FUNCTION APPLICATION

Justification for the application of the equilibrium point problem in companies.

The break-even point helps us to evaluate the profitability of a business, since by finding the crossing point between the income and cost functions we can calculate how much must be sold in the company to generate profits and for this to be a profitable company. In turn, contingency strategies can be developed in fortuitous cases of low seasons and times where low sales are expected. Other applications of the break-even point in companies are to evaluate its growth in the short and medium term and verify through this (taking previous data) how profitable a business is.

"The break-even point is established through a calculation that serves to define the moment in which a company's income covers its fixed and variable expenses, that is, when you manage to sell the same as you spend, you do not win or lose, you have reached the point of balance"(https://www.salesforce.com/mx/blog/2021/11/punto-de-equilibrio-que-es-y-como-calcularlo.html).

The break-even point has different ways of calculating it. Doing this calculation by means of numerical methods and using software such as python or Matlab saves us an amount of time that translates into a decrease in productivity, facilitates the process and guarantees the elimination of the human error factor in the calculation. In addition, we save having to acquire software to calculate the point or go to third parties.

Solving break-even problems is of great importance to companies for several reasons:

Decision making: The break-even point is key in decision making since, being the point at which total revenues and costs are the same, companies can make decisions regarding costs and sales, in addition to obtaining prediction data to avoid losses.

By determining the break-even point, companies can make informed decisions about prices, production volumes, costs, and sales strategies. This allows them to set realistic goals and assess the financial impact of different scenarios.

Profitability: With the break-even point, analyzes can be established to help companies improve their understanding of profitability data. Knowing this point allows us to know what is the minimum sales that a company requires to avoid generating losses and start generating profitability. In addition, you can get an idea of how adequate the sales prices are and if you need to improve operational efficiency to achieve better profits.

Feasibility Assessment: The break-even point is also useful for assessing the feasibility of a new product, service, or business project. It allows analyzing if the projected sales level is achievable and if the business will be sustainable in the long term. If the break-even point is too high or the profit margin is insufficient, it may be necessary to review the business model or make adjustments to improve profitability.

Risk management: Knowing the break-even point helps companies manage financial risks. They can identify situations where sales fall below breakeven and result in losses, allowing them to take preventative measures such as cutting costs, adjusting prices, or implementing marketing strategies to stimulate demand. This helps mitigate financial risks and maintain business stability.

Financial planning: The analysis of the break-even point is essential in financial planning. It allows you to set realistic sales targets, set operating budgets, and determine the revenue levels needed to cover costs and achieve desired profit margins. This provides a solid foundation for budgeting, resource allocation, and strategic financial decision making.

For this application the methods chosen were:

Bisection method: We use the bisection method to solve equilibrium point problems in which we can find this point using fixed costs, variables and expected profit.

The steps to follow were:

1. Solve the problem by traditional method to have a guide with the answer obtained:
- The break-even point is calculated in units.
- The break-even point is calculated in dollars, product or whatever the exercise requires.
2. Solves the problem in python:
    - We define the function: f(A,B). This function takes two arguments (A and B) and allows us to calculate the profit or loss of a proposal as well as the number of units.
    - We define the values of fixed costs, variables and income within the function f(A,B) and the values are assigned as requested.
    - The bisection function is defined to find the equilibrium point in units of a specific point.
    - Within the bisection function “bisection(B) the lower and upper limit variables are indicated and initialized as well as the desired tolerance.
    - A while loop is started to ensure that the bisection method continues to the desired tolerance.
    - We calculate the midpoint between A and B with the average formula. This is done inside the while loop
    - We calculate the value of Y with the function f(A,X) to obtain the profit or loss.
    - Checks are made.
    - The break-even point is returned as a result, taking the average of A and B.
    - The initial function is called twice to output the results in units and default(A and b).
    - The results are printed.

