

**PROJET**

***Instructor:***

**Dr. Umair Umar**

**Prepared by:**

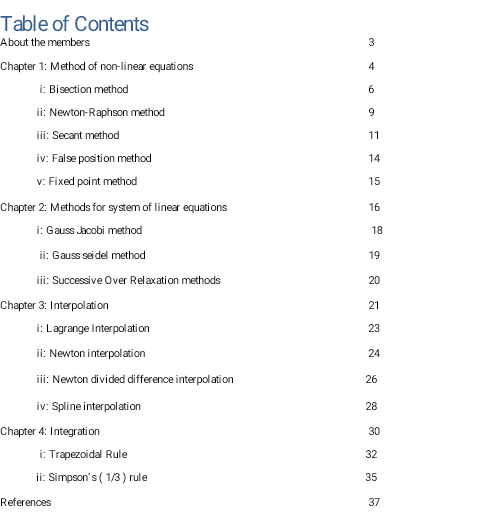
***NAME REG. NO***

**Muhammad Abuzar SP20-BSM-024**

**Esha Mumtaz SP20-BSM-012**

**Date:**

***June 21 ,2022***



***About the members***

This project was done under the supervision of Dr Umair Umar who was the instructor of the course Numerical Computations at the Department of Mathematics, CUI, and Islamabad. The project comprises of all the course contents studied in the semester of spring-2022.

Muhammad Abuzar alongside Esha Mumtaz worked upon the project. I Esha have done my FSC from Punjab groups of colleges. For now, I am enrolled in the Undergraduate program of Mathematics at CUI Islamabad. While Muhammad Abuzar completed FSC from FG Quaid e Azam inter college and completed my matriculation from Siddeeq public school . NOW I am enrolled in the Undergraduate program of Mathematics at CUI Islamabad.

***Some methods in numerical analysis:***

***1.*** ***Bisection Method***

***2.*** ***Regular false method***

***3.*** ***Newton Raphson method***

***4.*** ***Secant method***

***5.*** ***Fixed point method***

**BISECTION METHOD:**

The bisection method is used to find the roots of a polynomial equation. It separates the interval and subdivides the interval in which the root of the equation lies. The principle behind this method is the intermediate theorem for continuous functions. It works by narrowing the gap between the positive and negative intervals until it closes in on the correct answer. This method narrows the gap by taking the average of the positive and negative intervals. It is a simple method and it is relatively slow. The bisection method is also known as interval halving method, root-finding method, binary search method or dichotomy method.

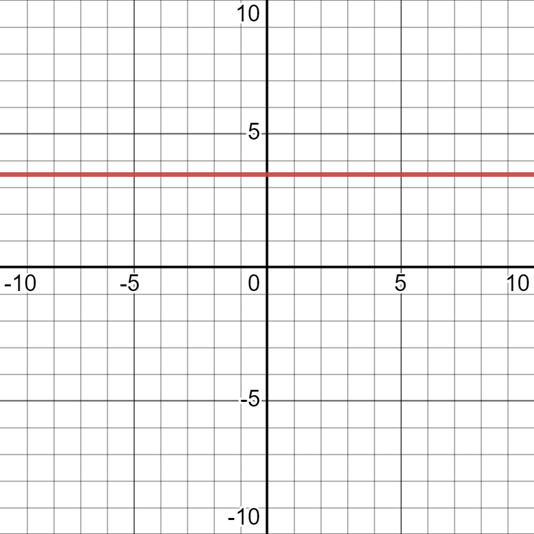
Let us consider a continuous function “f” which is defined on the closed interval [a, b], is given with f(a) and f(b) of different signs. Then by intermediate theorem, there exists a point x belong to (a, b) for which f(x) = 0.

**Algorithm**

|  |  |
| --- | --- |
| **Bisection method Steps (Rule)** | |
| **Step-1:** | Find points *a* and *b* such that *a*<*b* and *f*(*a*)⋅*f*(*b*)<0. |
| **Step-2:** | Take the interval [*a*,*b*] and find next value *x*0=*a*+*b*2 |
| **Step-3:** | If *f*(*x*0)=0 then *x*0 is an exact root, else if *f*(*a*)⋅*f*(*x*0)<0 then *b*=*x*0, else if *f*(*x*0)⋅*f*(*b*)<0 then *a*=*x*0. |
| **Step-4:** | Repeat steps 2 & 3 until *f*(*xi*)=0 or |*f*(*xi*)|≤Accuracy |

**EXAMPLE:**

### ***f*(*x*)=√12**



**Find a root of an equation *f*(*x*)=√12 using Bisection method**  
  
**Solution:**  
**Here** √12=0  
  
Let *f*(*x*)=√12  
  
x=√12  
  
∴*x*2=12  
  
∴*x*2-12=0  
  
Here

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***x*** | 0 | 1 | 2 | 3 | 4 |
| ***f*(*x*)** | -12 | -11 | -8 | -3 | 4 |

1*st* iteration :  
  
Here *f*(3)=-3<0 and *f*(4)=4>0  
  
∴ Now, Root lies between 3 and 4  
  
x0=3+42=3.5  
  
f(*x*0)=*f*(3.5)=3.52-12=0.25>0

Approximate root of the equation *x*2-12=0 using Bisection method is 3.4641 (After 12 iterations)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***n*** | ***a*** | ***f*(*a*)** | ***b*** | ***f*(*b*)** | ***c*=*a*+*b*2** | ***f*(*c*)** | **Update** |
| 1 | 3 | -3 | 4 | 4 | 3.5 | 0.25 | *b*=*c* |
| 2 | 3 | -3 | 3.5 | 0.25 | 3.25 | -1.4375 | *a*=*c* |
| 3 | 3.25 | -1.4375 | 3.5 | 0.25 | 3.375 | -0.6094 | *a*=*c* |
| 4 | 3.375 | -0.6094 | 3.5 | 0.25 | 3.4375 | -0.1836 | *a*=*c* |
| 5 | 3.4375 | -0.1836 | 3.5 | 0.25 | 3.4688 | 0.0322 | *b*=*c* |
| 6 | 3.4375 | -0.1836 | 3.4688 | 0.0322 | 3.4531 | -0.0759 | *a*=*c* |
| 7 | 3.4531 | -0.0759 | 3.4688 | 0.0322 | 3.4609 | -0.0219 | *a*=*c* |
| 8 | 3.4609 | -0.0219 | 3.4688 | 0.0322 | 3.4648 | 0.0051 | *b*=*c* |
| 9 | 3.4609 | -0.0219 | 3.4648 | 0.0051 | 3.4629 | -0.0084 | *a*=*c* |
| 10 | 3.4629 | -0.0084 | 3.4648 | 0.0051 | 3.4639 | -0.0016 | *a*=*c* |
| 11 | 3.4639 | -0.0016 | 3.4648 | 0.0051 | 3.4644 | 0.0018 | *b*=*c* |
| 12 | 3.4639 | -0.0016 | 3.4644 | 0.0018 | 3.4641 | 0.0001 | *b*=*c* |

