AIMS | African Institute for Mathematical Sciences CAMEROON

A Python Environment for Numerical Continuation Methods

Jafar Anafi (jafar.anafi@aims-cameroon.org)
African Institute for Mathematical Sciences (AIMS)
Cameroon

Supervised by:Prof. Georg Bader
Brandenburg Technical University, Cottbus-Senftenberg, Germany

7 May 2020

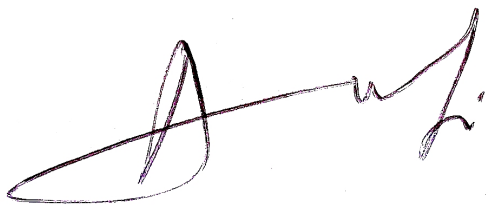
Submitted in Partial Fulfillment of a Structured Masters Degree at AIMS-Cameroon

Abstract

Developing computer models is a natural part of scientific computing. These models typically combine components of mature scientific software of varying levels of complexity. Using low-level software libraries leads to a better performance when running the computer model, but slows down and distracts from model development because of the need to deal with software related technical details. High-level interactive programming environments such as Python simplify and speed up development of computer models but the speed of the numerical solution of the model may be unacceptable for many research problems. One way to solve such problems is to write Software wrappers and make classical low level sources available in a Python environment.

Declaration

I, the undersigned, hereby declare that the work contained in this essay is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.



Firstname Middlename Lastname, 30 May 2016.

Contents

1. Introduction

Developing computer models is a natural part of scientific computing. These models typically combine components of mature scientific software of varying levels of complexity. Using low-level software libraries leads to a higher performance when running the computer model, but slows down and distracts from model development because of the need to deal with software related technical details. High-level interactive programming environments such as Python simplify and speed up development of computer models, however the performance of the numerical solution of the model may be unacceptable for many research problems.
One way to overcome these problems is to write software wrappers which make classical low level software available under a Python environment. An alternative is to re-implement classical software packages in modern computer languages such as Python. However, this is a very time consuming procedure. Particularly this is not just a formal translation and needs well trained scientists who are not just specialists in computer science but also in the particular field where the problems solved are coming from. In any case the related problems to be solved in this context are highly complex and need interdisciplinary knowledge.
One class of problems which appear in a large number of different fields is the solution of nonlinear equations and more so the solution of systems of nonlinear equations. The classical fields in which such problems naturally arise range from structural mechanics to fluid dynamics. These problems frequently appear in the context of partial differential equations. If these problems are discretized then large scale nonlinear systems of algebraic equations need to be solved. Further applications steam from the bifurcation analysis of nonlinear systems of algebraic equation. One property of the majority of nonlinear problems is that they mostly cannot be solved in a closed analytic form. As a consequence, such problems have to be solved by iterative methods. This adds a further level of complexity to the solution procedure.
Many important applications require the numerical treatment of nonlinear finite-dimensional systems of n equations

$$F(\mathbf{x}, \tau) = 0,$$

where the solution $\mathbf{x}$ must be studied as a function of a parameter τ. Standard continuation procedures based on τ-parametrization (see Wacker [14]) exhibit difficulties near points, at which the Jacobian matrix $D\mathbf{F}_{\mathbf{x}}$ is singular. These so-called critical points are either turning points or bifurcation points. Numerical pathfollowing beyond turning points may be performed by an explicit change of the embedding (replacing τ by some appropriate different parameter, see e.g. Anselone and Moore [1] and Deuflhard [7]). Chapter 3 of the present thesis deals with some implicit change of the embedding realized by means of a special (rank-deficient) Newton method for the extended variable $F(\mathbf{x}, \tau)$. This method, which basically seems to date back to suggestions of Haselgrove [10], is proved to converge locally and quadratically (like Newton's method), as long as bifurcation points are excluded. To speed up the computations, special Jacobian rank-1 updates are suggested and analyzed. This Newton's method is naturally combined with a discrete tangent continuation.

1.1 Statement of the Problem

Several forms of research have been conducted on numerical continuation. In this work, we computed some numerical codes for the numerical continuation method of systems for algebraic equations and made them it available to Python. The result of the computer model will be applied for solve some

typical problems for nonlinear systems of algebraic equations.

1.2 Aim and Objectives of the Study

This research aims at constructing software that can find a numerical continuation. We shall achieve this by the following objectives:

1. Study the theoretical background of the Newton method and of the continuation method.
2. Develop a python codes to deal with the Newton method and the continuation method.
3. Apply the python codes to solve two numerical examples and analyse the obtained results.

1.3 Scope and Limitations of Study

The general intent of this study is to find the numerical continuation method with a focus on nonlinear systems of algebraic equations. This study will mainly identify and assess the numerical continuation method and present the numerical codes in python.

2. Mathematical Background

In this chapter we review some of the mathematical concepts that we will be use in this essay.

2.0.1 Kronecker delta. This is a function of two variables, usually just non-negative integers. The function is 1 if the variables are equal, and 0 otherwise and is denoted by

$$\delta_{i,p} := \begin{cases} 1, \ if \ p = i \\ 0, \ if \ p \neq i. \end{cases} \tag{2.0.1}$$

2.0.2 Monotonic Decreasing. A function $f : \mathbb{R} \longrightarrow \mathbb{R}$ is said to be monotonic decreasing, when $x \leq y$ implies that $f(x) \geq f(y)$.

2.0.3 Monotonic Increasing. A function $f : \mathbb{R} \longrightarrow \mathbb{R}$ is said to be monotonic increasing, when $x \leq y$ implies that $f(x) \leq f(y)$.

2.0.4 Definition (Vector Norm)**.** A vector norm on the vector space X is a function $||\cdot|| : X \longrightarrow \mathbb{R}$ which satisfies the following properties:

1. $||\mathbf{x}|| \geq 0$ for all $x \in X$
2. $||\mathbf{x}|| = 0$ if and only if $\mathbf{x} = \mathbf{0}$
3. $||\alpha\mathbf{x}|| = |\alpha|||\mathbf{x}||$ for all $\alpha \in \mathbb{R}$ and $\mathbf{x} \in X$
4. $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$ for all $\mathbf{x}, \mathbf{y} \in X$

2.0.5 Definition (Lipschitz Continuity)**.** A function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is said to be Lipschitz continuous if there exists a constant $L > 0$ such that for all $x, y \in \mathbb{R}^n$ it holds that$||f(x) - f(y)|| = L|x - y|$. L is called a $Lipschitz\ constant$.

2.0.6 Definition. Linear Operator Let $(X, ||\cdot||_X), (Y, ||\cdot||_Y)$ be two normed vector spaces and $F : X \longrightarrow Y$ be an operator. F is called $linear\ operator$ if for all $x, y \in X$ and for all $\alpha, \beta \in \mathbb{R}$ it holds that $F(\alpha x + \beta y) = \alpha F(x) + \beta F(y)$.

2.0.7 Operator Norm. Let $(X, ||\cdot||_X), (Y, ||\cdot||_Y)$ be two normed vector spaces and $F : X \longrightarrow Y$ be a linear operator. The operator norm is defined as $|||F||| := \sup\{||F(x)||_Y : x \in X \ where \ ||x||_X \leq 1\}$.

2.0.8 Definition (Open Ball)**.** Let $(X, ||\cdot||_X)$ be a normed vector space. The open ball centered at $a \in X$ with Radius $r > 0$ denoted by $B(a, r)$ is defined to be the set $B(a, r) := \{x \in X \ \|x - a\|_X < r\}$. If $X = \mathbb{R}$, the open ball centered at $a \in \mathbb{R}$ with radius $r > 0$ is $B(a, r) = \{x \in \mathbb{R} : \|x - a\| < r\} = (a - r, a + r)$.

2.0.9 Definition (Closed Ball)**.** Let $(X, ||\cdot||_X)$ be a normed vector space. The closed ball centered at $a \in X$ with Radius $r > 0$ denoted by $\bar{B}(a, r)$ is defined to be the set $\bar{B}(a, r) := \{x \in X \ \|x - a\|_X \leq r\}$. If $X = \mathbb{R}$, the closed ball centered at $a \in \mathbb{R}$ with radius $r > 0$ is $\bar{B}(a, r) = \{x \in \mathbb{R} : \|x - a\| \leq r\} = [a - r, a + r]$.

2.0.10 Definition (Convergence)**.** [13]: Let x_n be a sequence that converges to x^*, where $x_n \neq x^*$. x_n is said to converge to x^* of order α if there exist constants $\mu > 0$ and $\alpha > 0$, such that

$$\lim_{n \longrightarrow \infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^\alpha} = \mu. \tag{2.0.2}$$

The case $\alpha = 1$ is called $linear\ convergence$ and the case $\alpha = 2$ is called $quadratic\ convergence$.

2.0.11 Definition (Locally Convergence)**.** An iterative method is $locally\ convergent$ if approximations x_k, where $k \in \mathbb{N}$ resulting from the method are guaranteed to converge to a solution x^* when the initial approximation x_0 is already chosen close enough to the solution.

2.0.12 Definition (Globally Convergence)**.** An iterative method that converges for an arbitrary initial guess x_0 is said to be $globally\ convergent$.

3. Newton Method

Over the decades, we learned how to solve equations using various algebraic methods such as the substitution method, elimination method, and graphical methods. However, when these methods are not successful, we use the concept of numerical methods. Numerical methods are used to approximate solutions of equations when exact solutions can not be determined via algebraic methods [13]. This chapter deals with the Newton method as such, a numerical method. Section 3.1 briefly describes this algorithm when it is applied to system of equations as well as the main components it uses. Section 3.2 focuses on the convergence properties of the Newton method with respect to systems of equations.

3.1 Newton Method for System of Equations

The procedure of the Newton method applied to system of equations is an iterative algorithm which is described after introducing the notions of a system of equations as well as the Jacobian matrix which is as follows:

1. **System of Equations**: The system of equations consists of n equations and n variables depicted below, where $n \in \mathbb{N}$:

$$\begin{aligned} f_1(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) &= 0 \end{aligned}$$

   This system of equations can be denoted as

$$\mathbf{f}(\mathbf{x}) = \mathbf{0},$$

   where $\mathbf{0}$ denotes the zero vector.

