DOI: 10.47743/eufire-2025-1-12

ADVANCED DATA MODELING IN BUSINESS USING DYNAMIC ARRAYS AND TRIPLE MATRICES

VALERICĂ GREAVU-ȘERBAN
Alexandru Ioan Cuza University of Iași
Iași, Romania
valy.greavu@outlook.com

VASILE-DANIEL PĂVĂLOAIA
Alexandru Ioan Cuza University of Iași
Iași, Romania
danpav@uaic.ro

Abstract
Recent developments in Microsoft Excel, particularly the introduction of Dynamic Arrays, have significantly enhanced the capabilities of this ubiquitous tool in financial and accounting analysis. This paper explores the application of modern Excel functions such as MAP(), LAMBDA(), SEQUENCE(), MAKEARRAY(), and LET() in automating complex tasks, minimizing formula redundancy, and enabling advanced modelling without VBA scripting. A structured application of these functions is demonstrated through real-life accounting and finance use cases, including budget projections, data reconciliation, scenario modelling, and error detection. Additionally, a novel methodology termed the "Triple Matrix" is introduced. Inspired by triple FOR loop logic, this approach enables multidimensional evaluation and aggregation directly within Excel using only native functions. The method systematically generates and processes combinations of values across three axes, allowing the implementation of iterative logic and scenario simulations without macros. This approach is tested in various use cases and evaluated based on automation efficiency, clarity of formulas, performance, and adaptability. Results highlight the added value of combining Dynamic Arrays and Triple Matrix logic for accountants and financial analysts, improving automation, accuracy, and process scalability using only standard Excel functionality.
Keywords: *Dynamic Arrays; Triple Matrix; financial modelling; spreadsheet analysis; no-code solutions.*
JEL Classification: M15; G19; H75.

1. INTRODUCTION

Functional programming might sound like something reserved for hardcore developers, but at its core, it's just a way of solving problems by using functions, small, reusable pieces of logic. Traditionally, this approach has been more

common in academic or data science environments than in everyday business tools. But that's starting to change, especially with recent updates to tools we all know, like Microsoft Excel (Greavu, 2024).

With the launch of Dynamic Arrays (Bartholomew, 2020) and Lambda functions in Excel's cloud version, functional programming concepts are now available right inside the spreadsheet. That means users can build powerful, flexible models without writing a single line of VBA code. Even better, these functions make it easier to collaborate in real time on the same file, something that's becoming the norm in today's hybrid workplaces. These features have also been added to Google Sheets, making the two tools more compatible than ever.

While functional programming (Bartholomew, 2023) isn't the most common approach in commercial software, it's become increasingly important in areas like big data, machine learning, and AI. Languages like R, known for their data crunching power, are now widely used in both universities and businesses. Microsoft, too, has gotten in on the action with Power Query M, a language designed for importing and transforming data in Excel and Power BI. Then there's Python, which supports multiple ways of thinking, including functional techniques like LAMBDA, MAP(), AND FILTER().

In this article, the authors walk through a few classic algorithmic challenges – like sorting, counting combinations, and making smart decisions using greedy algorithms (García, 2025) and show how they can be solved using modern Excel tools.

The goal is to show how far spreadsheet tools have come, and how one can apply powerful programming logic, often associated with advanced programming languages, directly in Excel, for Financial Modelling (Bastick, 2024). No macros, no scripts, just smart use of what's already built in.

2. STUDY CASES OF ADVANCED MODELLING IN BUSINESS

Specifically, this article will display the following subjects from the financial sector by using functional programming:

Sorting problems like MaxProductOfThree and Triangle

The Caterpillar method, applied to the CountTriangles problem

Greedy strategies, using MaxNonoverlappingSegments as an example

2.1. Sorting Algorithms in Functional Spreadsheets: The MaxProductOfThree Problem

MaxProductOfThree problem consists in identifying the maximum product obtainable by multiplying any three values from a given array of real numbers. Mathematically, given a finite list of real values.

R={r1,r2,...,rn}, the goal is to compute:

$$\max_{i<j<k}(r_i \cdot r_j \cdot r_k)$$

This is a classical sorting-based algorithmic problem often used to test numerical reasoning, array manipulation, and computational efficiency.

The use case in Finance that we want to use is Portfolio Growth Potential. Thus, in financial modelling, this problem is directly applicable to portfolio optimization scenarios. Given a list of estimated relative returns for various assets, the objective is to identify a triplet of assets whose combined multiplicative return yields the highest potential compounded growth. This serves as a simple model for identifying positively correlated high-growth combinations in investment portfolios.

