Problem 1:

1. Calculate the Taylor series at the expansion point $x = 0$ of the functions $\frac{1}{x-1}$ and $\frac{1}{x^2+1}$, and plot the Taylor polynomials for $n \in \{10, 20, 40\}$ on the interval $[-2, 2]$. Hint: A partial fraction decomposition of the second function explains the bad behaviour of the harmless looking function.

$$f(x) := \frac{1}{x - 1}$$

A taylor series of a given function f(x) at a point a is given as:

$$g(x) := \frac{1}{x^2 + 1}$$

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots,$$

Writing down the first derivatives of our functions and calculating for **a=0** gives us:

$$f_1(x) := \frac{d^1}{dx^1} f(x) \to \frac{-1}{(x-1)^2} \qquad f_1(0) \to -1$$

$$f_2(x) := \frac{d^2}{dx^2} f(x) \to \frac{2}{(x-1)^3} \qquad f_2(0) \to -2$$

$$f_3(x) := \frac{d^3}{dx^3} f(x) \to \frac{-6}{(x-1)^4} \qquad f_3(0) \to -6$$

$$f_4(x) := \frac{d^4}{dx^4} f(x) \to \frac{24}{(x-1)^5} \qquad f_4(0) \to -24$$

$$g_1(x) := \frac{d^1}{dx^1} g(x) \to \frac{-2}{\left(x^2+1\right)^2} \cdot x \qquad g_1(0) \to 0$$

$$g_2(x) := \frac{d^2}{dx^2} g(x) \to \frac{8}{\left(x^2+1\right)^3} \cdot x^2 - \frac{2}{\left(x^2+1\right)^2} \qquad g_2(0) \to -2$$

$$g_3(x) := \frac{d^3}{dx^3} g(x) \to \frac{-48}{\left(x^2+1\right)^4} \cdot x^3 + \frac{24}{\left(x^2+1\right)^3} \cdot x \qquad g_3(0) \to 0$$

$$g_4(x) := \frac{d^4}{dx^4} g(x) \to \frac{384}{\left(x^2+1\right)^5} \cdot x^4 - \frac{288}{\left(x^2+1\right)^4} \cdot x^2 + \frac{24}{\left(x^2+1\right)^3} \qquad g_4(0) \to 24$$

Thus we can determine the following pattern for both taylor series:

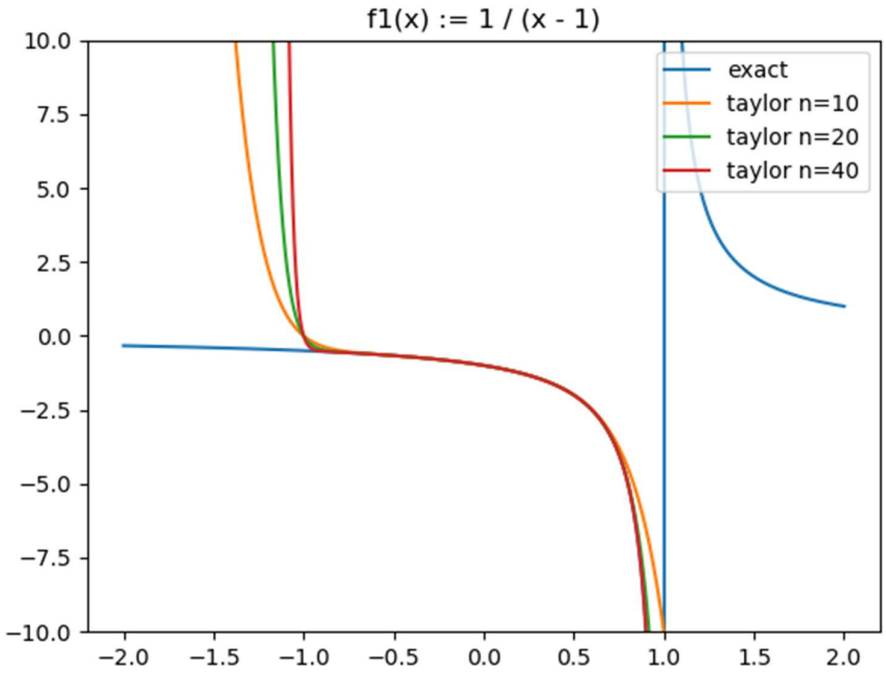
$$T_f(x) := -1 - 1 \cdot x - 1 \cdot x^2 - 1 \cdot x^3 - 1 \cdot x^4 - 1 \cdot x^5 - 1 \cdot x^6 - 1 \cdot x^7 - 1 \cdot x^8 - 1 \cdot x^9 - \ldots.$$