2. **Jacobian Matrix**: The Jacobian matrix which belongs to the function $\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is denoted as $D\mathbf{f}(\mathbf{x})$ and consists of n rows and n columns. The entry a_{ij} in this matrix is the derivative of the function f_i with respect to the variable x_j. Hence, the Jacobian matrix reads as follows:

$$D\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{pmatrix}. \tag{3.1.1}$$

3. **Algorithm**: The Newton method is an iterative algorithm following the procedure below:

   (a) Choose an initial guess $\mathbf{x}_0$

   (b) For all $m = 0, 1, 2, \ldots \quad n \in \mathbb{N}$

i. Compute the Jacobian matrix $D\mathbf{f}(\mathbf{x}_m)$
ii. Solve the linear system $D\mathbf{f}(\mathbf{x}_m)\mathbf{h}_m = -\mathbf{f}(\mathbf{x}_m)$
iii. set $\mathbf{x}_{m+1} = \mathbf{x}_m + \mathbf{h}_m$

(c) The converged value $\mathbf{x}^*$ is returned as the approximation root of $\mathbf{f}(\mathbf{x})$

3.2 Convergence of the Newton Method

The Newton-Kantorovich theorem explains the convergence of the newton method applied to the system of equations. It describes condition on the initial guess guaranteeing the convergence of the algorithm.

3.2.1 Setup of the Theorem.

1. Let $\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ be the system of equations to which the Newton method is supposed to be applied.
2. Let $||\cdot||$ be a norm on $\mathbb{R}^n$.
3. Let $X \subseteq \mathbb{R}^n$ be an open subset.
4. Assume that $\mathbf{f}$ is differentiable.
5. Assume that the Jacobian matrix $D\mathbf{f}(\mathbf{x})$ is locally Lipschitz, i.e. for any open subset $U \subseteq X$, there exists $L > 0$ such that $|||D\mathbf{f}(\mathbf{x}) - D\mathbf{f}(\mathbf{y})||| \leq L||\mathbf{x} - \mathbf{y}||$ holds for all $\mathbf{x}, \mathbf{y} \in U$.
6. Let $\mathbf{x_0} \in X$ be the initial guess.
7. Assume that $D\mathbf{f}(\mathbf{x_0})$ is invertible, i.e $D\mathbf{f}(\mathbf{x_0})^{-1}$ exists.
8. Let $\mathbf{h_0}$ be given as $\mathbf{h_0} = -D\mathbf{f}(\mathbf{x_0})^{-1}\mathbf{f}(\mathbf{x_0})$.
9. Let $\mathbf{x_1} := \mathbf{x_0} + \mathbf{h_0}$.
10. Assume that $B(\mathbf{x_1}, ||\mathbf{h_0}||) := \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x} - \mathbf{x_1}|| < ||\mathbf{h_0}||\} \subseteq X$. Notice that this set is open.

3.2.2 Statement of the Theorem.

If a setup as described above is given and if there exists a Lipschitz constant L on the ball $B(\mathbf{x_1}, ||\mathbf{h_0}||)$ such that $\alpha_0 := L \cdot |||D\mathbf{f}(\mathbf{x_0})^{-1}||| \cdot ||\mathbf{h_0}|| \leq \frac{1}{2}$, then it holds that

1. a solution $\mathbf{x}^*$ of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ exists in the close ball

   $\overline{B(\mathbf{x_1}, ||\mathbf{h_0}||)} := \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x} - \mathbf{x_1}|| \leq ||\mathbf{h_0}||\}$ and

2. the newton methods starting with the initial guess $\mathbf{x_0}$ converges to $\mathbf{x}^*$ with at least linear order of convergence, i.e. there exists $M > 0$ and $q \geq 1$ such that for sufficiently large k, the inequality $||\mathbf{x_k} - \mathbf{x}^*|| \leq M||\mathbf{x_{k+1}} - \mathbf{x}^*||^q$ holds.

If $\alpha_0 < \frac{1}{2}$, we have quadratic convergence, i.e. there exists $M > 0$ such that for sufficiently large k, the inequality $||\mathbf{x_k} - \mathbf{x}^*|| \leq M||\mathbf{x_{k+1}} - \mathbf{x}^*||^2$ holds [6].

3.3 Some Remarks

The theorem presented in the previous section shows that under certain conditions, the Newton method converges locally to the root of the system of equations and under certain conditions, we even get quadratic convergence.

3.3.1 Locally Lipschitz Condition.

For the function $f(x) = \frac{9}{5}x^{\frac{5}{3}}$, we get the derivative $f'(x) = 3x^{\frac{2}{3}}$. According to the Newton-Kantorovich theorem, we check the condition if the derivative f' is locally Lipschitz. Consider $\mathbb{R}$ and assume there is Lipschitz constant $L > 0$ such that

$|||f'(x) - f'(y)||| < L||x - y||$. However, if we consider $0 \leq x \leq \left(\frac{1}{L}\right)^3$ and choose $y = 0$, we obtain

$$|||f'(x) - f'(y)||| = |||3x^{\frac{2}{3}} - 0||| = |3x^{\frac{2}{3}}| = 3x^{\frac{2}{3}}$$

for the left hand side of the inequality and $L||x|| = L|x| = Lx$ for the right hand side.
To show that this condition does not hold, we need to compare the derivative such that $|||f'(x) - f'(y)||| < L||x - y||$ does not hold.
The derivative of the right hand side is $2x^{-\frac{1}{3}}$ which is monotonic decreasing and the derivative of the right and side is L which is constant. For $x = \left(\frac{1}{L}\right)^3$ we obtain

$$2\left(\left(\frac{1}{L}\right)^3\right)^{-\frac{1}{3}} = 2L \geq L$$

from which we can conclude that the derivative of the left hand side is greater than the derivative of the right hand side over the whole interval $0 \leq x \leq \left(\frac{1}{L}\right)^3$. Since for $x = 0$ the left hand side equals the right hand side, it follows that $|||f'(x) - f'(y)||| \geq L||x - y||$ contradicting our assumption. Hence, around $x = 0$, f' is not Lipschitz, i.e. f' is not locally Lipschitz. Considering the order of convergence, the calculation $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{3}{5}x_k = \frac{2}{5}x_k$ which implies linear convergence of the iteration. We conclude that in this case where the Lipschitz continuity does not hold, we do not have quadratic convergence.

3.3.2 Locality.

From the Newton-Kantorovich theorem, we conclude that the initial guess is essential for the convergence to the solution. If the provided guess that is sufficiently close to the root of the system, the Newton method will converge to it. However, if the initial guess is close to a critical point where the Jacobian matrix is singular, the Newton method will produce a "next guess" that is far away from the root of the system of equations. For instance, consider the problem we treat in 5.1.9. Here, the initial guess $[0, 0, 0.08, 0, 0]^T$ leads to convergence whereas we do not obtain convergence when we use the initial guess $[0, 0, 0.10, 0, 0]^T$.

Figure 3.1 shows the convergence of the Newton method applied on the system of equations in 5.1.9 when taking the initial guess $[0, 0, 0.08, 0, 0]^T$. Note that the Newton method stops at the latest after 15 iterations and that the tolerance is 10^{-6}.

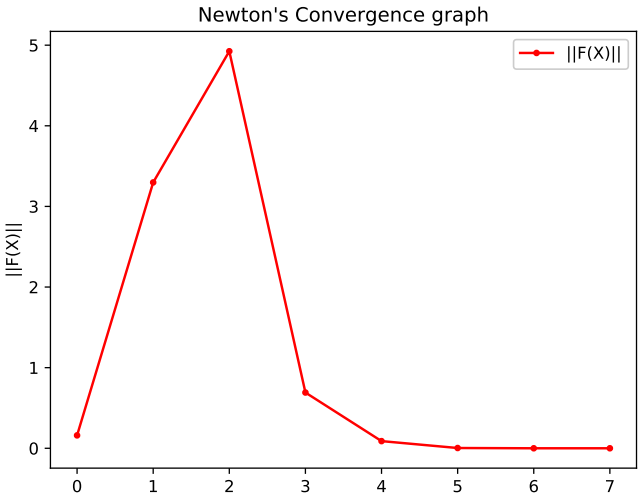


Figure 3.1: Convergence

Figure 3.2 shows the non-convergence for the same system of equations when taking the initial guess $[0, 0, 0.10, 0, 0]^T$.

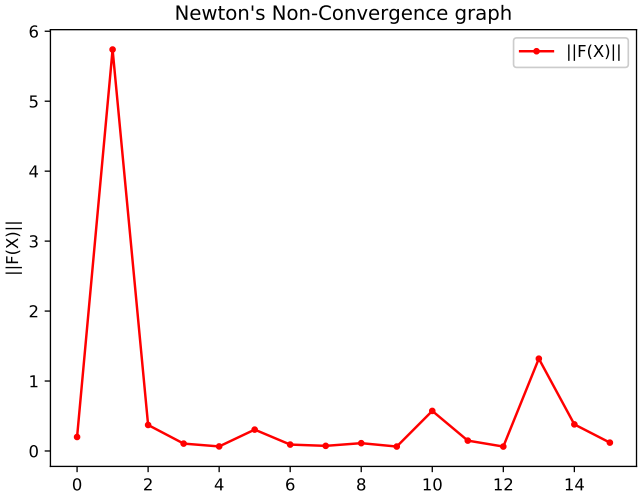


Figure 3.2: Non-Convergence

4. Fundamental Concepts

This chapter introduces some fundamental methods to approximate solutions of differential equations as well as the continuation method applied to solve nonlinear systems of equations. Based on these fundamental concepts, we will examine the numerical examples in (REFER TO CHAPTER 5).

