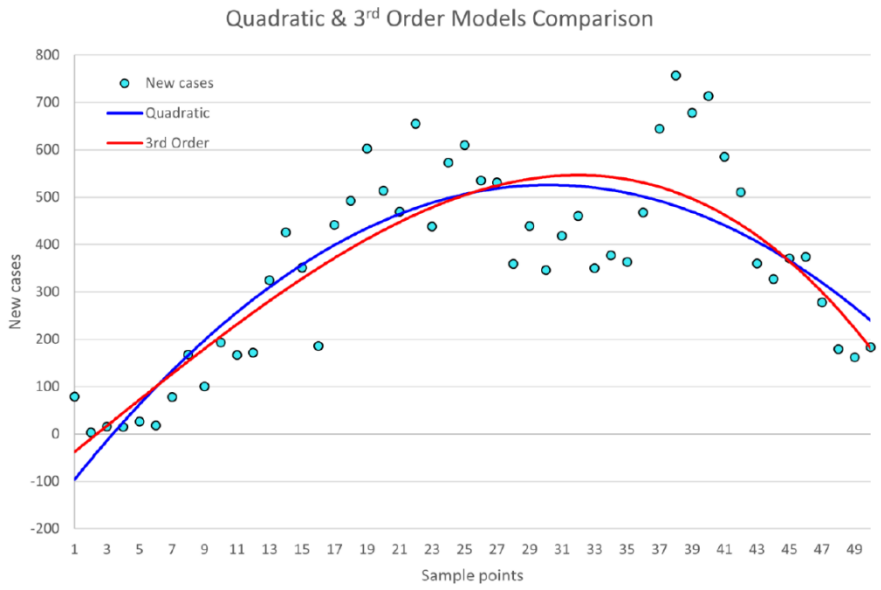
UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EENG 205: NUMERICAL ANALYSIS
COMPUTER ALGORITHMS AND PRACTICAL LABS SESSIONS

Algorithm 2: Function *meanValueRefinement()*, computes the refinement procedure using mean value extension

Data: f, X, N
Result: returns Y the value for the mean value extension form for f evaluated over a partition of X.

```
h ← (sup(X) − inf(X))/N
xi ← inf(X)
x1 ← xi
for i = 1 : N do
    xip1 ← x1 + i*h
    Xs(i) ← infsup(xi,xip1)  // Interval class constructor for each subinterval.
    xi ← xip1
Xs(N) ← infsup(inf(Xs(N)), sup(X))
Y ← meanValue(f,Xs(1))
if N > 1 then
    for i = 2 : N do
        Y ← hull(Y,meanValue(f,Xs(i)) // take the union of mean extension.
```



Prepared by : Prof. Ebrahim Mattar.

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO.1
PRACTICAL: MAXIMUM OF (10) MARKS

MULTIVARIABLE BISECTION METHOD ALGORITHM

For the given attached notes (the journal article source):

> *REVISTADELAUNI´ON MATEM´ATICA ARGENTINA Vol.60,No.1,2019,Pages79–98 Published online:March20,2019* https://doi.org/10.33044/revuma.v60n1a06
>
> ***THE MULTIVARIATEBISECTIONALGORITHM*** *MANUEL L´OPEZ GALV´AN*
> https://doi.org/10.33044/revuma.v60n1a06

write a computer algorithm (code) to solve the given problem in the attached article. The code can be written in (matlab, python, or c++). *I do prefer python*.

- Verify your coding and results. (5 marks)
- Compare your hand solution (3 steps), and the code solution. (5 marks)

Maximum of three students can work on this laboratory session.
Send your solution to my UoB email account: ebmattar@uob.edu.bh

Prepared by Prof. Ebrahim A. Mattar

REVISTA DE LA UNIÓN MATEMÁTICA ARGENTINA
Vol. 60, No. 1, 2019, Pages 79–98
Published online: March 20, 2019
https://doi.org/10.33044/revuma.v60n1a06

THE MULTIVARIATE BISECTION ALGORITHM

MANUEL LÓPEZ GALVÁN

Abstract. The aim of this paper is to study the bisection method in $\mathbb{R}^n$. We propose a multivariate bisection method supported by the Poincaré–Miranda theorem in order to solve non-linear systems of equations. Given an initial cube satisfying the hypothesis of the Poincaré–Miranda theorem, the algorithm performs congruent refinements through its center by generating a root approximation. Through preconditioning we will prove the local convergence of this new root finder methodology and moreover we will perform a numerical implementation for the two dimensional case.

1. Introduction

The problem of finding numerical approximations to the roots of a non-linear system of equations was the subject of various studies, and different methodologies have been proposed between optimization and Newton's procedures. In [4] D. H. Lehmer proposed a method for solving polynomial equations in the complex plane testing increasingly smaller disks for the presence or absence of roots. In other work, Herbert S. Wilf developed a global root finder of polynomials of one complex variable inside any rectangular region using Sturm sequences [12].

The classical Bolzano's theorem or Intermediate Value theorem ensures that a continuous function that changes sign in an interval has a root, that is, if $f : [a, b] \to \mathbb{R}$ is continuous and $f(a)f(b) < 0$ then there exists $c \in (a, b)$ such that $f(c) = 0$. In the multidimensional case the generalization of this result is the known Poincaré–Miranda theorem that ensures that if we have $f_1, \ldots, f_n$ n continuous functions of the n variables $x_1, \ldots, x_n$ and the variables are subjected to vary between a_i and $-a_i$, then if $f_i(x_1, \ldots, a_i, \ldots, x_n)f_i(x_1, \ldots, -a_i, \ldots, x_n) < 0$ for all x_i then there exists $c \in [-a_i, a_i]^n$ such that $f(c) = 0$. This result was announced for the first time by Poincaré in 1883 [8] and published in 1884 [9] with reference to a proof using homotopy invariance of the index. The result obtained by Poincaré has come to be known as the theorem of Miranda, who in 1940 showed that it is equivalent to the Brouwer fixed point [5]. For different proofs of the Poincaré–Miranda theorem in the n-dimensional case, see [3, 10].

2010 *Mathematics Subject Classification.* 65K05, 65H10, 65G30.
Key words and phrases. Root-finding algorithm; Non-linear preconditioning; Bisection method; Interval analysis.

Theorem 1.1 (Poincaré–Miranda theorem)**.** *Let K be the cube*

$$K = \{x \in \mathbb{R}^n : |x_j - \hat{x_j}| \leq \rho,\ j = 1(1)n\}$$

where $\rho \geq 0$ and $\mathbf{F} = (f_1, f_2, \ldots, f_n) : K \to \mathbb{R}^n$ a continuous map on K. Also, let

$$F_i^+ = \{x \in K : x_i = \hat{x_i} + \rho\}, \quad F_i^- = \{x \in K : x_i = \hat{x_i} - \rho\}$$

be the pairs of parallel opposite faces of the cube K.

If for $i = 1(1)n$ the i-th component f_i of $\mathbf{F}$ has opposite sign or vanishes on the corresponding opposite faces F_i^+ and F_i^- of the cube K, i.e.

$$f_i(x)f_i(y) \leq 0, \quad x \in F_i^+, y \in F_i^- \tag{1.1}$$

then the mapping $\mathbf{F}$ has at least one zero point $r = (r_1, r_2, \ldots, r_n)$ in K.

Throughout this paper we will recall the opposite signs condition (1.1) as the Poincaré–Miranda property P.M. The aim of this work is to develop a bisection method that allows us to solve the non-linear system of equations $\mathbf{F}(X) = 0$, $X = (x_1, x_2, \ldots, x_n)$, using the above Poincaré–Miranda theorem. The idea of the algorithm will be similar to the classical one dimensional algorithm: we perform refinements of the cube domain in order to check the sign conditions on the parallel faces. In one dimension it is clear that an initial sign change in the border of an interval produces another sign change in a half partition of it, but in several dimensions we cannot guarantee that the Poincaré–Miranda conditions maintain after a refinement. Even if r is an exact solution, there may not be any such K (for which (1.1) holds). However, J. B. Kioustelidis [2] has pointed out that, for $\hat{x}$ close to a simple solution (where the Jacobian is nonsingular) of $\mathbf{F}(X)$, Miranda's theorem will be applicable to some equivalent system for suitable K. Therefore in case of a fail in the sign conditions with the original system, we should try to transform it. The idea will be to find an equivalent system through non-linear preconditioning where the equations are better balanced in the sense that the new system could be close to some hyperplane in order to improve the chances to check the sign conditions in some member of the refinement.

We will denote the infinity norm by $\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$, the Euclidean norm by $\|x\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$, and the 1-norm by $\|x\|_1 = |x_1| + \cdots + |x_n|$. Given a vector norm on $\mathbb{R}^n$, the associated matrix norm for a matrix $M \in \mathbb{R}^{n \times n}$ is defined by

$$\|M\|_p = \max_{\|x\|_p = 1} \|Mx\|_p \quad \text{where } p = \infty,\ 2,\ \text{or } 1.$$

It is known that in the case of the ∞-matrix norm it can be expressed as a maximum sum of its rows, that is if $M = (m_{ij})$ then $\|M\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^n |m_{ij}|$, therefore it is easy to see that a sequence of matrices $(M_k)_k$ converge if and only if their coordinates converge. Since the domains involved are multidimensional cubes, the most proper norm to handle the distance will be the ∞-norm.

