Implementación en Haskell de un algoritmo general para resolución de ecuaciones por métodos iterativos

Sylvia da Rosa y Angel Caffa
Instituto de Computación
Facultad de Ingeniería
Julio Herrera y Reissig 565
Montevideo - URUGUAY
tel.fax : 598 2 710469
email: (darosa angel)@fing.edu.uy

1997

Abstract

En este trabajo se diseña e implementa en el lenguaje funcional Haskell, *un algoritmo genérico* para solucionar ecuaciones no lineales y sistemas de ecuaciones lineales. El diseño del algoritmo se basa en que los métodos numéricos usados en la resolución de ecuaciones y sistemas, presentan el mismo esquema de iteración, lo cual permite *abstraer* el método, y obtener cada algoritmo de resolución por un método determinado, como *una instancia* del algoritmo general. Uno de los aspectos más detacables del uso de Haskell en la implementación, radica en la posibilidad de lograr el nivel de abstracción requerido, definiendo una *función sobre el dominio de los métodos iterativos* para representar el algoritmo genérico, donde cada instancia se implementa como una *aplicación* de la función. Interesa resaltar también, el poder que brinda para la implementación de funciones sobre objetos infinitos, la semántica perezosa de Haskell.

Palabras claves : lenguajes de programación, lenguajes funcionales perezosos, métodos numéricos, ecuaciones no lineales, sistemas de ecuaciones lineales.

Implementación en Haskell de un algoritmo general para resolución de ecuaciones por métodos iterativos

1 Introducción

En este trabajo se presenta el diseño y la implementación en el lenguaje Haskell [Rep92] de un algoritmo genérico para resolver ecuaciones no lineales y sistemas de ecuaciones lineales por medio de métodos numéricos iterativos.
El diseño del algoritmo se basa en el hecho de que los métodos numéricos usados en la resolución de ecuaciones y sistemas presentan el mismo esquema de iteración. Esto permite construir un algoritmo general en el cual se abstrae el *método* de resolución utilizado. El algoritmo puede ser aplicado a alguno de los métodos iterativos siguientes:

Para ecuaciones no lineales:	**Para sistemas de ecuaciones lineales:**
Método de Bisección	Método de Jacobi
Método de Newton-Raphson	Método de Gauss-Seidel
Método de la Secante	
Método de la Falsa Posición	

Haskell es un lenguaje puramente funcional perezoso que permite definir una una función sobre el dominio de los métodos iterativos para representar el algoritmo, con el nivel de abstracción deseado. Interesa resaltar también, el poder que brinda para la implementación de funciones sobre objetos infinitos, la semántica perezosa de Haskell.
Este trabajo se enmarca dentro de otro más amplio cuyo objetivo consiste en utilizar programación funcional para la construcción de software orientado a la resolución de problemas de cálculo numérico. Los lenguajes funcionales modernos, y en especial Haskell, permiten obtener programas cuyo desempeño es comparable al de los programas clásicos implementados usando lenguajes imperativos. El uso de programación funcional redunda en múltiples ventajas, siendo una de las más importantes la estrecha relación que se logra entre el diseño y la implementación, como se describe especialmente en este trabajo.
Las siguientes secciones están organizadas como se indica a continuación:
En la sección 2 se describen algunos métodos para resolver ecuaciones no lineales y algunos métodos para resolver sistemas de ecuaciones lineales.
En la sección 3 se describe el algoritmo , la implemenatción en Haskell de las funciones principales y algunas instancias del algoritmo general.
En la sección 4 se presentan las conclusiones.

2 Presentación de algunos métodos numéricos.

En esta sección se formaliza el planteo de los dos tipos de problemas que resuelve el algoritmo: resolución de ecuaciones no lineales y resolución de sistemas de ecuaciones lineales, usando los métodos iterativos citados antes, los cuales se describen brevemente a continuación.

2.1 Ecuaciones no lineales.

No son pocos los problemas del "mundo real" que se modelan mediante ecuaciones no lineales. A diferencia de las ecuaciones lineales, la solución analítica de una ecuación no lineal es sólo posible

en algunos casos muy particulares[1]. Este hecho motiva la existencia de métodos iterativos para resolución de ecuaciones no lineales. También se debe tener en cuenta que en la mayoría de las aplicaciones es suficiente hallar una aproximación a la solución de la ecuación, siempre que se pueda controlar la precisión de dicha aproximación. Se presentan aquí algunos métodos numéricos para resolución de ecuaciones no lineales.