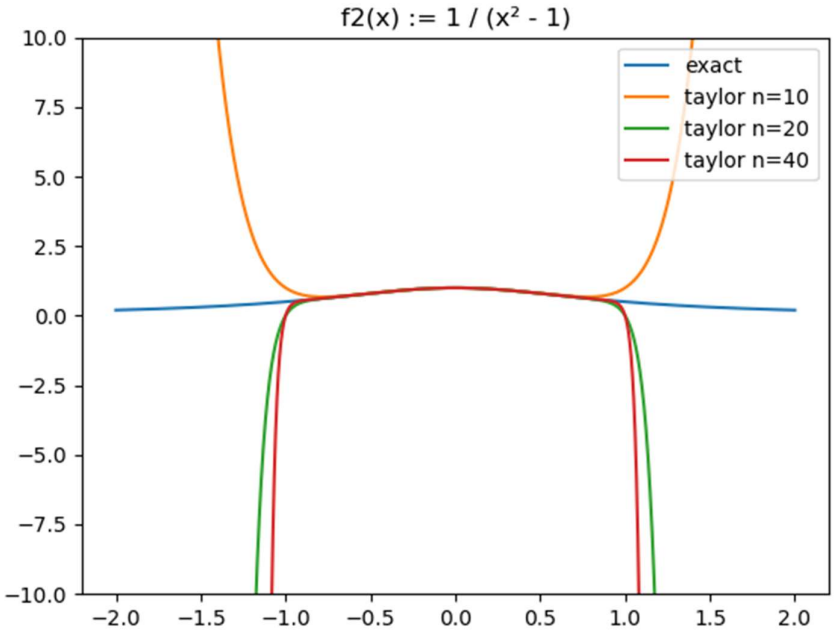
$$T_g(x) := 1 - 1 \cdot x^2 + 1 \cdot x^4 - 1 \cdot x^6 + 1 \cdot x^8 - \ldots.$$

The functions and their respective taylor series are implemented in Python as follows:

```
# Analytical function for f1
def f1(x):
    return 1.0 / (x - 1.0)

# Tatylor series for f1
# -1 - x - x^2 - x^3 - ...
def f1_taylor(x, n):
    ret_val = 0
    for i in range(n):
        ret_val = ret_val + -1.0*x**i
    return ret_val

# Analytical function for f2
def f2(x):
    return 1.0 / (x**2 + 1)

# Tatylor series for f2
# 1 - x^2 + x^4 - x^6 + x^8 ...
def f2_taylor(x, n):
    ret_val = 0
    for i in range(n):
        if not (i % 2): # if even
            # alternating signs
            if i % 4:
                ret_val -= x**i
            else:
                ret_val += x**i
    return ret_val
```

Plotting those functions with 1000 points each gives us the following results:





Conclusions:

- The taylor series can be a good approximation for non-polynomial functions within a certain area.

- The approximation is best around the point a where it is derived from. The further away we go from a the bigger the error
- If the original function is periodic or converges towards a finite value the maximum error of the Taylor approximation will always approach infinity because every polynomial of order >0 will go towards infinity
- The Taylor approximation is unable to correctly represent the pole in f1
- The higher the order of the Taylor series the better approximation. However increasing the order comes with diminishing returns. I.e. n=20 is better than n=10 but n=40 is only marginally better than n=20

Problem 2:

2. Compute and plot interpolation polynomials to $\frac{1}{x^2+1}$ on the interval $[-5,5]$, for $n \in \{10, 20, 40\}$. Choose uniformly distributed points, and Chebyshev points on $[-5,5]$: $x_i = 5\cos\frac{(i+0.5)\pi}{n+1}$ for $i = 0, \ldots, n$.

   Plot the Lagrange interpolation polynomials l_i for both choices of points. Investigate numerically

$$\max_{i\in\{0,\ldots,n\}} \max_{x\in[-5,5]} |l_i(x)| \qquad \text{and} \qquad \max_{x\in[-5,5]} \sum_{i=0}^{n} |l_i(x)|$$

   depending on n.

