Capítulo 6

Interpolación

6.1. Introducción

En esta práctica estudiamos los diferentes métodos de interpolación. En primer lugar vemos diferentes formas de construir el polinomio interpolador en un conjunto de puntos: algoritmos de Lagrange, Neville y Newton o diferencias divididas. En segundo lugar estudiaremos la interpolación por splines libres y sujetos. En esta práctica se presentan sólo los programas. Para el fundamento teórico de los métodos, consultar el Tema 5 de los apuntes de teoría de Cálculo Numérico en C++.

6.2. Interpolación polinomial

Hemos programado los diferentes métodos de interpolación polinomial como funciones de patrones, dentro de la cabecera `interpolación.h`, incluída en el directorio P6. Todos estos programas utilizan la librería TNT, por lo que hay que incluir `-I. -I./templates` como opción del compilador.

6.2.1. Método de Lagrange

El programa l`agrange.cpp` ilustra el uso del template Lagrange incluído en `interpolacion.h`.
:

```
template <class T>  T Lagrange(Vector<T> & x, Vector<T> & y, T& xx)
{
//Calcula el valor  en el punto xx del polinomio interpolador de orden n-1
//que interpola los n puntos cuyas abcisas y ordenadas son los vectores x e y

int n=x.dim();
Vector<T> l(n);
for (int i=1;i<=n;i++) l(i)=y(i);
```

```
for (int i=1;i<=n;i++){
  T num=(xx-x(i));
    for (int j=1; j<=n;j++)
      if (j != i) l(j)=l(j)*num/(x(j)-x(i));

}
T p=0;
for (int i=1;i<=n;i++) p=p+l(i);
return p;
}
```

Este programa realiza la interpolación por el método de Lagrange de una de tres funciones predefinidas, utilizando `n` puntos igualmente espaciados, comprendidos entre un valor máximo y mínimo. Seguidamente, calcula el valor del polinomio interpolador en un único punto y lo compara con el valor exacto:

```
#include <cmath>
#include <iostream>
#include "interpolacion.h"
typedef double  Real;
using namespace std;
using namespace CALNUM;
// Programa de interpolacion por Lagrange de orden n - 1
// en n puntos igualmente espaciados
// Calcula el valor de la funcion y
// del polinomio en un punto arbitrario

template < class T > T fun(T x, int i){
if (i == 1)
return sin(x);
else if (i == 2)
return tan(x);
else if (i = 3)
return T(1) / sqrt(1 + x * x);
}

int main(){
cout << "Entrar numero de puntos" << endl;
int             n;
cin >> n;
Vector < Real > x(n), y(n);
cout << "Entrar limite inferior" << endl;
```

```
Real            x0, x1;
cin >> x0;
cout << "Entrar limite superior" << endl;
cin >> x1;
Real            delta = (x1 - x0) / (n - 1);

cout << "Entrar funcion" << endl <<
"1. sin(x)" << endl <<
"2. tan(x)" << endl <<
"3. runge(x)" << endl;
int             ifun;
cin >> ifun;

for (int i = 1; i <= n; i++) {
x(i) = x0 + i * delta;
y(i) = fun(x(i), ifun);
}
Real            xint;
cout << " Entrar x" << endl;
cin >> xint;
Real            yint = Lagrange < Real > (x, y, xint);
cout << " x " << "y-interp   " << "   f(x) " << endl;
cout << xint << "  " << yint << "    " << fun(xint, ifun) << endl;
}
```

Para compilarlo escribid el comando:

```
g++ -I. -I./templates -o lagrange lagrange.cpp
```

6.2.2. Método de Newton

El templates `difdiv.cpp` calcula los coeficientes del polinomio interpolador de Newton, mientras que el template `eval_difdiv` toma como argumentos el vector de coeficientes del polinomio interpolador calculado por `difdiv.cpp`, el vector de los puntos de interpolación y un valor de la abcisa y evalúa el valor del polinomio interpolador en esta abcisa:

```
template <class T> Vector<T> difdiv(const Vector<T>  &x, const Vector<T>& y){
int n=x.size()-1;
Vector<T> a=y;
//n iteraciones
for (int j=1; j<=n; j++)
// En iteracion k se cambian los n-k-1 coeficientes de indice mas alto
    for (int i=n; i >= j; i--)
        a[i]=( a[i] - a[i-1] ) / ( x[i] - x[i-j] );
```

```
return a;
}

template <class T> T
    eval_difdiv(const T &xx, const Vector<T> &x, const  Vector<T> &a){
int n=x.size()-1;
//n iteraciones: evaluacion por Horner del polinomio
    T y=a[n];
    for (int i=n-1; i >= 0; i--)
        y = a[i] + (xx-x[i])*y;
return y;
}
```

