numerical analysis

**Project**

*NUMERICAL ANALYSIS*

Submitted by:

Aneeqa Altaf (CIIT/SP20-BSM-007/ISB)

Tehreem Amjad (CIIT/SP20-BSM-040/ISB)

Submitted to:

Dr. Umair Umer

**Table of content**

|  |  |  |
| --- | --- | --- |
| Sr.no. | Topics | Pg.no. |
| 1. | Solution of one variable   * Bisection method * False position method * Newton Raphson method * Secant method * Fix point method | 4 |
| 2. | Direct method for solving linear system   * Gauss elimination method * Pivot elimination method * Matrix transformation | 26 |
| 3. | Iterative techniques in matrix algebra   * Norm of the vector * Matrix norm * Eigenvalue * Eigenvector * Spectral radius * Convergence | 28 |
| 4. | Iterative method   * Gauss Jacobi method * Gauss-seidel method * Successive over relaxation | 32 |
| 5. | Interpolation and polynomial approximation   * Newton backward difference * Newton forward difference * Newton central difference * Shift operator * Average operator * Lagrange interpolation * Newton divided difference | 39 |
| 6. | Spline interpolation   * Linear interpolation * Quadratic interpolation * Cubic interpolation | 49 |
| 7. | Numerical integration   * Trapezoidal method * Simpson’s rule * Simpson’srule | 60 |
| 8. | Reference | 65 |

***Solution of one variable***

**Bisection Method:**

Introduction:

The bisection method is one of the techniques to find the root value of the polynomial equation. This method follows the principle of intermediate theorem which states “*An equation f(x)=0 is a real continuous function, has at least one root between and if f ()f()<0.* So, in this method we shall take two points as an initial guess such that both the points are used in function and product of those function is a negative number.

.

.

Algorithm:

1. Choose two points as an initial guess. Consider these initial guess as and such that f()f()<0 this means f(x) changes their sign between and .
2. Now find the midpoint () of the points as = and find the value of function at .
3. Then check the following
4. If f ()f()<0 then the root is in between . Thus .
5. If then the root value is in between .
6. If then the root value is .
7. Now find the absolute relative approximation error after every iteration as

where

.

.

1. Compare the absolute relative approximate error with the given tolerance .

* If then repeat the algorithm from 2nd step.
* If then stop the algorithm.

Advantages:

* It is convergent.
* Root bracket gets halved with each iteration.

Disadvantages:

* This method is slow convergence.
* If one of the initial guesses is close to the root, the convergence is slower.
* If function f (x) does not touches the x-axis then it is unable to find the root e.g.
* Function changes sign but root does not exist e.g.

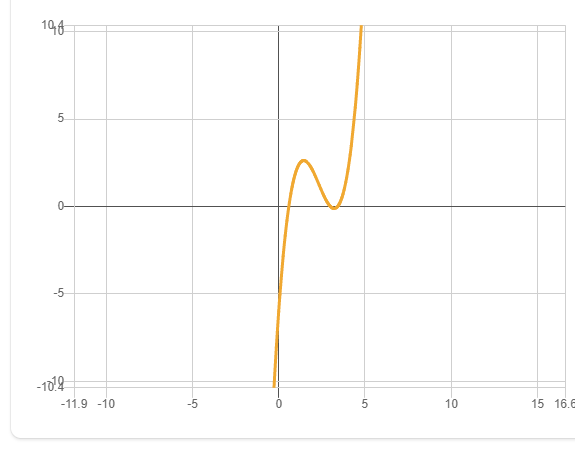
Example:

Let

So,

As, so according to theorem there exist at least one root between 0 and 1.

Graph:



Iteration 1:

.

.

= 0.5

.

.

.

Hence, the root lies between 0.5 and 1

Iteration 2:

.

.

As,

So,

Hence, the root lies between 0.5 and 0.75.

Absolute Relative approximation error:

The number of significant digits at least correct is given by the largest value or m for which

So, number of significant digits at least correct in the root at the end is 2.

Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| iteration | 1st initial guess=a | 2nd initial guess=b | F(a) | F(b) | D=(a+b)/2 | F(d) | Error |
| 1. | 0 | 1 | -6 | 2 | 0.5 | -0.625 | - |
| 2. | 0.5 | 1 | -0.625 | 2 | 0.75 | 0.9843 | 33.33% |
| 3. | 0.5 | 0.75 | -0.625 | 0.9844 | 0.625 | 0.2597 | 20% |
| 4. | 0.5 | 0.625 | -0.625 | 0.2598 | 0.5625 | -0.1618 | 11.11% |
| 5. | 0.5625 | 0.625 | -0.1618 | 0.2598 | 0.5938 | 0.0543 | 5.2711% |
| 6. | 0.5625 | 0.5938 | -0.1618 | 0.0544 | 0.5781 | -0.0524 | 2.7157% |
| 7. | 0.5781 | 0.5938 | -0.0524 | 0.0544 | 0.5859 | 0.0007 | 1.3312% |
| 8. | 0.5781 | 0.5859 | -0.0524 | 0.0007 | 0.582 | -0.0259 | 0.6701% |
| 9. | 0.582 | 0.5859 | -0.0259 | 0.0007 | 0.5839 | -0.0125 | 0.3253% |
| 10. | 0.5839 | 0.5859 | -0.0125 | 0.0007 | 0.5849 | -0.00605 | 0.1709% |
| 11. | 0.5849 | 0.5859 | -0.0060 | 0.0007 | 0.5854 | -0.00263 | 0.0854% |
| 12. | 0.5854 | 0.5859 | -0.0026 | 0.0007 | 0.5856 | -0.0001 | 0.0341% |
| 13. | 0.5856 | 0.5859 | -0.0001 | 0.0007 | 0.5857 | -0.0005 | 0.01707% |
| 14. | 0.5857 | 0.5859 | -0.0005 | 0.0007 | 0.5858 | 0.000009 | 0.01707% |
| 15. | 0.5857 | 0.5858 | -0.0005 | 0.00009 | 0.58575 | -0.00024 | 0.0085% |
| 16.. | 0.58575 | 0.5858 | -0.0002 | 0.00009 | 0.58578 | -0.00001 | 0.0051% |
| 17. | 0.58578 | 0.5858 | -0.00001 | 0.00009 | 0.58578 | -0.000004 | 0.0051% |
| 18. | 0.58578 | 0.5858 | -0.000004 | 0.00009 | 0.58579 | 0.00002 | 0.0017% |
| 19. | 0.58578 | 0.58579 | -0.000004 | 0.00002 | 0.58579 | -0.000009 | 0.00001% |

So, the root value = 0.58579.

Code:

from math import sin

def bisection (x0, x1, e):

step = 1

condition = True

while condition:

x2 = (x0+x1)/2

print ('iteration %d, x2 = %0.6f and f(x2)= %0.6f' %(step,x2,f(x2)))

if f(x0) \* f(x2) < 0:

x1 = x2

else:

x0 = x2

step = step +1

condition = abs(f(x2)) > e

print ('root is: %0.8f '%x2)

# return x2

def f(x):

return x\*\*3-7\*x\*\*2+14\*x-6

x0 = float (input ('first guess: '))

x1 = float (input ('second guess: '))

e = float (input ('tolerance: '))

if f(x0) \* f(x1) > 0.0:

print ('given guess values do not bracket the root')

else:

root = bisection (x0, x1, e)

Output:

first guess: 0

second guess: 1

tolerance: 0.00001

iteration 1, x2 = 0.500000 and f(x2) = -0.625000

iteration 2, x2 = 0.750000 and f(x2) = 0.984375

iteration 3, x2 = 0.625000 and f(x2) = 0.259766

iteration 4, x2 = 0.562500 and f(x2) = -0.161865

iteration 5, x2 = 0.593750 and f(x2) = 0.054047

iteration 6, x2 = 0.578125 and f(x2) = -0.052624

iteration 7, x2 = 0.585938 and f(x2) = 0.001031

iteration 8, x2 = 0.582031 and f(x2) = -0.025716

iteration 9, x2 = 0.583984 and f(x2) = -0.012322

iteration 10, x2 = 0.584961 and f(x2) = -0.005640

iteration 11, x2 = 0.585449 and f(x2) = -0.002303

iteration 12, x2 = 0.585693 and f(x2) = -0.000636

iteration 13, x2 = 0.585815 and f(x2) = 0.000198

iteration 14, x2 = 0.585754 and f(x2) = -0.000219

iteration 15, x2 = 0.585785 and f(x2) = -0.000010

iteration 16, x2 = 0.585800 and f(x2) = 0.000094

iteration 17, x2 = 0.585793 and f(x2) = 0.000042

iteration 18, x2 = 0.585789 and f(x2) = 0.000016

iteration 19, x2 = 0.585787 and f(x2) = 0.000003

root is :0.58578682

**False position method:**

Introduction:

False position method is also called regula falsi method is another technique of solving equation in one variable. In this method we start finding the root by taking two initial guesses.

