Eigen Kütüphanesi Array Sınıfı

1. Eigen Kütüphanesi Array Sınıfı

Array sınıfı, lineer cebir için tasarlanmış Matrix sınıfının aksine, genel amaçlı diziler sağlar. Ayrıca, Array sınıfı, dizideki her katsayıya bir sabit eklenmesi veya katsayı bazında iki dizili çarpma gibi doğrusal bir cebirsel anlamı olmayabilen katsayı bazlı işlemleri gerçekleştirmenin kolay bir yolunu sağlar.

1.1. Dizi Tipleri

Dizi, Matrix ile aynı şablon parametrelerini alan bir sınıf şablonudur. Matrix'te olduğu gibi, ilk üç şablon parametresi zorunludur:

```
Array<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>
```

Son üç şablon parametresi isteğe bağlıdır. Matrix için olduğu gibi bu da burada tekrar açıklanmayacak ve sadece The Matrix sınıfına bakacağız.

Eigen, ayrıca, bazı genel durumlarda, matris typedeflerine benzer, ancak "dizi" sözcüğü hem 1 boyutlu hem de 2 boyutlu diziler için kullanıldığı için hafif farklılıklar gösterecek şekilde typedefler sağlar. ArrayNt formundaki typedef'lerin, bu sayfada açıklanan Matrix typedef'lerde olduğu gibi N ve t boyut ve skaler türe ait 1 boyutlu dizileri temsil ettiğini belirtiriz. 2 boyutlu diziler için, ArrayNNt formunun typedeflerini kullanırız. Bazı örnekler aşağıdaki tabloda gösterilmektedir:

Type	**Typedef**
Array<float,Dynamic,1>	ArrayXf
Array<float,3,1>	Array3f
Array<double,Dynamic,Dynamic>	ArrayXXd
Array<double,3,3>	Array33d

1.2. Bir Array(dizi) içindeki değerlere erişme

Parantez operatörü, matrislerle olduğu gibi bir dizinin katsayılarına yazma ve okuma erişimi sağlamak için aşırı yüklenmiştir. Ayrıca << işleci, dizeleri başlatmak (virgül başlatıcısı aracılığıyla) veya bunları basmak için kullanılabilir.

```
#include <Eigen>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
   ArrayXXf  m(2,2);
   coefficient  m(0,0) = 1.0;
   m(0,1) = 2.0;  m(1,0) = 3.0;
   m(1,1) = m(0,1) + m(1,0);
   cout << m << endl << endl;
   m << 1.0,2.0,
        3.0,4.0;
   cout << m << endl;
}
```

Ekran Çıktısı

```
1 2
3 5

1 2
3 4
```

1.3. Toplama ve Çıkarma İşlemleri

İki dizinin eklenmesi ve çıkartılması matrislerle aynıdır. İşlem, her iki dizi de aynı boyuta sahipse geçerlidir ve toplama veya çıkarma katsayı ile yapılır.

Örnek

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
    ArrayXXf a(3,3);
    ArrayXXf b(3,3);
    a << 1,2,3,
         4,5,6,
         7,8,9;
```

```
    b << 1,2,3,
         1,2,3,
         1,2,3;
    cout << "a + b = " << endl << a + b << endl << endl; cout << "a - 2
 = " << endl << a - 2 << endl;
}
```

Ekran Çıktısı

```
a + b =
 2  4  6
 5  7  9
 8 10 12

a - 2 =
-1  0  1
 2  3  4
 5  6  7
```

1.4. Dizilerde Çarpma işlemi

Her şeyden önce, tabii ki bir diziyi bir skaler ile çarpabilirsiniz, bu matrisler gibi çalışır. Diziler temelde matristen farklı olduğu zaman, ikisini birden alarak çarparsınız. Matrisler, çarpımı matris ürünü olarak ve dizi ise çarpımı katsayı çarpımlı ürün olarak yorumluyor. Böylece, iki dizi aynı boyutlara sahip olsalar ve ancak çarpılabilir.

Örnek

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
   ArrayXXf a(2,2);
   ArrayXXf b(2,2);
   a << 1,2,
        3,4;
   b << 5,6,
        7,8;
   cout << "a * b = " << endl << a * b << endl;
```

```
}
```

