This is the revised version of homework 3, with corrections highlighted in red.

1) Let $f(x) = \sqrt{x}$

a) Find the absolute and relative condition numbers of f.

The absolute and relative condition numbers for a function f are given as

$$C(x) = |f'(x)|$$
$$\kappa(x) = \left|\frac{xf'(x)}{f(x)}\right|.$$

Since $f = \sqrt{x}$, then $f'(x) = \frac{1}{2\sqrt{x}}$, and

$$C(x) = \frac{1}{2\sqrt{x}}$$
$$\kappa(x) = \frac{1}{2\sqrt{x}}\frac{x}{\sqrt{x}} = \frac{1}{2}.$$

b) Where is f well conditioned in an absolute sense? Where is f well conditioned in a relative sense?

In an absolute sense, we can see that the absolute condition number is inversely proportional to the root of x, so it is not well conditioned near $x = 0$, but it becomes more well conditioned at larger values of x.

In a relative sense, the function is well conditioned everywhere since $\kappa(x)$ is a half everywhere (and at zero the limit approaches a half as well).

c) Suppose $x = 10^{-17}$ is replaced by $\hat{x} = 10^{-16}$. Using the absolute condition number of f, how much of a change is expected in f due to this change in the argument?

Recall $C(x) = \frac{1}{2\sqrt{x}}$, so we expect

$$|\hat{y} - y| = C\big(10^{-17}\big)\big|10^{-17} - 10^{-16}\big| \approx 1.423 \cdot 10^{-8}.$$

2) Lagrange interpolation

a) Determine the Lagrange form of the interpolating polynomial for the data (2,1), (3,3), and (4,5). What is the degree of this interpolating polynomial?

Recall that the i^{th} Lagrange polynomial for a data set with n data points is given as

$$L_i(x) = \prod_{k \neq i}^{n} \frac{x - x_k}{x_i - x_k}.$$

Hence we have the Lagrangian basis for the given data points:

$$\begin{aligned} L_0(x) &= \frac{(x-3)(x-4)}{(2-3)(2-4)} = \frac{(x-3)(x-4)}{2} \\ L_1(x) &= -(x-2)(x-4) \\ L_2(x) &= \frac{(x-2)(x-3)}{2}. \end{aligned}$$

From the Lagrangian basis, we can determine an interpolating polynomial as

$$p(x) = \sum_{i=0}^{n} y_i L_i(x).$$

For this problem, we then have

$$\begin{aligned} p(x) &= 1\left[\frac{(x-3)(x-4)}{2}\right] + 3[-(x-2)(x-4)] + 5\left[\frac{(x-2)(x-3)}{2}\right] \\ &= \frac{1}{2}(x-3)(x-4) - 3(x-2)(x-4) + \frac{5}{2}(x-2)(x-3). \end{aligned}$$

Observe that the interpolating polynomial is of degree 2.

b) Determine the a Lagrange form of the interpolating polynomial for the data (2,2), (3,3) and (4,5). What is the degree of this interpolating polynomial?

Since the x-values are the same as in part (a), the Lagrangian basis is the same, and the interpolating polynomial is slightly modified:

$$\begin{aligned} p(x) &= \frac{2}{2}(x-3)(x-4) - 3(x-2)(x-4) + \frac{5}{2}(x-2)(x-3) \\ &= (x-3)(x-4) - 3(x-2)(x-4) + \frac{5}{2}(x-2)(x-3). \end{aligned}$$

Notice that the degree of the polynomial is of degree 2 as in part (a).

c) Based on your observation, is the interpolating polynomial always of degree N for $N+1$ data points? Explain why.

Yes, the interpolating polynomial is always (by construction) of degree n when we have $n+1$ data points since we can solve for $n+1$ coefficients, meaning we can solve for $a_0, a_1, \ldots, a_n$.

We know that the interpolating polynomial has the form $a_n x^n + \ldots + a_1 x + a_0$, where the set of $n+1$ coefficients $\{a_0, a_1, \ldots, a_n\}$ is unique. It is clear that if a_n is nonzero, then our interpolating polynomial is of degree n. However, it may not always be the case that $a_n \neq 0$. For example, suppose we may have data which is interpolated by a linear polynomial (i.e. a line), but we have more than 2 data points.

3) For the data (-2,0), (0,4), and (1,9).

a) Determine the power series form of the interpolating polynomial, i.e. $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$.

For this set, we have 3 data points which allows us to solve for 3 coefficients. Equivalently, we then have a 2nd degree polynomial of the form

$$p\left(x\right) = a_0 + a_1 x + a_2 x^2.$$

Using the restriction that $p\left(x_i\right) = y_i$, we have the following matrix equation:

$$\begin{pmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 9 \end{pmatrix}.$$

Solving using standard linear algebra techniques gives

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 1 \end{pmatrix}.$$

This gives the interpolating polynomial as

$$p(x) = 4 + 4x + x^2.$$

b) Determine the Lagrange form of the interpolating polynomial.

