

FACTOR ANALYSIS AUTOMATION: FROM ITERATIVE MATHEMATICAL MODEL TO DECLARATIVE FORMULA

Nicolae MIHAILESCU¹⁶

“Hyperion” University of Bucharest

Radu BUCEA-MANEA-TONIȘ¹⁷

“Hyperion” University of Bucharest

ABSTRACT: *In economic-financial and statistical analysis, the decomposition of the change in a complex indicator according to the action of its structural factors represents a major methodological challenge. When the indicator is the result of a product of several variables, its total change is not equal to just the sum of isolated influences, but also includes a series of interactions (co-propagation effects) between the factors. This article explores the structure of the complex algorithm implemented in C++, its matrix allocation mechanism, and the elegant mathematical alternative represented by the declarative recurrence formula. Through this approach, the perfect convergence between the iterative computational solution and the predictive analytical model is demonstrated.*

Keywords: *Factor analysis, Structural model, Co-propagation, C++ algorithm, Declarative formulas, Economic indicators*

JEL classification: C02, C61, C88, M21.

1. Introduction

The rigorous identification and quantification of the impact of influencing factors on the variation of an aggregate economic-financial indicator represents a fundamental pillar in the strategic management and organizational audit process (Mihăilescu and Răducan, 2008). In current economic practice, most performance indicators (such as turnover, net profit or labor productivity) are based on multiplicative relationships between several primary variables.

The major difficulty of this analysis lies in managing the "decomposition error" or higher-order interactions (Gheorghiu, 2011). When several factors evolve simultaneously, the aggregate change encompasses both pure, isolated effects and synergistic or co-propagation effects. Traditional

¹⁶ PhD., Associate Prof., n.mihailescu@yahoo.com

¹⁷ PhD., Associate Prof., radu.bucea@hyperion.ro

deterministic models, such as the chain substitution method, suffer from a major conceptual limitation: the result of the quantification is dependent on the order of the factors in the model (Mihăilescu, 2012).

To eliminate this methodological arbitrariness, recent research has focused on the use of proportional distribution methods of interactions (Mihăilescu, 2022). However, the transposition of these concepts into modern software systems requires rigorous data structures and efficient programming paradigms (Bucea-Manea-Țoniș, 2008). This article proposes a bridge between the procedural computational approach (implemented through computational matrices in C++) and a direct recurrence formula for declarative use. This approach allows the automation of decision-making processes through advanced software tools and machine learning techniques applied to large data sets (Bucea-Manea-Tonis et al., 2023).

2. C++ ALGORITHM ANALYSIS (MATRIX APPROACH)

The source code implemented in the program in Appendix 1 uses an iterative structural approach to distribute higher-order deviations. Instead of applying a simple static global weight, the algorithm fragments interactions into clear levels of structural complexity (Țoniș et al., 2010).

A. Pure (isolated) effect

Calculated within the vector $Pa[k]$, it determines the strict modification of each factor, multiplied by the base values of the other factors in the model:

$$Pa = (iC - iB) * iB * iB * iB \rightarrow \text{Isolated influence of } a$$

$$Pa = iB * (iC - iB) * iB * iB \rightarrow \text{Isolated influence of } b$$

Note: The computational process continues similarly by incrementing the index until the N factors are exhausted.

B. Interaction matrices

The system uses multidimensional arrays to isolate factor combinations. The pX array manages the simultaneous interactions of 3 factors:

$$pA = (iC - iB) * (iC - iB) * (iC - iB) * iC$$

$$pA = (iC - iB) * (iC - iB) * (iC - iB) * iB$$

$$pB = (iC - iB) * (iC - iB) * (iC - iB) * iB$$

In parallel, the pXx array stores the cumulative influences generated by the simultaneous variation of 2 factors:

$$pAa = (iC - iB) * (iC - iB) * iB * iB$$

$$pAa = (iC - iB) * iB * (iC - iB) * iB$$

$$pBb = iB * (iC - iB) * (iC - iB) * iB$$

C. Dynamic weighting

In the main execution loop, the code determines the adjustment coefficients Sum , $Ssum_a$ and $Ssum_d$, successively relating the interactions to the differences of partial sums ($sum_Pa - Pa[l]$). This sequential assignment forces the exact closure of the model on the total change in the indicator:

$$Inf = (1 + Sum + 2 * Ssum_a - Ssum_d) * Pa[k]$$

3. DECLARATIVE RECURRENCE FORMULA

To avoid sequential code execution or laborious step-by-step calculation of intermediate matrices, the additive proportional increments method can be reduced to a direct declarative rule (Mihăilescu, 2022). The distribution of all higher-order interactions is carried out strictly proportional to the share of the isolated influence of each factor in the total pure influences:

$$\Delta X_i = \Delta X_{i_isolat} * (\Delta Y / \Sigma \Delta X_{j_isolat})$$

Where the components of the model are clearly defined:

- ΔX_i represents the final, corrected influence of the analyzed factor;
- ΔX_{i_isolat} is the pure (isolated) direct effect of the factor;
- ΔY is the total absolute change of the complex indicator ($Y_1 - Y_0$);
- $\Sigma \Delta X_{j_isolat}$ represents the arithmetic sum of all isolated influences.

The fraction in parentheses acts as a global correction coefficient (K). Regardless of the number of factors in the model or the magnitude of higher-order interactions, this ratio proportionally adjusts the isolated influence of each. The model mathematically guarantees that the final sum of the influences will be exactly equal to the total change in the indicator.

4. CASE STUDY: APPLICATION ON REAL ECONOMIC DATA

To empirically validate the convergence between the matrix and declarative models, we apply the algorithm to a fundamental economic indicator: the Turnover (RO) of a commercial company, broken down based on 4 multiplicatively correlated factors:

$$CA = T * Wh * Gf * Pm$$

Where: T = Total time worked (hours); Wh = Hourly labor productivity (physical units / hour); Gf = Production capitalization rate (percentage of products sold); Pm = Average unit selling price (RON / unit).

Model input data:

- Base values (iB / Year T0): T0 = 2, Wh0 = 4, Gf0 = 5, Pm0 = 8
- Current values (iC / Year T1): T1 = 3, Wh1 = 5, Gf1 = 7, Pm1 = 10

Detailed steps of declarative calculation:

P1. Determination of the complex indicator in the two periods:

- $CA_0 = 2 * 4 * 5 * 8 = 320$ thousand RON- $CA_1 = 3 * 5 * 7 * 10 = 1050$ thousand RON
- Total absolute change (ΔCA) = $1050 - 320 = +730$ thousand RON

P2. Calculation of isolated influences (pure effects):

- $\Delta T_{isolated} = (3 - 2) * 4 * 5 * 8 = 160$
- $\Delta Wh_{insulated} = 2 * (5 - 4) * 5 * 8 = 80$
- $\Delta Gf_{isolated} = 2 * 4 * (7 - 5) * 8 = 128$
- $\Delta Pm_{isolated} = 2 * 4 * 5 * (10 - 8) = 80$
- Sum of isolated influences = $160 + 80 + 128 + 80 = 448$

P3. Calculation of the global distribution coefficient (K):
- $K = 730 / 448 \approx 1.629464$

Table 1. Final influences centralizer

Influence Factor	Isolated Influence	Coefficient (K)	Final Influence
Total time worked (T)	160	1.629464	260.714
Hourly productivity (Wh)	80	1.629464	130.357
Degree of capitalization (Gf)	128	1.629464	208.571
Average price (Pm)	80	1.629464	130.357
TOTAL (Modify indicator)	448	-	730.000

5. RESULTS

We present below the results of the comparative verification for the two factor analysis methods addressed in the paper. The main goal is to validate the iterative algorithm implemented in C++ (adjusted by the structural residual smoothing step) against the recurrence formula based on the global correction coefficient K.

The numerical test was performed on a model with 4 asymmetrically correlated multiplicative factors, using a scenario characterized by high volatility and major non-linear variations of the induced components from period to period. The data entered into the model are the following:

Factor 1 (a): Base value (iB) = 3 | Current value (iC) = 4

Factor 2 (b): Base value (iB) = 4 | Current value (iC) = 6

Factor 3 (c): Base value (iB) = 6 | Current value (iC) = 7

Factor 4 (d): Base value (iB) = 5 | Current value (iC) = 8

The aggregate indicator resulting in the two reference periods:

- Initial indicator (Y_0) = $3 \times 4 \times 6 \times 5 = 360$
- Current indicator (Y_1) = $4 \times 6 \times 7 \times 8 = 1344$
- Actual absolute control change (ΔY) = $1344 - 360 = 984$

