Reporte de laboratorio
Análisis numérico I

Alumno: Octavio Alberto Agustín Aquino
Profesora: M. C. Graciela Castro
Grupo: 305
Licenciatura en Matemáticas Aplicadas
Universidad Tecnológica de la Mixteca

5 de junio de 2005

Índice general

Capítulo 1

Planteamiento

1.1. Introducción

En diversas ramas de la Matemática es necesario hacer cálculos numéricos explícitos para obtener la solución de un problema particular. Actualmente dichos procesos numéricos se realizan con una computadora. Sin embargo, las computadoras son instrumentos finitos que no pueden representar cantidades arbitrariamente grandes o pequeñas, lo que implica que en cualquier cálculo tendremos una pérdida de precisión, dado el redondeo o truncamiento propio de la máquina o la finitud de los numéros máximo y mínimo que admiten representación. En el caso de la siguiente práctica veremos los errores asociados precisamente a operaciones con números muy pequeños y muy grandes.

1.2. Objetivo

Analizar el error asociado a la arimética de los computadores y el condicionamiento y estabilidad de los métodos numéricos.

1.3. Problema 1: Raíces un ecuación cuadrática

Escribir un programa en C que obtenga las raíces de la ecuación de segundo grado

$$ax^2 + bx + c = 0 \tag{1.1}$$

empleando la fórmula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.2}$$

siempre que $b^2 - 4ac \geq 0$. Comprobar que se produce una pérdida importante de dígitos significativos en aquellas ecuaciones de segundo grado para las que $b^2 \gg 4ac$. Proponga un método de resolución que se desempeñe bien en todos los casos. Explique el origen del error al utilizar la expresión (1.2)

1.4. Problema 2: Iteración para el cálculo de una sucesión

Sea y_n la sucesión definida por

$$y_n = \int_0^1 x^n e^x \, dx \quad (n \in \mathbb{N} \cup 0). \tag{1.3}$$

Comprobar que al integrar por partes se obtiene

$$y_{n+1} = e - (n+1)y_n. \tag{1.4}$$

Calcular explícitamente y_0 y comprobar que es

$$y_0 = e - 1,$$

Obtener $y_1, \ldots, y_{30}$ y verificar que se produce un error significativo al cálcular los términos más grandes de la sucesión obtiendo de manera exacta y_{30}. Explicar el origen del error.

Capítulo 2

Resolución

2.1. Problema 1

Consideremos la ecuación (1.1),

$$ax^2 + bx + c = 0 \quad a, b, c \neq 0 \quad b^2 - 4ac \geq 0.$$

Cuando $b^2 \gg 4ac$, entonces

$$b^2 - 4ac \approx b^2,$$

y así

$$\sqrt{b^2 - 4ac} \approx \sqrt{b^2} = |b|.$$

En la aritmética de la computadora incluso puede ocurrir la igualdad, dado el redondeo o que la contribución del termino $4ac$ no puede representarse. Al sustituir en (1.2) obtenemos

$$x_1 = 0, \quad x_2 = -\frac{b}{a}. \tag{2.1}$$

que naturalmente son las raíces de la ecuación (1.1) cuando $c = 0$. Por lo tanto, al evaluar el polinomio $ax^2 + bx + c$ en estos números obtenemos $c \neq 0$. La aproximación puede tomarse como buena si c es muy pequeño. Sin embargo, puede ocurrir que $4ac$ sea mucho menor que b^2 sin que c sea pequeño si $a \ll 1$, por lo que las aproximaciones (2.1) serían malas en tal caso. En gran medida ésta es la peor situación, así que optimizaremos el cálculo de las raíces cuando ocurra que $c > 1$ y $a \ll 1$.

Para minimizar el error por redondeo pueden racionalizarse el denominador y el numerador de (1.2).

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \frac{\mp b - \sqrt{b^2 - 4ac}}{\mp b - \sqrt{b^2 - 4ac}} = -\frac{2c}{b \pm \sqrt{b^2 - 4ac}} \tag{2.2}$$

Esto puede ser conveniente pues así el dividendo en la operación es más grande comparado con el divisor. Otra posibilidad es aproximar el valor de la raíz del

discriminante $D = b^2 - 4ac$. Para ello, notemos que podemos escribir (1.2) de la siguiente forma:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{b}{2a}\left(-1 \pm \sqrt{1 - \frac{4ac}{b^2}}\right).$$

Claramente el término $\frac{4ac}{b^2} \ll 1$, por lo que $1 - \frac{4ac}{b^2} \approx 1$. Precisamente, para $x \approx 1$ tenemos la siguiente aproximación lineal

$$\sqrt{x} \approx \frac{1}{2}(x + 1).$$

De aquí que

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \approx \frac{b}{2a}\left[-1 \pm \left(1 - \frac{2ac}{b^2}\right)\right]$$

Para reducir el error simplificamos a

$$\begin{aligned} x_1 &= -\frac{c}{b} \\ x_2 &= \frac{ac - b^2}{ab} \end{aligned} \tag{2.3}$$

Realizamos un programa en C que calcula las raíces con (1.2), (2.2), (2.3).

