

Introducción

El objetivo de este trabajo es aplicar los métodos numéricos vistos a lo largo del semestre, para esto se implementaron 10 métodos en el lenguaje JAVA y posteriormente se le encuentra a cada uno una aplicación en la carrera de ingeniería eléctrica electrónica. En este trabajo se detallan uno por uno los métodos implementados con el código fuente de la clase main y de cada una de las clases (se realizaron al menos una por método), posteriormente se detalla en qué se aplicará el método y en base a ello se ejecutará el programa.

Construcción del proyecto

El proyecto se compone de 63 archivos dentro de 32 carpetas, para realizar este proyecto utilicé el IDE NetBeans v.8.2, y el tamaño total del código es de 445KB, contiene en total 17 clases. Los métodos que se desarrollaron son:

1. Newton-Raphson.
2. Crout.
3. Gauss-Seidel.
4. Krylov.
5. Interpolación por polinomio de Lagrange.
6. Newton-Gregory.
7. Derivación con tercer orden de interpolación.
8. Integración por regla de Simpson $\frac{1}{3}$.
9. Runge-Kutta de cuarto orden.
10. Factores cuadráticos.

Se creó un menú en la clase main que permite ejecutar la clase dentro de una estructura switch case, los métodos se encuentran ordenados conforme a la lista.

Desarrollo

Newton-Raphson

Para la implementación del método se utilizaron tres clases llamadas NewtonRaphson.java, DerivacionPol.java y Evaluación.java

Código Fuente:

Newton Raphson.java

```
package proyecto.analisis.numerico;
```

```

import java.io.*;
import java.lang.*;
import java.util.*;

public class NewtonRaphson
{

    int grado;
    double a[];
    double p0;
    double e1;
    double e2;
    double TOL;
    int N;

    public void leePol()
    {
        System.out.println("Ingrese grado del polinomio: ");
        Scanner sc = new Scanner(System.in);
        grado= sc.nextInt();
        a = new double[grado+1];
        int i;
        for(i=0;i<=grado;i++)
        {
            System.out.println("Valor del coeficiente "+ i+": ");
            a[i] = sc.nextDouble();
        }
    }

    public void pideValoresdeEntrada()
    {
        System.out.println("Ingresa el valor aproximado de la función: ");
        Scanner sc = new Scanner(System.in);
        p0 = sc.nextDouble();
        System.out.println("Ingresa el valor de la tolerancia: ");
        Scanner sc2 = new Scanner(System.in);
        TOL = sc2.nextDouble();
        System.out.println("Ingresa el número de iteraciones que quieres que realice el método: ");
        Scanner sc3 = new Scanner(System.in);
        N = sc3.nextInt();
    }

    public void DerivadasFuncion()
    {
        double d1[];
        double d2[];
        int g;
        g=grado-1;
        DerivacionPol d= new DerivacionPol();
        d1= d.Derivo(grado,a);
        Evaluacion e = new Evaluacion();
        e1= e.Evaluacion(grado,p0,d1);
    }
}

```

```

        d2=d.Derivo(grado, a);
        e2= e.Evaluacion(g,p0,d2);

    }
    public double FuncionNewtonRaphson()
    {
        int i=1;
        while (i<=N)
        {
            double p;
            p= p0-e1/e2;
            if(Math.abs(p-p0)<TOL)
            {
                System.out.println("Proceso exitoso, el valor obtenido es: "+p);
                return p;
            }
            else
            {
                i=i+1;
                p0=p;
            }
            System.out.println("Error, el método ha fracasado después de "+N+" iteraciones");
        }
        return 0;
    }
}

```

DerivaciónPol.java:

```

package proyecto.analisis.numerico;
/*Creado por Teresa Fiel Muñoz*/
public class DerivacionPol {
    double b[];
    public double[] Derivo(int grado, double a[])
    {
        double b[];
        int j;
        int grad=grado;
        b= new double[grado+1];
        for(j=0;j<=grado;j++)
        {
            b[j]=a[j]*grad;
            grad=grad-1;
        }
        return b;
    }
}

```

Evaluacion.java:

```

package proyecto.analisis.numerico;

public class Evaluacion{
    public double Evaluacion(int grado, double p0, double b[])
    {
        double c=0.0;
        int k;
        for(k=0;k<=grado-1;k++)
        {
            c=b[k]*(p0)*Math.pow(p0,grado);
        }
        return c;
    }
}

```

Consideraciones

Cabe resaltar que este método se diseñó únicamente para recibir de datos de entrada de funciones polinómicas de grado n.

En este caso se utilizó para calcular la raíz de un polinomio sencillo de segundo grado, que puede representar algún modelo físico para la aplicación de la ingeniería. El polinomio que utilicé fue:

$$x^2 + 2x + 1$$

Las raíces del polinomio son: -1 y -1.

Ingresamos el polinomio al programa

```
Output - Proyecto Analisis Numerico (run)  Inspector

run:
---PROYECTO ANÁLISIS NUMÉRICO--

Menú:
1.Newton-Raphson.
2.Crout.
3.Gauss-Seidel.
4.Krylov.
5.Interpolación por polinomio de Lagrange.
6.Newton-Gregory en avance.
7.Derivación con tercer orden de interpolación.
8.Integración con Regla de Simpson 1/3.
9.Runge-Kutta de 4to orden.
10.Factores cuadráticos.
1
Ingrese grado del polinomio:
2
Valor del coeficiente 0:
1
Valor del coeficiente 1:
2
Valor del coeficiente 2:
1
Ingresa el valor aproximado de la función:
-1
Ingresa el valor de la tolerancia:
2
Ingresa el número de iteraciones que quieres que realice el método:
2
Proceso exitoso, el valor obtenido es: 0.5
BUILD SUCCESSFUL (total time: 11 seconds)
|
```

Lo que sucedió aquí fue que no hubo una aproximación muy exacta, ya que la raíz conseguida tiene alrededor de un 50% de error.

Crout

Para la implementación del método se hizo uso de una clase llamada Crout.java

Código Fuente:

```
package proyecto.analisis.numerico;
import java.util.Scanner;
import static proyecto.analisis.numerico.ProyectoAnalisisNumerico.N;

/*Creado por Teresa Elpidia Fiel Muñoz*/
public class Crout
{
```

```

double[][] L = new double[N][N];
double[][] U = new double[N][N];
double[][] A = new double[N][N];
double[] B = new double[N];
double[] Y = new double[N];
double[] X = new double[N];
int error;

public void LlenarMatrices()
{
    N=N-1;
    int i,j;
    for(i=0;i<=N;i++)
    {
        for(j=0;j<=N;j++)
        {
            L[i][j]=0;
        }
    }
    for(i=0;i<=N;i++)
    {
        for(j=0;j<=N;j++)
        {
            if (i==j)
                U[i][j]=1;
            else
                U[i][j]=0;
        }
    }
    for(i=0;i<=N;i++)
    {
        for(j=0;j<=N;j++)
        {
            System.out.println("Ingresa el valor de A"+i+j);
            Scanner mat = new Scanner(System.in);
            A[i][j]= mat.nextDouble();
        }
    }
    for(i=0;i<=N;i++)
    {
        System.out.println("Ingresa el valor de B"+i);
        Scanner mat = new Scanner(System.in);
        B[i]= mat.nextDouble();
    }
}

public double Crout()
{
    int j=1,i=1;
    int ks,j0=0;
    double w=0;

