Práctica_1

October 25, 2022

1 Práctica 1

[1]:
```
addpath('./Biblioteca')
global ndig = 4
```

1.1 Ejercicio 1:

En la carpeta /Biblioteca se ha incluido la función redondeo.m. Dado un valor numérico x, de cualquier tipo (incluso matricial), y un entero numdig, redondeo(x,numdig) devuelve el valor de x redondeado al número de cifras decimales indicado por la variable numdig. Estudia el código de esta función teniendo en cuenta para ello que:

1. mat2str(x,n) convierte el valor de x en una cadena de caracteres, redondeando x a n dígitos de precisión.
2. eval(cad) ejecuta la cadena cad como si fuera código de Octave. El resultado en la función redondeo.m es un valor numérico (o matricial) en coma flotante.
3. El argumento "local" en la función output_precision permite que el efecto de esta función deje de estar activo al terminar la ejecución de la misma o de otro script que pueda utilizarla.

Crea un archivo script en el que se calculen los resultados de las operaciones que siguen con cuatro cifras decimales (redondeo el que utiliza Octave por defecto):

a) 0.6688 $ $ 0.3334;

b) 1000 $ $ 0.05001;

c) 2.000 $\otimes$ 0.6667;

d) 25.00 $\oslash$ 16.00

Calcula en cada caso los errores absolutos y relativos.

A la vista de los resultados (o de otros intentos que puedas hacer, por ejemplo, directamente en la ventana de comandos), ¿puedes deducir qué tipo de redondeo utiliza Octave?

1.1.1 Solución

[2]:
```
a_1 = 0.6688
a_2 = 0.3334
ra_1 = redondeo(a_1, ndig)
ra_2 = redondeo(a_2, ndig)
```

```
suma = ra_1 + ra_2
rsuma = redondeo(suma, ndig)
disp(suma)
disp(a_1 +a_2)
```

```
a_1 = 0.6688
a_2 = 0.3334
ra_1 =           0.6688
ra_2 =           0.3334
suma =           1.0022
rsuma =            1.002
          1.0022
          1.0022
```

En este caso podemos, un simple cálculo a mano nos mostrará que el valor redondeado no difiere del exacto, por tanto el error absoluto y relativo serán 0.

[3]:
```
b_1 = 1000
b_2 = 0.05001
redondeo_resta = redondeo(redondeo(b_1,ndig) - redondeo(b_2, ndig),ndig)
valor_real_resta = b_1 - b_2
error_absoluto = abs(valor_real_resta - redondeo_resta)
error_relativo = error_absoluto/valor_real_resta
```

```
b_1 =             1000
b_2 =          0.05001
redondeo_resta =            999.9
valor_real_resta =        999.94999
error_absoluto = 0.04998999999997977
error_relativo = 4.999250012491102e-05
```

En este caso el valor real de la resta es 999.94999, por tanto podemos calcular fácilmente tanto el error absoluto como el error relativo.

[4]:
```
c_1 = 2.000
c_2 = 0.6667
mult_real = c_1 * c_2
redondeo_mult = redondeo(redondeo(c_1, ndig) * redondeo(c_2, ndig),ndig)
c_error_absoluto = abs(redondeo_mult - mult_real)
c_error_relativo = 0.0004/mult_real
```

```
c_1 =               2
c_2 =          0.6667
mult_real =           1.3334
redondeo_mult =            1.333
c_error_absoluto = 0.0003999999999999559
c_error_relativo = 0.0002999850007499625
```

Aquí podemos observar que al calcular el error absoluto da lugar a un error en la resta, pues en

lugar de 0.0004 que debiera ser el valor correcto la forma en que calcula el valor Octave lo lleva a una aproximación que no es correcta.