Let R={1.12,0.95,1.25,1.08,1.30,0.90,1.15}, representing expected returns for seven financial instruments. We seek to determine the triplet (r_i, r_j, r_k) such that $r_i \cdot r_j \cdot r_k$ is maximized.

Modern versions of Microsoft Excel now support Dynamic Arrays (Bartholomew, 2020) and Lambda functions, which allow this problem to be solved without scripting (e.g., VBA). The following is an example implementation leveraging SEQUENCE, INDEX, LET, LAMBDA, MAKEARRAY, and BYROW.

The solution is based on the following three steps:

Step 1: Store the relative returns in a named range, e.g., Returns referring to the range B2:B8.

Step 2: Generate all combinations of three different elements. This can be done using a custom lambda function:

```
=LET(
  n, ROWS(Returns),
  combos, MAKEARRAY(n^3, 3, LAMBDA(row, col,
    INDEX(Returns,
      SWITCH(col,
        1, INT((row-1)/(n*n))+1,
        2, INT(MOD((row-1), n*n)/n)+1,
        3, MOD(row-1, n)+1
      )
    )
  )),
  filtered, FILTER(combos,
    (INDEX(combos,,1) <> INDEX(combos,,2)) *
    (INDEX(combos,,1) <> INDEX(combos,,3)) *
    (INDEX(combos,,2) <> INDEX(combos,,3))
  ),
  BYROW(filtered, LAMBDA(row, PRODUCT(INDEX(row, 1), INDEX(row, 2), INDEX(row, 3))))
)
```

The above computes all valid 3-element combinations, excludes duplicates, and calculates their product.

Step 3: Extract the maximum product:

```
=MAX(
  LET(
    ...
    BYROW(filtered, LAMBDA(row, PRODUCT(INDEX(row, 1), INDEX(row, 2), INDEX(row, 3))))
  )
)
```

This approach allows financial analysts to compute potential growth-maximizing asset triplets directly within Excel, avoiding both iterative logic and scripting. It exemplifies how algorithmic logic – traditionally requiring code – can be executed within a functional spreadsheet environment using native tools.

Such an application is useful in:

- Portfolio selection, where combinations of assets are evaluated for compounding effects.
- Stress testing scenarios, simulating various groupings of high/low performance instruments.
- Heuristic modelling, as a basis for more complex Monte Carlo simulations or optimization models.

By applying a classic sorting problem through Excel's new Dynamic Array (Rubin, 2024) capabilities, the solution is both transparent and auditable, supporting financial compliance and collaborative workflows.

2.2. The Caterpillar Method in spreadsheets: the CountTriangles problem

The *CountTriangles* problem involves determining how many unique combinations of three values from a given array can form a valid triangle. Formally, for a sorted array of positive real numbers A = $\{a_1, a_2, ..., a_n\}$, a triplet (a_i, a_j, a_k), with i<j<k, can form a triangle if and only if $a_i + a_j > a_k$.

The above is a classic algorithmic problem often solved using the Caterpillar Method, an efficient technique based on two-pointer traversal over sorted arrays. The method avoids brute-force enumeration by leveraging the sorted order to incrementally evaluate only feasible combinations.

In a business context, the CountTriangles logic can be applied to evaluate the stability of multilateral partnerships, such as trade alliances between countries or corporations. The triangle condition can be interpreted as a constraint ensuring balance or mutual reinforcement between the parties involved.

Given a list of countries and their respective export volumes, we want to determine how many combinations of three countries could form sustainable trade

partnerships, based on the assumption that the combined export of any two members must exceed the third. This models economic synergy, where no single participant dominates or destabilizes the partnership.

Such modelling can help:

- Assess the feasibility of trilateral economic agreements
- Design balanced regional trade blocks
- Evaluate cooperation strength in supply chain networks or international coalitions

Modern Excel does not support explicit pointer-based loops as in traditional programming. However, the Caterpillar logic can be implemented functionally by:

- Sorting the input array (to leverage the triangle inequality)
- Generating all possible combinations of three elements
- Filtering combinations that satisfy the triangle condition

Let's assume the input list of export volumes is in a named range Exports, defined as A2:A10.