```

```

double[] q=new double[N];
double[] z=new double[N];
L[0][0]=A[0][0];
if (L[0][0]*U[0][0]==0)
{
    System.out.println("Factorización imposible.");
    return 0;
}
while(j<=N)
{
    U[0][j]=A[0][j]/L[0][0];
    L[j][0]=A[j][0];
    j=j+1;
}
while(i<=N-1)
{
    double[] r=new double[N+1];
    double[] t=new double[N+1];
    double[] p=new double[N+1];
    double[] v=new double[N+1];
    int k;
    int i0=i,i01=i;
    int s=0;
    for(i01=1;i01<=N-1;i01++)
    {
        for(k=0;k<=i0;k++)
        {
            r[s]=r[s]+ L[i][k]*U[k][i];
        }
        s=s+1;
    }
    L[i][i]=A[i][i]-r[i-1];
    if(U[i][i]*L[i][i]==0)
    {
        System.out.println("Factorización Imposible");
        return 0;
    }
    s=0;
    for(j=i+1;j<=N;j++)//i=2 j=3 k=1
    {
        for(k=0;k<=i-1;k++)
        {
            t[s]=t[s]+L[i][k]*U[k][j];
            v[s]=v[s]+L[j][k]*U[k][i];
        }
        double h,l;
        h=1/L[i][i];
        l=A[i][j]-t[i-1];
        U[i][j]=h*l;
        L[j][i]=A[j][i]-v[i-1];
        s=s+1;
    }
}

```

```

    }
    i=i+1;
}
for(ks=0;ks<=N-1;ks++)
{
    w=w+L[N][ks]*U[ks][N];
}
L[N][N]=A[N][N]-w;
if(L[N][N]*U[N][N]==0)
{
    System.out.println("Factorización Imposible");
    return 0;
}
Y[0]=B[0]/L[0][0];
for(i=1;i<=N;i++)
{
    for(j=0;j<=i-1;j++)
    {
        q[j0]=q[j0]+L[i][j]*Y[j];
    }
    double m,n;
    m=1/L[i][i];
    n=B[i]-q[i-1];
    Y[i]=m*n;
    j0=j0+1;
}
X[N]=Y[N]/U[N][N];
int y=0;
for(i=N-1;i>=0;i--)/i=2 j=3
{
    for(j=i+1;j<=N;j++)
    {
        z[y]=z[y]+U[i][j]*X[j];
    }
    double ñ;
    ñ=1/U[i][i];
    X[i]=ñ*Y[i]-z[y];
    y=y+1;
}
System.out.println("La solución es: ");
for(i=0;i<=N;i++)
{
    System.out.println("X["+i+"]="+X[i]);
}
return 1;
}
}

```

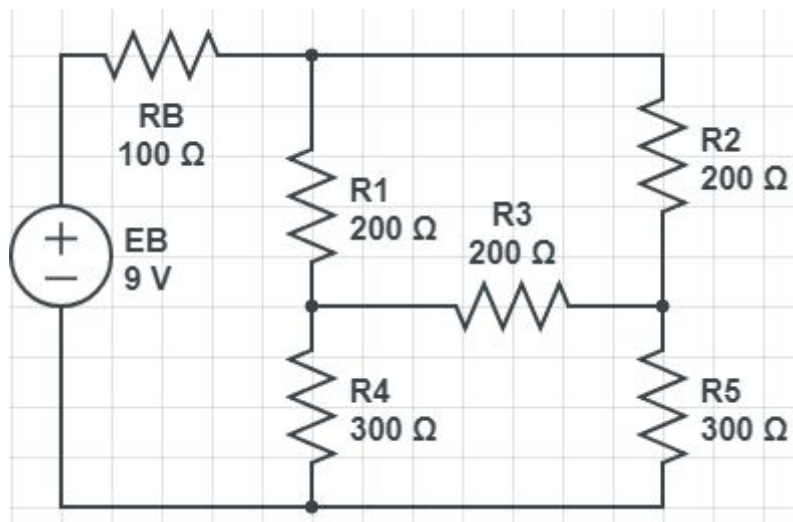
Consideraciones

El programa resuelve sistemas matriciales de cualquier orden, los valores de retorno son double para que no haya pérdida de información y también es necesario primero leer y guardar la matriz A y luego el vector B.

Aplicación

El uso en ingeniería que se utilizó abarca el campo de la ingeniería eléctrica, específicamente emplee el método para resolver un sistema de ecuaciones derivado del método de mallas. Este método se utiliza para calcular el valor de las intensidades en un circuito únicamente resistivo, donde se ubican las mallas y con una ecuación de equilibrio basándose en la ley de Kirchoff para voltajes se obtiene un sistema de ecuaciones de tercer grado.

Para el diagrama:



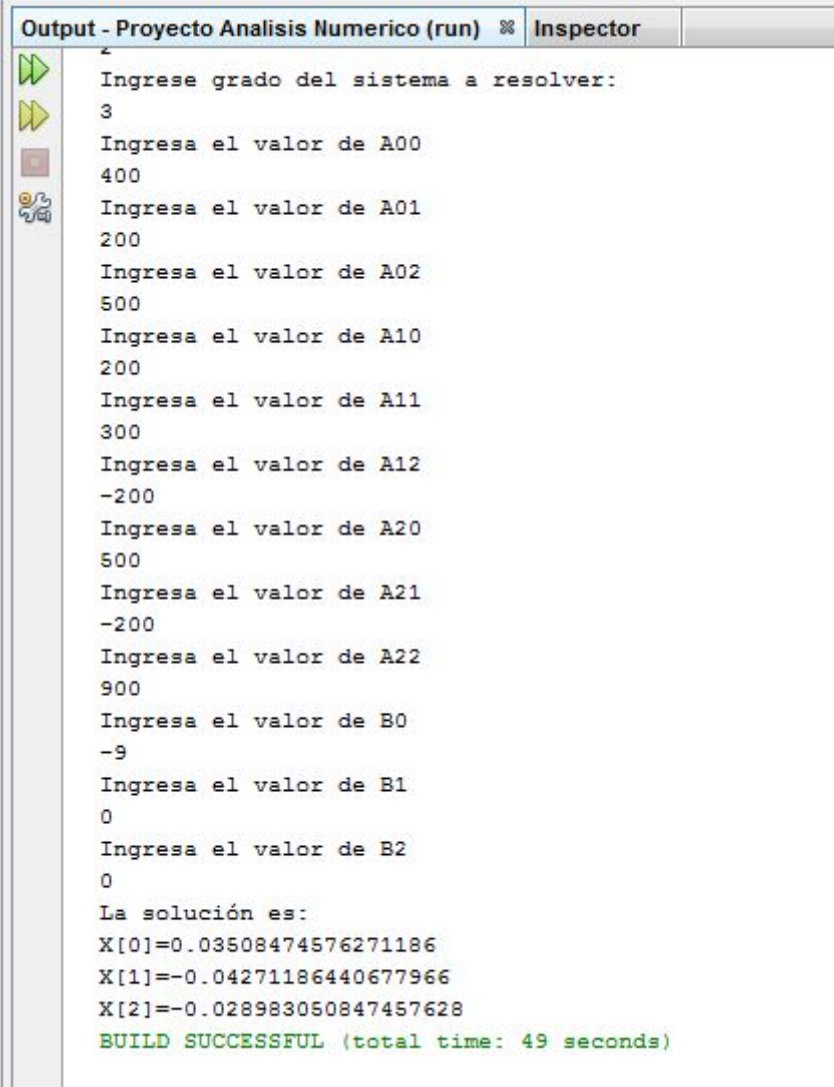
Se creó un sistema de ecuaciones para calcular la intensidad en la malla 1, 2 y 3.

