Métodos Numéricos
Implementación del algoritmo de búsqueda **k**-vector

Dores Piuma Francisco
Moser Plugge Tylbert
Noblía Martín
Vieras Fabián

Universidad Nacional de Quilmes

10 de julio de 2013

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*.

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*.Pero este método al ser su formulación general a cualquier base de datos(estática o pseudoestática) puede ser aplicado a una amplia gama de problemas, como:

- Interpolación (1D o 2D *look-up tables*)

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*.Pero este método al ser su formulación general a cualquier base de datos(estática o pseudoestática) puede ser aplicado a una amplia gama de problemas, como:

- Interpolación (1D o 2D *look-up tables*)
- Búsqueda de ceros de funciones no-lineales

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*.Pero este método al ser su formulación general a cualquier base de datos(estática o pseudoestática) puede ser aplicado a una amplia gama de problemas, como:

- Interpolación (1D o 2D *look-up tables*)
- Búsqueda de ceros de funciones no-lineales
- Inversión de funciones no-lineales

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*.Pero este método al ser su formulación general a cualquier base de datos(estática o pseudoestática) puede ser aplicado a una amplia gama de problemas, como:

- Interpolación (1D o 2D *look-up tables*)
- Búsqueda de ceros de funciones no-lineales
- Inversión de funciones no-lineales
- *Sampling* de distribuciones de probabilidades

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario,además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*.Pero este método al ser su formulación general a cualquier base de datos(estática o pseudoestática) puede ser aplicado a una amplia gama de problemas, como:

- Interpolación (1D o 2D *look-up tables*)
- Búsqueda de ceros de funciones no-lineales
- Inversión de funciones no-lineales
- *Sampling* de distribuciones de probabilidades

Que representa el problema de búsqueda?

Que representa el problema de búsqueda?

El problema de búsqueda en un rango de valores de una base de datos estática $\mathbf{y}(n)$ con un número de valores $n \gg 1$ es encontrar el subconjunto de elementos que satisfacen $\mathbf{y}(\mathbf{I})$, donde $\mathbf{I}$ es un vector de índices tal que $\mathbf{y}(\mathbf{I}) \in [y_a, y_b]$ donde $y_a < y_b$.

Lo que habitualmente se utiliza

Lo que habitualmente se utiliza

Este problema se resuelve habitualmente con el método de búsqueda binario, el cual tiene un orden de complejidad $\mathcal{O}(2log_2(n))$

Lo que habitualmente se utiliza

Este problema se resuelve habitualmente con el método de búsqueda binario, el cual tiene un orden de complejidad $\mathcal{O}(2log_2(n))$

La mejora sustancial de este método

El método de búsqueda **k**-vector tiene en cambio un orden de complejidad invariante con respecto a n y es de $\mathcal{O}(3)$.

definición 1

definición 1

$$y_{max} = \max_{i}\{\mathbf{y}(i)\} = \mathbf{s}(n) \tag{1}$$

definición 1

$$y_{max} = \max_i\{\mathbf{y}(i)\} = \mathbf{s}(n) \tag{1}$$

$$y_{min} = \min_i\{\mathbf{y}(i)\} = \mathbf{s}(1) \tag{2}$$

definición 1

$$y_{max} = \max_i\{\mathbf{y}(i)\} = \mathbf{s}(n) \tag{1}$$

$$y_{min} = \min_i\{\mathbf{y}(i)\} = \mathbf{s}(1) \tag{2}$$

definición 2

El corazón del algoritmo:

definición 1

$$y_{max} = \max_i\{\mathbf{y}(i)\} = \mathbf{s}(n) \tag{1}$$

$$y_{min} = \min_i\{\mathbf{y}(i)\} = \mathbf{s}(1) \tag{2}$$

definición 2

El corazón del algoritmo:

$$[1, y_{min} - \delta\epsilon] \rightarrow [n, y_{max} + \delta\epsilon] \tag{3}$$

definición 1

$$y_{max} = \underset{i}{\text{máx}}\{\mathbf{y}(i)\} = \mathbf{s}(n) \tag{1}$$

$$y_{min} = \underset{i}{\text{mín}}\{\mathbf{y}(i)\} = \mathbf{s}(1) \tag{2}$$

definición 2

El corazón del algoritmo:

$$[1, y_{min} - \delta\epsilon] \rightarrow [n, y_{max} + \delta\epsilon] \tag{3}$$