Step 1: Sort the array

= SORT(Exports)

Step 2: Generate all 3-combinations

It is recommended to us MAKEARRAY() to simulate nested loops, then FILTER to exclude duplicates and apply the triangle condition.

```
= LET(
  arr, SORT(Exports),
  n, ROWS(arr),
  combos, MAKEARRAY(n^3, 3, LAMBDA(row, col,
    INDEX(arr,
      SWITCH(col,
        1, INT((row-1)/(n*n))+1,
        2, INT(MOD((row-1), n*n)/n)+1,
        3, MOD(row-1, n)+1
      )
    )
  )),
  valid, FILTER(combos,
    (INDEX(combos,,1) < INDEX(combos,,2)) *
    (INDEX(combos,,2) < INDEX(combos,,3)) *
    (INDEX(combos,,1) + INDEX(combos,,2) > INDEX(combos,,3))
  ),
  ROWS(valid)
)
```

The above expression solves the initial problem as it enumerates all possible triplets (1), ensures $i<j<k$ (2), applies the triangle condition $a_i+a_j>a_k$ (3) and returns the count of valid combinations (4).

Through this approach, Excel becomes a platform for network viability analysis, leveraging functional programming logic to process combinations and validate constraints.

This method can be applied in:

- Geopolitical simulations involving trilateral cooperation agreements
- Corporate alliance analysis, determining stable joint ventures
- Supply chain planning, ensuring balance in distributed production or trade volumes

The elegance of the Caterpillar method lies in its efficiency – traditionally $O(n^2)$ as opposed to $O(n^3)$ and its suitability for spreadsheet environments when adapted with dynamic array logic.

To conclude, by adapting the CountTriangles problem using modern Excel features like LAMBDA, LET, MAKEARRAY, and FILTER, we demonstrate how functional programming patterns can be applied in financial and economic modelling. This strengthens the role of Excel as a lightweight, transparent, and powerful tool for computational tasks traditionally reserved for programming environments.

2.3. Greedy algorithms in functional spreadsheets: the MaxNonoverlappingSegments problem

The *MaxNonoverlappingSegments* problem seeks to determine the maximum number of non-overlapping intervals that can be selected from a given set of segments on a one-dimensional timeline. Formally, given a set of segments defined by pairs (a_i,b_i), where $a_i \leq b_i$, the goal is to select the largest subset of segments such that no two overlap.

This is a classic greedy algorithm problem, typically solved by applying the following steps:

1. Sorting the segments by their end time b_i
2. Iteratively selecting the next available segment that starts after or at the end of the previously selected one

The greedy (García, 2025) approach is optimal for this specific problem because choosing the segment that ends earliest leaves the maximum possible room for future selections.

In real-world financial planning, organizations often face temporally constrained opportunities, such as funding rounds, market openings, tax benefit windows, or budget cycles. Capital and risk constraints frequently prevent overlapping investments. Therefore, selecting the maximum number of non-

overlapping investments allows organizations to maximize capital deployment without exceeding risk or resource boundaries.

Suppose we have a list of investment opportunities, each active during a specific period (start_date,end_date). The objective is to choose as many investments as possible without overlapping periods, thus avoiding simultaneous risk exposure or budgetary strain. This model can also be applied in:

- Multi-year capital project planning
- Allocation of resources in public budgets
- Scheduling of grants or R&D initiatives with temporal dependencies

While greedy algorithms (Greavu, 2025) are traditionally implemented using loops, ***Excel's Dynamic Array*** and ***Lambda functions*** allow for a functional emulation of the greedy selection logic.

Let us assume the following input in Excel as seen in Figure 1 where StartDates and EndDates are defined as named ranges.

	A	B	C
1	**Investment**	**Start Date**	**End Date**
2	A	01.01.2025	15.03.2025
3	B	01.02.2025	30.04.2025
4	C	01.05.2025	31.07.2025
5	D	20.03.2025	15.05.2025
6	E	01.08.2025	15.10.2025

Figure 1. The model to be solved

The solution involves applying the following three steps.

Step 1: Sort intervals by end date which arrange the segments in ascending order of their end date, an essential step in greedy selection.

=SORTBY(A2:C6, C2:C6, 1)

Step 2: Greedy selection using functional filtering

While Excel lacks native iterative constructs, we can simulate the greedy selection using a recursive LAMBDA with helper columns or named recursive Lambdas (e.g., using SCAN or REDUCE in Office 365).

However, a practical and auditable approach is to create a helper column with the earliest available time to select a segment.

```
=LET(
  sorted, SORTBY(A2:C6, C2:C6, 1),
  startList, INDEX(sorted,,2),
  endList, INDEX(sorted,,3),
  result, SCAN(DATE(1900,1,1),
    SEQUENCE(ROWS(sorted)),
    LAMBDA(acc, i,
      IF(INDEX(startList,i)>=acc,
        INDEX(endList,i),
        acc
      )
    )
  )
)
```

This creates a cumulative selection where each interval is chosen only if its start is after the end of the last accepted segment.

