EmpiricalInterpolaton

December 12, 2022

[ ]:
```
# Import requisite packages including the ROM (reduced order model) package␣
↪that I have built

from ReducedModel import *
import numpy as np
from itertools import combinations
from scipy.special import jv as BesselJ
import matplotlib.pyplot as plt
```

The aim here is to utilise the ReduceModel package that I have developed to formulate interpolants that accurately approximate an intput function given an appropriate training space.

[ ]:
```
# Generate some test functions

def monomial(x_range, n=100):
    monomials = []
    for i in range(n):
        monomials.append(x_range**i)

    return monomials

def bessel_functions(x_range, k=100):
    bessels = []
    for i in range(1, k+1):
         bessels.append(np.array(BesselJ(i, x_range)))

    return bessels
```

[ ]:
```
# Generate some x values

def x_values(x_min=-1, x_max=1, n=1000):
    return np.linspace(x_min, x_max, n)
```

[ ]:
```
# Test function that tests whether a set of basis are orthonormal

def test_orthonormal(basis, xx, atol=5e-3, rtol=0):
    indices = range(len(basis))
    xy = combinations(indices, 2)
```

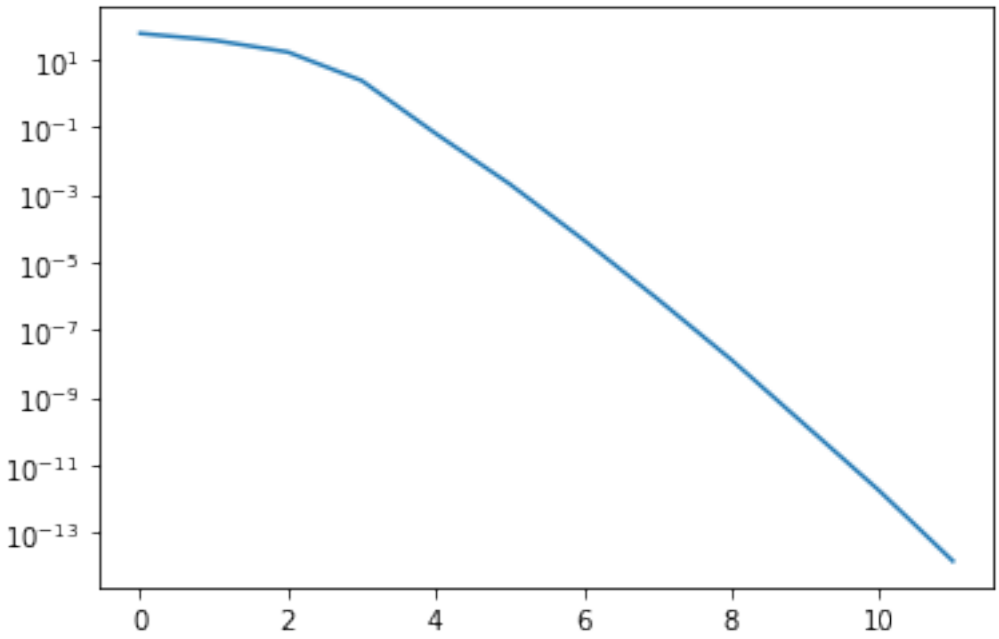
```
    for x, y in xy:
        ip = get_inner(basis[x], basis[y], xx)
        assert np.isclose(ip, 0, atol=atol, rtol=rtol), f"{x} and {y} aren't␣
↪orthonormal, ip: {ip}"
    for i in indices:
        assert np.isclose(get_inner(basis[i], basis[i], xx), 1, atol=atol,␣
↪rtol=rtol)
```

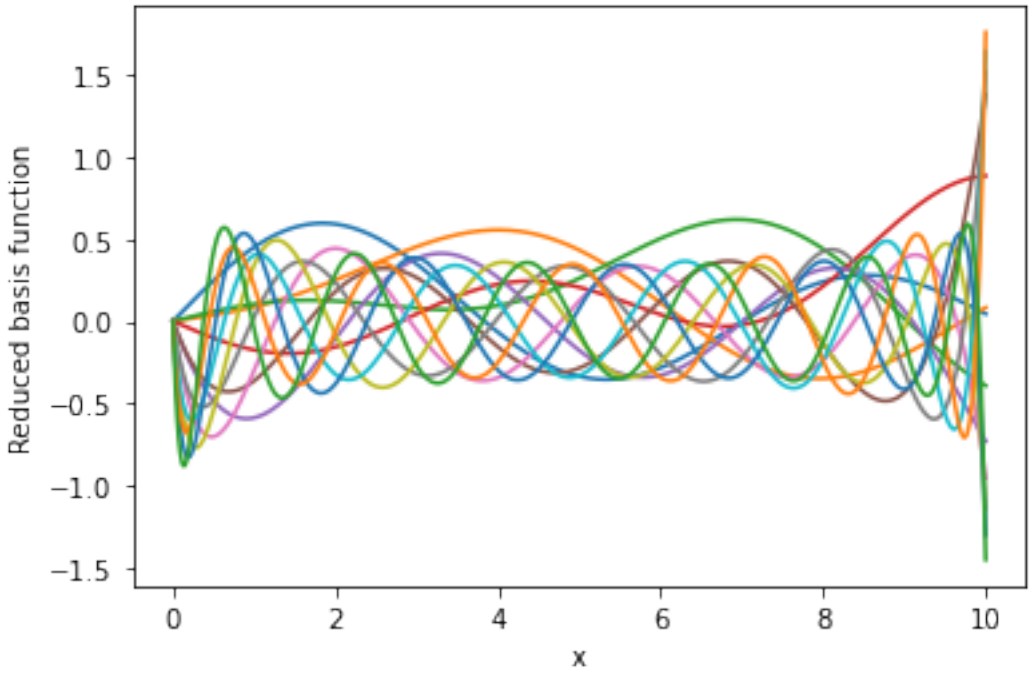
This test function is a sanity check as all the RB functions/vectors should be orthogonal to one another and have a norm of unity.

[ ]:
```
# Running the greedy and empirical interpolant using a test function

x_range = x_values(x_min=0, x_max=10, n=1001)
monomials = monomial(x_range)
bessels = bessel_functions(x_range)

test_functions = bessels

print('Running Greedy algorithm...')
RBs, errors = greedy(test_functions, x_range)
print('Done.')
print('There are', len(RBs), 'reduced basis vectors.')
plt.plot(range(len(errors)), errors)
plt.yscale('log')
plt.show()

test_orthonormal(RBs, x_range)

for RB in RBs:
    plt.plot(x_range, RB)

plt.xlabel('x')
plt.ylabel('Reduced basis function')
plt.show()
```

```
Running Greedy algorithm…
Iteration 2 : 57.76276609822013
Iteration 3 : 36.148314556227476
Iteration 4 : 16.17418985969258
Iteration 5 : 2.3143434873903503
Iteration 6 : 0.0635523291274822
Iteration 7 : 0.0020655525724948853
Iteration 8 : 4.607685711565288e-05
Iteration 9 : 8.37142546301389e-07
Iteration 10 : 1.3362955559118963e-08
Iteration 11 : 1.5664242805815831e-10
Iteration 12 : 1.827975129974337e-12
```

```
Iteration 13 : 1.502929494470351e-14
Done.
There are 13 reduced basis vectors.
```