Ekran Çıktısı

```
a * b =
 5 12
21 32
```

1.5. Sık kullanılan bazı diğer fonksiyonlar

Array sınıfı, yukarıda açıklanan toplama, çıkarma ve çarpma operatörlerinin yanı sıra diğer katsayı işlemleri tanımlar. Örneğin, .abs () yöntemi her katsayının mutlak değerini alırken, .sqrt () katsayıların karekökünü hesaplar. Aynı boyutta iki diziye sahipseniz, katsayıları iki verilen dizilerin karşılık gelen katsayılarının minimumu olan diziyi oluşturmak için .min (.) Işlevini çağırabilirsiniz. Bu işlemler aşağıdaki örnekte gösterilmektedir.

1.6. Örnek

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{
    ArrayXf a = ArrayXf::Random(5);
    a *= 2;
    cout << "a =" << endl << a << endl;
    cout << "a.abs() =" << endl << a.abs() << endl;
    cout << "a.abs().sqrt() =" << endl << a.abs().sqrt() << endl;
    cout << "a.min(a.abs().sqrt()) =" << endl  <<
 a.min(a.abs().sqrt())<< endl;
}
```

Ekran Çıktısı

```
a =
  1.36
-0.422
  1.13
  1.19
```

```
  1.65
a.abs() =
 1.36
0.422
 1.13
 1.19
 1.65
a.abs().sqrt() =
1.17
0.65
1.06
1.09
1.28
a.min(a.abs().sqrt()) =
  1.17
-0.422
  1.06
  1.09
  1.28
```

1.7. Dizi ve matris arasında dönüştürme işlemi

Matrix(Matris) işlemlerini diziler üzerinde uygulayamaz veya Dizi işlemlerini matrisler üzerinde uygulayamazsınız. Bu nedenle, matris çarpımı gibi doğrusal cebirsel işlemler yapmanız gerekiyorsa, matrisleri kullanmalısınız; Katsayılı işlemler yapmanız gerekiyorsa, dizileri kullanmalısınız. Bununla birlikte, bazen bu kadar basit değil, ancak Matrix ve Array işlemlerini kullanmanız gerekiyor. Bu durumda, bir matrisi bir diziye veya tersine çevirmeniz gerekir. Bu, nesneleri diziler veya matrisler olarak bildirme seçeneğine bakılmaksızın tüm işlemleri kullanmaya olanak tanır.

Matrix ifadeleri, onları katsayıya uygun işlemlerin kolayca uygulanabileceği şekilde, dizi ifadelerine 'dönüştüren' bir .array () yöntemine sahiptir. Tersine, dizi ifadelerinin bir .matrix () yöntemi vardır. Tüm Eigen ifade soyutlamalarıyla olduğu gibi, (derleyicinin optimize etmesine izin verilmiş olması koşuluyla) herhangi bir çalışma zamanı maliyeti yoktur. Hem .array () hem de .matrix () rvalues ve lvalues olarak kullanılabilir.

Matrisleri ve dizileri bir ifade ile karıştırmak, Eigen ile yasaktır. Örneğin, doğrudan bir matris ve dizi ekleyemezsiniz; Bir + operatörünün işlenenleri ya her ikisi de matris olmalıdır ya da her ikisi de dizi olmalıdır. Bununla birlikte, .array () ve .matrix

() ile birinden diğerine kolayca dönüştürülebilir. Bu kuralın istisnası atama işleci: bir dizi değişkene bir matris ifadesi atamak veya bir matris değişkenine bir dizi ifadesi atamak için izin verilir.

Aşağıdaki örnek, .array () yöntemini kullanarak bir Matrix nesnesinde dizi işlemlerinin nasıl kullanılacağını gösterir. Örneğin, deyim result = m.array () * n.array () iki matrisi m ve n alır, onları her ikisine de bir dizi dönüştürür, bunları katsayı ile çarpmak için kullanır ve sonucu matris değişkeni sonucuna atar Yasaldır çünkü Eigen, matris değişkenlerine dizi ifadeleri atamaya izin verir).

Nitekim, bu kullanım durumu o kadar yaygındır ki, Eigen, katsayı çarpımını hesaplamak için matrisler için bir const .cwiseProduct (.) Yöntemi sunar. Bu da örnek programda gösterilmektedir.

1.8. Örnek Kullanım

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{  MatrixXf m(2,2);
   MatrixXf n(2,2);
   MatrixXf result(2,2);
   m << 1,2,
        3,4;
   n << 5,6,
        7,8;
   result = m * n;
   cout << "-- Matrix m*n: --" << endl << result << endl << endl;
   result = m.array() * n.array();  cout << "-- Array m*n: --" << endl
 << result << endl << endl;
   result = m.cwiseProduct(n);
   cout << "-- With cwiseProduct: --" << endl << result << endl << endl;
   result = m.array() + 4;
   cout << "-- Array m + 4: --" << endl << result << endl << endl;
}
```

