Evaluación computacional para calcular los polinomios de Legendre de primera clase

Computational evaluation to compute first kind Legendre polynomials

César Julio Bustacara Medina, MsC.
Pontificia Universidad Javeriana, Colombia
Departamento de Ingeniería de Sistemas
cbustaca@javeriana.edu.co

Recibido para revisión 15 de abril de 2009, aceptado 4 de junio de 2010, versión final 30 de junio de 2010

Resumen— **Los Polinomios de Legendre son uno de los ejemplos más importantes de los Polinomios Ortogonales, porque aparecen como soluciones en varios problemas clásicos, tales como: movimiento de los planetas, aplicaciones matemáticas, campos de conservación de energía, propagación de calor, propagación de ondas, propagación de ondas de partículas, propagación de señales telegráficas, reconstrucción de señales digitales, reconocimiento de patrones, clasificación de objetos, etc. Debido a la importancia que representa el cálculo de los Polinomios de Legendre en el reconocimiento de patrones, ya sea en dos dimensiones (2D) o en tres dimensiones (3D), en este artículo se presenta un análisis computacional de la eficiencia del cálculo de los Polinomios de Legendre de primera clase. La implementación de los métodos para calcular los polinomios se realizó en dos lenguajes de programación; C++ y Java.**

Palabras Clave—**Polinomios de Legendre, Polinomios Ortogonales, C++, Java.**

Abstract—**Legendre Polynomials are one of the most important examples of orthogonal polynomials, because they appear as solutions in several classic problems such as planets movement, math applications, fields of energy conservation, heat propagation, wave propagation, particle wave propagation, spread telegraph signals, digital signals reconstruction, pattern recognition, object classification, etc. Due to the importance of computation of the Legendre polynomials in pattern recognition, either in two dimensions (2D) or three dimensions (3D), this paper presents a computational analysis of efficiency, to compute first class Legendre polynomials. The implementation of methods for computing polynomials was performed in two programming languages, C + + and Java.**

Keywords— **Legendre Polynomials, Orthogonal Polynomials, C++, Java.**

I. INTRODUCCIÓN

El problema fundamental en el reconocimiento de objetos, ya sea en 1D, 2D o 3D, es la descripción de los patrones de entrenamiento y luego la generación de los descriptores para los objetos a reconocer. El proceso de descripción de cualquier señal ya sea discreta o continua se puede realizar a través de polinomios ortogonales. Sin embargo, existen aun problemas computacionales para calcular los polinomios ortogonales, lo cual implica una baja utilización de los mismos porque reducen la eficiencia del proceso de descripción y a la vez de reconocimiento [1] [2] [6] [7] [15] [17]. En la actualidad, existen varias familias de polinomios ortogonales dentro de los cuales se encuentran: los polinomios de Legendre, Chevichev, Laguerre, Bessel, etc. Estos polinomios están definidos en un intervalo [a, b]. Para el caso particular de los polinomios de Legendre, las soluciones son ortogonales en el intervalo [-1,1].

Los polinomios de Legendre tienen diversas aplicaciones en la solución de problemas tales como: movimiento de planetas (ecuación de Kepler), resolución de modelos físicos con ecuaciones diferenciales en derivadas parciales tanto en coordenadas cartesianas como polares (campos de conservación, propagación de calor y propagación de señales), reconstrucción de imágenes digitales, termodinámica (descripción del comportamiento de las sustancias), descripción y reconocimiento de objetos, codificación y reconstrucción de

señales, y en diversos problemas físicos (gravitación, electrostática, etc.) donde aparecen fuerzas que dependen del inverso de la distancia entre dos cuerpos [2] [6] [7].

En este artículo se revisa en primera instancia la formulación matemática de los polinomios de Legendre de primera clase (una sola variable), posteriormente se definen los diferentes métodos para obtener los polinomios, luego se describe las técnicas utilizadas para su implementación computacional y finalmente se realiza un análisis de las pruebas y se analizan los resultados obtenidos [2] [16].

