Chapter 46: Equation Solving

Section 46.1: Linear Equation

There are two classes of methods for solving Linear Equations:

1. **Direct Methods**: Common characteristics of direct methods are that they transform the original equation into equivalent equations that can be solved more easily, means we get solve directly from an equation.
2. **Iterative Method**: Iterative or Indirect Methods, start with a guess of the solution and then repeatedly refine the solution until a certain convergence criterion is reached. Iterative methods are generally less efficient than direct methods because large number of operations required. Example- Jacobi's Iteration Method, Gauss-Seidal Iteration Method.

Implementation in C-

```
//Implementation of Jacobi's Method
void JacobisMethod(int n, double x[n], double b[n], double a[n][n]){
    double Nx[n]; //modified form of variables
    int rootFound=0; //flag

    int i, j;
    while(!rootFound){
        for(i=0; i<n; i++){              //calculation
            Nx[i]=b[i];

            for(j=0; j<n; j++){
                if(i!=j) Nx[i] = Nx[i]-a[i][j]*x[j];
            }
            Nx[i] = Nx[i] / a[i][i];
        }

        rootFound=1;                     //verification
        for(i=0; i<n; i++){
            if(!( (Nx[i]-x[i])/x[i] > -0.000001 && (Nx[i]-x[i])/x[i] < 0.000001 )){
                rootFound=0;
                break;
            }
        }

        for(i=0; i<n; i++){              //evaluation
            x[i]=Nx[i];
        }
    }

    return ;
}

//Implementation of Gauss-Seidal Method
void GaussSeidalMethod(int n, double x[n], double b[n], double a[n][n]){
    double Nx[n]; //modified form of variables
    int rootFound=0; //flag

    int i, j;
    for(i=0; i<n; i++){                  //initialization
        Nx[i]=x[i];
    }
```

```
    while(!rootFound){
        for(i=0; i<n; i++){              //calculation
            Nx[i]=b[i];

            for(j=0; j<n; j++){
                if(i!=j) Nx[i] = Nx[i]-a[i][j]*Nx[j];
            }
            Nx[i] = Nx[i] / a[i][i];
        }

        rootFound=1;                     //verification
        for(i=0; i<n; i++){
            if(!( (Nx[i]-x[i])/x[i] > -0.000001 && (Nx[i]-x[i])/x[i] < 0.000001 )){
                rootFound=0;
                break;
            }
        }

        for(i=0; i<n; i++){              //evaluation
            x[i]=Nx[i];
        }
    }

    return ;
}

//Print array with comma separation
void print(int n, double x[n]){
    int i;
    for(i=0; i<n; i++){
        printf("%lf, ", x[i]);
    }
    printf("\n\n");

    return ;
}

int main(){
    //equation initialization
    int n=3;    //number of variables

    double x[n];    //variables

    double b[n],    //constants
        a[n][n];    //arguments

    //assign values
    a[0][0]=8; a[0][1]=2; a[0][2]=-2; b[0]=8;    //8x₁+2x₂-2x₃+8=0
    a[1][0]=1; a[1][1]=-8; a[1][2]=3; b[1]=-4;   //x₁-8x₂+3x₃-4=0
    a[2][0]=2; a[2][1]=1; a[2][2]=9; b[2]=12;    //2x₁+x₂+9x₃+12=0


    int i;

    for(i=0; i<n; i++){                      //initialization
        x[i]=0;
    }
    JacobisMethod(n, x, b, a);
    print(n, x);


    for(i=0; i<n; i++){                      //initialization
```

```
        x[i]=0;
    }
    GaussSeidalMethod(n, x, b, a);
    print(n, x);

    return 0;
}
```

Section 46.2: Non-Linear Equation

An equation of the type `f(x)=0` is either algebraic or transcendental. These types of equations can be solved by using two types of methods-

1. **Direct Method**: This method gives the exact value of all the roots directly in a finite number of steps.
2. **Indirect or Iterative Method**: Iterative methods are best suited for computer programs to solve an equation. It is based on the concept of successive approximation. In Iterative Method there are two ways to solve an equation-
    - **Bracketing Method**: We take two initial points where the root lies in between them. Example- Bisection Method, False Position Method.
    - **Open End Method**: We take one or two initial values where the root may be any-where. Example- Newton-Raphson Method, Successive Approximation Method, Secant Method.