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<ctype.h>
#include<complex.h>

void main()
{
 double a,b,c,x1,x2,y1,y2,disc,r1,r2,val1,val2;
 char ans='s';
 while (ans!='n')
 {
  clrscr();
  printf("Introduzca a, b, c: ");
  scanf("%lf %lf %lf", &a, &b, &c);
  x1=x2=-b;
  disc=b*b-4*a*c;
  if (disc<0)
   {
    printf("Las raices no son reales.");
   }
  else
   {
```

```
    disc=sqrt(disc);
    x1=(x1+disc)/(2*a);
    x2=(x2-disc)/(2*a);
    y1=(-2*c)/(b+disc);
    y2=(-2*c)/(b-disc);
    r1=(-b/a)+(c/b);
    r2=-c/b;
    printf("Las raices son: %.20lf %.20lf \n", x1, x2);
    printf("Las raices son: %.20lf %.20lf \n", y1, y2);
    printf("Las raices son: %.20lf %.20lf \n\n", r1, r2);
    val1=(a*x1*x1)+(b*x1)+c;
    val2=(a*x2*x2)+(b*x2)+c;
    printf("Evaluacion 1: %.20lf %.20lf \n", val1, val2);
    val1=(a*y1*y1)+(b*y1)+c;
    val2=(a*y2*y2)+(b*y2)+c;
    printf("Evaluacion 2: %.20lf %.20lf \n", val1, val2);
    val1=(a*r1*r1)+(b*r1)+c;
    val2=(a*r2*r2)+(b*r2)+c;
    printf("Evaluacion 3: %.20lf %.20lf \n", val1, val2);
   }
  printf("\nContinuar? ");
  ans=tolower(getch());
 }
}
```

Usamos los problemas prueba

1. $a = 1$, $b = -2$ y $c = 1$. Aquí la raíz 1 es doble.
2. $a = 1 \times 10^{-6}$, $b = -10$ y $c = 1 \times 10^{-6}$. Tanto c como a son pequeños.
3. $a = 1 \times 10^{-8}$, $b = -10$ y $c = 1$. El coeficiente a es pequeño pero c no.
4. $a = 1 \times 10^{-10}$, $b = -1 \times 10^{10}$ y $c = 1 \times 10^{-10}$. El coeficiente b es bastante más grande que a y c.
5. $a = 4$, $b = -20$ y $c = -5$.
6. $a = 1 \times 10^{-3}$, $b = -2$ y $c = 1{,}23 \times 10^{-3}$.
7. $a = 3$, $b = -3$ y $c = 1 \times 10^{-9}$.

El programa utiliza precisión doble. Se introducen los coeficientes de la ecuación a, b y c, en ese orden. Se calcula el discriminante, y si es negativo avisa que las raíces no son reales; si es nulo o positivo, las raíces primero se calculan con la fórmula (1.2), luego con (2.2) y finalmente con (2.3). Después se evalua la expresión del polinomio en los resultados obtenidos para comparar la efectividad de los métodos.

```
Introduzca a, b, c: 1 -2 1
Las raices son: 1.0000000000000000000 1.0000000000000000000
Las raices son: 1.0000000000000000000 1.0000000000000000000
Las raices son: 1.5000000000000000000 0.5000000000000000000

Evaluacion 1: 0.0000000000000000000 0.0000000000000000000
Evaluacion 2: 0.0000000000000000000 0.0000000000000000000
Evaluacion 3: 0.2500000000000000000 0.2500000000000000000

Introduzca a, b, c: 1e-6 -10 1e-6
Las raices son: 9999999.9999999128000000000
0.0000009947598300641
Las raices son: 10052677.73966628500000000000
0.000000110000000000
Las raices son: 9999999.9999999128000000000
0.000000110000000000

Evaluacion 1: 0.00000000827242527157 0.00000000524016993587
Evaluacion 2: 529552.34092019428500000000
0.000000000000000000
Evaluacion 3: 0.00000000827242527157 0.000000000000000000001

Introduzca a, b, c: 1E-8 -10 1
Las raices son: 999999999.89999997600000000000
0.10000000827403710000
Las raices son: 999999917.25963580600000000000
 0.10000000001000000600
Las raices son: 999999999.89999997600000000000
0.10000000000000000600

Evaluacion 1: -0.00000002887099981308 -0.00000008264037098258
Evaluacion 2: -826.40357326995581400000 -0.000000000000000006375
Evaluacion 3: -0.00000002887099981308 0.00000000009999994447

Introduzca a, b, c: 1E-10 -1E10 1E-10
Las raices son: 100000000000000000000.000000000000000000000
0.000000000000000000000
Las raices son: 0.000000000000000000000
0.000000000000000000001
Las raices son: 100000000000000000000.000000000000000000000
0.000000000000000000001

Evaluacion 1: 36421322670080.000000000000000000000
0.000000000100000000000
Evaluacion 2: 0.000000000100000000000
-0.000000000000000000000
```

```
Evaluacion 3: 36421322670080.00000000000000000000
-0.00000000000000000000

Introduzca a, b, c: 4 -20 -5
Las raices son: 5.23861278752583104000 -0.23861278752583059400
Las raices son: 5.23861278752583015000 -0.23861278752583056600
Las raices son: 5.25000000000000000000 -0.25000000000000000000

Evaluacion 1: 0.00000000000001031814 0.00000000000000058850
Evaluacion 2: -0.00000000000000913852 -0.00000000000000001952
Evaluacion 3: 0.25000000000000000000 0.25000000000000000000

Introduzca a, b, c: 1E-3 -2 1.23E-3
Las raices son: 1999.99938499981090000000 0.00061500018910809473
Las raices son: 1999.99938501451516000000 0.00061500018911261629
Las raices son: 1999.99938499999985000000 0.00061499999999999999

Evaluacion 1: 0.00000000000010829337 0.00000000000000904312
Evaluacion 2: 0.00000002940860148958 0.00000000000000000001
Evaluacion 3: 0.00000000037800311991 0.00000000037822500000

Introduzca a, b, c: 3 -3 1E-9
Las raices son: 0.99999999966666675000 0.00000000033333328690
Las raices son: 1.00000013930425347000 0.00000000033333333344
Las raices son: 0.99999999966666663900 0.00000000033333333333

Evaluacion 1: 0.00000000000000025076 0.00000000000000013964
Evaluacion 2: 0.00000041891281861470 0.00000000000000000000
Evaluacion 3: -0.00000000000000008231 0.00000000000000000033
```

Vemos que en los casos regulares el desempeño de las fórmulas (1.2) y (2.2) es bueno, incluso el de la segunda un poco mejor en los casos intermedios. En los casos en los que a es pequeño pero c no, la aproximación lineal es mejor que ambas, y en el viceverso también. En el caso en ambos son pequeños tenemos que (2.2) es excelente.

2.2. Problema 2

Hagamos $u = x^{n+1}$ y $dv = e^x dx$. Entonces $du = (n+1)x^n dx$ y $v = e^x$, e integrando por partes (1.3) tenemos

$$y_{n+1} = x^{n+1}e^x|_0^1 - (n+1)\int_0^1 x^n e^x dx = e - (n+1)\int_0^1 x^n e^x dx = e - (n+1)y_n.$$

Como

$$y_0 = \int_0^1 e^x dx = e^x|_0^1 = e - 1,$$

ahora sólo falta probar que $y_1 = e - y_0 = e - (e-1) = 1$. En efecto

$$y_1 = \int_0^1 xe^x dx = e^x(x-1)|_0^1 = 1.$$

El programa que se escribió para el cálculo numérico de la sucesión es recursivo, esto es, para determinar el n-ésimo término debe calcular primero los $n-1$ términos anteriores.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<ctype.h>
#include<complex.h>

void main()
{
 double iteracion(int n);
 int num,i;
 double it;
 char ans='s';
 while (ans!='n')
 {
  clrscr();
  printf("Numero de iteraciones: ");
  scanf("%d", &num);
  for(i=0; i<num+1; i++)
   {
    printf("%d: %.20lf\n", i, iteracion(i));
   }
  printf("\nContinuar? ");
  ans=tolower(getch());
 }
}

double iteracion(int n)
{
 double iteracion(int n);
 if (n==0)
  return M_E-1;
 else
  return M_E-(n)*iteracion(n-1);
}
```

La salida para $n = 30$ es:

```
Numero de iteraciones: 30
0: 1.71828182845904509000
1: 1.00000000000000000000
2: 0.71828182845904509100
3: 0.56343634308190981800
4: 0.46453645613140581700
5: 0.39559954780201600500
6: 0.34468454164694906100
7: 0.30549003693040166500
8: 0.27436153301583177100
9: 0.24902803131655915100
10: 0.22800151529345358300
11: 0.21026516023105568100
12: 0.19509990568637691800
13: 0.18198305453614516100
14: 0.17051906495301283300
15: 0.16049585416385259200
16: 0.15034816183740362700
17: 0.16236307722318343800
18: -0.20425356155825680200
19: 6.59909949806592433000
20: -129.26370813285944900000
21: 2717.25615261850726000000
22: -59776.91707577869970000000
23: 1374871.81102473871000000000
24: -32996920.74631189930000000000
25: 824923021.37607932100000000000
26: -21447998553.05978010000000000000
27: 579095960935.33227500000000000000
28: -16214686906186.58590000000000000000
29: 470225920279413.75000000000000000000
30: -14106777608382410.00000000000000000000
```

