UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Civil, Arquitetura e Urbanismo

IC639: Métodos Numéricos para Engenharia Civil

List 7

IVB, BVP, Least Square Method

Student:

Carlos Henrique Chama Puga - 195416

Advisors:

Porf. Dr. Philippe Devloo

Dr. Giovane Avancini

Campinas

2024

Contents

1 Introduction

The final list of the course on Numerical Methods in Civil Engineering is divided into three main topics: Initial Value Problem (IVP), Boundary Value Problem (BVP), and Least Square Method. All three topics are better discussed in the following sections. During the resolution of the exercises, the following bibliography is used for further understanding of the topics: (1, 2).

1.1 Initial Value Problem (IVP)

In science and engineering, many problems can be modeled using differential equations. In these problems, the rate of change of one or more variables with respect to another (space or time, for example) is considered.

In the majority of cases, these differential equations are not easily solved analytically so numerical methods are used to approximate the solution. Initial Value Problems are a type of differential equation where the solution is known at a single point. The solution is then propagated to other points using numerical methods such as the Runge-Kutta method.

Runge-Kutta methods, in general, are a family of numerical methods used to solve ordinary differential equations. The biggest advantage of this approach, when compared to Taylor's methods, is that they do not require the computation of the derivatives of the function, which is computationally expensive. The most common Runge-Kutta method employed is the second- and fourth-order methods, which can be expressed through the Butcher Tableau.

Let Eq. (1.1) be the ordinary differential equation to be solved

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0, \tag{1.1}$$

then, the Runge-Kutta methods take the form of

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i, \tag{1.2}$$

in which h is the step size, s is the number of stages, b_i are the weights (found in the Butcher Tableau), and k_i are the intermediate values.

The intermediate values are calculated as

$$k_i = f(x_n + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} k_j), \tag{1.3}$$

where c_i and a_{ij} are the coefficients of the Butcher Tableau. A given Butcher Tableau is given of the form

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Exercise 1 implements a class capable of reading a given Butcher Tableau and performing the Runge-Kutta method to solve the equation

$$y' = 1 + (x - y)^2, \quad \forall 2 \leq x \leq 3, \quad y(2) = 1, \tag{1.4}$$

using the following Runge-Kutta method:

0				
1/3	1/3			
2/3	-1/3	1		
1	1	-1	1	
	1/8	3/8	3/8	1/8

1.2 Boundary Value Problem (BVP)

1.3 Least Square Method

The Least Square Method is used to approximate a set of data points to a given function. This new function now interpolates nontabulated points so that the error between the data points and the function is minimized.

The method is based on minimizing the sum of the squares of the residuals, which are the differences between the observed values and the values predicted by the model. Least Squares Methods are widely used in regression analysis, curve fitting, and solving overdetermined systems of equations.

Let E be the error function given by

$$E = \sum_{i=1}^{m} (y_i - P_n(x_i))^2, \tag{1.5}$$

where m is the number of data points, n is the degree of the polynomial, a_j are the coefficients of the polynomial, x_i and y_i are the data points, and $P_n(x_i)$ is the polynomial evaluated at x_i given by

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n. \tag{1.6}$$

The error function is minimized by solving the linear system of equations associated with the derivatives of the error function with respect to the coefficients a_{ij} of the polynomial

$$\frac{\partial E}{\partial a_j} = 0 \rightarrow \sum_{k=0}^{n} a_k \sum_{i=1}^{m} x_i^{j+k} = \sum_{i=1}^{m} y_i x_i^j, \quad \forall j = 0, 1, \ldots, n. \tag{1.7}$$

By solving the system of equations in Eq. (1.7) the coefficients of the polynomial are found and the function that best fits the data points is obtained. How-

ever, the data points may not be perfectly interpolated by the polynomial. In this case, an exponential function can be used to fit the data points

$$y_i = bx_i^a. \tag{1.8}$$

When the data points are fit by Eq. (1.8), two approaches can be taken: 1) linearize the equation by taking the logarithm of the data points or 2) use a nonlinear solver to find the coefficients a and b that minimize the error function.

In 1), the following procedure is employed: eq. (1.8) is transformed into

$$\ln(y_i) = \ln(b) + a\ln(x_i), \tag{1.9}$$

and the error function is minimized by solving the linear system as if it were a first-degree polynomial. Then the coefficients a and b are found by taking the exponential of the coefficients found in the linear system.

In 2), the error function is given by

$$E_{exp} = \sum_{i=1}^{m}(y_i - bx_i^a)^2, \tag{1.10}$$

and the nonlinear system is formed by the derivatives of Eq. (1.10) with respect to the coefficients a and b

$$\frac{\partial E_{exp}}{\partial a} = 2\sum_{i=1}^{m}(y_i - bx_i^a)(-b\ln(x_i)x_i^a) = 0, \tag{1.11}$$

$$\frac{\partial E_{exp}}{\partial b} = 2\sum_{i=1}^{m}(y_i - bx_i^a)(-x_i^a) = 0. \tag{1.12}$$

Solving the nonlinear system of equations formed by eqs. (1.11) and (1.12) leads to the coefficients a and b that minimize the error function. To solve the

system, any method can be used, such as the Newton method, the Broyden method, or the Secant method.

It is important to mention that solving the linearization of the exponential function, although computationally less expensive, does not return the Least Squares solution. In other words, the solution of the linearization is expected to be less accurate than the solution of the nonlinear system of equations.

For exercise 3, a set of data points is given, and a code must be implemented so that the data points are fit by an exponential function (linearized and nonlinearized). The results are then compared and the error between the both approximates is calculated.

2 Code Implementation

This section presents the implementation of the classes used to solve the exercises proposed. For solving IVP, the RungeKutta class is developed. For BVP, the Galerkin class is implemented. Finally, for the Least Square Method, the LeastSquare class is created.

Code 1 shows the main function for the Runge-Kutta problem.

```
def main_rk()->None:
    ft = lambda t, y: 1 + (t - y) ** 2

    rk = RungeKutta(ft, 2, 3, 0.1, 1)

    # 3/8 rule 4th order Runge-Kutta method
    rk.SetButcherTableau(method = "ThreeEights")
    rk.Run()

    print(rk.sol)

    rk.WriteResults("List7/Results_RK.txt")

```

```
14      return
```

Code Listing 1: Main function for the Runge-Kutta problem

On line 2, the right-hand side of the differential equation is defined. On line 4, the RungeKutta object is created with the function, initial and final times, time step, and initial condition. The Butcher Tableau is set on line 7, and the Run method is called on line 8. The results are printed on line 10, and written to a file on line 12.

The main function for the Least Squares Method is shown in Code 2

```
1  def LeastSquareMethod()->None:
2      data = [xi, yi]
3
4      squares = LeastSquare(*zip(*data), "NonLinear")
5      squares.Run()
6
7      print(f"{squares.alpha=}")
8      print(f"{squares.approx_solution=}")
9      print(f"{squares.errors=}")
10     print(f"{squares.total_error=}")
11
12     return
```

Code Listing 2: Main function for the Least Squares Method

Where line 2 defines the set of data points. The LeastSquare object is created on line 4, and the run method is called on line 5. The results are printed on lines 7 to 10.

2.1 The RungeKutta Class

The RungeKutta class' attributes are displayed in Code 3.

```
1  @dataclass
2  class RungeKutta:
3      ft: callable
4      t0: float
```

```
    tf: float
    Dt: float
    y0: float

    ButcherTableau: list[float] = field(init=False, default_factory=list)
    c: list[float] = field(init=False, default_factory=list)
    a: list[list[float]] = field(init=False, default_factory=list)
    b: list[float] = field(init=False, default_factory=list)

    sol: list[float] = field(init=False, default_factory=list)
    step: float = field(init=False, default=0)
```

Code Listing 3: Attributes of the RungeKutta class

Where f_t is the function to be solved, t0 and tf are the initial and final times, Dt is the time step, and y0 is the initial condition. The Butcher Tableau is a list of lists containing the coefficients of the Runge-Kutta method. The c, a, and b lists are the coefficients of the Butcher Tableau. The sol list stores the solution to the problem, and the step attribute is used to store the current time step.

