Week 6

The first two tasks were trivial; the third and final one was much harder. We are constrained to only consider two-dimensional input vectors with orders from zero to four; hence the polynomial basis transform of $\vec{x} = [x_1, x_2]$, parameterised by the order, is the following:

Order	Expression	# of Terms
0	$[1]$	1
1	$[1, x_1, x_2]$	3 (1+2)
2	$[\dots, x_1^2, x_1x_2, x_2^2]$	6 (3+3)
3	$[\dots, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3]$	10 (6+4)
4	$[\dots, x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4]$	15 (10+5)

Note that the number of terms follows the sequence of the triangular numbers (OEIS A000217). Now we can allocate an array for the feature-transformed data:

```
output_feature_count = math.comb(input_feature_count + order, order)

transformed = np.empty(shape=(output_feature_count, 1))
```

Now we must construct the transformed array, noting that the first element is always 1. Let $\mathcal{O}$ be an index set such that $\mathcal{O} = [0, D] \subset \mathbb{Z}$ where D is the target order. Then, for any $d \in \mathcal{O}$, the following product combinations of $\vec{x}$ components should be added to the transformed array in the given order; notice that these are all combinations that induce a polynomial term of order d:

- x_1^d (trivial)
- $x_1^{d-1}x_2$
- $x_1^{d-2}x_2^2$
- $\vdots$
- $x_1^2x_2^{d-2}$
- $x_1x_2^{d-1}$
- x_2^d (trivial)

This is easy to implement:

```
for current_order in range(1, order + 1):
    for exp in range(current_order, -1, -1):
        transformed[transformed_idx][0] = (data[0][0] ** exp) *
```

```
            (data[1][0] ** (current_order - exp))
        transformed_idx += 1
```

We can verify the full implementation with the SciKit-Learn polynomial library:

```
import numpy as np
from math import comb as binomial_coeff
import sklearn.preprocessing as sklp



def transform_polynomial_basis(data, order):
    """ Transforms the provided two-dimensional vector with its polynomial basis of the specified order
    :param data: A two-dimensional vector, i.e. a NumPy vector of shape (2, 1)
    :param order: The integral order to which the polynomial transform basis should be computed
    :return: The feature-transformed array in the (2, 1) input shape
    """
    (input_feature_count, sample_count) = data.shape

    if input_feature_count != 2 or sample_count != 1:
        raise ValueError("The given data is of an invalid dimension")

    output_feature_count = binomial_coeff(input_feature_count + order, order)
    transformed = np.empty(shape=(output_feature_count, 1))
    transformed_idx = 0
    transformed[transformed_idx] = 1
    transformed_idx += 1

    # We can construct the transformed array (known to consist of the precomputed number of output features) by
    # counting up to the target order, adding the relevant polynomial terms at each stage. For each step in the
order,
    # we must generate (in the following order): x_1^d, x_1^{d-1}x_2, ..., x_1x_2^{d-1}, x_2^d.
    for current_order in range(1, order + 1):
        for exp in range(current_order, -1, -1):
            transformed[transformed_idx][0] = (data[0][0] ** exp) * (data[1][0] ** (current_order - exp))
            transformed_idx += 1

    return transformed



TEST_DATA = np.array([[5], [10]])
TEST_ORDER = 3

result = transform_polynomial_basis(TEST_DATA, TEST_ORDER).flatten()
print(result)  # Transformer test
```

```
np.testing.assert_array_equal(sklp.PolynomialFeatures(TEST_ORDER).fit_transform(TEST_DATA.reshape((1, 2)))[0],
                              result)  # Correctness check
print("\nAll OK")
```

This could be generalised further to work with vectors from an N-dimensional input space; see the Multinomial Theorem for determining exponents of feature product combinations.