We will accept r as a root with a small tolerance level δ if $\|\mathbf{F}(r)\|_p \leq \delta$.

2. The algorithm and its description

This section gives a step-by-step description of our algorithm, whose core lies in the classical bisection algorithm in one dimension.

Definition 2.1. A 2^n-refinement of a cube $K \subset \mathbb{R}^n$ is a refinement into 2^n congruent cubes $Q = \{K^1, K^2, \ldots, K^{2^n}\}$.

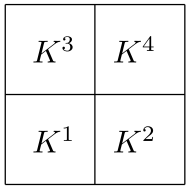


Figure 1. 4-refinement in $\mathbb{R}^2$.

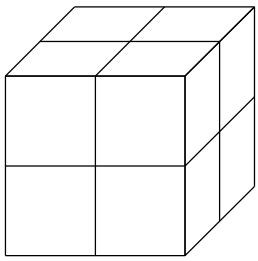


Figure 2. 8-refinement in $\mathbb{R}^3$.

We say that a 2^n-refinement of Q satisfies the Poincaré–Miranda condition if there exist $K^l \in Q$ such that $\mathbf{F} : K^l \to \mathbb{R}^n$ satisfies the condition of Theorem 1.1.

Given a system $\mathbf{F}(X) = 0$, the preconditioned system is $\mathbf{G}(X) = M\mathbf{F}(X) = 0$ for some matrix M such that the jacobian at X_0 satisfies $D\mathbf{G}(X_0) = \mathrm{Id}$. Since $D\mathbf{G}(X_0) = MD\mathbf{F}(X_0)$ it turns out that $M = D\mathbf{F}(X_0)^{-1}$ and it is clear that the preconditioned system is an equivalent system of $\mathbf{F}$ and both have the same roots. After preconditioning, the equations in $\mathbf{G}(X) = 0$ are close to a hyperplane having equation $x_i = k_i$, where k_i is some constant. This fact comes from the Taylor expansion of $\mathbf{G}$ around X_0, indeed if X is close to X_0 then

$$\mathbf{G}(X) \approx \mathbf{G}(X_0) + D\mathbf{G}(X_0)(X - X_0) = \mathbf{G}(X_0) + X - X_0$$

and therefore it is clear that the equations are close to some hyperplane. Moreover if X_0 is nearly a zero point of $\mathbf{F}$ then

$$\mathbf{G}(X) \approx X - X_0$$

and therefore it will behave like the components of $X - X_0$, and take nearly opposite values on the corresponding opposite faces of the cube.

2.1. **Algorithm procedure.** The multivariate bisection algorithm proceeds as follows:

(1) We start choosing an initial guess $K_0 = [a_{1_0}, b_{1_0}]\times[a_{2_0}, b_{2_0}]\times\cdots\times[a_{n_0}, b_{n_0}] \subset \mathbb{R}^n$ satisfying the Poincaré–Miranda condition on $\mathbf{F}$.

(2) We locate the center

$$c_1 = \left(\frac{a_{1_0} + b_{1_0}}{2}, \frac{a_{2_0} + b_{2_0}}{2}, \ldots, \frac{a_{n_0} + b_{n_0}}{2}\right)$$

of K_0.

(3) Generate a first 2^n-refinement Q_1 through c_1.

(4) If Q_1 satisfies the Poincaré–Miranda condition, let $K_1 = [a_{1_1}, b_{1_1}]\times[a_{2_1}, b_{2_1}]\times\cdots\times[a_{n_1}, b_{n_1}]$ be the quarter of Q_1 where the conditions of Theorem 1.1 are satisfied, we chose

$$c_2 = \left(\frac{a_{1_1} + b_{1_1}}{2}, \frac{a_{2_1} + b_{2_1}}{2}, \ldots, \frac{a_{n_1} + b_{n_1}}{2}\right)$$

the center of K_1. If Q_1 does not satisfy the Poincaré–Miranda condition we preconditioning the system in c_1 setting

$$\mathbf{G}_1(X) := D\mathbf{F}(c_1)^{-1}\mathbf{F}(X)$$

and then we check again the sign conditions with the preconditioned system $\mathbf{G}_1(X)$ in Q_1.

This recursion is repeated while the Poincaré–Miranda condition are satisfied, generating a sequence of equivalent systems

$$\mathbf{G}_k(X) := \begin{cases} \mathbf{G}_{k-1}(X) & \text{if } \mathbf{G}_{k-1} \text{ satisfies P.M. in } Q_k \\ D\mathbf{G}_{k-1}(c_k)^{-1}\mathbf{G}_{k-1}(X) & \text{if } \mathbf{G}_{k-1} \text{ does not satisfy P.M. in } Q_k \end{cases}$$

and a decreasing cube sequence K_k, such that

$$K_{k+1} \subset K_k = [a_{1_k}, b_{1_k}] \times \cdots \times [a_{n_k}, b_{n_k}],$$

where the vertices satisfy

$$a_{j_0} \leq a_{j_1} \leq a_{j_2} \leq \ldots \leq a_{j_k} \leq \ldots \leq b_{j_0} \tag{2.1}$$

$$b_{j_0} \geq b_{j_1} \geq b_{j_2} \geq \ldots \geq b_{j_k} \geq \ldots \geq a_{j_0} \tag{2.2}$$

for each $j = 1(1)n$ and where the length of the current interval $[a_{j_k}, b_{j_k}]$ is a half of the last iteration,

$$a_{j_k} - b_{j_k} = \frac{a_{j_{k-1}} - b_{j_{k-1}}}{2} = \ldots = \frac{a_{j_0} - b_{j_0}}{2^k}. \tag{2.3}$$

The root's approximation after the k-th iteration will be

$$c_k = \left(\frac{a_{1_k} + b_{1_k}}{2}, \frac{a_{2_k} + b_{2_k}}{2}, \ldots, \frac{a_{n_k} + b_{n_k}}{2}\right),$$

and the method is stopped until the zero's estimates gives sufficiently accuracy or until the Poincaré–Miranda condition no longer holds.

Remark 2.2. The kth-preconditioning system $\mathbf{G}_k(X) = (g_{1_k}(X), \ldots, g_{n_k}(X))$ can be expressed as $D\mathbf{F}(c_k)^{-1}\mathbf{F}(X)$, indeed, by induction suppose that it is true for $k-1$, then differencing and evaluating in c_k we have

$$\begin{aligned}\mathbf{G}_k(X) &= D\mathbf{G}_{k-1}(c_k)^{-1}\mathbf{G}_{k-1}(X)\\ &= D\mathbf{G}_{k-1}(c_k)^{-1}D\mathbf{F}(c_{k-1})^{-1}\mathbf{F}(X)\\ &= D\mathbf{F}(c_k)^{-1}D\mathbf{F}(c_{k-1})D\mathbf{F}(c_{k-1})^{-1}\mathbf{F}(X)\\ &= D\mathbf{F}(c_k)^{-1}\mathbf{F}(X).\end{aligned}$$

Since we cannot always ensure that a refinement of a given cube will satisfy the Poincaré–Miranda condition, we cannot ensure the converge for any map that only has a sign change in a given initial cube. So, in case of a fail in the sign conditions in some step, we try to rebalance the system using preconditioning in the center of the current box recursion. The preconditioning allows us to increase the chances to be more often in the sign conditions and therefore keep going with the quadrisection procedure in order to get a better root's approximation. In [2], J. B. Kioustelidis found sufficient conditions for the validity of the Poincaré–Miranda Miranda condition for preconditioning system, there it was proved that the sign conditions are always valid if the center of the cube K is close enough to some root of $\mathbf{F}$. So, if we start the multivariate bisection algorithm with an initial guess close to some root, Kioustelidis's theorem will guarantee the validity of Poincaré–Miranda in each step of our method allowing the local convergence of it.

In the next theorem we will prove the local convergence for the multivariate bisection algorithm when we preconditioning at each step.

Theorem 2.3. *Let $\mathbf{F} = (f_1, \ldots, f_n) : K_0 \to \mathbb{R}^n$ be a C^2 map defined on the cube $K_0 = \{x \in \mathbb{R}^n : \|x - c_1\|_\infty \leq \rho\} = [a_{1_0}, b_{1_0}] \times \cdots \times [a_{n_0}, b_{n_0}]$ with ρ small enough satisfying the Poincaré–Miranda sign condition; assume that $D\mathbf{F}(X)$ is invertible for all $X \in K_0$; furthermore, suppose that we perform the preconditioning at each step. Then the multivariate bisection algorithm generates a sequence c_k such that*

(1) *Starting at K_0, $c_k \xrightarrow{\|\cdot\|} r$ with $\mathbf{F}(r) = 0$.*
(2) $\|c_k - r\|_2 \leq \dfrac{\sum_{j=1}^{n} b_{j_0} - a_{j_0}}{2^k}$.

