Blok İşlemleri

1. Blok İşlemleri

Eigen’in en genel blok işlemi .block()'tur. İki versiyonu vardır, söz dizimi aşağıdaki gibidir:

- Dinamik boyutlu blok ifadesi oluşumu:

```
matrix.block(i, j, p, q);
```

- Sabit boyutlu blok ifadesi oluşumu:

```
matrix.block<p, q>(i, j);
```

Her iki versiyon da sabit ve dinamik boyutlu matris ve dizilerde kullanılabilir. İki ifade de anlamsal olarak aynıdır. Tek fark sabit boyut blok boyutu küçük ise daha hızlı bir kod sağlayacaktır fakat boyutu derlenme zamanında bilinmelidir.

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

using namespace Eigen;
using namespace std;

int main()
{
 Eigen::MatrixXf m(4, 4);
 m << 1, 2, 3, 4,
   5, 6, 7, 8,
   9, 10, 11, 12,
   13, 14, 15, 16;
 cout << "Ortadaki blok" << endl;
 cout << m.block<2, 2>(1, 1) << endl << endl;
 for (int i = 1; i <= 3; ++i)
 {
  cout << "Blok boyutu: " << i << "x" << i << endl;
  cout << m.block(0, 0, i, i) << endl << endl;
 }
```

```
    return 0;
}
```

Çıktı:

```
Ortadaki blok
 6  7
10 11

Blok boyutu: 1x1
1

Blok boyutu: 2x2
1 2
5 6

Blok boyutu: 3x3
 1  2  3
 5  6  7
 9 10 11
```

Yukarıdaki örnekte .block() fonksiyonu rvalue olarak kullanıldı, yani sadece okuma işlemi yapıldı. Fakat bloklar aynı zamanda lvalue olarak da kullanılabilirler yani bir bloğa atama yapılabilir.

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

using namespace Eigen;
using namespace std;

int main()
{
 Array22f m;
 m << 1, 2,
  3, 4;
 Array44f a = Array44f::Constant(0.6);
 cout << "a dizisi: " << endl << a << endl << endl;
 a.block<2, 2>(1, 1) = m;
 cout << "m, merkez 2x2 bloguna kopyalandiginda a:" << endl << a << endl
 << endl;
```

```
 a.block(0, 0, 2, 3) = a.block(2, 1, 2, 3);
 cout << "Sag-alt 2x3 blogu sol-ust 2x2 bloguna kopyalandgiginda a:" <<
 endl << a << endl << endl;

    return 0;
}
```

1.1. Satırlar ve Sütunlar

Satırlar ve sütunlar özel bloklardır. Eigen fonksiyonları bunları kolayca adreslemyi sağlar: .col() ve .row().

- i^{th} row :

```
matrix.row(i);
```

- j^{th} column* :

```
matrix.col(j);
```

col() ve row()'a gönderilen argüman erişilecek satır veya sütunun indeksidir.

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

using namespace Eigen;
using namespace std;

int main()
{
 Eigen::MatrixXf m(3, 3);
 m << 1, 2, 3,
   4, 5, 6,
   7, 8, 9;
 cout << "m matrisi : " << endl << m << endl;
 cout << "2. satir: " << m.row(1) << endl;
 m.col(2) += 3 * m.col(0);
 cout << "Birinci sutunun uc katini ucuncu sutuna ekleyince, m matrisi:
\n";
 cout << m << endl;
```

```
    return 0;
}
```

Çıktı:

```
m matrisi :
1 2 3
4 5 6
7 8 9
2. satir: 4 5 6
Birinci sutunun uc katini ucuncu sutuna ekleyince, m matrisi:
 1  2  6
 4  5 18
 7  8 30
```

Bu örnek aynı zamanda blok ifadelerinin diğer ifadeler gibi aritmetik işlemlerde kullanılabildiğini gösterir.

1.2. Köşe İlişkili İşlemler

Dinamik boyutlu blok	Sabit boyutlu blok
matrix.topLeftCorner(p,q);	matrix.topLeftCorner<p,q>();
matrix.bottomLeftCorner(p,q);	matrix.bottomLeftCorner<p,q>();
matrix.topRightCorner(p,q);	matrix.topRightCorner<p,q>();
matrix.bottomRightCorner(p,q);	matrix.bottomRightCorner<p,q>();
matrix.topRows(q);	matrix.topRows<q>();
matrix.bottomRows(q);	matrix.bottomRows<q>();
matrix.leftCols(p);	matrix.leftCols<p>();
matrix.rightCols(q);	matrix.rightCols<q>();

Örnek:

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

using namespace Eigen;
```

```
using namespace std;

int main()
{
 Eigen::Matrix4f m;
 m << 1, 2, 3, 4,
  5, 6, 7, 8,
  9, 10, 11, 12,
  13, 14, 15, 16;
 cout << "m.leftCols(2) =" << endl << m.leftCols(2) << endl << endl;
 cout << "m.bottomRows<2>() =" << endl << m.bottomRows<2>() << endl <<
 endl;
 m.topLeftCorner(1, 3) = m.bottomRightCorner(3, 1).transpose();
 cout << "Atama sonrasi, m = " << endl << m << endl;

    return 0;
}
```

Çıktı:

```
m.leftCols(2) =
 1  2
 5  6
 9 10
13 14

m.bottomRows<2>() =
 9 10 11 12
13 14 15 16

Atama sonrasi, m =
 8 12 16  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

1.3. Vektörler İçin Blok İşlemleri

Blok işlemi	Dinamik boyutlu blok	Sabit boyutlu blok
İlk n elemanı içeren blok	vector.head(n);	vector.head<n>();
Son n elemanı içeren blok	vector.tail(n);	vector.tail<n>();

i elemanından başlayıp n elemanı içeren blok	vector.segment(i,n);	vector.segment<n>(i);

Örnek:

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

using namespace Eigen;
using namespace std;

int main()
{
 Eigen::ArrayXf v(6);
 v << 1, 2, 3, 4, 5, 6;
 cout << "v.head(3) =" << endl << v.head(3) << endl << endl;
 cout << "v.tail<3>() = " << endl << v.tail<3>() << endl << endl;
 v.segment(1, 4) *= 2;
 cout << "after 'v.segment(1,4) *= 2', v =" << endl << v << endl;

    return 0;
}
```

Çıktı:

```
v.head(3) =
1
2
3

v.tail<3>() =
4
5
6

after 'v.segment(1,4) *= 2', v =
 1
 4
 6
 8
10
 6
```