La Matriz de Hilbert y su Inversa – Cálculo por medio de Maple, Mathemática, Gauss, Matlab y Macros en Excel

Jorge Mauricio Oviedo [1]

Resumen: El presente trabajo tiene por objetivo brindar un enfoque teórico accesible sobre Ecuaciones Diferenciales y su implementación diversos Softwares. Para llevar a cabo dicha tarea se presenta una revisión teórica de tal tópico y se procede a implementarlo en todo tipo de software tanto algebraico y numérico. Se exponen y se comparan alternativamente las resoluciones numéricas contra las algebraicas evaluando la eficiencia en cada programa. Se proveen además los códigos de programación usados en cada Software.

Palabras clave: Matriz de Hilbert, Inversa, Métodos Numéricos, Runge-kutha, Maple, Mathematica, Matlab, Gauss, Excel, Macros, Visual Basic

[1] joviedo@eco.unc.edu.ar

INTRODUCCIÓN TEÓRICA

Las matrices de Hilbert se caracterizan porque el patrón de generación de sus elementos responde a la siguiente estructura:

$$[a_{ij}] = \left(\frac{1}{i + j - 1} \right)$$

Es decir una matriz de Hilbert de orden n luce de la siguiente manera:

$$H_N = \left[a_{ij} \right]_N = \begin{pmatrix} 1 & 1/2 & \cdots & 1/N \\ 1/2 & 1/3 & \cdots & 1/N+1 \\ \vdots & \vdots & \text{O} & \vdots \\ 1/N & 1/N+1 & \cdots & 1/2N-1 \end{pmatrix}$$

La característica esencial de esta matriz es que su determinante se va aproximando a cero cuando el orden de la misma va en aumento. Mas formalmente se puede expresar lo anterior diciendo:

$$\lim_{n \to \infty} \det(H_n) = 0$$

Esta particularidad hace que el calculo de su inversa se sumamente dificultoso a medida de que aumente el orden la matriz a raíz de que cada vez dicha matriz esta mas cerca de ser singular. En efecto, todos los programas numéricos de computación de punto flotante cometen considerables errores de precisión al intentar invertir dicha matriz. Sin embargo, softwares como Mathematica y Maple, que trabajan en forma simbólico-algebraica, salen airosos ante este desafío a raíz de que no emplean punto flotante en las representaciones numéricas. En su lugar utilizan expresiones algebraicas tratadas como objetos lógicos tales y como operaría un verdadero matemático experto (que no esta dispuesto a trabajar con expresiones decimales que pierden exactitud si no con expresiones algebraicas fracciones, radicales, números trascendentales, etc.).

La intención de este ejercicio es evaluar la perdida de precisión en que incurren los programas numéricos de punto flotante en el proceso de invertir dicha matriz. A continuación se expone el enunciado del Ejercicio Nº 7 e inmediatamente se pasa a resolverlo mediante el empleo de todo tipo de Software numérico junto a una posterior comparación con el software Maple 6[2] que trabaja en forma algebraica.

Dado el sistema de ecuaciones A x = b, se le pide generar el vector b teniendo en cuenta que la matriz A es la matriz de Hilbert de dimensión 11. El vector x, por su parte, es igual a 1 para la primera componente,2 para la segunda, 3 para la tercera y así sucesivamente hasta llegar a 11 para la undécima. Calcular el vector solución x

[2] A raíz de que Mathematica y Maple trabajan exactamente del mismo modo se optó por incluir únicamente las salidas de Maple a los efectos de no ser redundante innecesariamente.

para el sistema de ecuaciones así planteado y comparar los resultados obtenidos con el vector de soluciones conocido. Realizar el ejercicio en Gauss y en Excel. Comentar.