Proof. (1) The Poincaré–Miranda sign conditions guarantee the existence of a root inside K_0 and given a refinement Q_1 of K_0 since ρ is small enough Item c of Theorem 2 in [2] guarantees the validity of Poincaré–Miranda sign conditions for a member of Q_1. Performing successive refinements we will always find a member K_k of the refinement Q_k satisfying the sign conditions for the preconditioned system $\mathbf{G}_k(X)$. For each $j = 1(1)n$ the sequences $(a_{j_k})_k, (b_{j_k})_k$ are monotone and bounded and therefore they converge. From equation (2.3) we have for each $j = 1(1)n$,

$$\lim_{k\to\infty} a_{j_k} = \lim_{k\to\infty} b_{j_k} = r_j \tag{2.4}$$

and from the border conditions,

$$\begin{gathered} g_{1_k}(a_{1_k}, x_2, \ldots, x_n) g_{1_k}(b_{1_k}, x_2, \ldots, x_n) \leq 0 \quad \forall\, x \in K_k \\ \cdots \\ g_{j_k}(x_1, \ldots, x_{j-1}, a_{j_k}, x_{j+1}, \ldots, x_n) g_{j_k}(x_1, \ldots, x_{j-1}, b_{j_k}, x_{j+1}, \ldots, x_n) \leq 0 \quad \forall\, x \in K_k \\ \cdots \\ g_{n_k}(x_1, \ldots, x_{n-1}, a_{n_k}) g_{n_k}(x_1, \ldots, x_{n-1}, b_{n_k}) \leq 0 \quad \forall\, x \in K_k \end{gathered} \tag{2.5}$$

Since the diameter of K_k tends to zero by Cantor's intersection theorem the intersection of the K_k contains exactly one point,

$$\{p\} = \bigcap_{n=0}^{\infty} K_k$$

and the equations (2.4) guarantee that $p = (r_1, \ldots, r_n)$. Then, we can evaluate equations (2.5) in $p = (r_1, \ldots, r_n)$, getting

$$\begin{gathered} g_{1_k}(a_{1_k}, r_2, \ldots, r_n) g_{1_k}(b_{1_k}, r_2, \ldots, r_n) \leq 0 \ \forall\ k \in \mathbb{N} \\ \cdots \\ g_{n_k}(r_1, \ldots, r_{n-1}, a_{n_k}) g_{n_k}(r_1, \ldots, r_{n-1}, b_{n_k}) \leq 0 \ \forall\ k \in \mathbb{N} \end{gathered} \tag{2.6}$$

It is clear that

$$c_k = \left(\frac{a_{1_k} + b_{1_k}}{2}, \frac{a_{2_k} + b_{2_k}}{2}, \ldots, \frac{a_{n_k} + b_{n_k}}{2} \right) \underset{k \to \infty}{\longrightarrow} (r_1, \ldots, r_n) = r;$$

then by the continuity of $D\mathbf{F}$ and the continuity of the inversion in the ∞-matrix norm we have

$$D\mathbf{F}(c_k)^{-1} \to D\mathbf{F}(r)^{-1}.$$

Let $G(X) = D\mathbf{F}(r)^{-1} F(X) = (g_1(X), \ldots, g_n(X))$; since

$$\|D\mathbf{F}(c_k)^{-1} F(X) - D\mathbf{F}(r)^{-1} F(X)\|_\infty \leq \|D\mathbf{F}(c_k)^{-1} - D\mathbf{F}(r)^{-1}\|_\infty \|F(X)\|_\infty \to 0$$

for each $X \in K_0$ then we get the punctual convergence for each coordinate function

$$g_{j_k}(X) \underset{k \to \infty}{\to} g_j(X).$$

From equations (2.4) we have

$$g_{j_k}(r_1, \ldots, r_{j-1}, a_{j_k}, r_{j+1}, \ldots, r_n) \to g_j(r_1, \ldots, r_j, \ldots, r_n) \quad \text{for each } j = 1(1)n,$$

$$g_{j_k}(r_1, \ldots, r_{j-1}, b_{j_k}, r_{j+1}, \ldots, r_n) \to g_j(r_1, \ldots, r_j, \ldots, r_n) \quad \text{for each } j = 1(1)n;$$

therefore taking limit in equations (2.6) we get

$$g_j(r_1, r_2, \ldots, r_n)^2 \leq 0 \quad \forall j = 1(1)n,$$

and finally it is clear that $\mathbf{F}(r_1, \ldots, r_n) = 0$.

(2) Let $(c_{j_k})_k$, $j = 1(1)n$, be the coordinates of the sequence $(c_k)_k$; we have the following estimation:

$$|c_{j_k} - r_j| \leq \frac{b_{j_0} - a_{j_0}}{2^k} \quad \text{for each } j = 1(1)n.$$

Indeed, since the sequences (a_{j_k}) and (b_{j_k}) are monotone and bounded by r_j we get for each $j = 1(1)n$,

$$\begin{aligned} c_{j_k} - r_j = \frac{a_{j_{k-1}}}{2} + \frac{b_{j_{k-1}}}{2} - r_j &\leq \frac{a_{j_{k-1}}}{2} + \frac{b_{j_{k-1}}}{2} - a_{j_{k-1}} \\ &= \frac{b_{j_{k-1}}}{2} - \frac{a_{j_{k-1}}}{2} = \frac{b_{j_0} - a_{j_0}}{2^k}. \end{aligned}$$

On the other hand,

$$\begin{aligned} c_{j_k} - r_j = \frac{a_{j_{k-1}}}{2} + \frac{b_{j_{k-1}}}{2} - r_j &\geq \frac{a_{j_{k-1}}}{2} + \frac{b_{j_{k-1}}}{2} - b_{j_{k-1}} \\ &= -\Big(\frac{b_{j_{k-1}}}{2} - \frac{a_{j_{k-1}}}{2}\Big) = -\Big(\frac{b_{j_0} - a_{j_0}}{2^k}\Big). \end{aligned}$$

Therefore,

$$\begin{aligned} \|c_k - r\|_2 = \sqrt{\textstyle\sum_{j=1}^n (c_{j_k} - r_j)^2} &\leq \textstyle\sum_{j=1}^n |c_{j_k} - r_j| \\ &\leq \frac{\sum_{j=1}^n b_{j_0} - a_{j_0}}{2^k}. \end{aligned}$$

□

As in the classical one dimensional bisection algorithm, Item 2 of Theorem 2.3 gives a way to determine the number of iterations that the bisection method would need to converge to a root to within a certain tolerance. The number of iterations needed, k, to achieve the given tolerance δ is given by

$$k = \log_2\left(\frac{\sum_{j=1}^n b_{j_0} - a_{j_0}}{\delta}\right) = \frac{\log\big(\sum_{j=1}^n b_{j_0} - a_{j_0}\big) - \log\delta}{\log 2}.$$

3. Implementation, performance, and testing

Throughout this section we will focus in the implementation and performance of the bisection algorithm in $\mathbb{R}^2$. The bisection algorithm was developed in MATLAB in a set of functions running from a main function. In order to check the P.M. conditions for the function $\mathrm{F} = (f_1, f_2)$ we need to compute the intervals $f_i(F_i^+), f_i(F_i^-)$ $(i = 1, 2)$ and one way to achieve this is by using interval analysis (IA). IA was marked by the appearance of the book *Interval Analysis* by Ramon E. Moore in 1966 [6] and it gives a fast way to find an enclosure for the range of the functions. A disadvantage of IA is the well-known overestimation. If intervals $f_i(F_i^+), f_i(F_i^-)$ are available then the P.M. follows from the condition

$$\sup\{y : y \in f_i(F_i^-)\} \leq 0 \leq \inf\{y : y \in f_i(F_i^+)\} \tag{3.1}$$

or

$$\sup\{y : y \in f_i(F_i^+)\} \leq 0 \leq \inf\{y : y \in f_i(F_i^-)\}. \tag{3.2}$$

Interval-valued extensions of real functions give a way to find an enclosure of the range of a given real-valued function. Most generally, if we note by $[\mathbb{R}]$ the set of all finite intervals, we say that $[f] : [\mathbb{R}]^n \to [\mathbb{R}]$ is an interval extension of $f : \mathbb{R}^n \to \mathbb{R}$ if

$$[f](X) \supseteq \{f(x) : x \in X\},$$

where $X = (X_1, \ldots, X_n)$ represents a vector of intervals. There are different kinds of interval functional extensions; if we have the formula of a real-valued function f then the natural interval extension is achieved by replacing the real variable x with an interval variable X and the real arithmetic operations with the corresponding interval operations. Another useful interval extension is the mean value form. Let $m = m(X)$ be the center of the interval vector X and let $[\frac{\partial f_i}{\partial x_i}]$ be an interval extension of $\frac{\partial f_i}{\partial x_i}$; by the mean value theorem we have

$$f(X) \subseteq [f_{mv}](X) = f(m) + \sum_{i=1}^{n} [\frac{\partial f_i}{\partial x_i}](X)(X_i - m_i),$$

where $[f_{mv}](X)$ is the mean value extension of f.

Let $[f_i](F_i^+), [f_i](F_i^-)$ be an interval extension of $f_i(F_i^+), f_i(F_i^-)$; then it is clear that if

$$\sup\{y : y \in [f_i](F_i^-)\} \leq 0 \leq \inf\{y : y \in [f_i](F_i^+)\} \tag{3.3}$$

or

$$\sup\{y : y \in [f_i](F_i^+)\} \leq 0 \leq \inf\{y : y \in [f_i](F_i^-)\}, \tag{3.4}$$

equations (3.1) and (3.2) are also true. So, in order to check the P.M. conditions along the edges we will compute equations (3.3) and (3.4).