II. POLINOMIOS DE LEGENDRE

Los polinomios de Legendre surgen como una alternativa para solucionar la ecuación diferencial de Legendre, que en su forma canónica se define como [2] [9]:

$$(1 - x^2)\, y'' - 2\, x\, y' + \lambda\, y = 0 \qquad (1)$$

Cuya solución general es la combinación lineal de dos soluciones linealmente independientes:

$$y(x) = A\, y_1(x) + B\, y_2(x) \qquad (2)$$

Como caso particular de estas soluciones si $\lambda = \upsilon(\upsilon+1)$ con $\upsilon = n \in \mathbb{N}$ una de dichas soluciones es un Polinomio de Legendre de orden n. En este caso la solución general toma la forma [9]:

$$y(x) = A\, P_n(x) + B\, Q_n(x) \qquad (3)$$

Los polinomios de Legendre $P_n(x)$ y $Q_n(x)$ que aparecen en la ecuación (3) son soluciones de la ecuación diferencial de Legendre (1-2). Los primeros seis polinomios de Legendre $P_n(x)$ están representados como se muestra en la Figura 1.

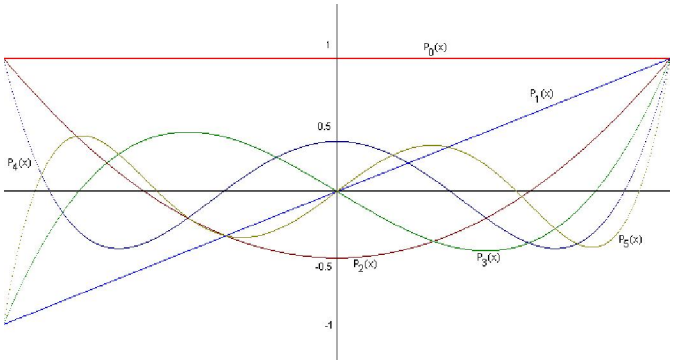

Figura 1. Representación de los primeros seis Polinomios de Legendre.

La ecuación de los polinomios de Legendre usando la representación de Rodríguez está dada por la ecuación (4) [2] [9] [13] [16].

$$P_n(x) = \frac{1}{2^n n!}\left(\frac{d}{dx}\right)^n (x^2 - 1)^n, \text{ donde } x \in [-1, 1] \qquad (4)$$

Como se mencionó, los polinomios de Legendre son ortogonales en el intervalo [-1, 1], lo cual permite que sean utilizados como una combinación de series infinitas de funciones linealmente independientes. Lo anterior implica que pueden ser utilizados en la proyección y reconstrucción de señales, es decir, son un mecanismo para encontrar la representación de una señal. La ecuación (5) muestra la notación de una función *f*, que es multiplicada por los polinomios de Legendre para obtener los coeficientes *k* que la representan [2] [9] [16] [18].

$$\int f(x) P_n(x)\, dx = k_n \qquad (5)$$

Este mecanismo se puede ver en la Figura 2(a), en la cual se multiplica cada uno de los polinomios de Legendre $P_n(x)$ por la función f y se encuentra cada uno de los valores k_n.

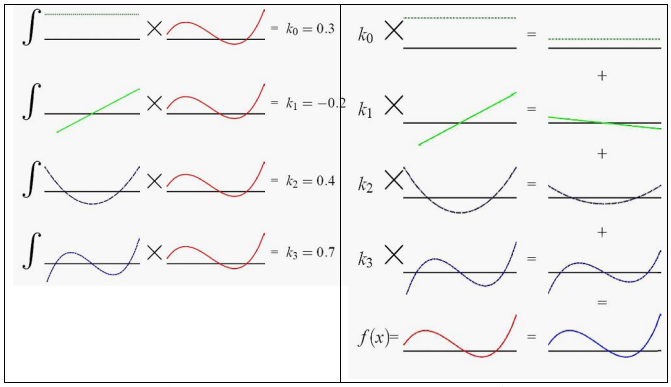

Figura 2. Proyección y reconstrucción de una función *f* usando los polinomios de Legendre (adaptada de [18])

A partir de los valores k_n se puede reconstruir la función f. La ecuación (6) muestra el proceso de reconstrucción [2] [9] [16] [18].

$$f(x) = \sum_{n=0}^{\infty} k_n P_n \qquad (6)$$

La Figura 2(b) muestra el proceso de reconstrucción de la función *f* a partir de los coeficientes k_n y los polinomios de Legendre $P_n(x)$.

III. APROXIMACIONES NUMÉRICAS DE LOS POLINOMIOS DE LEGENDRE

Computacionalmente existen varias alternativas para realizar el cálculo de los polinomios de Legendre [1] [2] [6] [9] [10] [12] [13] [14] [16], las cuales fueron consideradas y se implementaron para realizar su correspondiente comparación. La lista de métodos a implementar se enumera en la Tabla 1.

Tabla 1. Lista de métodos para calcular los polinomios de Legendre

Métodos	Ecuación
1	$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{\lfloor n/2 \rfloor} \frac{(-1)^k(2n-2k)!}{k!\,(n-k)!\,(n-2k)!}x^{n-2k}$ Si n es par existe una singularidad cuando x=0 y (n-2k)=0, es decir, 0^0 no está definida.
2	$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \binom{n}{k}\binom{2n-2k}{n}x^{n-2k}$ Si n es par existe una singularidad cuando x=0 y (n-2k)=0, es decir, 0^0 no está definida.
3	$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{n} \binom{n}{k}(x-1)^{n-k}(x+1)^k$ Existe una singularidad cuando x=-1 y k=0, es decir, 0^0 no está definida.
4	$P_{n+1}(x) = \frac{(2n+1)xP_n(x) - nP_{n-1}(x)}{n+1}$ Donde, $P_0(x) = 1$ y $P_1(x) = x$
5	$P_{n+1}(x) = 2xP_n(x) - P_{n-1}(x) - \frac{1}{n+1}(xP_n(x) - P_{n-1}(x))$ Donde, $P_0(x) = 1$ y $P_1(x) = x$

IV. TÉCNICAS DE IMPLEMENTACIÓN

Se definieron algoritmos tanto en lenguaje C++ como Java para cada uno de los métodos de aproximación numérica de los polinomios de Legendre [3] [4]. Para el caso especifico del lenguaje C++ se utilizó el tipo de dato *long double* que posee una longitud de 80 bits, que es el tipo de datos mayor tamaño para computación científica. En el caso del lenguaje Java se utilizó el tipo de dato *double*. Lo anterior con el fin de aumentar el número de polinomios que se puedan calcular, debido a la restricción de los cálculos de los factoriales que están incluidos en las ecuaciones de los tres primeros métodos.

A continuación se lista cada uno de los métodos de cálculo con su correspondiente implementación en cada uno de los dos lenguajes seleccionados.

Método 1: $P_n(x) = \frac{1}{2^n}\sum_{k=0}^{\lfloor n/2 \rfloor} \frac{(-1)^k(2n-2k)!}{k!\,(n-k)!\,(n-2k)!}x^{n-2k}$

Lenguaje C++

```
long double legendrePoly1(int n, double x){
   double max = 0;
   long double pn = 0;
   long double num, den;

   long double cte = 1.0/(pow(2.0, n));

   if (n>0)
      max = floor(n/2);

   for (int k=0; k<=max; k++){
      num = pow(-1.0, k)*factorial((2*n)-(2*k));
      den = factorial(k)*factorial(n-k);
      if ((n-(2*k)) != 0){
         num = num*pow(x, n-(2*k));
         den = den*factorial(n-(2*k));
      }
      pn = pn  + (num/den);
   }
   pn = cte*pn;
   return pn;
}
```

Lenguaje Java

```
public double legendrePoly1(int n, double x){
   double max = 0;
   double pn = 0;
   double num, den;

   double cte = 1.0/(Math.pow(2.0, n));

   if (n>0)
      max = Math.floor(n/2);

   for (int k=0; k<=max; k++){
      num = Math.pow(-1.0, k)*factorial((2*n)-(2*k));
      den = factorial(k)*factorial(n-k);
      if ((n-(2*k)) != 0){
         num = num*Math.pow(x, n-(2*k));
         den = den*factorial(n-(2*k));
      }
      pn = pn  + (num/den);
   }
   pn = cte*pn;
   return pn;
}
```

Método 2:

$$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{\lfloor n/2 \rfloor}(-1)^k \binom{n}{k}\binom{2n-2k}{n}x^{n-2k}$$

Lenguaje C++	Lenguaje Java

long double legendrePoly2(int n, double x){
double max = 0;
long double pn = 0;
long double val;
double p1;
 	
long double cte = 1.0/(pow(2.0, n));
 	
if (n>0)
max = floor(n/2);
 	
for (int k=0; k<=max; k++){
p1 = (2*n) - (2*k);
val = pow(-1.0, k)*binomial(n, k)*binomial(p1, n);
if ((n-(2*k)) != 0){
val = val*pow(x, n-(2*k));
}
pn = pn + val;
}
pn = cte*pn;
return pn;
}
	public double legendrePoly2(int n, double x){
	double max = 0;
	double pn = 0;
	double val;
	double p1;
	
	double cte = 1.0/(Math.pow(2.0, n));
	
	if (n>0)
	max = Math.floor(n/2);
	
	for (int k=0; k<=max; k++){
	p1 = (2*n) - (2*k);
	val = Math.pow(-1.0, k)*binomial(n, k)*binomial(p1, n);
	if ((n-(2*k)) != 0){
	val = val*Math.pow(x, n-(2*k));
	}
	pn = pn + val;
	}
	pn = cte*pn;
	return pn;
	}

