Hwk2StabilitySolution

February 13, 2024

1 Roundoff Error and Stability

The purpose of computing is insight, not numbers (Hamming).

As we discussed in class, roundoff error is a fact of life when computing numerically. As computations proceed these roundoff errors can accumulate and cause many difficulties. As long as the roundoff errors grow linearly with the number of steps, then we can expect that our final results should be reasonable (barring an ill-conditioned problem).

This problem demonstrates some of the pitfalls when trying to compute various integrals and some possible alternative formulations to avoid them.

1.1 Problem 5

Suppose we are asked to compute

$$y_n = \int_0^1 \frac{x^n}{x+10}dx, \;\; n = 1, 2, \ldots 30 \qquad (1)$$

a) ***Show that***

$$y_n + 10y_{n-1} = \frac{1}{n}$$

Solution:

Let's first note that

$$\begin{aligned}
y_n + 10y_{n-1} &= \int_0^1 \frac{x^n}{x+10}dx + 10\int_0^1 \frac{x^{n-1}}{x+10}dx && \text{by definition} \\
&= \int_0^1 \frac{x^n + 10x^{n-1}}{x+10}dx && \text{combine terms} \\
&= \int_0^1 \frac{x^{n-1}(x+10)}{x+10}dx, && \text{factor out } x^{n-1} \\
&= \int_0^1 x^{n-1}dx = \left.\frac{x^n}{n}\right|_0^1 \\
y_n + 10y_{n-1} &= \frac{1}{n}
\end{aligned}$$

or solving for y_n:

$$y_n = \frac{1}{n} - 10y_{n-1}$$

b) ***Write a code to numerically compute a value for*** y_0.

Solution:

Note that for $n = 0$ Equation (1) reduces to:

$$y_0 = \int_0^1 \frac{1}{x+10}dx = \ln(x+10)\Big|_0^1$$
$$= \ln(11) - \ln(10)$$
$$= 0.0953101798043$$

c) ***Using (a) and (b) propose an algorithm and write a code that computes*** $y_n, n = 1, 2, \ldots, 30$***. You may not use any other software except for what you write yourself.***

1.2 Algorithm:

This suggests the following procedure:

1. Evaluate $y_0 = \ln(11) - \ln(10)$
2. For $n = 1, 2, \ldots 30$
    - $y_n = \frac{1}{n} - 10y_{n-1}$

Important:

Before we even start, you may want to note that all of the integrals must be positive. In fact, we can show that

$$0 < y_n < 1, \quad \forall n = 1, 2, \ldots, 30$$

so we have a sense of what reasonable values should be.

Let's see how we might be able to implement the above algorithm in python

[10]:
```
# Setup numpy environment
import numpy as np
from scipy import io, integrate, linalg, signal
from scipy.sparse.linalg import cg, eigs

# Initialize
naxis = np.arange(31)
y = np.zeros(31)
y[0] = np.log(11) - np.log(10)

# Compute the rest of the integrals
for n in range(1,31):
    y[n] = 1/n - 10*y[n-1]

# View the results
y
```

```
[10]: array([ 9.53101798e-02,  4.68982020e-02,  3.10179804e-02,  2.31535290e-02,
               1.84647099e-02,  1.53529009e-02,  1.31376581e-02,  1.14805618e-02,
               1.01943816e-02,  9.16729515e-03,  8.32704853e-03,  7.63860560e-03,
               6.94727731e-03,  7.45030378e-03, -3.07446639e-03,  9.74113306e-02,
              -9.11613306e-01,  9.17495659e+00, -9.16940103e+01,  9.16992735e+02,
              -9.16987735e+03,  9.16988211e+04, -9.16988166e+05,  9.16988170e+06,
              -9.16988169e+07,  9.16988170e+08, -9.16988170e+09,  9.16988170e+10,
              -9.16988170e+11,  9.16988170e+12, -9.16988170e+13])
```