$$\begin{aligned} 400I_1 + 200I_2 + 500I_3 &= -9 \\ 200I_1 + 300I_2 - 200I_3 &= 0 \\ 500I_1 - 200I_2 + 900I_3 &= -9 \end{aligned}$$

La solución al sistema de ecuaciones es de:

$$\begin{aligned} I_1 &= 35[mA] \\ I_2 &= -42[mA] \\ I_3 &= -28[mA] \end{aligned}$$

Al ingresar los datos de entrada, la solución al sistema es:



```
Output - Proyecto Analisis Numerico (run) % Inspector
Ingrese grado del sistema a resolver:
3
Ingresa el valor de A00
400
Ingresa el valor de A01
200
Ingresa el valor de A02
500
Ingresa el valor de A10
200
Ingresa el valor de A11
300
Ingresa el valor de A12
-200
Ingresa el valor de A20
500
Ingresa el valor de A21
-200
Ingresa el valor de A22
900
Ingresa el valor de B0
-9
Ingresa el valor de B1
0
Ingresa el valor de B2
0
La solución es:
X[0]=0.03508474576271186
X[1]=-0.04271186440677966
X[2]=-0.028983050847457628
BUILD SUCCESSFUL (total time: 49 seconds)
```

Gauss-Seidel

Para la implementación del método se utilizaron dos clases llamadas GaussSeidel.java y Modulo.java.

Código Fuente

GaussSeidel.java

```
package proyecto.analisis.numerico;
import java.util.Scanner;
import static proyecto.analisis.numerico.ProyectoAnalisisNumerico.N;

public class GaussSeidel
```

```

{
    double[][] A=new double[N][N];
    double[] B=new double[N];
    double imax;
    double tolerancia;
    int error;
    public void LlenarDatos()
    {
        int i,j;
        N=N-1;
        for(i=0;i<=N;i++)
        {
            for(j=0;j<=N;j++)
            {

                System.out.println("Ingresa el valor de A"+i+j);
                Scanner mat = new Scanner(System.in);
                A[i][j]= mat.nextDouble();

            }
        }
        for(i=0;i<=N;i++)
        {
            System.out.println("Ingresa el valor de B"+i);
            Scanner mat = new Scanner(System.in);
            B[i]= mat.nextDouble();
        }
        System.out.println("Ingresa el máximo número de iteraciones: ");
        Scanner Imax = new Scanner(System.in);
        imax= Imax.nextDouble();
        System.out.println("Ingresa el valor de la tolerancia: ");
        Scanner tol = new Scanner(System.in);
        tolerancia= tol.nextDouble();
    }
    public double[] GaussSeidel()
    {
        int c,i,j,k=0;
        double M=0,L=0,P;
        double[] Xk = new double[N+1];
        double[] Xkk=new double[N+1];
        N=N-1;
        for(c=0;c<=N;c++)
        {
            Xk[c]=0;
            Xkk[c]=0;
        }
        while(k<=imax)
        {
            for(i=0;i<=N;i++)
            {
                int i0=i;
                for(j=0;j<=i+1;j++)

```

```

        {
            M=M+A[i][j]*Xkk[j];
        }
        for(j=i0+1;j<=N;j++)
        {
            L=L+A[i][j]*Xkk[j];
        }
        double U,S;
        U=1/A[i][i];
        S=B[i]-M-L;
        Xkk[i]=U*S;
    }
    double[] mod=new double[N+1];
    for(k=0;k<=N;k++)
    {
        mod[k]=Xk[k]+Xkk[k];
    }
    Modulo Modulo = new Modulo();
    P=Modulo.Modulo(mod, N);
    System.out.println("El valor de p es: "+P);
    if(P<=tolerancia)
    {
        System.out.println("Procedimiento exitoso.");
        System.out.println("El resultado es: ");
        System.out.print("|");
        for(i=0;i<=N;i++)
        {
            System.out.print(Xkk[i]+",");
        }
        System.out.print("|");
        System.out.println("|5,10|");
        System.exit(0);
    }
    for(c=0;c<=N;c++)
    {
        Xk[c]=Xkk[c];
    }
    k=k+1;
}
System.out.println("Máximo número de iteraciones alcanzado");
System.out.println("El resultado aproximado es: ");
for(i=0;i<=N;i++)
{
    System.out.println(Xk[i]+",");
}
System.out.print("|");
return Xk;
}
}

```

Modulo.java

```
package proyecto.analisis.numerico;
```

```
/*Creado por Teresa Fiel*/
```

```
public class Modulo {  
  
    public int Modulo(double[] X,int N)  
    {  
        int i,res=0;  
        double[] R=new double[N+1];  
        for(i=0;i<=N;i++)  
        {  
            R[i]=X[i];  
        }  
        for(i=0;i<=N;i++)  
        {  
            R[i]=Math.pow(R[i],2);  
        }  
        for(i=0;i<=N-1;i++)  
        {  
            R[i+1]=R[i+1]+R[i];  
        }  
        R[N]=res;  
        return res;  
    }  
}
```

Consideraciones

Es necesario ingresar al programa la tolerancia y el número de iteraciones, los resuelve para cualquier grado y además debes ingresar primero la matriz A y luego la B.

Aplicación

La aplicación que le di al método de Gauss-Seidel es la obtención de α y β en un problema de probabilidad que involucra a la distribución Gamma. Para obtener probabilidades con esta distribución es necesario crear un sistema de ecuaciones partiendo del concepto de media y variancia.

La media de la distribución gamma es de 50 semanas, y la variancia es de 500 semanas. Con las fórmulas de:

$$\begin{aligned}\mu &= \alpha\beta \\ \sigma^2 &= \alpha\beta^2\end{aligned}$$

El sistema de ecuaciones queda:

$$0\alpha + 50\beta = 500$$

$$10\alpha + 0\beta = 50$$

El resultado es: $\alpha = 5$ y $\beta = 10$

Ingresamos los valores al programa

```

Output - Proyecto Analisis Numerico (run)
4.Krylov.
5.Interpolación por polinomio de Lagrange.
6.Newton-Gregory en avance.
7.Derivación con tercer orden de interpolación.
8.Integración con Regla de Simpson 1/3.
9.Runge-Kutta de 4to orden.
10.Factores cuadráticos.
3
Ingrese grado del sistema a resolver:
2
Ingresar el valor de A00
0
Ingresar el valor de A01
50
Ingresar el valor de A10
10
Ingresar el valor de A11
0
Ingresar el valor de B0
500
Ingresar el valor de B1
50
Ingresar el máximo número de iteraciones:
5
Ingresar el valor de la tolerancia:
.0001
El valor de p es: 0.0
Procedimiento exitoso.
El resultado es:
|5,10|
BUILD SUCCESSFUL (total time: 31 seconds)