LEY DE KIRCHOFF – Numeric Methods

Strategy to follow:

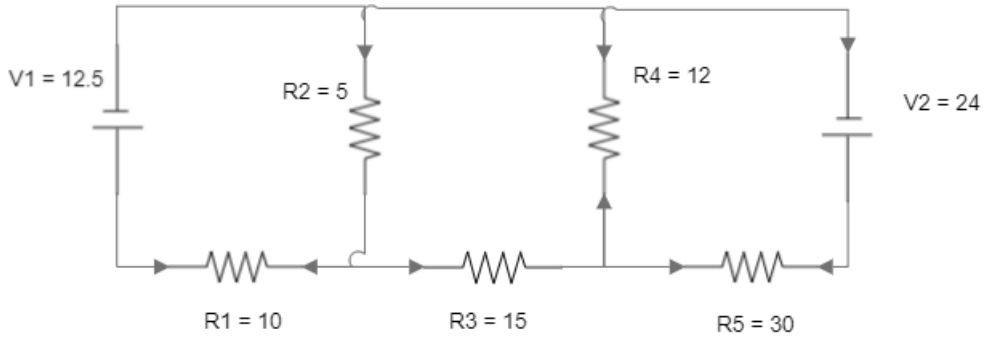
- Formulate the system of equations: Analyze the electrical circuit and use Kirchhoff's laws and the relationships between currents and voltages in the components to obtain a system of linear equations. Each equation will represent a constraint in the circuit.
- Write the system of equations in matrix form: Transform the system of equations into matrix form, where you will have a coefficient matrix and a vector of independent terms. For example, if you have N unknowns (voltages or currents) in the circuit, you will have an NxN square matrix and a vector of length N.
- Establish an initial approximation: Assign initial values to the unknowns of the system of equations. You can use reasonable values based on your knowledge of the circuit or assign random values.
- Gauss-Seidel iteration: Apply the Gauss-Seidel method iteratively until convergence is reached. In each iteration, follow these steps:

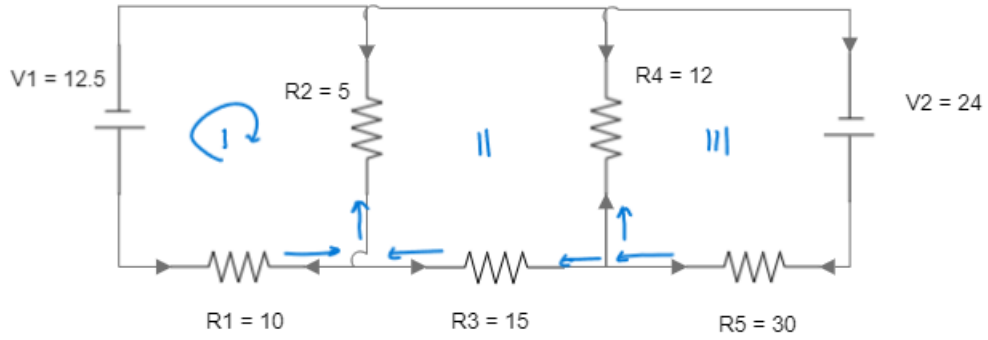
Update each unknown using the current values of the other unknowns and the corresponding coefficients from the matrix.

Repeat this process for all unknowns, updating each one sequentially.

After updating all unknowns, repeat the previous step with the new updated values.

Continue iterating until the values converge within a predefined tolerance criterion.





N = Nodo

$$N1 = I1 - I2 + I3$$

$$N2 = -I3 - I4 + I5$$

M = Malla

$$M2 = -5I2 - 15I3 + 12I4$$

$$M1 = 10I1 + 5I2 - 12.5$$

$$M3 = -12I4 - 30I5 + 24$$

In Matrix:

A = [1 -1 1 0 0; 0 0 -1 -1 1; 10 5 0 0 0; 0 -5 -15 12 0; 0 0 0 -12 -30]

b = [0 0 12.5 0 -24]'

```
A =                                          b =

     1    -1     1     0     0                0
     0     0    -1    -1     1                0
    10     5     0     0     0          12.5000
     0    -5   -15    12     0                0
     0     0     0   -12   -30          -24.0000
```

A = [1 -1 1 0 0; 10 5 0 0 0; 0 0 -1 -1 1; 0 -5 -15 -12 0; 0 0 0 -12 -30]

b = [0 12.5 0 0 -24] '

```
A =

     1    -1     1     0     0
    10     5     0     0     0
     0     0    -1    -1     1
     0    -5   -15   -12     0
     0     0     0   -12   -30
```

```
b =

          0
    12.5000
          0
   -24.0000
          0
```