The Lagrange form is

$$\begin{aligned} p(x) &= 4\left[\frac{\left(x+2\right)\left(x-1\right)}{-2}\right] + 9\left[\frac{\left(x+2\right)x}{3}\right] \\ &= 3x\left(x+2\right) - 2\left(x+2\right)\left(x-1\right). \end{aligned}$$

And simplifying for part (d) we find

$$p\left(x\right) = x^2 + 4x + 4.$$

c) Construct a divided difference table. Then determine the Newton divided difference interpolating polynomial

The equations dictate the construction of the divided difference table.

$$\begin{cases} F_{i,0} = f\left(x_i\right) \\ F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}} \end{cases}.$$

Thus, the table is:

$$\begin{array}{c||c|c|c} -2 & 0 & - & - \\ 0 & 4 & 2 & - \\ 1 & 9 & 5 & 1 \end{array}$$

The diagonal elements give the coefficients for Newton's interpolating polynomial, which is

$$\begin{aligned} p(x) &= 2(x+2) + x(x+2) \\ &= x^2 + 4x + 4. \end{aligned}$$

d) Do you answer for (a), (b), and (c) agree? Expain why.

As can be seen, all the different forms of polynomial interpolation given the same answer. This must be the case since there is a theorem which gurantees that a polynomial which interpolates a given set of data is unique. That is, if a polynomial interpolates the given data, then all other forms of polynomials which interpolate the data are in fact identical.

4) Write and execute a program to compute

$$\begin{aligned} f(x) &= \sqrt{x^2+1} - 1 \\ g(x) &= \frac{x^2}{\sqrt{x^2+1}+1}. \end{aligned}$$

for $x = 4^{-k}$, $k = 1, 2, \ldots, 20$. Please list a table for the function values and explain why.

The table is calculated (and formatted in LaTex!) with a python script, which is shown below the table.

k	x	$f(x)$	$g(x)$
1	2.500e-01	3.078e-02	3.078e-02
2	6.250e-02	1.951e-03	1.951e-03
3	1.562e-02	1.221e-04	1.221e-04
4	3.906e-03	7.629e-06	7.629e-06
5	9.766e-04	4.768e-07	4.768e-07
6	2.441e-04	2.980e-08	2.980e-08
7	6.104e-05	1.863e-09	1.863e-09
8	1.526e-05	1.164e-10	1.164e-10
9	3.815e-06	7.276e-12	7.276e-12
10	9.537e-07	4.547e-13	4.547e-13
11	2.384e-07	2.842e-14	2.842e-14
12	5.960e-08	1.776e-15	1.776e-15
13	1.490e-08	0.000e+00	1.110e-16
14	3.725e-09	0.000e+00	6.939e-18
15	9.313e-10	0.000e+00	4.337e-19
16	2.328e-10	0.000e+00	2.711e-20
17	5.821e-11	0.000e+00	1.694e-21
18	1.455e-11	0.000e+00	1.059e-22
19	3.638e-12	0.000e+00	6.617e-24
20	9.095e-13	0.000e+00	4.136e-25

```python
#!/usr/bin/env python3

import numpy as np
import pandas as pd

k = np.arange(1,21)
x = 4.0**(-k)
f = np.sqrt(x**2 + 1) - 1
g = x**2/(np.sqrt(x**2 + 1) + 1)

table = pd.DataFrame({
    '$k$': k,
    '$x$': x,
    '$f(x)$': f,
    '$g(x)$': g
    })

table.to_latex(buf='prob4.tex',
        index=False,
        escape=False,
        column_format='cccc',
        float_format="%.3e")
```

Here we can see that at around $k = 13$ the values for f and g begin to differ significantly even though $f = g$ except at $x = 0$, where g is not defined (although the limit of g as x

tends to zero is 0). This is a conditioning problem. It is observed that

$$f'(x) = \frac{x}{\sqrt{x^2+1}},$$

and

$$g'(x) = \frac{x\left(x^2+2\sqrt{x^2+1}+2\right)}{x^2\sqrt{x^2+1}+2x^2+2\sqrt{x^2+1}+2}.$$

It is clear that f is well conditioned generally, but g is not well conditioned as $x \to 0$ since the derivative does not exist at $x = 0$. It is also possible that round-off error occurs here since $x^2 \approx 0$, likely too small to be differentiated. Thus, f evaluates to $1 - 1 = 0$, while g gives $x^2/2$.

It was misstated above that f and g were conditioned differently around $x = 0$, but taking derivatives it is clear that this is not the case. Indeed the second explanation given about round-off error is the correct description for the numerical instability observed in the results. We actually generate another column for $\frac{x^2}{2}$ to demonstrate that for smaller values of 4^{-k}, g and the $\frac{x^2}{2}$ agree well.