El programa `interpol.cpp` permite interpolar una serie de funciones predefinidas (fácilmente ampliable) en un número `n` de puntos igualmente espaciados, o en los ceros de los polinomios de Chebychev, según opción, y dibuja los resultados con `Gnuplot`. El programa `interpol.cpp`, entre otras posibilidades, sirve para ilustrar las patologías de la interpolación polinomial con puntos igualmente espaciados para las funciones de Runge y Berstein.

```
//Programa de interpolacion polinomial
//con ilustracion de los  ejemplos de  Runge y Bernstein
//Compilar con g++ -I. -I./templates interpol.cpp gnuplot_i_calnum.cpp
#include <cmath>
#include <iostream>
#include <iomanip>
#include "algebralineal.h"
#include "interpolacion.h"
#include "gnuplot_i_calnum.h"
using namespace std;
inline double fun(double x, int ft)
{
    switch (ft)
    {
        case 0:
            return 1. / (1. + 25 * x * x);
            break;
        case 1:
            return abs(x);
            break;
        case 2:
//Definir F(x)
            return sin(x);
        case 3:
```

```
//Definir F(x)
            return x*exp(x);
        case 4:
//Definir F(x)
            return atan(x);
    }
}


int main()
{
//Programa de interpolacion
    double x0, x1;

    cout << "Entrar xmin" << endl;
    cin >> x0;
    cout << "Entrar xmax" << endl;
    cin >> x1;
    int nd;

    cout << "Entrar orden polinomio interpolador" << endl;
    cin >> nd;
    int ft=-1;

    while ((ft != 0) && (ft != 1) && (ft != 2)  && (ft != 3) && (ft != 4))
    {
        cout << "Entrar funcion: 0 - Runge   1 - Bernstein  2 - sin(x)
                 3 - x*exp(x)  4 - atan(x) " << endl;
        cin >> ft;
    }
    cout << "Entrar tipo de espaciado: 1: uniforme  2: Chebychev" << endl;
    int es = 0;

    while ((es != 1) && (es != 2))
    {
        cin >> es;
        if ((es != 1) && (es != 2))
        {
            cout << "Espaciado erroneo" << endl;
            cout << "Entrar tipo de espaciado: 1: uniforme  2:Chebychev" << endl;
        }
    }
    double xint;
```

```
    double pi = 4 * atan(1.);

//Espaciado entre los nodos
    double h = (x1 - x0) / nd;

    nd++;
//El numero de nodos es una unidad mayor que el orden
    Vector < double >x(nd);

//Construccion de vectores

// Calculo de las abcisas de interpolacion
    switch (es)
    {
        case 1:
            for (int i = 0; i < nd; i++)
                x[i] = x0 + i * h;
            break;
        case 2:
            for (int i = 0; i < nd; i++)
                x[i] = (x0 - x1) / 2 * cos((2 * i + 1) * pi / (2 * nd)) +
                    (x1 + x0) / 2.;
// cout << x << endl;
            break;
        default:
            cout << "Espaciado erroneo" << endl;
            return 1;
    }

//Calculo de las ordenadas de interpolacion y de los coeficientes del
//polinomio
    Vector < double >y(nd);

    for (int i = 0; i < nd; i++)
        y[i] = fun(x[i], ft);
    Vector < double >a = difdiv < double >(x, y);
    cout<<" Nodos de interpolacion"<<endl;
    cout<<"   i  "<<"    x   "<< "         y    "<<endl;
    for (int i=0;i<nd;i++) cout <<"   "<< i+1 <<setw(10)<<x[i]<<
            "   "<<setw(10)<<y[i] << endl;
    cout << endl<<"coeficientes diferencias divididas a= " << a << endl;
```

```
//evaluacion del polinomio y funcion en los puntos de dibujo(200)
// se dibuja un 2*deltax100 % fuera del intervalo
    int n = 200;
    double delta=0.;
    cout << " Entrar delta" <<
        "(Dibujo en [x0-(x1-x0)*delta, x1+(x1-x0)*delta] "
         << endl << endl;
    cin>>delta;
    x1+=(x1-x0)*delta;
    x0-=(x1-x0)*delta;
    double hh = (x1 - x0) / n;

    n++;
    Vector < double >xx(n), yy(n), ff(n);

    cout << "       x    " << "    p(x)    " <<
        "   f(x)    " << "  |f(x)-p(x)|  " << endl;
    for (int i = 0; i < n; i++)
    {
        xx[i] = x0 + i * hh;
        yy[i] = eval_difdiv < double >(xx[i], x, a);
        ff[i] = fun(xx[i], ft);

//Impresion de la funcion en 20 puntos igualmente espaciados
        if ((i / 10) * 10 == i)
            cout << setw(10) << xx[i] << "  " << setw(10) << yy[i] << "  " <<
                setw(10) << ff[i] << "  " << setw(10) << abs(yy[i] - ff[i]) << endl;
    }

//Dibujo del polinomio y la funcion
    Gnuplot g1 = Gnuplot();
//Seleccionamos escala logaritmica para ciertas funciones
// if ((ft == 0) || (ft == 1)) g1.cmd("set logscale y");
    if ((ft == 0))
        g1.cmd("set logscale y");
    g1.set_style("lines");
    g1.plot_xy(xx, ff, " ");
    g1.set_style("points");
    g1.plot_xy(xx, yy, " ");
//Eleccion del tiempo de permanencia de la grafica en pantalla
    int SLEEP_LGTH = 25;

    sleep(SLEEP_LGTH);
```

```
    return 0;
}
```

