A Python code for adjusting periodic soil temperature data

https://doi.org/10.56238/sevened2024.003-046

Geilson de Almeida Soares[1], Ana Maria Bersch Domingues[2], Jairo Valões de Alencar Ramalho[3] and Honório Joaquim Fernando[4]

ABSTRACT
This work aims to present a code in Python language for adjusting periodic curves from soil temperature simulation data. We present a methodology to obtain a sinusoidal function using the least squares method that represents the soil temperature of a given locality. The code also uses Newton's method to solve the resulting nonlinear system of equations. Soil temperature simulation data were used in a city in the state of Rio Grande do Sul, Brazil. The results of these curves can help in the study and definition of boundary conditions of new computational models of ground-to-air heat exchangers (TCSA). It is observed that the least squares method is not new , but there is a gap in the literature regarding the presentation of codes for the adjustment of trigonometric functions as presented here.

Keywords: Adjustment of Trigonometric Functions, Least Squares, Ground-to-Air Heat Exchangers, Python.

[1] Federal University of Pelotas
E-mail: geilson.soares@ufpel.edu.br
[2] Federal University of Pelotas
E-mail: ambdomingues@ufpel.edu.br
[3] Federal University of Pelotas
E-mail: jairo.ramalho@ufpel.edu.br
[4] Fluminense Federal University
E-mail: honoriofernando@id.ufl.br

INTRODUCTION

One way to reduce energy consumption in air conditioning systems is through the use of ground-to-air heat exchangers (TCSA). These devices are composed of fans and buried ducts, which blow the outside air inside the ducts, allowing heat exchanges between the air and the ground. As a result, the air comes out at a milder temperature. This is because the soil acts as a thermal reservoir, storing heat in the summer and releasing it in the winter. Thus, the temperature of the soil is generally higher than that of the outside air in winter, allowing a heating of the circulating air in buried pipes thanks to the heat exchange between the soil and the duct. In summer, the opposite occurs, when the soil temperature is lower than that of atmospheric air. Among some references on SAT, we can mention (DOMINGUES *et al.*, 2021; RAMALHO *et al.*, 2022; VAZ *et al.*, 2011)

To simulate TCSA, it is common to use models for the air and soil temperatures of a given region. This work evaluates the adjustment by trigonometric functions of the temperature simulation data in a city in the state of Rio Grande do Sul, Brazil, where the climate is subtropical and the annual variation of temperatures follows a periodic pattern.

There are few papers presenting codes for adjustments involving functions periodicals, as is the case of (BRUM *et al.*, 2011), where the authors adopted a continuous real function $f(x) = Asen(Bx + C) + D$ (with A, B, C, and D being real constants) to model, using the least squares method, a discrete set of ordered pairs $U = (xi, yi)$, $i = 1, 2, ..., n$, where n is a natural number. On the other hand, in several situations, the period T of the phenomenon is known and, consequently, the angular frequency $B = 2\pi/T$ is also known. For example, the soil temperature in Rio Grande do Sul follows an annual periodic variation, that is, we can take T as being 365 days. Thus, the objective of this study is to present a new version of the code proposed in (BRUM *et al.*, 2011), to determine only the coefficients A, C and D.

METHODOLOGY

DISCRETE LEAST SQUARES

One can use trigonometric functions of type $f(x) = Asen(Bx + C) + D$ to model soil temperature. Given that the value of $B = 2\pi/T$ is known, the other con- tants A, C and D can be determined by the least squares method, so as to minimize the error function:

$$Erro(A,C,D) := \sum_{i=1}^{N} [f(x_i) - y_i]^2 = \sum_{i=1}^{N} \left[A \operatorname{sen}\left(\frac{2\pi}{T} x_i + C\right) + D - y_i \right]^2 \qquad (1)$$

where N is the number of data.