```

Krylov

Para la implementación del método de Krylov utilicé cuatro clases distintas: Krylov.java, PotMatrices.java, GaussJordan.java y MultMatrices.java

Código Fuente

Krylov.java

```

package proyecto.analisis.numerico;
import java.util.Scanner;
import static proyecto.analisis.numerico.ProyectoAnalisisNumerico.N;

```

```

/**
 *

```

```

* @author Tere
*/
public class Krylov {

    double[][] A = new double[N][N];
    double[][] Ax= new double[N][N];
    double[] B = new double[N];
    double[][] Y= new double[N][1];
    int error;

    public void LLenarA()
    {
        int i,j;
        N=N-1;
        for(i=0;i<=N;i++)
        {
            for(j=0;j<=N;j++)
            {
                Ax[i][j]=0;
            }
        }
        for(i=0;i<=N;i++)
        {
            for(j=0;j<=N;j++)
            {
                System.out.println("Ingresa el valor de A"+i+j);
                Scanner mat = new Scanner(System.in);
                A[i][j]= mat.nextDouble();
            }
        }
        for(i=0;i<=N;i++)
        {
            B[i]=0;
            if(i==0)
            {
                Y[i][0]=1;
            }
            else
            {
                Y[i][0]=0;
            }
        }
    }

    public int Krylov()
    {
        int i,j;
        for(i=0;i<=N;i++)
        {
            for(j=0;j<=2;j++)
            {
                if(A[j][N-i]==0)
            }
        }
    }
}

```

```

        Ax[j][N-i]=Y[i][0];
    else
    {
        double[][] X=new double[N+1][N+1];
        double[][] S=new double[N+1][1];
        double[][] Z=new double[N+1][N+1];
        PotMatrices x = new PotMatrices();
        X=x.PMatrices(A,N,i);
        MultMatrices y = new MultMatrices();
        S=y.MMatrices(A,Y,N,N,1);
        MultMatrices z = new MultMatrices();
        Z=z.MMatrices(X, S, N, N, 1);
        Ax[j][N-i]=Z[j][N-i];
    }
}
double[][] T=new double[N+1][N+1];
double[][] F= new double[N+1][1];
double[][] R = new double[N+1][1];
PotMatrices An= new PotMatrices();
MultMatrices Ln=new MultMatrices();
MultMatrices Re=new MultMatrices();
T=An.PMatrices(A,N,N);
F=Ln.MMatrices(T,Y, N, N, 1);
R=Re.MMatrices(T, F, N, N, 1);
for(i=0;i<=N;i++)
{
    B[i]=-1*R[i][0];
}
}
double[] Res= new double[N];
GaussJordan res = new GaussJordan();
Res=res.GaussJordan(A, B, N);
System.out.println("El polinomio característico es: ");
int c;
for(c=0;c<=N;c++)
{
    System.out.println(Res[c]+" ");
}
return 1;
}
}

```

MultMatrices.java

```
package proyecto.analisis.numerico;
```

```
/**
```

```
*
```

```
* @author Tere
```

```
*/
```

```
public class MultMatrices {
```



```

public double[][] MMatrices(double[][] A,double[][] B,int m,int n,int p)
{
    int i,j,k;
    double[][] P = new double[m][n];
    for(i=0;i<=m;i++)
    {
        for(j=0;j<=n;j++)
        {
            for(k=0;k<=p;k++)
            {
                P[i][j]=P[i][j]+A[i][k]*B[k][j];
            }
        }
    }
    return P;
}
}

```

PotMatrices.java

```
package proyecto.analisis.numerico;
```

```

/**
 *
 * @author Tere
 */
public class PotMatrices {

    public double[][] PMatrices(double[][] A,int N,int x)
    {
        int i,j,k;
        double[][] P = new double[N][N];
        do
        {
            for(i=0;i<=N-1;i++)
            {
                for(j=0;j<=N-1;j++)
                {
                    for(k=0;k<=N-1;k++)
                    {
                        P[i][j]=P[i][j]+A[i][k]*A[k][j];
                    }
                }
            }
            x=x-1;
        }while(x>=2);
        return P;
    }
}

```

GaussJordan.java

```
package proyecto.analisis.numerico;
```

```
/*Creado por Teresa Fiel Muñoz*/
```

```
public class GaussJordan {  
  
    public double[] GaussJordan(double[][] A,double[] B,int N)  
    {  
        int i,s,x,y;  
        double d,c;  
        for(i=0;i<=N-1;i++)  
        {  
            d=A[i][i];  
            for(s=0;s<=N-1;s++)  
            {  
                A[i][s]=(A[i][s])/d;  
            }  
            B[i]=(B[i])/d;  
            for(x=0;x<=N-1;x++)  
            {  
                if(i!=x)  
                {  
                    c=A[x][i];  
                    for(y=0;y<=N-1;y++)  
                    {  
                        A[x][y]=A[x][y]-c*A[i][y];  
                    }  
                    B[x]=B[x]-c*B[i];  
                }  
            }  
        }  
        return B;  
    }  
}
```

Consideraciones

El método de Krylov sólo resuelve polinomios característicos dada una matriz A.

Aplicación

La aplicación del método de Krylov fue aplicado a los principios de la función de transferencia de un sistema, ésta se calcula como:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{a_0 + a_1 s + \dots + a_m s^n}{b_0 + b_1 s + \dots + b_m s^n}$$

El numerador calcula los ceros del sistema y el denominador los polos, en este caso, los polos se calculan con las raíces del polinomio característico.

Partimos de conseguir los polos de una matriz de orden dos que se puede representar como:

$$\begin{vmatrix} 4 & -5 \end{vmatrix}$$

$$| 2 \quad -3 |$$

Utilizando el método normal de $\det(A - \lambda I)$, el resultado del polinomio característico es:

$$\lambda^2 - \lambda - 2 = 0$$

Ingresando los datos al programa

Interpolación por polinomio de Lagrange

Para la implementación del polinomio de Lagrange se utilizó una sola clase llamada Lagrange.java

Código Fuente

```
package proyecto.analisis.numerico;
import static proyecto.analisis.numerico.ProyectoAnalisisNumerico.N;

import java.util.Scanner;

/*Método de Lagrange creado por Teresa Fiel Muñoz*/
public class Lagrange {

    double[] nodos= new double[N];
    double[] funcion= new double[N];
    double valor;
    public void PedirDatos()
    {
        int i=0;
        while(i<=N)
        {
            System.out.println("Ingresa el valor de x"+i+": ");
            Scanner nodo = new Scanner(System.in);
            nodos[i]=nodo.nextDouble();
            System.out.println("Ingresa el valor de fx"+i+": ");
            Scanner func = new Scanner(System.in);
            funcion[i]=func.nextDouble();
        }
        System.out.println("Ingresa el valor de x que deseas interpolar: ");
        Scanner val = new Scanner(System.in);
        valor=val.nextDouble();
    }
    public void Lagrange()
    {
        int i=0;
        double fvalor=0;
        while(i<=N-1)
        {
```

```

        int j = 0;
        double L = 1;
        while(j <=N-1)
        {
            if (i !=j)
            {
                L = L *((valor - nodos[j]) / (nodos[i] - nodos[j]));
            }
            j=j+1;
        }
        fvalor=fvalor+(L*funcion[i]);
        i = i + 1;
    }
    System.out.println("El valor es: " + fvalor);
}
}

```

Consideraciones

El polinomio de Lagrange aplica para valores constantes de una función y es el recurso más exacto en los métodos de interpolación que se implementan en este proyecto.