2.1.1 Métodos iterativos para resolución de ecuaciones no lineales.

Sea f una función, $f : R \to R$. Se dice que $x \in R$ es *raíz* de la función $f \iff f(x) = 0$. El objetivo es calcular las raíces de f. Los algoritmos de búsqueda de raíces se basan casi exclusivamente en métodos iterativos. Estos métodos siguen todos el mismo principio: en lugar de resolver la ecuación no lineal directamente, se resuelve una secuencia de problemas más simples, con la esperanza de que la secuencia de soluciones obtenidas converja a la solución del problema original [Dah74]. Para resolver iterativamente una ecuación no lineal, hay básicamente dos formas de construir dicha secuencia de subproblemas:

- Generar una sucesión de intervalos (de largo decreciente) que contengan a la raíz. El punto de partida es un intervalo que contiene a la raíz.
- Generar una sucesión de aproximaciones de la raíz a partir de uno o más valores iniciales.

Los métodos que se analizan aquí calculan una aproximación a *una raíz* de una función partiendo, o bien de un valor cercano, o bien de un intervalo que la contiene. La determinación del valor o del intervalo inicial es bastante simple si se dispone del gráfico de la función.

Para decidir la detención de la iteración se observan dos condiciones, a saber:

- Condición de detención por éxito: si se tiene una aproximación de la raíz que ya es satisfactoria, no es necesario continuar con la iteración. Es muy común usar la siguiente condición:
  - un valor x es una aproximación satisfactoria de la raíz de $f \Leftrightarrow |f(x)| < \epsilon$, donde ϵ es una determinada precisión.
- Condición de detención por fracaso: a veces la iteración no converge a una aproximación de la raíz. Esto puede ser causado por una mala elección de los valores iniciales para la iteración, o porque la función no cumple las condiciones de convergencia para el método de resolución elegido. En estos casos es necesario detener la iteración para evitar que el algoritmo entre en un ciclo infinito.
  Las siguientes son algunas condiciones usuales de detención:
  - Cantidad de pasos de iteración mayor que una cantidad máxima predeterminada: $i > max\_iter$
  - Dos aproximaciones sucesivas son demasiado cercanas: $|x_i - x_{i-1}| < \epsilon$
  - Los valores funcionales en dos aproximaciones sucesivas son demasiado cercanos: $|f(x_i) - f(x_{i-1})| < \epsilon$
  - Si el método genera una sucesión de intervalos: el ancho del intervalo actual es demasiado pequeño: $|a_i - b_i| < \epsilon$

A continuación se analizan algunos métodos de resolución de ecuaciones no lineales. Una explicación detallada de éstos y su convergencia se puede hallar en [Nak92], [Kah89], [Bur85] y [Bec67].

[1] Existen fórmulas generales para ecuaciones que involucran polinomios de hasta grado cuatro, y estas fórmulas suelen ser tan complicadas que no se usan en la práctica.

- Método de Bisección.
  El método de Bisección se basa en el Teorema del Valor Intermedio de Bolzano [Bur85]: Si una función f es continua en un intervalo $[a,b]$ y $f(a) \cdot f(b) < 0$ entonces existe $x \in [a,b]$ tal que $f(x) = 0$. De esta forma, cuando f es continua y se conoce un intervalo que cumple con las hipótesis del teorema anterior, se puede construir una sucesión de intervalos $[a_n, b_n]$ mediante el siguiente esquema de iteración:
  - $a_0 = a$, $b_0 = b$.
  - Se calcula el punto medio del intervalo $[a_i, b_i]$, $m_i = \frac{a_i + b_i}{2}$. En cada paso dicho m_i es la aproximación de la raíz.
  - Si m_i es una aproximación satisfactoria, se detiene la iteración, si no, se recalcula el intervalo que contiene la raíz de la forma siguiente:
    * Si $f(a_i) \cdot f(m_i) < 0$, como f es continua en $[a_i, m_i]$ al serlo en $[a_i, bi]$, entonces el teorema de Bolzano garantiza la existencia de una raíz en $[a_i, m_i]$, y por lo tanto es ese el nuevo intervalo para la iteración : $a_{i+1} = a_i$, $b_{i+1} = m_i$.
    * Análogamente, si $f(m_i) \cdot f(b_i) < 0$ entonces existe una raíz en $[m_i, b_i]$: $a_{i+1} = m_i$, $b_{i+1} = b_i$.
- Método de Newton-Raphson.
  El método de Newton-Raphson es ampliamente aceptado como uno de los mejores métodos para obtener aproximaciones a raíces de funciones [Bec67]. Sea f la función cuya raíz α deseamos calcular. El método construye una sucesión de aproximaciones x_i a α. Es necesaria una aproximación inicial x_0, y dependiendo de la calidad de ésta, el método puede converger a una aproximación de la raíz o no.
  Para dar el esquema de iteración se necesita una fórmula que permita, dada una aproximación de la raíz, hallar otra aproximación mejor. Aplicando la técnica de linearización local [Bec67], es posible suponer que f se comporta como una función lineal[2] cerca de x_i. Esta función lineal es la asociada a la recta de pendiente $f'(x_i)$ que pasa por el punto $(x_i, f(x_i))$. Por lo tanto una mejor aproximación para α es la raíz de dicha función lineal. Entonces la fórmula de iteración para el método de Newton-Raphson es:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- Método de la Secante.
  El método de la Secante es muy similar al de Newton-Raphson, ya que también genera una sucesión de aproximaciones a α, y el esquema de iteración se basa también en aplicar la técnica de linearización local. Esta vez, se aproxima f por la función lineal correspondiente a la recta que pasa por los puntos $(x_{i-1}, f(x_{i-1}))$ y $(x_i, f(x_i))$. El punto de corte de esta recta con el eje de las abscisas es una mejor aproximación para la raíz. De ese modo, se obtiene la fórmula de iteración para el método de la Secante:

$$x_{i+1} = x_i - f(x_i) \cdot \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

- Método de la Falsa Posición.
  El método de la Falsa Posición es una combinación del método de Bisección y el de la Secante en el sentido de que genera una sucesión de intervalos $[a_n, b_n]$, con un esquema de iteración que es en todo similar al del método de Bisección. La diferencia radica en que en cada paso de

[2] En definitiva se aproxima f por su desarrollo de Taylor de orden 1.

la iteración, se consideran los puntos $(a_i, f(a_i))$ y $(b_i, f(b_i))$, y la aproximación a α buscado es:

$$x_i = b_i - f(b_i) \cdot \frac{b_i - a_i}{f(b_i) - f(a_i)}$$

La elección del siguiente intervalo es igual que en el método de Bisección, pero tomando el x_i anterior, en lugar del punto medio del intervalo considerado.

2.2 Sistemas de ecuaciones lineales.

La necesidad de solucionar un sistema lineal puede aparecer directamente al modelar algún problema físico o indirectamente, como parte de la solución numérica de otro problema. Por lo tanto resulta interesante el estudio de algunas técnicas de resolución de sistemas lineales. Por la naturaleza de este trabajo, se hace énfasis en los métodos iterativos.

2.2.1 Métodos iterativos para resolución de ecuaciones lineales.

Se denomina $R_{m \times n}$ al conjunto de las matrices de dimensión $m \times n$ de números reales.
Dadas $A \in R_{m \times n}$, $b \in R_{m \times 1}$, se desea hallar $x \in R_{n \times 1}$ tal que $A \cdot x = b$, que es la notación matricial para un *sistema de ecuaciones lineales* (o sistema lineal). x se denomina *solución* del sistema.
Se considera solamente el caso en que la matriz A es invertible. En ese caso la solución *existe y es única*, y el sistema lineal se dice *compatible determinado*. Los métodos de resolución de sistemas lineales pertenecen básicamente a dos familias: *métodos directos* y *métodos iterativos*. Los métodos directos calculan exactamente la solución del sistema, en general a partir de eliminación Gaussiana. Los métodos iterativos construyen una sucesión $\{x_n\}$ que bajo ciertas hipótesis converge a la solución x buscada. Cada término de la sucesión se obtiene luego de un paso de la iteración del algoritmo. El algoritmo se detiene cuando se logra un cierto nivel de precisión en la solución. Un estudio detallado de métodos se encuentra en [Dah74] y en [Nak92].
A veces los métodos directos no son los más convenientes para resolver un sistema lineal. Cuando predominan los ceros en la matriz del sistema, la eliminación gaussiana hace operaciones innecesarias, en elementos que a priori se sabe serán nulos, por lo que los métodos iterativos se tornan más convenientes. El diseño de métodos iterativos de resolución de sistemas de ecuaciones se basa en la igualdad:

$$A \cdot x = b \Longleftrightarrow M \cdot x = (M - A) \cdot x + b$$

donde M es una matriz fácil de invertir.
A partir de la igualdad anterior, es posible definir la siguiente fórmula de iteración:

$$x^{(k+1)} = M^{-1} \cdot (M - A) \cdot x^{(k)} + b$$

que permite mejorar en cada paso la aproximación de la solución.
Para comenzar la iteración, se necesita una primera aproximación de la solución, $x^{(0)}$. Es probable que si esta primera aproximación no es lo suficientemente buena, el método no converja. Es claro que la convergencia depende también de la matriz M elegida. Si la solución $\{x^{(k)}\}$ converge, lo hace también la solución del sistema.
Al igual que con las ecuaciones no lineales, se desea hallar una aproximación a la solución del sistema utilizando algún método iterativo. Nuevamente se deben dar condiciones para la detención de la iteración:

- Condición de detención por éxito:
  Si se tiene una aproximación de la solución del sistema que es ya satisfactoria, no es necesario continuar con la iteración. Es muy común usar la siguiente condición:
  - un valor x es una aproximación satisfactoria de la solución del sistema de $A \cdot x = b \Leftrightarrow \|A \cdot x - b\| < \epsilon$.
- Condición de detención por fracaso:
  A veces la iteración no converge a una aproximación de la solución. Esto puede ser causado por una mala elección de los valores iniciales para la iteración, o porque el sistema no cumple las condiciones de convergencia para el método de resolución elegido. En estos casos es necesario detener la iteración para evitar que el algoritmo entre en un ciclo infinito.
  Las siguientes son algunas condiciones usuales de detención:
  - Cantidad de pasos de iteración mayor que una cantidad máxima predeterminada:
    $i > max\_iter$
  - Dos aproximaciones sucesivas son demasiado cercanas:
    $\|x_i - x_{i-1}\| < \epsilon$

Al elegir la matriz M, queda determinado el método. Se analizan a continuación dos métodos iterativos conocidos: el método de Jacobi y el método de Gauss-Seidel.

- Método de Jacobi.
  Consiste en tomar $M = diagonal(A)$. La fórmula de iteración es:

$$x^{(k+1)} = D^{-1} \cdot (D - A) \cdot x^{(k)} + b$$

  donde el cálculo de D^{-1} es trivial.

- Método de Gauss-Seidel.
  En este caso se considera la siguiente descomposición para la matriz del sistema: $A = D - (E + F)$ donde:

$$D = diagonal(A)$$

$$e_{ij} = \begin{cases} 0 & (i \leq j) \\ a_{ij} & (i > j) \end{cases}$$

$$f_{ij} = \begin{cases} 0 & (i \geq j) \\ a_{ij} & (i < j) \end{cases}$$

  Se toma $M = D - E$ y la fórmula de iteración es:

$$x^{(k+1)} = (D - E)^{-1} \cdot F \cdot x^{(k)} + b$$

3 Descripción del algoritmo

Se describe a continuación un algoritmo general para resolver el problema de hallar la solución de una ecuación no lineal o de un sistema de ecuaciones lineales, aplicando algún método numérico iterativo adecuado. Los pasos del algoritmo son:

1. Construir la lista:

$$x_0,\ (m\ f)\ x_0,\ (m\ f)\ ((m\ f)\ x_0),\ (m\ f)\ ((m\ f)\ ((m\ f)\ x_0)),\ \ldots$$

de las sucesivas aplicaciones de $(m\ f)$ al valor inicial x_0, donde m es el método y f es la función. Cada uno de los elementos de la lista, contiene una mejor aproximación a la raíz y además la información necesaria para decidir en cada paso de la iteración, si parar o no y en caso negativo, determinar el paso siguiente en la iteración.

2. Inspeccionar los elementos de la lista hasta que se cumpla una de las siguientes condiciones:
   - algún elemento satisface la deseada precisión, en cuyo caso se devuelve ese valor
   - algún elemento satisface la condición de parada, en cuyo caso se devuelve error

El algoritmo es paramétrico en :