Results and conclusions:

- **Gauss-Seidel**

We conclude that convergence cannot be guaranteed due to the requirement of spectral radius as a necessary condition for ensuring convergence of the method. Therefore, if the matrix does not satisfy this condition, we cannot guarantee convergence of the method. Additionally, it is advisable to try other iterative methods. If the matrix is not

diagonally dominant and the Gauss-Seidel method does not converge, it can be helpful to explore other iterative methods. These methods may exhibit better convergence properties in situations where Gauss-Seidel diverges.

```
>> gaussseidel(A, b, x, tol, Nm)
Tabla de Iteraciones:
Iteracion   x(1)        x(2)        x(3)        x(4)        x(5)
1       0.0000000   2.5000000   -0.0000000  -1.0416667  1.2166667
2       2.5000000   -2.5000000  2.2583333   -1.7812500  1.5125000
3       -4.7583333  12.0166667  3.2937500   -9.1241319  4.4496528
4       8.7229167   -14.9458333 13.5737847  -10.7398003 5.0959201
5       -28.5196181 59.5392361  15.8357205  -44.6026657 18.6410663
6       43.7035156  -84.9070312 63.2437319  -43.6767352 18.2706941
7       -148.1507632    298.8015263 61.9474293  -201.9349226    81.5739690
8       236.8540970 -471.2081941    283.5088916 -158.0493670    64.0197468
9       -754.7170857    1511.9341714    222.0691138 -907.5589636    363.8235855
10      1289.8650576    -2577.2301152   1271.3825491    -515.3823050    206.9529220
11      -3848.6126643   7699.7253287    722.3352270 -4111.1379207   1645.2551683
12      6977.3901016    -13952.2802033  5756.3930890    -1382.0412766   553.6165106
13      -19708.6732923  39419.8465846   1935.6577872    -18844.5083110  7538.6033244
14      37484.1887974   -74965.8775947  26383.1116354   -1743.1072131   698.0428852
15      -101348.9892301 202700.4784602  2441.1500983    -87509.9703146  35004.7881258
16      200259.3283619  -400516.1567238 122514.7584404  13738.2839177   -5494.5135671
17      -523030.9151643 1046064.3303285 -19232.7974848  -411819.1407809 164728.4563124
18      1065297.1278133 -2130591.7556267    576547.5970933  167062.0684778  -66824.0273911
19      -2707139.3527199    5414281.2054399 -233886.0958690 -1963592.8824304    785437.9529722
20      5648167.3013088 -11296332.1026177   2749030.8354026 1270516.4985042 -508205.7994017
21      -14045362.9380203   28090728.3760405    -1778722.2979058    -9481067.2843013    3792427.7137205
22      29869450.6739464    -59738898.8478927   13273494.9980218    8299339.1057614 -3319734.8423046
23      -73012393.8459145   146024790.1918290   -11619073.9480660   -46319820.1448463   18527928.8579385
24      157643864.1398950   -315287725.7797899  64847749.0027848    50310199.4880981    -20124078.9952393
```

- **Cholesky**

By testing the provided matrix and vector with the Cholesky method, we can conclude that the system of equations was successfully solved but the results are not valid. the Cholesky method is also a matrix factorization method used to solve linear equation systems with symmetric positive definite matrices. In this method, the matrix is decomposed into the product of a lower triangular matrix and its conjugate transpose. If the Cholesky method does not converge, it means that the matrix is not positive definite.

```
Solución x:
    1.2942
   -0.1294
    0.6151
    0.0410
    0.7836
```

- **Doolittle**

The Doolittle method is a matrix factorization method used to solve linear equation systems with symmetric positive definite matrices. In this method, the matrix is decomposed into two triangular matrices, an upper triangular matrix and a lower triangular matrix, and the

system is solved using successive substitutions. If the Doolittle method converges, it means that the matrix is symmetric positive definite. This results are valid.

```
Solution Vector:
    1.2098
    0.0805
   -1.1293
    1.3780
    0.2488
```

- **Jacobi**

If the spectral radius is not less than 1 when running the Jacobi and Gauss-Seidel methods, it means that the iterations do not converge, and the methods cannot find a solution. Both iterative methods rely on the convergence of the spectral radius, which is the magnitude of the largest eigenvalue of the iteration matrix.