Under assumptions such as those considered in (BRUM *et al.*, 2011), it can be stated that the minimum value of the function in (1) is reached when its gradient is zero. In doing so, we are led to

$$\begin{cases} \sum_{i=1}^{N} \left[A \operatorname{sen}\left(\frac{2\pi}{T}x_i + C\right) + D - y_i \right] \operatorname{sen}\left(\frac{2\pi}{T}x_i + C\right) = 0\,, \\ \sum_{i=1}^{N} \left[A \operatorname{sen}\left(\frac{2\pi}{T}x_i + C\right) + D - y_i \right] A\cos\left(\frac{2\pi}{T}x_i + C\right) = 0\,, \\ \sum_{i=1}^{N} \left[A \operatorname{sen}\left(\frac{2\pi}{T}x_i + C\right) + D - y_i \right] = 0\,. \end{cases} \tag{2}$$

To solve the nonlinear system in (2), Newton's method was adopted.

NEWTON'S METHOD

Given a nonlinear and differentiable vector field

$$\vec{F}(A,C,D) = \begin{pmatrix} u(A,C,D) \\ v(A,C,D) \\ w(A,C,D) \end{pmatrix}, \tag{3}$$

You can get closer $\vec{F}$ in the vicinity of a point $x0 = (A0, C0, D0)$, by the fieldo vetorial linear:

$$\vec{L}(A,C,D) = \begin{pmatrix} u(x_0) + u_A(x_0)(A - A_0) + u_C(x_0)(C - C_0) + u_D(x_0)(D - D_0) \\ v(x_0) + v_A(x_0)(A - A_0) + v_C(x_0)(C - C_0) + v_D(x_0)(D - D_0) \\ w(x_0) + w_A(x_0)(A - A_0) + w_C(x_0)(C - C_0) + w_D(x_0)(D - D_0) \end{pmatrix}. \tag{4}$$

Therefore, instead of looking for a solution for $\vec{F}(A,C,D) = 0$, the approximate problem $\vec{L}(A,C,D) = 0$ that has the matrix representation is solved

$$J(x_0)x = J(x_0)x_0 - \vec{F}(x_0)\,, \tag{5}$$

where

$$J(s) = \begin{bmatrix} u_A(s) & u_C(s) & u_D(s) \\ v_A(s) & v_C(s) & v_D(s) \\ w_A(s) & w_C(s) & w_D(s) \end{bmatrix}, \quad x = \begin{pmatrix} A \\ C \\ D \end{pmatrix}, \quad \text{e} \quad x_0 = \begin{pmatrix} A_0 \\ C_0 \\ D_0 \end{pmatrix}, \qquad (6)$$

With $J(s)$ denoting the Jacobian matrix of the vector field at (3) evaluated at point $s \in R3$. Therefore, each input $zp(s)$, $z \in \{u,v,w\}$ and $p \in \{A,C,D\}$ of $J(s)$, indicates the partial derivative of z with respect to p.

Thus, when one wants to find an approximate value of the *xr root* of a vector field, this can be done iteratively by choosing an initial approximation *x0* and calculating successive approximations by solving the system of linear equations

$$J(x_k)x_{k+1} = J(x_k)x_k - \vec{F}(x_k)\,, \qquad (7)$$

With $k = 0, 1, 2,...$This solving technique is called Newton's Method (BURDEN; FAIRES; BURDEN, 2015).

In order to reduce the computational cost associated with the direct implementation of (7), the auxiliary variable *tk+1 := xk+1 − xk is introduced,* which transforms the problem into the following sequence:

1. Determine *tk+1* em $\boldsymbol{J}(xk)tk+1 = -\mathrm{F}\rightarrow(xk)$,
2. Calculate $xk+1 = xk + tk+1$,

Thus avoiding the calculation of the first term on the right side of (7), which can demand a significant computational cost in the entire process of searching for the *root xr* of ~~L→~~

Newton's method can converge relatively quickly or diverge depending on certain conditions, such as the initial approximation chosen and the values of the partial derivatives in the Jacobian matrix. There is a discussion on this subject available in the papers (BURDEN; FAIRES; BURDEN, 2015; BRUM *et al.*, 2011).

CODE USED ANNOTATED

In order to show how the nonlinear system is solved numerically, the developed code is presented, which was implemented in the Python programming language (version 3.11.2). In Python, text to the right of the # symbol, as well as text enclosed by 3 quotation marks (' ' '), are comments. NumPy is a Python language library that has the capacity to handle large arrays (multidimensional arrays) and offers a wide variety of advanced mathematical functions to work with these arrays. The functions used in the code are described below:

- array() : defines an array;
- max() : finds the maximum value in an array;
- mean() : calculates the arithmetic mean,
- argmax() : returns the position of the maximum value;
- linalg.solve() : returns the solution of a system of linear equations,
- abs(): returns the absolute value;
- sum() : returns the sum.