Aplicación

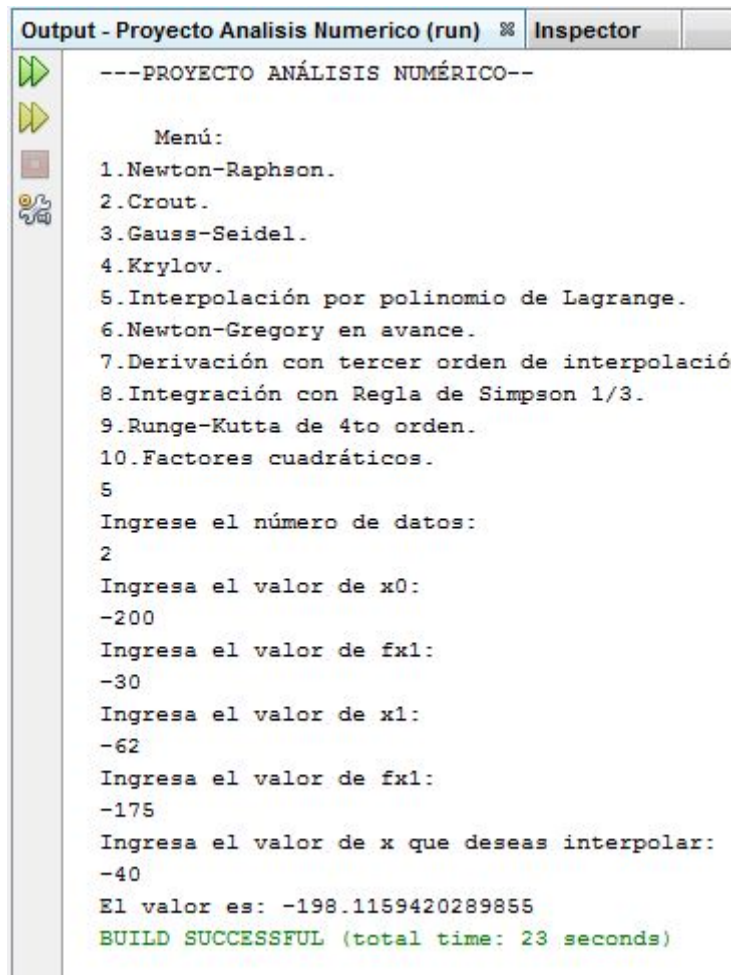
Para la aplicación del polinomio de interpolación tenemos que conseguir el valor de una temperatura en una escala creada que no puede resolverse con las conversiones de temperatura en el SI.

Consideramos dos escalas de temperatura: ($^{\circ}\alpha$) y ($^{\circ}\Omega$)

	$^{\circ}\Omega$	$^{\circ}\alpha$
Fusión del hielo	-200,0	-30,0
Ebullición del agua	-62,0	-175,0

Y nos piden calcular el valor en ($^{\circ}\alpha$) cuando se miden . El resultado por regresión lineal es de: $-198.11(^{\circ}\alpha)$.

Ingresamos los datos al programa



```
Output - Proyecto Analisis Numerico (run) % Inspector
---PROYECTO ANÁLISIS NUMÉRICO--

Menú:
1.Newton-Raphson.
2.Crout.
3.Gauss-Seidel.
4.Krylov.
5.Interpolación por polinomio de Lagrange.
6.Newton-Gregory en avance.
7.Derivación con tercer orden de interpolación
8.Integración con Regla de Simpson 1/3.
9.Runge-Kutta de 4to orden.
10.Factores cuadráticos.
5
Ingrese el número de datos:
2
Ingresa el valor de x0:
-200
Ingresa el valor de fx1:
-30
Ingresa el valor de x1:
-62
Ingresa el valor de fx1:
-175
Ingresa el valor de x que deseas interpolar:
-40
El valor es: -198.1159420289855
BUILD SUCCESSFUL (total time: 23 seconds)
```

Derivación por tercer orden de interpolación

Para la derivación por tercer orden se creó una sola clase llamada TerderOrdenD.java.

Código fuente

```
package proyecto.analisis.numerico;

import java.util.Scanner;
import static proyecto.analisis.numerico.ProyectoAnalisisNumerico.N;

/*Creado por Teresa Fiel*/
public class TercerOrdenD {
    double[] nodos= new double[N];
    double[] funcion= new double[N];
    double valor,r,r2,h1,h2,H;
    int subindice;

    public void PedirDatosTercerOD()
    {
        int i=0;
```

```

while(i<=N-1)
{
    System.out.println("Ingresa el valor de x"+i+": ");
    Scanner nodo = new Scanner(System.in);
    nodos[i]=nodo.nextDouble();
    System.out.println("Ingresa el valor de fx"+i+": ");
    Scanner func = new Scanner(System.in);
    funcion[i]=func.nextDouble();
    i=i+1;
}
System.out.println("Ingresa el valor de x para el que deseas obtener la derivada: ");
Scanner val = new Scanner(System.in);
valor=val.nextDouble();
i=0;
while(valor!=nodos[i])
{
    i=i+1;
}
subindice=i;
H=nodos[1]-nodos[0];
}
public int DerivadaTO()
{
    int[] Y0={-11,18,-9,2};
    int[] Y1={-2,-3,6,-1};
    int[] Y2={1,-6,3,2};
    int[] Y3={1,-6,3,2};
    int[] Yx0={2,-5,4,-1};
    int[] Yx1={1,-2,1,0};
    int[] Yx2={0,1,-2,1};
    int[] Yx3={1,4,-5,2};
    double[] D=new double[4];
    double[] D2=new double[4];
    int i,n=N;

    switch(subindice)
    {
        case 0:
            for(i=0;i<=3;i++)
            {
                D[i]=funcion[i]*Y0[i];
                D2[i]=funcion[i]*Yx0[i];
            }
            for(i=0;i<=2;i++)
            {
                D[i+1]=D[i+1]+D[i];
                D2[i+1]=D2[i+1]+D2[i];
            }
            break;
        case(1):
            for(i=0;i<=3;i++)