Ekran Çıktısı

```
-- Matrix m*n: --
```

```
19 22
43 50

-- Array m*n: --
 5 12
21 32

-- With cwiseProduct: --
 5 12
21 32

-- Array m + 4: --
5 6
7 8
```

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
using namespace std;
int main()
{  MatrixXf m(2,2);
   MatrixXf n(2,2);
   MatrixXf result(2,2);
   m << 1,2,
        3,4;
   n << 5,6,
        7,8;
   result = (m.array() + 4).matrix() * m;
   cout << "-- Combination 1: --" << endl << result << endl << endl;
   result = (m.array() * n.array()).matrix() * m;
   cout << "-- Combination 2: --" << endl << result << endl << endl;
}
```

Ekran Çıktısı

```
-- Combination 1: --
23 34
31 46

-- Combination 2: --
 41  58
```

```
117 170
```

2. Eigen Kütüphanesi Map Sınıfı

2.1. Map tipleri ve Map değişkenleri Tanımlama

Map tanımı:

```
Map<Matrix<typename Scalar, int RowsAtCompileTime, int
 ColsAtCompileTime>
```

Bu varsayılan durumda, bir Map'in yalnızca tek bir şablon parametresi gerektirdiğini unutmayın.

Bir Map değişkeni oluşturmak için, diğer iki bilgi parçasına ihtiyacınız vardır: katsayı dizisini tanımlayan belleğin alanına işaretçi ve istenen matris veya vektör şekli. Örneğin, derleme zamanında belirlenen boyutlara sahip bir float matrisi tanımlamak için aşağıdakileri yapabilirsiniz:

```
Map<MatrixXf> mf(pf,rows,columns)
```

Burada pf, bir bellek dizisine işaret eden bir float * pointerdır. Sabit büyüklükte salt okunur bir tam sayı vektörü şu şekilde tanımlanabilir.

```
Map<const Vector4i> mi(pi);
```

Burada pi bir int * pointerdır. Bu durumda, zaten Matrix / Array türünün belirttiği için boyutun yapıcıya geçirilmesi gerekmez.

Map'in bir varsayılan kurucuya sahip olmadığını unutmayın; nesneyi başlatmak için bir işaretçi iletmeniz gerekir.Bununla birlikte, bu gereksinimi etrafında çalışabilirsiniz.

Map, çeşitli veri gösterimlerine uyacak kadar esnektir. İki tane de (isteğe bağlı) şablon parametresi daha vardır:

```
Map<typename MatrixType,    int MapOptions,    typename StrideType>
```

2.2. Kullanımı:

```
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 typedef Matrix<float, 1, Dynamic> MatrixType;
 typedef Map<MatrixType> MapType;
 typedef Map<const MatrixType> MapTypeConst;   // a read-only map
 const int n_dims = 5;