Therefore, the Python language provides a wide range of out-of-the-box resources that facilitate the implementation of the most varied computational algorithms. Detailed explanations of the Python programming language can be found in (LANGTANGEN, 2011). The following list presents in its entirety the code developed in this work.

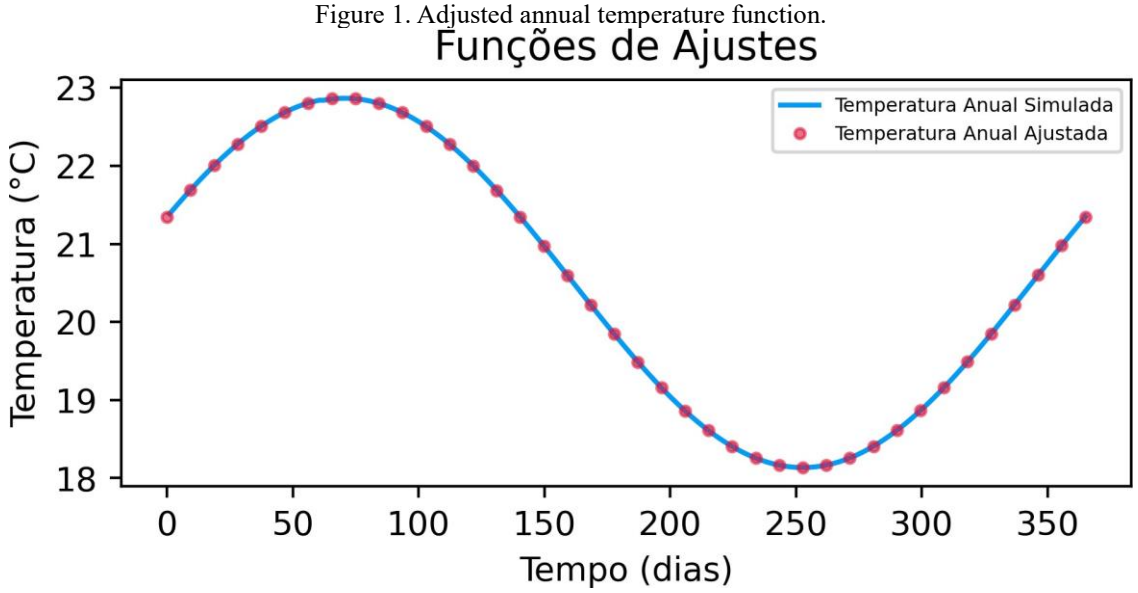
```
import numpy as np

'''
Esta função calcula os coeficientes a,b,c,d para uma função do tipo
a*sin(bx+c)+d que ajusta a solução da temperatura anual.
'''
def ajustes(lista_dias, dados):
    # Converter lista_dias para um array numérico
    x = np.array(lista_dias).astype(float)
    # Atribuindo os dados ao y
    y = dados

    # Estimativas iniciais para os valores dos coeficientes da função seno
    y_max = np.max(y)
    dia_max = np.argmax(y)
    d = np.mean(y)  # Média aritmética dos elementos da matriz
    a = y_max - d
    b = 2*np.pi/365
    c = np.pi/2 - 2*np.pi*dia_max/365

    # Tolerância e número máximo de iterações
    tol = 1e-5
    nMax = 200

    # Intervalos de tempo (x) e Valores de temperatura (y)
    # Número de intervalos de tempo
```

```
    n = max(x)

    cont = 1  # Contador para o número de iterações

    while cont < nMax:

        # Computação da Matriz Jacobiana
        jac = np.array([
            [np.sum(np.sin(b*x+c)**2), np.sum(np.cos(b*x+c)*(2*a*np.sin(b*x+c)+
    d-y)), np.sum(np.sin(b*x+c))],
            [np.sum(np.sin(b*x+c)*np.cos(b*x+c)), np.sum((a*(np.cos(b*x+c))
    **2-(a*np.sin(b*x+c)+d-y)*np.sin(b*x+c))), np.sum(np.cos(b*x+c))],
            [np.sum(np.sin(b*x+c)), np.sum(a*np.cos(b*x+c)), n]
        ])

        # Computação do Campo Vetorial F
        F = np.array([
            [np.sum((a*np.sin(b*x+c)+d-y)*np.sin(b*x+c))],
            [np.sum((a*np.sin(b*x+c)+d-y)*np.cos(b*x+c))],
            [np.sum(a*np.sin(b*x+c)+d-y)]
        ])

        # Resolvendo o sistema linear jac t =-F
        t = np.linalg.solve(jac, -F)

        # Verificar a convergência na norma do infinito
        if np.max(np.abs(t)) < tol: break
        else:
            cont += 1

            # Atualiza a solução
            a += t[0]
            c += t[1]
            d += t[2]

    # Retorno dos coeficiêntes
    return a[0], c[0], d[0]
```