[12]

4.1 Approximating Solutions for Differential Equations

Differential equations appear in several fields of science, Hence, solving them is an essential issue. In this section we explain some fundamental methods used to approximate solutions of differential equations.

4.1.1 Euler Method.

In mathematics and computational science, the Euler method (also called forward Euler method) is a method for solving first-order ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations. The Euler method is named after Leonhard Euler, who treated it in his book Institutionum calculi integralis [5]. The problem that is given reads as

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0. \end{cases}$$

For the Euler method, the time axis is discretized. One step of the Euler method from t_n to $t_{n+1} = t_n + h$[9] is

$$y_{n+1} = y_n + hf(t_n, y_n),$$

where $h > 0$ and $n \in \mathbb{N}_0$. The value of y_n is an approximation of the solution to the ODE at time t_n, i.e. $y_n \approx y(t_n)$. The Euler method is explicit, i.e. the solution y_{n+1} is an explicit function of y_i for $i \leq n$.

4.1.2 Converting a High Order Differential Equation to a First Order Differential Equation.

Any ODE of order N can be represented as a system of first-order ODEs. To treat the equation

$$y^{(N)}(t) = f(t, y(t), y'(t), ..., y^{(N-1)}(t)),$$

we introduce the vector

$$\mathbf{z}(t) := \begin{pmatrix} z_1(t) \\ \vdots \\ z_{N-1}(t) \\ z_N(t) \end{pmatrix} = \begin{pmatrix} y^{(1)}(t) \\ \vdots \\ y^{(N-1)}(t) \\ y^{(N)}(t) \end{pmatrix}.$$

The vector $\mathbf{z}'(t) := \begin{pmatrix} z_1'(t) \\ \vdots \\ z_{N-1}'(t) \\ z_N'(t) \end{pmatrix}$ can be formulated as

$$\mathbf{z}'(t) = \begin{pmatrix} y^{(1)}(t) \\ \vdots \\ y^{N-1}(t) \\ y^N(t) \end{pmatrix} = \begin{pmatrix} z_2(t) \\ \vdots \\ z_{N-1}(t) \\ f(t, z_1(t), ..., z_N(t)) \end{pmatrix} =: g(\mathbf{z}(t)).$$

This is a first-order system in the variable $\mathbf{z}(t)$ and can be handled by Euler's method or, in fact, by any other scheme for first-order systems[2].

4.1.3 Trapezoidal Rule.

The trapezoidal rule is a numerical method to solve ordinary differential equations where the main idea stems from computing integrals. The trapezoidal rule is an implicit method which can be considered as both a Runge-Kutta method and a linear multistep method. Suppose that we want to solve the differential equation

$$y' = f(t, y(t)). \tag{4.1.1}$$

The trapezoidal rule is given by the formula

$$y_{n+1} = y_n + \frac{1}{2}h(f(t_n, y_n) + f(t_{n+1}, y_{n+1})), \tag{4.1.2}$$

where h and t_n are given as in the previous subsection and $y_n$denotes $y(t_n)$.
Since the value y_{n+1} appears on both sides of the equation, this method is implicit. Integrating the $f(t, y(t))$ from t_n to t_{n+1} yields

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f(t, y(t))dt. \tag{4.1.3}$$

The trapezoidal rule states that the integral on the right-hand side can be approximated as

$$\int_{t_n}^{t_{n+1}} f(t, y(t))dt = \frac{1}{2}h(f(t_n, y_n) + f(t_{n+1}, y_{n+1})). \tag{4.1.4}$$

Substituting the right-hand side in (4.1.3) into (4.1.4), we obtain

$$y_{n+1} = y_n + \frac{1}{2}h(f(t_n, y_n) + f(t_{n+1}, y_{n+1})). \tag{4.1.5}$$

4.2 Continuation Method

The continuation method delivers a sequence of closely-spaced solutions to a given set of equations[4]. Hence, there are several fields where this method can be applied such as parameter dependent nonlinear equations or nonlinear boundary value problems of ordinary differential equations.
In the continuation method, the system $\mathbf{f}(\mathbf{x}) = 0$ is augmented to the system $\mathbf{g}(\mathbf{x}) = 0$ which reads as

$$\mathbf{g}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_{m-1}(\mathbf{x}) \\ x_p - \alpha \end{pmatrix} = \begin{pmatrix} f_1(x_1, x_2, \ldots, x_m) \\ f_2(x_1, x_2, \ldots, x_m) \\ \vdots \\ f_{m-1}(x_1, x_2, \ldots, x_m) \\ x_p - \alpha \end{pmatrix}, \tag{4.2.1}$$

where $p \in \{1, 2, ..., m\}$.
The corresponding Jacobian matrix to the system $g(\mathbf{x})$ is

$$D\mathbf{g}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\delta x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{m-1}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_{m-1}(\mathbf{x})}{\partial x_m} \\ \delta_{1,p} & \cdots & \delta_{m,p} \end{pmatrix}. \tag{4.2.2}$$

Once we apply the Newton method and obtain a solution $\mathbf{x_0}$ to the augmented system $g(\mathbf{x})$, we move along the tangent $\vec{\mathbf{t}}$ at this point with a step of size $h > 0$ according to the formula

$$\mathbf{x_1} = \mathbf{x_0} + h\vec{\mathbf{t}}.$$

This point is used as the next initial guess for the Newton method to obtain another solution to the system. If the step size h is small enough, $\mathbf{x}_0$ and the solution obtained by using $\mathbf{x}_1$ as initial guess will be close to each other. However, if h is too large, the Newton method may run into a point where the Jacobian matrix is singular. To overcome this problem, h can be adjusted. In addition, the value for p can be varied.
For the computation of the tangent vector, we use the fact that it is perpendicular to the gradient of $\mathbf{f}$:

$$\nabla\mathbf{f} \cdot \vec{\mathbf{t}} = f_{x_1}t_1 + f_{x_2}t_2 + \cdots + f_{x_{m-1}}t_{m-1} = 0.$$

If the jacobian matrix is not singular we can compute the solution of the linear system of the form

$$\begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\delta x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{m-1}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_{m-1}(\mathbf{x})}{\partial x_m} \\ \delta_{1,p} & \cdots & \delta_{m,p} \end{pmatrix} \cdot \begin{pmatrix} t_1 \\ \vdots \\ t_{m-1} \\ t_m \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}, \tag{4.2.3}$$

where we insert the value one for the last entry of the right hand side to guarantee that any tangent vector does not equal the zero vector.

4.2.1 Example: Unit Circle.

The following example to illustrate the continuation method and it is extracted from Burkhart's paper with the title "The Continuation Method for Algebraic Nonlinear Equations" [4]. Consider the equation for the unit circle is

$$x^2 + y^2 = 1 \tag{4.2.4}$$

Obviously, if you have one equation with 2 unknowns, it can't be solved uniquely either for x nor for y. Hence, the single equation is extended by an equation of the form

$$x = x_0$$

for some $-1 \leq x_0 \leq 1$. Now we arrange both equations in a single system

$$F(x,y) = \begin{pmatrix} x^2 + y^2 \\ x \end{pmatrix} = \begin{pmatrix} 1 \\ x_0 \end{pmatrix} \tag{4.2.5}$$

Next, we rewrite the nonlinear system in a homogeneous form

$$F(x,y) = \begin{pmatrix} x^2 + x^2 - 1 \\ x - x_0 \end{pmatrix} \tag{4.2.6}$$

considered for the special value $x_0 = 1/2$. By applying the Newton method for the systems of equations for 4.2.6 taking the initial guess

$$z^0 = \begin{pmatrix} x^0 \\ y^0 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -1 \end{pmatrix},$$

F reads as

$$F(0.5,-1) = \begin{pmatrix} \frac{1}{4} + 1 - 1 \\ 0.5 - 0.5 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ 0 \end{pmatrix}.$$

It can be verified that this system has a Jacobian of the form

$$J(x,y) = \begin{pmatrix} 2x & 2y \\ 1 & 0 \end{pmatrix}.$$

After 3 iterations, we obtain the solution

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -0.86603 \end{pmatrix}.$$

Now, we try the application of Newton's method for the starting guess

$$z^0 = \begin{pmatrix} x^0 \\ y^0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

When we try to apply Newton's method we experience difficulties. The reason is because, for this point (z^0) the Jacobian matrix

$$J(x^*,y^*) = \begin{pmatrix} 2 & 0 \\ 1 & 0 \end{pmatrix}$$

is singular.

This implies that Newton's method will break down at the point $x^* = [1,0.]$ while away from this point, 4.2.6 is a good method to use. A way out of this dilemma is to use a different nonlinear system for the computation. Hence, we consider the alternative augmented system

$$F(x,y) = \begin{pmatrix} x^2 + y^2 - 1 \\ y - y_0 \end{pmatrix}. \tag{4.2.7}$$

Note that this augmented system can again be iteratively solved. In particular, Points on the circle that are close to the turning point from above. Note that the Jacobian of system 4.2.7 is given by

$$J(x,y) = \begin{pmatrix} 2x & 2y \\ 0 & 1 \end{pmatrix} \tag{4.2.8}$$

while the Jacobian of system 4.2.6 is of the form

$$J(x,y) = \begin{pmatrix} 2x & 2y \\ 1 & 0 \end{pmatrix}. \tag{4.2.9}$$

It is straight forward to verify that the Jacobian in 4.2.6 is singular at $z = [0, \pm 1]$ while the Jacobian of 4.2.9 is singular at $z = [\pm 1, 0]$. Hence, in order to compute points on the unit circle close to the points $z = [\pm 1, 0]$, it is advisable to apply Newtons method for the augmented system 4.2.7 while for the computation of points close to $z = [0, \pm 1]$, the augmented system 4.2.6 is advised to be used. Hence, Newton's method for the computation of points of the circle 4.2.4 should be implemented in such a way that it can use either the augmented system 4.2.7 or augmented system 4.2.6. Moving along the tangent to get new initial guesses and switching between both presented systems appropriately delivers points on the unit circle as below.