**FALSE METHOD:**

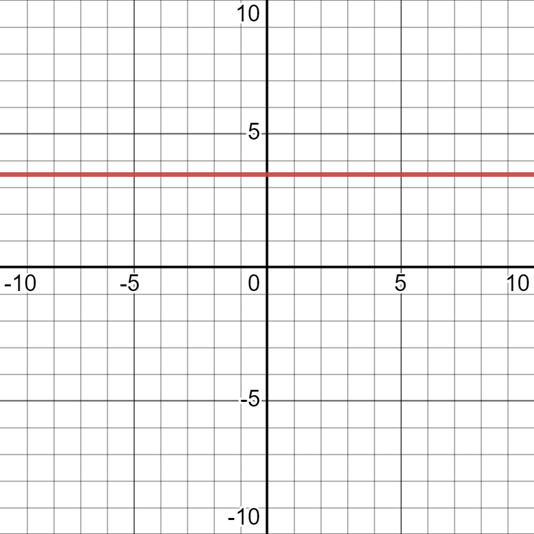
In mathematics, an ancient method of solving an equation in one variable is the **false position method** (method of false position)or **regula falsi method**. In simple words, the method is described as the trial and error approach of using “false” or “test” values for the variable and then altering the test value according to the result.

**Algorithm:**

|  |  |
| --- | --- |
| **False Position method (regula falsi method) Steps (Rule)** | |
| **Step-1:** | Find points *x*0 and *x*1 such that *x*0<*x*1 and *f*(*x*0)⋅*f*(*x*1)<0. |
| **Step-2:** | Take the interval [*x*0,*x*1] and find next value using Formula-1 : *x*2=*x*0-*f*(*x*0)⋅*x*1-*x*0*f*(*x*1)-*f*(*x*0) or Formula-2 : *x*2=*x*0⋅*f*(*x*1)-*x*1⋅*f*(*x*0)*f*(*x*1)-*f*(*x*0) or Formula-3 : *x*2=*x*1-*f*(*x*1)⋅*x*1-*x*0*f*(*x*1)-*f*(*x*0) (Using any of the formula, you will get same x2 value) |
| **Step-3:** | If *f*(*x*2)=0 then *x*2 is an exact root, else if *f*(*x*0)⋅*f*(*x*2)<0 then *x*1=*x*2, else if *f*(*x*2)⋅*f*(*x*1)<0 then *x*0=*x*2. |
| **Step-4:** | Repeat steps 2 & 3 until *f*(*xi*)=0 or |*f*(*xi*)|≤Accuracy |

**EXAMPLE:**

### ***f*(*x*)=√12**

  
**Find a root of an equation *f*(*x*)=√12 using False Position method (regula falsi method)**  
  
**Solution:**  
**Here** √12=0  
  
Let *f*(*x*)=√12  
  
x=√12  
  
∴*x*2=12  
  
∴*x*2-12=0  
  
Here

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***x*** | 0 | 1 | 2 | 3 | 4 |
| ***f*(*x*)** | -12 | -11 | -8 | -3 | 4 |

1*st* iteration :  
  
Here *f*(3)=-3<0 and *f*(4)=4>0  
  
∴ Now, Root lies between *x*0=3 and *x*1=4  
  
x2=*x*0-*f*(*x*0)⋅*x*1-*x*0*f*(*x*1)-*f*(*x*0)  
  
x2=3-(-3)⋅4-34-(-3)  
  
x2=3.4286  
  
f(*x*2)=*f*(3.4286)=3.42862-12=-0.2449<0  
  
Approximate root of the equation *x*2-12=0 using False Position method is 3.4641 (After 4 iterations)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***n*** | ***x*0** | ***f*(*x*0)** | ***x*1** | ***f*(*x*1)** | ***x*2** | ***f*(*x*2)** | **Update** |
| 1 | 3 | -3 | 4 | 4 | 3.4286 | -0.2449 | *x*0=*x*2 |
| 2 | 3.4286 | -0.2449 | 4 | 4 | 3.4615 | -0.0178 | *x*0=*x*2 |
| 3 | 3.4615 | -0.0178 | 4 | 4 | 3.4639 | -0.0013 | *x*0=*x*2 |
| 4 | 3.4639 | -0.0013 | 4 | 4 | 3.4641 | 0 | *x*0=*x*2 |

**DIFFERENCE:**

The difference between bisection method and false-position method is that in bisection method, both limits of the interval have to change. This is not the case for false position method, where one limit may stay fixed throughout the computation while the other guess converges on the root.

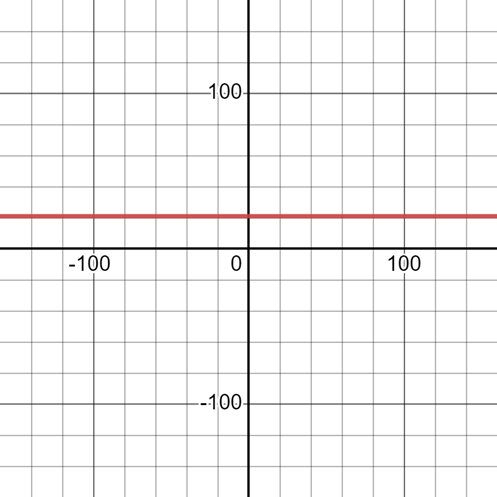
**SECANT METHOD:**

The **secant method** is a root-finding procedure in numerical analysis that uses a series of roots of secant lines to better approximate a root of a function f.

**Algorithm:**

|  |  |
| --- | --- |
| **Secant method Steps (Rule)** | |
| **Step-1:** | Find points *x*0 and *x*1 such that *x*0<*x*1 and *f*(*x*0)⋅*f*(*x*1)<0. |
| **Step-2:** | find next value using Formula-1 : *x*2=*x*0-*f*(*x*0)⋅*x*1-*x*0*f*(*x*1)-*f*(*x*0) or Formula-2 : *x*2=*x*0⋅*f*(*x*1)-*x*1⋅*f*(*x*0)*f*(*x*1)-*f*(*x*0) or Formula-3 : *x*2=*x*1-*f*(*x*1)⋅*x*1-*x*0*f*(*x*1)-*f*(*x*0) (Using any of the formula, you will get same x2 value) |
| **Step-3:** | If *f*(*x*2)=0 then *x*2 is an exact root, else *x*0=*x*1 and *x*1=*x*2 |
| **Step-4:** | Repeat steps 2 & 3 until *f*(*xi*)=0 or |*f*(*xi*)|≤Accuracy |