Implementation in C:

```
/// Here define different functions to work with
#define f(x) ( ((x)*(x)*(x)) - (x) - 2 )
#define f2(x) ( (3*(x)*(x)) - 1 )
#define g(x) ( cbrt( (x) + 2 ) )


/**
* Takes two initial values and shortens the distance by both side.
**/
double BisectionMethod(){
    double root=0;

    double a=1, b=2;
    double c=0;

    int loopCounter=0;
    if(f(a)*f(b) < 0){
        while(1){
            loopCounter++;
            c=(a+b)/2;

            if(f(c)<0.00001 && f(c)>-0.00001){
                root=c;
                break;
            }

            if((f(a))*(f(c)) < 0){
                b=c;
            }else{
                a=c;
            }
```

```
        }
    }
    printf("It took %d loops.\n", loopCounter);

    return root;
}

/**
* Takes two initial values and shortens the distance by single side.
**/
double FalsePosition(){
    double root=0;

    double a=1, b=2;
    double c=0;

    int loopCounter=0;
    if(f(a)*f(b) < 0){
        while(1){
            loopCounter++;

            c=(a*f(b) - b*f(a)) / (f(b) - f(a));

            /*/printf("%lf\t %lf \n", c, f(c));/**////test
            if(f(c)<0.00001 && f(c)>-0.00001){
                root=c;
                break;
            }

            if((f(a))*(f(c)) < 0){
                b=c;
            }else{
                a=c;
            }
        }
    }
    printf("It took %d loops.\n", loopCounter);

    return root;
}

/**
* Uses one initial value and gradually takes that value near to the real one.
**/
double NewtonRaphson(){
    double root=0;

    double x1=1;
    double x2=0;

    int loopCounter=0;
    while(1){
        loopCounter++;

        x2 = x1 - (f(x1)/f2(x1));
        /*/printf("%lf \t %lf \n", x2, f(x2));/**////test

        if(f(x2)<0.00001 && f(x2)>-0.00001){
            root=x2;
            break;
        }
```

```
        x1=x2;
    }
    printf("It took %d loops.\n", loopCounter);

    return root;
}

/**
* Uses one initial value and gradually takes that value near to the real one.
**/
double FixedPoint(){
    double root=0;
    double x=1;

    int loopCounter=0;
    while(1){
        loopCounter++;

        if( (x-g(x)) <0.00001 && (x-g(x)) >-0.00001){
            root = x;
            break;
        }

        /*/printf("%lf \t %lf \n", g(x), x-(g(x)));/**////test

        x=g(x);
    }
    printf("It took %d loops.\n", loopCounter);

    return root;
}

/**
* uses two initial values & both value approaches to the root.
**/
double Secant(){
    double root=0;

    double x0=1;
    double x1=2;
    double x2=0;

    int loopCounter=0;
    while(1){
        loopCounter++;

        /*/printf("%lf \t %lf \t %lf \n", x0, x1, f(x1));/**////test

        if(f(x1)<0.00001 && f(x1)>-0.00001){
            root=x1;
            break;
        }

        x2 = ((x0*f(x1))-(x1*f(x0))) / (f(x1)-f(x0));

        x0=x1;
        x1=x2;
    }
    printf("It took %d loops.\n", loopCounter);

    return root;
}
```

```
int main(){
    double root;

    root = BisectionMethod();
    printf("Using Bisection Method the root is: %lf \n\n", root);

    root = FalsePosition();
    printf("Using False Position Method the root is: %lf \n\n", root);

    root = NewtonRaphson();
    printf("Using Newton-Raphson Method the root is: %lf \n\n", root);

    root = FixedPoint();
    printf("Using Fixed Point Method the root is: %lf \n\n", root);

    root = Secant();
    printf("Using Secant Method the root is: %lf \n\n", root);

    return 0;
}
```