**Derivation:**

Where,

1st initial guess.

2nd initial guess.

.

Is the required formula for calculating the root value in false position method.

Algorithm:

1. Choose two points as an initial guess. Consider these initial guess as and .
2. Now find the root value () as
3. Then check the following
4. If f ()f()<0 then the root is in between . Thus .
5. If then the root value is in between .
6. If then the root value is .
7. Now find the absolute relative approximation error after every iteration as

where

.

.

1. Compare the absolute relative approximate error with the given tolerance .

* If then repeat the algorithm from 2nd step.
* If then stop the algorithm.

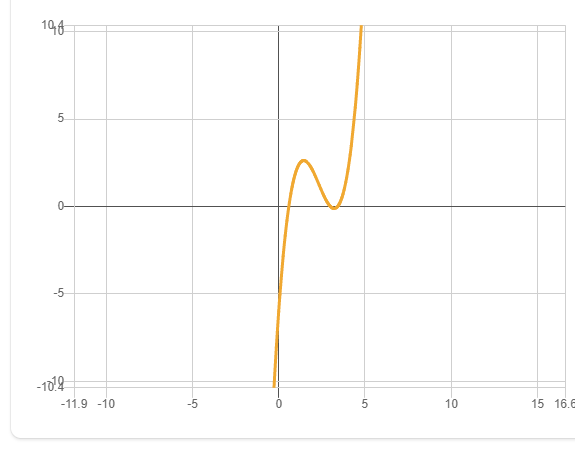
Example:

Consider a function

where

As, so according to theorem there exist at least one root between 0 and 1.

Graph:



Iteration 1:

Putting values

Since, so

Iteration 2:

Since,

Absolute approximation error:

Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| iteration | 1st initial guess =a | 2nd initial guess =b | F(a) | F(b) | D=(bf(a)-af(b))/(f(a)-f(b)) | F(d) | Error |
| 1. | 0 | 1 | -6 | 2 | 0.75 | 0.9843 | - |
| 2. | 0 | 0.75 | -6 | 0.9843 | 0.6443 | 0.3819 | 16.405% |
| 3. | 0 | 0.6443 | -6 | 0.3819 | 0.6057 | 0.1343 | 6.3727% |
| 4. | 0 | 0.6057 | -6 | 0.1343 | 0.5925 | 0.0455 | 2.2278% |
| 5. | 0 | 0.5925 | -6 | 0.0455 | 0.5880 | 0.0153 | 0.7594% |
| 6. | 0 | 0.5880 | -6 | 0.0153 | 0.5865 | 0.0048 | 0.2557% |
| 7. | 0 | 0.5865 | -6 | 0.0048 | 0.5860 | 0.0016 | 0.0853% |
| 8. | 0 | 0.5860 | -6 | 0.0016 | 0.5858 | 0.0009 | 0.0341% |
| 9. | 0 | 0.5858 | -6 | 0.0009 | 0.5858 | 0.0009 | 0.0000% |

So, the root value

Code:

from math import sin

def reg\_falsi (f, x1, x2, tol=1.0e-6, maxfpos=100):

if f(x1) \* f(x2) <0:

for fpos in range (1, maxfpos+1):

xh = x2 - (x2-x1)/(f(x2)-f(x1)) \* f(x2)

if abs(f(xh)) < tol:

break

elif f(x1) \* f(xh) < 0:

x2 = xh

else:

x1 = xh

else:

print ('No roots exists within the given interval')

return xh, fpos

y = lambda x: x\*\*3-7\*x\*\*2+14\*x-6

x1 = float (input ('enter x1: '))

x2 = float (input ('enter x2: '))

r, n = reg\_falsi (y, x1, x2)

print ('The root = %f at %d false position'% (r, n))

Output:

enter x1: 0

enter x2: 1

The root = 0.585787 at 14 false position

**Newton Raphson method:**

Introduction:

Newton Raphson method is one of the techniques of solving polynomial equation. It is quadratically convergent when get approach to the root value.

Algorithm:

1. Find the first derivative of the function.
2. Consider the initial guess and find the new estimate value as
3. Now find the absolute relative approximation error after every iteration as

where

.

.

1. Compare the absolute relative approximate error with the given tolerance .

* If then repeat the algorithm from 2nd step.
* If then stop the algorithm.

Advantages:

* This method is converging fast to the root value.
* At the start only one initial guess is required.

