```
       [ 0.06402464, -0.0154395 ,  1.51759022]])
```

11.3. Solving linear equations

Back substitution. Let's first implement back substitution (VMLS Algorithm 11.1) in Python, and check it.

In [ ]:
```
def back_subst(R,b_tilde):
    n = R.shape[0]
    x = np.zeros(n)
    for i in reversed(range(n)):
        x[i] = b_tilde[i]
        for j in range(i+1,n):
            x[i] = x[i] - R[i,j]*x[j]
        x[i] = x[i]/R[i,i]
    return x
R = np.triu(np.random.random((4,4)))
b = np.random.random(4)
x = back_subst(R,b)
np.linalg.norm(R @ x - b)
```

Out[ ]:
```
1.1102230246251565e-16
```

The function `np.triu()` gives the upper triangular part of a matrix, *i.e.*, it zeros out the entries below the diagonal.

Solving system of linear equations. Using the `gram_schmidt`, `QR_factorization` and `back_subst` functions that we have defined in the previous section, we can define our own function to solve a system of linear equations. This function implements the algorithm 12.1 in VMLS.

In [ ]:
```
def solve_via_backsub(A,b):
    Q,R = QR_factorization(A)
    b_tilde = Q.T @ b
    x = back_subst(R,b_tilde)
    return x
```

This requires you to include the other functions in your code as well.

Alternatively, we can use the numpy function `np.linalg.solve(A,b)` to solve a set of linear equations

$$Ax = b.$$

This is faster than `x = np.linalg.inv(A)@ b`, which first computes the inverse of A and then multiplies it with b. However, this function computes the exact solution of a well-determined system.

In [ ]:
```
import time
n = 5000
A = np.random.normal(size = (n,n))
b = np.random.normal(size = n)
start = time.time()
x1 = np.linalg.solve(A,b)
end = time.time()
print(np.linalg.norm(b - A @ x1))
print(end - start)
```

```
4.033331000615254e-09
1.2627429962158203
```

In [ ]:
```
start = time.time()
x2 = np.linalg.inv(A) @ b
end = time.time()
print(np.linalg.norm(b - A @ x2))
print(end - start)
```

```
8.855382050136278e-10
4.3922741413116455
```

11.4. Pseudo-inverse

In Python the pseudo-inverse of a matrix `A` is obtained with `np.linalg.pinv()`. We compute the pseudo-inverse for the example of page 216 of VMLS using the `np.linalg.pinv()` function and via the formula $A^{\dagger} = R^{-1}Q^T$, where $A = QR$ is the QR factorization of A.

In [ ]:
```
A = np.array([[-3,-4],[4,6],[1,1]])
np.linalg.pinv(A)
Q, R = np.linalg.qr(A)
R
```