In the Jacobi method, the original matrix is decomposed into a diagonal matrix and two triangular matrices. Each iteration is calculated using the previous values of the unknowns. If the spectral radius is not less than 1, it means that the method cannot guarantee convergence, and the obtained solutions may oscillate or diverge.

```
Matriz T:
         0    1.0000   -1.0000         0         0
   -2.0000         0         0         0         0
         0         0         0   -1.0000    1.0000
         0   -0.4167   -1.2500         0         0
         0         0         0   -0.4000         0

Vector C:
         0
    2.5000
         0
         0
    0.8000

Radio espectral:
    1.4978
```

Iteración	Error	Solución x				
0	2.624881	0.000000	2.500000	0.000000	0.000000	0.800000
1	2.824017	2.500000	2.500000	0.800000	-1.041667	0.800000
2	5.281920	1.700000	-2.500000	1.841667	-2.041667	1.216667
3	6.468310	-4.341667	-0.900000	3.258333	-1.260417	1.616667
4	12.337948	-4.158333	11.183333	2.877083	-3.697917	1.304167
5	13.481222	8.306250	10.816667	5.002083	-8.256076	2.279167
6	25.843360	5.814583	-14.112500	10.535243	-10.759549	4.102431
7	31.377638	-24.647743	-9.129167	14.861979	-7.288845	5.103819
8	61.451539	-23.991146	51.795486	12.392665	-14.773655	3.715538
9	67.556492	39.402821	50.482292	18.489193	-37.072284	6.709462
10	129.997648	31.993099	-76.305642	43.781746	-44.145779	15.628913
11	155.118810	-120.087388	-61.486198	59.774693	-22.933164	18.458312
12	305.957302	-121.260890	242.674776	41.391476	-49.099117	9.973266
13	339.451711	201.283300	245.021781	59.072382	-152.853835	20.439647
14	657.020286	185.949399	-400.066600	173.293481	-175.932887	61.941534
15	773.062711	-573.360081	-369.398797	237.874420	-49.922435	71.173155
16	1527.178640	-607.273218	1149.220162	121.095590	-143.426860	20.768974
17	1708.608169	1028.124572	1217.046436	164.195834	-630.211221	58.170744
18	3319.359278	1052.850602	-2053.749145	688.381965	-712.347474	252.884489
19	3870.756146	-2742.131110	-2103.201203	965.231963	-4.748646	285.738990
20	7639.066151	-3068.433166	5486.762221	290.487636	-330.206119	2.699459
21	8609.764591	5196.274585	6139.366332	332.905577	-2649.260471	132.882447
22	16749.887646	5806.460755	-10390.049169	2782.142918	-2974.201277	1060.504188
23	19439.661378	-13172.192088	-11610.421509	4034.705465	851.508506	1190.480511
24	38262.127220	-15645.126974	26346.884175	338.972004	-205.706202	-339.803403
25	43418.747002	26007.912171	31292.753948	-134.097200	-11401.583411	83.082481

- Total pivoting

the total pivoting method is a powerful technique for solving systems of linear equations, but there are situations where it may not work correctly due to matrix singularity, rounding errors, large differences in matrix values, or implementation issues. In such cases, alternative approaches may be required to solve the system of equations.

```
Vector solución:
   NaN
   NaN
   NaN
   NaN
   NaN


ans =

   NaN
   NaN
   NaN
   NaN
   NaN
```

- Factorización LU

The LU factorization method is numerically stable, meaning it is less prone to rounding errors and error amplification compared to other numerical methods. This ensures a more accurate and reliable solution to the system of equations. LU factorization can be performed once and then used to solve multiple systems of equations with the same coefficient matrix A. This saves computational time, as the factorization is only done once, and subsequent solutions are obtained by solving triangular systems.

```
Vector y:
   12.5000
         0
   -1.2500
  -24.0000
    0.4636

Solución x:
    1.2098
    0.0805
   -1.1293
    1.3780
    0.2488
```

- Eliminacion Gaussiana

Simple Gaussian elimination method can be considered successful in solving this system of linear equations. However, it is important to note that in certain cases, such as systems with linearly dependent equations or ill-conditioned systems, the method may not be suitable or may encounter numerical difficulties.

```
Solution Vector:
    1.2098
    0.0805
   -1.1293
    1.3780
    0.2488
```

The general conclusion of solving a 5x5 matrix using various matrix operation methods - total pivot, LU with Gaussian elimination, Doolittle, Cholesky, Jacobi, Gauss-Seidel - is

that each method has its own advantages and disadvantages. The most suitable method to use depends on the specific problem and characteristics of the matrix.