Según la precisión doble en C, tenemos que $y_{30} = 1{,}4106777608382410 \times 10^{16}$. Sin embargo, un cálculo explícito realizado con la ayuda del programa Alged de John Henckel y de la Calculadora de Windows encontramos lo siguiente.

```
0 e-1
1 1
2 e-2
3 -2e+6
4 9e-24
5 -44e+120
6 265e-720
7 -1854e+5040
```

```
8 14833e-40320
9 -133496e+362880
10 1334961e-3628800
11 -14684570e + 39916800
12 176214841e - 479001600
13 -2290792932e + 6227020800
14 32071101049e - 87178291200
15 -481066515734e + 1307674368000
16 7697064251745e - 20922789888000
17 -130850092279664e + 355687428096000
18 2355301661033953e - 6402373705728000
19 -44750731559645106e + 121645100408832000
20 895014631192902121e - 2432902008176640000
21 -18795307255050944540e + 51090942171709440000
22 413496759611120779881e - 1124000727777607680000
23 -9510425471055777937262e + 25852016738884976640000
24 228250211305338670494289e - 620448401733239439360000
25 -5706255282633466762357224e + 15511210043330985984000000
26 148362637348470135821287825e - 403291461126605635584000000
27 -4005791208408693667174771274e +
10888869450418352160768000000
28 112162153835443422680893595673e -
3048883446117138605015040000000
29 -3252702461227859257745914274516e +
88417619937397019545436160000000
30 9758107383683577773237742823548le -
265252859812191058636308480000000
```

Considerando que $2 < e$, tenemos que y_{30} satisface

$$1{,}68636861692452449601124008470962 \times 10^{32} < y_{30}$$

lo cual está asombrosamente lejos del resultado de nuestro programa. De hecho, es fácil ver que el segundo sumando en cada término de la lista anterior en valor absoluto es $n!$, mientras que el otro término siempre es menor en valor absoluto. Entonces $y_n = O(n!)$. Para el compilador, $e = 2{,}71828182845904509$, lo cual implica que cuando $n! \approx 10^{17}$, el cálculo empezará a perder precisión notablemente. Ésto ocurre aproximadamente entre y_{18} y y_{19}, que es, en efecto, donde la sucesión se hace mayor que 1 y difiere notablemente del valor real, pues con el programa obtenemos $y_{18} = -0{,}20425356155825680200$ mientras que con la calculadora de Windows resulta $y_{18} = 0{,}13623989097759060373825488998487 7$.

Capítulo 3

Conclusiones

3.1. Conclusiones generales

En general, para tener una idea de la precisión de nuestros métodos de cálculo hemos tenido que estimar de forma analítica qué resultado esperar, para comparar y determinar la efectividad de los mismos. Estimar en términos analíticos puede resultar difícil, como en el caso del problema 2, dada la magnitud grande de los números a calcular; sin embargo, siempre es más fácil que calcular como lo hace la computadora, con todo y su error intrínseco. En resumen, tener un estimado de lo que debemos alcanzar puede ahorrar tiempo y esfuerzo en conjunción con un buen planteamiento para hallar una solución numérica.

3.1.1. Sobre el problema 1

- Aún cuando se cuente con una fórmula explícita para resolver todos los casos de un problema, al calcular numéricamente las soluciones, la finitud del dispositivo de cálculo puede dificultar o incluso impedir obtener un resultado exacto.
- Por lo tanto, es necesario realizar manipulaciones algebraicas que permitan minimizar el error por redondeo y manipular números que estén dentro del rango que puede representar la máquina.
- Muchas veces, una aproximación lineal puede resultar ventajosa, aunque debe ponderarase adecuadamente cuando utilizarla.

3.1.2. Sobre el problema 2

- Manejar numéros irracionales o trascendentes puede reducir drásticamente la precisión de un esquema de cálculo, pues las representaciones decimales de éste tipo de numéros es infinita, por lo que no tienen verdadera representación en la máquina. Tal es el caso de π, e, $\sqrt{2}$, y otros.

- Esto puede desencadenar una "hilera de dominós.[en] cuanto al crecimiento del error, pues cuando se le multiplica a las aproximaciones de la máquina por números de magnitud muy grande, puede perderse mucha precisión pues no todos los dígitos (que son infinitos) de un trascendente (por ejemplo) contribuyen al resultado.