gobiernan el modelo. Utilizando técnicas estándares de resolución de ecuaciones, tales como eliminación de Gauss, podemos encontrar las soluciones de problemas en dos y tres dimensiones.

**Conclusiones**

Considerando lo expuesto, concluimos que el método BEM es más laborioso que el FEM, matemáticamente hablando. Las bases matemáticas son sumamente complejas, pero su amplia aplicación, su versatilidad y sus soluciones precisas hacen valioso el esfuerzo. Otra manera interesante de enfocar la derivación de la BIE seria utilizando la segunda identidad de Green como lo han hecho [1] y [3]. Pero el enfoque de modelaje de sistemas utilizando el método de pesos residuales resulta más extenso en cuanto a aplicación se refiere.

**Referencias**


1. Becker, A.A., **The Boundary Element Method in Engineering**, McGraw-Hill Book Company, Great Britain, 1992.
2. Brebbia, C.A. y Domínguez, J., **Boundary Elements: An Introductory Course**, 2da ed., Mc Graw-Hill, Great Britain, 1992.
3. Banerjee, P.K. y Butterfield, R., **The Boundary Element Methods in Engineering**, 2da ed., McGraw-Hill, Great Britain, 1993.
4. Roach, G.F., **Green´s Functions**, 2da ed., Cambridge University Press, 1989.


# Aritmética Racional vs Aritmética de Punto Flotante


Por: Anatoli Markelov Ph.D.


Es bien conocido que la aritmética de enteros es mucho más rápida que la aritmética de punto flotante. Pero la aritmética de enteros no puede alcanzar precisiones de cálculo necesarias en cualquier programa real serio.

Al mismo tiempo en programas de control automático en tiempo real el uso de la aritmética de punto flotante resuelve el problema de precisión, pero a cuenta de una desaceleración significante de cálculos.

Especialmente esta desaceleración afecta a la calidad de control para los sistemas automáticos los cuales tienen plantas de control con muy pequeños valores de constantes de tiempo (sistemas de control dinámicos rápidos) y en los cuales se implementan algoritmos de control complicados. En estos casos una velocidad baja de cálculos exige un aumento del periodo de muestreo con todas las consecuencias negativas para la calidad de funcionamiento de sistemas de control.

En el trabajo actual se propone usar en lugar de la aritmética de punto flotante la aritmética racional, que se basa en el uso de la aritmética de enteros para realizar cálculos con números representados por medio de fracciones racionales. De la teoría de los números reales es conocido que cualquier número real puede ser aproximado con una precisión deseada por medio de una razón de dos números enteros. Entonces se puede realizar cálculos muy rápidos con pares de números enteros (numerador - **n** y denominador - **d**) sin pérdidas de precisión. El error de los cálculos se limita solamente con un error de aproximación inicial de los números reales.

La implementación más eficiente del dicho método se basa en el uso de sobrecarga de operadores aritméticos en un lenguaje de programación que soporta la POO – programación orientada a objetos, por ejemplo C++. En este lenguaje al sobrecargar los operadores '+', '-', '*', '/' y otros obtenemos la posibilidad de usar expresiones aritméticas estándar para los pares de números enteros como si ellos fueran simples números de tipo primitivo, por ejemplo si $\mathbf{p_1}$ representa una fracción $\mathbf{(n_1,d_1)}$ y $\mathbf{p_2}$ representa una fracción $\mathbf{(n_2,d_2)}$, entonces para sumar estas dos fracciones nosotros podemos escribir la expresión siguiente:

$$\mathbf{p = p_1+p_2}$$

Análogamente se puede escribir cualquier fórmula aritmética con los operandos representados por medio de pares **(n,d)**.

En el programa presentado se realiza la aritmética racional que usa solamente cuatro operaciones aritméticas principales, pero se puede definir cualquier operación aritmética por medio de sobrecarga de operadores. Como ejemplo el programa calcula cíclicamente los valores de fracciones racionales según la fórmula **(p1+p2)*(p1-p2)/(p2+p3)** para cualquier conjunto de tres pares enteros **(n,d)** introducidos por usuario. Para salir del programa se necesita presionar la tecla ESC.

```
#include "iostream.h"
#include "math.h"
#include "conio.h"
const char ESC=0x1b;
class F_R
{     long n,d;
public:
      F_R(){n=0,d=0;}; F_R(long,long);
      void pr(F_R);
      void reducir(F_R*);
      void normalizar(F_R*);
      F_R operator+ (F_R);
      F_R operator- (F_R);
      F_R operator* (F_R);
      F_R operator/ (F_R);};
F_R::F_R(long n1, long d1) {n=n1; d=d1;}
void F_R::pr(F_R t) {cout << t.n << '\t' <<
t.d << endl;}
void F_R::reducir(F_R* t)
{     long a,b,r;
      if(t->n  >  t->d)  {a=t->n;  b=t->d;}
else {a=t->d; b=t->n;}
      while(b != 0) {r=a%b; a=b; b=r;}
      t->n /= labs(a); t->d /= labs(a);}
void F_R::normalizar(F_R* t) {if(t->d < 0)
{ t->n *= -1; t->d *= -1; }
F_R F_R::operator+ (F_R b)
{     F_R c;
      c.n=n*b.d+d*b.n;          c.d=d*b.d;
c.reducir(&c);  return c;}
F_R F_R::operator- (F_R b)
{     F_R c;
      c.n=n*b.d-d*b.n;          c.d=d*b.d;
c.reducir(&c);  return c;}
F_R F_R::operator* (F_R b)
{     F_R c;
      c.n=n*b.n;  c.d=d*b.d;  c.reducir(&c);
c.normalizar(&c); return c;}
F_R F_R::operator/ (F_R b)
{     F_R c;
      c.n=n*b.d;  c.d=d*b.n;  c.reducir(&c);
c.normalizar(&c); return c;}

int main(int argc, char* argv[])
{
      long u,v;
      char ch;
      do
      {
            F_R p;
            cout    <<    "1-ra    fraccion
(numerador denominador):" << endl;
            cin >> u >> v;
            F_R p1(u,v);
            cout    <<    "2-da    fraccion
(numerador denominador):" << endl;
            cin >> u >> v;
            F_R p2(u,v);
            cout    <<    "3-ra    fraccion
(numerador denominador):" << endl;
            cin >> u >> v;
            F_R p3(u,v);
            p=(p1+p2)*(p1-p2)/(p2+p3);

            p.pr(p);
      } while ((ch=getch()) != ESC);
      return 0;}
```