```

```

{
    D[i]=funcion[i]*Y1[i];
    D2[i]=funcion[i]*Yx1[i];
}
for(i=0;i<=2;i++)
{
    D[i+1]=D[i+1]+D[i];
    D2[i+1]=D2[i+1]+D2[i];
}
break;
case(2):
for(i=0;i<=3;i++)
{
    D[i]=funcion[i]*Y2[i];
    D2[i]=funcion[i]*Yx2[i];
}
for(i=0;i<=2;i++)
{
    D[i+1]=D[i+1]+D[i];
    D2[i+1]=D2[i+1]+D2[i];
}
break;
default:
if(subindice==N-2)
{
    for(i=3;i<=0;i--)
    {
        D[4-i]=funcion[N-i+1]*Y1[i-4];
        D2[4-i]=funcion[N-i+1]*Yx1[i-4];
    }
    for(i=0;i<=2;i++)
    {
        D[i+1]=D[i+1]+D[i];
        D2[i+1]=D2[i+1]+D2[i];
    }
}
else
{
    if(subindice==N-1)
    {
        for(i=3;i<=0;i--)
        {
            D[4-i]=funcion[N-i+1]*Y2[i-4];
            D2[4-i]=funcion[N-i+1]*Yx2[i-4];
        }
        for(i=0;i<=2;i++)
        {
            D[i+1]=D[i+1]+D[i];
            D2[i+1]=D2[i+1]+D2[i];
        }
    }
}
}

```

```

else
{
    if(subindice==N)
    {
        for(i=3;i<=0;i--)
        {
            D[4-i]=funcion[N-i+1]*Y3[i-4];
            D2[4-i]=funcion[N-i+1]*Yx3[i-4];
        }
        for(i=0;i<=2;i++)
        {
            D[i+1]=D[i+1]+D2[i];
            D2[i+1]=D2[i+1]+D2[i];
        }
    }
    else
    {
        if(subindice!=0)
        {
            for(i=0;i<=3;i++)
            {
                D[i]=funcion[subindice+i-1]*Y1[i];
                D2[i]=funcion[subindice+i-1]*Yx1[i];
            }
            for(i=0;i<=2;i++)
            {
                D[i+1]=D[i+1]+D[i];
                D2[i+1]=D2[i+1]+D2[i];
            }
        }
    }
}
break;
}
h1=1/(6*H);
h2=1/(H*H);
r=h1*D[3];
r2=h2*D2[3];
System.out.println("La primera derivada de "+valor+" es: "+r+"er");
System.out.println("La segunda derivada de "+valor+" es: "+r2+"er");
return 1;
}
}

```

Consideraciones

El programa de tercer orden de interpolación recibe n datos para llenar una tabla de funciones de la forma (x,f(x)) y posteriormente calcula la primera y segunda derivada.

Aplicación

Para este programa la aplicación que utilicé fue calcular la velocidad y aceleración de un objeto siguiendo el modelo de caída libre. Para esto debemos recordar las fórmulas de caída libre:

$$y = H - \frac{1}{2}gt^2$$

$$v = -gt$$

$$a = -g$$

Donde H es la altura desde donde se deja caer el cuerpo, t es la variable de tiempo, g es la gravedad (consideramos $9.81 \frac{m}{s^2}$ en este caso), v es velocidad y a es aceleración.

Para este problema consideraremos un cuerpo que cae desde 40m. Realizamos entonces una tabla:

X	f(x)= s[m]	f'(x)=v[$\frac{m}{s}$]	f''(x)=a[$\frac{m}{s^2}$]
0.3	39.56	-2.94	9.81
0.6	38.12	-6.08	9.81
0.9	36.03	-8.82	9.81
1.2	32.94	-11.76	9.81

Deseamos calcular la velocidad y aceleración para el punto 0.6, ingresamos entonces los datos al programa.

```
Output - Proyecto Analisis Numerico (run) % Inspector
3.Gauss-Seidel.
4.Krylov.
5.Interpolación por polinomio de Lagrange.
6.Newton-Gregory en avance.
7.Derivación con tercer orden de interpolación.
8.Integración con Regla de Simpson 1/3.
9.Runge-Kutta de 4to orden.
10.Factores cuadráticos.
7
Ingrese el número de datos:
4
Ingresar el valor de x0:
0.3
Ingresar el valor de fx1:
39.56
Ingresar el valor de x1:
0.6
Ingresar el valor de fx1:
38.12
Ingresar el valor de x2:
0.9
Ingresar el valor de fx1:
36.03
Ingresar el valor de x3:
1.2
Ingresar el valor de fx1:
32.94
Ingresar el valor de x para el que deseas obtener la derivada:
0.6
La primera derivada de 0.6 es: -5.68888888888888784+er
La segunda derivada de 0.6 es: -7.22222222222222127+er
BUILD SUCCESSFUL (total time: 2 minutes 17 seconds)
```

El programa contempla ciertos errores relativos para el tercer orden de interpolación, pero tenemos un porcentaje bajo de éste.

Integración por regla de Simpson $\frac{1}{3}$

El método de integración por regla de Simpson utiliza una sola clase que se llama Simpson.java.

Código fuente

```
package proyecto.analisis.numerico;
import java.util.Scanner;
```

```
/*Creado por Teresa Fiel*/
public class Simpson {
```

```
    double limi,limsu;
    int CASO=0;
```

```

String funcion;
double func;

public void PedirDatosSimpson()
{
    /*Scanner caso = new Scanner(System.in);
    System.out.println("Ingresa la naturaleza de la función que deseas integrar: ");
    System.out.println("1.Trigonométrica.");
    System.out.println("2.Polinómica.");
    System.out.println("3.Trascendente.");
    CASO=caso.nextInt();
    System.out.println("Ingresa la función que deseas integrar: ");
    Scanner ffunc = new Scanner(System.in);
    funcion = ffunc.nextLine();*/
    Scanner limiteinferior = new Scanner(System.in);
    System.out.print("Ingresa el límite inferior: ");
    limi = limiteinferior.nextDouble();
    Scanner limitesuperior = new Scanner(System.in);
    System.out.print("Ingresa el límite superior: ");
    limsu = limitesuperior.nextDouble();
}

public void Simpson()
{
    final int n = 1000;
    double h=(limsu-limi)/n;
    double suma=f(limi)+f(limsu);
    if(CASO==1)
    {
        limi=limi*(Math.PI/180);
        limsu=limsu*(Math.PI/180);
    }
    for(int i=1; i<=n; i=i+2)
        suma+=4*f(limi+i*h);

    for(int i=2; i<=n; i=i+2)
        suma+=2*f(limi+i*h);

    System.out.println("La integral aproximada de tu función es: "+(suma*h/3));
}

public double f(double x)
{
    return Math.pow(1.6,2)*9*1/Math.pow(x,2);
}
}
Consideraciones

```

Este método fue difícil de implementar dada la naturaleza de los valores de entrada, se planteaba recibir funciones de índole polinómico, trascendente y trigonométrico pero al final

para la aplicación, al igual que en Runge-Kutta, se tuvo que agregar la función ya explícita como variable de retorno.

Aplicación

La aplicación que le día al método de integración fue el cálculo del trabajo producido por dos cargas puntuales.

