

# How to Calculate SUMPRODUCT Excluding Negative Values in Excel

Authored by  
**stats writer**

December 1, 2025

## RECOMMENDED CITATION

stats writer (2025). *How to Calculate SUMPRODUCT Excluding Negative Values in Excel*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=102903>

The SUMPRODUCT function in Excel is an immensely powerful tool, often overlooked, that calculates the sum of the products of corresponding items in supplied arrays. While its primary function is simple multiplication and summation across ranges, its true utility emerges when combined with conditional logic. This methodology allows users to perform highly granular calculations, effectively filtering data before it contributes to the final product sum. This article will detail the specific technique required to utilize SUMPRODUCT exclusively with values that meet a specific criterion--specifically, those greater than zero--thereby optimizing data processing and ensuring accuracy when dealing with mixed datasets containing positive, negative, and zero values.

## Understanding the Power of SUMPRODUCT

The core purpose of the SUMPRODUCT function is to efficiently handle complex calculations that would otherwise require multiple columns of intermediate steps. Imagine needing to multiply Column A by Column B, and then summing up all those individual products; Excel's SUMPRODUCT accomplishes this in a single cell, streamlining your spreadsheet and improving computational efficiency. However, in real-world financial or statistical modeling, data often includes entries that should be ignored, such as negative adjustments, zero transactions, or invalid measurements. This necessitates a method to incorporate dynamic filtering directly into the calculation engine of the function itself, ensuring that only relevant data contributes to the final aggregated result.

Traditional usage involves listing the ranges (or arrays) that need to be multiplied together. For instance, `=SUMPRODUCT(A1:A10, B1:B10, C1:C10)` would multiply  $A1*B1*C1$ ,  $A2*B2*C2$ , and so on, before summing up the entire series of products. When negative values or zeros exist in the input ranges, they inherently influence the outcome, which may be undesirable if the objective is to calculate totals based only on positive contributions, such as sales volumes or measurable output. Therefore, mastering conditional application of this function is paramount for sophisticated data analysis and manipulation within Excel.

The challenge lies in integrating a logical test--a condition like "is the value greater than zero?"--into a function designed primarily for arithmetic operations. Standard Excel functions often require helper columns or nesting within complex array formulas (like `CTRL+SHIFT+ENTER` combinations) to achieve conditional summation. The elegance of the conditional SUMPRODUCT technique is that it bypasses these complexities entirely, allowing the filtering criteria to be defined internally, thus producing a concise, single-cell formula that is both powerful and highly readable, provided one understands the mechanism of type coercion inherent in its structure.

## The Mechanics of SUMPRODUCT: Default Behavior

By default, the SUMPRODUCT function operates on a multiplication principle across all elements in the supplied arrays. If your dataset includes negative numbers, those negative numbers will actively contribute to reducing the overall final sum, potentially skewing results when the intention is to aggregate only positive activity. For instance, if Column A represents units sold and Column B represents price, a negative value in Column A (perhaps due to returns) multiplied by the price in Column B will yield a negative product, reducing the total revenue calculation. While mathematically correct in a standard accounting context, this is often not what is desired when calculating performance metrics that only focus on growth or positive contributions.

The **SUMPRODUCT** function in Excel returns the sum of the products of two corresponding arrays.

The standard syntax simply lists the arrays separated by commas. However, when we introduce a conditional test, we must modify how Excel processes the resulting TRUE/FALSE outputs. If we simply include a logical test like `(A1:A9>0)` within the SUMPRODUCT function without any manipulation, the function will attempt to multiply this Boolean array (containing TRUES and FALSEs) directly by the numerical arrays. Since arithmetic operations in Excel treat TRUE as 1 and FALSE as 0, this setup often works, but due to internal array handling and compatibility across various versions of Excel, a more explicit conversion method is strongly recommended to ensure reliability and maintain clean code execution. This is where the powerful and slightly mysterious double negative operator comes into play, ensuring deterministic conversion of logical outcomes into usable arithmetic values (ones and zeros).

## The Role of the Double Negative (--) and Type Coercion

To successfully integrate a condition into the SUMPRODUCT function, we must translate the Boolean results (TRUE or FALSE) generated by the logical test into numerical values (1 or 0). This process is known as Type Coercion. When Excel evaluates the condition, such as `(A1:A9>0)`, it creates a new temporary array of the same size, populated entirely by TRUES and FALSEs. For example, if A1=10, A2=-5, and A3=3, the resulting array would be {TRUE, FALSE, TRUE}.