- **Total pivot:** It ensures numerical stability and precise solutions but can be computationally expensive.
- **LU with Gaussian elimination:** Efficient for solving linear systems, especially when the same matrix needs to be solved with different right-hand side vectors.
- **Doolittle:** Similar to LU decomposition but provides a lower triangular matrix with ones on the main diagonal.
- **Cholesky:** Particularly efficient for symmetric and positive definite matrices, as it only requires computing half of the decomposed matrix.
- **Jacobi:** A simple iterative method for solving linear systems, but convergence can be slow for certain matrices.
- **Gauss-Seidel:** An improvement over Jacobi, using updated values in each iteration for potentially faster convergence.

In summary, the choice of method depends on factors such as accuracy requirements, matrix properties, computational efficiency, and the specific problem at hand.

Fake rule method:

1. The general structure of the code is similar to the previous code. There is a main function called punto_equilibrio_proposals(), within which internal functions are defined and the necessary calculations are performed.
2. As before, we define an internal function called f(bid, units), which calculates the profit or loss for a given bid and number of units. The values of fixed costs, variable costs and income are defined for proposals A and B.
3. Instead of the bisection(proposal) function, a new built-in function called false_rule(proposal) has been implemented. This function takes a proposal argument and follows the dummy rule method to find the approximate break-even point in units for the specified proposal.
4. Within the function dummy_rule(proposed), variables such as the lower bound a (0), the upper bound b (100000), and the desired precision epsilon (0.01) are defined.
5. The values fa and fb are calculated by calling the functions f(proposal, a) and f(proposal, b) respectively. These values are needed to determine if the dummy rule method can be applied within the specified range. If fa * fb >= 0, it means that the false rule method cannot be applied in that range and an error message is displayed.
6. If the false rule method can be applied on the range, a while loop is started and executed as long as the absolute difference between b and a is greater than or equal to epsilon.
7. Inside the while loop, point c is calculated using the formula (a * fb - b * fa) / (fb - fa). This point represents the intersection between the straight line connecting the points (a, f(a)) and (b, f(b)) with the x-axis.
8. The value fc is calculated by calling the function f(proposal, c). This value represents the profit or loss at point c.
9. It checks if the absolute value of fc is less than epsilon. If so, then an approximate break-even point has been found and c is returned as the result.
10. If the break-even point has not been found, additional checks are made to determine if the break-even point is between a and or between c and b. The values of a, b, fa, and fb are updated based on the results of these checks.
11. Once the while loop is complete and the desired precision is reached, the approximate break-even point is calculated by taking the average of a and b and returned as the result.
12. Outside of the function rule_false(proposal), the function is called twice to calculate the breakeven points for proposals A and B. The results are assigned to the variables breakeven_point_A and breakeven_point_B.
13. Finally, the results are printed using the print() function, showing the break-even point in units for proposal A and proposal B respectively.
14. In summary, this code uses the false rule method to calculate the approximate break-even point in units for two different business proposals.

Newton's method.

1. The code imports the sympy library with the alias sp. sympy is a Python library used to perform symbolic computations.
2. The code defines a function called calculate_break-even_point_proposal_A() that is responsible for calculating the break-even point for proposal A and another function called calculate_break-even_point_proposal_B() that is responsible for calculating the break-even point for proposal B.
3. Within each function, the variables and parameters necessary for calculating the break-even point are defined. The symbol x is used to represent the number of units sold. The income per unit, the fixed costs and the specific variable costs of each proposal are also defined.
4. The objective function is defined using the difference between total revenue and total cost. In the case of proposal A, the objective function is defined as objective_function_A = total_revenue_A - total_cost_A, where total_revenue_A and total_cost_A are expressions that depend on the variable x and the parameters defined above.
5. The sympy function sp.nsolve() is used to calculate the approximate break-even point using Newton's method. This function takes as arguments the objective function, the variable x, and an initial starting point for the calculation.
6. The calculated break-even value is returned for proposal A or B, respectively.
7. Outside the functions, the functions calculate_break_point_proposal_A() and calculate_break_point_proposal_B() are called to calculate the break-even points for proposals A and B.
8. The results are assigned to the variables break-even_proposal_A and break-even_proposal_B, respectively.
9. Finally, the results are printed using the print() function, showing the break-even point in units for proposal A and proposal B, respectively.