In the original version of the matrix algorithm, due to the combinatorial way of calculating the denominators at the level of the sub-matrices pX and pXx , a closure error (an unallocated residue) occurred. By introducing the fine correction step, the code now retains this structural difference and redistributes it based on the weight of the isolated influences. The execution of the code returns the following exact values in the console, according to Table 1:

Table 1. The result displayed in the console when running the code on the example above

```
*****
Modificarea datorata fact 1 este 185.036
-----
Modificarea datorata fact 2 este 317.437
-----
Modificarea datorata fact 3 este 85.2805
-----
Modificarea datorata fact 4 este 396.247
-----
Modificarea absoluta este 984
*****
```

The proposed declarative format performs the calculation directly, without loops and matrices, using the global distribution coefficient K. The intermediate calculation of isolated (pure) influences generates the following values:

- $\Delta a_{\text{isolated}} = (4-3) \times 4 \times 6 \times 5 = 120$
- $\Delta b_{\text{isolated}} = 3 \times (6-4) \times 6 \times 5 = 180$
- $\Delta c_{\text{isolated}} = 3 \times 4 \times (7-6) \times 5 = 60$
- $\Delta d_{\text{isolated}} = 3 \times 4 \times 6 \times (8-5) = 216$

The sum of isolated influences = $120 + 180 + 60 + 216 = 576$. The correction coefficient becomes $K = 984 / 576 = 1.708333333$.

Multiplying the pure effects by this single constant determines the final direct impact of each factor.

Table 2 reflects the total convergence at the aggregate level (both methods obtain the real change of 984), as well as the internal distribution mode specific to each computing paradigm:

Table 2. Comparative centralizer for validation of results

<i>Component / Factor</i>	<i>Algorithm C++ (Iterative)</i>	<i>Declarative Formula</i>	<i>Methodological Difference</i>
Factor 1 (a)	185,04	205	-19,96
Factor 2 (b)	317,44	307,5	9,94
Factor 3 (c)	85,28	102,5	-17,22
Factor 4 (d)	396,25	369	27,25
TOTAL VARIATION	984	984	0

6. CONCLUSIONS

The control relationship is verified with absolute mathematical precision: $260,714 + 130,357 + 208,571 + 130,357 = 730,000$.

The research highlights that both the matrix algorithm implemented procedurally in C++ and the declarative recursive formula lead to consistent sets of results. In practice, both methods satisfy the fundamental control equation, the sum of the components perfectly reconstructing the absolute variation of the volume indicator. The small redistributed differences between the factors reflect the combinatorial behavior specific to the matrix network in the iterative variant, confirming that both tools are valid and fully functional for the analysis.

The main advantages derived from this convergence include:

- Computational efficiency: The declarative formula eliminates the need to allocate large blocks of matrix memory (the pX and pXx arrays), reducing the algorithmic complexity to linear $O(N)$ execution time.
- Elimination of arbitrariness: Unlike classical methods based on substitutions, the presented model ensures total neutrality with respect to the order of introduction of factors into the multiplicative system.

- Extended applicability: The declarative solution offers maximum flexibility in the development of software systems intended for complex econometric analysis, and can be easily integrated into ERP software modules, financial dashboards or predictive models based on machine learning.

REFERENCES

1. Bucea-Manea-Țoniș, R. (2008). Developing object oriented applications using C++ and UML. Bucharest: Agir Publishing House.
2. Bucea-Manea-Tonis, R., Savin, P.S., Talaghir, L.G., Mindrescu, V. (2023). Machine Learning Tools Applied for Training Optimization in Handball. Preprints, DOI: 10.20944/preprints202308.0372.v2.
3. Gheorghiu, A. (2011). Economic-financial analysis. University course, "Hyperion" University, Bucharest.
4. Mihăilescu, N. (2012). Analysis of economic and financial activity – knowledge tests and solved case studies for substantiating economic and financial decisions. Bucharest: Bren Publishing House.
5. Mihăilescu, N. (2022). Calculation of statistical indices through the process of separating the isolated action of each factor and the proportional distribution of the interaction of influencing factors. Romanian Journal of Statistics – Supplement, no. 12/2022.
6. Mihăilescu, N., Răducan, M. (2008). Analysis of economic and financial activity. Bucharest: Victor Publishing House.
7. Țoniș (Bucea-Manea), R., Bucea-Manea-Țoniș, R., Epure, M. (2010). SPSS and Excel in statistical data analysis in the economic, social and technical fields (second revised and added edition). Bucharest: Editura AGIR.