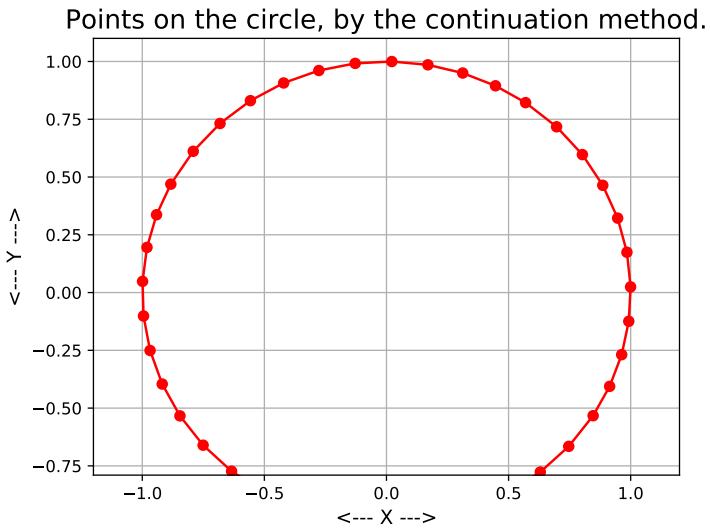


Figure 4.1: Result of Continuation

5. Numerical Examples

After having described the Newton method as well as the continuation method, we proceed with two numerical examples in this chapter. In the first section, we deal with a boundary value problem, namely the $Bratu\ problem$ and in the second section, we deal with a system of equations describing a chemical reaction.

5.1 Bratu Problem

The $Bratu\ problem$ is a boundary value problem and has a wide field of applications such as radiative heat transfer [11] or in nano-technology[3]. The problem reads as

$$u'' + \tau e^u = 0,\ for\ 0 \leq t \leq 1\ and\ \tau \neq 0, \tag{5.1.1}$$

where the boundary conditions are

$$u(0) = 0 \quad \& \quad u(1) = 0. \tag{5.1.2}$$

We can convert the second order ODE to a first order ODE in 5.1.1 by defining

$$x_1 := u$$

$$x_2 := u'$$

$$z := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

Then we obtain

$$f(z,\tau) := z' = \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} x_2 \\ -\tau e^{x_1} \end{pmatrix}.$$

This first orders system is discretized by
$0 = t_0 < t_1 < t_1 < \cdots < t_{Nt} = 1$ where $t_{j+1} - t_j = h = \frac{1}{Nt},\ j = 0, \ldots, Nt-1.$
The implicit trapezoidal method now reads as

$$y_{j+1} = y_j + \frac{h}{2}\{f(y_j,\tau) + f(y_{j+1},\tau)\},$$

where $y_j := \begin{pmatrix} x_1(t_j) \\ x_2(t_j) \end{pmatrix}$ and where $0 \leq j \leq Nt$.
The discretized boundary value can now be formulated as

$$\begin{cases} x_1(t_0) = 0 \\ x_1(t_{Nt}) = 0 \\ y_{j+1} = y_j + \frac{h}{2}\{f(y_j,\tau) + f(y_{j+1},\tau)\} \quad where \quad 0 \leq j < Nt. \end{cases} \tag{5.1.3}$$

For Nt=2, we get the discretization $t_0 = 0$,$t_1 = 0.5$, and $t_2 = 1$. By using the definition of $f(z, \tau)$, we get

$$\begin{cases} x_1(0) = 0 \\ x_1(1) = 0 \\ x_1(0.5) - x_1(0) - \frac{0.5}{2}\left(x_2(0) + x_2(0.5)\right) = 0 \\ x_2(0.5) - x_2(0) - \frac{0.5}{2}\left(-\tau exp(x_1(0)) - \tau exp(x_1(0.5))\right) = 0 \\ x_1(1) - x_1(0.5) - \frac{0.5}{2}\left(x_2(0.5) + x_2(1)\right) = 0 \\ x_2(1) - x_2(0.5) - \frac{0.5}{2}\left(-\tau exp(x_1(0.5)) - \tau exp(x_1(1))\right) = 0. \end{cases} \tag{5.1.4}$$

The corresponding Jacobian matrix has the form

$$D\mathbf{f}(\mathbf{x}, \tau) = \begin{bmatrix} \frac{\partial f_0}{\partial x_1(0)} & \frac{\partial f_0}{\partial x_2(0)} & \frac{\partial f_0}{\partial x_1(0.5)} & \frac{\partial f_0}{\partial x_2(0.5)} & \frac{\partial f_0}{\partial x_1(1)} & \frac{\partial f_0}{\partial x_2(1)} \\ \frac{\partial f_1}{\partial x_1(0)} & \frac{\partial f_1}{\partial x_2(0)} & \frac{\partial f_1}{\partial x_1(0.5)} & \frac{\partial f_1}{\partial x_2(0.5)} & \frac{\partial f_1}{\partial x_1(1)} & \frac{\partial f_1}{\partial x_2(1)} \\ \frac{\partial f_2}{\partial x_1(0)} & \frac{\partial f_2}{\partial x_2(0)} & \frac{\partial f_2}{\partial x_1(0.5)} & \frac{\partial f_2}{\partial x_2(0.5)} & \frac{\partial f_2}{\partial x_1(1)} & \frac{\partial f_2}{\partial x_2(1)} \\ \frac{\partial f_3}{\partial x_1(0)} & \frac{\partial f_3}{\partial x_2(0)} & \frac{\partial f_3}{\partial x_1(0.5)} & \frac{\partial f_3}{\partial x_2(0.5)} & \frac{\partial f_3}{\partial x_1(1)} & \frac{\partial f_3}{\partial x_2(1)} \\ \frac{\partial f_4}{\partial x_1(0)} & \frac{\partial f_4}{\partial x_2(0)} & \frac{\partial f_4}{\partial x_1(0.5)} & \frac{\partial f_4}{\partial x_2(0.5)} & \frac{\partial f_4}{\partial x_1(1)} & \frac{\partial f_4}{\partial x_2(1)} \\ \frac{\partial f_5}{\partial x_1(0)} & \frac{\partial f_5}{\partial x_2(0)} & \frac{\partial f_5}{\partial x_1(0.5)} & \frac{\partial f_5}{\partial x_2(0.5)} & \frac{\partial f_5}{\partial x_1(1)} & \frac{\partial f_5}{\partial x_2(1)} \end{bmatrix}. \tag{5.1.5}$$

Evaluating the above expressions yields the matrix

$$D\mathbf{f}(\mathbf{x}, \tau) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & -\frac{0.5}{2} & 1 & -\frac{0.5}{2} & 0 & 0 \\ \frac{0.5}{2}\tau exp(x_1(0)) & -1 & \frac{0.5}{2}\tau exp(x_1(0.5)) & 1 & 0 & 0 \\ 0 & 0 & -1 & -\frac{0.5}{2} & 1 & -\frac{0.5}{2} \\ 0 & 0 & \frac{0.5}{2}\tau exp(x_1(0.5)) & -1 & \frac{0.5}{2}\tau exp(x_1(1)) & 1 \end{bmatrix}. \tag{5.1.6}$$

5.1.1 Augmented System of Equations.

We can augment the system 5.1.6 by one of the following equations:

$$\begin{cases} \tau - \alpha = 0 \\ or \\ x_1(0.5) - \alpha = 0. \end{cases} \tag{5.1.7}$$

Note that τ is a parameter in the given system of equations whereas α is a quantity which is dynamically adjusted during the continuation. This augmentation leads to a quadratic Jacobian matrix such that the Newton method can be applied. The Jacobian matrix of the augmented system reads as

$$D\mathbf{f}(\mathbf{x}, \tau) = \begin{bmatrix} \frac{\partial f_0}{\partial x_1(0)} & \frac{\partial f_0}{\partial x_2(0)} & \frac{\partial f_0}{\partial x_1(0.5)} & \frac{\partial f_0}{\partial x_2(0.5)} & \frac{\partial f_0}{\partial x_1(1)} & \frac{\partial f_0}{\partial x_2(1)} & \frac{\partial f_0}{\partial \tau} \\ \frac{\partial f_1}{\partial x_1(0)} & \frac{\partial f_1}{\partial x_2(0)} & \frac{\partial f_1}{\partial x_1(0.5)} & \frac{\partial f_1}{\partial x_2(0.5)} & \frac{\partial f_1}{\partial x_1(1)} & \frac{\partial f_1}{\partial x_2(1)} & \frac{\partial f_1}{\partial \tau} \\ \frac{\partial f_2}{\partial x_1(0)} & \frac{\partial f_2}{\partial x_2(0)} & \frac{\partial f_2}{\partial x_1(0.5)} & \frac{\partial f_2}{\partial x_2(0.5)} & \frac{\partial f_2}{\partial x_1(1)} & \frac{\partial f_2}{\partial x_2(1)} & \frac{\partial f_2}{\partial \tau} \\ \frac{\partial f_3}{\partial x_1(0)} & \frac{\partial f_3}{\partial x_2(0)} & \frac{\partial f_3}{\partial x_1(0.5)} & \frac{\partial f_3}{\partial x_2(0.5)} & \frac{\partial f_3}{\partial x_1(1)} & \frac{\partial f_3}{\partial x_2(1)} & \frac{\partial f_3}{\partial \tau} \\ \frac{\partial f_4}{\partial x_1(0)} & \frac{\partial f_4}{\partial x_2(0)} & \frac{\partial f_4}{\partial x_1(0.5)} & \frac{\partial f_4}{\partial x_2(0.5)} & \frac{\partial f_4}{\partial x_1(1)} & \frac{\partial f_4}{\partial x_2(1)} & \frac{\partial f_4}{\partial \tau} \\ \frac{\partial f_5}{\partial x_1(0)} & \frac{\partial f_5}{\partial x_2(0)} & \frac{\partial f_5}{\partial x_1(0.5)} & \frac{\partial f_5}{\partial x_2(0.5)} & \frac{\partial f_5}{\partial x_1(1)} & \frac{\partial f_5}{\partial x_2(1)} & \frac{\partial f_5}{\partial \tau} \\ \frac{\partial f_6}{\partial x_1(0)} & \frac{\partial f_6}{\partial x_2(0)} & \frac{\partial f_6}{\partial x_1(0.5)} & \frac{\partial f_6}{\partial x_2(0.5)} & \frac{\partial f_6}{\partial x_1(1)} & \frac{\partial f_6}{\partial x_2(1)} & \frac{\partial f_6}{\partial \tau} \end{bmatrix}, \tag{5.1.8}$$

where f_6 refers to one of the equations in 5.1.7 which is equivalent to

$$D\mathbf{f}(\mathbf{x},\tau) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & -\frac{0.5}{2} & 1 & -\frac{0.5}{2} & 0 & 0 & 0 \\ \frac{0.5}{2}\tau exp(x_1(0)) & -1 & \frac{0.5}{2}\tau exp(x_1(0.5)) & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -\frac{0.5}{2} & 1 & -\frac{0.5}{2} & 0 \\ 0 & 0 & \frac{0.5}{2}\tau exp(x_1(0.5)) & -1 & \frac{0.5}{2}\tau exp(x_1(1)) & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

if we have the constraint $f_6 = \tau - \alpha$. If we choose the constraint $f_6 = x_1(0.5) - \alpha$, the position of the value 1 in the last row changes and the Jacobian matrix becomes

$$D\mathbf{f}(\mathbf{x},\tau) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & -\frac{0.5}{2} & 1 & -\frac{0.5}{2} & 0 & 0 & 0 \\ \frac{0.5}{2}\tau exp(x_1(0)) & -1 & \frac{0.5}{2}\tau exp(x_1(0.5)) & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -\frac{0.5}{2} & 1 & -\frac{0.5}{2} & 0 \\ 0 & 0 & \frac{0.5}{2}\tau exp(x_1(0.5)) & -1 & \frac{0.5}{2}\tau exp(x_1(1)) & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

5.1.2 Application of the Continuation Method.

For this numerical example, we choose $f_6 = x_1\left(t_{Nt/2}\right) - \alpha$ as the additional function and proceed according to the steps below:

1. Start with $\alpha = 10^{-4}$.
2. Apply the Newton method to the augmented system of equations which yields a solution for u where its maximum has the value of α.
3. Move along the tangent to get the new initial guess with a step size of $h = 0.5$. From this initial guess, read the value for $x_1(t_{Nt/2})$ and set this value for α.
4. Apply the Newton method with the new initial guess and the new alpha obtained in the previous step.
5. Repeat steps 2 to 5.

For the dynamical update of the additional constraint which was also performed in this project see the corresponding code in the Appendix.

5.1.3 Results.

The figures below show the results obtained from the continuation method as described in the previous subsection. Figure (5.1) depicts the plot of the u_{max} against τ where $Nt = 2$ whereas figure (5.2) shows the cases for different values of Nt. Concretely, each point in the curves is the pair of u_{max} corresponding to $x_1(t_{Nt/2})$ and τ obtained through the Newton method starting with a certain initial guess. The initial guesses are updated by following the tangent as described in the previous subsection. We observe that for almost each τ, we obtain two values u_{max}, i.e. for almost each τ, we get two

solutions u of the Bratu problem. The only value for τ where we have one solution is the turning point where the Jacobian matrix is singular.
We observe that after the turning point, the curve increases exponentially. In particular, τ converges towards 0 and u_{max} is unbounded on the upper part of the curve.
The second figure shows that as Nt increases, the turning point moves to the left. However, in any choice of Nt, the curves show the same properties as described in the case of $Nt = 2$. In A, we illustrate the meaning of Nt. Concretely, for different values of Nt, we get different solutions of u and hence, different values for u_{max}. Increasing Nt leads to smoother solutions.

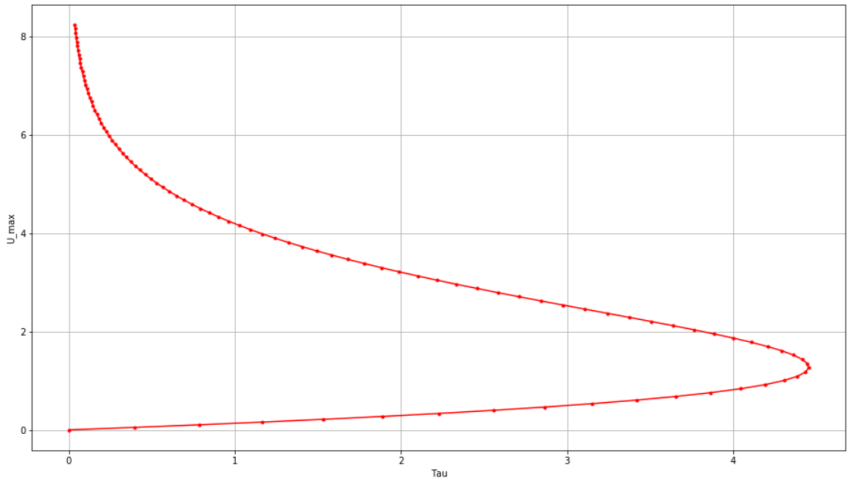


Figure 5.1: Bratu Problem

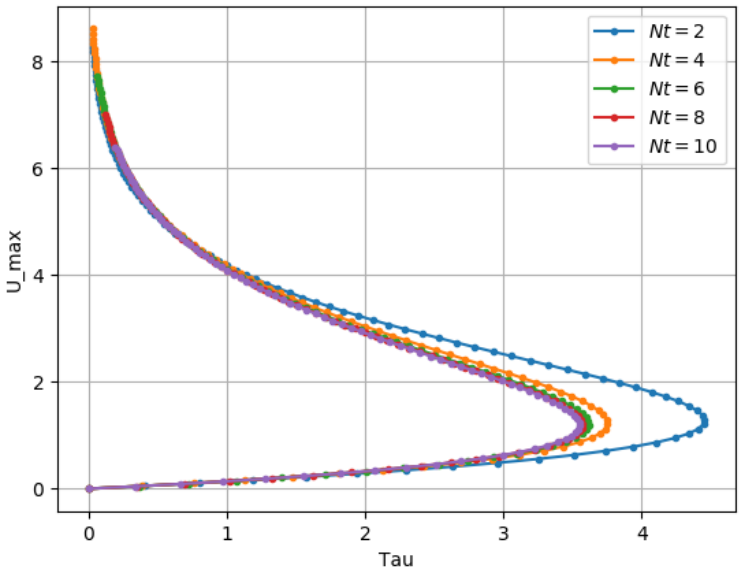


Figure 5.2: Bratu Problem

5.1.4 Chemical Reaction.

Chemical reactions that can be described by nonlinear equations have been studied experimentally by many researchers [8]. The system of nonlinear equations that we deal with in this section indicates a chemical reagent $\mathbf{x}$ to the medium of the chemical reaction and the change of its concentration in this medium. These equations present interesting mathematical properties (stability and turning point). The problem reads as

$$\begin{cases} f_0 &= \alpha(1-x_3)exp\left(\frac{10x_1}{1+\frac{10x_1}{\gamma}}\right) - x_3 = 0 \\ f_1 &= \alpha B(1-x_3)exp\left(\frac{10x_1}{1+\frac{10x_1}{\gamma}}\right) + \beta_1\theta_{c_1} - 10(1+\beta_1)x_1 = 0 \\ f_2 &= x_3 - x_4 + \alpha(1-x_4)exp\left(\frac{10x_2}{1+\frac{10x_2}{\gamma}}\right) = 0 \\ f_3 &= 10x_1 - 10(1+\beta_2)x_2 + \alpha B(1-x_4)exp\left(\frac{10x_2}{1+\frac{10x_2}{\gamma}}\right) + \beta_2\theta_{c2} = 0 \end{cases} \tag{5.1.9}$$

where $\gamma, B, \beta_1, \beta_2, \theta_{c_1}, \theta_{c_2}$ and α are physical parameters such that $\gamma = 1000, B = 22, \beta_1 = \beta_2 = 2, \theta_1 = \theta_2 = 0, x_0 = \{0,0,0,0,0\}$.
The corresponding Jacobian matrix has the form

$$D\mathbf{f}(\mathbf{x},\tau) = \begin{bmatrix} \frac{\partial f_0}{\partial x_1} & \frac{\partial f_0}{\partial x_2} & \frac{\partial f_0}{\partial x_3} & \frac{\partial f_0}{\partial x_4} \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \end{bmatrix}. \tag{5.1.10}$$

Evaluating the above expressions yields the matrix

$$D\mathbf{f}(\mathbf{x},\tau) = \begin{bmatrix} \alpha(1-x_3)E_1E_3 & 0 & -\alpha E_1 - 1 & 0 \\ \alpha B(1-x_3)E_1E_3 - 10(1+\beta_1) & 0 & -\alpha BE_1 & 0 \\ 0 & \alpha(1-x_3)E_2E_4 & 1 & -1-\alpha E_2 \\ 10 & -10(1+\beta_2) + \alpha B(1-x_4)E_2 * E_4 & 0 & -\alpha BE_2 \end{bmatrix},$$

where

$$\begin{cases} E_1 = \exp(10x_1/(1+10x_1/\gamma)) \\ E_2 = \exp(10x_2/(1+10x_2/\gamma)) \\ E_3 = ((1+10x_1/\gamma)10 - 10x_1(10/\gamma))/(1+10x_1/\gamma)^2 \\ E_4 = ((1+10x_2/\gamma) * 10 - 10x_2(10/\gamma))/(1+10x_2/\gamma)^2. \end{cases} \tag{5.1.11}$$

5.1.5 Augmented System of Equation.

Define $x_5 := \alpha$. We can augment the system 5.1.9 by introducing the equation $x_p - \tau = 0$, where p ranges from 1 to 5.
Note that α is a parameter in the given system of equations whereas τ is a quantity which is dynamically adjusted during the continuation. This augmentation leads to a quadratic Jacobian matrix such that the Newton method can be applied.