Various interval-based software packages for MATLAB are available; we have chosen the well-known INTLAB toolbox [11]. The toolbox has several interval class constructor for intervals, affine arithmetic, gradients, hessians, slopes and more. Ordinary interval arithmetic has sometimes problems with dependencies and wrapping effect given large enclosures of the range and therefore overestimating the sign behaviour. A way to fight with this is affine arithmetic. In affine arithmetic an interval is stored as a midpoint X_0 together with error terms $E_1, \ldots, E_k$ and it represents

$$X = X_0 + E_1U_1 + E_2U_2 + \cdots + E_kU_k,$$

where $U_1, \ldots, U_k$ are parameters independently varying within $[-1, 1]$. In order to avoid an overestimation in the range enclosure of $f_i(F_i^-)$ and $f_i(F_i^+)$ we also compute the interval extension using the affine arithmetic.

Another way to improve the enclosure of the range and get sharper lower and upper bounds is through subdivision or refinements. In this methodology we perform subdivision of the domain and then we take the union of interval extensions over the elements of the subdivision; this procedure is called a refinement of $[f]$ over X. Let N be a positive integer; we define

$$X_{i,j} = [\inf X_i + (j-1)\frac{w(X_i)}{N}, \inf X_i + j\frac{w(X_i)}{N}], \quad j = 1, \ldots, N.$$

We have $X_i = \cup_{j=1}^{N} X_{i,j}$ and $w(X_{i,j}) = \dfrac{w(X_i)}{N}$ and furthermore,

$$X = \cup_{j_i=1}^{N}(X_{1,j_1}, \ldots, X_{n,j_n}) \quad \text{with } w(X_{1,j_1}, \ldots, X_{n,j_n}) = \frac{w(X)}{N}.$$

The interval quantity

$$[f]_N(X) = \cup_{j_i=1}^{N}[f](X_{1,j_1}, \ldots, X_{n,j_n})$$

is the refinement of $[f]$ over X.

The algorithms that we have performed to compute equations (3.3) and (3.4) combine all the above methodologies and were adapted from [7]. In the following steps we summarize the routines that we have performed. The mean value extension was implemented using an approximation of $[\frac{\partial f_i}{\partial x_i}]$ through the central finite difference of the natural interval extension of f, that is

$$[\frac{\partial f_i}{\partial x_i}](X) \approx \frac{[f](X + 0.0001) - [f](X - 0.0001)}{2\ 0.0001}.$$

Algorithm 1 shows the routine for the mean value extension.

Algorithm 1: Function *meanValue()*, computes the mean value extension

Data: f, X

Result: returns Fmv the value for the mean value extension form for f evaluated over the interval X.

```
1 m ← mid(X)
2 fm ← f(m)
3 derf ← f(X + 0.0001) − f(X − 0.0001)/(2 ∗ 0.0001)
4 Fmv ← fm + derf ∗ (X − m)
```

The refinement procedure was implemented twice, one for the case of mean value extension and other for the affine arithmetic implementation. Algorithm 2 computes the mean value extension over an uniform refinement of the interval X with N subintervals and Algorithm 3 computes the natural extension using affine arithmetic.

Algorithm 2: Function *meanValueRefinement()*, computes the refinement procedure using mean value extension

Data: f, X, N
Result: returns Y the value for the mean value extension form for f evaluated over a partition of X.

```
h ← (sup(X) − inf(X))/N
xi ← inf(X)
x1 ← xi
for i = 1 : N do
    xip1 ← x1 + i*h
    Xs(i) ← infsup(xi,xip1)  // Interval class constructor for each subinterval.
    xi ← xip1
Xs(N) ← infsup(inf(Xs(N)), sup(X))
Y ← meanValue(f,Xs(1))
if N > 1 then
    for i = 2 : N do
        Y ← hull(Y,meanValue(f,Xs(i)) // take the union of mean extension.
```

Algorithm 3: Function *affineIntervalRefinement()*, computes the refinement procedure using affine natural extension

Data: f, X, N
Result: returns Y the value for the affine natural extension form for f evaluated over a partition of X.

```
h ← (sup(X) − inf(X))/N
xi ← inf(X)
x1 ← xi
for i = 1 : N do
    xip1 ← x1 + i*h
    Xs(i) ← infsup(xi,xip1)  // Interval class constructor for each subinterval.
    xi ← xip1
Xs(N) ← infsup(inf(Xs(N)), sup(X))
Y ← f(affine(Xs(1)))
if N > 1 then
    for i = 2 : N do
        Y ← hull(Y,f(affine(Xs(i))) // take the union of natural affine extension.
```

Now we are ready to compute equations (3.3) and (3.4) using the above algorithms. Let $K^l = [a_{1_l}, b_{1_l}] \times [a_{2_l}, b_{2_l}] = I_{1_l} \times I_{2_l}$ be a member of the refinement Q

and let

$$f_{11} = f_2(\cdot, a_{2_l}) : [a_{1_l}, b_{1_l}] \to \mathbb{R}, \quad f_{12} = f_2(\cdot, b_{2_l}) : [a_{1_l}, b_{1_l}] \to \mathbb{R},$$

$$f_{21} = f_1(a_{1_l}, \cdot) : [a_{2_l}, b_{2_l}] \to \mathbb{R}, \quad f_{22} = f_1(b_{1_l}, \cdot) : [a_{2_l}, b_{2_l}] \to \mathbb{R}$$

be the coordinate functions on the edges of K^l ($l = 1 \dots 4$), Algorithm 4 summarizes the routine that we have performed using IA in order to compute the sign along the edges.

Algorithm 4: Function *signeval()*, computes the sign along the edges of K^l

Data: f_{ij}, $I_{il} = [a_{i_l}, b_{i_l}]$, N
Result: returns signf_{ij}, the sign of f_{ij} on I_{il}, 1 means positive, -1 negative and NaN indicates an empty output when the sign is not constant.

```
Dom ← infsup(a_{i_l}, b_{i_l})   // interval class constructor for I_{il}
Fmv ← meanValueRefinement(f_{ij},Dom,N)   // Apply Algorithm 2
extmin ← inf(Fmv)
extmax ← sup(Fmv)   // computes the max and min of the mean extension
if extmin ≥ 0 then
    sign f_{ij} ← 1   // check equation (3.3)
    return
 if extmax ≤ 0 then
    sign f_{ij} ← −1
    return
aff ← affineIntervalRefinement(f_{ij},Dom,N)   // Apply Algorithm 3
extmin ← inf(aff)
extmax ← sup(aff)   // computes the max and min of the affine extension
if extmin ≥ 0 then
    sign f_{ij} ← 1   // check equation (3.3)
    return
if extmax ≤ 0 then
    sign f_{ij} ← −1
    return
sign f_{ij} ← NaN
```

Algorithm 5 summarizes the implementation of the Bisection Algorithm that we have performed in MATLAB.

Algorithm 5: Bisection algorithm

Data: K_0, F= (f_1, f_2) system to solve, DF Jacobian of F, δ, N
Result: c root's approximation

if K_0 satisfies P.M. **then**
  $c \leftarrow$ center of K_0;
  error $\leftarrow \|\mathrm{F}(c)\|$;
  stop $\leftarrow 1$;
  F1orig $\leftarrow f_1$;
  F2orig $\leftarrow f_2$;
  while (error $> \delta$) $\wedge$ (stop < 3) **do**
    $(K^1, K^2, K^3, K^4) \leftarrow$ Generate a refinement of K_0 through c;
    (signf_{11},signf_{12},signf_{21},signf_{22}) $\leftarrow$ *signeval* $(f_{ij}, I_{i1}$,N) $i, j = 1, 2$ *// Apply Algorithm 4 on each edge of* K^1
    stop $\leftarrow$ stop+1;
    if signf_{11}sign$f_{12} \leq 0 \wedge$ signf_{21}sign$f_{22} \leq 0$ **then**
      $c \leftarrow$ center of K^1;
      $K_0 \leftarrow K^1$;
      error $\leftarrow \|\mathrm{F}(c)\|$;
      stop $\leftarrow$ stop $- 1$;
      Pass to next iteration
    (signf_{11},signf_{12},signf_{21},signf_{22}) $\leftarrow$ *signeval* $(f_{ij}, I_{i2}$,N) $i, j = 1, 2$; *// Apply Algorithm 4 on each edge of* K^2
    if signf_{11}sign$f_{12} \leq 0 \wedge$ signf_{21}sign$f_{22} \leq 0$ **then**
      $c \leftarrow$ center of K^2;
      $K_0 \leftarrow K^2$;
      error $\leftarrow \|\mathrm{F}(c)\|$;
      stop $\leftarrow$ stop $- 1$;
      Pass to next iteration
    (signf_{11},signf_{12},signf_{21},signf_{22}) $\leftarrow$ *signeval* $(f_{ij}, I_{i3}$,N) $i, j = 1, 2$; *// Apply Algorithm 4 on each edge of* K^3
    if signf_{11}sign$f_{12} \leq 0 \wedge$ signf_{21}sign$f_{22} \leq 0$ **then**
      $c \leftarrow$ center of K^3;
      $K_0 \leftarrow K^3$;
      error $\leftarrow \|\mathrm{F}(c)\|$;
      stop $\leftarrow$ stop $- 1$;
      Pass to next iteration
    (signf_{11},signf_{12},signf_{21},signf_{22}) $\leftarrow$ *signeval* $(f_{ij}, I_{i4}$,N) $i, j = 1, 2$; *// Apply Algorithm 4 on each edge of* K^4
    if signf_{11}sign$f_{12} \leq 0 \wedge$ signf_{21}sign$f_{22} \leq 0$ **then**
      $c \leftarrow$ center of K^4;
      $K_0 \leftarrow K^4$;
      error $\leftarrow \|\mathrm{F}(c)\|$;
      stop $\leftarrow$ stop $- 1$;
      Pass to next iteration
    DFc $\leftarrow$ DF(c); *// Build the preconditioning* $\mathbf{G}(X)$
    invDFc $\leftarrow$ inv(DFc);
    $f_1 \leftarrow$ invDFc(1,1)*F1orig+invDFc(1,2)*F2orig;
    $f_2 \leftarrow$ invDFc(2,1)*F1orig+invDFc(2,2)*F2orig;
else
  return Wrong R_0