Como puede apreciarse el hecho de poder generar el vector de términos independientes conociendo primero el vector solución y después el recalculo del vector solución vía el método de la inversa es una magnífica ejemplificación de los errores de redondeo que se pueden llegar a cometer por medio de un software numérico. Es decir, primero se generará el vector de términos independientes *b* mediante la siguiente expresión:

$$\mathbf{H_{11}X = b}$$

donde H_{11} es la matriz de Hilbert de orden once mientras que X es el vector solución que se conoce de antemano de acuerdo a las condiciones del ejercicio. Una vez obtenido el vector de términos independientes se puede proceder a resolver el sistema de ecuaciones para hallar el valor de X mediante la siguiente fórmula:

$$\mathbf{X = H_{11}^{-1}b}$$

Como puede apreciarse, si en el proceso de hallar la matriz inversa se trabajasen con valores algebraico-simbólicos (números fraccionarios) se arribaría al mismo valor para X con el cual se generó b en el proceso anterior. Sin embargo en la medida que se usan aproximaciones numéricas para representar tales fracciones se incurrirán en significantes errores de redondeo que sumados a los errores de acumulación en los algoritmos internos de los sofwares para el cálculo de la inversa hacen que los resultados se aparten considerablemente de los valores correctos originales. Además poseen una seria limitación al trabajar con punto flotante ya que la máxima precisión que pueden alcanzar es solamente 16 dígitos en una computadora de 32 bits. En casos mas generales cuando se tratan de matrices comunes tales errores de redondeo y de los algoritmos suelen ser despreciables sin embargo cuando se está ante la presencia de sistemas mal comportados como en el caso de este ejercicio los resultados serán sumamente grandes.

Entonces, como se dijo, si se utilizara la verdadera inversa, los vectores x y b de ambas expresiones deberían ser los mismos, pero los softwares numéricos utilizan una aproximación de la inversa de la matriz, por lo que el resultado obtenido en realidad es:

$$\mathbf{X' = (H_{11}')^{-1}b}$$

A los efectos de contar con un indicador de la magnitud del error se podría definir el error del programa por medio de la siguiente expresión:

$$\mathbf{E = X - X'}$$

A continuación se exponen las programaciones realizadas en los diversos Softwares:

EXCEL 2002

Para generar una Matriz de Hilbert de orden *n* se recurrió a utilizar los comandos de Visual Basic programando en los módulos de los mismos mediante el desarrollo de macros. A continuación se expone tal programación:

```
   Sub hilbert()
   '
   ' hilbert Macro
   ' Macro escrita el 27/08/2002 por JORGE MAURICIO OVIEDO
   '

   '
   n = Cells(1, 1).Value          'en la celda A1 debe aparecer el
orden'

   Range(Cells(2, 2), Cells(255, 255)).ClearContents
   Range(Cells(2, 2), Cells(255, 255)).ClearFormats

   For i = 1 To n
    For j = 1 To n
      Cells(1 + i, 1 + j).Value = 1 / (i + j - 1)
    Next
   Next

    Range(Cells(2, 2), Cells(1 + n, 1 + n)).Select
       Selection.Borders(xlDiagonalDown).LineStyle = xlNone
       Selection.Borders(xlDiagonalUp).LineStyle = xlNone
       With Selection.Borders(xlEdgeLeft)
           .LineStyle = xlContinuous
           .Weight = xlMedium
           .ColorIndex = xlAutomatic
       End With
       With Selection.Borders(xlEdgeTop)
           .LineStyle = xlContinuous
           .Weight = xlMedium
           .ColorIndex = xlAutomatic
       End With
       With Selection.Borders(xlEdgeBottom)
           .LineStyle = xlContinuous
           .Weight = xlMedium
           .ColorIndex = xlAutomatic
       End With
       With Selection.Borders(xlEdgeRight)
           .LineStyle = xlContinuous
           .Weight = xlMedium
           .ColorIndex = xlAutomatic
       End With
       Selection.Borders(xlInsideVertical).LineStyle = xlNone
       Selection.Borders(xlInsideHorizontal).LineStyle = xlNone


   Cells(1, 1).Select

   End Sub
```