The polynomial interpolation is given by:

Theorem 1.1 (Lagrange interpolation) *Let the points ("knots") x_i, $i = 0, \ldots, n$, be distrinct. Then there exists, for all values $(f_i)_{i=0}^n \subset \mathbb{R}$, a unique interpolating polynomial $p \in \mathcal{P}_n$. It is given by*

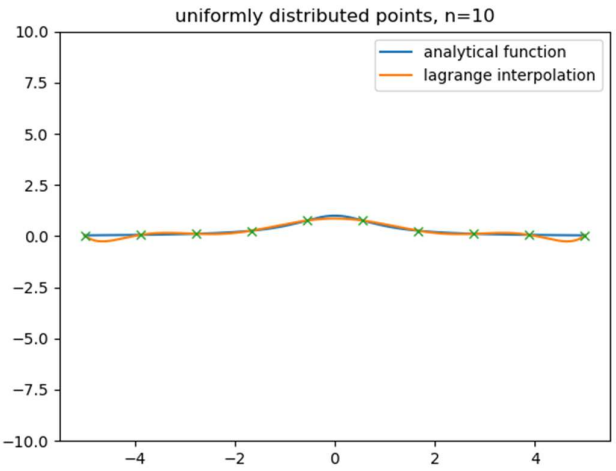
$$p(x) = \sum_{i=0}^{n} f_i \ell_i(x), \qquad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{1.2}$$

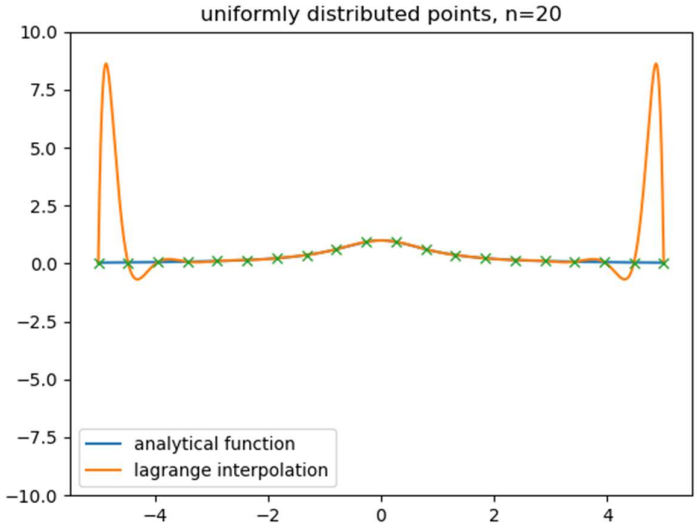
The polynomials $(\ell_i)_{i=0}^n$ are called Lagrange basis *of the space $\mathcal{P}_n$ w.r.t. the points $(x_i)_{i=0}^n$.*

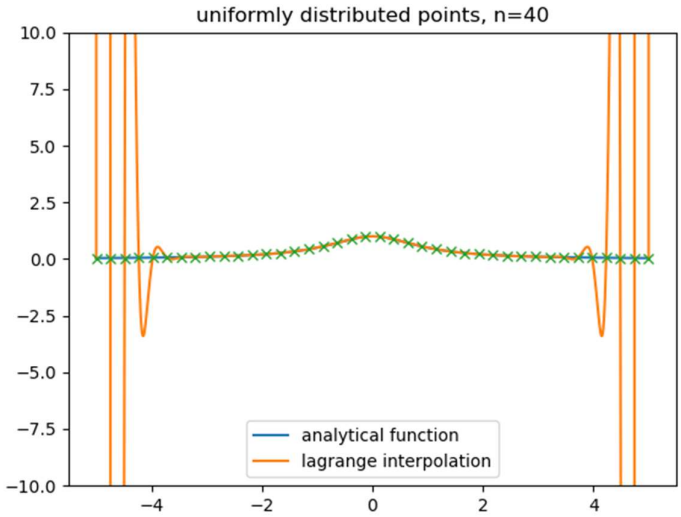
We can implement p(x), l_i(x) and our given function func = $(x^2+1)^{-1}$ generally in Python as follows:

```
# l_i
def Lagrange (pts, i, x):
    prod = 1
    for j in range(len(pts)):
        if j != i:
            prod = prod * (x-pts[j])/(pts[i]-pts[j])
    return prod

# p(x)
def InterpolationPolynomial (fun, pts, x):
    sum = 0
    for i in range(len(pts)):
        sum = sum + fun(pts[i]) * Lagrange(pts, i, x)
    return sum

# f(x) = 1 / (x² + 1)
# returns a number if x is a number or a list if x is iterable object
def func(x):
    if (type(x) == np.float64) or (type(x) == float): return 1.0/(1 + x*x)
    return [1.0/(1 + x_i*x_i) for x_i in x]
```

Plotting the interpolated functions with 1000 points each gives the following results:


uniformly distributed points, n=10
analytical function
lagrange interpolation
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−4
−2
0
2
4


uniformly distributed points, n=20
analytical function
lagrange interpolation
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−4
−2
0
2
4


uniformly distributed points, n=40
analytical function
lagrange interpolation
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−4
−2
0
2
4

It can be clearly seen that for higher orders the lagrange interpolation leads to some “interesting” behaviour.