[5]:
```
d_1 = 25.00
d_2 = 16.00
div_real = 25/16
red_div = redondeo(redondeo(d_1, ndig)/redondeo(d_2, ndig), ndig)
d_er_abs = abs(div_real - red_div)
d_er_rel = d_er_abs/div_real
```

```
d_1 =                25
d_2 =                16
div_real =            1.5625
red_div =             1.562
d_er_abs = 0.0004999999999999449
d_er_rel = 0.0003199999999999647
```

1.2 Ejercicio 2:

Consideremos la ecuación de segundo grado

$$1.002x^2 - 11.01x + 0.01265 = 0$$

Trabajando con la función redondeo, calcula las soluciones de la ecuación realizando todos los cálculos con cuatro cifras significativas de las dos formas detalladas a continuación. Para una ecuación $ax^2 + bx + c = 0$ las raíces se pueden calcular: 1. mediante las expresiones bien conocidas $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$, y 2. calculando una raíz x_1 de forma que se evite la pérdida de cifras significativas y teniendo en cuenta entoces que las raíces verifican $x_1 x_2 = \frac{c}{a}$.

Para hacernos una idea de la precisión de ambos cálculos vamos a reconstruir el polinomio original utilizando las dos funciones siguientes.

- La función **poly[x1,x2,...,xn]** de Octave devuelve un vector compuesto por los coeficientes del polinomio cuyas raíces son x1,x2,…,xn; los coeficientes están ordenados de forma decreciente, es decir, desde el coeficiente del término de mayor grado, hasta el término independiente.
- La función **polyout(v,'x')** de Octave, devuelve una cadena igual a la expresión del polinomio en la variable x que tiene los coeficientes incluidos en el vector v.

Realiza los mismos cálculos con 2 y 8 cifras significativas.

1.2.1 Solución

- Con cuatro cifras significativas

[6]:
```
a = 1.002
b = -11.01
c = 0.01265
x1 = redondeo(redondeo(-b + redondeo(sqrt(redondeo(redondeo(b^2, ndig) -␣
 ↪redondeo(4*a*c, ndig), ndig)), ndig), ndig)/redondeo(2*a, ndig),ndig)
```

```
x2 = redondeo(redondeo(-b - redondeo(sqrt(redondeo(redondeo(b^2, ndig) -␣
 ↪redondeo(4*a*c, ndig), ndig)), ndig), ndig)/redondeo(2*a, ndig),ndig)
x2_2 = redondeo(redondeo(c/a, ndig) * x1,ndig)
```

```
a =            1.002
b =           -11.01
c =          0.01265
x1 =            10.98
x2 =          0.00499
x2_2 =          0.1386
```

Procedamos ahora a la reconstrucción inversa a partir de las raíces.

[7]:
```
coef1 = poly([x1,x2])
expr1 = polyout(coef1, 'x')
```

```
coef1 =

                        1               -10.98499                0.0547902

expr1 = 1*x^2 - 10.985*x^1 + 0.05479
```

$$1 * x^2 - 10.985 * x^1 + 0.05479$$

[8]:
```
coef2 = poly([x1,x2_2])
expr2 = polyout(coef2, 'x')
```

```
coef2 =

                        1               -11.1186                 1.521828

expr2 = 1*x^2 - 11.119*x^1 + 1.5218
```

$$1 * x^2 - 11.119 * x^1 + 1.5218$$

- Con dos cifras significativas

[9]:
```
global ndig = 2
a = 1.002
b = -11.01
c = 0.01265
x1 = redondeo(redondeo(-b + redondeo(sqrt(redondeo(redondeo(b^2, ndig) -␣
 ↪redondeo(4*a*c, ndig), ndig)), ndig), ndig)/redondeo(2*a, ndig),ndig)
x2 = redondeo(redondeo(-b - redondeo(sqrt(redondeo(redondeo(b^2, ndig) -␣
 ↪redondeo(4*a*c, ndig), ndig)), ndig), ndig)/redondeo(2*a, ndig),ndig)
x2_2 = redondeo(redondeo(c/a, ndig) * x1,ndig)
```

```
a =            1.002
b =           -11.01
```

```
c =          0.01265
x1 =             10.98
x2 =           0.00499
x2_2 =            0.1386
```