Haciendo uso del programa anterior para calcular la matriz de Hilbert y hacienda uso tambien del comando MINVERSA se calculó la matriz inversa de Hilbert de orden once. Sin embargo la inversa solo puede calcularse si la matriz H_{11} es no singular por lo

que un requisito previo que debe cumplir dicha matriz es que su determinante sea distinto de cero. Utilizando el comando Mdeterm, Excel nos proporciona un valor del determinante de H_{11} de 3,02712E-65; por lo que si bien estamos en condiciones de obtener su inversa, es notable lo cercano a cero que es su determinante.

El valor de la inversa de Hilbert de orden once se expone a continuación junto al valor del vector de términos independientes, calculado como se explico en líneas anteriores, quedando en consecuencia el siguiente producto para calcular el valor estimado de X^3:

.

1.21E+2	-7.25E+3	1.41E+5	-1.3E+06	6.92E+06	-2.2E+07	4.48E+07	-5.8E+07	4.56E+07	-2E+07	3.87E+06	1
-7.25E+3	5.80E+5	-1.27E+7	1.27E+08	-2.28E+09	6.92E+8	-6.14E+09	4.70E+9	-2.21E+09	4.92E+9	-4.25E+8	.897
1.41E+5	-1.27E+7	2.98E+8	-3.1E+09	1.74E+10	-5.8E+10	1.22E+11	-1.6E+11	1.31E+11	-5.9E+10	1.15E+10	.640
-1.32E+6	1.27E+8	-3.29E+9	3.08E+10	-6.45E+11	1.9E+11	-1.8E+12	1.4E+12	-6.81E+12	1.5E+12	-1.3E+11	.745
6.92E+6	-6.92E+8	1.74E+10	-1.9E+11	1.10E+12	-3.8E+12	8.17E+12	-1.1E+13	9.05E+12	-4.1E+12	8.13E+11	.060
-2.21E+7	2.28E+9	-6.45E+10	5.8E+11	-1.33E+13	3.8E+12	-3.90E+13	2.9E+13	-1.49E+13	3.2E+13	-2.9E+12	.513
4.48E+7	-4.70E+9	1.22E+11	-1.4E+12	8.17E+12	-2.9E+13	6.27E+13	-8.6E+13	7.11E+13	-3.2E+13	6.51E+12	.063
-5.76E+7	6.14E+9	-1.83E+11	1.6E+11	-3.90E+12	1.1E+13	-1.17E+13	8.6E+13	-4.5E+13	9.8E+13	-9E+12	.684
4.56E+7	-4.92E+9	1.31E+11	-1.4E+12	9.05E+12	-3.2E+13	7.11E+13	-9.8E+13	8.21E+13	-3.8E+13	7.62E+12	.361
-2.03E+7	2.21E+9	-6.81E+10	5.9E+10	-1.5E+12	4.1E+12	-4.55E+13	3.3E+13	-1.79E+13	3.8E+13	-3.6E+12	.081
3.87E+6	-4.26E+8	1.15E+10	-1.3E+11	8.13E+11	-2.9E+12	6.51E+12	-9E+12	7.62E+12	-3.6E+12	7.15E+11	.836

Haciendo el producto indicado se arriba al siguiente valor para la estimación de X junto a su error de estimación:

X^1	E
0.984	0.01586
2.063	-0.06349
2.396	0.60416
5.41	-1.41644
4.347	0.65137

[3] Por un problema de exposición solo se incluyen algunos dígitos de los valores a los efectos de que la matriz quepa en una misma línea.

5.322	0.67773
7.684	-0.68359
7.840	0.16016
9.039	-0.03906
9.973	0.02734
11.006	-0.00586

Como se aprecia, los valores de los errores son bastante significativos. Ello se debe a las diferencias significativas existentes entre el vector x real y el estimado por el método de la inversa.