 MatrixType m1(n_dims), m2(n_dims);
 m1.setRandom();
 m2.setRandom();
 float *p = &m2(0);  // get the address storing the data for m2
 MapType m2map(p, m2.size());   // m2map shares data with m2
 MapTypeConst m2mapconst(p, m2.size());  // a read-only accessor for m2
 cout << "m1: " << m1 << endl;
 cout << "m2: " << m2 << endl;
 cout << "Squared euclidean distance: " << (m1 - m2).squaredNorm() <<
 endl;
 cout << "Squared euclidean distance, using map: " <<
  (m1 - m2map).squaredNorm() << endl;
 m2map(3) = 7;   // this will change m2, since they share the same array
 cout << "Updated m2: " << m2 << endl;
 cout << "m2 coefficient 2, constant accessor: " << m2mapconst(2) <<
 endl;
 /* m2mapconst(2) = 5; */   // this yields a compile-time error
 cin.get();
 return 0;
}
```

Tüm Eigen işlevleri, diğer Eigen türleri gibi Map nesnelerini kabul edecek şekilde yazılmıştır. Bununla birlikte, Eigen tiplerini alarak kendi işlevlerinizi yazarken, bu otomatik olarak gerçekleşmez: Bir Eşleme türü, Yoğunluğu eşdeğeri ile aynı değildir.

3. Yeniden Şekillendirme ve Dilimlendirme

Yeniden biçimlendirme işlemi, aynı katsayıları koruyarak bir matrisin boyutlarını değiştirmeyi içerir. Derleme zamanı boyutları için mümkün olmayan giriş matrisini kendisi değiştirmek yerine yaklaşım, Map classını kullanarak saklama alanında farklı bir görünüm oluşturmaktan oluşur. Bir matrisin 1D lineer görünümünü oluşturmak için bir örnek aşağıda verilmektedir:

```
//#include "stdafx.h" //Visual Studio eklentisi
#include <Eigen>
#include <iostream>
using namespace Eigen;

int main()
{
 MatrixXf M1(3, 3);    // Column-major storage
 M1 << 1, 2, 3,
  4, 5, 6,
  7, 8, 9;
 Map<RowVectorXf> v1(M1.data(), M1.size());
 std::cout << "v1:" << std::endl << v1 << std::endl;
 Matrix<float, Dynamic, Dynamic, RowMajor> M2(M1);
 Map<RowVectorXf> v2(M2.data(), M2.size());
 std::cout << "v2:" << std::endl << v2 << std::endl;
    return 0;
}
```

3.1. Ekran Çıktısı

```
v1:
1 4 7 2 5 8 3 6 9
v2:
1 2 3 4 5 6 7 8 9
```

Giriş matrisinin saklama sırasının doğrusal görünümde katsayıların sırasını nasıl değiştirdiğini belirtin. 2x6 matrisi 6 x 2'ye yeniden şekillendiren başka bir örnek:

```
//#include "stdafx.h" //VS eklentisi
#include <iostream>
#include <Eigen>
```

```
using namespace Eigen;

int main()
{

 MatrixXf M1(2, 6);    // Column-major storage
 M1 << 1, 2, 3, 4, 5, 6,
  7, 8, 9, 10, 11, 12;
 Map<MatrixXf> M2(M1.data(), 6, 2);
 std::cout << "M2:" << std::endl << M2 << std::endl;
    return 0;
}
```

Kaynak kod

3.2. Dilimleme(Slicing)

Dilimleme, bir matris içinde eşit aralıklarla yerleştirilmiş bir dizi satır, sütun veya eleman almaktan oluşur.

Örnek

```
RowVectorXf v = RowVectorXf::LinSpaced(20,0,19); cout << "Input:" <<
 endl << v << endl; Map<RowVectorXf,0,InnerStride<2> > v2(v.data(),
 v.size()/2); cout << "Even:" << v2 << endl;
```

4. Eigen üyeleri kullanan yapılar