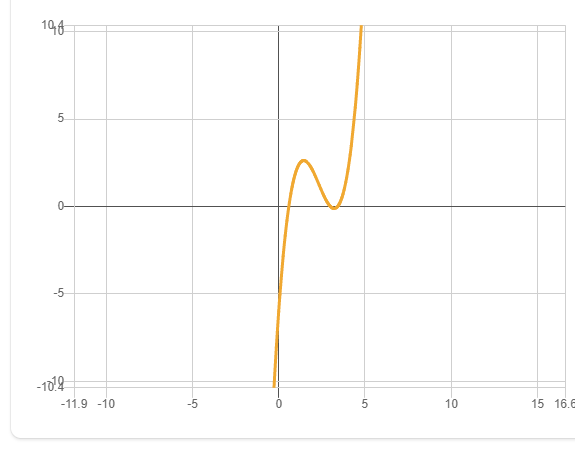
Disadvantages:

* If the initial guess is selected near the inflection point of the function then the function starts diverges away from the root.
* The initial value on which the value of the derivative of the function equal to zero are not allowed because division by zero is undefined.
* If initial guess is chosen near the root value then sometimes it may jump to another root value.

Example:

Consider a function

Graph:



Iteration 1:

Let

By Newton Raphson method,

Here,

Absolute approximation error:

Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| iteration |  | A= | B= | D= | Error |
| 1. | 5 | 14 | 19 | 4.2631 | 17.2855% |
| 2. | 4.2631 | 3.9429 | 8.8386 | 3.8169 | 11.6901% |
| 3. | 3.8169 | 1.0629 | 4.2695 | 3.5679 | 6.9789% |
| 4. | 3.5679 | 0.2603 | 2.2391 | 3.4516 | 3.3694% |
| 5. | 3.4516 | 0.0483 | 1.4182 | 3.4174 | 1.0007% |
| 6. | 3.4174 | 0.0037 | 1.1922 | 3.4142 | 0.0937% |
| 7. | 3.4142 | -0.0015 | 1.1715 | 3.4142 | 0.0000% |

So, the root value

Code:

from math import sin

def newton (fn, dfn, x, tol, maxiter):

for i in range(maxiter):

xnew = x - fn(x)/dfn(x)

if abs(xnew-x) <tol:

break

x = xnew

return xnew, i

y = lambda x: x\*\*3-7\*x\*\*2+14\*x-6

dy = lambda x: 3\*x\*\*2-14\*x+14

x, n = newton (y, dy, 5, 0.0001, 100)

print ('the root is %.3f at %d iterations.'% (x, n))

Output:

the root is 3.414 at 6 iterations.

**Secant method:**

Introduction:

The secant method is a technique in which a series of roots of secant lines are used to find the root values of functions.

Derivation:

Algorithm:

1. Consider two initial guesses as and
2. Now find the estimate value as
3. Now find the absolute relative approximation error after every iteration as

where

.

.

1. Compare the absolute relative approximate error with the given tolerance .

* If then repeat the algorithm from 2nd step.
* If then stop the algorithm.

Advantages:

* This method is converging fast to the root value.
* Without bracket the root it requires two initial guesses.

Disadvantages:

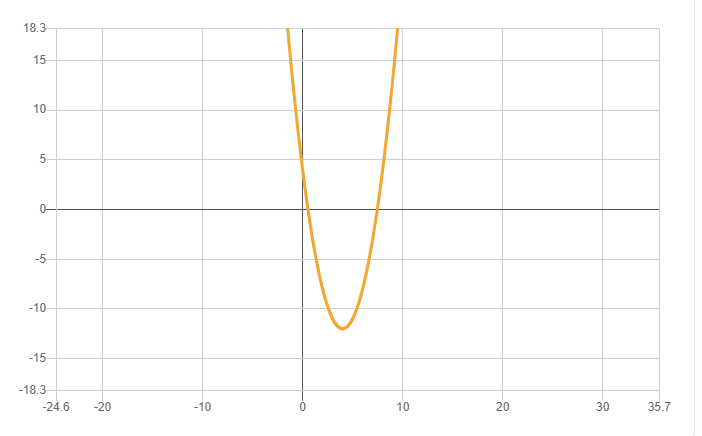
* As division by zero is undefined so in this method if we such value in which subtraction of function on these values is equal to zero then it gives error.
* If initial guess is chosen near the root value then sometimes it may jump to another root value.