GAUSS LIGHT 4.0

A continuación se utilizó Gauss Light 4.0 para analizar nuevamente el problema de la matriz de Hilbert.

Al ejecutarse, el programa le solicita al operador que le introduzca como un imput la dimensión de la matriz. Al introducir un número entero, el programa generará una matriz de Hilbert de dicha dimensión, su inversa y generará un vector "*x*" de dimensión *N* como una secuencia aditiva desde 1 hasta N.

A continuación resolverá el sistema lineal original $H_N X$ para obtener el vector *b* (que también será de dimensión *N*) y por último utilizará este vector *b* resultante para formar un nuevo sistema, aplicando el método de la inversa, para estimar el vector "*x*" y así obtener un nuevo vector *E* (de dimensión *N*) que nos muestre el error en que incurrió el programa al estimar "*x*". Ello nos dará una medida de la precisión de Gauss Light 4.0 para obtener la inversa de la matriz de Hilbert.

El programa escrito fue:

```
   cls;
   format(16,16);
   "¿Dimención de la matriz de Hilbert?";
   c=con(1,1);
   h=zeros(c,c);
   for i (1, c, 1);    /*bucle para generar la Matriz de
Hilbert*/
       for j (1, c, 1);
          h[i,j]=1/(i+j-1);
       endfor;
   endfor;
```

```
"Inversa de Hilbert";
h;
x=seqa(1,1,c); /*vector X*/
b=h*x;         /*Sistema original*/
print;
xc=inv(h)*b;    /*Método de la inversa*/
print;
print "     b        x    invh*b=x^ E=x-x^";
b~x~xc~x-xc;
```

El output obtenido consiste en:

```
¿Dimención de la matriz de Hilbert?
? 11
Inversa de Hilbert

 1.0000  0.50000  0.33333  0.25000  0.20000  0.16667  0.14286  0.12500  0.11111  0.10000
0.090909
 0.50000  0.33333  0.25000  0.20000  0.16667  0.14286  0.12500  0.11111  0.10000 0.090909
0.08333
 0.33333  0.25000  0.20000  0.16667  0.14286  0.12500  0.11111  0.10000 0.090909 0.08333
0.07692
 0.25000  0.20000  0.16667  0.14286  0.12500  0.11111  0.10000 0.090909 0.08333  0.07692
0.07143
 0.20000  0.16667  0.14286  0.12500  0.11111  0.10000 0.090909 0.08333  0.07692  0.07143
0.06667
 0.16667  0.14286  0.12500  0.11111  0.10000 0.090909 0.08333  0.07692  0.07143  0.06667
0.06250
 0.14286  0.12500  0.11111  0.10000 0.090909 0.08333  0.07692  0.07143  0.06667  0.06250
0.05882
 0.12500  0.11111  0.10000 0.090909 0.08333  0.07692  0.07143  0.06667  0.06250  0.05882
0.05556
 0.11111 0.10000 0.090909  0.08333  0.07692  0.07143  0.06667  0.06250  0.05882  0.05556
0.05263
 0.10000 0.090909 0.08333  0.07692  0.07143  0.06667  0.06250  0.05882  0.05556  0.05263
0.05000
 0.090909 0.08333 0.07692  0.07143  0.06667  0.06250  0.05882  0.05556  0.05263  0.05000
0.047619


     b         x    invh*b=x^     E=x-x^
11.000    1.0000    1.0199    -0.019892
8.8968    2.0000    2.0017    -0.001662
7.6397    3.0000    2.9923     0.007736
6.7453    4.0000    3.9435     0.056519
6.0604    5.0000    5.1035    -0.103520
5.5130    6.0000    5.7402     0.259770
5.0627    7.0000    7.5469    -0.546880
4.6842    8.0000    7.3203     0.679690
4.3609    9.0000    9.5352    -0.535160
4.0811    10.000    9.7344     0.265630
3.8361    11.000    11.014    -0.014160
```

La primera línea muestra la solicitud del programa por el imput que debe introducir el operador, la segunda línea es introducida por el operador y representa la dimensión del sistema de ecuaciones y el resto de la salida corresponde a la información solicitada al crear el programa.