The double negative operator (`--`) is the cleanest and most common way to force this coercion. A single negative sign converts TRUE to -1 and FALSE to 0. Applying the second negative sign flips the signs back, resulting in TRUE becoming 1 and FALSE remaining 0. This manipulation is critical because SUMPRODUCT treats these coerced values as a multiplier. If the condition is TRUE (value greater than zero), it multiplies the corresponding product by 1, allowing it to be included. If the condition is FALSE (value less than or equal to zero), it multiplies the corresponding product by 0, effectively excluding it from the final sum without requiring complex IF statements or intermediate steps.

Other methods for coercing Boolean values exist, such as multiplying the condition by 1 (e.g.,  $(A1:A9>0)*1$ ) or adding zero (e.g.,  $(A1:A9>0)+0$ ), but the double negative operator ( $--$ ) is generally preferred by experienced Excel users due to its succinctness and efficiency in array calculations. Understanding that  $--$  acts as a mathematical gatekeeper, converting logical outcomes into numerical weights of either full inclusion (1) or complete exclusion (0), is the key to leveraging conditional array formulas effectively.

## Constructing the Conditional SUMPRODUCT Formula

The formula structure for calculating the sum of products while respecting the "greater than zero" condition requires three main components, defined within the SUMPRODUCT function. The first component is the logical test, specifically bracketed and coerced using the double negative. The subsequent components are the actual data arrays whose products we wish to sum. The beauty of this syntax is that the first component acts as a filtering array, multiplying element-by-element against the products of the remaining arrays.

To use this function only with values that are greater than zero, you can use the following formula:

**=SUMPRODUCT(--(A1:A9>0),A1:A9,B1:B9)**

Let's break down the components of the formula  $=\text{SUMPRODUCT}(\text{--}(A1:A9>0), A1:A9, B1:B9)$ . The first argument,  $\text{--}(A1:A9>0)$ , evaluates the range A1 through A9. For every cell that holds a value greater than zero, this array produces a 1; for every cell less than or equal to zero, it produces a 0. The subsequent arrays,  $A1:A9$  and  $B1:B9$ , hold the actual numerical data. When SUMPRODUCT runs, it multiplies the corresponding elements of all three arrays. If the filtering array (the first argument) yields 0, the entire product for that row becomes 0, regardless of the values in A and B, thereby achieving the desired exclusion.

This particular formula will only return the sum of the products of the two arrays for the values that are greater than zero in the range **A1:A9**.

It is important to note that the range specified in the logical test (A1:A9 in this case) dictates the criteria for inclusion. If you wanted to filter based on a condition in B1:B9 instead, the formula would simply be adjusted to  $=\text{SUMPRODUCT}(\text{--}(B1:B9>0), A1:A9, B1:B9)$ . This flexibility allows the user to apply the filter to any controlling array necessary, ensuring the calculation accurately reflects the desired subset of data. This powerful technique minimizes computational overhead and maximizes the analytical depth achievable within a single Excel cell.

The following example show how to use this formula in practice.

## Practical Example: Isolating Positive Values for Calculation

To fully appreciate the utility of this conditional filtering technique, let us examine a concrete scenario involving two corresponding arrays of values. Suppose Array A represents inventory adjustments, including positive additions and negative deductions, and Array B represents the unit cost associated with those adjustments. Our goal is to calculate the total cost associated only with the net positive additions (the items greater than zero) in Array A, ignoring all zero or negative adjustments, as they represent items removed or static inventory.

Suppose we have the following two arrays of values in Excel:

	A	B	C	D	E	F	G
1	10	3					
2	5	4					
3	0	5					
4	-3	2					
5	6	5					
6	4	5					
7	10	2					
8	-5	4					
9	10	5					
10							
11							
12							
13							
14							
15							
16							
17							
18							

In this dataset, Column A contains mixed values: positive numbers (10, 5, 6, 4, 10, 10), zero (0), and negative numbers (-3, -5). Column B contains the corresponding cost data. If we were to calculate the sum of products without any filtering, every row would contribute to the result, including the negative products resulting from rows 4 and 8. This standard calculation provides a baseline for comparison, highlighting the difference between a raw aggregation and a conditionally filtered result.

If we use the **SUMPRODUCT** function as usual, we can take the sum of the products between the values in column A and column B:

	A	B	C	D	E	F	G
1	10	3		SUMPRODUCT	144		
2	5	4					
3	0	5					
4	-3	2					
5	6	5					
6	4	5					
7	10	2					
8	-5	4					
9	10	5					
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							

The standard, unconditional use of SUMPRODUCT applied across the entire range A1:A9 and B1:B9 results in a calculation that accounts for all data points. As demonstrated in the screenshot above, the raw calculation incorporates the negative products. Row 4 ( $-3 * 2 = -6$ ) and Row 8 ( $-5 * 4 = -20$ ) significantly reduce the total. Row 3 ( $0 * 5 = 0$ ) also contributes nothing, but does not actively penalize the total sum. Therefore, the resulting sum of products reflects the net effect across the entire dataset, including both additions and deductions.