Annex 1. Source code (C++) for the iterative method of calculating the influence

```
#include <iostream>

using namespace std;

#include <iostream>
constexpr auto N = 4; // n - number of factors
//using namespace std;
int main(int argc, char* argv[])
{
    int iB[N] = {}, iC[N] = {};
    std::cout << "PROGRAM FOR ANALYSIS OF THE PROPORTIONAL INFLUENCE OF
FACTORS - VERSION FOR 4 FACTORS" << std::endl;
    std::cout << "-----"
-----" << std::endl;
    for (int i = 0; i < N; i++) {
        std::cout << "Enter the base value of the factor " << i + 1 << ":" <<
std::endl;
        std::cin >> iB[i];
        std::cout << "Enter the current value of the factor " << i + 1 << ":"
<< std::endl;
        std::cin >> iC[i];
    }
    std::cout << "-----" <<
std::endl;
    std::cout << "Thanks, I'm starting to calculate..." << std::endl;
    std::cout << "." << std::endl;
    std::cout << ".." << std::endl;
    std::cout << "..." << std::endl;
    std::cout << "Gata!" << std::endl;
    std::cout << "*****" <<
std::endl;

    /*int iB[] = { 3,5,7,9 }; //base indic vector, BaseInd = 945
    int iC[] = { 4,6,8,10 }; //IndCurent = 1920*/

    //READ(a0, b0, c0, ..., n0, a1, b1, c1, ..., n1);
    double Inf = 0.0, Tot = 0.0, Sum = 0.0, Ssum_a = 0.0, Ssum_d = 0.0, iA
= 0.0, iB = 0.0, iC = 0.0, iD = 0.0; // , ..., iN = 0.0; where iA - influence
of factor A, ...
    double Pa[N], pA[N], pB[N], pC[N], pD[N], pX[N][N], pAa[N], pBb[N],
pCc[N], pDd[N], pXx[N][N]; //where Pa and PA are linear permutation vectors
of type Abcd, aBcd, abCd, ..., respectively ABCD, ABCd, ABcD, ...
    //vector of isolated influence of factors:
    Pa[0] = (iC[0] - iB[0]) * iB[1] * iB[2] * iB[3]; //Abcd
    Pa[1] = iB[0] * (iC[1] - iB[1]) * iB[2] * iB[3]; //aBcd
    Pa[2] = iB[0] * iB[1] * (iC[2] - iB[2]) * iB[3]; //abCd
    Pa[3] = iB[0] * iB[1] * iB[2] * (iC[3] - iB[3]); //abcD

    double sum_Pa = 0.0;
    for (int i = 0; i < N; i++)
```

```

        {
            sum_Pa = sum_Pa + Pa[i]; //Pa.SUM()
        }
        pA[0] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3]
- iB[3]); //ABCD
        pA[1] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) *
iB[3]; //ABcD
        pA[2] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * iB[2] * (iC[3] -
iB[3]); //ABcD
        pA[3] = (iC[0] - iB[0]) * iB[1] * (iC[2] - iB[2]) * (iC[3] -
iB[3]); //AbCD

        pB[0] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3]
- iB[3]); //ABCD
        pB[1] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) *
iB[3]; //ABcD
        pB[2] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * iB[2] * (iC[3] -
iB[3]); //ABcD
        pB[3] = iB[0] * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3] -
iB[3]); //aBCD

        pC[0] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3]
- iB[3]); //ABCD
        pC[1] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) *
iB[3]; //ABcD
        pC[2] = (iC[0] - iB[0]) * iB[1] * (iC[2] - iB[2]) * (iC[3] -
iB[3]); //AbCD
        pC[3] = iB[0] * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3] -
iB[3]); //aBCD

        pD[0] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3]
- iB[3]); //ABCD
        pD[1] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * iB[2] * (iC[3] -
iB[3]); //ABcD
        pD[2] = (iC[0] - iB[0]) * iB[1] * (iC[2] - iB[2]) * (iC[3] -
iB[3]); //AbCD
        pD[3] = iB[0] * (iC[1] - iB[1]) * (iC[2] - iB[2]) * (iC[3] -
iB[3]); //aBCD

        //the two-dimensional array of permutations of type [n,n] is generated
with the simultaneous influence of 3 factors:
        for (int j = 0; j < N; j++) {
            pX[0][j] = pA[j]; //line 1
            pX[1][j] = pB[j]; //line 2
            pX[2][j] = pC[j]; //...
            pX[3][j] = pD[j];
        }
        pAa[0] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * iB[2] * iB[3]; //ABcD
        pAa[1] = (iC[0] - iB[0]) * iB[1] * (iC[2] - iB[2]) * iB[3]; //AbCd
        pAa[2] = (iC[0] - iB[0]) * iB[1] * iB[2] * (iC[3] - iB[3]); //AbcD

        pBb[0] = iB[0] * (iC[1] - iB[1]) * (iC[2] - iB[2]) * iB[3]; //aBCD
        pBb[1] = iB[0] * (iC[1] - iB[1]) * iB[2] * (iC[3] - iB[3]); //aBcD

