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.

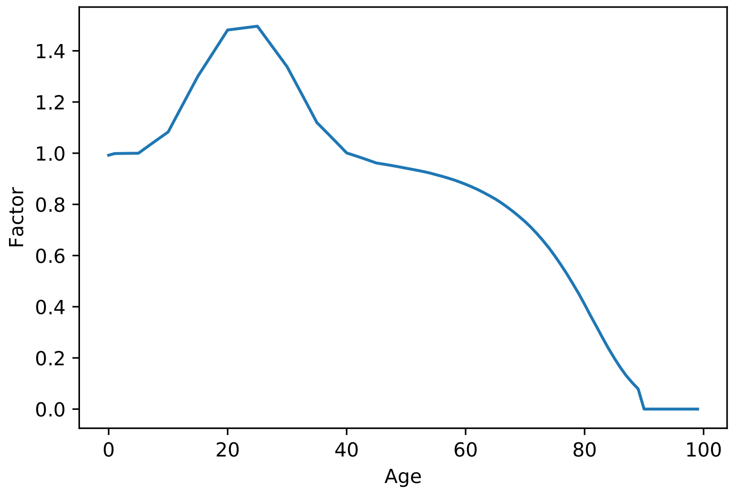

Figure 10.1.: Contribution factor per age in 2010 to the total population in 2020. The value of age $i-1$ is the ith component of the row vector $\mathbf{1}^T\mathbf{A}^10$.

In [ ]:
```
def QR_factorization(A):
    Q_transpose = np.array(gram_schmidt(A.T))
    R = Q_transpose @ A
    Q = Q_transpose.T
    return Q, R
Q, R = QR_factorization(A)
```

Alternatively, we can use the numpy function `np.linalg.qr(A)`.

In [ ]:
```
Q, R = np.linalg.qr(A)
```

Here we would like to highlight a minor difference in the following examples with the VMLS definition. The R factor computed in the `QR_factorization` function may have negative elements on the diagonal, as opposed to only positive elements if we follow the definition used in VMLS and in `np.linalg.qr(A)`. The two definitions are equivalent, because R_{ii} is negative, one can change the sign of the ith row of R and the ith column of Q, to get an equivalent factorization with $R_{ii} > 0$. However, this step is not needed in practice, since negative elements on the diagonal do not pose any problem in applications of the QR factorization.

```
In [ ]: A = np.random.normal(size = (6,4))
        Q, R = np.linalg.qr(A)
        R
```

```
Out[ ]: array([[-2.96963114,  0.77296876,  1.99410713,  0.30491373],
               [ 0.        ,  0.73315602,  1.0283551 , -0.13470329],
               [ 0.        ,  0.        , -3.03153124,  0.60705777],
               [ 0.        ,  0.        ,  0.        , -1.0449406 ]])
```

```
In [ ]: q, r = QR_factorization(A)
        r
```

```
Out[ ]: array([[ 2.96963114e+00, -7.72968759e-01, -1.99410713e+00,
        ↪  -3.04913735e-01],
               [-7.77156117e-16,  7.33156025e-01,  1.02835510e+00,
               ↪  -1.34703286e-01],
               [-8.32667268e-17, -1.66533454e-16,  3.03153124e+00,
               ↪  -6.07057767e-01],
               [-2.22044605e-16, -1.11022302e-16,  8.88178420e-16,
               ↪  1.04494060e+00]])
```

```
In [ ]: np.linalg.norm(Q @ R - A)
```

```
Out[ ]: 1.3188305818502897e-15
```

```
In [ ]: Q.T @ Q
```

```
Out[ ]: array([[ 1.00000000e+00,  9.39209220e-17,  1.90941200e-16,
        ↪  -4.69424568e-17],
               [ 9.39209220e-17,  1.00000000e+00,  2.25326899e-16,
               ↪  2.04228665e-16],
               [ 1.90941200e-16,  2.25326899e-16,  1.00000000e+00,
               ↪  -4.57762562e-16],
               [-4.69424568e-17,  2.04228665e-16, -4.57762562e-16,
               ↪  1.00000000e+00]])
```