Manual para la Clase Polinomio

Descripción

El objetivo de la plantilla de la clase *Polinomio* -codificada en C++- es la creación, evaluación y derivación de polinomios ($p \in P_{m-1}(\mathbb{R}^d)$) requeridos en la interpolación con funciones de base radial y en la solución numérica de ecuaciones en derivadas parciales. Al ser una plantilla (template) podemos instanciar el objeto Polinomio con distintos tipos de datos, por ejemplo: float, double, o algún tipo de dato definido por el usuario. Para evitar conflictos con los nombres de las clases, definimos el espacio de nombre *pol*.

Las característica principales de la clase *Polinomio* son:

- Datos en $\mathbb{R}^d$. !!Funciona para datos en cualquier dimensión, esta es la diferencia principal con las versiones anteriores.
- Crea polinomios de cualquier grado.
- Cálculo de las derivadas de cualquier orden y sobre cualquier variable.
- Operaciones binarias entre polinomios: suma, resta y multiplicación por un escalar.

Código Fuente: polinomio_v_1.9.cpp

Desarrollado: José Antonio Muñoz Gómez

Sugerencias: jose.munoz@cucsur.udg.mx

El presente manual forma parte del proyecto "Métodos de Funciones Radiales para la Solución de Ecuaciones en Derivadas Parciales" www.dci.dgsca.unam.mx/pderbf, coordinado por Pédro González-Casanova.

Funciones de la Clase Polinomio

En la carpeta de *examples* se muestran 22 ejemplos documentos de como emplear la clase Polinomio. La declaración de un polinomio se realiza mediante la instrucción

Polinomio<T> p;

donde T es un tipo de dato en C o en C++, por ejemplo T puede ser un float o double. Se recomienda utilizar

Polinomio<double> p;

En lo subsecuente se empleará a T para indicar el tipo de dato empleado.

make(d, m)

Descripción: crea internamente un polinomio $p \in P^d_{m-1}$, el polinomio internamete se almacena como $p = \{p_1, p_2, \ldots, p_M\}$. Posteriormente se puede evaluar u obtener las derivadas de p.

Parámetros de Entrada

d	ENTERO que especifica la dimensión del espacio.
m	ENTERO que especifica el orden m del polinomio.

deriva(parcial, orden)

Descripción: deriva el polinomio $p(x)$ creado con *make*.

Parámetros de Entrada

parcial	ENTERO vector con los datos, longitud $d \cdot n$.
	STRING vector con los datos, longitud $d \cdot n$.
orden	ENTERO que especifica el orden de derivación.

Ejemplos:

Para obtener la primera derivada del polinomio con respecto de la variable x para datos en una dimensión se puede realizar:

deriva("x",1) o deriva(1,1)

Para obtener la segunda derivada del polinomio con respecto de la variable y para datos en dos dimensiones $\bar{x} = (x, y)$ se puede realizar:

deriva("y",1) o deriva(2,1)

Para obtener la primera derivada del polinomio con respecto de la variables z para datos en tres dimensiones $\bar{x} = (x, y, z)$ se puede realizar:

deriva("z",1) o deriva(3,1)

Para datos definidos en $d > 3$, se debe utilizar como primer argumento un entero.

eval(x, d, px, dim-px)

Descripción: evalúa cada elemento de la base del polinomio creado en un punto $x \in \mathbb{R}^d$. Recordando que $p = \{p_1, p_2, \ldots, p_M\}$, entonces la evaluación de x en p corresponde a $p(x) = \{p_1(x), p_2(x), \ldots, p_M(x)\}$. Si el polinomio fue previamente derivado entonces evalua la derivada del polinomio en dicho punto.

Parámetros de Entrada

x	T* vector con un elemento $x \in \mathbb{R}^d$, x tiene longitud d.
d	ENTERO que especifica la dimensión del espacio.
dim-px	ENTERO que especifica el número de elementos del polinomio.

Parámetros de Salida

px	T* vector de longitud dim-px.

La longitud de px depende de los parámetros m, d utilizados en la función *make*. Para conocer el valor de dim-px se requiere utilizar la función *get_M*, la cual regresa el número de elementos que tiene el polinomio creado.

P ← build(x, n, d)

Descripción: crea una matriz P de dimensiones $n \times M$ la cual contiene por renglón la evaluación del polinomio p en un punto x (ver función *eval*). Si el polinomio fue previamente derivado entonces la matriz P contiene la evaluación del polinomio derivado evaluado en dichos puntos.

Parámetros de Entrada

x	DOUBLE vector con los datos, longitud $d * n$.
n	ENTERO que especifica el número de elementos del vecto x.
d	ENTERO que especifica la dimensión del espacio.

Parámetros de Salida

P	T** matriz de dimensiones $n \times M$.