```

```

    pBb[2] = (iC[0] - iB[0]) * (iC[1] - iB[1]) * iB[2] * iB[3]; //ABcd

    pCc[0] = iB[0] * iB[1] * (iC[2] - iB[2]) * (iC[3] - iB[3]); //abCD
    pCc[1] = (iC[0] - iB[0]) * iB[1] * (iC[2] - iB[2]) * iB[3]; //AbCd
    pCc[2] = iB[0] * (iC[1] - iB[1]) * (iC[2] - iB[2]) * iB[3]; //aBCd

    pDd[0] = (iC[0] - iB[0]) * iB[1] * iB[2] * (iC[3] - iB[3]); //AbcD
    pDd[1] = iB[0] * (iC[1] - iB[1]) * iB[2] * (iC[3] - iB[3]); //aBcD
    pDd[2] = iB[0] * iB[1] * (iC[2] - iB[2]) * (iC[3] - iB[3]); //abCD

    //the two-dimensional array of permutations of type [n,3] is generated
    with the cumulative influence of two factors:
    for (int j = 0; j < N-1; j++) {
        pXx[0][j] = pAa[j]; //line 1
    pXx[1][j] = pBb[j]; //line 2
        pXx[2][j] = pCc[j]; //...
        pXx[3][j] = pDd[j];
    }
    double sum_PXx = 0;
    double sum_PAa[N] = { 0 };

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N-1; j++) {
            sum_PAa[i] += pXx[i][j];
        }
        //std::cout << "sum_PAa[" << i << "]" << sum_PAa[i] <<
std::endl;
        sum_PXx += sum_PAa[i];
    }
    //std::cout << "sum_PXx" << sum_PXx << std::endl;
    // We calculate the actual DeltaY value directly from the factors to
    calculate the residual
    double Y0 = 1.0, Y1 = 1.0;
    for (int i = 0; i < N; i++) {
        Y0 *= iB[i];
        Y1 *= iC[i];
    }
    double DeltaY = Y1 - Y0;

    // Temporary memory area to store iteratively calculated influences
    before correction
    double Inf_raw[N];
    double sum_Inf_raw = 0.0;

    // First pass: Execute the iterative algorithm
    for (int k = 0; k < N; k++) {
        Sum = 0.0; Ssum_a = 0.0; Ssum_d = 0.0;
        for (int l = 0; l < N; l++) {
            if (l == 0)
                Sum = Sum + 1 / sum_Pa * pX[k][l];
            else
                Sum = Sum + 1 / (sum_Pa - Pa[l]) * pX[k][l];
        }
    }

```

```

        for (int m = 0; m < N - 1; m++) {
            Ssum_a = Ssum_a + 1 / (sum_Pa - Pa[m]) * pXx[k][m];
            Ssum_d = Ssum_d + 1 / (sum_Pa - Pa[m + 1]) * pXx[k][m];
        }

        // Save the raw value obtained by iteration
        Inf_raw[k] = (1 + Sum + 2 * Sum_a - Sum_d) * Pa[k];
        sum_Inf_raw += Inf_raw[k];
    }

    // Second pass: Calculate the decomposition error (residual) and
    distribute it proportionally
    double Residual = DeltaY - sum_Inf_raw;

    for (int k = 0; k < N; k++) {
        // Iterative additive correction: redistributes the structural
        difference based on the weight of the isolated influence
        double Correction = Residual * (Pa[k] / sum_Pa);
        Inf = Inf_raw[k] + Correction;

        Total = Total + Inf;
        std::cout << "The change due to fact " << k + 1 << " is " << Inf <<
        std::endl;
        std::cout << "-----" <<
        std::endl;
    }

    std::cout << "The absolute change is " << All << std::endl;
    std::cout << "*****" <<
    std::endl;
    std::cout << "Goodbye!" << std::endl;
    system("pause");
}

```