El programa `interpol1.cpp` calcula el valor de los polinomios interpoladores, en un intervalo dado, de grados comprendidos entre e 1 y un valor máximo n, en un conjunto de puntos, situados dentro o fuera del intervalo de interpolación. Los puntos de interpolación en el intervalo pueden estar uniformente espaciados, o tomarse como los ceros de los polinomios de Chebychev, según opción. Este programa produce como resultado las diferencias entre cada uno de los polinomios intterpoladores y la función exacta y entre cada uno de los polinomios interpoladores y el polinomio de grado inmediatamente más bajo. La finalidad de este programa es estudiar la convergencia del polinomio interpolador en un punto dado, dentro o fuera del intervalo de interpolación. El programa dif_div_triangulo.cpp genera el triángulo de valores que se produce en el cálculo de los coeficientes del algoritmo diferencias divididas (de utilidad para la resolución de problemas).

6.2.3. Método de Neville

La funcion para interpolar por el método de Neville es la siguiente, incluida en `templates.h`:

```
template <class T>
 T Neville(const Vector<T>  &x, const Vector<T>& y, const T &xx){
int n=x.size();
Vector<T> xdif(n);
for (int i=0; i<n; i++) xdif[i]=xx-x[i];
Vector<T> p=y;
n--;
//n iteraciones
for (int j=1; j<=n; j++)
// En iteracion k se cambian los n-k-1 coeficientes de indice mas alto
    for (int i=n; i >= j; i--)
        p[i]=(xdif[i]* p[i-1] - xdif[i-j]*p[i] ) / ( xdif[i] - xdif[i-j] );
return p[n];
}
```

El programa `neville.cpp` ilustra la forma de llamar a esta función patrón para interpolar funciones:

```
//Compilar con g++ -I. -I./templates neville.cpp
#include <cmath>
#include <iostream>
#include "algebralineal.h"
#include<iomanip>
```

```
#include "interpolacion.h"
typedef double Real;
using namespace std;
using namespace CALNUM;


template <class T>  T fun( T x, int i){
if (i==1)
  return sin(x);
else if(i==2)
  return tan(x);
else if (i=3)
  return T(1)/sqrt(1+x*x);
}

int main()
{
cout<< "Entrar numero de puntos"<< endl;
int n;
cin >> n;
Vector<Real> x(n), y(n);
cout<<"Entrar limite inferior"<<endl;
Real x0, x1;
cin>> x0;
cout<<"Entrar limite superior"<<endl;
cin>>x1;
Real delta= (x1-x0)/(n-1);

cout<< "Entrar funcion" << endl<<
"1. sin(x)"<<endl<<
"2. tan(x)" <<endl <<
"3. runge(x)" << endl;

// Entrar tipo de funcion
int ifun;
cin >>ifun;

for (int i=1; i<=n;i++) {
x(i) = x0 + i*delta;
y(i)= fun(x(i), ifun);
}
Real xint;
cout<<" Entrar x"<<endl;
```

```
cin>> xint;
Real yint = Neville<Real>(x,y,xint);
cout<<setprecision(10)<<" x "<< "y-interp   " <<  "   f(x) "<< endl;
cout<<xint<< "  "<<yint<<"    "<<fun(xint,ifun)<<endl;
return 0;
}
```

El programa `interpol_triangulo_neville.cpp` genera el triángulo de Neville que se produce en el cálculo (de utilidad para la resolución de problemas). Un fichero de datos es `interpolNM.dat`.

6.3. Generación de datos

Se proporciona el programa `gendat.cpp` para generar datos de diferentes funciones, que son directamente legibles como vectores de la librería TNT y por *Gnuplot*. Para compilar este programa escribid simplemente el comando:

```
g++ -o gendat gendat.cpp
```

El programa `gendat` escribe tres ficheros: `funcion.d`, `funcion.gpl` y `gnuplot.macro`. donde `funcion` es el nombre de la función elegida. Así, si se elige el polinomio, escribe `pol.d` y `pol.gpl`, además de `gnuplot.macro`. En esta práctica, tomaremos nulo el error de los datos, es decir, daremos como 0 los valores de `sigma` y `dispersion`. El programa genera `n` puntos entre `xmin` y `xmax`, de una de 6 funciones previstas: 1) $ax+b$, 2) ax^2+bx+c, 3) Definida por el usuario, 4) $N\exp(-t/\tau)$, 5) $(a+0{,}1x)^b$ y 6) Combinación lineal de funciones. Una vez elegida la función, pide los parámetros correspondientes a dicha función. La opción sexta, por ejemplo, contruye una combinación lineal de funciones, dando como resultado

$$f(x) = \sum_{i=0}^{7} a(i) * fun(x,i)$$

donde el conjunto `fun(x,i)` está compuesto, por orden, por las siguientes funciones: $\sin(x)$, $\exp(-x)$, $\sqrt{x}$, $\log(x)$, $\tan(x)$, $\cosh(x)$, $\cos(x)$ y x^3. Si, por ejemplo, sólo se dan 3 parámetros, se toma una combinación lineal de las tres primeras funciones. Si queremos generar la función x^3, debemos de dar 7 parámetros, todos nulos salvo el séptimo, que tomaremos como 1. Si damos como parámetros 1, 2, 0, 0, 0, 0, 3, 1, construiremos la función

$$f(x) = \sin(x) + 2\exp(-x) + 3\cos(x) + x^3$$

6.4. Splines

El método de splines es cada vez más frecuentemente utilizado para aproximar funciones con comportamiento abrupto, o muy variable, para las que consigue una reproducción excelente, mientras que la aproximación polinomial empeora con el grado del polinomio. Casos típicos

donde la aproximación polinomial falla miserablemente son las funciones de Runge y Bernstein, ilustradas en los anteriores programas. El programa `splines.cpp` realiza tanto la interpolación por splines naturales como de splines sujetos. Pide primero el número de nodos, y luego los valores de las abcisas en los nodos, cuyo único requerimiento es que sean distintas y estén ordenasdas de menor a mayor. Tiene como posibilidades las funciones de Runge y Bernstein, la función sin(x) y una función definida libremente por el usuario. Imprime como resultado los coeficientes de cada spline y los valores de los splines y la función original en una serie de puntos igualmente espaciados en el intervalo, tanto para splines libres como sujetos. También dibuja los splines y la función interpolada, para splines libres y sujetos, utilizando la librería `gnuplot_i++`. Este es el código del programa:

```
//g++ -I.  -I./templates splines.cpp gnuplot_i_calnum.cpp
//

#include <cmath>
#include <iostream>
#include<iomanip>
#include "algebralineal.h"
#include "interpolacion.h"
#include "gnuplot_i_calnum.h"
using namespace std;
using namespace CALNUM;

inline double fun(double x, int ft)
{
    switch(ft)
    {
        case 0:
            return 1./(1.+25*x*x);
            break;
        case 1:
            return abs(x);
            break;
        case 2:
            return sin(x);
        case 3:
            return 2*atan(x) +exp(5-x)  + 5*cos(x);
    }
}


template <class T> int
splines(Vector<T> &x, Vector<T> &a,  Vector<T> &b, Vector<T> &c, Vector<T> &d)
```

```
{
    cout<<"splines naturales"<<endl;
    int n=x.size();                                  //n numero de puntos
    int nint= n-1;                                   //numero de intervalos
    int neq =nint -1;                                //numero de ecuaciones

//Creacion vector diagonal superior
    Vector<T> h(nint);
    for( int i=0; i<nint; i++) h[i]=x[i+1]-x[i];

//Creacion Vector diagonal
    Vector<T> u(nint), v(nint);  //u(0), v(0) no estan definidos
    for( int i=1; i<nint; i++) u[i]=2*(h[i-1]+h[i]);
    for( int i=0; i<neq; i++)
        v[i+1]=3*( (a[i+2]-a[i+1])/h[i+1] - (a[i+1]-a[i])/h[i] );

//Eliminacion de Gauss
    for( int i=1; i<neq; i++)
    {
        u[i+1]=u[i+1]-h[i]*h[i]/u[i];
        v[i+1]=v[i+1]-h[i]*v[i]/u[i];
    }

//Sustitucion hacia atras
    c[nint-1]=v[nint-1]/u[nint-1];
    d[nint-1]= -c[nint-1]/(3*h[nint-1]);
    for( int k=nint-2; k>0; k--)
    {
        c[k]=(v[k]-h[k]*c[k+1])/u[k];
        d[k]=(c[k+1]-c[k])/(3*h[k]);
    }

//Valores especiales en x0
    c[0]= T(0);
    d[0]=c[1]/(3*h[0]);
    b[0]=(a[1]-a[0])/h[0]-c[1]*h[0]/3;

// Calculo de b[k]
    for( int k=0; k<nint-1; k++)  b[k+1]=b[k]+(c[k]+c[k+1])*h[k];

//Impresion de coeficientes
    cout<< "     a     "<<"    b     "<<"     c     "<< "     d     "<<endl;
    for(int i=0; i<nint; i++) cout<<setprecision(4)<<setw(8)<<a[i]<<"  "<<
```

```
            setw(8)<<b[i]<<"  "<<setw(8)<<c[i]<<"  "<<setw(8)<<d[i]<<endl;
    cout<<endl;

    return 0;
}


template <class T> int
splines_sujetos(Vector<T> &x, Vector<T> &a,  Vector<T> &b,
                  Vector<T> &c, Vector<T> &d, T& fp0, T& fp1)
{
    cout<<"splines sujetos"<<endl;
    int n=x.size();                                 //n numero de puntos
    int nint= n-1;                                  //numero de intervalos
// neq =nint -1; //numero de ecuaciones

//Creacion vector diagonal superior
    Vector<T> h(nint);
    for( int i=0; i<nint; i++) h[i]=x[i+1]-x[i];

//Creacion Vector diagonal y de terminos independientes
    Vector<T> u(n), v(n);
// Valores especiales de u(0), v(0)
    u[0]=2*h[0];
    v[0]= 3*( (a[1]-a[0])/h[0] -fp0);

    for( int i=1; i<nint; i++) u[i]=2*(h[i-1]+h[i]);
    for( int i=0; i<nint-1; i++)
        v[i+1]=3*( (a[i+2]-a[i+1])/h[i+1] - (a[i+1]-a[i])/h[i] );

//Valores especiales de u(n-1), v(n-1)
    v[nint]=3*(fp1- (a[nint]-a[nint-1])/h[nint-1] );
    u[nint]=2*h[nint-1];

//Eliminacion de Gauss
    for( int i=0; i<nint; i++)
    {
        u[i+1]=u[i+1]-h[i]*h[i]/u[i];
        v[i+1]=v[i+1]-h[i]*v[i]/u[i];
    }

//Calculo de c[k] y d[k]
//Sustitucion hacia atras
```

```