In order to check the accuracy and performance of the algorithm, we test it through different systems of equations. We start testing the new algorithm in a transcendental system of equations used by Broyden in [1],

$$f_1 = \frac{1}{2}\sin(x_1 x_2) - \frac{x_2}{4\pi} - \frac{x_1}{2}$$
$$f_2 = (1 - \frac{1}{4\pi})(e^{2x_1} - e) + \frac{e x_2}{\pi} - 2e x_1.$$

The initial guess rectangle used for this problem was $K_0 = [0.4, 0.55] \times [3, 3.5]$ and the tolerance level was setted in $\delta = 10^{-15}$. The interval analysis refinement used to compute the sign along the edges is $N = 3$. Figure 3 illustrates the algorithm behaviour with the respective preconditioning procedure and Table 1 shows the numerical solution.

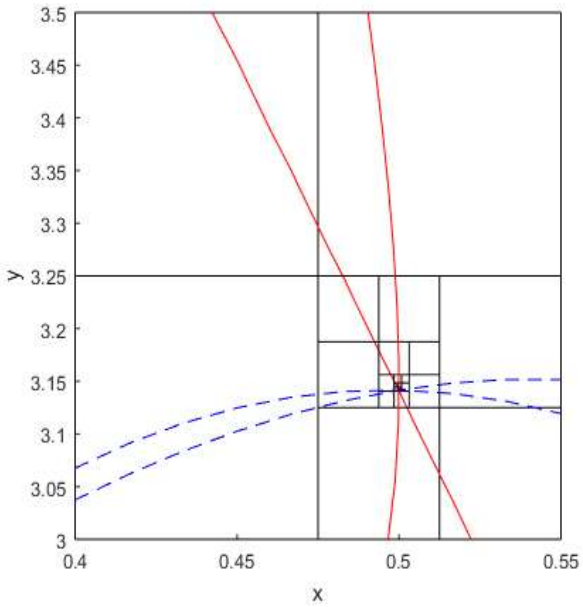


FIGURE 3. Bisection algorithm procedure for Broyden system. The solid red line represents f_1 and the dashed blue line represents f_2 for the equivalent system $\mathbf{G}_k(X) = 0$.

c	$\mathbf{F}(c)$	iter	K_0
0.500000000000000	-0.227513305973815 1e-15	48	$[0.4, 0.55] \times [3, 3.5]$
3.141592653589793	-0.643347227536409 1e-15		

TABLE 1. Root's approximation, evaluation, iteration performed and initial guess for Broyden system.

A straightforward computation shows that the solution for the system is $(x_1, x_2) = (0.5, \pi)$ and therefore we can check the consistency of the error estimation given

by Theorem 2.3 at each iteration,

$$e_k = \|c_k - (0.5, \pi)\|_2 \leq \frac{(0.55 - 0.4) + (3.5 - 3)}{2^k} = \frac{0.65}{2^k}.$$

Figure 4 illustrates the error behaviour in logarithmic scale at each iteration.

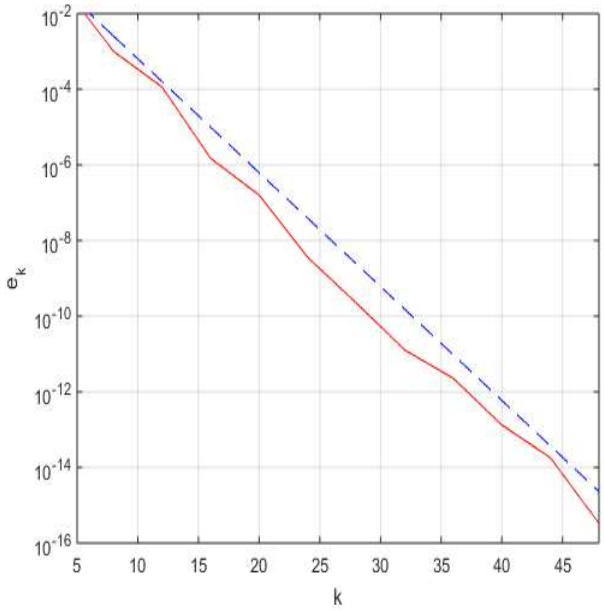


FIGURE 4. Error procedure for Broyden transcendental system. The red line represents e_k and the dashed blue line represents the error bound.

In the following steps we test the algorithm in several other problems. We will see that in some systems the algorithm needs preconditioning in order to guarantee the P.M. conditions through the refinement. Let

$$\begin{aligned}
\mathbf{F}_1(x, y) &= (x^2 + y^2 - 1, x - y^2) \\
\mathbf{F}_2(x, y) &= (2x - y - e^{-x}, -x + 2y - e^{-y}) \\
\mathbf{F}_3(x, y) &= (\sin(x) + \cos(y) + 2(x - 1), y - 0.5(x - 0.5)^2 - 0.5) \\
\mathbf{F}_4(x, y) &= (x^2 - \cos(xy), e^{xy} + y) \\
\mathbf{F}_5(x, y) &= (x\cos(y) + y\sin(x) - 0.5, e^{e^{-(x+y)}} - y(1 + x^2)) \\
\mathbf{F}_6(x, y) &= (x + 5(x - y)^3 - 1, 0.5(y - x)^3 + y)
\end{aligned}$$

be the testing maps. In Table 2 we show the numerical performance for the testing maps. The method was implemented setting the tolerance level in $\delta = 10^{-15}$ and the interval analysis refinement in $N = 3$.

$\mathbf{F}$	c	$\mathbf{F}(c)$	iter	K_0
$\mathbf{F}_1$	0.618033988749895	0.004965068306495 1e-14	51	$[0,1]\times[0,1]$
	0.786151377757422	-0.123942463016433 1e-14		
$\mathbf{F}_2$	0.567143290409784	0.111022302462516 1e-15	50	$[0,1]\times[0,1]$
	0.567143290409784	0.111022302462516 1e-15		
$\mathbf{F}_3$	0.378316940137480	0.139577647543639 1e-15	51	$[0,1]\times[0,1]$
	0.507403383528753	-0.072495394968176 1e-15		
$\mathbf{F}_4$	0.926174872358938	0.129347223584252 1e-15	49	$[0,1]\times[-1,0]$
	-0.582851662173280	-0.115653908517277 1e-15		
$\mathbf{F}_5$	0.353246619596717	-0.244439451327881 1e-15	52	$[0,1.1]\times[0,2]$
	0.606081736641465	0.047257391058546 1e-15		
$\mathbf{F}_6$	0.510030862987151	-0.045236309398304 1e-13	42	$[0.4,1]\times[0,0.4]$
	0.048996913701194	-0.904901681894059 1e-13		

TABLE 2. Root's approximation, evaluation, iteration performed and initial guess for testing maps.

Figure 5 illustrates the algorithm behaviour for the testing maps with the refinement procedure. The systems of equations and their successive possible preconditionings are represented by a zero contour level on an mesh on the initial guess K_0 and the refinement procedure was illustrated using the rectangle MATLAB's functions.

4. Newton's comparison

In this section we are going to compare the performance of our method against the classical multivariate Newton algorithm. It is well-known that the classical Newton's method has several numerical problems when the systems are ill-conditioned, i.e., systems having a "nearly singular" Jacobian at some iterate, getting slower rate of convergence and large numerical errors. The main advantage of our methodology is that it is not always necessary to perform the preconditioning at each step and therefore it can skip the ill-conditioned problem.

Let us consider the following system, $\mathbf{F}(x,y) = (x^2+y^2-1, xy-x^2)$. This function has a zero in $x^* = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$ and a straightforward computation shows that the system is ill-conditioned for any value sufficiently close to the origin. Newton's method cannot be initialized in $(0,0)$, however our algorithm can be evaluated in $(0,0)$ avoiding the singularity of the Jacobian.

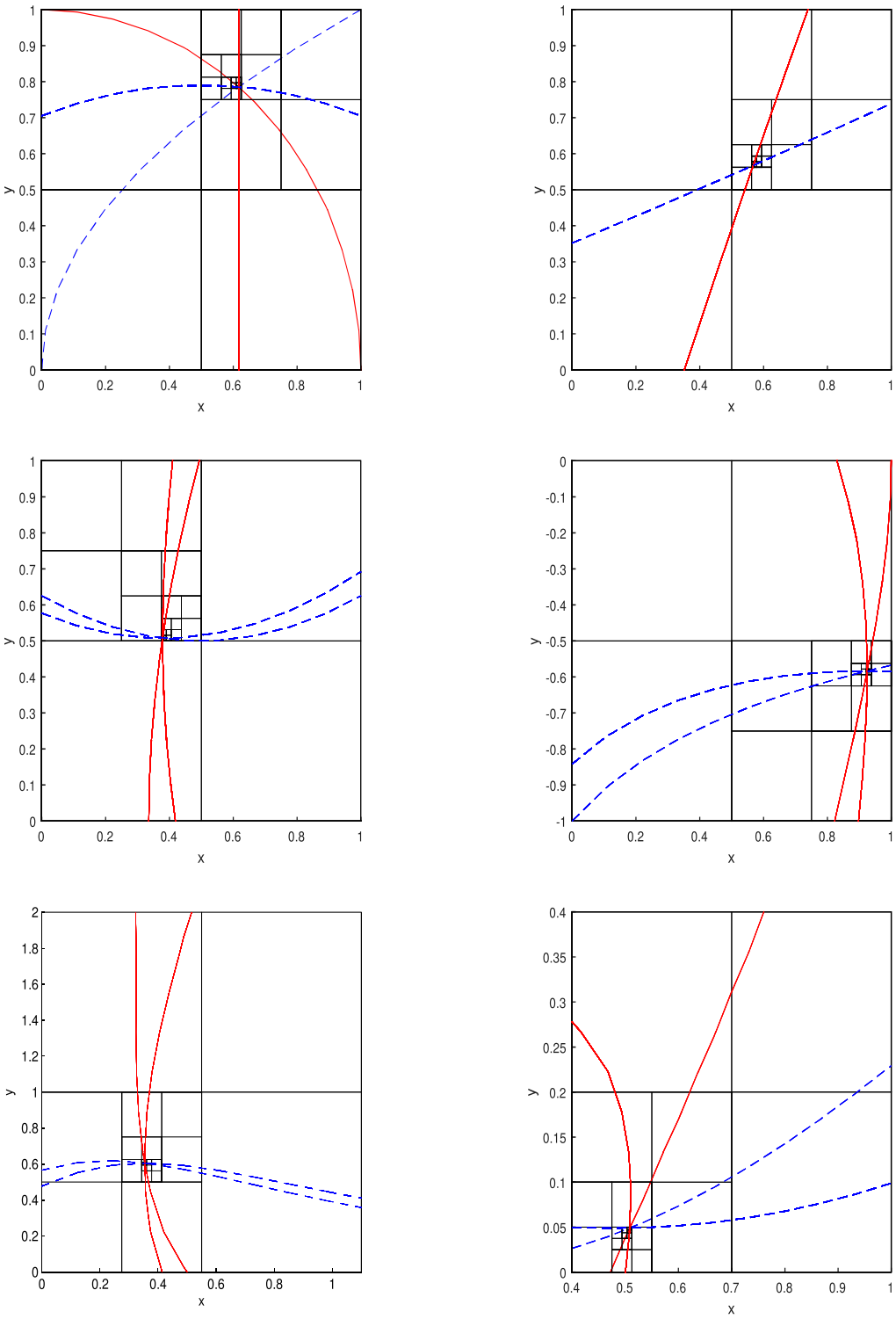


FIGURE 5. The first row illustrates the algorithm procedure for $\mathbf{F}_1, \mathbf{F}_2$, the second for $\mathbf{F}_3, \mathbf{F}_4$ and the third for $\mathbf{F}_5, \mathbf{F}_6$. The solid red line represents the first coordinate, while the dashed blue line represents the second coordinate for the equivalent system $\mathbf{G}_k(X) = 0$.

Figure 6 and Table 3 show the behaviour of the reduction of error norms $\|c_k - x^*\|$ and $\|\mathbf{F}(c_k)\|$ through iteration steps for the Bisection and Newton methods.

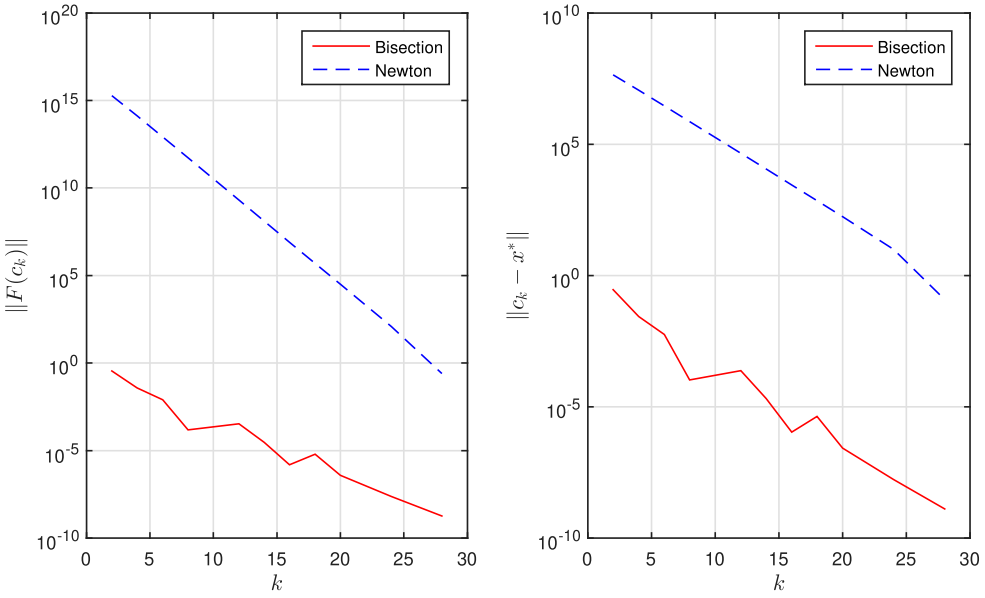


FIGURE 6. Newton vs Bisection performance.

	Newton		Bisection	
Step	$\|\mathbf{F}(c_k)\|$	$\|c_k - x^*\|$	$\|\mathbf{F}(c_k)\|$	$\|c_k - x^*\|$
2	1,88e+15	4,32e+07	3,54e-01	2,93e-01
4	1,34e+14	1,15e+07	3,87e-02	2,77e-02
6	8,38e+12	2,89e+06	7,94e-03	5,63e-03
8	5,24e+11	7,22e+05	1,51e-04	1,07e-04
12	2,05e+09	4,51e+04	3,37e-04	2,38e-04
14	1,28e+08	1,13e+04	2,90e-05	2,05e-05
16	7,99e+06	2,82e+03	1,53e-06	1,08e-06
18	4,99e+05	7,04e+02	6,10e-06	4,32e-06
20	3,12e+04	1,75e+02	3,82e-07	2,70e-07
24	1,22e+02	1,00e+01	2,42e-08	1,71e-08
28	2,51e-01	1,22e-01	1,85e-09	1,31e-09

TABLE 3. Error norms $\|\mathbf{F}(c_k)\|$, $\|c_k - x^*\|$ for the Newton and the Bisection methods after the iteration step for both cases.

The initial guess used to perform the Newton's method was $(10^{-8}, 10^{-9})$ which is inside and very close to the vertex of the initial square $K_0 = [0,1] \times [0,1]$ used to perform the Bisection method. As depicted in Figure 6 the Bisection method is highly superior to the Newton method in respect to achieve convergence. Bisection's method achieves a norm error of 1,31e-09 after 28 iteration, however Newton's method achieves a norm error of 1,22e-01 demanding much higher numerical effort.

As a second example, we try the system $\mathbf{F}(x,y) = (x^2 - 4y + y^2 - 1, 2x - y^2)$. In Figure 7 and Table 4 we compare the Newton method and the Bisection; the starting square was $K_0 = [1.0000001, 1.98] \times [1,2]$ and the initial guess for Newton was the lower left vertex $(1.0000001, 1)$ of K_0.

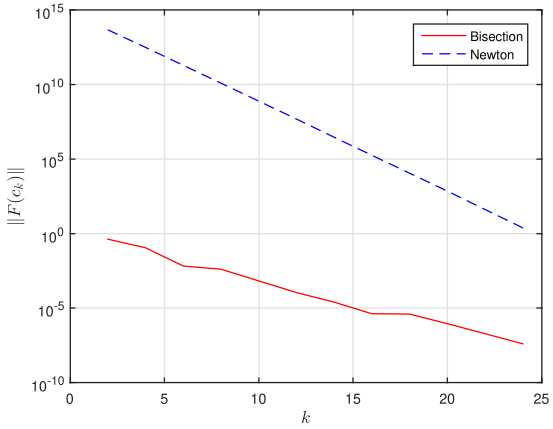


FIGURE 7. Newton vs Bisection performance.

Step	Newton $\|\mathbf{F}(c_k)\|$	Bisection $\|\mathbf{F}(c_k)\|$
2	4,76e+13	4,14e-01
4	3,03e+12	1,10e-01
6	1,91e+11	6,55e-03
8	1,19e+10	4,08e-03
12	4,66e+07	1,12e-04
14	2,91e+06	2,48e-05
16	1,82e+05	4,26e-06
18	1,14e+04	4,07e-06
20	7,10e+02	8,86e-07
24	2,33e+00	4,06e-08

TABLE 4. Error norms $\|\mathbf{F}(c_k)\|$ for the Newton and the Bisection method after the iteration step for both cases.