Donde:

$$\delta\epsilon = (n-1)\epsilon \tag{4}$$

La recta que une esos puntos la defimos:

La recta que une esos puntos la defimos:

$$z(x) = mx + q \tag{5}$$

Donde:

La recta que une esos puntos la defimos:

$$z(x) = mx + q \tag{5}$$

Donde:

$$\begin{cases} m = (y_{max} - y_{min} + 2\delta\epsilon)/(n-1) \\ q = y_{min} - m - \delta\epsilon \end{cases} \tag{6}$$

Definición **k**-vector

Definición **k**-vector

Definición **k**-vector

nuestro **k**-vector debe cumplir :

Definición **k**-vector

Definición **k**-vector

nuestro **k**-vector debe cumplir :

$$\mathbf{k}(1) = 0 \tag{7}$$

Definición **k**-vector

Definición **k**-vector

nuestro **k**-vector debe cumplir :

$$\mathbf{k}(1) = 0 \tag{7}$$

$$\mathbf{k}(n) = n \tag{8}$$

Definición **k**-vector

Definición **k**-vector

nuestro **k**-vector debe cumplir :

$$\mathbf{k}(1) = 0 \tag{7}$$

$$\mathbf{k}(n) = n \tag{8}$$

Y además en la posición i-ésima $\mathbf{k}(i) = j$, donde j es el índice más grande que cumple:

Definición **k**-vector

Definición **k**-vector

nuestro **k**-vector debe cumplir :

$$\mathbf{k}(1) = 0 \tag{7}$$

$$\mathbf{k}(n) = n \tag{8}$$

Y además en la posición i-ésima $\mathbf{k}(i) = j$, donde j es el índice más grande que cumple:

$$\mathbf{s}(j) \leq \mathbf{y}(\mathbf{l}(i)) \tag{9}$$

Como se interpreta la ecuación (9)

Como se interpreta la ecuación (9)

Desde un punto de vista práctico los valores de $\mathbf{k}(i)$ representan el número de elementos de la base de datos $\mathbf{y}$ que están por debajo del valor $z(i)$.

Una vez que el **k**-vector ha sido creado podemos conocer cuantos y en que posición del vector **s** están los datos que buscamos ($[y_a, y_b]$). De hecho los índices asociados con estos valores en el vector **s** están dados por:

Una vez que el **k**-vector ha sido creado podemos conocer cuantos y en que posición del vector **s** están los datos que buscamos ($[y_a, y_b]$). De hecho los índices asociados con estos valores en el vector **s** están dados por:

índices de búsqueda

Una vez que el **k**-vector ha sido creado podemos conocer cuantos y en que posición del vector **s** están los datos que buscamos ($[y_a, y_b]$). De hecho los índices asociados con estos valores en el vector **s** están dados por:

índices de búsqueda

$$j_b = \lfloor \frac{y_a - q}{m} \rfloor \qquad \text{and} \qquad j_t = \lceil \frac{y_b - q}{m} \rceil \tag{10}$$

Una vez que el **k**-vector ha sido creado podemos conocer cuantos y en que posición del vector **s** están los datos que buscamos ($[y_a, y_b]$). De hecho los índices asociados con estos valores en el vector **s** están dados por:

índices de búsqueda

$$j_b = \lfloor \frac{y_a - q}{m} \rfloor \qquad \text{and} \qquad j_t = \lceil \frac{y_b - q}{m} \rceil \tag{10}$$

Una vez que tenemos los indices podemos obtener los indices de el rango de búsqueda de acuerdo a:

Una vez que el **k**-vector ha sido creado podemos conocer cuantos y en que posición del vector **s** están los datos que buscamos ($[y_a, y_b]$). De hecho los índices asociados con estos valores en el vector **s** están dados por:

índices de búsqueda

$$j_b = \lfloor \frac{y_a - q}{m} \rfloor \qquad \text{and} \qquad j_t = \lceil \frac{y_b - q}{m} \rceil \tag{10}$$

Una vez que tenemos los indices podemos obtener los indices de el rango de búsqueda de acuerdo a:

Indices en el k-vector

$$k_{start} = \mathbf{k}(j_b) + 1 \qquad \text{and} \qquad k_{end} = \mathbf{k}(j_t) \tag{11}$$

Elementos buscados

Asi finalmente los elementos buscados en la base de datos $\mathbf{y}(i) \in [y_a, y_b]$ son los elementos $\mathbf{y}(\mathbf{l}(k))$, donde $k \in [k_{start}, k_{end}]$

Elementos buscados

Asi finalmente los elementos buscados en la base de datos $\mathbf{y}(i) \in [y_a, y_b]$ son los elementos $\mathbf{y}(\mathbf{l}(k))$, donde $k \in [k_{start}, k_{end}]$Sin embargo en la base de datos pueden aparecer elementos extraños originados por el hecho que en promedio los índices k_{star} y k_{end} tienen cada uno 50 % de probabilidades de no pertenecer a los intervalos $[y_a, z(j_a + 1)]$ y $[z(j_b, y_b)]$ respectivamente.