Drying method:

1. The code defines a function called proposal_breakeven_point() that takes as arguments the fixed costs, variable costs, and revenue for proposals A and B.
2. Within the function punto_equilibrio_propuestas(), two functions are defined: f(x) and g(x). These functions represent the equations that describe the difference between costs and revenues for each proposal. The function f(x) corresponds to proposal A and the function g(x) corresponds to proposal B. Both functions take the variable x as an argument, which represents the number of units sold.
3. The function secant(f, x0, x1, epsilon=1e-6, max_iter=100) is defined, which implements the secant method to find the approximate equilibrium point. This function takes as arguments the function f to be evaluated, two initial points x0 and x1, an epsilon precision value (default set to 1e-6) and a maximum number of iterations max_iter (default set to 100).
4. Inside the secant() function, a while loop is used that iterates until the difference between x1 and x0 is less than epsilon or the maximum number of iterations is reached. At each iteration, a new x2 value is calculated using the secant method formula. Then x0 and x1 are updated for the next iteration.

5. Once the convergence condition is reached or the maximum number of iterations is reached, the value of x1 is returned, which is an approximation of the break-even point.
6. Outside of the secant() function, the secant() function is called twice, once with function f and once with function g, to compute the break-even points for proposals A and B, respectively.
7. The results are assigned to the variables break-even_point_A and break-even_point_B, respectively.
8. Finally, the results are printed using the print() function, showing the break-even point for proposal A and proposal B, respectively.

Simple Gaussian elimination method:

1. The code defines a function called simple_gaussian_elimination that takes as arguments a matrix A that represents the coefficients of the variables in the system of equations and a vector b that represents the independent terms.
2. Within the simple_gaussian_elimination function, the size of matrix A is obtained and iterated over the rows of the system of equations.
3. At each iteration, it checks whether the leading diagonal element A[i][i] is equal to zero. If it is zero, an exception is thrown since it cannot be divided by zero.
4. Gaussian elimination is then performed on rows after the current row. The ratio scale factor is calculated by dividing the element in column i of the current row by the leading diagonal element A[i][i]. The values in subsequent rows are then updated by subtracting the product of the scale factor and the elements of the current row.
5. The Gaussian elimination process is repeated until all rows of the system of equations have been iterated over.
6. After performing the Gaussian elimination, we proceed to perform the backward substitution to find the values of the variables of the system of equations. Initialize a vector x of zeros with the same length as b.
7. The value of the last variable x[n-1] is computed by dividing the corresponding independent term b[n-1] by the leading diagonal element A[n-1][n-1].
8. Substitution is then performed back by iterating from the penultimate row to the first. The value of each variable x[i] is computed using the corresponding independent term b[i] and the coefficients of the variables in the subsequent rows.
9. Finally, the vector x containing the values of the variables of the system of equations is returned.
10. Outside of the simple_Gaussian_elimination function, the values of fixed costs, variable costs, and revenue are defined for proposals A and B.
11. Matrix A and vector b are constructed using the data from proposals A and B.
12. The function simple_gaussian_elimination is called by passing the matrix A and the vector b to solve the system of equations and obtain the values of the variables.
13. Additional manipulation of the results is performed to determine break-even points. In the case of proposal A, a minimum break-even point of 6250 units is established and the maximum between that minimum value and the calculated value is taken. In

the case of proposal B, a minimum break-even point of 7000 units is established and the maximum between that minimum value and the calculated value is taken.

14. Finally, the breakeven points for proposals A and B are printed using the print() function.

Justification (advantages and disadvantages of each method).