### ***f*(*x*)=3√48**

  
**Find a root of an equation *f*(*x*)=*cube*48√ using Secant method**  
  
**Solution:**  
**Here** 4813=0  
  
Let *f*(*x*)=4813  
  
x=*cube*48√  
  
∴*x*3=48  
  
∴*x*3-48=0  
  
Here

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***x*** | 0 | 1 | 2 | 3 | 4 |
| ***f*(*x*)** | -48 | -47 | -40 | -21 | 16 |

1*st* iteration :  
  
x0=3 and *x*1=4  
  
f(*x*0)=*f*(3)=-21 and *f*(*x*1)=*f*(4)=16  
  
∴*x*2=*x*0-*f*(*x*0)⋅*x*1-*x*0*f*(*x*1)-*f*(*x*0)  
  
x2=3-(-21)⋅4-316-(-21)  
  
x2=3.5676  
  
∴*f*(*x*2)=*f*(3.5676)=3.56763-48=-2.5936

Approximate root of the equation *x*3-48=0 using Secant method is 3.6342 (After 4 iterations)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***n*** | ***x*0** | ***f*(*x*0)** | ***x*1** | ***f*(*x*1)** | ***x*2** | ***f*(*x*2)** | **Update** |
| 1 | 3 | -21 | 4 | 16 | 3.5676 | -2.5936 | *x*0=*x*1 x1=*x*2 |
| 2 | 4 | 16 | 3.5676 | -2.5936 | 3.6279 | -0.2513 | *x*0=*x*1 x1=*x*2 |
| 3 | 3.5676 | -2.5936 | 3.6279 | -0.2513 | 3.6344 | 0.0047 | *x*0=*x*1 x1=*x*2 |
| 4 | 3.6279 | -0.2513 | 3.6344 | 0.0047 | 3.6342 | 0 | *x*0=*x*1 x1=*x*2 |

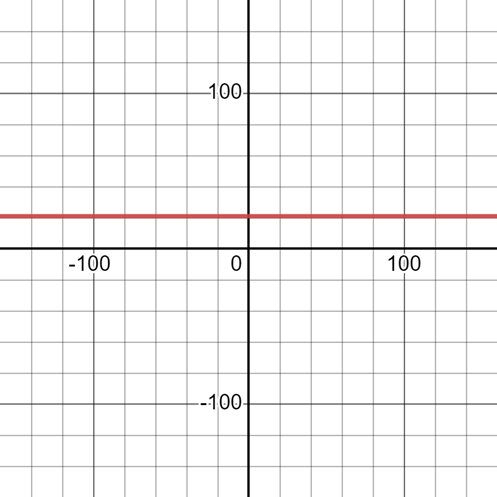
**Newton Raphson method:**

*In* [*numerical analysis*](https://en.wikipedia.org/wiki/Numerical_analysis)*, Newton's method, also known as the Newton–Raphson method, named after* [*Isaac Newton*](https://en.wikipedia.org/wiki/Isaac_Newton) *and* [*Joseph Raphson*](https://en.wikipedia.org/wiki/Joseph_Raphson)*, is a* [*root-finding algorithm*](https://en.wikipedia.org/wiki/Root-finding_algorithm) *which produces successively better* [*approximations*](https://en.wikipedia.org/wiki/Numerical_analysis) *to the* [*roots*](https://en.wikipedia.org/wiki/Root_of_a_function) *(or zeroes) of a* [*real*](https://en.wikipedia.org/wiki/Real_number)*-valued* [*function*](https://en.wikipedia.org/wiki/Function_(mathematics))*.*

**Algorithm**

|  |  |
| --- | --- |
| **Newton Raphson method Steps (Rule)** | |
| **Step-1:** | Find points *a* and *b* such that *a*<*b* and *f*(*a*)⋅*f*(*b*)<0. |
| **Step-2:** | Take the interval [*a*,*b*] and find next value *x*0=*a*+*b*2 |
| **Step-3:** | Find *f*(*x*0) and *f*′(*x*0) x1=*x*0-*f*(*x*0)*f*′(*x*0) |
| **Step-4:** | If *f*(*x*1)=0 then *x*1 is an exact root, else *x*0=*x*1 |
| **Step-5:** | Repeat steps 3 and 4 until *f*(*xi*)=0 or |*f*(*xi*)|≤Accuracy |

### ***f*(*x*)=3√48**

  
**Find a root of an equation *f*(*x*)=*cube*48√ using Newton Raphson method**  
  
**Solution:**  
**Here** 4813=0  
  
Let *f*(*x*)=4813  
  
x=*cube*48√  
  
∴*x*3=48  
  
∴*x*3-48=0  
  
d*dx*(*x*3-48)=3*x*2  
∴*f*′(*x*)=3*x*2  
  
Here

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***x*** | 0 | 1 | 2 | 3 | 4 |
| ***f*(*x*)** | -48 | -47 | -40 | -21 | 16 |

Here *f*(3)=-21<0and*f*(4)=16>0  
  
∴ Root lies between 3 and 4  
  
x0=3+42=3.5  
  
x0=3.5  
  
  
1*st* iteration :  
  
f(*x*0)=*f*(3.5)=3.53-48=-5.125  
  
f′(*x*0)=*f*′(3.5)=3⋅3.52=36.75  
  
x1=*x*0-*f*(*x*0)*f*′(*x*0)  
  
x1=3.5--5.12536.75  
  
x1=3.6395

Approximate root of the equation *x*3-48=0 using Newton Raphson method is 3.6342 (After 3 iterations)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***n*** | ***x*0** | ***f*(*x*0)** | ***f*′(*x*0)** | ***x*1** | **Update** |
| 1 | 3.5 | -5.125 | 36.75 | 3.6395 | *x*0=*x*1 |
| 2 | 3.6395 | 0.2069 | 39.7369 | 3.6342 | *x*0=*x*1 |
| 3 | 3.6342 | 0.0003 | 39.6233 | 3.6342 | *x*0=*x*1 |

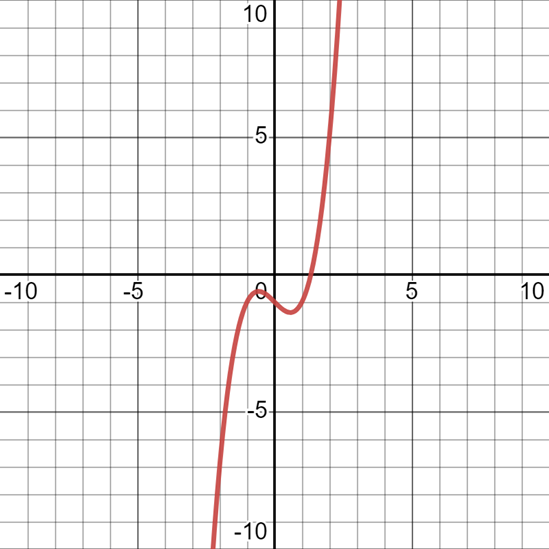
**FIXED POINT ITERATION METHOD:**

The fixed point iteration method uses the concept of a fixed point in a repeated manner to compute the solution of the given equation. A fixed point is a point in the domain of a function g such that g(x) = x. In the fixed point iteration method, the given function is algebraically converted in the form of g(x) = x.

