

A flexible C++ Library for Adaptive Numerical Methods

Matrix- and Vector-Module

Dipl.-Math.oec. Michael C. Lehn, Alexander Stippler

`lehn@mathematik.uni-ulm.de`

University of Ulm

Department of Numerical Analysis



Aims and scopes



Goal: Application in **teaching** and **research**

- Undergraduate and graduate classes
- Master theses
- Adaptive Wavelet Methods
- Computational Fluid Dynamics
- Numerical Finance
- ...



Requirements



Standard requirements for software

- Usability
- Flexibility
- Efficiency
- Maintainability

↔ **Conflicting goals!**



Usability



Some Matlab code...

```
x = [1, 5, 8, 10];
```

```
A(1:2, 2:3) = [1, 5; 8, 10];
```

```
x = A(1:3, 5:10) * b;
```



Usability



Some Matlab code...

```
x = [1, 5, 8, 10];  
A(1:2, 2:3) = [1, 5; 8, 10];  
x = A(1:3, 5:10) * b;
```

...and our notation in C++

```
x = 1, 5, 8, 10;  
A(_(1,2), _(2,3)) = 1, 5, 8, 10;  
x = A(_(1,3), _(5,10)) * b;
```



Flexibility: Requirements



- vector/matrix types
 - dense
 - sparse
 - matrices of matrices
 - ...



Flexibility: Requirements



- vector/matrix types
 - dense
 - sparse
 - matrices of matrices
 - ...
- index range
 - finite
 - infinite (e.g. $c(i)$ where $i \in \mathbb{Z}$).