Viewing the results by typing the variable is the easiest, but sometimes the output is hard to read and therefore hard to interpret. Let's see how we can make the output look a bit clearer using the python package pandas

```
[11]: import pandas as pd

      # Create a data frame using pandas

      table = np.column_stack((naxis,y))
      table = pd.DataFrame(table, columns=['n', 'y_n'])
      table
```

```
[11]:        n           y_n
      0    0.0  9.531018e-02
      1    1.0  4.689820e-02
      2    2.0  3.101798e-02
      3    3.0  2.315353e-02
      4    4.0  1.846471e-02
      5    5.0  1.535290e-02
      6    6.0  1.313766e-02
      7    7.0  1.148056e-02
      8    8.0  1.019438e-02
      9    9.0  9.167295e-03
      10  10.0  8.327049e-03
      11  11.0  7.638606e-03
      12  12.0  6.947277e-03
      13  13.0  7.450304e-03
      14  14.0 -3.074466e-03
      15  15.0  9.741133e-02
      16  16.0 -9.116133e-01
      17  17.0  9.174957e+00
      18  18.0 -9.169401e+01
      19  19.0  9.169927e+02
      20  20.0 -9.169877e+03
      21  21.0  9.169882e+04
      22  22.0 -9.169882e+05
      23  23.0  9.169882e+06
      24  24.0 -9.169882e+07
      25  25.0  9.169882e+08
```

```
26  26.0 -9.169882e+09
27  27.0  9.169882e+10
28  28.0 -9.169882e+11
29  29.0  9.169882e+12
30  30.0 -9.169882e+13
```

1.3 Observations

List any insights you have after viewing your preliminary results.

1.
2.
3.

(d) ***The true values for the integrals are given in Table 1 of the homework sheet. For each of the y_n, compute the actual and relative errors (where applicable). Analyze and explain your results in terms of what we discussed in class on the stability of algorithms.***

Note: You could input the values from the table, but for the sake of convenience we'll provide you the code that produced the numbers. Let's compute the actual values of the integrals and check the error. We will introduce methods for numerically computing integrals later in the course. For now, let's just say that the function np.trapz, will numericall evaluate the integral above.

[12]:
```
x = np.linspace(0,1,num=50)
yint = np.zeros(31)

# Compute the integrals using the Trapezoidal rule
for n in range(0,30):
    yn = x**n / (x + 10)
    yint[n] = np.trapz(yn,x)

yint
```

[12]:
```
array([0.09531024, 0.0468976 , 0.031024  , 0.02316271, 0.01847704,
       0.01536839, 0.0131563 , 0.01150236, 0.01021934, 0.0091953 ,
       0.00835922, 0.00766384, 0.00707652, 0.00657401, 0.00613926,
       0.0057595 , 0.00542499, 0.00512816, 0.00486303, 0.00462483,
       0.00440971, 0.00421449, 0.00403658, 0.00387381, 0.00372434,
       0.00358665, 0.00345942, 0.00334152, 0.00323199, 0.00312998,
       0.        ])
```

1.4 Error Analysis

To fully understand what is going on, we need to take a look at the error between the algorithm's output and the "true" value computed by the np.trapz function.

What does the data show?

[13]:
```
# Compute the absolute error

abserr = abs(yint - y)
table = np.column_stack((naxis,abserr))
table = pd.DataFrame(table, columns=['n', 'abserr'])
table
```

[13]:
```
       n        abserr
0    0.0  6.023656e-08
1    1.0  6.023656e-07
2    2.0  6.023656e-06
3    3.0  9.178958e-06
4    4.0  1.233371e-05
5    5.0  1.548820e-05
6    6.0  1.864230e-05
7    7.0  2.179464e-05
8    8.0  2.495730e-05
9    9.0  2.800810e-05
10  10.0  3.216742e-05
11  11.0  2.523001e-05
12  12.0  1.292469e-04
13  13.0  8.762934e-04
14  14.0  9.213722e-03
15  15.0  9.165183e-02
16  16.0  9.170383e-01
17  17.0  9.169828e+00
18  18.0  9.169887e+01
19  19.0  9.169881e+02
20  20.0  9.169882e+03
21  21.0  9.169882e+04
22  22.0  9.169882e+05
23  23.0  9.169882e+06
24  24.0  9.169882e+07
25  25.0  9.169882e+08
26  26.0  9.169882e+09
27  27.0  9.169882e+10
28  28.0  9.169882e+11
29  29.0  9.169882e+12
30  30.0  9.169882e+13
```

1.5 Error Analysis

If we take a look at the error between the two formulations, you'll see that we have an accurate approximation at the start - on the order of 10^{-8}. But the error quickly builds up until by the 16th value the absolute error is ≈ 1.

Since the formula for computing each integral involves a subtraction,

$$y_n = \frac{1}{n} - 10y_{n-1}$$

it makes sense to suspect there might be some cancellation error. If we take a look at the two quantities involved we might be able to determine what is happening.

Another explanation is that at each iteration the value of the integral y_{n-1} has some roundoff error. When we multiply that quantity by 10, the error is also magnified by a factor of 10.

As a result, the next computed integral's error is off by another factor of 10.