**Algorithm:**

|  |  |
| --- | --- |
| **Fixed Point Iteration method Steps (Rule)** | |
| **Step-1:** | First write the equation *x*=*ϕ*(*x*) |
| **Step-2:** | Find points *a* and *b* such that *a*<*b* and *f*(*a*)⋅*f*(*b*)<0. |
| **Step-3:** | If *f*(*a*) is more closer to 0 then *f*(*b*) then *x*0=*a* else *x*0=*b* |
| **Step-4:** | *x*1=*ϕ*(*x*0) x2=*ϕ*(*x*1) x3=*ϕ*(*x*2) ... Repeat until |*f*(*xi*)-*f*(*xi*-1)|≈0 |

**Find a root of an equation *f*(*x*)=*x*3-*x*-1 using Fixed Point Iteration method**  
  
**Solution:**

  
**Method-1**  
**Let** *f*(*x*)=*x*3-*x*-1  
  
x3-*x*-1=0  
  
∴*x*3=*x*+1  
  
∴*x*=3√*x*+1  
  
∴*ϕ*(*x*)=3√*x*+1  
  
Here

|  |  |  |  |
| --- | --- | --- | --- |
| ***x*** | 0 | 1 | 2 |
| ***f*(*x*)** | -1 | -1 | 5 |

Here *f*(1)=-1<0 and *f*(2)=5>0  
  
∴ Root lies between 1 and 2  
  
x0=1+22=1.5  
  
  
x1=*ϕ*(*x*0)=*ϕ*(1.5)=1.35721  
  
x2=*ϕ*(*x*1)=*ϕ*(1.35721)=1.33086  
  
x3=*ϕ*(*x*2)=*ϕ*(1.33086)=1.32588  
  
x4=*ϕ*(*x*3)=*ϕ*(1.32588)=1.32494  
  
x5=*ϕ*(*x*4)=*ϕ*(1.32494)=1.32476  
  
  
Approximate root of the equation *x*3-*x*-1=0 using Iteration method is 1.32476

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***n*** | ***x*0** | ***x*1=*ϕ*(*x*0)** | **Update** | **Difference** **x1-*x*0** |
| 2 | 1.5 | 1.35721 | *x*0=*x*1 | 0.14279 |
| 3 | 1.35721 | 1.33086 | *x*0=*x*1 | 0.02635 |
| 4 | 1.33086 | 1.32588 | *x*0=*x*1 | 0.00498 |
| 5 | 1.32588 | 1.32494 | *x*0=*x*1 | 0.00094 |
| 6 | 1.32494 | 1.32476 | *x*0=*x*1 | 0.00018 |

**SYSTEM OF LINEAR EQUATIONS:**

In the system if linear equations we will solve Jacobi and GuassSiedal methods and their convergence criteria.

**JACOBI METHOD:**

**Jacobi method** is one the iterative methods for approximating the solution of a system of n linear equations in n variables. The Jacobi iterative method is considered as an iterative algorithm which is used for determining the solutions for the system of linear equations in numerical [linear algebra](https://byjus.com/maths/linear-algebra/), which is diagonally dominant. In this method, an approximate value is filled in for each diagonal element. Until it converges, the process is iterated. This algorithm was first called the Jacobi transformation process of matrix diagonalization. Jacobi Method is also known as the simultaneous displacement method.

## **Properties of Jacobian Method**

Adding the applications of the [Jacobian matrix](https://byjus.com/maths/jacobian/) in different areas, this method holds some important properties. The simplicity of this method is considered in both the aspects of good and bad. This method can be stated as good since it is the first iterative method and easy to understand. However, the method is also considered bad since it is not typically used in practice. Though there are cons, is still a good starting point for those who are willing to learn more useful but more complicated iterative methods.

**Problem: Gauss Jacobi 7y+2x=11,3x-y=5**  
  
**Solution:**

**7y+2x=11,3x-y=5**  
  
  
Total Equations are 2  
  
2*x*+7*y*=11  
  
3*x*-*y*=5  
  
  
The coefficient matrix of the given system is not diagonally dominant.  
Hence, we re-arrange the equations as follows, such that the elements in the coefficient matrix are diagonally dominant.  
3*x*-*y*=5  
  
2*x*+7*y*=11  
  
  
From the above equations  
*xk*+1=13(5+*yk*)  
  
*yk*+1=17(11-2*xk*)  
  
Initial gauss (*x*,*y*)=(0,0  
  
Solution steps are  
1*st* Approximation  
  
*x*1=13[5+(0)]=13[5]=1.6667  
  
*y*1=17[11-2(0)]=17[11]=1.5714

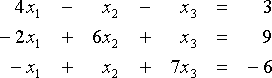
Solution By Gauss Jacobi Method.  
*x*=1.9998≅2  
  
*y*=0.9999≅1  
  
Iterations are tabulated as below

|  |  |  |
| --- | --- | --- |
| **Iteration** | **x** | **y** |
| 1 | 1.6667 | 1.5714 |
| 2 | 2.1905 | 1.0952 |
| 3 | 2.0317 | 0.9456 |
| 4 | 1.9819 | 0.9909 |
| 5 | 1.997 | 1.0052 |
| 6 | 2.0017 | 1.0009 |
| 7 | 2.0003 | 0.9995 |
| 8 | 1.9998 | 0.9999 |

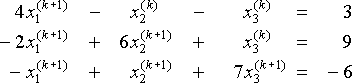
**GUASS SIEDAL METHOD:**

In [numerical linear algebra](https://en.wikipedia.org/wiki/Numerical_linear_algebra), the Gauss–Seidel method, also known as the Liebmann method or the method of successive displacement, is an [iterative method](https://en.wikipedia.org/wiki/Iterative_method) used to solve a [system of linear equations](https://en.wikipedia.org/wiki/System_of_linear_equations). It is named after the [German](https://en.wikipedia.org/wiki/Germany) [mathematicians](https://en.wikipedia.org/wiki/Mathematician) [Carl Friedrich Gauss](https://en.wikipedia.org/wiki/Carl_Friedrich_Gauss) and [Philipp Ludwig von Seidel](https://en.wikipedia.org/wiki/Philipp_Ludwig_von_Seidel), and is similar to the [Jacobi method](https://en.wikipedia.org/wiki/Jacobi_method). Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either [strictly diagonally dominant](https://en.wikipedia.org/wiki/Diagonally_dominant_matrix),[[1]](https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method#cite_note-1) or [symmetric](https://en.wikipedia.org/wiki/Symmetric_matrix) and [positive definite](https://en.wikipedia.org/wiki/Positive-definite_matrix). It was only mentioned in a private letter from Gauss to his student [Gerling](https://en.wikipedia.org/wiki/Christian_Ludwig_Gerling) in 1823.[[2]](https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method#cite_note-2) A publication was not delivered before 1874 by Seidel.[[3]](https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method#cite_note-3)

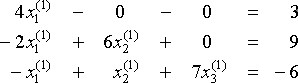
**EXAMPLE:**

 .