El trabajo de un electrón parte de la ley de Coulomb, y se define como:

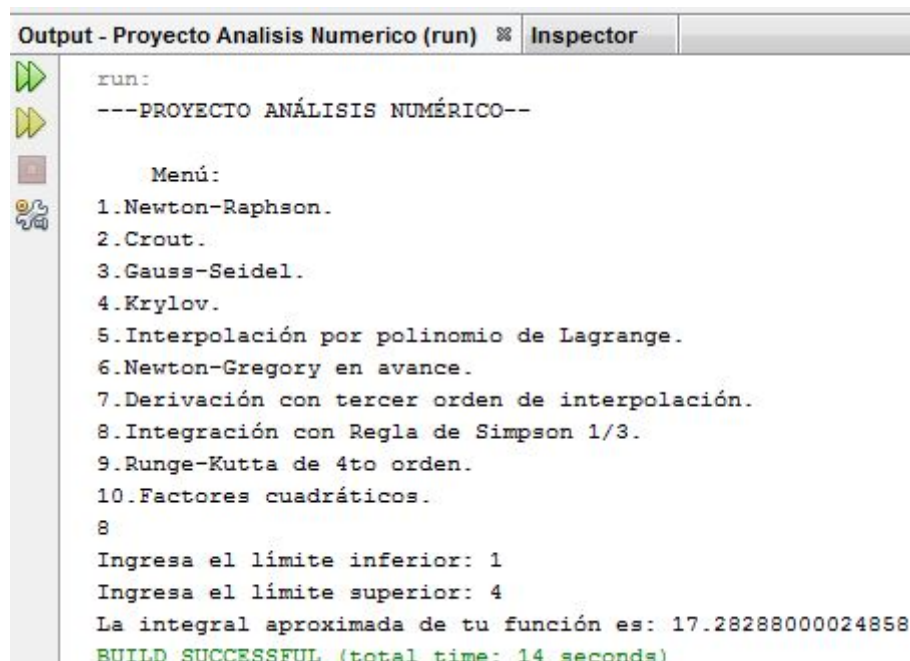
$$W = \int_a^b \frac{kq_1q_2}{x^2} dx$$

El problema que queremos resolver es encontrar el trabajo producido por dos electrones en una distancia de 4[mm], por lo que buscamos calcular:

$$W = \int_{1 \times 10^{-3}}^{4 \times 10^{-3}} \frac{(9 \times 10^9)(1.6 \times 10^{-19})}{x^2} dx$$

El resultado real de esta integral es de $1.728 \times 10^{-16} [J]$

Introducimos los valores al programa con números enteros (sin notación científica)



```
Output - Proyecto Analisis Numerico (run) Inspector
run:
---PROYECTO ANÁLISIS NUMÉRICO---

Menú:
1.Newton-Raphson.
2.Crout.
3.Gauss-Seidel.
4.Krylov.
5.Interpolación por polinomio de Lagrange.
6.Newton-Gregory en avance.
7.Derivación con tercer orden de interpolación.
8.Integración con Regla de Simpson 1/3.
9.Runge-Kutta de 4to orden.
10.Factores cuadráticos.
8
Ingresa el límite inferior: 1
Ingresa el límite superior: 4
La integral aproximada de tu función es: 17.28288000024858
BUILD SUCCESSFUL (total time: 14 seconds)
```

Los coeficientes del trabajo son los mismos pero por el análisis de unidades el resultado tiene que estar en Femto Joules (10^{-15}), se realizó de esta manera para evitar que el valor de retorno tuviera pérdida de información ya que la variable es de tipo Double. Además de que la fórmula ya estaba implementada en la clase Simpson.java.

Runge-Kutta (4to orden)

Para la implementación del método de Runge-Kutta se utilizó una clase llamada RungeKutta.java

Código Fuente

```
package proyecto.analisis.numerico;
import java.util.Scanner;
/*Creado por Teresa Fiel*/
public class RungeKutta {

    double x0,y0,x,h;

    public void pedirDatos()
    {
        System.out.println("Ingresa el valor de x0: ");
        Scanner a0 = new Scanner(System.in);
        x0=a0.nextDouble();
        System.out.println("Ingresa el valor de y0: ");
        Scanner b0 = new Scanner(System.in);
        y0=b0.nextDouble();
        System.out.println("Ingresa el valor de x: ");
        Scanner a = new Scanner(System.in);
        x=a.nextDouble();
        System.out.println("Ingresa el valor de h: ");
        Scanner H = new Scanner(System.in);
        h=H.nextDouble();
    }

    double Deriv(double x, double y)
    {
        return (1000-3*y);
    }

    public void RungeKutta()
    {
        RungeKutta d1 = new RungeKutta();

        int i;
        int n = (int)((x - x0) / h);
        double r1, r2, r3, r4;
        double y = y0;

        for (i=1;i<=n;i++)
        {
            r1 = h*(d1.Deriv(x0, y));
            r2 = h*(d1.Deriv(x0 +0.5*h,y+0.5*r1));
            r3 = h*(d1.Deriv(x0 +0.5*h,y+0.5*r2));
            r4 = h*(d1.Deriv(x0 +h,y+r3));

            y =(y+(1.0/6.0)*(r1+2*r2+2*r3+r4));
```

```

        x0=x0+h;
    }
    System.out.println("El valor calculado de y en x="+x+" es: "+y);
}
}

```

Consideraciones

El código fuente debe modificarse para ingresar el valor de la ecuación diferencial ya despejada, la función Deriv() regresa el valor de esa función valuada en un punto (x,y). Además está vigente el problema de que la función aún no aplica para valores muy grandes ya que sus variables de retorno son del tipo Double.

Aplicación

La aplicación que utilicé para Runge Kutta fue calcular la velocidad de un motor que recibe de fuente 20V a lo largo del tiempo con una ecuación diferencial dada.

$$20 = (0.2)\frac{dv}{dt} + 0.06v$$

Siendo v la velocidad que tendremos en el valor de y, y t el tiempo que tendremos en el valor de x. La ecuación diferencial despejada es:

$$\frac{dv}{dt} = 1000 - 3v$$

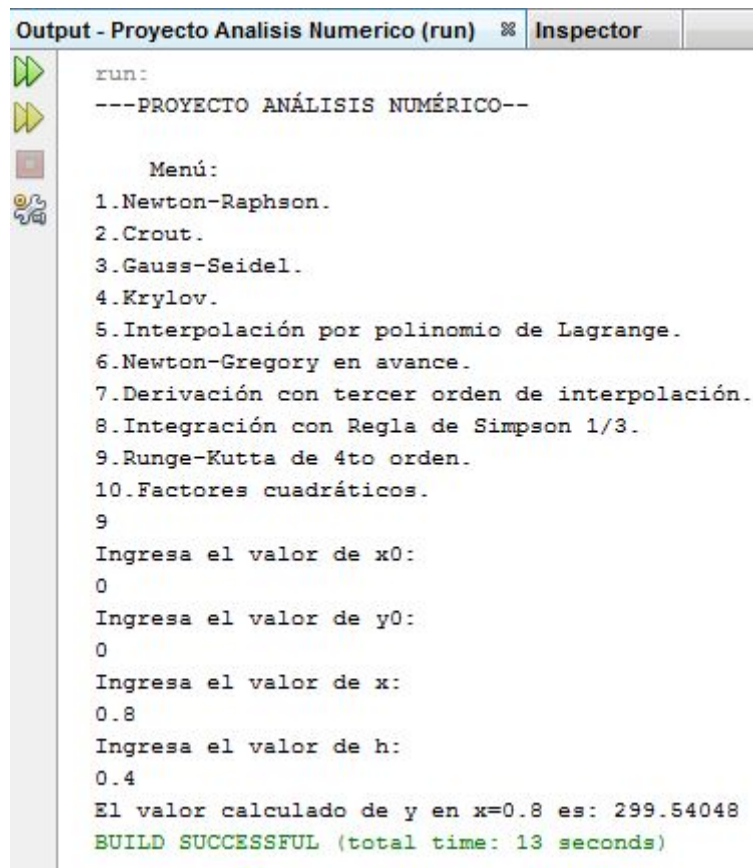
Las condiciones iniciales son $t_0 = 0, v_0 = 0$. Podemos calcular la velocidad en un tiempo igual a 0.8 segundos en intervalos de 0.4 segundos.