The condition number of the Jacobian matrix is close to 4e7 and therefore we are present to a ill-conditioned problem induced by a bad initial guess given by $(1.0000001, 1)$. However, the bisection method can start in this same vertex and can skip the Jacobian illness without preconditioning in the first steps. Bisection's method achieves a norm error of 4,06e-08 after 24 iteration, however Newton's method achieves a norm error of 2,33e+00.

5. Conclusion

In this work we have clarified how a multidimensional bisection algorithm should be performed extending the idea of the classic one dimensional bisection algorithm. Due to the preconditioning at each step we could prove a local convergence theorem and we also found an error estimation. Interval analysis allowed a fast and reliable way of computing the Poincaré–Miranda conditions and the numerical implementation showed that the method has a very good accuracy similar with the classic methods like Newton or continuous optimization. We also have compared the performance of the Bisection method against the classical Newton method and we found that our methodology improves the speed of convergence in ill-conditioned problems.

References

[1] C. G. Broyden; A new method of solving nonlinear simultaneous equations, Computer J. 12 (1969), 94–99. MR 0245197.

[2] J. B. Kioustelidis, Algorithmic error estimation for approximate solutions of nonlinear systems of equations, Computing 19 (1978), 313–320. MR 0474779.

[3] W. Kulpa, The Poincaré–Miranda theorem, Amer. Math. Monthly 104 (1997), no. 6, 545–550. MR 1453657.

[4] D. H. Lehmer, A machine method for solving polynomial equations, J. ACM 8 (1961), 151–162.

[5] C. Miranda, Un'osservazione su un teorema di Brouwer, Boll. Un. Mat. Ital. (2) 3 (1940), 5–7. MR 0004775.

[6] R. E. Moore, Interval Analysis. Prentice-Hall, Englewood Cliffs, NJ, 1966. MR 0231516.

[7] R. E. Moore, R. B. Kearfott and M. J. Cloud, Introduction to Interval Analysis, SIAM, Philadelphia, PA, 2009. MR 2482682.

[8] H. Poincaré, Sur certaines solutions particulières du problème des trois corps, C. R. Acad Sci. Paris 97 (1883), 251–252.

[9] H. Poincaré, Sur certaines solutions particulières du problème des trois corps, Bulletin Astronomique 1 (1884), 65–74.

[10] N. Rouche and J. Mawhin, Équations Différentielles Ordinaires. Tome I: Théorie Générale, Masson, Paris, 1973. MR 0481181.

[11] S. M. Rump, INTLAB - INTerval LABoratory. In: Developments in Reliable Computing, 77–104, Kluwer Academic Publishers, Dordrecht, 1999.

[12] H. S. Wilf, A global bisection algorithm for computing the zeros of polynomials in the complex plane, J. ACM 25 (1978), no. 3, 415–420. MR 0483384.

M. López Galván
Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. Intendente Güiraldes 2160, Ciudad Universitaria, C1428EGA Buenos Aires, Argentina
amlopezgalvan@gmail.com, mlopezgalvan@hotmail.com

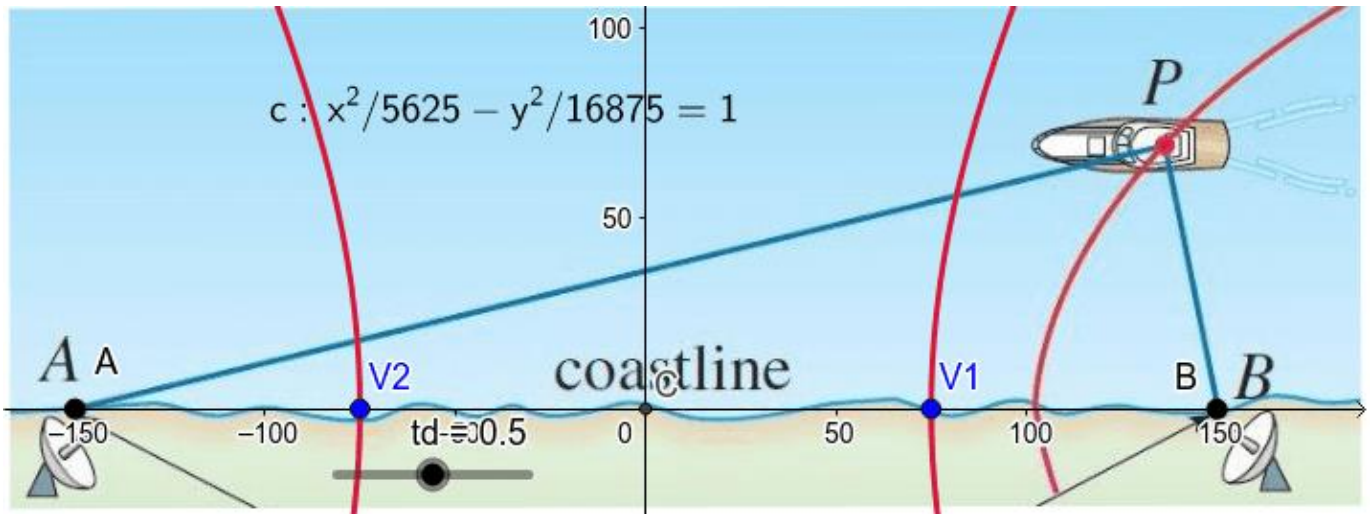
Received: January 10, 2018
Accepted: August 29, 2018

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO.2
PRACTICAL: MAXIMUM OF (10) MARKS

NEWTON' S METHOD (ALSO KNOWN AS NEWTON---RAPHSON METHOD)



Quoted [1] picture.

Quoted [1]

(The LORAN (LOng RAnge Navigation) system calculates the position of a boat at sea using signals from fixed transmitters. From the time differences of the incoming signals, the boat obtains differences of distances to the transmitters. This leads to two equations each representing hyperbolas defined by the differences of distance of two points (foci). An example of such equations from are).

$$f_1(x,y) = \frac{y^2}{300-186^2} + \frac{x^2}{186^2} = 1$$
$$f_2(x,y) = \frac{(y-500)^2}{279^2} - \frac{(x-300)^2}{(500^2-279)^2} = 1$$

Solving two quadratic equations with two unknowns, would require solving a 4 degree polynomial equation. We could do this by hand, but for a navigational system to work well, it must do the calculations automatically and numerically. We note that the Global Positioning System (GPS) works on similar principles and must do similar computations.

- Solve this set of nonlinear equation by hand (Newton's Method):

$$f_1(x, y) = y + 1.02x^3$$
$$f_2(x, y) = -0.9x + y^3$$

- Graph the solution. Hence, show that (1,0), is an initial point of solution.
- Solve the above problem by hand using NEWTON'S METHOD (ALSO KNOWN AS NEWTON, RAPHSON METHOD).
- Write a computer algorithm - code to solve the above NEWTON'S METHOD (ALSO KNOWN AS NEWTON, RAPHSON METHOD).
- Show the code, hence verify your computer algorithm and the hand results.

[1] LORAN (LOng RAnge Navigation) system.

Prepared by Prof. Ebrahim A. Mattar

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO.3
PRACTICAL: MAXIMUM OF (10) MARKS

LEAST SQUARE APPROXIMATION AND CURVE-FITTING

Write the experiment objectives, aims, procedures, results, and conclusions.

PART A: (5 MARKS)

- Find through hand calculations the polynomial (3rd order - polynomial) which passes through (-0.5,-1), (2.05,3.2) , (4,10.3), and (6,15.2) using the LEAST SQUARE APPROXIMATION.
- Use the model for Linear Interpolation to find the value of the Y axis, once X= 5.5.

PART B: (5 MARKS)

- Write a code (in Matlab, python, or n C++) (your code) for the creation of LINEAR INTERPOLATION (LEAST SQUARE APPROXIMATION AND CURVE-FITTING. (5 marks).
- Verify your coding and results using (matlab, c++, or python) solution, for any point on space.
- Compare your hand algorithm solution with the script coding.

Prof. Ebrahim Mattar.

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO. 4
PRACTICAL: MAXIMUM OF (10) MARKS

For the following 6th order linear matrix system,

$$\varphi = \begin{pmatrix} 3 & -0.4 & 0 & -0.9 & 0.02 & 0.01 \\ -1.2 & 4.2 & -1 & 0 & -0.88 & 0 \\ 0.03 & -1.2 & 4.5 & 0 & 0 & -1.2 \\ -0.9 & 0 & 0 & 3.7 & -1 & 0 \\ 0 & -1.2 & 0 & -1.1 & 4 & -1 \\ 0 & 0.02 & -1.4 & 0 & -0.7 & 3.5 \end{pmatrix}, \quad b = \begin{pmatrix} 1.2 \\ 0 \\ 0 \\ 0 \\ 0.03 \\ 0 \end{pmatrix}$$

Apply the Jacobi, Gauss-Seidel, and OR (with optimal relaxation factor) techniques, in such a way to find … the followings.

- Write a computer algorithm using (matlab, python or c++) to solve for the (non OR Gauss-Seidel) to get the results. 4
- Write a computer algorithm using (matlab, python or c++) to solve for the (OR Gauss-Seidel) to get the results. 4
- Write a brief Conclusion. 2
- You might use all zeros initial values.

Prepared by Prof. Ebrahim A. Mattar.

