Virgül Öndeğeri

1. Virgül Öndeğeri

Eigen kullanıcının matrise, vektöre veya diziye kolayca tüm elemanları ekleyebilmesi için virgül öndeğer sözdizimini sunar. Yapılması gereken sol üst köşeden sağ alt köşeye soldan sağa olacak şekilde listelenmesidir. Nesnenin boyutu öncesinde belirtilmelidir. Listedeki eleman sayısı daha az ya da daha fazlaysa Eigen hata verecektir.

```
Matrix3f m;
m << 1, 2, 3,
     4, 5, 6,
     7, 8, 9;
std::cout << m;
```

Üstelik öndeğer listesindeki elemanlar vektör veya matris olabilirler. Virgül öndeğerini kullanmadan önce boyutun belirtilmesi gerektiği unutulmamalıdır.

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 RowVectorXd vec1(3);
 vec1 << 1, 2, 3;
 std::cout << "vec1 = " << vec1 << std::endl;
 RowVectorXd vec2(4);
 vec2 << 1, 4, 9, 16;
 std::cout << "vec2 = " << vec2 << std::endl;
 RowVectorXd joined(7);
 joined << vec1, vec2;
 std::cout << "joined = " << joined << std::endl;

    return 0;
}
```

Çıktı:

```
vec1 = 1 2 3
vec2 =  1  4  9 16
joined =  1  2  3  1  4  9 16
```

Aynı yöntem matris oluşturmak için de kullanılabilir.

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 MatrixXf matA(2, 2);
 matA << 1, 2, 3, 4;
 MatrixXf matB(4, 4);
 matB << matA, matA / 10, matA / 10, matA;
 std::cout << matB << std::endl;

    return 0;
}
```

Çıktı:

```
  1   2 0.1 0.2
  3   4 0.3 0.4
0.1 0.2   1   2
0.3 0.4   3   4
```

Virgül öndeğeri aynı zamanda m.row(i) gibi blok ifadelerini doldurmak için de kullanılır. İlk örneğin daha kompleks yolu aşağıda gösterilmiştir:

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;
```

```
int main()
{
 MatrixXf matA(2, 2);
 matA << 1, 2, 3, 4;
 MatrixXf matB(4, 4);
 matB << matA, matA / 10, matA / 10, matA;
 std::cout << matB << std::endl;

    return 0;
}
```

Çıktı:

```
  1   2 0.1 0.2
  3   4 0.3 0.4
0.1 0.2   1   2
0.3 0.4   3   4
```

1.1. Özel Matrisler ve Diziler

Matrix ve Array sınıflarında Zero() gibi statik fonksiyonlar vardır. Bu fonksiyon tüm elemanlara 0 atamayı sağlar. 3 versiyonu vardır. İlk versiyon argüman almaz ve sadece sabit boyutlu nesnelerce kullanılabilir. Eğer dinamik boyutlu bir nesneye 0 atanmak isteniyorsa, bu durumda boyut belirtilmelidir. Bu yüzden ikinci versiyon bir argüman alır ve tek boyutlu dinamik nesneler için kullanılır, üçüncü versiyon ise iki argüman alır ve iki boyutlu dinamik nesneler için kullanılır.

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 std::cout << "Sabit boyutlu dizi:\n";
 Array33f a1 = Array33f::Zero();
 std::cout << a1 << "\n\n";
 std::cout << "Dinamik tek boyutlu dizi:\n";
 ArrayXf a2 = ArrayXf::Zero(3);
```

```
 std::cout << a2 << "\n\n";
 std::cout << "Dinamik iki boyutlu dizi:\n";
 ArrayXXf a3 = ArrayXXf::Zero(3, 4);
 std::cout << a3 << "\n";

    return 0;
}
```

Çıktı:

```
Sabit boyutlu dizi:
0 0 0
0 0 0
0 0 0

Dinamik tek boyutlu dizi:
0
0
0

Dinamik iki boyutlu dizi:
0 0 0 0
0 0 0 0
0 0 0 0
```

Benzer şekilde, Constant(value) statik fonksiyonu tüm elemanları value değerine eşitler. Eğer nesnenin boyutunun belirlenmesi gerekiyorsa, ek argümanlar value argümanından önce girilir. MatrixXd::Constant(rows, cols, value). Random() fonksiyonu matrisi veya diziyi rastgele elemanlarla doldurur. Birim matris Identity() fonksiyonu ile oluşturulabilir. Bu fonksiyon sadece Matrix için geçerlidir, Array'de kullanılamaz çünkü "birim matris" bir lineer cebir konusudur.

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 const int size = 6;
 MatrixXd mat1(size, size);
```

```
 mat1.topLeftCorner(size / 2, size / 2) = MatrixXd::Zero(size / 2,
 size / 2);
 mat1.topRightCorner(size / 2, size / 2) = MatrixXd::Identity(size / 2,
 size / 2);
 mat1.bottomLeftCorner(size / 2, size / 2) = MatrixXd::Identity(size /
 2, size / 2);
 mat1.bottomRightCorner(size / 2, size / 2) = MatrixXd::Zero(size / 2,
 size / 2);
 std::cout << mat1 << std::endl << std::endl;
 MatrixXd mat2(size, size);
 mat2.topLeftCorner(size / 2, size / 2).setZero();
 mat2.topRightCorner(size / 2, size / 2).setIdentity();
 mat2.bottomLeftCorner(size / 2, size / 2).setIdentity();
 mat2.bottomRightCorner(size / 2, size / 2).setZero();
 std::cout << mat2 << std::endl << std::endl;
 MatrixXd mat3(size, size);
 mat3 << MatrixXd::Zero(size / 2, size / 2), MatrixXd::Identity(size /
 2, size / 2),
  MatrixXd::Identity(size / 2, size / 2), MatrixXd::Zero(size / 2,
 size / 2);
 std::cout << mat3 << std::endl;


    return 0;
}
```

Çıktı:

```
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0

0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0

0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
```

```
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
```