At each step, given the current values *x*1(*k*), *x*2(*k*), *x*3(*k*), we solve for *x*1(*k*+1), *x*2(*k*+1), *x*3(*k*+1) in

 .

To compare our results from the two methods, we again choose ***x***(0) = (0, 0, 0). We then find ***x***(1) = (*x*1(1), *x*2(1), *x*3(1)) by solving

 .

Let us be clear about how we solve this system. We first solve for *x*1(1) in the first equation and find that

*x*1(1) = 3/4 = 0.750.

We then solve for *x*2(1) in the second equation, using the new value of *x*1(1) = 0.750, and find that

*x*2(1) = [9 + 2(0.750)] / 6 = 1.750.

Finally, we solve for *x*3(1) in the third equation, using the new values of *x*1(1) = 0.750 and *x*2(1) = 1.750, and find that

*x*3(1) = [-6 + 0.750 − 1.750] / 7 = − 1.000.

The result of this first iteration of the Gauss-Seidel Method is

***x***(1) = (*x*1(1), *x*2(1), *x*3(1)) = (0.750, 1.750, − 1.000).

We iterate this process to generate a sequence of increasingly better approximations ***x***(0), ***x***(1), ***x***(2), … and find results similar to those that we found for Example 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***k*** | ***x*(*k*)** | | | ***x*(*k*) − *x*(*k*-1)** | | | ***e*(*k*) = *x* − *x*(*k*)** | | | **||*e*(*k*)||** |
| 0 | -0.000 | -0.000 | -0.000 | − | − | − | -0.000 | -0.000 | -1.000 | 2.449 |
| 1 | 0.750 | 1.750 | -1.000 | -0.000 | -0.000 | -1.000 | 0.250 | 0.250 | 0.000 | 0.354 |
| 2 | 0.938 | 1.979 | -1.006 | 0.188 | 0.229 | -0.006 | 0.063 | 0.021 | 0.006 | 0.066 |
| 3 | 0.993 | 1.999 | -1.001 | 0.056 | 0.020 | 0.005 | 0.007 | 0.001 | 0.001 | 0.007 |
| 4 | 0.999 | 2.000 | -1.000 | 0.006 | 0.001 | 0.001 | 0.001 | 0.000 | 0.000 | 0.001 |
| 5 | 1.000 | 2.000 | -1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

**P: (800) 331-1622**

**SOR:**

### In numerical linear algebra, the method of successive over-relaxation (SOR) is a variant of the Gauss–Seidel method for solving a linear system of equations, resulting in faster convergence. A similar method can be used for any slowly converging iterative process.

**EXAMPLE:**

### 10*x*+2*y*-*z*=7,*x*+8*y*+3*z*=-4,-2*x*-*y*+10*z*=9

**Solution:**  
We know that, for symmetric positive definite matrix the SOR method converges for values of the relaxation parameter *w* from the interval 0<*w*<  
The iterations of the SOR method  
1. Total Equations are 3  
  
10*x*+2*y*-*z*=7  
  
*x*+8*y*+3*z*=-4  
  
-2*x*-*y*+10*z*=9  
  
  
2. From the above equations, First write down the equations for Gauss Seidel method  
*xk*+1=110(7-2*yk*+*zk*)  
  
*yk*+1=18(-4-*xk*+1-3*zk*)  
  
*zk*+1=110(9+2*xk*+1+*yk*+1)  
  
3. Now multiply the right hand side by the parameter *w* and add to it the vector *xk* from the previous iteration multiplied by the factor of (1-*w*)  
  
*xk*+1=(1-*w*)⋅*xk*+*w*⋅110(7-2*yk*+*zk*)  
  
*yk*+1=(1-*w*)⋅*yk*+*w*⋅18(-4-*xk*+1-3*zk*)  
  
*zk*+1=(1-*w*)⋅*zk*+*w*⋅110(9+2*xk*+1+*yk*+1)  
  
4. Initial gauss (*x*,*y*,*z*)=(0,0,0) and *w*=1.25  
  
Solution steps are  
1*st* Approximation  
  
*x*1=(1-1.25)⋅0+1.25⋅110[7-2(0)+(0)]=(-0.25)⋅0+1.25⋅110[7]=0+0.875=0.875  
  
*y*1=(1-1.25)⋅0+1.25⋅18[-4-(0.875)-3(0)]=(-0.25)⋅0+1.25⋅18[-4.875]=0±0.7617=-0.7617  
  
*z*1=(1-1.25)⋅0+1.25⋅110[9+2(0.875)+(-0.7617)]=(-0.25)⋅0+1.25⋅110[9.9883]=0+1.2485=1.2485

iterations are tabulated as below

|  |  |  |  |
| --- | --- | --- | --- |
| **Iteration** | **x** | **y** | **z** |
| 1 | 0.875 | -0.7617 | 1.2485 |
| 2 | 1.0027 | -1.1765 | 0.9165 |
| 3 | 1.033 | -0.9219 | 1.0389 |
| 4 | 0.9771 | -1.0342 | 0.9803 |
| 5 | 1.0118 | -0.9841 | 1.0099 |
| 6 | 0.9943 | -1.0077 | 0.9951 |
| 7 | 1.0027 | -0.9962 | 1.0024 |
| 8 | 0.9987 | -1.0018 | 0.9988 |
| 9 | 1.0007 | -0.9991 | 1.0006 |
| 10 | 0.9997 | -1.0004 | 0.9997 |
| 11 | 1.0002 | -0.9998 | 1.0001 |

**INTERPOLATION:**

It is an estimation if values with in two known values in a sequence if value.

**NEWTON FARWARD DIFFERENCE:**

Newton's forward interpolation is a polynomial interpolation that is based on Newton's forward operator's initial value and degrees. The amount of data points is one less than the degree of polynomial fitted. This form of interpolation is only utilised when the data points are evenly spaced.

**Formula**

|  |
| --- |
| Newton's Forward Difference formula |
| *p*=*x*-*x*0*h* *y*(*x*)=*y*0+*p*Δ*y*0+*p*(*p*-1)2!⋅Δ2*y*0+*p*(*p*-1)(*p*-2)3!⋅Δ3*y*0+*p*(*p*-1)(*p*-2)(*p*-3)4!⋅Δ4*y*0+... |

**Find Solution of an equation x^3-x+1 using Newton's Forward Difference formula**

**x1 = 2 and x2 = 4**

**x = 2.25**

**Step value (h) = 0.5**

**Finding f(2)**

**Solution:**

Equation is *f*(*x*)=*x*3-*x*+1.