Método 3:

$$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{n}\binom{n}{k}(x-1)^{n-k}(x+1)^k$$

Lenguaje C++	Lenguaje Java

long double legendrePoly3(int n, double x){
double max = 0;
long double pn = 0;
long double val;
 	
long double cte = 1.0/(pow(2.0, n));
 	
for (int k=0; k<=n; k++){
val = pow(binomial(n, k), 2);
if ((n-k)!= 0){
val = val*pow(x-1, n-k);
}
if (k!=0){
val = val*pow(x+1, k);
}
pn = pn + val;
}
pn = cte*pn;
return pn;
}
	public double legendrePoly3(int n, double x){
	double max = 0;
	double pn = 0;
	double val;
	
	double cte = 1.0/(Math.pow(2.0, n));
	
	for (int k=0; k<=n; k++){
	val = Math.pow(binomial(n, k), 2);
	if ((n-k)!= 0){
	val = val*Math.pow(x-1, n-k);
	}
	if (k!=0){
	val = val*Math.pow(x+1, k);
	}
	pn = pn + val;
	}
	pn = cte*pn;
	return pn;
	}

Método 4:

$$P_{n+1}(x) = \frac{(2n+1)xP_n(x) - nP_{n-1}(x)}{n+1}$$

donde, $P_0(x) = 1$ y $P_1(x) = x$

Lenguaje C++	Lenguaje Java

long double legendrePoly4(int n, double x){
double p0 = 1;
double p1 = x;
 	
long double pn = 0;
 	
if (n==0)
return p0;
 	
if (n==1)
return p1;
 	
for (int k=1; k<n; k++){
pn = (((2*k+1)*x*p1) - (k*p0))/(k+1);
p0 = p1;
p1 = pn;
}
return pn;
}
	public double legendrePoly4(int n, double x){
	double p0 = 1;
	double p1 = x;
	
	double pn = 0;
	
	if (n==0)
	return p0;
	
	if (n==1)
	return p1;
	
	for (int k=1; k<n; k++){
	pn = (((2*k+1)*x*p1) - (k*p0))/(k+1);
	p0 = p1;
	p1 = pn;
	}
	return pn;
	}

Método 5:

$$P_{n+1}(x) = 2xP_n(x) - P_{n-1}(x) - \frac{1}{n+1}(xP_n(x) - P_{n-1}(x))$$

donde, $P_0(x) = 1$ y $P_1(x) = x$

Lenguaje C++	Lenguaje Java

long double legendrePoly5(int n, double x){
long double p0 = 1.0;
long double p1 = x;
long double pn = 0;
long double pn1 = 0;
 	
if (n==0)
return p0;
 	
if (n==1)
return p1;
 	
for (int k=1; k<n; k++){
pn1 = x*p1;
pn = ((2*pn1) - p0) - (pn1 - p0)/(k+1);
p0 = p1;
p1 = pn;
}
return pn;
}
	public double legendrePoly5(int n, double x){
	double p0 = 1.0;
	double p1 = x;
	double pn = 0;
	double pn1 = 0;
	
	if (n==0)
	return p0;
	
	if (n==1)
	return p1;
	
	for (int k=1; k<n; k++){
	pn1 = x*p1;
	pn = ((2*pn1) - p0) - (pn1 - p0)/(k+1);
	p0 = p1;
	p1 = pn;
	}
	return pn;
	}

V. PRUEBAS Y RESULTADOS

Los polinomios de Legendre $P_n(x)$ fueron evaluados para los cinco métodos en cada uno de los lenguajes de programación y el número de valores de la variable x en el intervalo [-1, 1] fue de 752 muestras. Para medir el tiempo de computo requerido se realizaron 10.000 ejecuciones de cada uno de los algoritmos en un equipo con procesador quad core de 2.4 GHz y una memoria RAM de 2GB. El hecho de tener varias unidades de procesamiento implica que las operaciones de cómputo se dividieron entre ellas. Adicionalmente, se calcularon los primeros 45 polinomios $P_n(x)$, debido a que se presentaron inconvenientes en los tres primeros métodos. Esto se puede ver en las figuras 3, 4 y 5, en las cuales se muestra un problema de precisión en el cálculo de los polinomios $P_n(x)$ cuando n=57, n=58 y n=59 respectivamente.

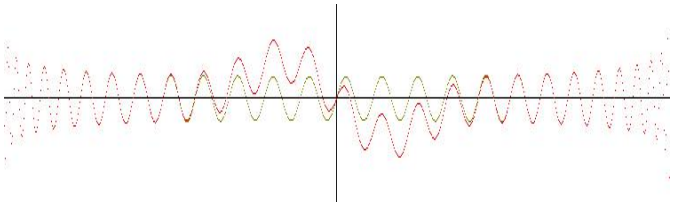

Figura 3. Cálculo del polinomio de Legendre cuando n=57

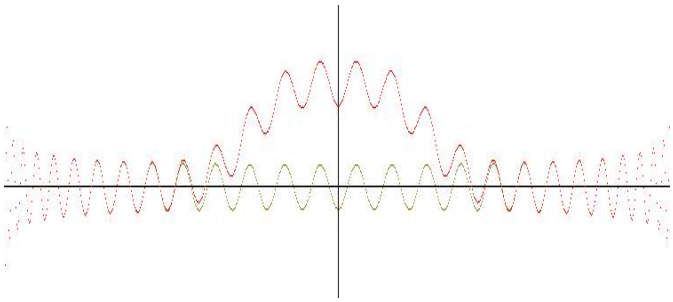

Figura 4. Cálculo del polinomio de Legendre cuando n=58

En la Figura 6 se muestra un análisis comparativo de los cinco métodos implementados en cada uno de los lenguajes. La coordenada *x* indica el número de polinomios *n* y la coordenada *y* indica el tiempo en milisegundos requerido para ejecutar el cálculo de los *n* polinomios de Legendre. De acuerdo a la figura 6 se puede deducir que los métodos 4 y 5 son mucho más eficientes que los demás métodos. Esto es evidente porque corresponden a ecuaciones de recurrencia que permiten reducir el uso de sumatorias y de cálculo de factoriales y binomiales. Adicionalmente, los métodos 1 y 2 son más eficientes que el método 3, debido a que se debe realizar la mitad de las operaciones para encontrar el valor del polinomio. Por otra parte se ve que en el caso particular del método 3, es mucho más eficiente el lenguaje Java que C++.

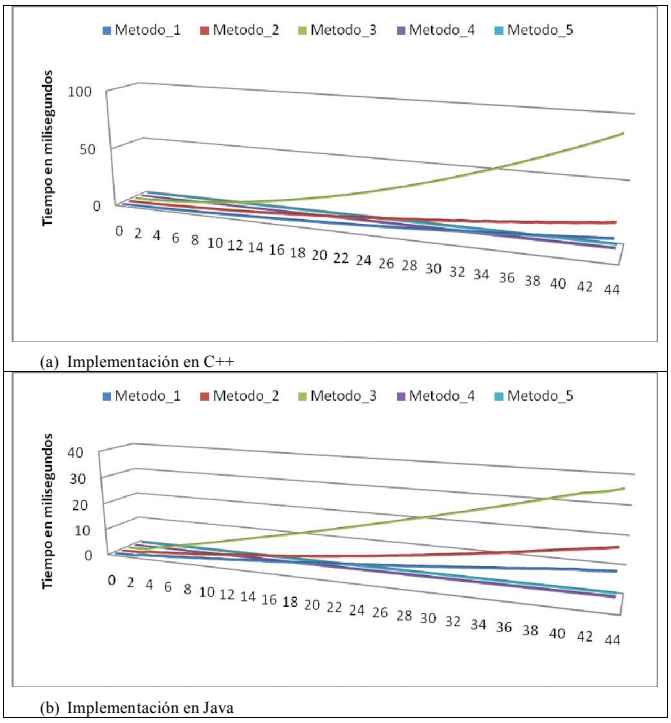

Figura 6. Comparación de los métodos para calcular los polinomios de Legendre

En la Figura 7 se realiza una comparación de los métodos 4 y 5 para cada uno de los lenguajes de implementación. En la columna izquierda (implementación C++) se puede ver que los tiempos de respuesta fluctúan entre los dos métodos, por lo cual los dos son validos para realizar el cálculo de los polinomios. Mientras que en la columna derecha (implementación en Java) se puede observar que el método 5 es más eficiente que el método 4 para realizar el cálculo de los polinomios de grado superior a n=10. Adicionalmente, se mantiene la tendencia de eficiencia de Java con respecto a C++, pero esto no concuerda con el supuesto de que C++ es más eficiente que Java [3] [4] [5] [11], por lo cual se realizaron pruebas con las diferentes formas de obtener los tiempos en C++, usando las estructuras *time_t*, *clock_t* y la función *gettimeofday*. El resultado obtenido fue el mismo, es decir, se presentó mayor eficiencia en Java.

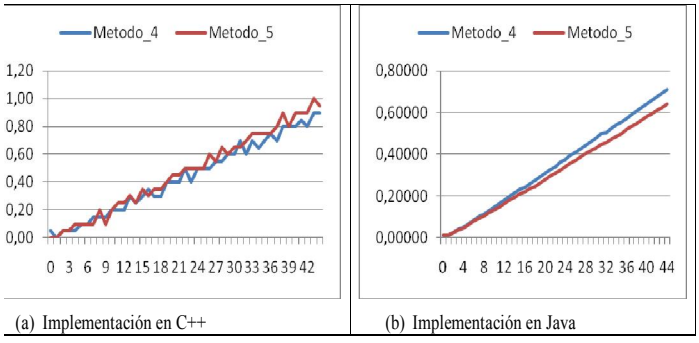

Figura 7. Comparación de los métodos 4 y 5 para calcular los polinomios de Legendre

VI. CONCLUSIONES Y TRABAJO FUTURO

Los algoritmos basados en recurrencias son mucho más eficientes que los basados en sumatorias, esto ocurre tanto en C++ como en Java. Es importante trabajar sobre la paridad de los polinomios para reducir el tiempo computacional, sobre todo cuando se vayan a utilizar en la descripción de señales.

Los métodos 4 y 5 no presentan mucha diferencia en el lenguaje C++, mientras que en lenguaje Java se aprecia un mejor rendimiento en el método 5 a medida que crece el valor de *n* del polinomio a calcular. Sería bueno explorar valores superiores y analizar el comportamiento de los métodos tanto en C++ como en Java.

Dentro del análisis de los polinomios se utilizaron únicamente coordenadas cartesianas, pero sería deseable extender la solución a coordenadas polares para verificar el impacto de las funciones coseno y seno dentro del rendimiento de los métodos.

Dentro del proceso de cálculo de los polinomios de Legendre se encontró inconveniente en los métodos 1, 2 y 3 en órdenes superiores de n, específicamente para n>46. Esto refleja una gran desventaja en el momento de encontrar vectores de descriptores de orden mayor a 46.

El rendimiento más bajo lo ofrece el método 3, puesto que la cota superior de la sumatoria es el doble de los métodos 1 y 2.

En el proceso de cómputo de los métodos 1, 2 y 3 existen singularidades en las funciones, lo cual implica un mayor tiempo de ejecución por la incorporación de condicionales.

REFERENCIAS

[1] Abramowitz M. and Stegun I., 1972. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, National Bureau of Standards Applied Mathematics Series – 55 P.

[2] Arfken G. B. and Weber H. J., 2005. Mathematical Methods for Physicists, Fourth Edition, Academic Press.

[3] Besset D., 2001. Object-Oriented Implementations of Numerical Methods, Morgan –Kaufmann Press.

[4] Boisvert R. F., Moreira J., Philippsen M. and Pozo R., 2000. Java and Numerical Computing, NIST.

[5] Bruckschlegel T., 2005. Microbenchmarking C++, C# and Java, En: Dr. Dobbs, June 17.

[6] Chow T., 2000. Mathematical Methods for Physicists: A concise introduction, Cambridge University Press.

[7] Davis J., 2004. Mathematical Modeling of Earth's Magnetic Field, Technical Note, Virginia Tech, Blacksburg.

[8] Guseinov I., 1996. On the evaluation of rotation coefficients for spherical harmonics using binomial coefficients, En: Journal of Molecular Structure, pp. 119-121.

[9] Hobson E.W., LL. D. and F.R.S., 1965. The theory of Spherical and Ellipsoidal Harmonics, Chelsea Publishing Company, New York.

[10] Huazhong S., Limin L., Xudong B., Wenxue Y. and Guoniu H., 2000. An Efficient Method for Computation of Legendre Moments, En: Graphical Models, pp. 237–262.

[11] Martin R. C., 1997. Java and C++: A critical comparison, Technical Note, Object Mentor.

[12] Mukundan R. and Ramakrishnan K., 1995. Fast Computation of Legendre and Zernike Moments, En: Pattern Recognition Vol. 28, No. 9, pp. 1433-1442.

[13] Poularikas A.D., 1999. Legendre Polynomilas, The Handbook of Formulas and Tables for Signal Processing, CRC Press LLC.

[14] Pew-Thian Y. and Raveendran P., 2005. An Efficient Method for the Computation of Legendre Moments, En: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 12.

[15] Schwaha P., Giani C., Heinzl R. and Selberherr S., 2007. Visualisation of Polynomials Used in Series Expansions, En: Proceedings of the 4th High-End Visualization Workshop, pp. 139 - 148.

[16] Sacerdoti J., 2002. Polinomios y Funciones de Legendre, Departamento de Matemática, Facultad de Ingeniería, Universidad de Buenos Aires.

[17] Simon X. L. and Miroslaw P., 1996. On Image Analysis by Moments, En: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 3.

[18] Schönefeld V., 2005. Spherical Harmonics, Computer Graphics and Multimedia Group, Technical Note. RWTH Aachen University, Germany.

Universidad Nacional de Colombia Sede Medellín
Facultad de Minas

120 años TRABAJO Y RECTITUD

Reseña Histórica

La Escuela Nacional de Minas fue fundada el 11 de abril de 1887, bajo la dirección del general Pedro Nel Ospina como rector y como Vice-rector Luís Tisnés, aunque el general Pedro Nel Ospina no se posesiono, elaboro con ayuda de su hermano Tulio los estatutos y reglamentos de la escuela, los cuales fueron una adaptación de los estatutos y reglamentos de la Escuela de Minas de California (Berkeley)los cuales fueron cambiando de acuerdo a las necesidades de cada década, en ellos se fomento una filosofía con valores cívicos, éticos y de orden por medio del estimulo y el ejemplo que comprometían el comportamiento del estudiante no solo dentro de la escuela sino fuera de ella, a demás se introdujeron hábitos de sobriedad, de economía y principios morales de honradez, honestidad y respeto.

En sus inicios contó con 22 alumnos matriculados, y luego de tres meses fue cerrada por la poca cantidad de estudiantes, fue reabierta un año después, el 2 de enero de 1888, bajo la rectoría de Tulio Ospina V, esta vez contó con 27 alumnos matriculados y con un plan de estudios de 4 años de un mejor control de los programas curriculares y adaptarlos a nuevas condiciones adelantándose a las necesidades futuras de la educación y asegurando así un buen desempeño de los futuros profesionales.

En 1906 la Escuela Nacional de Minas se anexo a la universidad de Antioquia, a la que perteneció durante cinco años más, en 1911 paso a ser de nuevo una entidad independiente.

En 1940 la institución fue incorporada a la Universidad Nacional y continúo con el nombre de Escuela Nacional de Minas, ese mismo año comenzó la construcción de la actual sede, la cual fue inaugurada el 19 de diciembre de 1944, en el marco del primer Congreso Nacional de Ingenieros.

Entre 1941 y 1950 se crean las carreras de ingeniería geológica y petróleos y arquitectura, la cual se separo de la facultad de Minas en 1954, en1960 se crea la carrera de ingeniería administrativa, luego se crearon los programas de ingeniería industrial, ingeniería mecánica e ingeniería química y se separaron los programas de ingeniería geológica y petróleos en dos programas diferentes, actualmente la Facultad de Minas Administra 11 programas de pregrado en ingeniería, 17 de posgrado y cuatro doctorados.

La Facultad a lo largo de su existencia ha sido motora del desarrollo de la ciudad, del departamento y del país, a través de sus 12.000 egresados quienes han constituido la mayor parte del personal dirigente y técnico en las explotaciones mineras, las construcciones de distinto tipo, la infraestructura vial, los desarrollos hidroeléctricos, las obras de abastecimiento de agua, las obras sanitarias y la industria, así como en los planes de desarrollo físico, económico y social.