven approximations of two-dimensional functions can be completed. As a result, quality drawing is sufficient only for smooth functions.

On the other hand, this example nicely demonstrates limits of numerical mathematics. Real calculations must be done in real environments. Not all environments are equally apt for all calculations. Similar analyses can explain and exemplify differences between them to students.

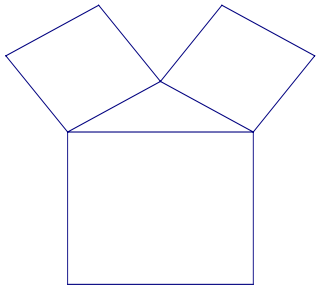

Figure 4 a. *Pythagoras Tree after its first expansion*

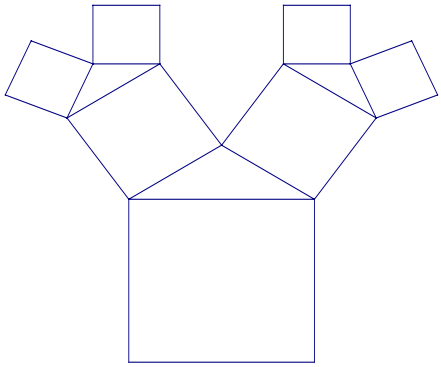

Figure 4 b. *Pythagoras Tree after its second expansion*

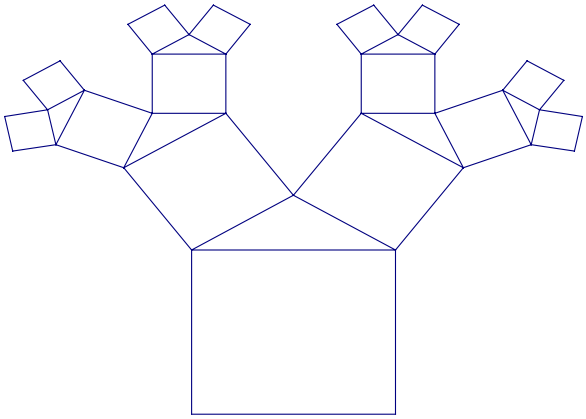

Figure 4 c. *Pythagoras Tree after its third expansion*

References

1. J. Hvorecky, I. Trencansky: **Recursive Computations in Spreadsheets**. Wei-Chi Yang, Kiyoshi Shirayanagi, Sung-Chi Chu, Gary Fitz-Gerald (Editors): *Proceedings of the Third Asian Technology Conference in Mathematics*. Springer, Tokyo, 1998, pp. 290-298.
2. **Microsoft Excel User's Guide**. Microsoft Press, 1998
3. Sivertsen, P.: **L'usage d'un tableur dans l'enseignement des mathematiques**. Zborník bratislavského seminára z teórie vyucovania matematiky, Bratislava 1999, pp. 91-99