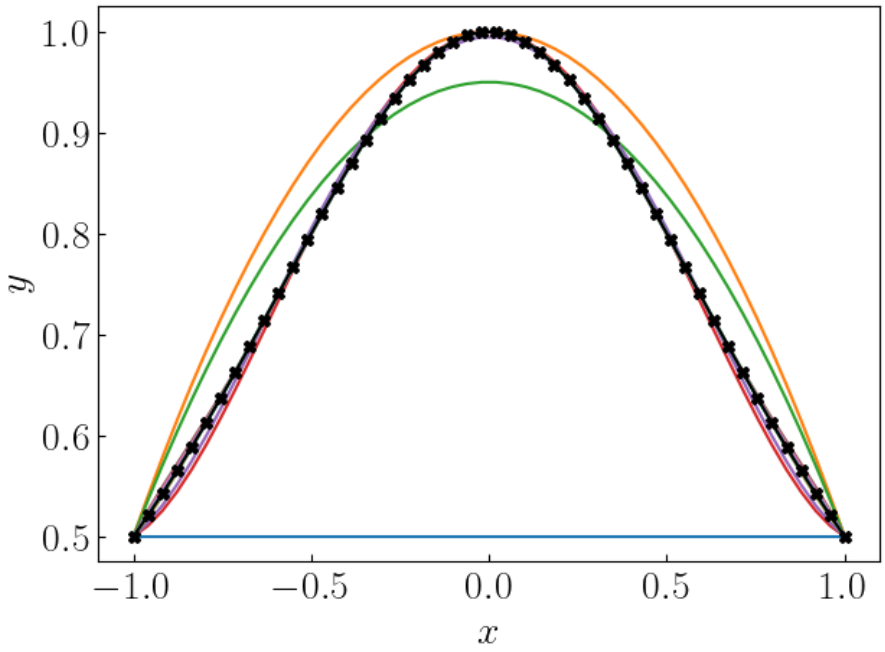
k	x	$f(x)$	$g(x)$	$x^2/2$
1	2.500e-01	3.078e-02	3.078e-02	3.125e-02
2	6.250e-02	1.951e-03	1.951e-03	1.953e-03
3	1.562e-02	1.221e-04	1.221e-04	1.221e-04
4	3.906e-03	7.629e-06	7.629e-06	7.629e-06
5	9.766e-04	4.768e-07	4.768e-07	4.768e-07
6	2.441e-04	2.980e-08	2.980e-08	2.980e-08
7	6.104e-05	1.863e-09	1.863e-09	1.863e-09
8	1.526e-05	1.164e-10	1.164e-10	1.164e-10
9	3.815e-06	7.276e-12	7.276e-12	7.276e-12
10	9.537e-07	4.547e-13	4.547e-13	4.547e-13
11	2.384e-07	2.842e-14	2.842e-14	2.842e-14
12	5.960e-08	1.776e-15	1.776e-15	1.776e-15
13	1.490e-08	0.000e+00	1.110e-16	1.110e-16
14	3.725e-09	0.000e+00	6.939e-18	6.939e-18
15	9.313e-10	0.000e+00	4.337e-19	4.337e-19
16	2.328e-10	0.000e+00	2.711e-20	2.711e-20
17	5.821e-11	0.000e+00	1.694e-21	1.694e-21
18	1.455e-11	0.000e+00	1.059e-22	1.059e-22
19	3.638e-12	0.000e+00	6.617e-24	6.617e-24
20	9.095e-13	0.000e+00	4.136e-25	4.136e-25

5) For $f(x) = \frac{1}{1+x^2}$ on $[-1, 1]$, write a code for polynomial interpolation, where the interpolating points are uniformly distributed on the interval. Plot the figures of $f(x)$ and $p_n(x)$, with $n = 1, 2, \ldots, 10$.

For this problem we use Lagrange interpolation since it is the easiest to code and there is no advadvantage to using divided differences since there is no data in common between sets other than at the end points. The code is shown below:

```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['text.latex.preamble'] = r'\usepackage{amsmath}'
rcParams['text.usetex']         = True
rcParams['font.family']         = 'sans-serif'
rcParams['font.sans-serif']     = ['Helvetica']

def Lagrange_poly(t,x,i):
    res = 1.0
    for j in range(len(x)):
        if j == i:
            continue
        else:
            res *= (t-x[j])/(x[i]-x[j])
    return res

def Lagrange_interp(t,x,y):
    res = 0.0
    for i in range(len(x)):
        res += y[i]*Lagrange_poly(t,x,i)
    return res

if __name__ == '__main__':

    f = lambda x: 1.0/(1.0 + x**2.0)
    T = np.linspace(-1.0,1.0)

    fig,ax = plt.subplots(nrows=1,ncols=1,figsize=(7,5))
    for n in range(1,11):
        x_data = np.linspace(-1.0,1.0,n+1)
        y_data = f(x_data)

        interp = np.array([Lagrange_interp(t,x_data,y_data) for t in T])
        ax.plot(T,interp)

    ax.plot(T,f(T),'k-X')
    ax.set_xlabel(r'$x$',size=20)
    ax.set_ylabel(r'$y$',size=20)
    ax.tick_params(axis='both',which='major',labelsize=20,direction='in')
    plt.savefig('fig1.png',bbox_inches='tight')
```

The resulting plot is shown below:



Note that the darker marks on the plot highlight the location of $f(x)$. It can be seen that as more points are added, the interpolating polynomial becomes a better approximation of the function, oscillating about the function with nodes at the data points.