The sum of the products turns out to be **144**.

We can manually verify that this is correct:

$$\text{Sum of Products} = 10*3 + 5*4 + 0*5 + (-3)*2 + 6*5 + 4*5 + 10*2 + (-5)*4 + 10*5 = \mathbf{144}.$$

## Applying the Conditional Filter

Now, let us execute the requirement: calculating the sum of the products only where the value in Column A is strictly greater than zero. This application demonstrates the true power of the conditional SUMPRODUCT approach, isolating the positive contributions from the noise of negative or zero entries. By introducing the coerced logical test as the first multiplier, we effectively

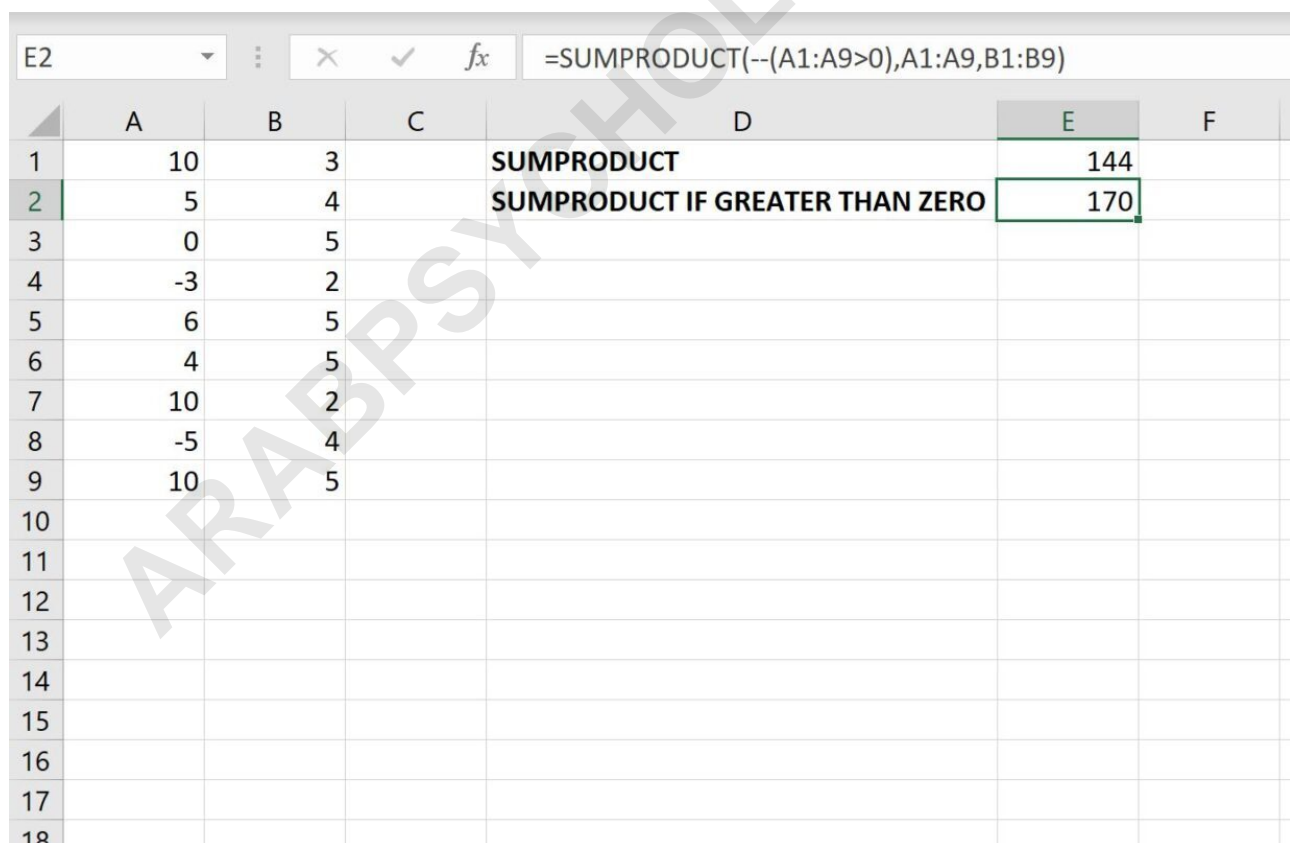
create a dynamic mask over the data, ensuring that only the desired rows are allowed to pass through and contribute a non-zero value to the final sum.