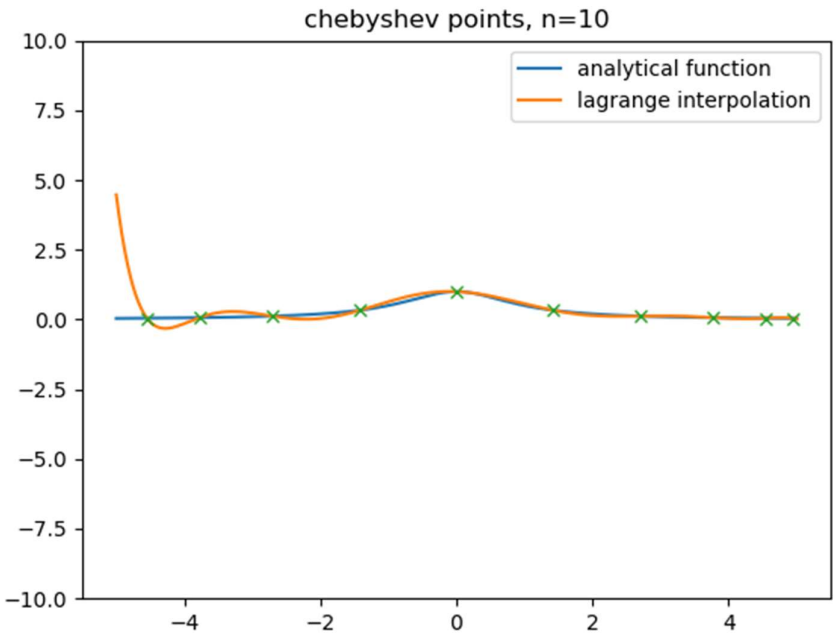
One way of fixing this issue is to use Chebyshev points as described in the instructions to the problem.

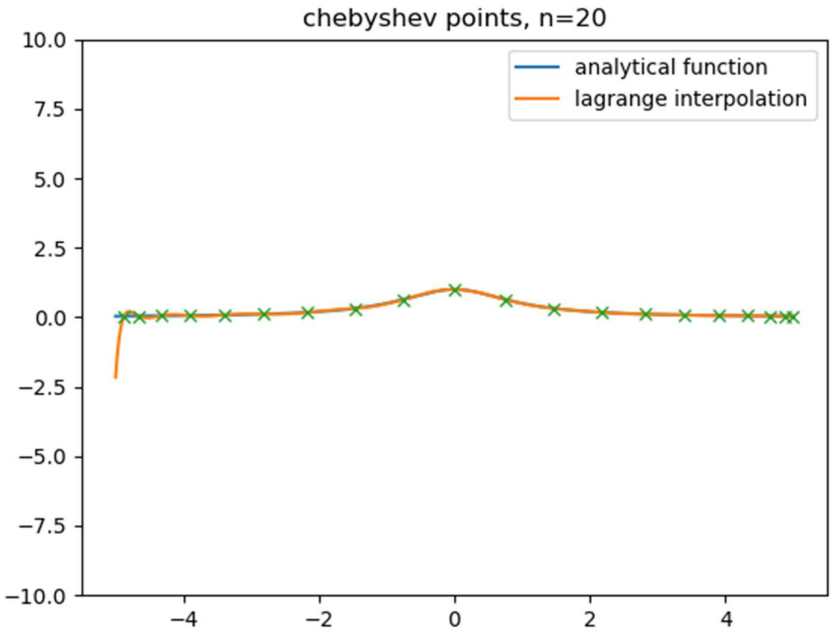
$$x_i = 5\cos\frac{(i+0.5)\pi}{n+1} \text{ for } i = 0, \ldots, n.$$

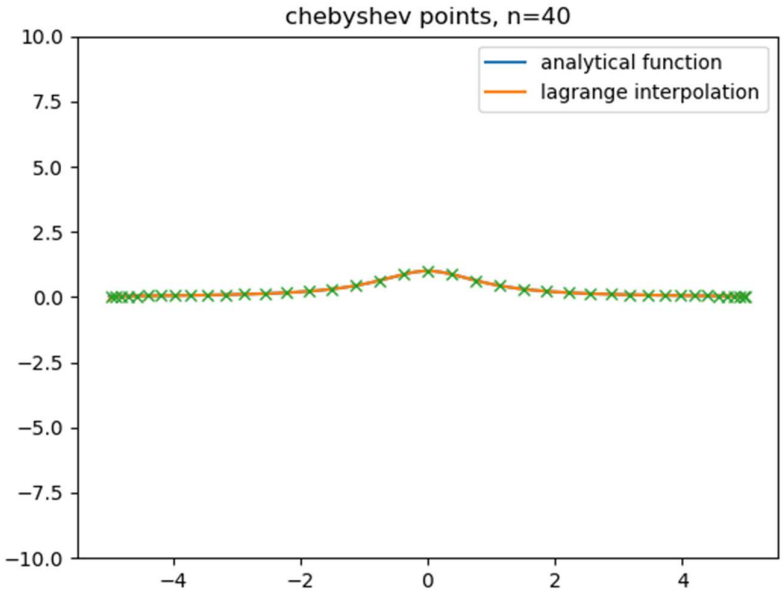
This is implemented in Python as follows:

```
def chebyshev_points(start, end, count):
    return [5*math.cos( (i+0.5)*pi / (count + 1)) for i in range(count)]
```

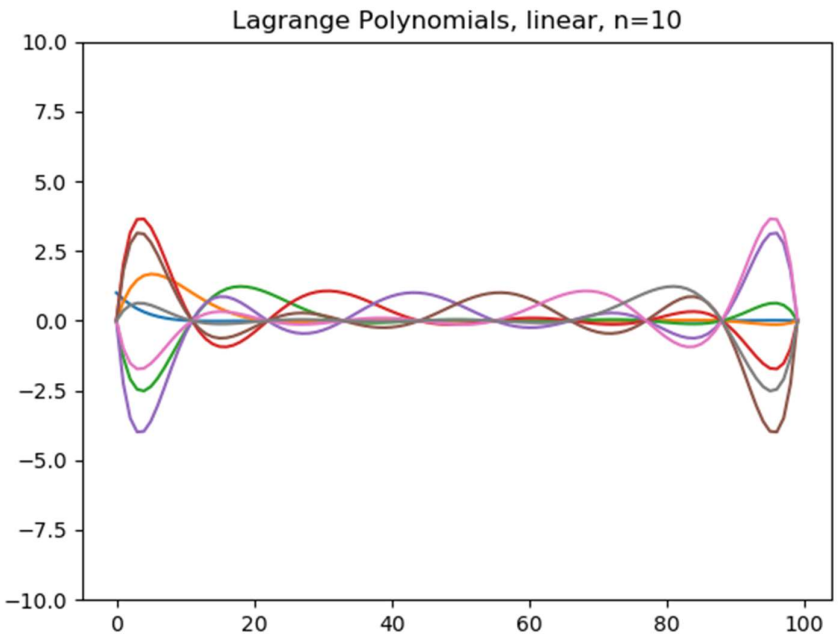
Using these points we get the following interpolations for n = 10,20,40

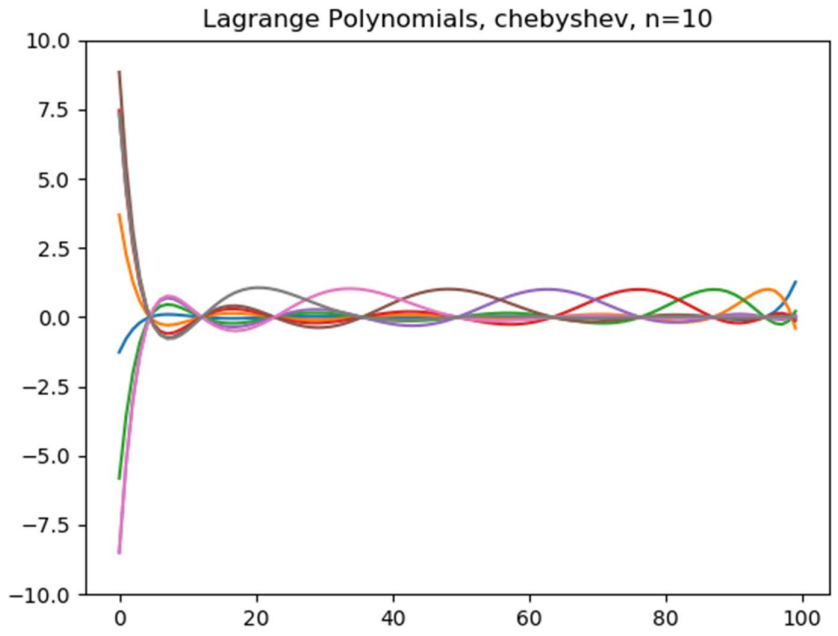



chebyshev points, n=20
analytical function
lagrange interpolation
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−4
−2
0
2
4


chebyshev points, n=40
analytical function
lagrange interpolation
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−4
−2
0
2
4

The first 8 basis functions (Lagrange polynomials) for linear distributed and chebyshev points for n=10 are as follows:





Last we are asked to plot the following maxima depending on n:

$$\max_{i\in\{0,\dots,n\}} \max_{x\in[-5,5]} |l_i(x)| \qquad \text{and} \qquad \max_{x\in[-5,5]} \sum_{i=0}^{n} |l_i(x)|$$

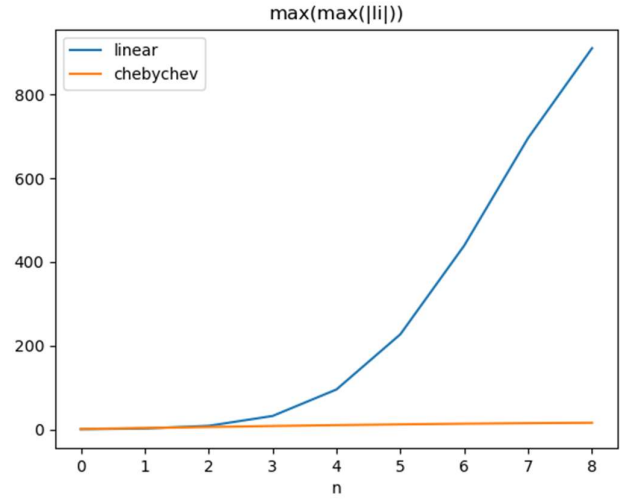
The maxima are calculated using:

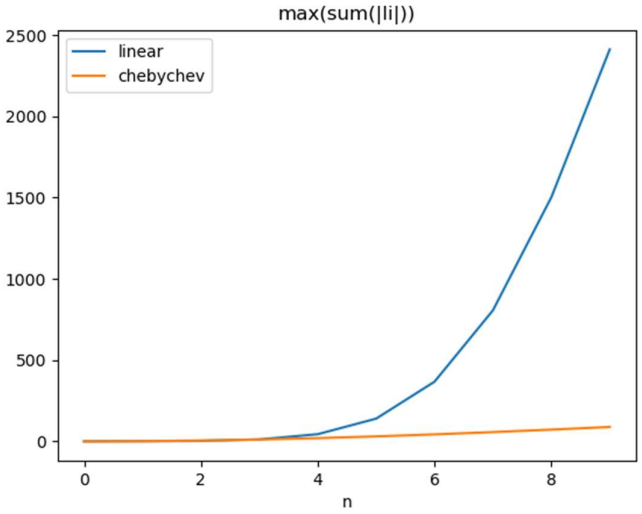
```
# max ( max (|l_i(x)|))
max_vals1 = [max([max(abs(Lagrange(pts, i, x))) for i in range(n)]) for n in range(1,n_max)]
```

and

```
# max ( sum( |l_i(x)| ))

max_vals2 = []

for n in range(n_max):
    sum = np.zeros(len(x), dtype=float)
    for i in range(n):
        sum += abs(Lagrange(pts, i, x))
    max_vals2.append(max(sum))

return max_vals1, max_vals2
```

plotting these for both types of points gives us:





Conclusions:

- The lagrange interpolation is a simple but numerically expensive ($O(n^3)$) way to approximate functions
- Equally distributed points may have the tendency to oscillate around the original function. This tendency is known as Runge's phenomenon and gets worse with an increasing number of points.
- One way of solving this issue is to use Chebyshev points

Problem 3

3. Chebyshev polynomials are recursively defined as

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \qquad \text{for } n = 1, 2, \ldots \end{aligned}$$

- Plot the first few polynomials on $[-1, 1]$. What range do you expect?
- Determine the leading coefficient $lc(T_n)$.
- Show that there holds

$$T_n(x) = \cos(n \operatorname{acos}(x)) \qquad \text{for } x \in [-1, 1]$$

  Find all roots of T_n.
- Let q be a polynomial of exact degree n such that $lc(q) = lc(T_n)$. Proof that

$$\max_{x\in[-1,1]} |q(x)| \geq \max_{x\in[-1,1]} |T_n(x)|$$

Plot:

Chebyshev polynomials can be recursively defined in Python as follows:

```
def T_n(x, n):
    if n == 0: return 1
    if n == 1: return x
    return 2*x*T_n(x,n-1) - T_n(x, n-2)
```

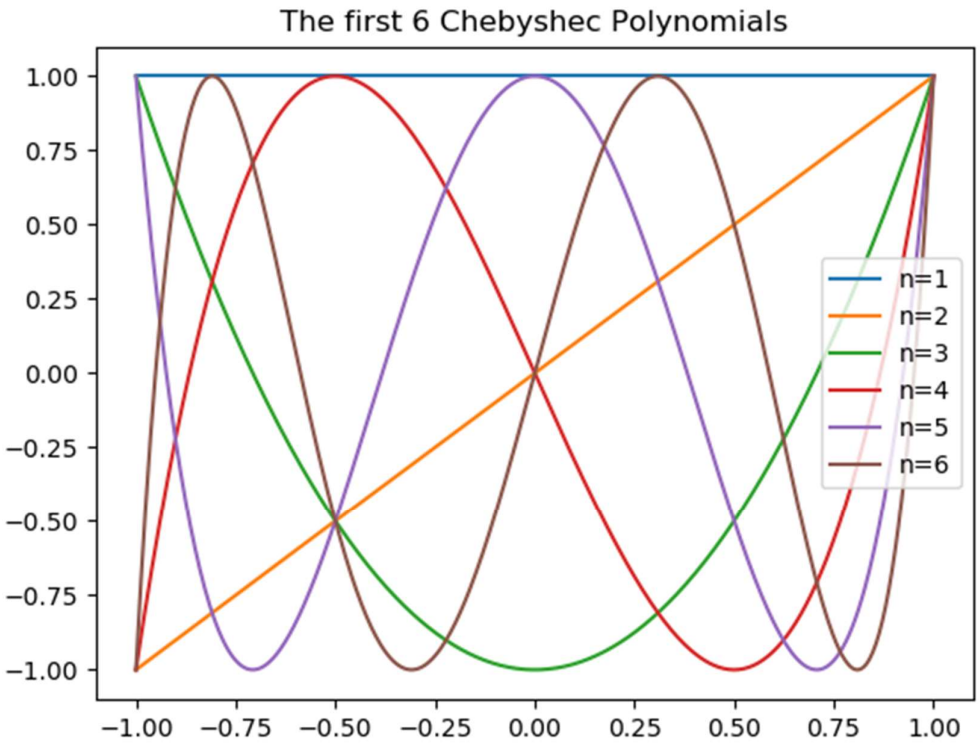
Looking at $2xT_n(x)$: As long as $x \in [-1,1]$ every $T_n(x)$ can only return a value between -1 and 1. Thus $2x$ is multiplied by a number between -1 and 1. Therefore it can never be above 1. A maximum can be found at x = 1 and n = 2 =>

$$2xT_n(x, n-1) - T_n(x, n-2) = 2 * 1 * T_n(1,1) - T_n(1,0) = 2 * 1 - 1 = 1$$

The minimum can be calculated the same by using x = -1 and n = 2 => min = -1

Thus we can expected polynomials to be in the range of [-1,1]. Beyond that T_n will be multiplied by some number >1 which drives the function to infinity

Plotting the first polynomials for $x \in [-1,1]$ gives us:



Leading Coefficients:

To determine the leading coefficient, we can look at the recursive formula with. The expression $2xT(x, n-1)$ results in the previous leading coefficient always to be doubled. This leads to the following pattern of leading coeffients: 1, 2, 4, 8, …

Therefore we can derive the following formula:

$$lc(T_n) = 2^{n-1}$$

Proof 1:

We can show that

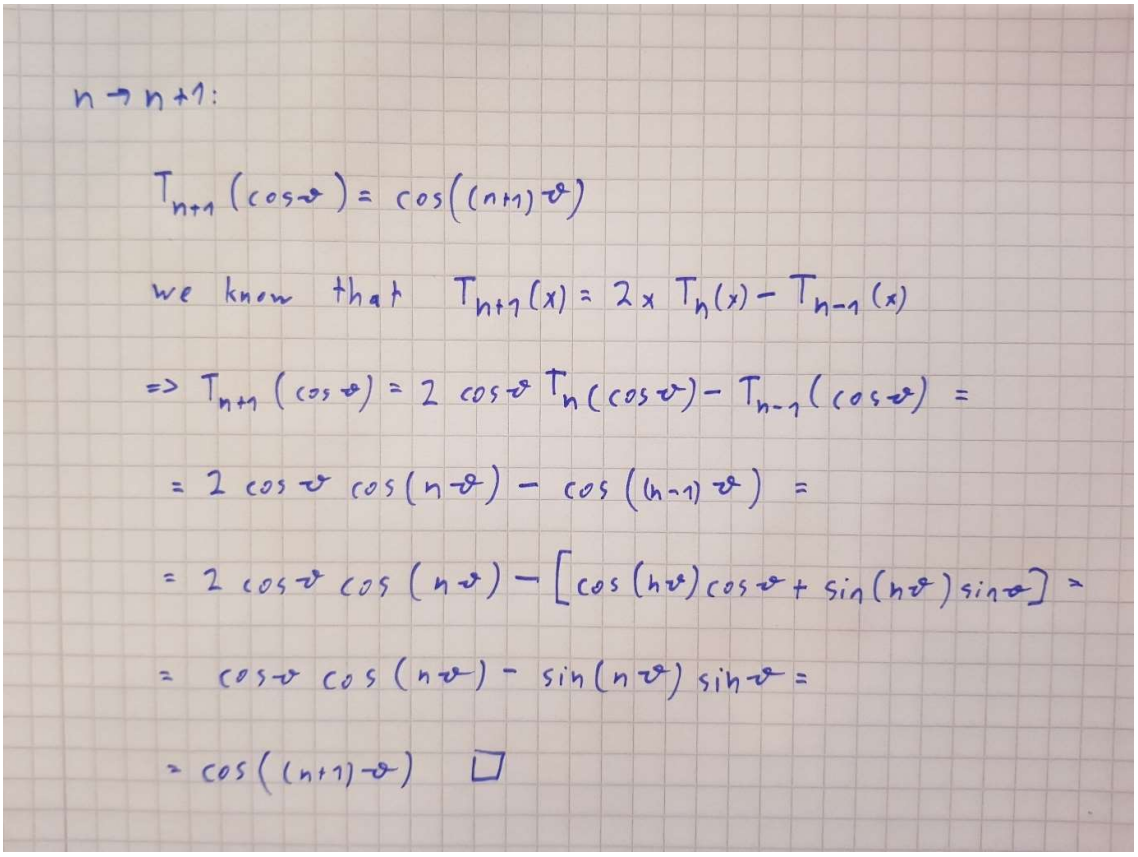
$$T_n(x) = \cos(n \operatorname{acos}(x)) \qquad \text{for } x \in [-1, 1]$$

By using making a transformation from $x \rightarrow \cos(\theta)$ and using proof by induction

$$T_n(x) = \cos\big(n\ acos(x)\big) \rightarrow T_n(\cos(\theta)) = \cos(n \operatorname{acos}(\cos(\theta))) = \cos(n\,\theta)$$

n=1:

$$T_1(\cos(\theta) = \cos(1\,\theta) = x$$



Roots:

Using the transformation from before

$$T_n(\cos(\theta)) = \cos(n\theta) = 0$$

The roots of the cosine function are given by:

$$\cos\left((2k+1)\frac{\pi}{2}\right) = 0 \quad k = 1,2,3,\dots$$

$$\Longrightarrow \cos(n\,\theta) = \cos\left((2k+1)\frac{\pi}{2}\right)$$

$$n\,\theta = (2k+1)\frac{\pi}{2}$$

$$\cos(\theta) = \cos\left(\frac{2k+1}{2n}\,\pi\right) = x$$

Proof2:

Let $p(x) \coloneqq q(x) - T_n(x)$ Since q and T_n have the same leading coefficient p(x) is a polynomial of degree n-1

No we do proof by contradiction and assume: $|q(x)| < |T_n(x)|$

We can calculate the maxima of T_n with

$$T_n(x_k) = 1 \qquad x_k = \frac{2k\pi}{n} \quad k = 0,1,2,\ldots$$

$$\Longrightarrow q(x_k) < 1 \ \ \forall x_k = \frac{2k\pi}{n}$$

$$\Longrightarrow p(x_k) \ < 0 \ \ \forall x_k = \frac{2k\pi}{n}$$

We can do the same for he minima:

$$T_n(x_k) = -1 \qquad x_k = \frac{2(k+1)\pi}{n} \quad k = 0,1,2,\ldots$$

$$\Longrightarrow q(x_k) > -1 \ \ \forall x_k = \frac{2(k+1)\pi}{n}$$

$$\Longrightarrow p(x_k) > 0 \ \ \forall x_k = \frac{2(k+1)\pi}{n}$$

Therefore p(x) has to change sign at least n time on [-1, 1] which is not possible as p(x) is of degree n-1