In order to compute new solutions, it is essential to stay close to the solution curve corresponding to the system of equations. By choosing the step size h to be small, we ensure that we do not deviate too much from the solution curve. If the step size h is large, the system may not converge or it may take

a long period to converge. Moving along the tangent to get new initial guesses and switching between the different constraints appropriately makes it possible to move beyond the turning points.

The corresponding Jacobian matrix has the form

$$D\mathbf{f}(\mathbf{x},\tau) = \begin{bmatrix} \frac{\partial f_0}{\partial x_1} & \frac{\partial f_0}{\partial x_2} & \frac{\partial f_0}{\partial x_3} & \frac{\partial f_0}{\partial x_4} & \frac{\partial f_0}{\partial x_5} \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} & \frac{\partial f_1}{\partial x_5} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} & \frac{\partial f_2}{\partial x_5} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} & \frac{\partial f_3}{\partial x_5} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} & \frac{\partial f_4}{\partial x_5} \end{bmatrix}. \tag{5.1.12}$$

Evaluating the expressions from above yields the matrix

$$D\mathbf{f}(\mathbf{x},\tau) = \begin{bmatrix} \alpha(1-x_3)E_1E_3 & 0 & -\alpha E_1 - 1 & 0 & (1-x_3)E_1 \\ \alpha B(1-x_3)E_1E_3 - 10(1+\beta_1) & 0 & -\alpha BE_1 & 0 & B(1-x_3)E_1 \\ 0 & \alpha(1-x_3)E_2E_4 & 1 & -1-\alpha E_2 & (1-x_3)E_2 \\ 10 & -10(1+\beta_2)+\alpha B(1-x_4)E_2 * E_4 & 0 & -\alpha BE_2 & B(1-x_4)E_2 \\ \delta_{1,p} & \delta_{2,p} & \delta_{3,p} & \delta_{4,p} & \delta_{5,p} \end{bmatrix},$$

where E_1, E_2, E_3 and E_4 are as in 5.1.11.

5.1.6 Application of Continuation Method.

For this numerical example, we choose $f_4 = x_p - \tau$ where $1 \leq p \leq 5$. The procedure that we follow is described as follows:

1. Start with $\tau = 0$ and $1 \leq p \leq 5$.
2. Apply the Newton method to the augmented system of equations which yields a solution for the augmented system of equations.
3. Move along the tangent to get the new initial guess with a step size of $h = 0.02$.
4. Set the index of the entry of the tangent which has the highest absolute value to be equal to p.
5. From this initial guess, read the value for x_p and set this value for τ.
6. Apply the Newton method with the new initial guess and the new τ obtained in the previous step.
7. Repeat steps 2 to 6.

5.1.7 Results.

The figures below show the results obtained from the continuation method as described in the previous subsection. Concretely, each point in the curves is the pair of α corresponding to x_5 and x_2 obtained through the Newton method starting with a certain initial guess. The initial guesses are updated by following the tangent as described in the previous subsection. We observe that we have several solutions where the pairs of α and x_2 differ from each other. We observed that we also have six turning points in the curve where we plot α against x_2. We observe that after the point around the pair $\begin{pmatrix} x_2 \\ \alpha \end{pmatrix} \approx \begin{pmatrix} 0.012 \\ 0.22 \end{pmatrix}$, the curve becomes stable. In particular, α converges towards 0.25 whereas x_2 is unbounded.

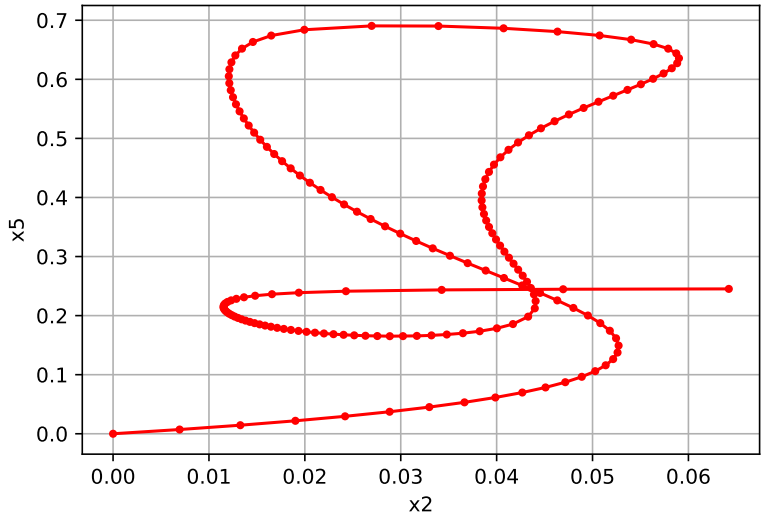


Figure 5.3: Chemical Reaction

6. Conclusion

In this project, we studied the theoretical background of the Newton method. In particular, we focused on its application to systems of equations. Here, the Newton-Kantorovich theorem was explained to show the conditions and properties of the convergence of this method.
With the python codes that we developed, we solved two numerical examples, namely the $Bratu\ problem$ and a system of equations describing a chemical reaction. We adjusted the codes such that p was changed dynamically according to the tangent vector.
The examination of the obtained results showed that we could move beyond the turning points with the implemented continuation methods in both numerical examples.
Further research can be conducted in python by considering different problems and by using the implementations developed in this project.

Appendix A. Some additional data

A.1 Bratu Problem

Here we provided the explicit expressions for the system of the equations corresponding to the Bratu problem for the cases Nt=4 and Nt=8.

For $Nt = 4$,we get the discretization $t_0 = 0$,$t_1 = 0.25$, $t_2 = 0.5$,$t_3 = 0.75$ and $t_4 = 1$.
By using the definition of $f(z, \tau)$ we get

$$\begin{cases} x_1(0) = 0 \\ x_1(1) = 0 \\ x_1(0.25) - x_1(0) - \frac{0.25}{2}(x_2(0) + x_2(0.25)) = 0 \\ x_2(0.25) - x_2(0) - \frac{0.25}{2}(-\tau exp(x_1(0)) - \tau exp(x_1(0.25))) = 0 \\ x_1(0.5) - x_1(0.25) - \frac{0.25}{2}(x_2(0.25) + x_2(0.5) = 0 \\ x_2(0.5) - x_2(0.25) - \frac{0.25}{2}(-\tau exp(x_1(0.25)) - \tau exp(x_1(0.5))) = 0 \\ x_1(0.75) - x_1(0.5) - \frac{0.25}{2}(x_2(0.5) + x_2(0.75)) = 0 \\ x_2(0.75) - x_2(0.5) - \frac{0.25}{2}(-\tau exp(x_1(0.5)) - \tau exp(x_1(0.75))) = 0 \\ x_1(1) - x_1(0.75) - \frac{0.25}{2}(x_2(0.75) + x_2(1)) = 0 \\ x_2(1) - x_2(0.75) - \frac{0.25}{2}(-\tau exp(x_1(0.75)) - \tau exp(x_1(1))) = 0 \end{cases} \tag{A.1.1}$$

If we consider τ as variable will yield

$$\begin{cases} x_1(0) = 0 \\ x_1(1) = 0 \\ x_1(0.25) - x_1(0) - \frac{0.25}{2}(x_2(0) + x_2(0.25)) = 0 \\ x_2(0.25) - x_2(0) - \frac{0.25}{2}(-\tau exp(x_1(0)) - \tau exp(x_1(0.25))) = 0 \\ x_1(0.5) - x_1(0.25) - \frac{0.25}{2}(x_2(0.25) + x_2(0.5) = 0 \\ x_2(0.5) - x_2(0.25) - \frac{0.25}{2}(-\tau exp(x_1(0.25)) - \tau exp(x_1(0.5))) = 0 \\ x_1(0.75) - x_1(0.5) - \frac{0.25}{2}(x_2(0.5) + x_2(0.75)) = 0 \\ x_2(0.75) - x_2(0.5) - \frac{0.25}{2}(-\tau exp(x_1(0.5)) - \tau exp(x_1(0.75))) = 0 \\ x_1(1) - x_1(0.75) - \frac{0.25}{2}(x_2(0.75) + x_2(1)) = 0 \\ x_2(1) - x_2(0.75) - \frac{0.25}{2}(-\tau exp(x_1(0.75)) - \tau exp(x_1(1))) = 0 \\ \tau - \alpha = 0 \end{cases} \tag{A.1.2}$$

For $Nt = 8$, $t_0 = 0$, $t_1 = 0.125$, $t_2 = 0.25$, $t_3 = 0.375$,$t_4 = 0.5$, $t_5 = 0.625$, $t_6 = 0.75$, $t_7 = 0.875$ and $t_8 = 1$.
By using the definition of $f(z, \tau)$ we get

