In [ ]:
```
# Standard modules needed
import numpy as np
import pandas as pd
import datetime as dt
from types import SimpleNamespace
%load_ext autoreload
%autoreload 2

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn-whitegrid')
```

Approximating a function

In this exercise, you will implement an algorithm to approximate a function $f(x)$ if x is on the interval $[-1,1]$.

A degree N approximation of $f(x)$ takes the general form

$$\hat{f}(x) = \sum_{i=0}^{N} a_i T_i(x)$$

for which you need 3 elements:

1. the functions $T_i(x)$
2. M evaluation nodes $\{z_k\}$.
3. N coefficients $\{a_i\}$

1.
The functions $T_i(x)$ take the form

$$T_i(x) = \cos(i \times \arccos(x))$$

2.
The true function f needs to be evaluated on a set of nodes so that we can use these function evaluations for our approximation. The set of nodes where f is evaluated, $\{z_k\}$, has to be chosen wisely such that the approximation error is minimized.
It turns out to be on the form

$$z_k = -cos(\frac{2k-1}{2M}\pi), \quad k = 1,2,3,\dots,M$$

3.
The N coefficients of the approximation are obtained by what is essentially a least squares regression. They are on the form

$$a_i = \frac{\sum_{k=1}^{M} f(z_k) T_i(z_k)}{\sum_{k=1}^{M} T_i(z_k)^2}, \quad i = 0,1,2,\dots,N$$

Notes: in general one can let $N < M$.
Observe that we are using M evaluations of $f(z)$ to create **each** of the N approximation coefficients. This can be done up front and needs only to be done once even if you need to approximate f on multiple x's. This is why such an approximation is useful in the context of solving an economic model. For instance, a value function may be very computationally intensive, so you'll benefit from only having to to evaluate it M times in order to get, say, $K >> M$ function approximations.

Question 1

Create an approximator $\hat{f}(x)$ at an $x \in [-1,1]$ by implementing the following algorithm:

1. For each $k = 1,\dots,M$: compute $z_k = -cos(\frac{2k-1}{2M}\pi)$
2. For each $k = 1,\dots,M$: compute $y_k = f(z_k)$
3. For each $i = 0,\dots,N$: compute $a_i = \frac{\sum_{k=1}^{M} y_k T_i(z_k)}{\sum_{k=1}^{M} T_i(z_k)^2}$
4. Return $\sum_{i=0}^{N} a_i T_i(x)$

In [ ]:
```
def f_approx(x, f, N, M):
    pass
```

Note: you can use the numpy functions `np.arccos` in T_i and `np.cos` in z_k.

Question 2

Evaluate `f_approx` at $x \in \{-0.5, 0.0, 0.98\}$ and report in each case also the deviation from the true value `f(x)`.

Use the following

In [ ]:
```
f = lambda x: 1/(1+x**2) + x**3 - 0.5*x
N = 5
M = 8
xs = np.array([-0.5, 0.0, 0.98])
```