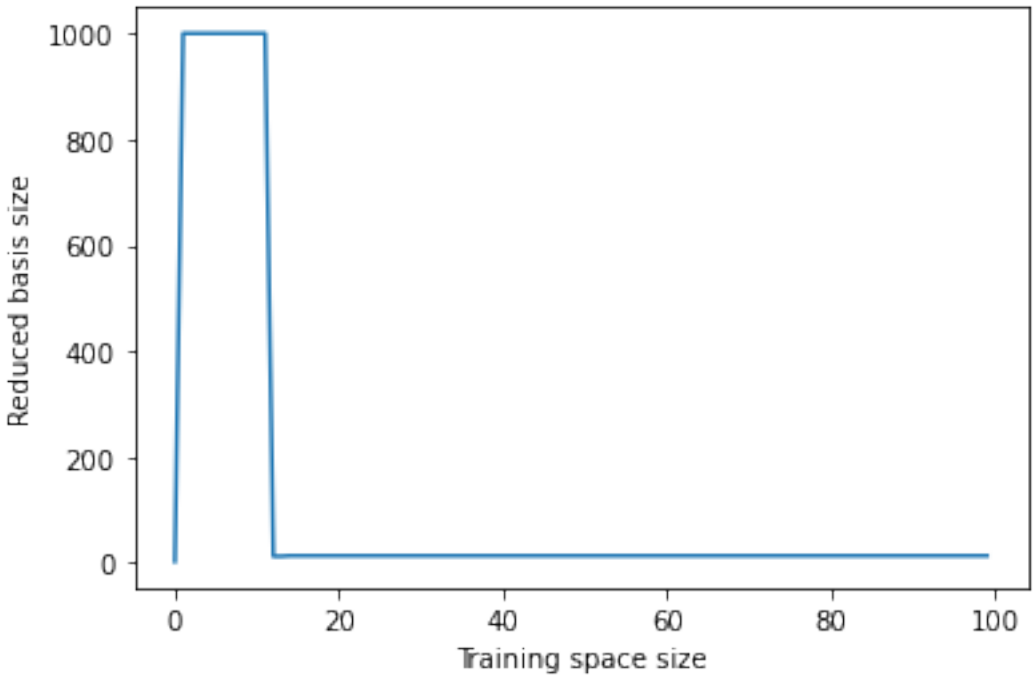


The errors clearly decrese over time which is expected. This suggests that the Gram-Schmidt procedure is becoming more accurate upon each iteration.

The Bessel functions are being successfully orthonormalised to a set of functions that match the Legendre polynomials.

[ ]:
```
# Test convergence of the size of the RB

basis_lengths = []
n = 100
bessels_range = bessel_functions(x_range, k=n)

for i in range(0,n):
    basis, errors = greedy(bessels_range[:i+1], x_range, verbose=False)
    basis_lengths.append(len(basis))
```

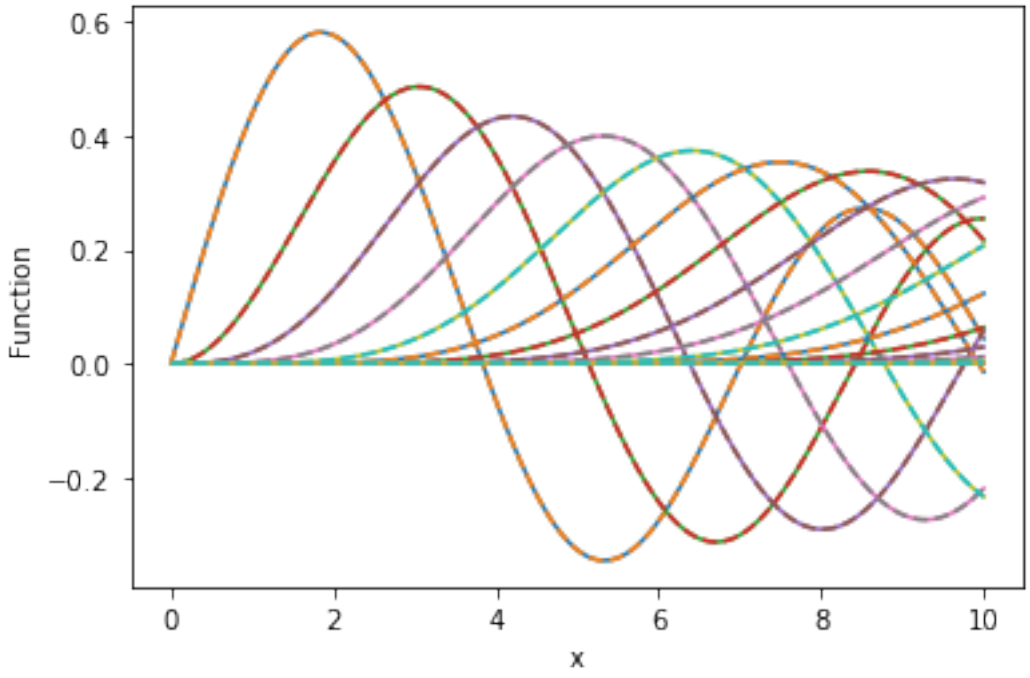
[ ]:
```
plt.plot(range(0,len(basis_lengths)), basis_lengths)
plt.xlabel('Training space size')
plt.ylabel('Reduced basis size')
plt.show()
```



The RB converges on a size of 13 once the training space is sufficiently large. Hence, it is to be decided what erro to use that will best optimise the accuracy of the approximations while keeping the RB as small as possible.

[ ]:
```
print('Running empirical interpolant algorithm...')
EI    = empirical_interpolation(RBs)
B     = EI[0]
nodes = EI[1]

# Plot the original function and the interpolant of the test functions. They␣
 ↪should match almost perfectly

for i in range(len(test_functions)):
    I = get_interpolant(B, nodes, test_functions[i])
    plt.plot(x_range, test_functions[i])
    plt.plot(x_range, I, linestyle = '--')
plt.xlabel('x')
plt.ylabel('Function')
plt.show()
```

```
Running empirical interpolant algorithm…
```



The interpolants match the Bessel functions very well. That concludes this section of investigation. The ReducedModel package that I have developed successfully compute interpolants for a given training space. The next aim is to apply this to a waveform and to be able to recreate waveforms given only the initial parameters.