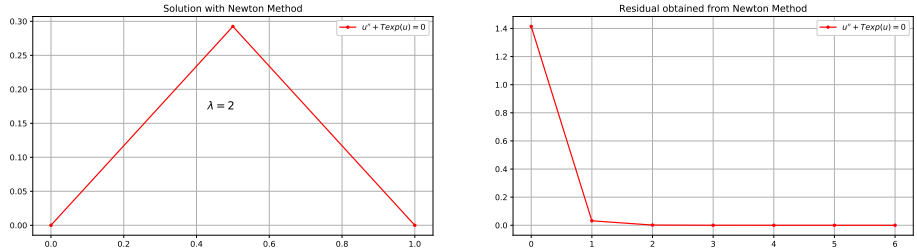
$$\begin{cases} x_1(0) = 0 \\ x_1(1) = 0 \\ x_1(0.125) - x_1(0) - \frac{0.125}{2}(x_2(0) + x_2(0.125)) = 0 \\ x_2(0.125) - x_2(0) - \frac{0.125}{2}(-\tau exp(x_1(0)) - \tau exp(x_1(x_1(0.125))) = 0 \\ x_1(0.25) - x_1(0.125) - \frac{0.125}{2}(x_2(0.125) + x_2(0.25)) = 0 \\ x_2(0.25) - x_2(0.125) - \frac{0.125}{2}(-\tau exp(x_1(0.125)) - \tau exp(x_1(0.25))) = 0 \\ x_1(0.375) - x_1(0.25) - \frac{0.125}{2}(x_2(0.25) + x_2(0.375)) = 0 \\ x_2(0.375) - x_2(0.25) - \frac{0.125}{2}(-\tau exp(x_1(0.25)) - \tau exp(x_1(0.375))) = 0 \\ x_1(0.5) - x_1(0.375) - \frac{0.125}{2}(x_2(0.375) + x_2(0.5)) = 0 \\ x_2(0.5) - x_2(0.375) - \frac{0.125}{2}(-\tau exp(x_1(0.375)) - \tau exp(x_1(0.5))) = 0 \\ x_1(0.625) - x_1(0.5) - \frac{0.125}{2}(x_2(0.5) + x_2(0.625)) = 0 \\ x_2(0.625) - x_2(0.5) - \frac{0.125}{2}(-\tau exp(x_1(0.5)) - \tau exp(x_1(0.625)) = 0 \\ x_1(0.75) - x_1(0.625) - \frac{0.125}{2}(x_2(0.625) + x_2(0.75)) = 0 \\ x_2(0.75) - x_2(0.625) - \frac{0.125}{2}(-\tau exp(x_1(0.625)) - \tau exp(x_1(0.75))) = 0 \\ x_1(0.875) - x_1(0.75) - \frac{0.125}{2}(x_2(0.75) + x_2(0.875)) = 0 \\ x_2(0.875) - x_2(0.75) - \frac{0.125}{2}(-\tau exp(x_1(0.75)) - \tau exp(x_1(0.875))) = 0 \\ x_1(1) - x_1(0.875) - \frac{0.125}{2}(x_2(0.875) + x_2(1)) = 0 \\ x_2(1) - x_2(0.875) - \frac{0.125}{2}(-\tau exp(x_1(0.875)) - \tau exp(x_1(1))) = 0 \end{cases} \tag{A.1.3}$$

If we consider τ as a variable

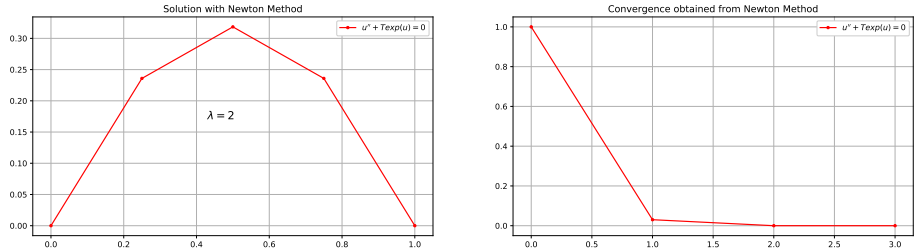
$$\begin{cases} x_1(0) = 0 \\ x_1(1) = 0 \\ x_1(0.125) - x_1(0) - \frac{0.125}{2}(x_2(0) + x_2(0.125)) = 0 \\ x_2(0.125) - x_2(0) - \frac{0.125}{2}(-\tau exp(x_1(0)) - \tau exp(x_1(x_1(0.125))) = 0 \\ x_1(0.25) - x_1(0.125) - \frac{0.125}{2}(x_2(0.125) + x_2(0.25)) = 0 \\ x_2(0.25) - x_2(0.125) - \frac{0.125}{2}(-\tau exp(x_1(0.125)) - \tau exp(x_1(0.25))) = 0 \\ x_1(0.375) - x_1(0.25) - \frac{0.125}{2}(x_2(0.25) + x_2(0.375)) = 0 \\ x_2(0.375) - x_2(0.25) - \frac{0.125}{2}(-\tau exp(x_1(0.25)) - \tau exp(x_1(0.375))) = 0 \\ x_1(0.5) - x_1(0.375) - \frac{0.125}{2}(x_2(0.375) + x_2(0.5)) = 0 \\ x_2(0.5) - x_2(0.375) - \frac{0.125}{2}(-\tau exp(x_1(0.375)) - \tau exp(x_1(0.5))) = 0 \\ x_1(0.625) - x_1(0.5) - \frac{0.125}{2}(x_2(0.5) + x_2(0.625)) = 0 \\ x_2(0.625) - x_2(0.5) - \frac{0.125}{2}(-\tau exp(x_1(0.5)) - \tau exp(x_1(0.625)) = 0 \\ x_1(0.75) - x_1(0.625) - \frac{0.125}{2}(x_2(0.625) + x_2(0.75)) = 0 \\ x_2(0.75) - x_2(0.625) - \frac{0.125}{2}(-\tau exp(x_1(0.625)) - \tau exp(x_1(0.75))) = 0 \\ x_1(0.875) - x_1(0.75) - \frac{0.125}{2}(x_2(0.75) + x_2(0.875)) = 0 \\ x_2(0.875) - x_2(0.75) - \frac{0.125}{2}(-\tau exp(x_1(0.75)) - \tau exp(x_1(0.875))) = 0 \\ x_1(1) - x_1(0.875) - \frac{0.125}{2}(x_2(0.875) + x_2(1)) = 0 \\ x_2(1) - x_2(0.875) - \frac{0.125}{2}(-\tau exp(x_1(0.875)) - \tau exp(x_1(1))) = 0 \\ \tau - \alpha = 0 \end{cases} \tag{A.1.4}$$

A.2 Some Additional Simulations for the Bratu Problem

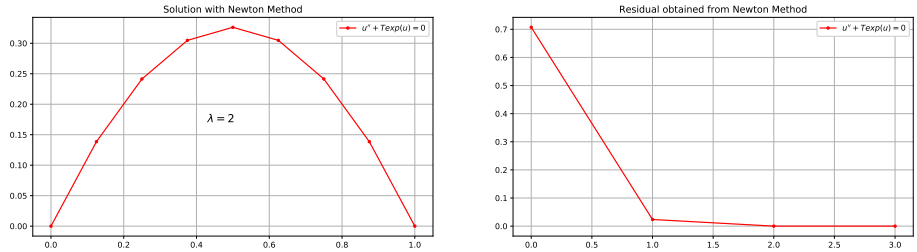
Here some addition simulation to illustrate the solutions of u for different values of Nt. Note that $m = 10$ (highest number of iterations) and that the tolerance is 10^{-7}



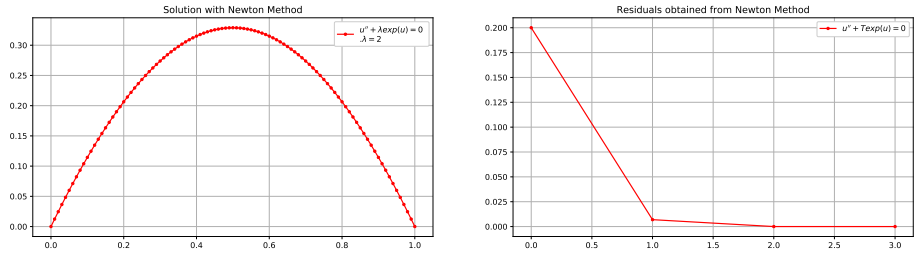
(a) The left graph shows the solution of u for $Nt = 2$ and the right one shows the residuals obtained during the Newton method.



(b) The left graph shows the solution of u for $Nt = 4$ and the right one shows the residuals obtained during the Newton method.



(c) The left graph shows the solution of u for $Nt = 8$ and the right one shows the residuals obtained during the Newton method.



(d) The left graph shows the solution of u for $Nt = 100$ and the right one shows the residuals obtained during the Newton method.

Algorithm 0: Newton's method for Bratu Problem

```
import numpy as np
from scipy.linalg import solve, norm
import matplotlib.pyplot as plt
def newtonBratu(f, df, Nt, m, tol, x0, p):
    x = x0
    c = [ ]
    alpha = x0[p]
for i in range(0, m): do
    fnorm=norm(f(x0, Nt, alpha, p), 2)
    c.append(fnorm)
    if fnorm<tol: break then
        else:
            s = solve(df(x0, Nt, p), -f(x0, Nt, alpha, p))
            x = x0 + s
            x0 = x
    end
    return x, c, p
end
```

Algorithm 1: Tangent for Bratu Problem

```
import numpy as np
from scipy.linalg import solve, norm
import matplotlib.pyplot as plt
def tangent (Nt, x, tanVecprevious, p, df2):
    x0= x
    df= df2(x0, Nt, p)
    b= np.zeros(len(df))
    b[len(b) - 1]= 1
    tanVec = solve(df, b)
    tanVec = tanVec/norm(tanVec, 2)
if (np.dot(tanVec, tanVecprevious) < 0): then
    tanVec = tanVec * (-1)
    pnew = np.argmax(tanVec)
end
    return tanVec, pnew
```

Algorithm 2: Stepsize for Bratu Problem

```
import numpy as np
from scipy.linalg import solve, norm
import matplotlib.pyplot as plt
from tangent import tangent
def step(Nt, sol, tanVecprevious, p, h, df2):
    tan = tangent(Nt, sol, tanVecprevious, p, df2)
    tanVec = tan[0]
    pnew = tan[1]
    xnewguess = sol + h * tanVec
  return xnewguess, pnew, tanVec
```