[10]:
```
coef1 = poly([x1,x2])
expr1 = polyout(coef1, 'x')
```

```
coef1 =

                        1                -10.98499                0.0547902

expr1 = 1*x^2 - 10.985*x^1 + 0.05479
```

$$1 * x^2 - 10.985 * x^1 + 0.05479$$

[11]:
```
coef2 = poly([x1,x2_2])
expr2 = polyout(coef2, 'x')
```

```
coef2 =

                        1                -11.1186                 1.521828

expr2 = 1*x^2 - 11.119*x^1 + 1.5218
```

$$1 * x^2 - 11.119 * x^1 + 1.5218$$

- Con 8 cifras significativas

[12]:
```
global ndig = 8

a = 1.002
b = -11.01
c = 0.01265
x1 = redondeo(redondeo(-b + redondeo(sqrt(redondeo(redondeo(b^2, ndig) -␣
 ↪redondeo(4*a*c, ndig), ndig)), ndig), ndig)/redondeo(2*a, ndig),ndig)
x2 = redondeo(redondeo(-b - redondeo(sqrt(redondeo(redondeo(b^2, ndig) -␣
 ↪redondeo(4*a*c, ndig), ndig)), ndig), ndig)/redondeo(2*a, ndig),ndig)
x2_2 = redondeo(redondeo(c/a, ndig) * x1,ndig)
```

```
a =           1.002
b =          -11.01
c =          0.01265
x1 =             10.98
x2 =           0.00499
x2_2 =            0.1386
```

[13]:
```
coef1 = poly([x1,x2])
expr1 = polyout(coef1, 'x')
```

```
coef1 =

                        1                -10.98499                0.0547902

expr1 = 1*x^2 - 10.985*x^1 + 0.05479
```

$$1 * x^2 - 10.985 * x^1 + 0.05479$$

[14]:
```
coef2 = poly([x1,x2_2])
expr2 = polyout(coef2, 'x')
```

```
coef2 =

                        1                -11.1186                 1.521828

expr2 = 1*x^2 - 11.119*x^1 + 1.5218
```

$$1 * x^2 - 11.119 * x^1 + 1.5218$$

2 Ejercicio 3

De acuerdo con el ejercicio 1.27 (con las precauciones sugeridas por el apartado c) del mismo ejercicio) parece que será más preciso realizar una suma de términos positivos de forma que el tamaño de los sumandos sea creciente. Vamos a realizar la experiencia con una suma bien conocida:

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

El Objetivo es calcular las sumas

$$\sum_{k=1}^{n} \frac{1}{k^2}$$

para distintos valores de n en orden creciente y decreciente y comparar los resultados.

Al final del enunciado de este ejercicio se incluyen detalles sobre las órdenes de Octave necesarias para desarrollar el ejercicio.

1. Calcula, en **precisión simple**, la suma en orden creciente y decreciente de sumandos para n igual a 50, 100, 500, 1000, 10000, 100000. Haz una tabla con los valores de n, los dos valores de la suma y los respectivos errores relativos (usando el valor exacto $\pi^2/6$).
2. Haz lo mismo en precisión doble.

3. Repite el resultado utilizando la función redondeo con cinco cifras decimales (evita en principio el caso n = 100000 en este apartado, porque es muy lento).

4. Una vez hayas depurado todos los apartados anteriores introduce los comandos tic y toc que permiten controlar el tiempo de ejecución: tic inicia el cronómetro, toc lo detiene. Puedes asignar el valor de toc a una variable (por ejemplo, t=toc), y así mostrar el tiempo de ejecución de un proceso en el formato que desees.