Elementos buscados

Asi finalmente los elementos buscados en la base de datos $\mathbf{y}(i) \in [y_a, y_b]$ son los elementos $\mathbf{y}(\mathbf{I}(k))$, donde $k \in [k_{start}, k_{end}]$Sin embargo en la base de datos pueden aparecer elementos extraños originados por el hecho que en promedio los índices k_{star} y k_{end} tienen cada uno 50 % de probabilidades de no pertenecer a los intervalos $[y_a, z(j_a + 1)]$ y $[z(j_b, y_b)]$ respectivamente.Asi si E_0 es la cantidad de elementos esperados en el rango $[z(j), z(j + 1)]$ entonces el número de elementos extraños esperados en el rango $[y_a, y_b]$ es $\frac{2E_0}{2} = E_0 = \frac{n}{(n-1)} \approx 1$ para $n \gg 1$.

Elementos buscados

Asi finalmente los elementos buscados en la base de datos $\mathbf{y}(i) \in [y_a, y_b]$ son los elementos $\mathbf{y}(\mathbf{I}(k))$, donde $k \in [k_{start}, k_{end}]$Sin embargo en la base de datos pueden aparecer elementos extraños originados por el hecho que en promedio los índices k_{star} y k_{end} tienen cada uno 50 % de probabilidades de no pertenecer a los intervalos $[y_a, z(j_a + 1)]$ y $[z(j_b, y_b)]$ respectivamente.Asi si E_0 es la cantidad de elementos esperados en el rango $[z(j), z(j + 1)]$ entonces el número de elementos extraños esperados en el rango $[y_a, y_b]$ es $\frac{2E_0}{2} = E_0 = \frac{n}{(n-1)} \approx 1$ para $n \gg 1$. Por ello en presencia de este elemento extraño, debemos hacer dos búsquedas locales más para eliminar dicho elemento. Formalmente:

Elementos buscados

Asi finalmente los elementos buscados en la base de datos $\mathbf{y}(i) \in [y_a, y_b]$ son los elementos $\mathbf{y}(\mathbf{I}(k))$, donde $k \in [k_{start}, k_{end}]$Sin embargo en la base de datos pueden aparecer elementos extraños originados por el hecho que en promedio los índices k_{star} y k_{end} tienen cada uno 50 % de probabilidades de no pertenecer a los intervalos $[y_a, z(j_a + 1)]$ y $[z(j_b, y_b)]$ respectivamente.Asi si E_0 es la cantidad de elementos esperados en el rango $[z(j), z(j + 1)]$ entonces el número de elementos extraños esperados en el rango $[y_a, y_b]$ es $\frac{2E_0}{2} = E_0 = \frac{n}{(n-1)} \approx 1$ para $n \gg 1$. Por ello en presencia de este elemento extraño, debemos hacer dos búsquedas locales más para eliminar dicho elemento. Formalmente:

Eliminación de elementos extraños

Eliminación de elementos extraños

$$\begin{cases} \text{verificar si} & \mathbf{y}(\mathbf{l}(\mathbf{k})) < y_a \\ \text{verificar si} & \mathbf{y}(\mathbf{l}(\mathbf{k})) > y_b \end{cases} \quad \text{Donde} \quad \begin{matrix} k = k_{start} \\ k = k_{end} \end{matrix} \tag{12}$$

Principales limitaciones del algoritmo

Principales limitaciones del algoritmo

- Pueden aparecer elementos extraños en una búsqueda

Principales limitaciones del algoritmo

- Pueden aparecer elementos extraños en una búsqueda
- El número de elementos extraños en una búsqueda depende de n

Principales limitaciones del algoritmo

- Pueden aparecer elementos extraños en una búsqueda
- El número de elementos extraños en una búsqueda depende de n
- el orden de complejidad($\mathcal{O}(3)$) se mantiene si el**k**-vector es lineal o cuasi-lineal con respecto a los indices

El método de **k**-vector exibe ventajas cuando es aplicado a bases de datos estáticas, por ello puede ser utilizado de manera eficiente para invertir funciones no-lineales de la forma $y = f(\mathbf{x})$, donde f es una función no lineal de n variables independientes. Éste método no es el adecuado para invertir una función sólo una vez, ya que su eficiencia se ve reflejada cuando la inversión debe hacerse repetidas veces
en algún rango de valores de la función. Una vez que el preprocesamiento se ha completado, la inversión puede ser creada y usada en cualquier momento. La metodología puede ser resumida en los siguientes diagramas de flujo:

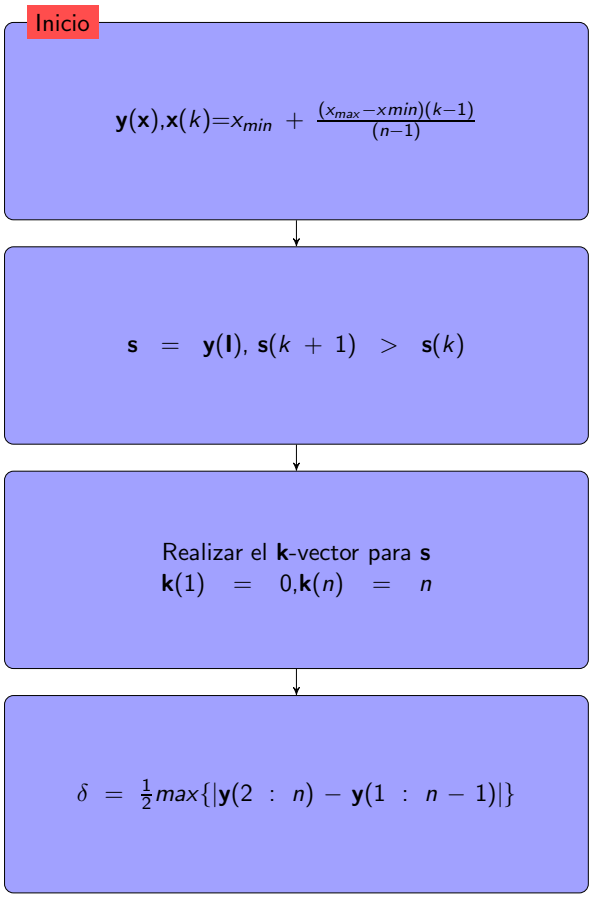


Figura: Preprocesamiento

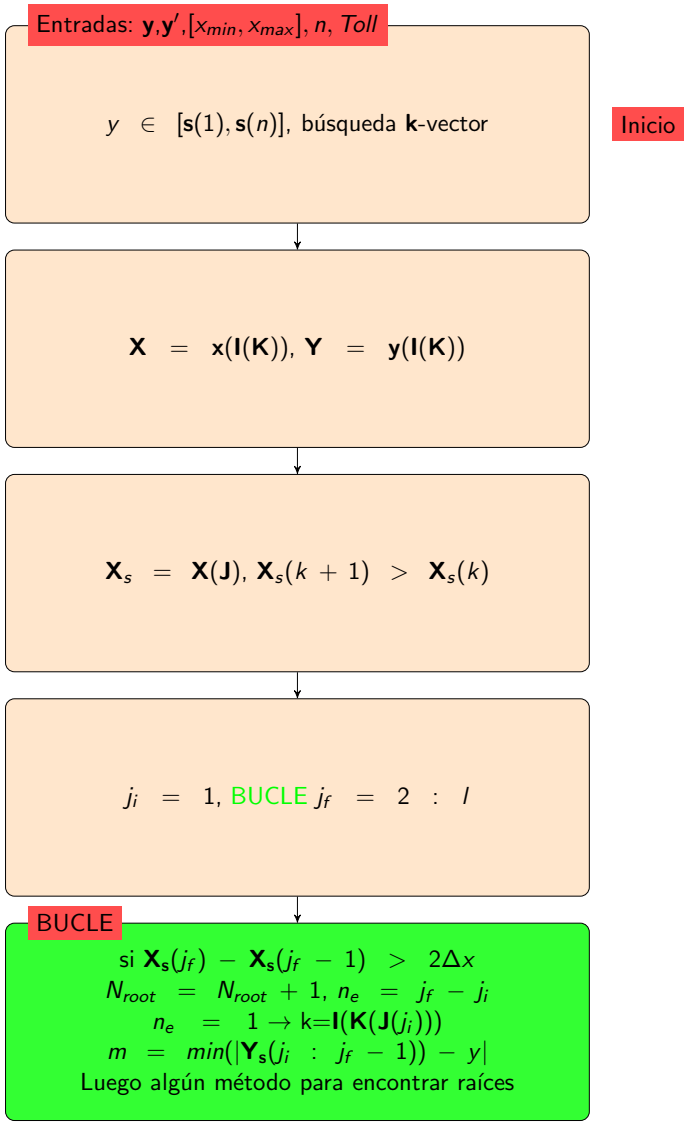


Figura: Inversión de una función no-lineal

Resolución de sistemas de dos ecuaciones no lineales

Sea el siguiente sistema de dos ecuaciones:

Resolución de sistemas de dos ecuaciones no lineales

Sea el siguiente sistema de dos ecuaciones:

$$y = f(\mathbf{x}) \qquad \text{and} \qquad y = g(\mathbf{x}) + c \tag{13}$$

Resolución de sistemas de dos ecuaciones no lineales

Sea el siguiente sistema de dos ecuaciones:

$$y = f(\mathbf{x}) \qquad \text{and} \qquad y = g(\mathbf{x}) + c \tag{13}$$

Donde $f(\mathbf{x})$ y $g(\mathbf{x})$ pueden ser dos funciones nolineales dadas y c una constante cuyo valor puede cambiar. Se puede ver que el problema anterior es equivalente a resolver la siguiente ecuación:

Resolución de sistemas de dos ecuaciones no lineales

Sea el siguiente sistema de dos ecuaciones:

$$y = f(\mathbf{x}) \qquad \text{and} \qquad y = g(\mathbf{x}) + c \tag{13}$$

Donde $f(\mathbf{x})$ y $g(\mathbf{x})$ pueden ser dos funciones nolineales dadas y c una constante cuyo valor puede cambiar. Se puede ver que el problema anterior es equivalente a resolver la siguiente ecuación:

$$c = f(\mathbf{x}) - g(\mathbf{x}) = h(\mathbf{x}) \tag{14}$$

Problema que se puede resolver con la metodología anterior.

Busqueda de datos en una tabla e interpolación

Si consideramos una *look-up table* discreta de una dimensión con coordenadas $\mathbf{x} = \{x_1, x_2, x_3 .... x_n\}^T$ que mapea valores a otro vector $\mathbf{y} = \{y_1, y_2, y_3 .... y_n\}^T$.

Busqueda de datos en una tabla e interpolación

Si consideramos una *look-up table* discreta de una dimensión con coordenadas $\mathbf{x} = \{x_1, x_2, x_3....x_n\}^T$ que mapea valores a otro vector $\mathbf{y} = \{y_1, y_2, y_3....y_n\}^T$.La intepolación con **k**-vector es primero iniciada construyendo el **k**-vector fuera del procesamiento en tiempo real. En este caso el **k**-vector esta dado por $\mathbf{k} = \{k_1, k_2, k_3....k_n\}^T$ donde k_j es el número de valores de x que estan por debajo de la linea que une los puntos $[1, -\delta]$ $[n, x_n + \delta]$ al j-ésimo indice. Entonces el indice del **k**-vector es calculado de acuerdo a:

Busqueda de datos en una tabla e interpolación

Si consideramos una *look-up table* discreta de una dimensión con coordenadas $\mathbf{x} = \{x_1, x_2, x_3....x_n\}^T$ que mapea valores a otro vector $\mathbf{y} = \{y_1, y_2, y_3....y_n\}^T$.La intepolación con **k**-vector es primero iniciada construyendo el **k**-vector fuera del procesamiento en tiempo real. En este caso el **k**-vector esta dado por $\mathbf{k} = \{k_1, k_2, k_3....k_n\}^T$ donde k_j es el número de valores de x que estan por debajo de la linea que une los puntos $[1, -\delta]$ $[n, x_n + \delta]$ al j-ésimo indice. Entonces el indice del **k**-vector es calculado de acuerdo a:

$$j = floor(\frac{x^* - \delta - q}{m}) \tag{15}$$

Busqueda de datos en una tabla e interpolación

Si consideramos una *look-up table* discreta de una dimensión con coordenadas $\mathbf{x} = \{x_1, x_2, x_3....x_n\}^T$ que mapea valores a otro vector $\mathbf{y} = \{y_1, y_2, y_3....y_n\}^T$.La intepolación con **k**-vector es primero iniciada construyendo el **k**-vector fuera del procesamiento en tiempo real. En este caso el **k**-vector esta dado por $\mathbf{k} = \{k_1, k_2, k_3....k_n\}^T$ donde k_j es el número de valores de x que estan por debajo de la linea que une los puntos $[1, -\delta]$ $[n, x_n + \delta]$ al j-ésimo indice. Entonces el indice del **k**-vector es calculado de acuerdo a:

$$j = floor(\frac{x^* - \delta - q}{m}) \tag{15}$$

Donde $m = \frac{x_n - x_1 + 2\delta}{(n-1)}$. Entonces, para calcular el valor interpolado y^*, se usa alguno de los métodos conocidos de interpolación (lineal, más alto orden, spline, spline cúbico..) para aproximar entre x_{k_j} y $x_{k_{j+1}}$.

Realizamos una implementación del algoritmo descripto en (2)basados en el preprocesamiento y la búsqueda en si, mediante dos funciones que automatizan el método. Obtuvimos resultados muy buenos en varios sentidos comparados con los convencionales:

Realizamos una implementación del algoritmo descripto en (2)basados en el preprocesamiento y la búsqueda en si, mediante dos funciones que automatizan el método. Obtuvimos resultados muy buenos en varios sentidos comparados con los convencionales:

Implementación preprocesamiento y busqueda de raíces

Implementación preprocesamiento y busqueda de raíces

Como ejemplo de salida típica de la implementación para la función $y = sinc(x)$ en $[-5, 5]$ es

```
octave:3> nonlinear_inversion
Raiz:1 ---> -5
Raiz:2 ---> -4
Raiz:3 ---> -3
Raiz:4 ---> -2
Raiz:5 ---> -1
Raiz:6 ---> 1
Raiz:7 ---> 2
Raiz:8 ---> 3
Raiz:9 ---> 4
Raiz:10 ---> 5
elapsed_time =  0.033457
```

Utilizando las mismas funciones de la implementación anterior, realizamos un script que obtiene las intersecciones de dos funciones no-lineales, $sinc(t)$ y $sin(t)$ y el resultado que obtuvimos fue el siguiente:

Resultado

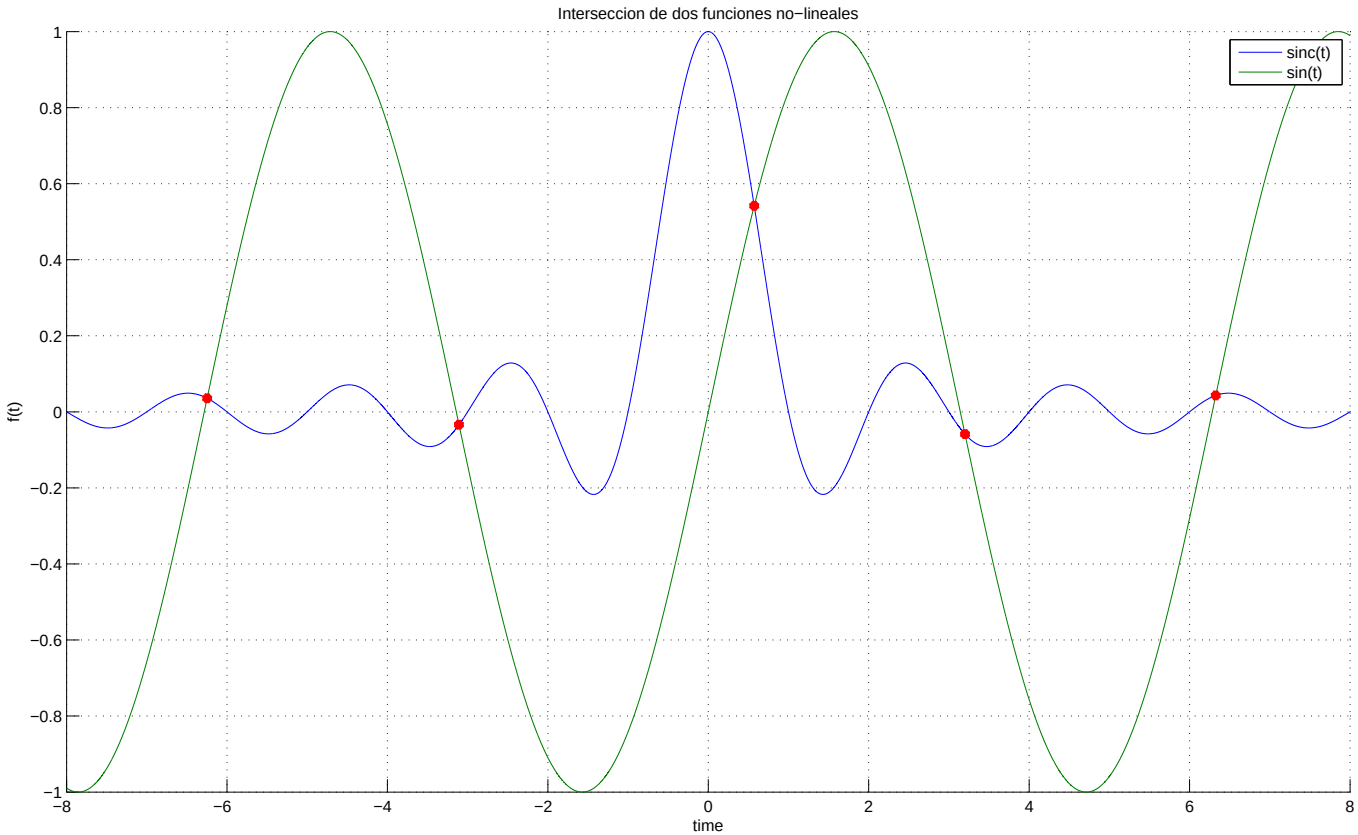


Figura: Solución a la intersección de dos funciones no-lineales

Prueba con una look-up table

Para verificar que la magnitud de la base de datos no afecta considerablemente a la velocidad de resolución, realizamos un script que genera una base de datos variable y una búsqueda en particular con el método antes mencionado, a cada búsqueda es tomado el tiempo de resolución y luego se grafica Los resultados fueron:

Resultado

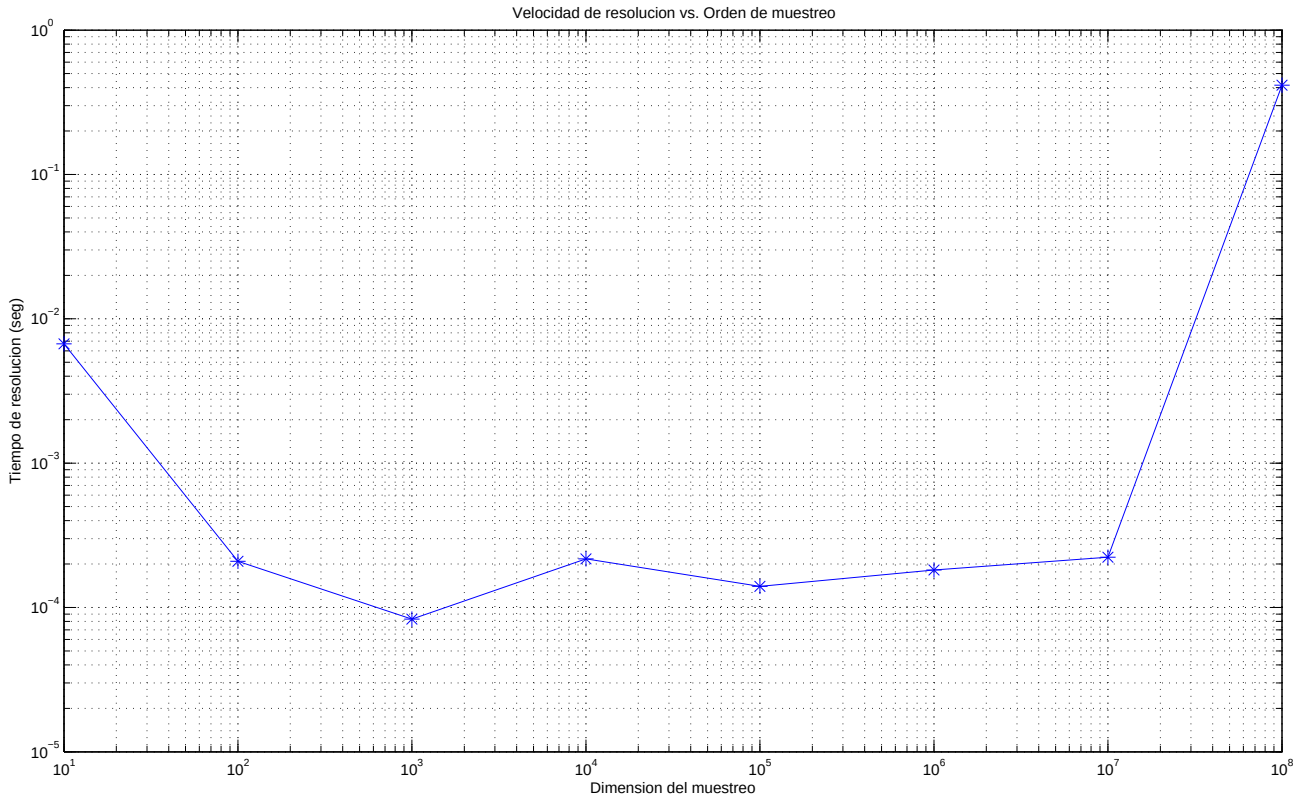


Figura: Velocidad de resolución vs orden de la base de datos

Se puede afirmar que es indiscutible la efectividad de dicho método comparado con otras metodologías para encontrar raíces en un determinado intervalo. Ésto se debe a que determinadas búsquedas de raíces, como ser el método de Newton-Raphson sin ser optimizado por el **k**-vector, encuentran únicamente una raíz de todas las posibles en dicho intervalo con una gran cantidad de iteraciones. Por otra parte este método posee una gran precisión y obtiene todas las raíces de la función en el intervalo solicitado.

Como se puede observar en la figura 3, es factible encontrar los puntos de intersección de dos funciones no-lineales bajo el método **k**-vector. Éste proceso lleva un leve costo en cuanto a la programación, debido a que es posible implementarlo directamente utilizando la búsqueda de raíces de funciones no-lineales: Al restar las funciones entre si, se genera una nueva función, y se le hallan a ésta sus ceros. Éstos ceros van a representar las coordenada de las intersecciones de las funciones originales. Como se menciono con anterioridad, el método de búsqueda de raíces es muy efectivo tanto en tiempo de resolución como en precisión. Al ser el método para encontrar las intersecciones de dos funciones una variante del método anterior, éste posee las mismas características.

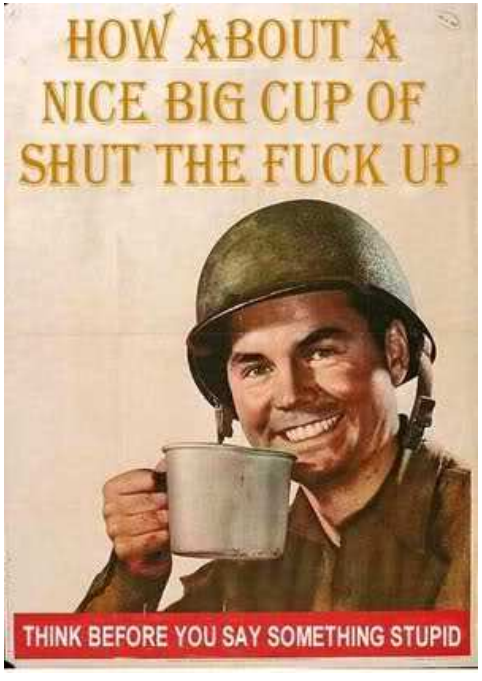
Al observar la Figura (4) se puede ver que el método de interpolación utilizado es muy efectivo, no solo cuando se necesita un método de resolución rápido, sino también para situaciones en las cuales el problema conlleva una gran serie de datos independientes, obtenidos empíricamente. Éste tipo de metodología es recomendable cuando se enfrenta a situaciones en las cuales la base de datos cambia continuamente, ya que el método posée un tiempo de resolución muy inferior a otros métodos cuando las bases de datos muestreados es considerablemente grande. Éste tipo de base de datos no afecta al método del **k**-vector en términos de eficiencia ni de velocidad, como puede observarse en la figura (4).

Referencias

 A **k**-vector approach to sampling, interpolation, and aproximation. Daniele Mortari and Jonathan Rogers

 **k**-vector Range searching techniques. Daniele Mortari, Beny Neta

Preguntas?


HOW ABOUT A
NICE BIG CUP OF
SHUT THE FUCK UP
THINK BEFORE YOU SAY SOMETHING STUPID

Gracias ;)