```
       [ 0.,  0.,  0., -1.,  1.]])
```

In [ ]:
```
D(4)
```

Out[ ]:
```
array([[-1.,  1.,  0.,  0.],
       [ 0., -1.,  1.,  0.],
       [ 0.,  0., -1.,  1.]])
```

In [ ]:
```
Delta = D(4) @ D(5) #Second difference matrix
print(Delta)
```

```
[[ 1. -2.  1.  0.  0.]
 [ 0.  1. -2.  1.  0.]
 [ 0.  0.  1. -2.  1.]]
```

10.3. Matrix power

The kth power of a square matrix A is denoted A^k. In Python, this power is formed using `np.linalg.matrix_power(A,k)`. Let's form the adjacency matrix of the directed graph on VMLS page 186. Then let's find out how many cycles of length 8 there are, starting from each node. (A cycle is a path that starts and stops at the same node.)

In [ ]:
```
A = np.array([[0,1,0,0,1], [1,0,1,0,0], [0,0,1,1,1], [1,0,0,0,0],
↪  [0,0,0,1,0]])
np.linalg.matrix_power(A,2)
```

Out[ ]:
```
array([[1, 0, 1, 1, 0],
       [0, 1, 1, 1, 2],
       [1, 0, 1, 2, 1],
       [0, 1, 0, 0, 1],
       [1, 0, 0, 0, 0]])
```

In [ ]:
```
np.linalg.matrix_power(A,8)
```

Out[ ]:
```
array([[18, 11, 15, 20, 20],
       [25, 14, 21, 28, 26],
       [24, 14, 20, 27, 26],
       [11,  6,  9, 12, 11],
       [ 6,  4,  5,  7,  7]])
```

In [ ]:
```
num_of_cycles = np.diag(np.linalg.matrix_power(A,8))
print(num_of_cycles)
```

```
[18 14 20 12  7]
```

Population dynamics. Let's write the code for figure 10.2 in VMLS, which plots the contribution factor to the total US population in 2020 (ignoring immigration), for each age in 2010. The Python plot is in figure 10.1. We can see that, not surprisingly, $20-25$ years olds have the highest contributing factor, around 1.5. This means that on average, each $20-25$ year old in 2010 will be responsible for around 1.5 people in 2020. This takes into account any children they may have before then, and (as a very small effect) the few of them who will no longer be with us in 2020.

In [ ]:
```
import matplotlib.pyplot as plt
plt.ion()
D = population_data();
b = D['birth_rate'];
d = D['death_rate'];
# Dynamics matrix for populaion dynamics
A = np.vstack([b, np.column_stack([np.diag(1-d[:-1]),
↪  np.zeros((len(d)-1))])])
# Contribution factor to total poulation in 2020
# from each age in 2010
cf = np.ones(100) @ np.linalg.matrix_power(A,10) # Contribution
↪  factor
plt.plot(cf)
plt.xlabel('Age')
plt.ylabel('Factor')
```

10.4. QR factorization

In Python, we can write a `QR_factorization` function to perform the QR factorization of a matrix A using the `gram_schmidt` function on page 41.

Remarks. When we wrote the `gram_schmidt` function, we were checking whether the *rows* of the numpy array (matrix A) are linearly independent. However, in QR factorisation, we should perform the Gram-Schmidt on the *columns* of matrix A. Therefore, A^T should be the input to the `QR_factorization` instead of A.