Esta ecuación se resuelve con diferenciales exactas y la función del tiempo está dada por:

$$v(t) = \frac{1000}{3} - \frac{1000}{3}e^{-3t}$$

Y valuada en $t=0.8$ la velocidad corresponde a $303.09 \frac{rad}{s}$

Al ingresar los datos de entrada, la solución al sistema es:



```
run:
---PROYECTO ANÁLISIS NUMÉRICO--

Menú:
1.Newton-Raphson.
2.Crout.
3.Gauss-Seidel.
4.Krylov.
5.Interpolación por polinomio de Lagrange.
6.Newton-Gregory en avance.
7.Derivación con tercer orden de interpolación.
8.Integración con Regla de Simpson 1/3.
9.Runge-Kutta de 4to orden.
10.Factores cuadráticos.
9
Ingresa el valor de x0:
0
Ingresa el valor de y0:
0
Ingresa el valor de x:
0.8
Ingresa el valor de h:
0.4
El valor calculado de y en x=0.8 es: 299.54048
BUILD SUCCESSFUL (total time: 13 seconds)
```

$$v(0.8) = 299.5448 \frac{rad}{s}$$

La solución con el programa es bastante aproximada, el porcentaje de error es de 1.1696%, recordemos que el porcentaje de error puede disminuir si el valor de h (0.4) decrece.

Factores cuadráticos

Para la implementación del método se utilizó una sola clase llamada FactoresCuadraticos.java.

Código Fuente

```
package proyecto.analisis.numerico;
import java.util.Scanner;
```

```
public class FactoresCuadraticos
{
    double a,b,c;
    double r1 = 0;
    double r2 = 0;
```

```
    public void pedirDatosFC()
    {
```

```

        System.out.println("Ingresa el valor de a: ");
        Scanner ax = new Scanner(System.in);
        a = ax.nextDouble();
        System.out.println("Ingresa el valor de b: ");
        Scanner bx = new Scanner(System.in);
        b = bx.nextDouble();
        System.out.println("Ingresa el valor de c: ");
        Scanner cx = new Scanner(System.in);
        c = cx.nextDouble();

    }

    public void FactoresCuadraticos()
    {
        double discriminante=(b*b)-4*(a*c);

        if (discriminante>0)
        {
            r1=(-b+Math.sqrt(discriminante))/(2*a);
            r2=(-b-Math.sqrt(discriminante))/(2*a);
            System.out.println("Las raíces de la ecuación son: " + r1 + " y " + r2);
        }
        if (discriminante==0)
        {
            r1=-b/(2*a);
            r2=-b/(2*a);
            System.out.println("Las raíces de la ecuación son: " +r1+" y "+r2);
        }
        if (discriminante<0)
        {
            System.out.println("Lo siento, tu ecuación no tiene raíces reales.");
        }
    }
}

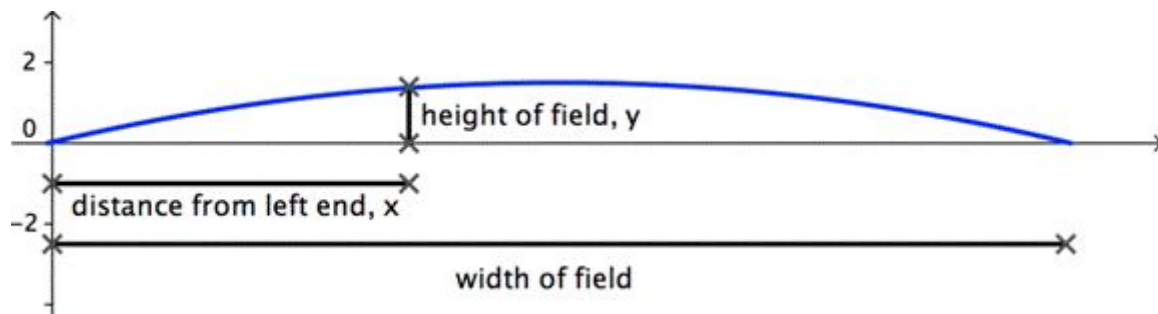
```

Consideraciones

El programa de FactoresCuadraticos resuelve únicamente sistemas de orden dos, es decir solo calcula raíces para polinomios de la forma $ax^2 + bx + c$, además no está diseñado para resolver raíces imaginarias.

Aplicación

Para el método de factores cuadráticos calcularemos la distancia recorrida en una función con tiro parabólico.



Si queremos representar la sección transversal del campo podríamos modelar la superficie con una ecuación de segundo grado, para este ejemplo utilizaremos la función dada por:

$$y = -0.0241x^2 + x + 5.5$$

donde x es la distancia de izquierda a derecha y y es la altura del campo. Buscamos calcular el ancho, por lo tanto igualamos la ecuación a cero y despejamos la ecuación para ingresarla al programa.

$$-0.0241x^2 + x + 5.5 = 0$$

Las raíces de la ecuación son -4.9 y 46.4, analizando los resultados sabemos que no hay distancias negativas, por lo que -4.9 queda descartada y concluimos que la distancia del campo es 46.4.

Al ingresar estos datos al programa obtenemos la misma solución.

Output - Proyecto Analisis Numerico (run) %	Inspector
<pre> Menú: 1.Newton-Raphson. 2.Crout. 3.Gauss-Seidel. 4.Krylov. 5.Interpolación por polinomio de Lagrange. 6.Newton-Gregory en avance. 7.Derivación con tercer orden de interpolación. 8.Integración con Regla de Simpson 1/3. 9.Runge-Kutta de 4to orden. 10.Factores cuadráticos. 10 Este método solo calcula raíces de polinomios de la forma ax*2+b*x+c Ingresa el valor de a: -0.0241 Ingresa el valor de b: 1 Ingresa el valor de c: 5.5 Las raíces de la ecuación son: -4.917272411863403 y 46.411048345473354 BUILD SUCCESSFUL (total time: 17 seconds) </pre>	

Conclusiones

En este proyecto se examinaron diez métodos numéricos distintos para obtener la solución aproximada a ciertas operaciones comunes en la ingeniería, se implementaron programas para resolver estos métodos de manera más eficaz y de un grado más alto de complejidad. Además se dio un uso particular para cada método en distintas áreas de la ingeniería.

Fue posible comprender mejor los métodos y obtener un enfoque distinto al ver la utilidad que el análisis numérico tiene en mi carrera, en particular lo interesante es el tiempo de ejecución que llevaba resolver sistemas de orden n , ya que con métodos convencionales que se han visto en materias pasadas no sería posible llegar a una solución tan rápida. Para esto también hay que considerar la propagación del error que muchos métodos numéricos manejan pero en general fue muy ilustrativo tener que programar cada método y darle un uso específico.