To count how many intervals were selected, we can use:

```
=LET(
  sorted, SORTBY(A2:C6, C2:C6, 1),
  startList, INDEX(sorted,,2),
  endList, INDEX(sorted,,3),
  selected, SCAN(DATE(1900,1,1),
    SEQUENCE(ROWS(sorted)),
    LAMBDA(acc, i,
      IF(INDEX(startList,i)>=acc,
        INDEX(endList,i),
        acc
      )
    )
  ),
  COUNTUNIQUE(selected) - 1
)
```

In the above, the -1 excludes the initial dummy value.

By applying greedy logic in a functional way, this model provides an efficient, transparent means to plan investment sequences within Excel. It enables analysts to:

- Maximize the utilization of non-overlapping investment opportunities
- Design risk-aware capital deployment calendars
- Improve decision-making in contexts with time-sensitive constraints

This approach also supports policy-level planning – such as government spending or institutional grants – where no two commitments should occur simultaneously due to compliance or logistical limitations.

To conclude, the *MaxNonoverlappingSegments* problem illustrates how a well-known greedy algorithm can be effectively implemented in Excel using pure functional constructs, without relying on procedural scripting or macros. This enables professionals in finance and economics to model and solve time-constrained optimization problems using familiar spreadsheet environments – enhancing both accessibility and auditability.

3. CONCLUSION, LIMITATIONS AND FUTURE PATHS

While the adoption of functional programming in Excel introduces new modelling capabilities, it also exposes several limitations. Complex formulas based on Dynamic Arrays and Lambda functions can lead to performance issues, especially when processing large datasets or generating numerous combinations in many domains including Financial Modelling (Rubin, 2024). Excel is not inherently optimized for iterative or recursive logic, and although functions like SCAN and REDUCE offer partial solutions, they lack the full flexibility of traditional control structures.

Another challenge lies in readability and maintainability. As models grow in complexity, the nested and abstract nature of functional formulas can hinder understanding, especially for users unfamiliar with this paradigm. Additionally, Excel offers limited tools for debugging or tracing functional logic, which can complicate development and auditing processes.

Despite these constraints, the article demonstrates that classical algorithmic problems can be effectively translated into Excel using purely native functions. This expands Excel's role from a data analysis tool to a platform capable of lightweight, functional computation suitable for financial and economic modelling. Future work should explore hybrid models integrating Power Query and Python, as well as ways to simplify and document functional workflows to make them more accessible to non-programmers.

In conclusion, functional programming in Excel is a promising direction for building transparent, flexible, and auditable models, especially in business contexts where specialized programming tools are not always available.

References

1) Bastick, L. (2024) *Financial Modelling Using Dynamic Arrays: Let Lambdas Extend Your Range*. Packt Publishing Ltd, London. ISBN: 978-1-83702-318-9. [online] Available at https://www.amazon.com/Financial-Modelling-using-Dynamic-Arrays/dp/1615470875 [Accessed 30.05.2025].
2) Bartholomew, P. (2020). *Will Dynamic Arrays finally change the way Models are built?*. arXiv preprint arXiv:2006.14706.

3) Bartholomew, P. (2023). *Excel as a turing-complete functional programming environment*. arXiv preprint arXiv:2309.00115.
4) García, A. (2025). Greedy algorithms: a review and open problems. *Journal of Inequalities and Applications,* 2025, Article 11. https://doi.org/10.1186/s13660-025-03254-1.
5) Greavu, V. (2024) *Programarea funcțională în Excel Modern*. [online] Available at: https://pinmagazine.ro/programarea-functionala-in-excel-modern/ [Accessed 30.05.2025].
6) Greavu, V. (2025) *Modele de algoritmi în #Excel – Greedy algorithms (16)*, *Valy Greavu – Live Blog*, 16 februarie. [online] Available at: https://valygreavu.com/2025/02/16/modele-de-algoritmi-in-excel-greedy-algorithms-16/ [Accessed 30.05.2025].
7) Rubin, L. (2024) *How Dynamic Arrays will revolutionize Financial Modelling?*, EXL Cloud, 1 June. [online] Available at: https://www.exlcloud.io/articles/how-dynamic-arrays-are-revolutionising-financial-modelling?utm_source=chatgpt.com [Accessed 30.05.2025].