The value of table for *x* and *y*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | 2 | 2.5 | 3 | 3.5 | 4 |
| **y** | 7 | 14.125 | 25 | 40.375 | 61 |

Newton's forward difference interpolation method to find solution

Newton's forward difference table is

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | **y** | **Δ*y*** | **Δ2*y*** | **Δ3*y*** | **Δ4*y*** |
| **2** | **7** |  |  |  |  |
|  |  | **7.125** |  |  |  |
| 2.5 | 14.125 |  | **3.75** |  |  |
|  |  | 10.875 |  | **0.75** |  |
| 3 | 25 |  | 4.5 |  | **0** |
|  |  | 15.375 |  | 0.75 |  |
| 3.5 | 40.375 |  | 5.25 |  |  |
|  |  | 20.625 |  |  |  |
| 4 | 61 |  |  |  |  |

The value of *x* at you want to find the *f*(*x*):*x*=2.25  
  
h=*x*1-*x*0=2.5-2=0.5  
  
p=*x*-*x*0*h*=2.25-20.5=0.5  
  
Newton's forward difference interpolation formula is  
y(*x*)=*y*0+*p*Δ*y*0+*p*(*p*-1)2!⋅Δ2*y*0+*p*(*p*-1)(*p*-2)3!⋅Δ3*y*0+*p*(*p*-1)(*p*-2)(*p*-3)4!⋅Δ4*y*0  
  
y(2.25)=7+0.5×7.125+0.5(0.5-1)2×3.75+0.5(0.5-1)(0.5-2)6×0.75+0.5(0.5-1)(0.5-2)(0.5-3)24×0  
  
y(2.25)=7+3.5625-0.4688+0.0469+0  
  
y(2.25)=10.1406

**NEWTON BACKWARD DIFFERENCE:**

In order to reduce the number of numerical computations required to compute a large number of interpolated values using the existing interpolation formula.

**Formula**

|  |
| --- |
| Newton's Backward Difference formula |
| *p*=*x*-*xnh* *y*(*x*)=*yn*+*p*∇*yn*+*p*(*p*+1)2!⋅∇2*yn*+*p*(*p*+1)(*p*+2)3!⋅∇3*yn*+*p*(*p*+1)(*p*+2)(*p*+3)4!⋅∇4*yn*+... |

**2. Find Solution using Newton's Backward Difference formula**

|  |  |
| --- | --- |
| x | f(x) |
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 10 |

**x = 4**  
  
**Solution:**  
**The value of table for** *x* and *y*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **x** | 0 | 1 | 2 | 3 |
| **y** | 1 | 0 | 1 | 10 |

Newton's backward difference interpolation method to find solution  
  
Newton's backward difference table is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **x** | **y** | **∇*y*** | **∇2*y*** | **∇3*y*** |
| 0 | 1 |  |  |  |
|  |  | -1 |  |  |
| 1 | 0 |  | 2 |  |
|  |  | 1 |  | **6** |
| 2 | 1 |  | **8** |  |
|  |  | **9** |  |  |
| **3** | **10** |  |  |  |

The value of x at you want to find the *f*(*x*):*x*=4

*h*=*x*1-*x*0=1-0=1  
  
p=*x*-*xnh*=4-31=1  
  
Newton's backward difference interpolation formula is  
y(*x*)=*yn*+*p*∇*yn*+*p*(*p*+1)2!⋅∇2*yn*+*p*(*p*+1)(*p*+2)3!⋅∇3*yn*  
  
*y*(4)=10+1×9+1(1+1)2×8+1(1+1)(1+2)6×6  
  
y(4)=10+9+8+6  
  
y(4)=33

**NEWTONS DIVIDED DIFFERENCE INTERPOLATION METHOD:**

(x - xk-1) f [x0, x1, . . ., xk]. This formula is called Newton's Divided Difference Formula. Once we have the divided differences of the function f relative to the tabular points then we can use the above formula to compute f(x) at any non tabular point. using Newton's divided difference formula.

**Formula**

|  |
| --- |
| Newton's Divided Difference Interpolation formula |
| *y*(*x*)=*y*0+(*x*-*x*0)*f*[*x*0,*x*1]+(*x*-*x*0)(*x*-*x*1)*f*[*x*0,*x*1,*x*2]+... |

**Examples**  
**1. Find Solution using Newton's Divided Difference Interpolation formula**

|  |  |
| --- | --- |
| x | f(x) |
| 300 | 2.4771 |
| 304 | 2.4829 |
| 305 | 2.4843 |
| 307 | 2.4871 |

**x = 301**  
  
**Solution:**  
**The value of table for** *x* and *y*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **x** | 300 | 304 | 305 | 307 |
| **y** | 2.4771 | 2.4829 | 2.4843 | 2.4871 |

Numerical divided differences method to find solution  
  
Newton's divided difference table is

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **y** | **1*st* order** | **2*nd* order** |
| 300 | 2.4771 |  |  |
|  |  | 0.00145 |  |
| 304 | 2.4829 |  | 0 |
|  |  | 0.0014 |  |
| 305 | 2.4843 |  | 0 |
|  |  | 0.0014 |  |
| 307 | 2.4871 |  |  |

The value of *x* at you want to find the *f*(*x*):*x*=301

Newton's divided difference interpolation formula is

f(*x*)=*y*0+(*x*-*x*0)*f*[*x*0,*x*1]+(*x*-*x*0)(*x*-*x*1)*f*[*x*0,*x*1,*x*2]  
  
y(301)=2.4771+(301-300)×0.00145+(301-300)(301-304)×0  
  
y(301)=2.4771+(1)×0.00145+(1)(-3)×0  
  
y(301)=2.4771+0.00145+0  
  
y(301)=2.47858

**Spline interpolation:**

Spline interpolation similar to the Polynomial interpolation x’ uses low-degree polynomials in each of the intervals and chooses the polynomial pieces such that they fit smoothly together. The resulting function is called a spline.

**Spline interpolation divided into 3 types**

**•** **Linear spline**

**•** **Quardatic spline**

**•** **Cubic spline**

**LINEAR SPLINE:**

The linear spline represents a set of line segments between the two adjacent data points (Vk,Ik) and (Vk+1,Ik+1). The equations for each line segment can be immediately found in a simple form: Ik(V) = Ik + ( Ik+1 - Ik) ( V - Vk ) / (Vk+1 - Vk), where V = [Vk,Vk+1] and k = 0,1,...,(n-1).

For a set of data points usually termed **knots**: Shape

Description automatically generated with medium confidence, the linear spline is



The C0 conditions, Shape

Description automatically generated with medium confidence and Shape

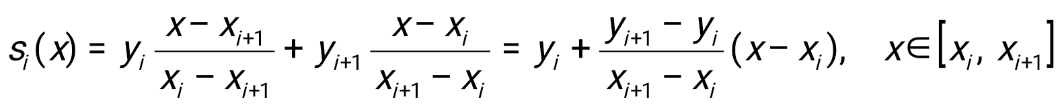
Description automatically generated with medium confidence yield Shape

Description automatically generated with medium confidence equations at interior points. These along with one condition at each end point give a total of Shape

Description automatically generated with medium confidence equations to match the Shape

Description automatically generated with medium confidence unknowns: Shape

Description automatically generated with medium confidence. By applying these, we get



which results in straight lines joining neighboring points.

Clearly, Shape

Description automatically generated with medium confidence is the Lagrange interpolation formula for the data set consisting of the following two points: Shape

Description automatically generated with medium confidence and Shape

Description automatically generated with medium confidence. Hence it is the solution approximation for linear finite elements in one dimension.

Find the linear splines for the following data set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 |
| x | 0 | 5 | 7 | 8 | 10 |
| y | 0 | 2 | -1 | -2 | 20 |

Linear spline fit

Shape

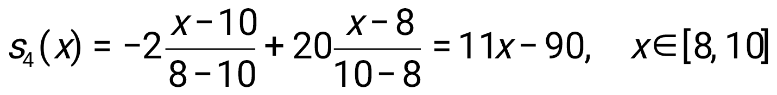
Description automatically generated with medium confidence

Shape

Description automatically generated with medium confidence

Shape

Description automatically generated with medium confidence



The “[*linear\_spline.m*](http://www.esm.psu.edu/courses/emch407/njs/m-files02/linear_spline.m)” function is built according to the following algorithm:

(1) Input x where the interpolation value y is needed;

Check to see if x is out of range; if yes, abort the program; if no, proceed;

(2) Find the data interval in which x resides;

(3) Calculate y according to Equation

![Shape

Description automatically generated with medium confidence](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABQAAAADCAYAAACTWi8uAAAAAXNSR0IArs4c6QAAAARzQklUCAgICHwIZIgAAAAVSURBVAiZY2AgHvynukJqm3WGGEUA08cDypz+83IAAAAASUVORK5CYII=)

![Shape

Description automatically generated with medium confidence](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABQAAAADCAYAAACTWi8uAAAAAXNSR0IArs4c6QAAAARzQklUCAgICHwIZIgAAAAbSURBVAiZY2CgAfhHhp5ULGJMuBT/J8MCOAAABQcCZb6QwGMAAAAASUVORK5CYII=)

**CUBIC SPLINE:**

Cubic spline interpolation is a way of finding a curve that connects data points with a degree of three or less. Splines are polynomial that are smooth and continuous across a given plot and also continuous first and second derivatives where they join.

**Formula**

|  |
| --- |
| Cubic spline formula is *f*(*x*)=(*xi*-*x*)36*hMi*-1+(*x*-*xi*-1)36*hMi*+(*xi*-*x*)*h*(*yi*-1-*h*26*Mi*-1)+(*x*-*xi*-1)*h*(*yi*-*h*26*Mi*)→(1) |
| *Mi*-1+4*Mi*+*Mi*+1=6*h*2(*yi*-1-2*yi*+*yi*+1)→(2) |

**Calculate Cubic Splines**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 1 | 2 | 3 | 4 |
| Y | 1 | 5 | 11 | 8 |

**y(1.5), y'(2)**  
  
**Solution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **x** | 1 | 2 | 3 | 4 |
| **y** | 1 | 5 | 11 | 8 |

Cubic spline formula is  
f(*x*)=(*xi*-*x*)36*hMi*-1+(*x*-*xi*-1)36*hMi*+(*xi*-*x*)*h*(*yi*-1-*h*26*Mi*-1)+(*x*-*xi*-1)*h*(*yi*-*h*26*Mi*)→(1)  
  
We have, *Mi*-1+4*Mi*+*Mi*+1=6*h*2(*yi*-1-2*yi*+*yi*+1)→(2)  
  
Here *h*=1,*n*=3  
  
M0=0,*M*3=0  
  
Substitute *i*=1 in equation (2)  
  
M0+4*M*1+*M*2=6*h*2(*y*0-2*y*1+*y*2)  
  
⇒0+4*M*1+*M*2=61⋅(1-2⋅5+11)  
  
⇒4*M*1+*M*2=12  
  
Substitute *i*=2 in equation (2)

*M*1+4*M*2+*M*3=6*h*2(*y*1-2*y*2+*y*3)  
  
⇒*M*1+4*M*2+0=61⋅(5-2⋅11+8)  
  
⇒*M*1+4*M*2=-54  
  
Solving these 2 equations using elimination method  
Substitute *i*=1 in equation (1), we get cubic spline in 1*st* interval [*x*0,*x*1]=[1,2]  
  
f1(*x*)=(*x*1-*x*)36*hM*0+(*x*-*x*0)36*hM*1+(*x*1-*x*)*h*(*y*0-*h*26*M*0)+(*x*-*x*0)*h*(*y*1-*h*26*M*1)  
  
f1(*x*)=(2-*x*)36⋅0+(*x*-1)36⋅6.8+(2-*x*)1(1-16⋅0)+(*x*-1)1(5-16⋅6.8)  
  
f1(*x*)=1.1333*x*3-3.4*x*2+6.2667*x*-3, for 1≤*x*≤2  
  
  
  
Substitute *i*=2 in equation (1), we get cubic spline in 2*nd* interval [*x*1,*x*2]=[2,3]  
  
f2(*x*)=(*x*2-*x*)36*hM*1+(*x*-*x*1)36*hM*2+(*x*2-*x*)*h*(*y*1-*h*26*M*1)+(*x*-*x*1)*h*(*y*2-*h*26*M*2)  
  
f2(*x*)=(3-*x*)36⋅6.8+(*x*-2)36⋅-15.2+(3-*x*)1(5-16⋅6.8)+(*x*-2)1(11-16⋅-15.2)  
  
f2(*x*)=-3.6667*x*3+25.4*x*2-51.3333*x*+35.4, for 2≤*x*≤3

Substitute *i*=3 in equation (1), we get cubic spline in 3*rd* interval [*x*2,*x*3]=[3,4]  
  
f3(*x*)=(*x*3-*x*)36*hM*2+(*x*-*x*2)36*hM*3+(*x*3-*x*)*h*(*y*2-*h*26*M*2)+(*x*-*x*2)*h*(*y*3-*h*26*M*3)  
  
f3(*x*)=(4-*x*)36⋅-15.2+(*x*-3)36⋅0+(4-*x*)1(11-16⋅-15.2)+(*x*-3)1(8-16⋅0)  
  
f3(*x*)=2.5333*x*3-30.4*x*2+116.0667*x*-132, for 3≤*x*≤4  
  
  
  
For *y*(1.5), 1.5∈[1,2], so substitute *x*=1.5 in *f*1(*x*), we get  
  
f1(1.5)=2.575  
  
For *y*′(2), 2∈[1,2], so find *f*′1(*x*)  
  
f′1(*x*)=3.4*x*2-6.8*x*+6.2667  
  
Now substitute *x*=2 in *f*′1(*x*), we get  
  
f′1(2)=6.2667

**QUADRATIC SPLINE:**

A Quadratic Spline is the creation of a set of polynomial functions that are quadratic, or, easier to understand, follow the format f(x)=ax²+bx+c, where a, b and c are the values obtained while doing the Splines to create the desired functions.

A quadratic spline has a quadratic function for each data interval



which is constrained to satisfy the C0 and C1 conditions.

For the C0 condition, we get

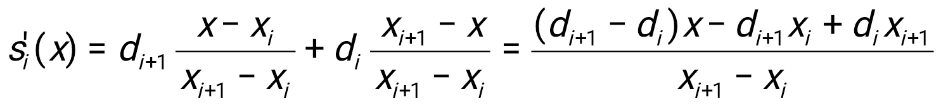


For the C1 condition, we get



From the above three requirements, there are *3n-4* constraint conditions. But the splines require a total of *3n-3* conditions, so we are one condition short and the problem is not completely defined. Usually, s′1(x1) = 0 is used for the additional condition. This results in the so-called **natural** quadratic spline. Certainly other conditions can be used, such as s′1(x1) = s′n-1(xn ).

To find the formula of the splines, first denote di = s′i (xi ), then since s′(x) is a linear spline for the data set (xi, di, i = 1,2, …, n), we have



Shape

Description automatically generated with medium confidence

which is of course is the linear equation for the slope.

Integrating over *x*, we get

Shape

Description automatically generated with medium confidence

after solving for the constant of integration using the C0 boundary condition: Shape

Description automatically generated with medium confidence. So the quadratic spline is defined once we have the Shape

Description automatically generated with medium confidence’s.

Letting *x = xi+1*, we get

Shape

Description automatically generated with medium confidence

which leads to

Shape

Description automatically generated with medium confidence

which is a recursive scheme. If *d*1 is known, then *di* can be derived from the above equation.

Various conditions can be imposed to obtain *d*1:

* For the natural spline, Shape

  Description automatically generated with medium confidence.
* If *d1 = d2*, then *d1* can be calculated by

Shape

Description automatically generated with medium confidence

* If *d1 = dn*, then *d1* can be calculated by

Shape

Description automatically generated with medium confidence

This condition can not be applied when n is an odd number.

The “[*quadratic\_spline.m*](http://www.esm.psu.edu/courses/emch407/njs/m-files02/quadratic_spline.m)” function is built according to the following algorithm

1. Input *x* where the interpolation value *y* is needed;

Check to see if *x* is out of range; if yes, abort the program; if no, proceed;

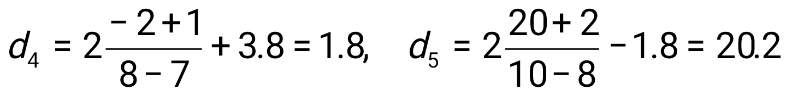
1. Find the data interval where *x* belongs;
2. Calculate *d1*; Calculate *di* (Equation 4.2.5);
3. Calculate *y* according to Equation .

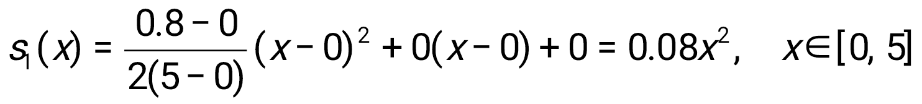
Example 4.2.1. Given the same data points in Example 4.1.1, derive the quadratic splines.

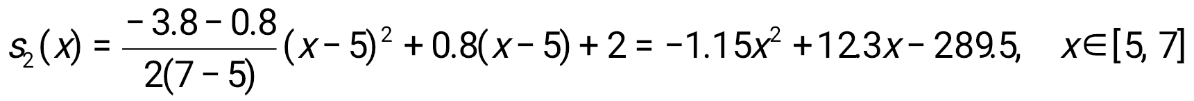
Using the natural quadratic splines, where *d1 = 0*, then we have

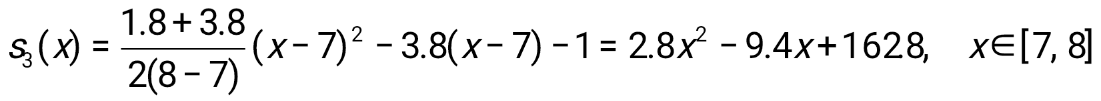
Shape

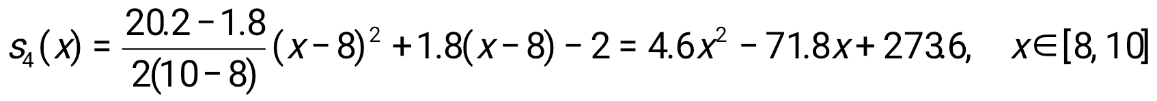
Description automatically generated with medium confidence











The results and the splines due to *d1 = d2*, are plotted in Figure 4.2.1. Near *x = 0*, the natural spline has a flatter curve due to zero slope at x = 0. The other spline has a linear slope in the first interval.

**BISECTION METHOD**

**Graphical user interface, text, application, email

Description automatically generated**

**REGULA FALSE METHODText

Description automatically generated**

**Newton Raphson Method**

**Text

Description automatically generated**

**Secant Method**

**Text

Description automatically generated**

**Fixed Point Method**

**Text

Description automatically generated**

**Jacobi Method**

**Graphical user interface, text

Description automatically generated**

**References:**

<https://sites.google.com/site/knowyourrootsmaxima/introduction/bisectionmethod>

<https://sites.google.com/site/knowyourrootsmaxima/introduction/newtonmethod> <https://www.youtube.com/watch?v=vfEq-WKyVbQ&t=30s>

<https://www.youtube.com/watch?v=x7m0m5A5EiQ> <http://article.sapub.org/10.5923.j.ajsp.20170702.01.html> <http://compmath-journal.org/dnload/Robin-Kumar-and-Vipan-/CMJV06I06P0290.pdf>

<https://www.youtube.com/watch?v=8F-IY4oihR4>

<http://compmath-journal.org/dnload/Robin-Kumar-and-Vipan-/CMJV06I06P0290.pdf>

<https://www.math.usm.edu/lambers/mat772/fall10/lecture17.pdf>

<https://dmpeli.math.mcmaster.ca/Matlab/Math4Q3/NumMethods/Lecture2-3.html>

<https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/10RootFinding/bisection/examples.html>

<http://compmath-journal.org/dnload/Robin-Kumar-and-Vipan-/CMJV06I06P0290.pdf>

<https://files.eric.ed.gov/fulltext/EJ1231189.pdf>