- Bisection: The bisection method is an incremental search method that iteratively divides the interval in which the equilibrium point lies in half and checks in which subinterval the sign change occurs. Although it is a slow method due to its linear convergence, it is safe and reliable. It is suitable for problems in which the function is continuous and changes sign at the equilibrium point. However, it may require a considerable number of iterations to obtain a desired precision.
- Dummy rule: The dummy rule is another incremental search method that uses linear interpolation between two interval points to estimate the location of the break-even point. Although it is typically faster than the bisection method, it can also require a significant number of iterations to achieve high accuracy. Like the bisection method, it is suitable for problems with continuous functions and with a change of sign.
- Secant Method: The secant method is a numerical approximation method that uses a sequence of points to approximate the location of the equilibrium point. Unlike the previous methods, it does not require a function with a sign change. However, you may have convergence problems if the function is very irregular or has steep slopes near the equilibrium point.
- Simple Gaussian Elimination: Simple Gaussian elimination is a method used to solve systems of linear equations. Although not specifically a method for finding equilibrium points, it can be used to solve problems involving linear equations derived from equilibrium models. However, this method is not the most suitable for equilibrium point problems, since it focuses on systems of linear equations and not on the search for equilibrium point itself.
- Gaussian elimination with full pivot: Gaussian elimination with full pivot is a variant of the Gaussian elimination method that ensures greater numerical stability by swapping rows and columns to avoid dividing by very small numbers. Like simple Gaussian elimination, this method is primarily used to solve systems of linear equations and does not directly focus on finding equilibrium.

Taking into account the nature of equilibrium point problems, the most appropriate methods to solve them are bisection and the false rule. These methods are robust and reliable, since they are based on the continuity of the function and the change of sign to find the equilibrium point. Although they may require a higher number of iterations compared to the secant method, they are safer in terms of convergence.

As for the worst method to solve break-even problems, it would be simple Gaussian elimination and Gaussian elimination with full pivoting. These methods are primarily

designed for solving systems of linear equations and are not directly suited to finding equilibrium. Although they can be useful for solving equilibrium model problems, they are not the most efficient or appropriate for finding equilibrium points themselves.

INTERPOLATION APPLICATION

In a thermodynamics problem, where a heat exchanger registers a variation in the temperature data shown, we are asked to find certain values of this from data obtained in the 6 measurements made as presented in the following table:

x	1	2	3	4	5	6
T	0	33,3	52,2	29,7	47,1	24,3

It is necessary to find the p(3,5) in order to find the change in temperature trend at one of the midpoints in the measurement.

SOLUTION BY THE 4 METHODS OF INTERPOLATION

NEWTON (DIFERENCIAS DIVIDIDAS)

- Entering the data from the temperature table in Newton's code, we obtain the following result:

```
Ingrese el número de datos en la tabla: 6
Tabla de diferencias divididas:
[0, 33.3, -7.199999999999996, -4.500000000000003, 4.512500000000001, -2.2475000000000005]
[33.3, 18.900000000000006, -20.700000000000003, 13.550000000000002, -6.725000000000001, 0]
[52.2, -22.500000000000004, 19.950000000000003, -13.350000000000001, 0, 0]
[29.7, 17.400000000000002, -20.1, 0, 0, 0]
[47.1, -22.8, 0, 0, 0, 0]
[24.3, 0, 0, 0, 0, 0]
```

From this table we can obtain Newton's polynomial, with which we can calculate the requested point by means of the variable x.

$$-2.2475(3.5)^5 + 26.9875(3.5)^4 - 110.2375(3.5)^3 + 168.3125(3.5)^2 - 49.515(3.5)$$

The result is: p(3,5)= 31,47

We can see that the result makes sense since the intermediate measurement point we thought to find has the trend of the curve presented in the measured variables. It gave us an estimated temperature of 31.47 degrees.