RESULTS

In this study, simulated data from research carried out in the study (RAMALHO *et al.*, 2022). The data refer to the average daily soil temperatures in the city of Viamão/RS, over the course of a year. Thus, we sought to determine the best sinusoidal function of type $f(x) = Asen((2\pi/\mathrm{T})x + C) + D$ that would fit the data of all 365 days of the year. For this, the methodology described in the previous section was used. Figure 1 presents a graph that illustrates both the simulated data and the simulated temperature adjustment curve over a year. As can be seen, the sinusoidal function is an adequate and natural representation of the variation in temperature throughout the seasons, reaching its maximum extreme in summer and minimum in winter, in addition to accompanying the gradual fall and increase in temperatures during autumn and spring, respectively. In fact, as evidenced in the same figure, the adjusted curve was very close to the simulated data.

Figure 1. Adjusted annual temperature function.



Source: from the author.

CONCLUSIONS

The objective of this study was to develop a computational code in the Python programming language for periodic curve adjustments. Inspired by a similar work done in the MatLab programming language, a simplified version was developed for the case in which the period of the phenomenon is known to reduce the computational cost. Simulated soil temperature data from a city in the state of Rio Grande do Sul were used.

It was observed that the adjustment of the Python code presented a very satisfactory approximation in relation to the simulated data. The analysis of the quality of the fit will be done in future works. In addition, this study contributed to fill a gap in the literature regarding the computational implementation of the least squares method for fitting trigonometric functions.

ACKNOWLEDGMENT

Geilson de Almeida Soares thanks CNPq for the scientific initiation scholarship. Likewise, Ana Maria Bersch Domingues is grateful for the financial support of Capes.

REFERENCES

1. Brum, R. S., Ramalho, J. V. A., Rocha, L. A. O., Isoldi, L. A., Santos, E. D. (2015). A Matlab code to fit periodic data. Revista Brasileira de Computação Aplicada, 7, 16–25.

2. Burden, R. L., Faires, J. D., Burden, A. M. (2015). Análise Numérica (10a ed.). São Paulo: Cengage Learning. ISBN: 978-85-2212-341-4.

3. Domingues, A.M.B., Nóbrega, E.S.B., Ramalho, J.V.A., Brum, R.S., Quadros, R.S. (2021). Parameter analysis of Earth-air heat exchangers over multi-layered soils in South Brazil. Journal Geothermics, 93, 1–14.

4. Ramalho, J. V. A., Fernando, H. J., Brum, R. S., Domingues, A. M. B., Pastor, N. R. N., Olivera, M. R. B. (2022). Accessing the thermal performance of Earth-air heat exchangers surrounded by galvanized structures. Sustainable Energy Technologies and Assessments, 54, 1–11.

5. Vaz, J., Sattler, M.A., Santos, E.D., Isoldi, L.A. (2011). Experimental and numerical analysis of an earth-air heat exchanger. Journal Energy and Buildings, 43, 2476–2482.

6. Langtangen, H. P. (2011). A Primer on Scientific Programming with Python (2nd ed.). Springer-Verlag, Berlin, Heidelberg.