- **el método m**: el algoritmo *abstrae* el método de resolución del problema, pudiendo aplicarse a cualquiera de los métodos descritos anteriormente de acuerdo al problema en cuestión.
- **el problema f**: Se distinguen dos tipos:
  - El problema de hallar la solución de una ecuación no lineal, que se plantea como:
    Sea f una función, $f : R \to R$. Hallar $x \in R$ tal que $f(x) = 0$ lo cual es equivalente a solucionar la ecuación $f(x) = 0$.
  - El problema de hallar la solución de un sistema de ecuaciones lineales, que se plantea como:
    Dadas $A \in R_{m\times n}$, $b \in R_{m\times 1}$, hallar $x \in R_{n\times 1}$ tal que: $A \cdot x = b$, o sea $A \cdot x - b = 0$.
    En este caso se trata también de solucionar la ecuación $f(x) = 0$, donde f depende de las matrices A y b, es decir, $f(A, b, x) = A \cdot x - b$.
- **criterios de detención y constantes para afinar el comportamiento del algoritmo**: el algoritmo puede detenerse por *éxito*, en cuyo caso se encontró la raíz, con la precisión indicada por una de las constantes y según algún criterio de detención, o por *fracaso*, en cuyo caso el método no converge y es necesario detener la iteración, impidiendo que el algoritmo entre en un ciclo infinito, para lo cual se utiliza otra constante y otro criterio de detención.
- **valor inicial $\mathbf{x_0}$**: es el valor inicial del cual parte la iteración. El tipo del valor inicial depende del método.

Si llamamos *resolver* a la función que representa el algoritmo, tenemos la siguiente implementación en Haskell de las principales funciones del algoritmo :

```
resolver metodo f satisface parar sacar_valor x0 eps1 eps2 =
               revisar (satisface g eps1)
                    (parar eps2) sacar_valor
                    (repetir (metodo f) x0)
                where g = f . sacar_valor

revisar ins_sat ins_parar sacar_valor (x:xs) =
           if (ins_sat x)
           then (Exitoso (sacar_valor x))
           else
               (if (ins_parar x)
               then Error
               else (revisar ins_sat ins_parar sacar_valor xs))

repetir f a = a : repetir f (f a)
```

Como se puede apreciar, el resultado está dado por la aplicación de la función *revisar* a sus argumentos. La función *revisar* inspecciona secuencialmente los elementos de la lista verificando para

cada uno, si satisface la condición de precisión (caso de éxito). En caso afirmativo, se aplica al elemento la función *sacar_valor* para obtener la aproximación a la solución que se devuelve como resultado. En caso negativo, se verifica si el elemento satisface la condición de detención (caso de error), y si es asi, se devuelve error. Si no se satisface ninguna de las condiciones de detención, se continua inspeccionando la lista. De esta manera, la función *revisar* realiza el paso 2 de la descripción del algoritmo, donde la detención por éxito está representada por el argumento *(satisface g* eps_1*)* y la detención por fracaso está representada por el argumento *(parar* eps_2*)*.
El argumento *repetir (método ecuación)* x_0 representa el paso 1 de la descripción del algoritmo ya que la función *repetir* construye la lista infinita cuyos elementos son los valores obtenidos en cada paso de la iteración. Uno de los aspectos destacables del uso de Haskell como lenguaje de programación, es que la implementación de esta lista infinita es posible debido a la *semántica perezosa de Haskell*. Esto implica que la verificación de las condiciones de detención por los elementos se realiza sin necesidad de construir toda la lista.

3.1 Los métodos de resolución particulares como instancias del algoritmo general

El aspecto más destacable del uso de Haskell como lenguaje de programación es la posibilidad de *abstraer el método de resolución* de la ecuación, de modo de obtener efectivamente una implementación del algoritmo general, por medio de la función *resolver*. Las *instancias* del algoritmo general para cada método, constituyen algoritmos particulares para resolver el problema por el método en cuestion. A nivel de la implementación, cada una de las instancias, es la aplicación de la función *resolver* a una función que represente el método.
Para los métodos descritos en este trabajo se obtienen las siguients implementaciones:

- **Para ecuaciones no lineales:**
  - *Método de biseccción*:

```
biseccion = resolver bisec
       where
          bisec f (a,b) = if (f a)*(f m) <= 0
                          then (a,m)
                          else (m,b)
                          where
                             m = (a+b)/2
```

  - *Método de Newton-Raphson*:

```
newton = resolver newt
     where
        newt f (x1,x2) = (x2,x2- y/f')
             where
                y = f x2
                f' = derivada f x2 h eps
                  where
                    eps = 0.000000000001
                    h = 0.001
```

    La función *derivada* implementa la derivación numérica y se estudia en [Hug84].
  - *Método de la Secante*:

```
secante = resolver sec
      where
         sec f (x1,x2) = (x2,x2-(f x2)*(x2-x1)/((f x2)-(f x1)))
```

- *Método de la Falsa Posición*:

```
Falsa_Pos = resolver falsa_pos
      where
        falsa_pos f (a,b) = if (f a)*(f m) <= 0
                            then (a,m)
                            else (m,b)
                            where
                               m =b-(f b)*(b-a)/((f b)-(f a))
```