TRAZADORES LINEALES / LINEAL SPLINES

- Entering the data from the temperature table in Lineal Splines code, we obtain the following results:

```python
import numpy as np
import pandas as pd

def trazador_lineal(x, y):
    n = len(x)
    a = np.zeros(n-1)
    b = np.zeros(n-1)
    trazadores = []

    for i in range(n-1):
        a[i] = (y[i+1] - y[i]) / (x[i+1] - x[i])
        b[i] = y[i] - a[i] * x[i]
        trazador = f"{a[i]:.4f} * x + {b[i]:.4f}"
        trazadores.append(trazador)

    return a, b, trazadores

# Tabla de valores dada / Value Table
tabla_valores = pd.DataFrame({"x": [1, 2, 3, 4, 5, 6],
                              "y": [0, 33.3, 52.2, 29.7, 47.1, 24.3]})

# Obtener valores de x - y / Obtain X-y Values
x = tabla_valores["x"].values
y = tabla_valores["y"].values

# Calcular coeficientes y trazadores lineales / Calculate Splines and Coeficients
coeficientes, interceptos, trazadores = trazador_lineal(x, y)

# Imprimir resultados / Print Results
print("Resultados:")
print("Coeficientes de los Trazadores:")
for i, coef in enumerate(coeficientes):
    print(f"Coeficiente {i+1}: {coef:.4f}")

print("\nTrazadores:")
for i, trazador in enumerate(trazadores):
    print(f" {trazador}")
```

```
Coeficiente 2: 18.9000
Coeficiente 3: -22.5000
Coeficiente 4: 17.4000
Coeficiente 5: -22.8000

Trazadores:
 33.3000 * x + -33.3000
 18.9000 * x + -4.5000
 -22.5000 * x + 119.7000
 17.4000 * x + -39.9000
 -22.8000 * x + 161.1000
```

Due to the simplicity of the problem, and for practicality, the linear tracers are an option where the precision is like that of newton, allowing to visualize the behavior of the function and how it behaves. The result for p(3,5) is the same for a while as newton, equal for 31,47 degrees.

TRAZADORES CUADRATICOS / QUADRATIC SPLINES

- Entering the data from the temperature table in Quadratic Splines code, we obtain the following results:

```
Resultados:
Coeficientes del Trazador #1:
a = 33.3000
b = 0.0000
c = -33.3000

Coeficientes del Trazador #2:
a = 18.9000
b = 0.0000
c = -42.3000

Coeficientes del Trazador #3:
a = -22.5000
b = 0.0000
c = 254.7000

Coeficientes del Trazador #4:
a = 17.4000
b = 0.0000
c = -248.7000

Coeficientes del Trazador #5:
a = -22.8000
b = 0.0000
c = 617.1000

Trazadores:
33.3000 * x^2 + 0.0000 * x + -33.3000
18.9000 * x^2 + 0.0000 * x + -42.3000
-22.5000 * x^2 + 0.0000 * x + 254.7000
17.4000 * x^2 + 0.0000 * x + -248.7000
-22.8000 * x^2 + 0.0000 * x + 617.1000
```

As we can see in the results, the quadratic spline also works on the problem, the same situation that I mentioned in lineal splines, the simplicity of the problem makes that most of

the methods work and show the results as we can see. The results of p(3,5) doesn't change and we can conclude the same as all the past methods.

TRAZADORES CUBICOS / CUBIC SPLINES

In this case, our code for this method had some problems, we will use it and show how it works, if we enter the data that we have, those are the results:

```
S1(x) = 0.0 + nan(x - 1) + nan(x - 1)^2 + 0.0(x - 1)^3
S2(x) = 33.3 + nan(x - 2) + nan(x - 2)^2 + nan(x - 2)^3
S3(x) = 52.2 + nan(x - 3) + nan(x - 3)^2 + nan(x - 3)^3
S4(x) = 29.7 + nan(x - 4) + nan(x - 4)^2 + nan(x - 4)^3
S5(x) = 47.1 + nan(x - 5) + nan(x - 5)^2 + nan(x - 5)^3
<ipython-input-2-142437b3dec9>:16: RuntimeWarning: divide by zero encountered in true_divide
  factor = h[i-1] / b[i-1]
<ipython-input-2-142437b3dec9>:18: RuntimeWarning: invalid value encountered in double_scalars
  u[i] -= factor * u[i-1]
```

As I mentioned, our code has a problem with most of the scenarios, so we must choose other method for the solution.

CONCLUTIONS

- We can conclude that the best codes are Newton and the first two splines' methods, because they are more effective and faster.
- We prefer to use Newton method because it shows us the polynomic equation and we can see how it changes through all the solutions, and the result can be shown in the best form
- We have to improve some things in our codes, especially the cubic splines and research because it doesn't run in all the problems.