Obligatorisk oppgave 1, MAT-INF1100, Høst 2020

Cory Alexander Balaton

September 2020

Task 1

a)

```python
import math

def for_form(n, a_1, a_0):
    s = 0
    for i in range(n+1):
        if i == 1 or i == 0:
            s = 1.0
        else:
            s = 4.0*a_1 + a_0
            a_0 = a_1
            a_1 = s
    return s

print("Initial conditions: a_0 = 1, a_1 = 1 \n")


for i in range(2, 101):
    f = for_form(i, 1.0, 1.0)
    print(f"n = {i:3d}: {f:>24}")

#Something
```

Under is the printout of the 4 first and 4 last terms of the program:

```
n =   2:                      5.0
n =   3:                     21.0
n =   4:                     89.0
n =   5:                    377.0
                  ...
n =  97:      1.8068856563238e+60
n =  98:     7.65409046775694e+60
n =  99:    3.242324752735156e+61
n = 100:   1.3734708057716316e+62
```

b)

```python
import math

def for_form(n, a_1, a_0):
    s = 0
    for i in range(n+1):
        if i == 1 or i == 0:
            s = 1.0
        else:
            s = 4.0*a_1 + a_0
            a_0 = a_1
            a_1 = s
    return s

print("Initial conditions: a_0 = 1, a_1 = 2 - sqrt(5) \n")

for i in range(2, 101):
    f = for_form(i, 2.0 - math.sqrt(5.0), 1.0)
    print(f"n = {i:3d}: {f:>24}")
```

Under is the printout of the 4 first and 4 last terms of the program:

```
n =   2:       0.05572809000084078
n =   3:    -0.013155617496426686
n =   4:     0.003105620015134036 3
n =   5:    -0.0007331374358905407
                    ...
n =  97:  -1.5881345077615722e+44
n =  98:  -6.7274457322911874e+44
n =  99:  -2.8497917436926322e+45
n = 100:  -1.2071911547999648e+46
```

c)

For the equation $x_{n+2} - 4x_{n+1} - x_n = 0$, we can set $x_n = x^n$ and get:

$$\begin{aligned} x^{n+2} - 4x^{n+1} - x^n &= 0 \\ x^n(x^2 - 4x - 1) &= 0 \\ x &= \frac{4 \pm \sqrt{(-4)^2 - 4 \cdot 1 \cdot (-1)}}{2} \\ x &= \frac{4 \pm \sqrt{16 + 4}}{2} \\ x &= \frac{4 \pm \sqrt{20}}{2} \\ x &= 2 \pm \sqrt{5} \\ x_1 = 2 + \sqrt{5} \wedge x_2 &= 2 - \sqrt{5} \end{aligned} \tag{1}$$

Knowing the roots, we can write the equation:

$$x_n = C(2 + \sqrt{5})^n + D(2 - \sqrt{5})^n \tag{2}$$

We can then find C and D with the initial conditions $x_0 = 1$ and $x_1 = 2 - \sqrt{5}$:

$$\begin{aligned} C(2 + \sqrt{5})^0 + D(2 - \sqrt{5})^0 &= 1 \\ C + D &= 1 \\ C &= 1 - D \end{aligned} \tag{3}$$

$$\begin{aligned} C(2 + \sqrt{5})^1 + D(2 - \sqrt{5})^1 &= 2 - \sqrt{5} \\ C(2 + \sqrt{5}) + D(2 - \sqrt{5}) &= 2 - \sqrt{5} \end{aligned} \tag{4}$$

By substituting C in equation (4) with $1 - D$, we get:

$$\begin{aligned} (1 - D)(2 + \sqrt{5}) + D(2 - \sqrt{5}) &= 2 - \sqrt{5} \\ 2 - 2D + \sqrt{5} - \sqrt{5}D + 2D - \sqrt{5}D &= 2 - \sqrt{5} \\ 2 - \cancel{2D} + \sqrt{5} + \cancel{2D} - 2 \cdot \sqrt{5}D &= 2 - \sqrt{5} \\ -2 \cdot \sqrt{5}D &= -2 \cdot \sqrt{5} \\ D &= 1 \end{aligned} \tag{5}$$

By substituting D in equation 3 with 1 we get:

$$\begin{aligned} C &= 1 - 1 \\ C &= 0 \end{aligned} \tag{6}$$

Now that we have found C and D, we can substitute them in equation (2) and get:

$$\begin{aligned} x_n &= 0 \cdot (2+\sqrt{5})^n + 1 \cdot (2-\sqrt{5})^n \\ &= (2-\sqrt{5}) \end{aligned} \tag{7}$$

□

d)

When comparing the results of the program in b, and a program that uses the analytical form to calculate x_n, the results vary slightly in the beginning because of slight rounding errors. What was really interesting is when the program in b started to dive downwards in the negative direction at about $n = 13$. If we look at the analytical form, the number should be negative when n is odd, and positive when n is even, because $x_n = (2-\sqrt{5})^n$ and $2-\sqrt{5} < 0$. Another fact is that $0 < |2-\sqrt{5}| < 1$, which means that $0 < |2-\sqrt{5}|^n < 1$. Since these two statements are not being fulfilled when the program in b is run, the answers must then be wrong.

I wrote an extended program from b where each row shows x_n, x_{n-1}, and x_{n-2}. Program:

```
import math

def for_form(n, a_1, a_0):
    s = 0
    for i in range(n+1):
        if i == 1 or i == 0:
            s = 1.0
        else:
            s = 4.0*a_1 + a_0
            res_0 = a_0
            res_1 = a_1
            a_0 = a_1
            a_1 = s
    return [s, res_1, res_0]

print("Initial conditions: a_0 = 1, a_1 = 2 - sqrt(5) \n")

for i in range(2, 101):
    f = for_form(i, 2.0 - math.sqrt(5.0), 1.0)
    print(f"n = {i:3d}: 4*a_{i-1} = {4*f[1]:>24} <==> a_{i-2} = {f
    [2]:>24} <==> 4*a_{i-1} + a_{i-2} = {f[0]:>24}")
```

And under I have added some of the rows printed that are significant:

```
n =  12:   4*a_11 =   -5.083030174546366e-07 <==> a_10 =    5.374453024842296e-07
<==> 4*a_11 + a_10 =    2.914228502959304e-08

n =  13:   4*a_12 =   1.1656914011837216e-07 <==> a_11 =  -1.2707575436365914e-07
<==> 4*a_12 + a_11 =  -1.0506614245286983e-08

n =  14:   4*a_13 =   -4.202645698114793e-08 <==> a_12 =    2.914228502959304e-08
<==> 4*a_13 + a_12 =   -1.288417195155489e-08

n =  15:   4*a_14 =   -5.153668780621956e-08 <==> a_13 =  -1.0506614245286983e-08
<==> 4*a_14 + a_13 =   -6.204330205150654e-08
```

We see that the problem arises at x_{14} because in x_{15}, both x_{14} and x_{13} will be negative, which means that if you add them together, they will become more negative, which in turn means that x_n will go towards $-\infty$. Since we are dealing with small number in the printouts over, there must have been a rounding error that caused x_{14} and x_{13} to be negative, which in turn snowballed into a huge error when the program calculated x_{100}.

Task 2

a)

```python
def binomial_coefficient(n, i):
    p = 1
    for j in range(1, i + 1):
        p *= (n - j + 1)/j
    return p


print(f"n =   5000, i =     4 ==> {binomial_coefficient(5000, 4)}")
print(f"n =   1000, i =  5000 ==> {binomial_coefficient(1000, 500)}
    ")
print(f"n = 100000, i = 99940 ==> {binomial_coefficient(100000,
    99940)}")
```

The reason why it's better to use a floating point number, is because a float can represent much larger numbers than an integer can.
The results from the program is shown under:

n = 5000, i = 4 ==> 26010428123750.0
n = 1000, i = 500 ==> 2.702882409454359e+299
n = 100000, i = 99940 ==> inf

b)

It is possible to get an overflow if the binomial coefficient is less than the maximum possible floating point number that can be represented on my machine, because if one wants to calculate a binomial coefficient where $i > n//2$ the coefficient will be smaller or equal than when $i = n//2$, and since the code essentially calculates all of the coefficients from $\binom{n}{1}$ to $\binom{n}{i}$, it can give an overflow during the calculation.

c)

No, the method is not as effective for all values, because if we look at the equation for the binomial coefficient:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{8}$$

If we choose that $i = n - i$ we get:

$$\begin{aligned}\binom{n}{n-i} &= \frac{n!}{(n-i)!(n-(n-i))!} \\ &= \frac{n!}{(n-i)!(n-n+i)!} \\ &= \frac{n!}{(n-i)!i!}\end{aligned} \tag{9}$$

This means that $\binom{n}{i} = \binom{n}{n-i}$, which also means that if we want to calculate the binomial coefficient of $\binom{n}{i}$ where $i > \frac{n}{2}$, then we can just calculate the binomial coefficient of $\binom{n}{n-i}$ and get the same answer. This means that we can optimize our program by first choosing the lowest value of i or $n - i$, and then calculate the binomial coefficient. Here is how the code would look like:

```
def binomial_coefficient(n, i):
    p = 1
    i = min(i, n-i)
    for j in range(1, i + 1):
        p *= (n - j + 1)/j
    return p


print(f"n =   5000, i =     4 ==> {binomial_coefficient(5000, 4)}")
print(f"n =   1000, i =  5000 ==> {binomial_coefficient(1000, 500)}
    ")
print(f"n = 100000, i = 99940 ==> {binomial_coefficient(100000,
    99940)}")
```

Task 3

a)

The program goes through a for loop where three random numbers between 0 and 1, and then compares $(x + y) \cdot z$ and $x \cdot z + y \cdot z$, which should yield the same result since this is just the distributive law. It then checks if they are not equal, and if that statement is true, it increments a fault counter by 1. The last thing the program does is to print out the error rate percentage, the last three random numbers chosen, and the error margin between the last two results.

b)

```
from random import random

antfeil = 0; N=100000

for i in range(N):
    x = random(); y = random(); z = random()
    res1 = (x + y) * (y + z)
    res2 = x*y + y*y + x*z + y*z
    if res1 != res2:
        antfeil += 1
        x0 = x; y0 = y; z0 = z
        ikkelik1 = res1
        ikkelik2 = res2

print (100.*antfeil/N)
print (x0, y0, z0, ikkelik1 - ikkelik2)
```

When running the code over, the first number that was printed out was $41,577$.

One reason that the first number differs in the two comparisons could be that since the program uses random numbers, the "success rate" of the comparisons will vary slightly. I think that the biggest reason why they differ is that since we are doing more multiplicative operations in the second program, the chance for the answers not coresponding with each other is greater.