To only take the sum of the products where the value in column A is greater than zero, we can use the following formula:

**=SUMPRODUCT(--(A1:A9>0),A1:A9,B1:B9)**

When Excel processes this formula, the first array `--(A1:A9>0)` generates a multiplier array. For the rows where A is positive (Rows 1, 2, 5, 6, 7, 9), the value is 1. For rows where A is zero or negative (Rows 3, 4, 8), the value is 0. This filtering array is then multiplied against the values in A1:A9 and B1:B9, ensuring that the products corresponding to the filtered rows are instantly zeroed out, thus achieving the exclusion criteria without altering the original source data or requiring temporary calculations.

The following screenshot shows how to use this formula in practice:



The screenshot shows an Excel spreadsheet with the following data and formula:

	A	B	C	D	E	F
1	10	3		SUMPRODUCT	144	
2	5	4		SUMPRODUCT IF GREATER THAN ZERO	170	
3	0	5				
4	-3	2				
5	6	5				
6	4	5				
7	10	2				
8	-5	4				
9	10	5				
10						
11						
12						
13						
14						
15						
16						
17						
18						

The formula bar shows: `=SUMPRODUCT(--(A1:A9>0),A1:A9,B1:B9)`

As illustrated by the result 170, the conditional formula successfully identified and isolated only the rows where A > 0. The rows corresponding to the negative and zero values in Column A (Rows 3,

4, and 8) were effectively ignored by the array calculation. This technique provides a robust and elegant solution for data aggregation where specific constraints must be applied to the contributing arrays, making it invaluable for auditing, reporting, and complex financial modeling.

We can manually verify that this is correct:

Sum of Products =  $10 \times 3 + 5 \times 4 + 6 \times 5 + 4 \times 5 + 10 \times 2 + 10 \times 5 = 170$ .

## Advanced Applications and Considerations

While filtering for values greater than zero is a common requirement, the methodology involving Boolean coercion using `--` is extensible to almost any logical condition. Users can easily adapt this structure to include multiple criteria, simulate AND logic, or perform complex lookups far beyond the capabilities of simpler functions like SUMIF or SUMIFS, which are limited to summing a single range based on criteria.

For example, if you wanted to sum the products only where the value in A is greater than 0 AND the corresponding value in B is greater than 10, you would chain the logical tests together: `=SUMPRODUCT(--(A1:A9>0), --(B1:B9>10), A1:A9, B1:B9)`. Each separate logical test must be enclosed in its own parentheses and preceded by the double negative (`--`) to ensure proper conversion into a 1 or 0 multiplier. The resulting calculation only occurs when both filtering arrays produce a 1 (i.e., both conditions are TRUE for that specific row). This ability to handle complex, multi-layered constraints within a single formula showcases why SUMPRODUCT is often considered an essential tool in advanced Excel mastery, offering immense flexibility.

Furthermore, when working with conditional logic in array formulas, users must pay close attention to the consistency of array sizes. All arrays supplied to SUMPRODUCT--whether they are filtering arrays (like `--(A1:A9>0)`) or numerical data arrays (like `B1:B9`)--must be of identical dimensions (e.g., 9 rows by 1 column). Failure to match array dimensions will result in a `#VALUE!` error, which is a common pitfall when first implementing these sophisticated techniques. Always ensure your criteria ranges are aligned precisely with your data ranges to guarantee successful and accurate results.

## Summary of Conditional SUMPRODUCT Implementation

The method of using SUMPRODUCT with conditional filtering provides a robust, non-volatile alternative to traditional methods involving helper columns or complex nested functions. By leveraging the double negative operator (`--`) to perform type coercion, we ensure that logical TRUE/FALSE outcomes are converted into numerical 1/0 multipliers. This creates an elegant filtering mechanism that multiplies away any unwanted data points, allowing only those that meet the criteria (e.g., values greater than zero) to contribute to the final sum of products.

Key takeaways for implementing this successfully include understanding that the logical test must be enclosed in parentheses and immediately preceded by `--`. Furthermore, maintaining strict consistency in array dimensions is critical. This powerful technique transforms SUMPRODUCT from a simple aggregation tool into a dynamic, conditional calculation engine, greatly expanding the analytical capabilities available to advanced users of Excel. Mastering array manipulation techniques like this is essential for minimizing file size, improving calculation speed, and creating more transparent and manageable spreadsheet models.

For those seeking to implement similar logic but focused solely on summation (and not multiplication), equivalent conditional techniques apply to functions like SUM, using the same coercion methods and requiring entry as a traditional array formula (using `CTRL+SHIFT+ENTER`). However, SUMPRODUCT has the unique advantage of handling array input natively, meaning it does not require the `CTRL+SHIFT+ENTER` combination, simplifying data entry and reducing user error.

**Note:** You can find the complete documentation for the **SUMPRODUCT** function [here](#).