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO. 5
PRACTICAL: MAXIMUM OF (10) MARKS

LEAST SQUARE APPROXIMATION AND CURVE-FITTING

Write the experiment objectives, aims, procedures, results, and conclusions.

PART A: (5 MARKS)

- Find through hand calculations the polynomial (3rd order - polynomial) which passes through (-0.5,-1), (2.05,3.2) , (4,10.3), and (6,15.2) using the LEAST SQUARE APPROXIMATION.
- Use the model for Linear Interpolation to find the value of the Y axis, once X= 5.5.

PART B: (5 MARKS)

- Write a code (in Matlab, python, or n C++) (your code) for the creation of LINEAR INTERPOLATION (LEAST SQUARE APPROXIMATION AND CURVE-FITTING. (5 marks).
- Verify your coding and results using (matlab, c++, or python) solution, for any point on space.
- Compare your hand algorithm solution with the script coding.

Prepared by Prof. Ebrahim Mattar.

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO. 6
PRACTICAL: MAXIMUM OF (10) MARKS

NONLINEAR ITERATIVE METHODS GAUSS-SEIDEL AND GAUSS-JACOBI TECHNIQUES

For the following 6th order linear set or system, convert the below set of linear equations to be nonlinear as below (use the form for the nonlinear case) … given - nonzero - initial values,

$$\varphi = \begin{pmatrix} 3 & -0.4 & 0 & -0.9 & 0.02 & 0.01 \\ -1.2 & 4.2 & -1 & 0 & -0.88 & 0 \\ 0.03 & -1.2 & 4.5 & 0 & 0 & -1.2 \\ -0.9 & 0 & 0 & 3.7 & -1 & 0 \\ 0 & -1.2 & 0 & -1.1 & 4 & -1 \\ 0 & 0.02 & -1.4 & 0 & -0.7 & 3.5 \end{pmatrix}, \quad b = \begin{pmatrix} 1.2 \\ 0 \\ 0 \\ 0 \\ 0.03 \\ 0 \end{pmatrix}$$

$$f(\omega, \delta) = \omega + e^{-\omega} + \delta^3,$$

$$g(\omega, \delta) = \omega^2 + 2\omega\delta - \delta^2 + \tan(\omega)$$

Apply the (Gauss-Seidel), and (Gauss-Jacobi) techniques, in such a way to perform the following.

- Write a code for (Gauss-Seidel) to get the results. 4
- Write a code for (Gauss-Jacobi) to get the results. 4
- Compare number of iterations, for both Gauss-Seidel and Gauss-Jacobi. 2

You might use all zeros initial values. For the coding and scripting, please use (c++, or python, or matlab), with .. preferably python.

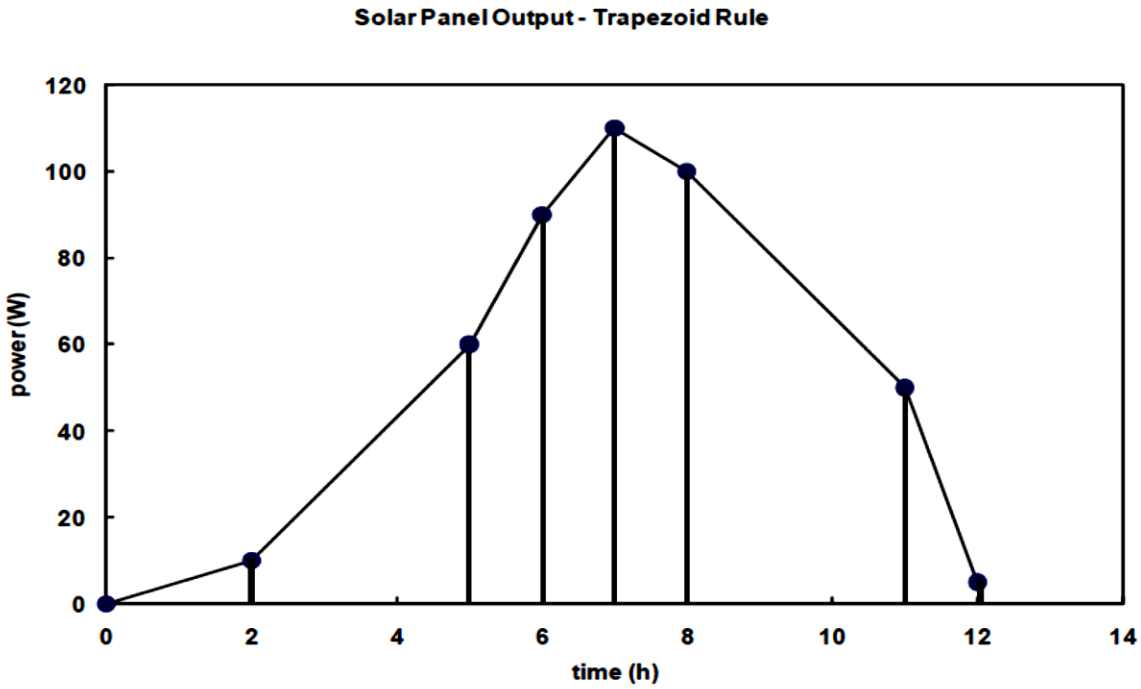
Prepared by Prof. Ebrahim A. Mattar.

UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO.7
PRACTICAL: MAXIMUM OF (10) MARKS.

NUMERICAL INTEGRATION AND DIFFERENTIATION PRACTICE



Source: MATLAB Tutorial – NUMERICAL INTEGRATION

```
% Numerical Integration ..
clear
clc

%  xl is the lower bound,
%  xu is the upper bound,
%  and wid is the interval width
```

```
xl=0.2;
   xu=10;
        wid=1;
 k=0;

I=0; % the integral needs to be initialized
for ii=xl:wid:xu-wid
   k=k+1;
   AI=((1/ii)+4*(1/(ii+wid/2))+(1/(ii+wid)))/6*wid;
     I=I+AI
     pp(k)=I;
end

plot(pp)
   grid

% USe different fuctions
% Covert this to a function ..
% function [I,err]=int1x(xl,xu,wid)
% End ..
```

- *Do the Numerical Differentiation and Numerical Integration for the same example (The given function) by hand. Show your steps.*
- *Do Numerical Differentiation and Numerical Integration for the same example (The given function) by computing coding (I prefer python). Show your steps.*
- *Explain the hand and code algorithms, and elaborate on the difficulties.*
- *Compare the two results.*

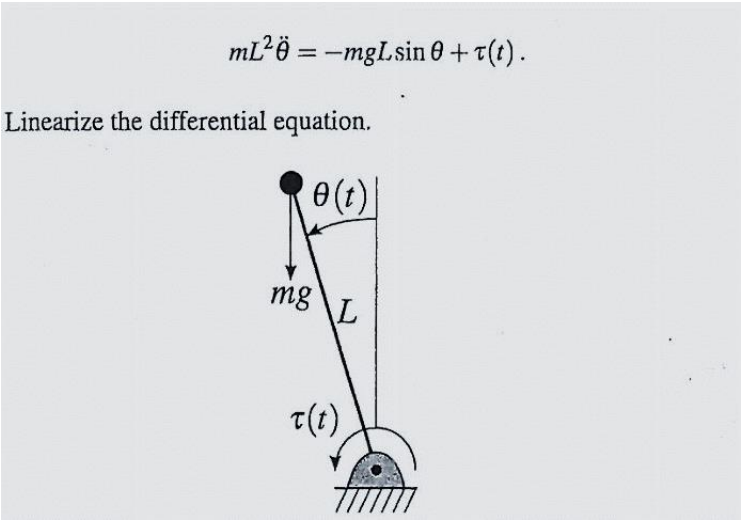
Prepared by Prof. Ebrahim A. Mattar

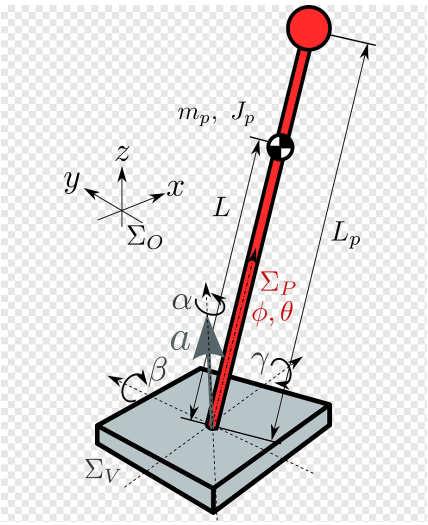
UNIVERSITY OF BAHRAIN
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NUMERICAL ANALYSIS
EENG 205

COMPUTER LAB NO.8
PRACTICAL: MAXIMUM OF (10) MARKS.

NUMERICAL SOLUTION OF NONLINEAR DIFFERENTIAL EQUATIONS





Quoted Figure, from nonlinear systems.
https://www.pngwing.com/en/search?q=inverted+Pendulum

For the above nonlinear 2nd order inverted pendulum dynamic system:

- Solve the above differential equation numerically by hand, show only 4 steps. (3 marks)
- Write a computer algorithm - script-code (*using c++ or matlab or python, I prefer python*) to solve the above nonlinear system dynamics and differential equation numerically. (3 marks)
- Use the matlab (ode23) function to simulate the system dynamics, hence comment on the hand calculations and the computer algorithm calculations. (4 marks)

Prepared by Prof. Ebrahim A. Mattar