MATH49111 Coursework 2

Rudi Agnew 1013652

Abstract

The aim of this project was to use different variable types when calculating a recurrence relation and then compare how each type fared in accuracy. All code was written in C++ whilst graphing done in MATLAB.

1 Using Float type variables

Here I was use **Float** type to store my variables and calculate the following recurrence relation

$$p_n = \frac{19}{3}p_{n-1} - 2p_{n-2}, \quad n \geq 2$$

with initial conditions

$$p_0 = 1 \quad \text{and} \quad p_1 = \frac{1}{3}.$$

I was also given that the unique solution to this recurrence relation with said initial conditions is

$$p_n = \left(\frac{1}{3}\right)^n, \quad n \geq 0.$$

I did this by creating a function that generated the n^{th} value of the recurrence sequence and then outputted the results to a file in the main function. I also included the relative and absolute error of the results compared to the analytic solution of p_n.

n	analytical	recurrence	relative error	abs error
0	1	1	0	0
1	0.333333	0.333333	2.98023e-08	9.93411e-09
2	0.111111	0.111111	4.76837e-07	5.29819e-08
3	0.037037	0.0370374	9.05991e-06	3.35552e-07
4	0.0123457	0.0123477	0.000163555	2.0192e-06
5	0.00411523	0.00412735	0.00294507	1.21196e-05
6	0.00137174	0.00144446	0.0530133	7.27206e-05
7	0.000457247	0.000893572	0.954243	0.000436325

8	0.000152416	0.00277037	17.1764	0.00261795
9	5.08053e-05	0.0157585	309.175	0.0157077
10	1.69351e-05	0.0942632	5565.15	0.0942462
11	5.64503e-06	0.565483	100173	0.565477
12	1.88168e-06	3.39287	1.80311e+06	3.39286
13	6.27225e-07	20.3572	3.24559e+07	20.3572
14	2.09075e-07	122.143	5.84207e+08	122.143
15	6.96917e-08	732.859	1.05157e+10	732.859
16	2.32306e-08	4397.15	1.89283e+11	4397.15
17	7.74352e-09	26382.9	3.40709e+12	26382.9
18	2.58117e-09	158298	6.13277e+13	158298
19	8.60392e-10	949785	1.1039e+15	949785
20	2.86797e-10	5.69871e+06	1.98702e+16	5.69871e+06

Clearly the error is increasing with n. To get a better picture I graphed the relative error against n with a logarithmic scale on the vertical axis.

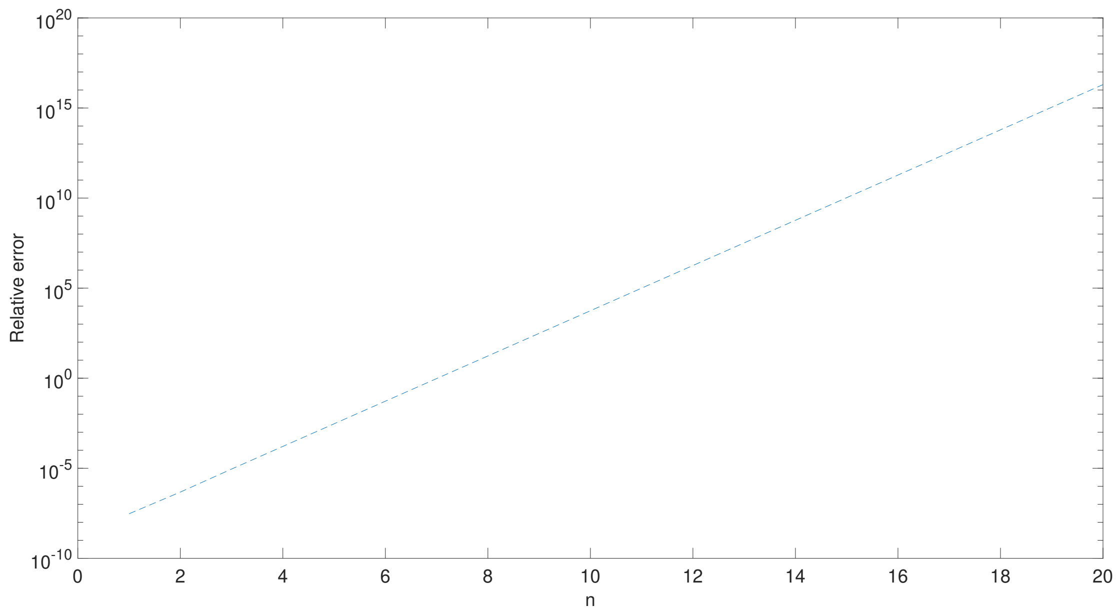

Figure 1: Plot of relative error vs n with a logarithmic scale.

The graph shows that the error increases by a factor of around 10 each iteration. This is due to the fact that every iteration we are multiplying one float by 19/3 and another by 2, adding these gives a total factor of around 10 that we are multiplying the floats by. Floats store around 7 decimal places accurately, multiply this once by 10 and you have 6 decimal places of accuracy then 5 then 4 etc. By the time we have hit n = 6 we have run the recurrence relation a total of 4 times and hence losing 4 decimal places of accuracy i.e. the value is now only accurate to 3 decimal places as seen.

2 Using Double type variables

I then used the same code just with double type variables instead.

n	analytical	recurrence	relative error	abs error
0	1	1	0	0
1	0.333333	0.333333	0	0
2	0.111111	0.111111	3.4972e-15	3.88578e-16
3	0.037037	0.037037	6.72587e-14	2.49106e-15
4	0.0123457	0.0123457	1.21838e-12	1.50418e-14
5	0.00411523	0.00411523	2.19377e-11	9.02785e-14
6	0.00137174	0.00137174	3.94889e-10	5.41686e-13
7	0.000457247	0.000457247	7.10801e-09	3.25012e-12
8	0.000152416	0.000152416	1.27944e-07	1.95007e-11
9	5.08053e-05	5.08051e-05	2.303e-06	1.17004e-10
10	1.69351e-05	1.69344e-05	4.14539e-05	7.02026e-10
11	5.64503e-06	5.64082e-06	0.000746171	4.21216e-09
12	1.88168e-06	1.8564e-06	0.0134311	2.52729e-08
13	6.27225e-07	4.75588e-07	0.241759	1.51638e-07
14	2.09075e-07	-7.00751e-07	4.35167	9.09826e-07
15	6.96917e-08	-5.38926e-06	78.33	5.45895e-06
16	2.32306e-08	-3.27305e-05	1409.94	3.27537e-05
17	7.74352e-09	-0.000196515	25378.9	0.000196522
18	2.58117e-09	-0.00117913	456821	0.00117913
19	8.60392e-10	-0.0070748	8.22277e+06	0.00707481
20	2.86797e-10	-0.0424488	1.4801e+08	0.0424488

This time the results are a lot more accurate, this is due to the fact doubles can store around 15/16 decimal places accurately compared to a floats 7. This is observed using another log plot comparing the relative errors.

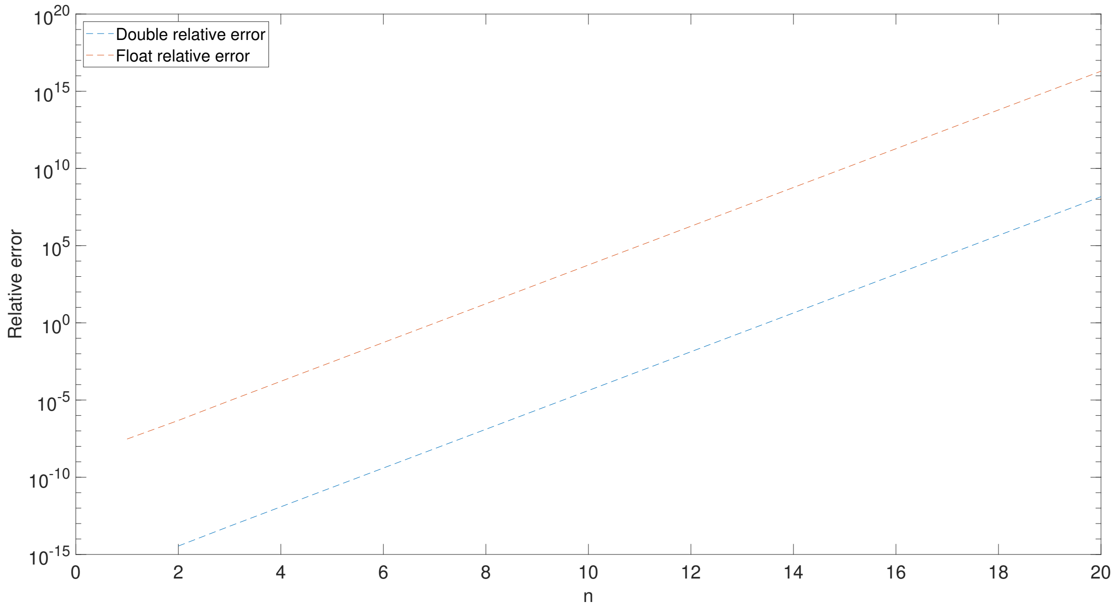

Figure 2: Plot of relative error for float and double type vs n with a logarithmic scale.

Both lines are parallel so the errors are increasing at the same rate as we are still multiplying by that factor of 10. However doubles start at a much lower error due to the higher initial precision. Hence once we hit n = 10 we have ran the recurrence relation 8 times resulting in a loss of 8 decimal places of accuracy approximately, which is seen.

A Source code

Listing 1: recurrence_f

```
float recurrence_f(int n)
{
        // calculates nth term in the recurrence relation with floats
        float p1 = 1.0f / 3, p0 = 1.0f, pn = 0.0f;
        if (n == 0)
        {
                return p0;
        }
        else if (n == 1)
        {
                return p1;
```

```
        }
        else
        {
                for (int i = 2; i <= n; i++)
                {
                        pn = 19.0f / 3 * p1 - 2.0f * p0;
                        p0 = p1;
                        p1 = pn;
                }
                return pn;
        }

}
```

Listing 2: main

```
int main()
{
        // declare object 'File2'
        std::ofstream File2;

        // try to open File2
        File2.open("data2.txt");

        //if opening fails , exit main
        if (!File2) return 1;

        // labelling data columns
        File2.width(20); File2 << "i";
        File2.width(20); File2 << "analytical";
        File2.width(20); File2 << "recurrence";
        File2.width(20); File2 << "relative_error";
        File2.width(20); File2 << "abs_error" << std::endl;

        for (int i = 0; i <= 20; i++)
        {
                // abs error |x-x'|, relative error |(x-x')|/|x|
                File2.width(20); File2 << i;
                File2.width(20); File2 << analytical(i);
```

```
                File2.width(20); File2 << recurrence_f(i);
                File2.width(20); File2 << abs(analytical(i) - recurrence_f(i))
                / abs(analytical(i));
                File2.width(20); File2 << abs(analytical(i) - recurrence_f(i))
                << std::endl;
        }
        File2.close();
        return 0;
}
```