Example:

Consider a function

Let the initial guesses are

Graph:



Iteration 1:

In secant method root value is given as

Here,

Then

Absolute approximation error:

Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| iteration | 1st initial guess= | 2nd initial guess= |  |  |  |  | Error |
| 1. | 0 | 1 | 4 | -3 | 0.5714 | -0.2447 | 75% |
| 2. | 0 | 0.5714 | 4 | -0.2447 | 0.5385 | -0.0177 | 6.109% |
| 3. | 0 | 0.5385 | 4 | -0.0177 | 0.5361 | -0.0013 | 0.447% |
| 4. | 0 | 0.5361 | 4 | -0.0013 | 0.5359 | -0.0001 | 0.037% |
| 5. | 0 | 0.5359 | 4 | -0.0001 | 0.5359 | -0.0001 | 0.000% |

So, the root value is 0.5359.

Code:

from math import sin

def secant (fn, x1, x2, tol, maxiter):

for i in range(maxiter):

xnew = x2 - (x2-x1)/(fn(x2)-fn(x1))\*fn(x2)

if abs(xnew-x2) < tol:

break

else:

x1 = x2

x2 = xnew

else:

print ('warning: Maximum number of iterations is reached')

return xnew, i

y = lambda x: x\*\*2-8\*x+4

x1 = float (input ('enter x1: '))

x2 = float (input ('enter x2: '))

r, n = secant (y, x1, x2, 1.0e-6, 100)

print ('Root = %f at %d iterations'% (r, n))

Output:

enter x1: 0

enter x2: 1

Root = 0.535898 at 4 iterations

**Fix point iteration method:**

Introduction:

A fixed point for a function is a number at which the function does change its value when the function is applied. Here the number p is considered as the fixed point of the function g.

Convergence:

The convergence of fixed point is based on existence and uniqueness theorem. According to this theorem:

* If g ∈ C [a, b] and g(x) ∈ [a, b] for all x ∈ [a, b], then g has at least one fixed point in [a, b].
* If, in addition, g’(x) exists on (a, b) and a positive constant k < 1 exists with |g (x)| ≤ k, for all x ∈ (a, b), then there is exactly one fixed point in [a, b].

Then, for any number in [a, b], the sequence defined by

Converges to the unique fixed-point p in [a, b].

Algorithm:

* choose some function and make it in the form of such that is continuous.
* Now take some initial guess as value of x.
* Use x in the function .
* Calculate the error after every iteration as

Where,

.

.

* Compare the absolute relative approximate error with the given tolerance .
* If then repeat the algorithm from 3rd step.
* If then stop the algorithm.

Advantages:

* Fixed point iteration method is easier to solve.
* It is a good root finding strategy.
* Converges fast.

Example:

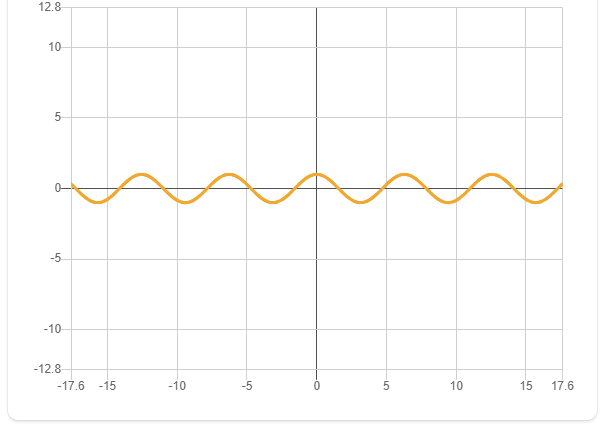
Consider the function

After splitting in the form

Since is continuous

Take the initial guess as

Graph:



Iteration 1:

Iteration 2:

Absolute approximation error:

Table:

|  |  |  |  |
| --- | --- | --- | --- |
| iteration | A | F (a) | error |
| 1. | 0.25 | 1 | - |
| 2. | 1 | 0.9998 | 75% |
| 3. | 0.9998 | 0.9998 | 0.02% |
| 4. | 0.9998 | 0.9998 | 0% |

***Direct methods for solving linear system***

**Gauss elimination method:**

There are two method use to solve gauss elimination method that are list below;

1. Pivot elimination method
2. Matrix transformation

*Pivot elimination method:*

In this method first select the equation as a pivot equation which have largest coefficient. By applying several operations on the pivot equation find the values of the unknown variables.

Example:

Consider the system of equation

In this system the pivot equation is 1st equation for x because it has the largest x coefficient which is

Multiplying equation (1) by

Subtracting equation (2) by (4)

Multiplying equation (1) by

Subtracting equation (3) by (6)

Now from the equation (1), (5) and (7) the system of equation becomes

In this system the pivot equation is 5th equation for y because it has the largest y coefficient which is

Multiplying equation (5) by

Subtracting equation (7) by (8)

Putting the value of z in equation (5) we get

By putting the value of y and z in equation (1) we get

Solution set

*Matrix transformation:*

In this elimination method matrix is transform into upper triangular matrix to find the value of unknown variables.

Example:

Let us consider a linear system of equation

First convert it in the form of Ax=b

Where , and

So, the matrix that is used to form upper triangular matrix

First Interchange row1 and row3

Then

Then

Then

Hence,

Solution set

These results show that solving the same system of linear equation using both types of elimination method gives the same answers.

***Iterative techniques in matrix algebra:***

**Norm of a vector:**

A function on defined on mapping of is said to be vector norm if it satisfies the following properties;

There are some common norms which are given below:

* The norm is given by
* The norm or Euclidean norm is given by
* The norm or max norm is given by

**Matrix norm:**

A matrix norm on the set of all matrices is a real valued function defined on the set A and B such that

* is a null set.

Theorem:

If is a vector norm on then

Is a matrix norm, natural norm or induced matrix norm. This describes a stretchiness of the matrix.

**Eigenvalue:**

The eigenvalue of a matrix A is defined as

Where,

is the identity matrix

is eigenvalue of the matrix A.

**Eigenvector:**

The eigenvector corresponding to that eigenvalue of the matrix A which satisfies that for every .

**Spectral radius:**

The spectral radius of matrix A is denoted by and is defined as

Where, is the eigenvalue of the matrix A.

Theorem:

If A is an matrix then

* for any natural norm

**Convergence:**

Convergences of n-dimensional vectors are in is said to be converges to with respect to the norm if for any there exist any integer such that

A matrix is convergent when

Where

***Iterative Methods:***

The method that have discussed are not suitable for solving large number of systems of linear equations. Iterative method is very useful for a computer storage and also manages the time.

Some of the advantages of iterative system are:

* Less calculation is required to solve a large matrix in less time.
* As compared to elimination method the approximation error is less.
* These methods work on the self-correction technique if an error is made.
* In computer these methods use less memory.
* These are quicker and easier to use in case of large matrix.

There are two iterative methods which are as follow;

1. Jacobi method
2. Gauss-Seidel method

**Gauss Jacobi method:**

Jacobi method is one method to solve linear equation of the system. The system written in form of matrix as Ax=b must have a unique solution and have a non-zero diagonal entries. This method only possible when the is strictly dominant which means the magnitude of the diagonal entries must be greater than the sum of other entries of that row. If the matrix is not dominant then make it dominant by interchanging the rows.

Algorithm:

* Write system of equation in the form of Ax=b as
* We have to find the value of as

Similarly,

* Find the error of approximation as

Example:

Consider the system of linear equation

The matrix is strictly dominant because magnitude of all the diagonal entries are greater than the sum of all other entries of the corresponding rows.

Then,

Initially at

Iteration 1:

Iteration 2:

Error:

Table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | K=1 | K=2 | K=3 | K=4 | K=5 | K=6 |
|  | 0 | 1.3333 | 0.641 | 1.1316 | 0.8739 | 0.9946 |
|  | 0 | 1.2 | 0.505 | 0.7686 | 0.6254 | 0.7723 |
|  | 0 | 0.875 | 0.1 | 0.6096 | 0.3909 | 0.3909 |

**Gauss-Seidel method:**

This method is similar to the Jacobi method but the only difference is that it has use the updated values to find the other value for example,

Similarly,

As in the first equation value of find by using previous value of other variable in the equation and in 2nd equation value of is used which we find in first equation. Similarly, is the case for last equation where all the new value of the variable is used.

Example:

Consider the system of linear equation

The matrix is strictly dominant because magnitude of all the diagonal entries are greater than the sum of all other entries of the corresponding rows.

Then,

Iteration 1:

Initially at

Iteration 2:

Error:

Table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | K=1 | K=2 | K=3 | K=4 | K=5 | K=6 |
|  | 0.0000 | 1.3333 | 1.0417 | 0.9865 | 0.9786 | 0.9778 |
|  | 0.0000 | 0.6667 | 0.7416 | 0.7456 | 0.7448 | 0.7445 |
|  | 0.0000 | 0.2083 | 0.2989 | 0.3186 | 0.3218 | 0.3222 |

**Successive over relaxation method:**

Introduction:

Successive over relaxation is another method to solve the iterative problems. In short it is called as SOR method. There is one parameter as (w) is used which is equal to

Where, is the spectral radius which is given as

There are some conditions for w:

* If this method is converted into Gauss-Seidel method.
* If it is known as Over relaxation method.
* If it known as Under relaxation method.
* The domain of w lies between the interval . Only in this interval this method is converges otherwise diverges.

General formula for SOR method is given as

Example:

Consider the system of linear equation with w=1.45

The matrix is strictly dominant because magnitude of all the diagonal entries are greater than the sum of all other entries of the corresponding rows.

Then,

The matrix form Ax=b is

Iteration 1:

Table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | K=1 | K=2 | K=3 | K=4 | K=5 | K=6 |
|  | 1.6667 | 0.9289 | 1.1770 | 1.2181 | 0.9808 | 0.9614 |
|  | 0.6665 | 0.2489 | 0.7570 | 0.6483 | 0.7808 | 0.7433 |
|  | 0.1042 | 0.3689 | 0.2132 | 0.2668 | 0.3233 | 0.3299 |

***Interpolation and Polynomial Approximation***

**Newton backward difference method:**

Let are the given points. Using technique of newton backward difference, it is solved through table as

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Let be the intervals then

Where,

is the distance between two consecutive points

is the backward difference operator

for any point backward difference of the function is given as

Example:

Consider the points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Find by using backward difference method.

Solution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

As 1.5 is between the first interval so,

**Newton forward difference method:**

Let are the given points. Using technique of newton forward difference, it is solved through table as

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Let be the intervals then

Where,

is the distance between two consecutive points

is the forward difference operator

For any point forward difference of the function is given as

Example:

Consider the points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Find by using forward difference method

Solution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Newton central difference method:**

Let are the given points. Using technique of newton central difference, it is solved through table as

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Let be the intervals then

Where,

is the distance between two consecutive points

is the average operator which is given as

For any point forward difference of the function is given as

Example:

Consider the points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Find by using central difference method

Solution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Code:

import numpy as np

f=lambda x:0.1\*x\*\*5-0.2\*x\*\*3+0.1\*x-0.2

x=0.1

h=0.1

df1 = 0.09405

df2 = -0.118

print ("\t f'(x)\t\t err\t\t f''(x)\t\t err")

#forward difference

dff1 = (f(x+h)- f(x))/h

dff2 = (f(x+2\*h) - 2\*f(x+h) + f(x))/h\*\*2

print ("FFD\t% f\t% f\t% f\t% f"% (dff1, dff1-df1, dff2, dff2-df2))

#backward difference

dff1 = (f(x+h)- f(x+h))/h

dff2 = (f(x) - 2\*f(x+h) + f(x-2\*h))/h\*\*2

print ("BFD\t% f\t% f\t% f\t% f"% (dff1, dff1-df1, dff2, dff2-df2))

#central difference

dff1 = (f(x+h)- f(x+h))/2\*h

dff2 = (f(x+h) - 2\*f(x) + f(x-h))/h\*\*2

print ("FFD\t% f\t% f\t% f\t% f"% (dff1, dff1-df1, dff2, dff2-df2))

Output:

f'(x) err f''(x) err

FFD 0.086310 -0.007740 -0.222000 -0.104000

BFD 0.000000 -0.094050 -3.686400 -3.568400

FFD 0.000000 -0.094050 -0.117000 0.001000

**Shift operator:**

For forward difference

For backward difference

**Average operator:**

**Lagrange interpolation:**

Where,

Similarly,

Example:

Use Lagrange interpolation to find the polynomial

Calculate the value of ?

Solution:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

**Newton divided difference interpolation formula:**

Introduction:

Newton divided difference interpolation formula is a technique of interpolation in which the intervals are unequally space means difference between the intervals is not same. This method is used to find a polynomial function. Here is the formula to calculate the value of function.

Let be the intervals which are unequally space to each other. Newton divided difference to find the value of polynomial.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x |  | 1st order DD | 2nd order DD | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  |  | |
| 3rd order DD | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |

Example:

Consider the points and find the 3rd order polynomial function.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Solution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 1st order | 2nd order | 3rd order |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

***Spline interpolation:***

An interpolation in which special type of piecewise polynomial are solve.

Spline interpolation is further divided into three methods which are as follow;

* Linear spline interpolation
* Quadratic spline interpolation
* Cubic spline interpolation

*Linear spline interpolation:*

The linear spline interpolation is used to represent set of line segment between two data points.

Let be the data points. We should keep in value of x is in proper order either it is in ascending order or in descending order. If the given data is not in the particular order then first arrange in order such that or in the other way it is represented as .

In linear spline interpolation linear equations are solve between different intervals of the data point. Using this method equation are form as

In this method the number of intervals is n-1 where n is the total number of the data point and the number of equations formed from the given intervals is same as total number of unknown variables found which is equal to 2n.

Standard formula if linear spline interpolation is given as;

Where,

For the above equation become;

Example:

Consider the data points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Then find the value of

Solution:

Total number of data points = 4

Total number of intervals = n-1 = 3

Total number of unknown = 6

Total number of equations = 6

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Intervals:

Equations:

In matrix form it is written as

Solving these equations, we get values as

As we have to find which lies between the interval so first we find the function for as

Where,

For this equation becomes

*Quadratic spline interpolation:*

In quadratic spline interpolation quadratic function are solve between different intervals of the data point. Using this method equation are form as

In this method the number of intervals is n-1 where n is the total number of the data point and the number of equations formed from the given intervals is same as total number of unknown variables found which is equal to 3n.

Algorithm:

* Find all the equation on the given data point. Each equation is made by using two consecutive data points as

Here the number of equations is 2n

* As at the interior point the first derivative of the function is zero so

Which gives equations like

This gives n-1 equations.

* Now the total number of equations are 2n+n-1=3n-1
* If then.

This makes the first equation as linear. So, the number of equations is 3n.

Example:

Consider the data points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Calculate the value of ?

Solution:

Total number of data points = 4

Number of intervals = 4-1 = 3

Total number of equations = 9

Total number of unknown variables = 9

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Intervals:

Equations:

As so,

in the first equation then the equation becomes

Now writing the above equations in the form of Ax=b

By solving this matrix value of unknown variables are;

*Cubic spline interpolation:*

It is most common piecewise polynomial approximation in which cubic equations are formed between intervals. The equation in cubic spline has both first and second derivative is the continuous function.

Algorithm:

* Find all the equation on the given data point. Each equation is made by using two consecutive data points as

Here the number of equations is 2n

* As at the interior point the first and second derivative of the function is zero so

First derivative:

Which gives equations like

Second derivative:

Which gives equations like

This gives 2n-2 equations.

* Now the total number of equations are 2n+2n-2=4n-2
* If then

This makes the first equation is quadratic and second equation is linear. So, the number of equations is 4n.

Example:

Consider the data points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Calculate the value of using cubic spline interpolation?

Solution:

Total number of data points = 4

Number of intervals = 4-1 = 3

Total number of equations = 12

Total number of unknown variables = 12

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Intervals:

Equations:

As so,

First equation becomes

Second equation is

Now writing the above equations in the form of Ax=b

By solving the matrix values of unknown variables are:

Code:

#def cubicinterpolation1():

x\_pts = np. linspace(0,2\*np.pi,10) # 10 equidistant coords from 0 to 10

y\_pts = np. sin(x\_pts)

x\_vals = np. linspace(0,2\*np.pi,50) # 50 desired parts

f = interpolate. interp1d (x\_pts, y\_pts,'cubic')

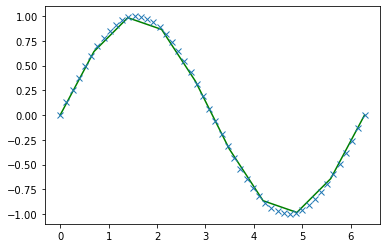
y\_vals = f(x\_vals) # cubic interpolation

plt. Plot (x\_pts, y\_pts, 'g') # plot known data points

plt. Plot (x\_vals, y\_vals, 'x') # plot interpolated points

plt. show ()

Output:



***Numerical Integration:***

Integration is basically area under the curve bounded between the interval (a, b). it is given by formula as

Numerical integration is divided into further rule which are given below:

* Trapezoidal rule
* Simpson’s rule
* Simpson’s rule

Trapezoidal rule:

Trapezoidal rule works in the linear state. The general formula is given as:

Where, h= distance between two consecutive points

Generalized error:

Example:

Consider the function and find the value of the function between the points and taking

Solution:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Generalized error:

Simpson’s rule:

Simpson’s rule works in quadratic state. The general formula of this method is:

Error:

Generalized error:

Example:

Consider the function and find the value of the function between the points and taking

Solution:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Simpson’srule:

Simpson’s rule works in quadratic state whose error is in cubic state. The general formula of this method is:

Error:

Generalized error:

Example:

Consider the function and find the value of the function between the points and taking

Solution:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Generalized error:

**Reference:**

* A first course in numerical analysis by prof Saeed Akhtar Bhatti and Mr. Naeem Akhtar Bhatti 5th edition.
* Numerical analysis by Richard L Burden and J. Douglas Faires 9th edition.
* <https://byjus.com/maths>.