Recall from class lecture: - Let E_n be the error at the nth step of some computation. If the error $E_n \approx C \cdot nE_0$, where C is a constant and E_0 is the original error, then the growth of error is said to be ***linear***.- If the error $E_n \approx C^n E_0$, where C is a constant and E_0 is the original error, then the growth of error is said to be ***exponential***.

This is a clear signal of an **unstable algorithm**

[14]:
```
# Compute the relative error

abserr = abs(yint - y)/abs(yint)
table = np.column_stack((naxis,abserr))
table = pd.DataFrame(table, columns=['n', 'abserr'])
table
```

```
/var/folders/ks/_91630850zn018jzskzpdw1c0000gp/T/ipykernel_7615/1550262451.py:3:
RuntimeWarning: divide by zero encountered in divide
  abserr = abs(yint - y)/abs(yint)
```

[14]:
```
       n        abserr
0    0.0  6.320052e-07
1    1.0  1.284427e-05
2    2.0  1.941612e-04
3    3.0  3.962817e-04
4    4.0  6.675151e-04
5    5.0  1.007796e-03
6    6.0  1.416987e-03
7    7.0  1.894798e-03
8    8.0  2.442164e-03
9    9.0  3.045914e-03
10  10.0  3.848138e-03
11  11.0  3.292087e-03
12  12.0  1.826417e-02
13  13.0  1.332966e-01
14  14.0  1.500788e+00
15  15.0  1.591317e+01
16  16.0  1.690397e+02
17  17.0  1.788133e+03
18  18.0  1.885632e+04
```

```
19  19.0  1.982748e+05
20  20.0  2.079477e+06
21  21.0  2.175798e+07
22  22.0  2.271695e+08
23  23.0  2.367151e+09
24  24.0  2.462148e+10
25  25.0  2.556670e+11
26  26.0  2.650701e+12
27  27.0  2.744225e+13
28  28.0  2.837226e+14
29  29.0  2.929690e+15
30  30.0           inf
```

e) ***Modify your code so that it can compute more accurate values for the integral (without resorting to numerical integration or any other software). Justify your algorithm, demonstrate why it is more accurate, and provide all code.***

1.6 Solution

[15]:
```python
# Initialize
naxis = np.arange(31)
yrev = np.zeros(31)
yrev[0] = np.log(11) - np.log(10)

# Compute the rest of the integrals
for n in range(30, 1, -1):
    yrev[n-1] = (1/n - yrev[n])/10

yrev
```

[15]:
```
array([0.09531018, 0.0468982 , 0.03101798, 0.02315353, 0.01846471,
       0.0153529 , 0.01313766, 0.01148056, 0.01019439, 0.0091672 ,
       0.00832797, 0.00762944, 0.00703898, 0.00653332, 0.00609542,
       0.00571251, 0.00537486, 0.00507489, 0.00480663, 0.0045653 ,
       0.00434704, 0.00414869, 0.00396765, 0.00380175, 0.00364916,
       0.00350838, 0.00337771, 0.00325993, 0.00311494, 0.00333333,
       0.        ])
```

[16]:
```python
# Compute the error - note the use of absolute rather than relative since the␣
↪last value of yint = 0.0

abserr2 = abs(yint - yrev)
table = np.column_stack((naxis,abserr2))
table = pd.DataFrame(table, columns=['n', 'abserr2'])
table
```

[16]:
```
      n       abserr2
0   0.0  6.023656e-08
```

```
1    1.0  6.023656e-07
2    2.0  6.023656e-06
3    3.0  9.178958e-06
4    4.0  1.233370e-05
5    5.0  1.548821e-05
6    6.0  1.864221e-05
7    7.0  2.179556e-05
8    8.0  2.494813e-05
9    9.0  2.809980e-05
10  10.0  3.125043e-05
11  11.0  3.439989e-05
12  12.0  3.754805e-05
13  13.0  4.069479e-05
14  14.0  4.383996e-05
15  15.0  4.698344e-05
16  16.0  5.012511e-05
17  17.0  5.326483e-05
18  18.0  5.640247e-05
19  19.0  5.953790e-05
20  20.0  6.267101e-05
21  21.0  6.580165e-05
22  22.0  6.892974e-05
23  23.0  7.205475e-05
24  24.0  7.518049e-05
25  25.0  7.826768e-05
26  26.0  8.170763e-05
27  27.0  8.158584e-05
28  28.0  1.170459e-04
29  29.0  2.033498e-04
30  30.0  0.000000e+00
```