MATLAB 5.3

Matlab tiene la particularidad de contar con algoritmos especialmente diseñados a la hora de calcular la matriz de Hilbert como su inversa. Pero antes de hacer uso e

tales algoritmos se efectuarán los mismos procedimientos descriptos en secciones anteriores.

A continuación se resuelve el ejercicio haciendo uso de un programa escrito usando solo comandos generales para la creación e inversión de matrices

```
n=11;
h=zeros(n,n);
 for i=1:n       %bucle para generar la Matriz de Hilbert%
    for j=1:n;
       h(i,j)=1/(i+j-1);
    end;
 end;
x=1:1:n;         %vector X%
b=h*x';          %Sistema original%
xc=inv(h)*b;     %Método de la inversa%
d=x'-xc;
[b,x',xc,d]
```

b	X	X1	E
11	1	1.00000003352761	-3.35276126861572e-008
8.89678932178932	2	1.99999690055847	3.09944152832031e-006
7.63973248973249	3	3.00020599365234	-0.00020599365234375
6.74531302031302	4	3.99774169921875	0.00225830078125
6.06041736041736	5	5.0166015625	-0.0166015625
5.5130217005217	6	5.96875	0.03125
5.06268486415545	7	7.109375	-0.109375

4.6842434526258 0.0703125	8	7.9296875	
4.36093988570769 0.00390625	9	8.99609375	
4.08105737142115 0.068359375	10	9.931640625	
3.83609549205524 0.0166015625	11	11.0166015625	-

Como se anticipó en parrafos anteriores Matlab cuenta también con algoritmos especiales para la creación (comando `hilb`) e inversión (comando `invhilb`) de matrices de Hilbert. Se reproducen las programacaiones que utiliza dicho Software a los efectos de mostrar como los creadores de Matlab lo han implementado (correponde al archivo Hilb.m)

```
function H = hilb(n)
%HILB   Hilbert matrix.
%   HILB(N) is the N by N matrix with elements 1/(i+j-1),
%   which is a famous example of a badly conditioned matrix.
%   See INVHILB for the exact inverse.
%
%   This is also a good example of efficient MATLAB programming
%   style where conventional FOR or DO loops are replaced by
%   vectorized statements.  This approach is faster, but uses
%   more storage.

%   C. Moler, 6-22-91.
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.6 $  $Date: 1997/11/21 23:28:56 $

%   I, J and E are matrices whose (i,j)-th element
%   is i, j and 1 respectively.

J = 1:n;
J = J(ones(n,1),:);
I = J';
E = ones(n,n);
H = E./(I+J-1);
```

```
function H = invhilb(n)
%INVHILB Inverse Hilbert matrix.
%   INVHILB(N) is the inverse of the N by N matrix with elements
%   1/(i+j-1), which is a famous example of a badly conditioned
```

```
%   matrix.  The result is exact for  N  less than about 15.
%
%   See also HILB.

%   C. Moler, 4-30-87.
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.6 $  $Date: 1997/11/21 23:28:59 $

p = n;
H = zeros(n,n);
for i = 1:n
    if i > 1, p = ((n-i+1)*p*(n+i-1))/(i-1)^2; end
    r = p*p;
    H(i,i) = r/(2*i-1);
    for j = i+1:n
        r = -((n-j+1)*r*(n+j-1))/(j-1)^2;
        H(i,j) = r/(i+j-1);
        H(j,i) = r/(i+j-1);
    end
end
```

Bien, haciendo uso de los comandos anteriores se resuelve nuevamente el ejercicio con estos nuevos algoritmos a la espera de una mayor precisión. Se presentan a continuación tales programaciones:

```
n=11;
h=hilb(n);
x=1:1:n;              %vector X%
b=h*x';               %Sistema original%
xc=invhilb(n)*b;      %Método de la inversa%
d=x'-xc;
[b,x',xc,d]
```

b	X	X1	E
11	1	1.00000005029142	-5.02914190292358e-008
8.89678932178932	2	1.99999165534973	8.34465026855469e-006
7.63973248973249	3	3.00018310546875	-0.00018310546875
6.74531302031302	4	3.9990234375	0.0009765625

6.06041736041736 0.0078125	5	5.0078125	
5.5130217005217 0.03125	6	5.96875	
5.06268486415545 0.0625	7	7.0625	-
4.6842434526258 0.09375	8	7.90625	
4.36093988570769 0.0625	9	9.0625	-
4.08105737142115 0.041015625	10	9.958984375	
3.83609549205524 0.00927734375	11	11.00927734375	-

Como puede dilucidarse los resultados son realmente mejores con la aplicación de tales algoritmos especialmente diseñados por los fabricantes de Matlab.

MAPLE 6.0

Por último, se presentaran las programaciones efectuadas en un software con características distintivas en cuanto a su modo de operar. Como adelantó desde el comienzo este software algebraico-simbólico no comete errrores cuando trabaja en forma exacta. para ello se muestran ls programaciones en el mismo junto a las salidas que arrojó.

```
with(linalg):
n:=11:
x:=[seq(i, i=1..n)]:
b_gen_exact:=evalm(hilbert(n)&*x);
x_aprox:=evalm(inverse(hilbert(n))&*b_gen_exact);
Error:=evalm(x-x_aprox);
```

b_gen_exact :=

$$\left[11, \frac{246619}{27720}, \frac{1376527}{180180}, \frac{810247}{120120}, \frac{545983}{90090}, \frac{794669}{144144}, \frac{2067641}{408408}, \frac{546595}{116688}, \frac{8459953}{1939938}, \frac{105559977}{25865840}, \frac{89301449}{23279256}\right]$$

x_aprox := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Error := [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

>

Se desprende de lo anterior que Maple sale airoso ante el desafío de invertir una matriz de Hilbert ya que la opera en forma exacta mediante el uso de representaciones fraccionarias exactas de los números y no como punto flotante. En consecuencia tampoco tiene el problema de los 16 dígitos de precisión en computadoras de 32 bits.

Adicionalmente Maple posee la opción de trabajar con variables de precisión aritmética. Éstas son representaciones de los números con un numero arbitrario de dígitos manejados a discreción del usuario del programa mediante el comando eval.. A continuación se exponen la resolución del problema trabajando con una precisión de 70 dígitos.

```
> with(linalg):
  n:=11:
  precision:=70:
  x:=[seq(i, i=1..n)]:
  b_gen_exact:=evalm(hilbert(n)&*x);
  b_gen_numer:=evalf(evalm(hilbert(n)&*x),precision);
  x_aprox:=evalf(evalm(inverse(hilbert(n))&*b_gen_numer),precision);
  evalf(evalm(x-x_aprox),precision);
>
```

b_gen_exact :=

$$\left[11, \frac{246619}{27720}, \frac{1376527}{180180}, \frac{810247}{120120}, \frac{545983}{90090}, \frac{794669}{144144}, \frac{2067641}{408408}, \frac{546595}{116688}, \frac{8459953}{1939938}, \frac{105559977}{25865840}, \frac{89301449}{23279256}\right]$$

b_gen_numer := [11., 8.896789321789321789321789321789321789321789321789321789321789321789322,

7.639732489732489732489732489732489732489732489732489732489732489732490,

6.745313020313020313020313020313020313020313020313020313020313020313020,

6.060417360417360417360417360417360417360417360417360417360417360417360,

5.513021700521700521700521700521700521700521700521700521700521700521701,

5.062684864155452390746508393567217096628861334743687684864155452390747,

4.684243452625805566982037570272864390511449334978746743452625805566982,

4.360939885707687565272704591590040506449175179825334624096234003354746,

4.081057371421148510931792665538795569755322077303501452108263253774090,

3.836095492055244377225801374408185553696389609702303200755213139114068]

$x\_aprox := [.99999999999999999999999999999999999999999999999999999999999990,$

2.000000000000000000000000000000000000000000000000000000000002,
3.000000000000000000000000000000000000000000000000000000000010,
4.000000000000000000000000000000000000000000000000000000000000,
5.000000000000000000000000000000000000000000000000000000000000,
6.000000000000000000000000000000000000000000000000000000000001,
7.000000000000000000000000000000000000000000000000000000000000,
8.000000000000000000000000000000000000000000000000000000000010,
8.999999999999999999999999999999999999999999999999999999999990,
9.999999999999999999999999999999999999999999999999999999999998,
11.000000000000000000000000000000000000000000000000000000000003]

$[.10\ 10^{-60}, -.2\ 10^{-59}, -.10\ 10^{-57}, 0, 0, -.1\ 10^{-55}, 0, -.10\ 10^{-54}, .10\ 10^{-54}, .2\ 10^{-55}, -.3\ 10^{-56}]$

Como se aprecia la ultima fila corresponde a los errores por haber usado 30 dígitos de precisión. Evidentemente la magnitud del error es sumamente pequeña superando enormemente a los demás softwares utilizados anteriormente.

COMPARACIÓN DE RESULTADOS

Para estar en condiciones de comparar los resultados obtenidos, se repiten a continuación los vectores de errores de estimación ***E*** que nos proporcionaron Excel, Gauss, Matlab y Maple respectivamente:

Excel	Gauss	Matlab	Matlab Alg. especial	Maple 70 dígitos	Maple exacto
0.01586	-0.019892	-3.352761e-008	-5.029141902e-008	$0.10 * 10^{-60}$	0
-0.06349	-0.001662	3.0994415e-006	8.3446502e-006	$-0.2 * 10^{-59}$	0

0.60416	0.007736	-0.0002059936	-0.000183105	$-0.1*10^{-57}$	0
-1.41644	0.056519	0.00225830078	0.0009765625	0	0
0.65137	-0.103520	-0.0166015625	0.0078125	0	0
0.67773	0.259770	0.03125	0.03125	$-0.1*10^{-55}$	0
-0.68359	-0.546880	-0.109375	-0.0625	0	0
0.16016	0.679690	0.0703125	0.09375	$-0.1*10^{-55}$	0
-0.03906	-0.535160	0.00390625	-0.0625	$-0.1*10^{-55}$	0
0.02734	0.265630	0.068359375	0.041015625	$0.2*10^{-55}$	0
-0.00586	-0.014160	-0.0166015625	-0.00927734375	$-0.3*10^{-56}$	0

En los primeros siete elementos del vector "x", el error incurrido por Gauss es menor (en valor absoluto) que el incurrido por Excel, sin embargo, en las últimas cuatro dicha relación se invierte. Por lo tanto no podemos establecer en forma inequívoca que programa logró una mejor aproximación. Algo similar ocurre con la comparación entre los algoritmos de Matlab. En este sentido se necesario definir un estimador que condense toda la información de cada elemento en un valor único. Para ello, utilizó como indicadores la sumatoria de los valores absolutos de los elementos del vector E de error y la sumatoria de los elementos al cuadrado de del vector E de error. Los valores de ambos indicadores fueron:

Indicador	Excel	Gauss	Matlab	Matlab Alg. especial	Maple 70 dígitos	Maple exacto
$\mathbf{E} = \sum_{i=1}^{11} \left\| \mathbf{X}_i - \mathbf{X}_i \right\|$	4.34507	2.49059	0.31887367	0.30927353	$0.2914*10^{-57}$	0

Como se aprecia Gauss resulta superior a Excel, Matlab superior a Gauss, El algoritmo especial para la matriz de Hilbert es superior al comando común de Matlab y por último Maple, como se anticipó desde un principio, resultó superior a todos no solo cuando se toman un número arbitrario de dígitos de precisión (70 en este caso) si no que además puede resolver estos problemas con total exactitud. La desventaja de los programas simbólicos radica en que son relativamente mas lentos que los numéricos tradicionales, desventaja que suele ser importante cuando se trabajan con simulaciones que requieren un elevado numero de operaciones. Como conclusión se puede establecer que existe un trade-off entre precisión-exactitud y velocidad de cálculo. La elección de uno u otro programa dependerá del tipo de problema que se enfrente el usuario y de su inclinación hacia la exactitud o hacia la velocidad.

EXTENSIONES Y GENERALIZACIONES

El objetivo de esta sección es extender las conclusiones hacia órdenes mas elevados de la matriz de Hilbert. Para ello se construirán programas en Matlab y Maple[4] por medios de bucles que permitan calcular la suma los valores absolutos de la diferencia de los errores en la estimación para diversos tamaños de H desde 10 hasta 20

Matlab 5.3

[4] A raíz de que Matlab resultó ser el programa mas eficiente dentro de los programas numéricos se decide utilizar solo este en comparación con Maple dejando de lado a los demás (Excel y Gauss).

```
k=20;
 for n=10:k;
    h=hilb(n);
    x=1:1:n;          %vector X%
    b=h*x';           %Sistema original%
    xc=invhilb(n)*b;     %Método de la inversa%
    d=x'-xc;
    S=sum(abs(d));
    [n,S]

   end
```

```
   ans =

                         10        0.00646472070366144


   ans =

                         11          0.309273531660438


   ans =

                         12           21.5332416296005


   ans =

                         13            590.42208981514


   ans =

                         14            9873.1360013485


   ans =

                         15           572269.797637939


   ans =

                         16           17055944.0080261


   ans =

                         17           532645947.208221


   ans =
```

```
                          18          20966386421.6324

ans =

                          19          692985626578.771

ans =

                          20          43016990495564.4
```

Maple 6.0

```
with(linalg):

for i from 10 by 1 while i < 21 do
n:=i:
precision:=70:
x:=[seq(i, i=1..n)]:
b_gen_exact:=evalm(hilbert(n)&*x):
b_gen_numer:=evalf(evalm(hilbert(n)&*x),precision):
evalm(inverse(hilbert(n))&*b_gen_exact):
x_aprox:=evalf(evalm(inverse(hilbert(n))&*b_gen_numer),precision):
S:=evalf(sum(abs(x[j]-x_aprox[j]),j=1..n),precision):
print([N=i,Suma=S]);
od:
```

$$[N = 10, Suma = .291401\ 10^{-57}]$$

$$[N = 11, Suma = .23310210\ 10^{-54}]$$

$$[N = 12, Suma = .661103913\ 10^{-53}]$$

$$[N = 13, Suma = .1941300102\ 10^{-51}]$$

$$[N = 14, Suma = .9970031302\ 10^{-50}]$$

$$[N = 15, Suma = .53204264003\ 10^{-49}]$$

$$[N = 16, Suma = .1308360313200\ 10^{-46}]$$

$$[N = 17, Suma = .3105363995124\ 10^{-45}]$$

$$[N = 18, Suma = .174440964581091\ 10^{-43}]$$

$$[N = 19, Suma = .3091290086891\ 10^{-42}]$$

$$[N = 20, Suma = .21897017610537988\ 10^{-40}]$$

Como se dilucida de las salidas anteriores la suma de los valores absolutos de los errores en Matlab crece casi exponencialmente a medida que aumenta el orden de la matriz. En Maple, si bien los errores son casi despreciables (trabajando no con valores exactos si no con variables de precisión aritmética de 70 dígitos de precisión), los mismos aumentan también al aumentar el orden de la Matriz de Hilbert.