Implementación de condiciones de detención al resolver ecuaciones no lineales:

- Detención por éxito:

```
satisface f sacar_valor eps x = (abs(f (sacar_valor x)) < eps)
```

- Detención por fracaso:

```
detener f sacar_valor eps (x,y) = (abs(x-y)<eps)
```

- **Para sistemas de ecuaciones lineales:**
  - *Método de Jacobi*:

```
jacobi = resolver (jac A b)
     where
        jac A b x = producto d1 (suma (producto (resta d A) x) b)
        d = diagonal A
        d1 = inversa d
```

  - *Método de Gauss-Seidel*:

```
g_seidel = resolver (g_s A b)
     where g_s A b x = solve asist bsist
         asist= resta (hallo_d A) (hallo_e A)
         bsist= suma b (producto (hallo_f A) x)
```

    La función *solve* resuelve un sistema que se plantea para avanzar cada paso en la iteración. La implementación de $hallo_d$, $hallo_e$ y $hallo_f$ es muy simple.

Implementación de condiciones de detención al resolver sistemas lineales:

- Detención por éxito:

```
satisface f sacar_valor eps x =
      ((norma (resta  (f (sacar_valor x)) (sacar_valor x))) < eps)
```

- Detención por fracaso:

```
detener f sacar_valor eps (x_n_1,x_n) =
      ((norma (resta x_n_1 x_n)) < eps)
```

4 Conclusiones

El desarrollo de los lenguajes funcionales de programación viene dando muy rapidamente como resultado lenguajes cada vez más adecuados para su aplicación al diseño y desarrollo de software. Una consecuencia importante de este hecho es que el uso de los lenguajes funcionales se extiende velozmente a los más variados campos; desde la computación gráfica hasta las telecomunicaciones, pasando por sistemas de bases de datos, lo cierto es que hoy en día es un mito del pasado aquello de que "la programación funcional es cosa de laboratorio".

Las ventajas que proporciona programar usando lenguajes funcionales han sido repetidamente enumeradas, siendo una de las más populares la rapidez y facilidad con que se pueden obtener buenos prototipos con la consecuencia de disminuir riesgos: se ha probado que el costo de corregir un error en una etapa temprana del ciclo de vida del software es exponencialmente menor que el costo de corregirlo más tarde. [HJ94]
Este trabajo es una pequeña muestra de las facilidades que brinda la programación funcional al área de cálculo numérico. Se ha implementado un algoritmo general que puede ser aplicado a diferentes métodos numéricos iterativos ya sea para resolución de ecuaciones no lineales o para resolución de sistemas de ecuaciones lineales. De esta forma se logra un alto nivel de abstracción, que aumenta la reusabilidad del software producido, lo cual redunda en flexibilidad y comodidad desde el punto de vista del usuario.
Se puede apreciar la correspondencia intrínseca que existe entre la especificación del algoritmo por pasos en lenguaje natural y la función que lo representa en Haskell, lo que permite que el programador pueda concentrarse en la escencia de la solución del problema, olvidando detalles de implementación. Es especialmente sencilla la traducción de expresiones del lenguaje matemático a Haskell.

References

[Dah74] "Numerical Methods. Germund Dahlquist, Ake Bjorck, Ned Anderson." *Prentice Hall, Inc.*

[Bur85] "Análisis Numérico. Richard Burden, Douglas Faires.
Numerical Methods." *Grupo Editorial Iberoamérica.*

[Nak92] "Métodos Numéricos Aplicados con Software. Shoichiro Nakamura'.' *Prentice Hall, Inc.*

[Kah89] "Numerical Methods and Software. David Kahaner, Cleve Moler, Stephen Nash." *Prentice Hall International Edition.*

[Bec67] "Numerical Calculations and Algorithms. Royce Beckett, James Hurt." *Robert E Krieger Publishing Company.*

[Rep92] "Report on the Programming Languaje Haskell. A Non-Strict Purely Functional Language. P. Hudack, S. Peyton Jones, P. Wadler (editors)." *ACM SIGPLAN Notices.*

[Hug84] "Why Functional Programming Matters" R.J.M. Hughes. *The Programming Methodology Group's Report Series. Number 16.*

[HJ94] "Haskell vs. Ada vs. C++ vs. Awk vs. ... An Experiment in Software Prototyping Productivity" Paul Hudak, Mark Jones. *Work supported by the Advanced Research Project Agency and the Office of Naval Research under Arpa Order 8888,Contract N00014-92-C-0153*