As methods, the class has the SetButcherTableau, ConstantK, and Run. Other methods such as the EulerMethod, RK2, RK4, ThreeEightsRule, and WriteResults are also implemented.

The SetButcherTableau method is shown in Code 4

```
def SetButcherTableau(self, **var)->None:
    butcher = {"euler": self.EulerMethod(), "rk2": self.RK2(), "rk4": self.
    RK4(), "ThreeEights": self.ThreeEighthRule()}

    if 'method' in var:
        butcher[var['method']]

    elif 'ButcherTableau' in var:
        self.ButcherTableau = var['ButcherTableau']

    else:
```

```
        raise ValueError("Invalid Butcher Tableau")
```

Code Listing 4: SetButcherTableau method

This method sets the Butcher Tableau according to the method chosen. The Euler method, Runge-Kutta 2nd order, Runge-Kutta 4th order, and Three-Eighths Rule are implemented. The user can also set the Butcher Tableau manually, by passing the desired rule.

The ConstantK method is shown in Code 5

```
def ConstantK(self, index, t, y, k)->float:
    a = t + self.c[index] * self.Dt
    b = y + self.Dt * sum([self.a[index][j] * k[j] for j in range(index)])

    return self.ft(a, b)
```

Code Listing 5: ConstantK method

This method is responsible for evaluating the intermediate values of the Runge-Kutta method. The index is the stage of the method, t is the current step time, y is the current value of the function, and k is the list of intermediate values. The return is the function evaluated at the intermediate point.

Finally, the Run method is shown in Code 6

```
def Run(self)->None:
    t = self.t0
    y = self.y0

    self.c, self.a, self.b = self.ButcherTableau

    k = []
    for _ in range(self.step):
        for i in range(len(self.c)):
            k.append(self.ConstantK(i, t, y, k))

        y += self.Dt * sum([self.b[j] * k[j] for j in range(len(k))])
        t += self.Dt

```

```
15          k.clear()
16          self.sol.append((t, y))
```

Code Listing 6: Run method

The Run method performs the Runge-Kutta method for a given Butcher Tableau. On lines 2 and 3, the initial values of t and y are set. The Butcher Tableau coefficients (c, a, and b) are unpacked on line 5.

From line 8 on, the method iterates over the number of steps. For each step, the intermediate values are calculated on line 10. The new value of y is calculated on line 12, and the new value of t is calculated on line 13. The intermediate values are cleared on line 15, and the t and y values are appended to the sol list on line 16.

2.2 The Galerkin Class

2.3 The LeastSquare Class

3 Results

The results for exercises 1, 2, and 3 are briefly presented in the following sections.

3.1 Exercise 1

For exercise 1, the Runge-Kutta method was implemented using the Butcher Tableau. As a result, the approximated value for the ordinary differential equation is obtained for each time step. The results are displayed in Table 3.1.

3.2 Exercise 2

3.3 Exercise 3

Table 3.1: Results for the Runge-Kutta method

x	y
2.0	1.0
2.1	1.19091
2.2	1.36667
2.3	1.53077
2.4	1.68571
2.5	1.83333
2.6	1.97500
2.7	2.11176
2.8	2.24444
2.9	2.37368
3.0	2.50000

4 Conclusions

References

1 CUNHA, M. C. de C. *Métodos numéricos*. [S.l.]: Editora da UNICAMP, 2000.

2 BURDEN, R. L.; FAIRES, J. D. *Numerical analysis, brooks*. [S.l.]: Cole publishing company Pacific Grove, CA:, 1997.

A GitHub Repository

The source code for this report and every code mentioned can be found in the following GitHub repository: CarlosPuga14/MetodosNumericos_2024S1.