Algorithm 3: Continuation for Bratu

```
import numpy as np
from scipy.linalg import solve, norm
import matplotlib.pyplot as plt
from newton import newtonBratu
from tangent import tangent
from step import step
def dfNt(x, Nt, p):
df = np.zeros((2 * Nt + 2 + 1,2 * Nt + 2 + 1))
h = (1./Nt)/2
df[0, 0]= 1
df[1, 2 * Nt]= 1
df[len(df) - 1, p]= 1
for i in range(2 * Nt):
  j = i + 2 do
    if (j % 2 == 0): then
        df[j, j - 2] = -1
        df[j, j - 2 + 1] = -h
        df[j, j - 2 + 2] = 1
        df[j, j - 2 + 3] = -h
      else:
        df[j, j - 3] = h * x[len(x)-1] * np.exp(x[j - 3])
        df[j, j - 3 + 1] = -1
        df[j, j - 3 + 2] = h * x[len(x) - 1] * np.exp(x[j - 1])
        df[j,j - 3 + 3] = 1
        df[j, len(df) - 1] = h * (np.exp(x[j - 3])+np.exp(x[j - 1]))
    end
    return df
end
def fNt(x, Nt, alpha, p):
f = np.zeros(2 * Nt + 2 + 1)
h = (1./Nt)/2
f[0]= x[0]
f[1]= x[len(f) - 3]
f[len(f) - 1] = x[p] - alpha
for i in range(2 * Nt):
j = i + 2 do
    if (j % 2 == 0): then
        f[j] = x[j] - x[j - 2] - h * (x[j - 1] + x[j + 1])
      else:
        f[j] = x[j] - x[j - 2] + h * x[len(x) - 1] * (np.exp(x[j - 3]) + np.exp(x[j - 1]))
    end
    return f
end
```

Algorithm 4: Continuation for Bratu Problem

```
def Continuation(numIter, f, df, Nt, m, tol, x0, p, tanVecprevious, h):
retTau=[]
retPeak=[]
for i in range(numIter): do
    sol = newtonBratu(f, df, Nt, m, tol, x0, p)
    tan = tangent (Nt, sol[0], tanVecprevious, p, df)
    x01 = sol[0] + h * tan[0]
    x0 = x01
    p = tan[1]
    retTau.append(sol[0][len(sol[0]) - 1])
    retPeak.append(sol[0][2])
    tanVecprevious = tan[0]
    return retTau, retPeak
end
tol = 1.e-5
m = 20
p = 2
continuationit = 100
h = 0.5
for i in range(2, 11): do
    Nt= i
    x0= np.zeros(2 * Nt + 2 + 1)
    tanVecprevious= np.ones(2 *Nt + 2 + 1)
    a= Continuation(continuationit, fNt, dfNt, Nt, m, tol, x0, p, tanVecprevious, h)
    plt.plot(a[0], a[1], ".-", label="Nt =" + str(Nt))
    plt.xlabel("Tau")
    plt.ylabel("Umax")
    plt.grid()
    plt.legend()
    plt.savefig("Bratu-Problem.Pdf")
end
plt.show()
```

Algorithm 5: Newton for Chemical Reaction

```
import numpy as np
from scipy.linalg import solve, norm
def newtonChemical(f, df, m, tol, x0, p):
x= x0
n= len(x0)
alpha= x0[p]
for i in range(0,m): do
    fnorm = norm(f(n, x0, alpha, p),2)
    if (fnorm<tol): break then
        else:
            s= solve(df(n, x0, p), -f(n, x0, alpha, p))
            x= x0 + s
            x0= x
    end
    return x, p
end
```

Algorithm 6: Tangent for Chemical Reaction

```
import numpy as np
from scipy.linalg import solve, norm
def tangent (x, tanVecprevious, p, df):
    x0= x
    n= len(x0)
    df= df(n, x0, p)
    b= np.zeros(len(df))
    b[len(b)-1] = 1
    tanVec = solve(df, b)
    tanVec = tanVec/norm(tanVec,2)
if (np.dot(tanVec, tanVecprevious) < 0): then
    tanVec = tanVec * (-1)
end
   pnew = np.argmax(abs(tanVec))
   return tanVec,pnew
```

Algorithm 7: continuation for Chemical Reaction

```
import numpy as np
from scipy.linalg import solve, norm
import matplotlib.pyplot as plt
from newton import newtonChemical
from tangent import tangent
def fcn(n, u, alpha, p):
    f = np.zeros(n)
    E1 = np.exp(10 * u[0]/(1. + 0.01 * u[0]))
    E2 = np.exp(10 * u[1]/(1. + 0.01 * u[1]))
    f[0] = u[4] * (1. - u[2]) * E1 - u[2]
    f[1] = 22 * u[4] * (1. - u[2]) * E1 - 30 * u[0]
    f[2] = u[2] - u[3] + u[4] * (1. - u[3]) * E2
    f[3] = 10 * u[0] - 30 * u[1] + 22 * u[4] * (1. - u[3]) * E2
    f[4] = u[p] - alpha
  return f def dfcn(n, u, p):
    df = np.zeros((n - 1 + 1, n))
    E1 = np.exp(10 * u[0]/(1. + 0.01 * u[0]))
    E2 = np.exp(10 * u[1]/(1. + 0.01 * u[1]))
    E1d = E1 * (10 * (1. + 0.01 * u[0])- 10 * u[0] * 0.01)/pow((1. + 0.01 * u[0]), 2)
    E2d = E2 * (10 * (1. + 0.01 * u[1])- 10 * u[1] * 0.01)/pow((1. + 0.01 * u[1]), 2)
    df[0,0] = u[4] * (1 - u[2]) * E1d
    df[0,2] = -u[4] * E1 - 1
    df[0,4] = (1. - u[2]) * E1
    df[1,0] = 22 * u[4] * (1. - u[2]) * E1d - 30
    df[1,2] = -22 * u[4] * E1
    df[1,4] = 22 * (1. - u[2]) *E1
    df[2,1] = u[4] * (1. - u[3]) * E2d
    df[2,2] = 1
    df[2,3] = -1 - u[4] * E2
    df[2,4] = (1. - u[3]) *E2
    df[3,0] = 10
    df[3,1] = -30 + 22 * u[4] * (1. - u[3]) *E2d
    df[3,3] = -22 * u[4] * E2
    df[3,4] = 22 * (1. - u[3]) * E2
    df[4,p] = 1
  return df
```

Algorithm 8: continuation for Chemical Reaction

```
def Continuation(numIter, x0, tanVecprevious, p, h, f, df, m, tol):
    cu5 = [ ]
    cu2 = [ ]
for i in range(numIter): do
    sol = newtonChemical(f, df, m, tol, x0, p)
    tan = tangent (sol[0], tanVecprevious, p, df)
    x01 = sol[0] + h * tan
    x0 = x01
    p = tan[1]
    print("In iteration", i, "the value of p is", p)
    cu5.append(sol[0][4])
    cu2.append(sol[0][1])
    tanVecprevious = tan[0]
end
  return cu5, cu2
m= 10
tol= 1e-4
alpha= 0.00
x0= np.zeros(5)
p= 4
tanVecprevious = np.ones(5)
numIter= 162
h= 0.02
b= Continuation(numIter, x0, tanVecprevious, p, h, fcn, dfcn, m, tol)
fig= plt.figure(figsize=(6,4))
plt.plot(b[0], b[1], "r.-")
plt.xlabel("x2")
plt.ylabel("x5")
plt.grid()
plt.savefig("Chemical Reaction.pdf')
plt.show()
```

Acknowledgements

First and foremost, I wish to express my gratitude to the almighty Allah whose guidance and strength have sustained me in the course of writing this project. Many thanks to the African Institute for Mathematical Sciences (AIMS-Cameroon) for the scholarship that covered the entire cost of my studies,and for the facilities they offered to us. I am sincerely happy to thank my supervisor, Prof.Georg Bader, for providing me with this essay topic and the encouragement to face it with great enthusiasm. I strongly recognize the sufficient support from Alkan Goktug who was constantly there for me in times of need and all the brilliant corrections he made for this thesis to be a success. To all my lecturers at Aims-cameroon during the 2019/2020 academic year, I say thank you for the knowledge imparted in me that guided me throughout this essay. The tutors at Aims-cameroon are not left out for their constant advice and encouragement that kept me stress-free during the course of the essay phase. It is impossible for me to finish writing this acknowledgement without expressing my deepest appreciation to my beloved family, the Alh Jafar Aminu family.
Finally, I feel indebted to many people who helped me in one way or the other for the successful completion of this project may Allah(S.W.T) bless you all.

References

[1] Anselone, P. and Moore, R. (1966). An extension of the newton-kantorovič method for solving nonlinear equations with an application to elasticity. *Journal of Mathematical Analysis and Applications*, 13(3):476–501.

[2] Atkinson, K. E. (2008). *An introduction to numerical analysis*. John wiley & sons.

[3] Bebernes, J. and Eberly, D. (2013). *Mathematical problems from combustion theory*, volume 83. Springer Science & Business Media.

[4] Burkardt, J. (2014). The continuation method for algebraic nonlinear equations.

[5] Butcher, J. C. (2016). *Numerical methods for ordinary differential equations*. John Wiley & Sons.

[6] Deuflhard, P. (2011). *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*, volume 35. Springer Science & Business Media.

[7] Deuflhard, P., Fiedler, B., and Kunkel, P. (1987). Efficient numerical pathfollowing beyond critical points. *SIAM journal on numerical analysis*, 24(4):912–927.

[8] Ganji, D., Nourollahi, M., and Mohseni, E. (2007). Application of heâs methods to nonlinear chemistry problems. *Computers & Mathematics with Applications*, 54(7-8):1122–1132.

[9] Hairer, E., Nørsett, S. P., and Wanner, G. (1993). Solving ordinary differential equations i. nonstiff problems, volume 8 of.

[10] Haselgrove, C. (1961). The solution of non-linear equations and of differential equations with two-point boundary conditions. *The Computer Journal*, 4(3):255–259.

[11] Jacobsen, J. and Schmitt, K. (2002). The liouville–bratu–gelfand problem for radial operators. *Journal of Differential Equations*, 184(1):283–298.

[12] Kubíček, M. (1976). Algorithm 502: Dependence of solution of nonlinear systems on a parameter [c5]. *ACM Transactions on Mathematical Software (TOMS)*, 2(1):98–107.

[13] Remani, C. (2013). Numerical methods for solving systems of nonlinear equations.

[14] Wacker, H. (1978). *Continuation methods: proceedings of a symposium at the University of Linz, Austria, October 3-4, 1977*. Academic Press.