La función *build* es requerida para construir el polinomio en la matriz de Gramm, así como la submatriz -corresponde al polinomio derivado- requerida en el esquema de colocación asimétrico de E. J. Kansa.

build(x, n, d, P, k, m, op)

Descripción: evalua el polinomio p en los puntos x, el resultado lo almacena en un vector de datos P. Si el polinomio fue previamente derivado entonces P contendra la evaluación del polinomio derivado evaluado en dichos puntos. En esta función no se reserva la memoria para P, esta debe ser previamente creada.

Parámetros de Entrada

x	T* vector con los datos, longitud $d \cdot n$.
n	ENTERO que especifica el número de elementos del vecto x.
d	ENTERO que especifica la dimensión del espacio.
k	ENTERO que especifica el número de renglones de P, k debe coincidir con n.
m	ENTERO que especifica el número de columnas de P, m debe coincidir con dim-px.
op	ENTERO si op=0 se almacenan los datos en orden ROW-MAJOR, si op=1 se almacenan los datos en orden COL-MAJOR como en Fortran. Si no se especifica el parámetro op, este vale 0.

Parámetros de Salida

P	T* vector de longitud $n \cdot m$. Requiere que previamente se reserve la memoria para P.

La función *build* es requerida para construir el polinomio en la matriz de Gramm, así como la submatriz -corresponde al polinomio derivado- requerida en el esquema de colocación asimétrico de E. J. Kansa.

Matrix<T> ← build_tnt(x, n, d)

Descripción: cuando se compila con la libreria Template Numerical ToolKit, ver README, se crea una matriz P de dimensiones $n \times M$ la cual contiene por renglón la evaluación del polinomio p en un punto x (ver función *eval*). Si el polinomio fue previamente derivado entonces la matriz P contiene la evaluación del polinomio derivado evaluado en dichos puntos. Los parámetros de entrada son similares a la primera función *build*.

+, -

Descripción: sean $p, q \in P_{m-1}(\mathbb{R}^d)$, la clase Polinomio tiene métodos públicos para poder realizar las siguientes tres operaciones:

$$p + q, \qquad p - q, \qquad \alpha \cdot p$$

con $\alpha \in \mathbb{R}$. Las tres operaciones anteriores son requeridas cuando trabajamos con ecuaciones en derivadas parciales.

int get_m()

Descripción: obtiene el grado del polinomio m utilizada en la construcción del polinomio con la función *make*.

int get_d()

Descripción: obtiene la dimensión del espacio d utilizada en la construcción del polinomio con la función *make*.

int get_M()

Descripción: obtiene el número de elementos M que conforman la base del polinomio: $p(x) = \{p_1(x), p_2(x), \ldots, p_M(x)\}$. El valor de M se determina internamente en la función *make* y es calculado como:

$$M = \begin{pmatrix} d+m-1 \\ m-1 \end{pmatrix}$$

string version()

Descripción: obtiene la versión de la clase Polinomio.

Ejemplos

El siguiente ejemplo en 1d contenido en /examples/1d/eje_9.cpp muestra las distintas formas de construir la matriz P. Para compilar el programa sólo realize: c++ eje_9.cpp, el archivo compilado será a.out, el cual arroja de salida:

```
Calculo de la matriz P con   p(x) = 1 + x + x*x

P =
    1.0 1.1 1.2
    1.0 1.2 1.4
    1.0 1.3 1.7

P =
    1.0  1.1  1.2  1.0  1.2  1.4  1.0  1.3  1.7

P =
    1.0  1.0  1.0  1.1  1.2  1.3  1.2  1.4  1.7
```

Para esclarecer detalles, se muestra el programa utilizado.

```
int main(void)
{
 Polinomio<double> p;
 int       n,m,d;
 double    x[3];
 double    px[2];
 int       dim_px;
 double    **P1;
 double    *P2;
 double    *P3;


//muestro algunos datos
  cout<<endl;
  cout<<"Calculo de la matriz P con   p(x) = 1 + x + x*x"<<endl;
  cout<<endl;

//defino la dimension del espacio (datos en una dimension) y orden del polinomio
  d=1; m=3;

//construyo el polinomio p(x) = 1 + x + x*x + x*x*x
  p.make(d,m);

//obtengo el numero de elmentos del polinomio
  dim_px = p.get_M();

//defino los puntos a evaluar
   n = 3;
   x[0] =  1.1;  x[1] =  1.2;  x[2] =  1.3;

//construyo la matriz con las evaluaciones del polinomio en los nodos
```

```
     P1 = p.build( x, n, d);

   //muestro la matriz
     show_mat(P1,  n, dim_px);
     cout<<endl;

//construyo la matriz con las evaluaciones del polinomio en los nodos
//empleo ROW_MAJOR
 P2 = new double[n*dim_px];
 p.build( x, n, d, P2, n, dim_px);

//muestro la matriz
  show_mat(P2,n,dim_px);
  cout<<endl;

//construyo la matriz con las evaluaciones del polinomio en los nodos
//empleo COL_MAJOR
 P3 = new double[n*dim_px];
 p.build( x, n, d, P3,n,dim_px,1);

//muestro la matriz
 show_mat(P3,n,dim_px);
 cout<<endl;

 return 0;
}
```