```
import numpy as np
import pandas as pd

def power_method(A, number_iterations=5, option='ones', return_errors=False):
    count = 0
    lambda_prev = 1
    errors_max = np.zeros(number_iterations)
    lambda_max_new = 0
    lambda_min_new = 0
    #Finding max lambda
    if option == 'ones':
        b = np.ones(len(A))
    else:
        b = np.random.random(len(A))

    while count< number_iterations:
        eigenvec = np.linalg.matmul(A, b)
        lambda_max_new = np.linalg.norm(eigenvec)
        b = eigenvec/lambda_max_new
        errors_max[count] = ((lambda_max_new-lambda_prev)/lambda_max_new)
        lambda_prev = lambda_max_new
        count += 1

    #Finding min lambda
    count = 0
    lambda_prev = 1
    A_1 = np.linalg.inv(A)
    errors_min = np.zeros(number_iterations)
    if option == 'ones':
        b = np.ones(len(A))
    else:
        b = np.random.random(len(A))

    while count< number_iterations:
        eigenvec = np.linalg.matmul(A_1, b)
        lambda_min_new = np.linalg.norm(eigenvec)
        b = eigenvec/lambda_min_new
        errors_min[count] = ((lambda_min_new-lambda_prev)/lambda_min_new)
        lambda_prev = lambda_min_new
        count += 1

    lambda_min_new = 1/lambda_min_new
    if not return_errors:
        return lambda_max_new, lambda_min_new
    return lambda_max_new, lambda_min_new, errors_max, errors_min


def buckling_load_analytical(nodes):
    E = 10*10**9        #Pa
    I = 1.25*10**-5     #m^4
    L = 3               #m
    return  np.pi**2 * E * I /( L**2)

def load_from_p_squared(pp):
    E = 10*10**9        #Pa
    I = 1.25*10**-5     #m^4
    return pp*(E*I)

def create_buckling_matrix(num_interior_nodes):
    A = np.zeros((num_interior_nodes, num_interior_nodes))
    A = A + np.diag(2 * np.ones(num_interior_nodes))
    A = A + np.diag(-1 * np.ones(num_interior_nodes - 1), 1)
    A = A + np.diag(-1 * np.ones(num_interior_nodes - 1), -1)
    return A

def solve_buckling_load(number_nodes, number_iterations):
    num_interior_nodes = number_nodes - 2
    A = create_buckling_matrix(num_interior_nodes)
    L = 3
    lmax, lmin = power_method(A, number_iterations)
    h_squared = (L / (number_nodes-1))**2
    p_squared = lmin / h_squared
    buckling_load = load_from_p_squared(p_squared)
    buck_load_analy = buckling_load_analytical(number_nodes)
    true_relative_error = np.abs((buck_load_analy - buckling_load)) / buck_load_analy

    # Collect data in a dictionary for each iteration
    data = {
```

```python
        'number_nodes': number_nodes,
        'number_iterations': number_iterations,
        'lmax': lmax,
        'lmin': lmin,
        'buckling_load': buckling_load,
        'buck_load_analy': buck_load_analy,
        'true_relative_error': true_relative_error
        }
    return data

#Script body ------------------------------------------------------------------

print(f'The buckling load is {buckling_load_analytical(1)} for one node using the analytic method')

#5 nodes, so 3 interior
part3data=[]
for number_iterations in range(1,6):
    part3data.append(solve_buckling_load(5, number_iterations))

dfp3 = pd.DataFrame(part3data)
print('Variation on number iterations')
print(dfp3)

data = []  # Initialize an empty list to collect data
number_nodes = 3
true_relative_error = 1
while number_nodes < 1000:
    for number_iterations in range(1,6):
        data.append(solve_buckling_load(number_nodes, number_iterations))
        true_relative_error = data[-1]['true_relative_error']
        if true_relative_error < 0.01:
            break
    if true_relative_error < 0.01:
        break
    number_nodes += 1


df = pd.DataFrame(data)
# Display the DataFrame
print('Variation on number of nodes')
print(df)
```