    c[nint]=v[nint]/u[nint];
    for( int k=nint-1; k>=0; k--)
    {
        c[k]=(v[k]-h[k]*c[k+1])/u[k];
        d[k]=(c[k+1]-c[k])/(3*h[k]);
    }

//Calculo de b[k]
//Valores especiales en x0
    b[0]=fp0;
    for( int k=0; k<nint-1; k++)  b[k+1]=b[k]+(c[k]+c[k+1])*h[k];

//Impresion de coeficientes
    cout<< "     a     "<<"    b     "<<"     c     "<< "     d     "<<endl;
    for(int i=0; i<nint; i++) cout<<setprecision(4)<<setw(8)<<a[i]<<"  "<<
            setw(8)<<b[i]<<"  "<<setw(8)<<c[i]<<"  "<<setw(8)<<d[i]<<endl;
    return 0;
}


template <class T> T
eval_splines(T xx, const Vector<T> &x, const  Vector<T> &a,
             const Vector<T> &b, const Vector<T> &c, const  Vector<T> &d)
{
//buscar indice k
    int n=x.size();
    int k=0;
    int kinf=0;
    int ksup=n-1;
    while(ksup-kinf>1)
    {
        k=(ksup+kinf)/2;
        if (x[k]>xx) ksup=k;
        else kinf=k;
    }
    k=kinf;
//   cout<<"xx= "<<xx<<" k= "<<k;
    xx=xx-x[k];
    return a[k]+ xx*( b[k]+ xx*( c[k] +d[k]*xx ) );
}


int main()
```

```
{
    int nd;
    cout<< "Entrar numero de nodos"<<endl;
    cin>>nd;

//Construccion de vectores de nodos
    Vector<double> x(nd);

    cout<<"Entrar valores de los nodos"<<endl;
    for (int i=0; i<nd; i++) cin>>x[i];

    double x0=x[0], x1=x[nd-1];

//Eleccion de la funcion a interpolar
    int ft=-1;
    while((ft!=0)&&(ft!=1)&&(ft!=2)&&(ft!=3))
    {
      cout<<"Entrar funcion: 0- Runge   1- Bernstein  2- sin(x)"
        <<"   3- user "<<endl;
      cin>>ft;
    }

//Calculo de las ordenadas de interpolacion y de los coeficientes del
//polinomio
    Vector < double >  y(nd);
    for (int i=0; i< nd; i++) y[i]=fun(x[i],ft);

//Derivadas en los extremos
    double  fp0,fp1;

//Calculo de las derivadas en los extremos
//Se puede fijar el error de la derivada por extrapolacion
    double   hhh=0.00001;
    fp0= (fun(x0+hhh,ft)-fun(x0-hhh,ft))/(2*hhh);
    fp1=(fun(x1+hhh,ft)-fun(x1-hhh,ft))/(2*hhh);

    int nint= nd - 1;
    Vector<double> b(nint), c(nd), d(nint);

//Impresion de los nodos
    cout<<"      x        y"<<endl;
    for(int i=0;i<nd;i++)
        cout<<setprecision(4)<<setw(8)<<x[i]<<"  "<<setw(8)<<y[i]<<endl;
```

```
    cout<<endl<<endl;

//Splines naturales

//Calculo de los coeficientes
    Vector < double >  a=y;
    splines<double>(x,y,b,c,d);

//evaluacion del polinomio y funcion en los puntos de dibujo (200)
    int n=200;
    double hh=(x1-x0)/n;
    n++;
    Vector <double> xx(n), yy(n) ,ff(n);
    cout<<"   x  " <<  "        f(x)  "<< "     splin(x)" <<endl;
    for (int i = 0; i <n; i++)
    {
        xx[i]=x0+ i*hh;
        yy[i] = eval_splines<double>(xx[i], x, y, b,c,d);
        ff[i] = fun(xx[i],ft);
        if(!(i-(i/5)*5))
            cout<<setw(6)<<xx[i]<<"   "<<setprecision(8)<<
                setw(10)<<ff[i]<<"   "<<setw(8)<<yy[i]<<endl;
    }
    cout<<endl;

//Dibujo del polinomio y la funcion
    Gnuplot g1 = Gnuplot();
// Seleccionamos escala logaritmica para ciertas funciones
//   if( (ft==0)||(ft==1))  g1.cmd("set logscale y");
// if( (ft==0))  g1.cmd("set logscale y");
    g1.set_style("lines");
    g1.plot_xy(xx,ff,"  ");
    g1.set_style("points");
    g1.plot_xy(xx,yy," splines naturales ");

//Splines sujetos
//Calculo de los coeficientes

    splines_sujetos<double>(x,y,b,c,d, fp0,fp1);

    cout<<endl<<"   x  " <<  "        f(x)  "<< "     splin(x)" <<endl;
    for (int i = 0; i <n; i++)
    {
```

```
        xx[i]=x0+ i*hh;
        yy[i] = eval_splines<double>(xx[i], x, y, b,c,d);
        ff[i] = fun(xx[i],ft);
        if(!(i-(i/5)*5))
            cout<<setw(6)<<xx[i]<<"   "<<setprecision(8)<<
                  setw(10)<<ff[i]<<"   "<<setw(8)<<yy[i]<<endl;
    }

//Dibujo del polinomio y la funcion
    Gnuplot g2 = Gnuplot();
// Seleccionamos escala logaritmica para ciertas funciones
//   if( (ft==0)||(ft==1))  g1.cmd("set logscale y");
// if( (ft==0))  g1.cmd("set logscale y");
    g2.set_style("lines");
    g2.plot_xy(xx,ff," ");
    g2.set_style("points");
    g2.plot_xy(xx,yy,"splines sujetos ");

//Eleccion del tiempo de permanencia de la grafica en pantalla

    int SLEEP_LGTH=15;
    sleep(SLEEP_LGTH);
    return 0;
}
```

6.5. Ejercicios a presentar como memoria

1. Compilad y ejecutad el programa `interpol.cpp` para las funciones de Runge, Berstein y sin(x), con órdenes del polinomio interpolador de 5, 10 y 20 puntos, en el intervalo [-1,1], tomando los nodos tanto espaciados uniformemente, como en los ceros de los polinomios de Chebychev (En total 3x3x2=18 ejecuciones). Describir brevemente una síntesis del conjunto de los resultado. (10 líneas). Adjuntad en la memoria únicamente, aparte de las 10 líneas, las tres gráficas que consideréis más ilustrativas. Los comentarios deben de ser realmente **personales**, incluso si se ha hecho la práctica compartiendo ordenador,

2. Interpolad la función $x\exp(x)$ en 10 puntos igualmente espaciados, en el intervalo [0,2] (polinomio de grado 9). Representad el resultado gráficamente en un intervalo mayor que [0,2](delta > 0, se deja a vuestra opción el tamaño, si probáis para varios tamaños distintos, seguro que tendréis algo que opinar) y comentad el resultado. Entregad sólo la gráfica que ilustre lo mejor posible vuestros comentarios.

3. Repetid el ejercicio anterior para la función $f(x) = \arctan(x)$ en el intervalo [-1,1]. Compilad y ejecutad `interpol1.cpp` poniendo como puntos adicionales de cálculo del polino-

mio interpolador $x = 0{,}3$ y $x = 2$. ¿Cúal es el polinomio de orden más bajo que consigue una precisión de 10^{-5} en $x = 0{,}3$? Si se no se conociese la forma analítica de la función y solo se dispusiese de puntos de interpolación, una forma de estimar el error en un punto, es observar la diferencia entre polinomios de orden consecutivo. Con este criterio ¿Para que orden del polinomio podemos estimar que se ha alcanzado una precisión de 10^{-5} en $x = 0{,}3$? Comentad el comportamiento para $x = 2$. Presentad aquí solo los resultados del programa.

4. Escribid un programa que lea un fichero de datos producido por `gendat.cpp` (unos 10 o 15 puntos) y calcule el polinomio interpolador en 100 puntos entre el primer y último dato, utilizando la función
   `T Neville (const Vector <T >& x, const Vector <T >y, T & xx)`
   y los dibuje como línea continua mediante la librería `gnuplot_i++,` junto con los datos iniciales que se dibujarán como puntos discretos. Presentad el programa, un conjunto de datos y sus resultados correspondientes.

5. Interpolad la función de Runge en el intervalo [-1, 1] con 5 y 10 nodos igualmente espaciados, mediante splines. Comparad con el ejercicio 1. ¿Cúal es la conclusión? ¿Que tipo de splines funciona mejor, libres o sujetos? Comentad el resultado. En la memoria en papel presentad sólo las gráficas, aparte de vuestro comentario **personal.**
   Nota: En esta práctica es muy fácil generar una cantidad impresionante de gráficas y resultados. Limitaros a presentar únicamento lo especificado en cada apartado para que la memoria sea legible fácilmente y ocupe un tamaño en papel razonable.