Observaciones

- Para realizar un cálculo en precisión simple basta escribir single(calculo). Incluso puedes escribir single(a) siendo a un valor fijo o una variable numérica. Para realizar los cálculos en precisión doble no hay que tomar ninguna precaución, Octave los hace con esta precisión por defecto.
- Los bucles en Octave los puedes escribir de varias formas.
- Una idea para obtener tablas en la salida de Octave es utilizar especificaciones de formato en la orden de escritura. Por ejemplo: printf(form,x1,x2,x3) escribirá las variables x1, x2 y x3 con el formato indicado por la especificación form, que es una cadena o una variable de tipo cadena que debemos haber definido previamente. Por ejemplo, podríamos definir (o utilizar la cadena siguiente directamente en printf): form=' %12.7e, %16.8f, %6u'; lo que significa que: la primera variable se imprimirá ocupando el ancho de 12 caracteres con 7 cifras decimales en notación exponencial, la segunda ocupará el ancho de 16 caracteres con 8 decimales y la tercera será un entero sin signo (de «unsigned») ocupando el ancho de 6 caracteres.

2.0.1 Solución

[15]:
```
nvalues = [50, 100, 500, 1000, 10000];
```

- Precisión simple

[22]:
```
tic
for i = 1:columns(nvalues),
    suma_crec(i) = 0;
    suma_decrec(i) = 0;
    for j = 1:i,
        single(suma_crec(i) += single(1/(j^2)));
        single(suma_decrec(i) += single(1/((nvalues(1,i)-j)^2)));
    endfor
    errorr_c(i) = abs(suma_crec(i)-pi/6)/pi/6;
    errorr_d(i) = abs(suma_decrec(i)-pi/6)/pi/6;
    matriz_single(i, :) = [nvalues(1,i), suma_crec(i), errorr_c(i),␣
 ↪suma_decrec(i), errorr_d(i)];
endfor
temp = toc %tiempo de ejecución

display(matriz_single)
```

```
temp = 1.352609872817993
matriz_single =
```

```
 Columns 1 through 3:

                    50                     1     0.02527386991952067
                   100                  1.25     0.03853678184384528
                   500      1.361111164093018     0.04443141217654478
                  1000      1.423611164093018     0.04774714015762593
                 10000      1.463611125946045      0.0498692040417581

 Columns 4 and 5:

   0.0004164931306149811     0.02775568213094405
   0.0002061536943074316     0.02776684098461588
   1.209667607326992e-05     0.02777713602918043
   4.020090273115784e-06     0.02777756450536489
   5.003001746217706e-08     0.02777777512360291
```

- Precisión doble

[23]:
```
matriz_double = zeros(3,5);
tic
for i = 1:columns(nvalues),
    suma_crec(i) = 0;
    suma_decrec(i) = 0;
    for j = 1:i,
        double(suma_crec(i) += double(1/(j^2)));
        double(suma_decrec(i) += double(1/((nvalues(1,i)-j)^2)));
        errorr_c(i) = abs(suma_crec(i)-pi/6)/pi/6;
        errorr_d(i) = abs(suma_decrec(i)-pi/6)/pi/6;
    endfor
    matriz_double(i, :) = [nvalues(1,i), suma_crec(i), errorr_c(i),␣
 ↪suma_decrec(i), errorr_d(i)];
endfor
temp = toc %tiempo

matriz_double
```

```
temp = 1.332026958465576
matriz_double =

 Columns 1 through 3:

                    50                     1     0.02527386991952067
                   100                  1.25     0.03853678184384528
                   500      1.361111111111111     0.04443140936576734
                  1000      1.423611111111111     0.04774713734684849
                 10000      1.463611111111111     0.04986920325474042
```

```
Columns 4 and 5:

  0.0004164931278633903     0.02775568213109002
  0.0002061536870265556     0.02776684098500214
   1.20966766395734e-05     0.02777713602915038
  4.020090401777835e-06     0.02777756450535807
   5.00300165090049e-08     0.02777777512360297
```

[ ]: