which is the net present value (NPV) of the cash flow c.
Let's check this in Python using an interest rate of 5% per period, and the specific cash flow $c = (1, 2, -3)$.

In [ ]:
```
import numpy as np
r = 0.05
e1 = np.array([1,0,0])
l1 = np.array([1, -(1+r), 0])
l2 = np.array([0,1,-(1+r)])
c = np.array([1,2,-3])
# Coefficients of expansion
alpha3 = -c[2]/(1+r)
alpha2 = -c[1]/(1+r) - c[2]/((1+r)**2)
alpha1 = c[0] + c[1]/(1+r) + c[2]/((1+r)**2) #NPV of cash flow
print(alpha1)
```

```
0.18367346938775508
```

In [ ]:
```
print(alpha1*e1 + alpha2*l1 + alpha3*l2)
```

```
[ 1.,  2., -3.]
```

(Later in the course we'll introduce an automated and simple way to find the coefficients in the expansion of a vector in a basis.)

5.3. Orthonormal vectors

Expansion in an orthonormal basis. Let's check that the vectors

$$a_1 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \quad a_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad a_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix},$$

form an orthonormal basis, and check the expansion of $x = (1, 2, 3)$ in this basis,

$$x = (a_1^T x)a_1 + \ldots + (a_n^T x)a_n.$$

In [ ]:
```
a1 = np.array([0,0,-1])
a2 = np.array([1,1,0])/(2)**0.5
a3 = np.array([1,-1,0])/(2)**0.5
print('Norm of a1 :', (sum(a1**2))**0.5)
```

```
print('Norm of a2 :', (sum(a2**2))**0.5)
print('Norm of a3 :', (sum(a3**2))**0.5)
```

```
Norm of a1 : 1.0
Norm of a2 : 0.9999999999999999
Norm of a3 : 0.9999999999999999
```

In [ ]:
```
print(a1 @ a2, a1 @ a3, a2 @ a3)
```

```
0.0, 0.0, 0.0
```

In [ ]:
```
x = np.array([1,2,3])
#Get coefficients of x in orthonormal basis
beta1 = a1 @ x
beta2 = a2 @ x
beta3 = a3 @ x
#Expansion of x in basis
x_expansion = beta1*a1 + beta2*a2 + beta3*a3
print(x_expansion)
```

```
[1. 2. 3.]
```

5.4. Gram-Schmidt algorithm

The following is a Python implementation of the Gram-Schmidt algorithm. It takes as input an array `a` which expands into `[a[0], a[1], ..., a[k]]`, containing k vectors $a_1, \dots, a_k$. If the vectors are linearly independent, it returns an array `q` with orthonormal set of vectors computed by the Gram-Schmidt algorithm. If the vectors are linearly dependent and the Gram-Schmidt algorithm terminates early in the $i+1$th iteration, it returns the array `q[0], ..., q[i-1]` of length i.

In [ ]:
```
import numpy as np
def gram_schmidt(a):
    q = []
    for i in range(len(a)):
        #orthogonalization
        q_tilde = a[i]
        for j in range(len(q)):
            q_tilde = q_tilde -  (q[j] @ a[i])*q[j]
        #Test for dependennce
```