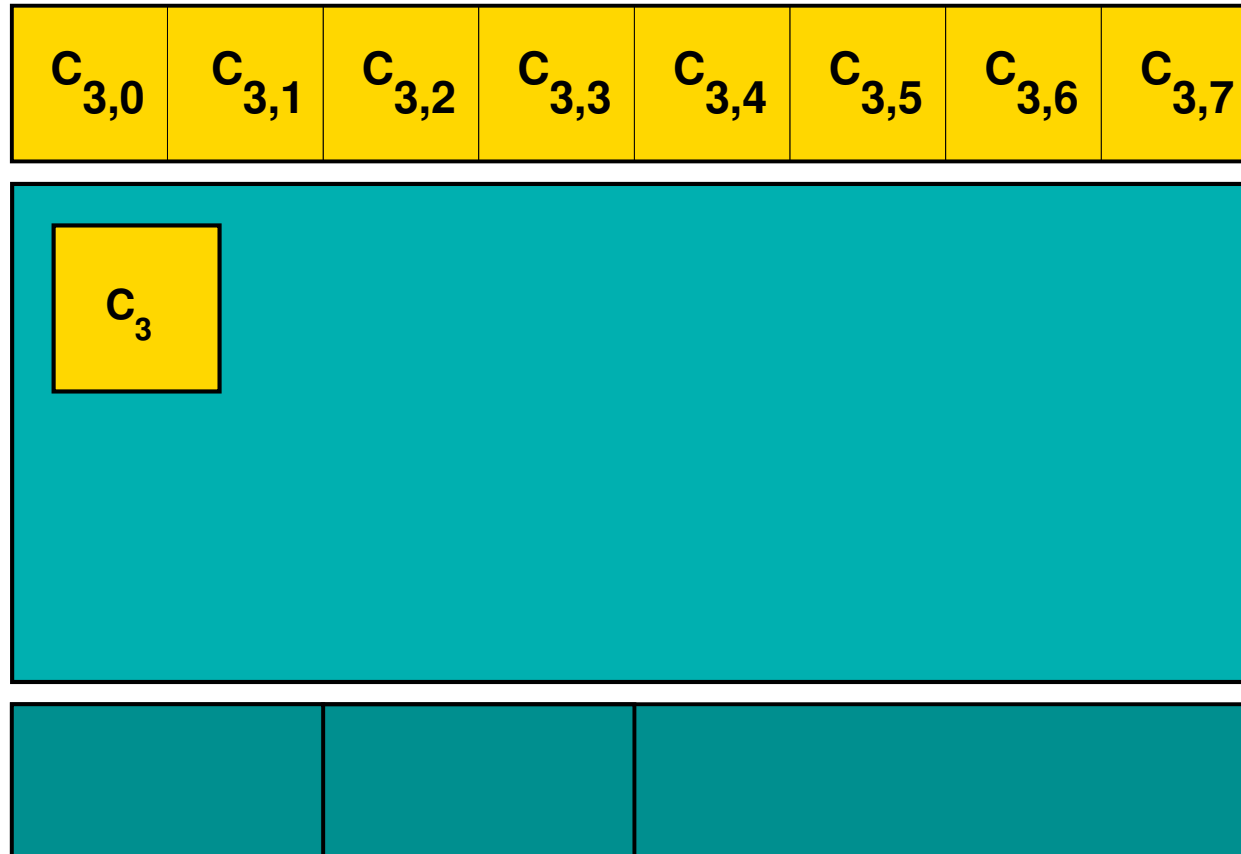
Flexibility: Requirements



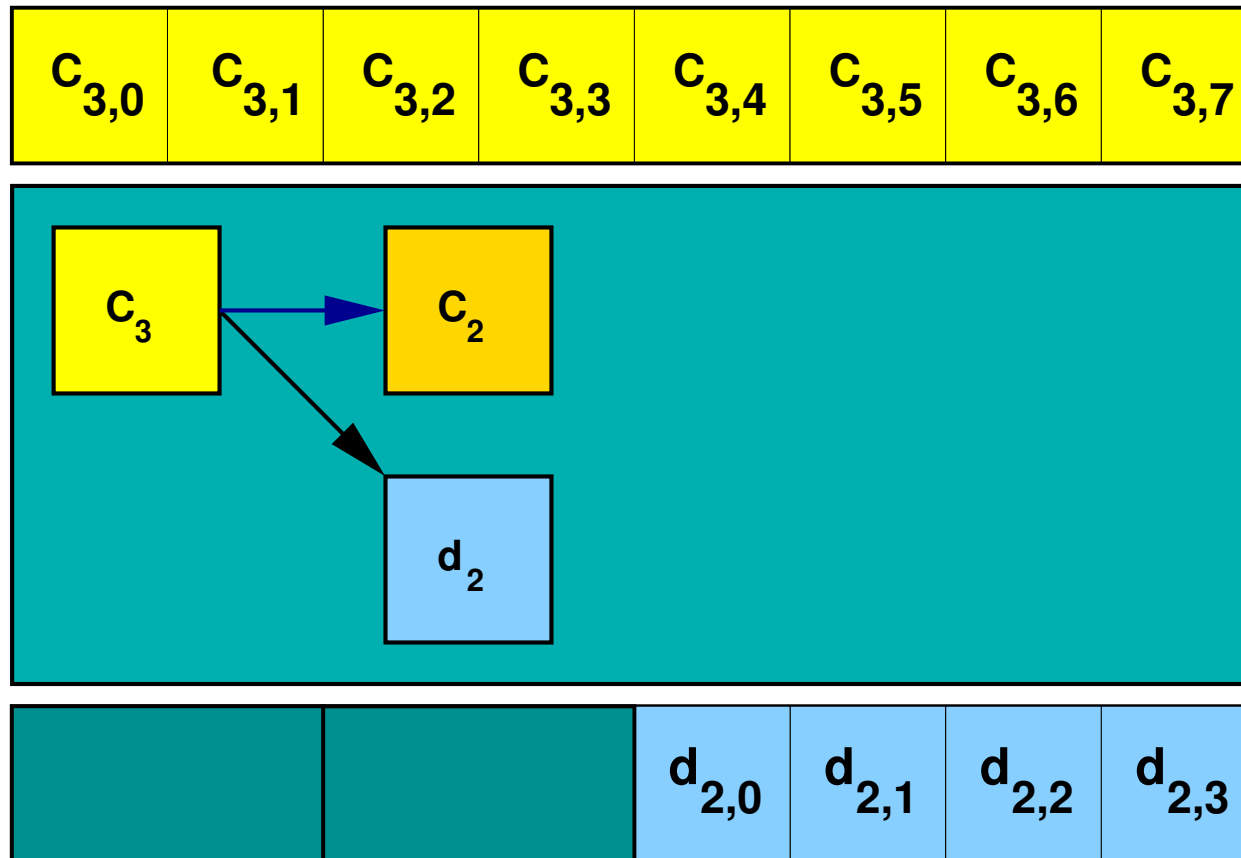
- vector/matrix types
 - dense
 - sparse
 - matrices of matrices
 - ...
- index range
 - finite
 - infinite (e.g. $c(i)$ where $i \in \mathbb{Z}$).
- index type (e.g. fast wavelet transform)



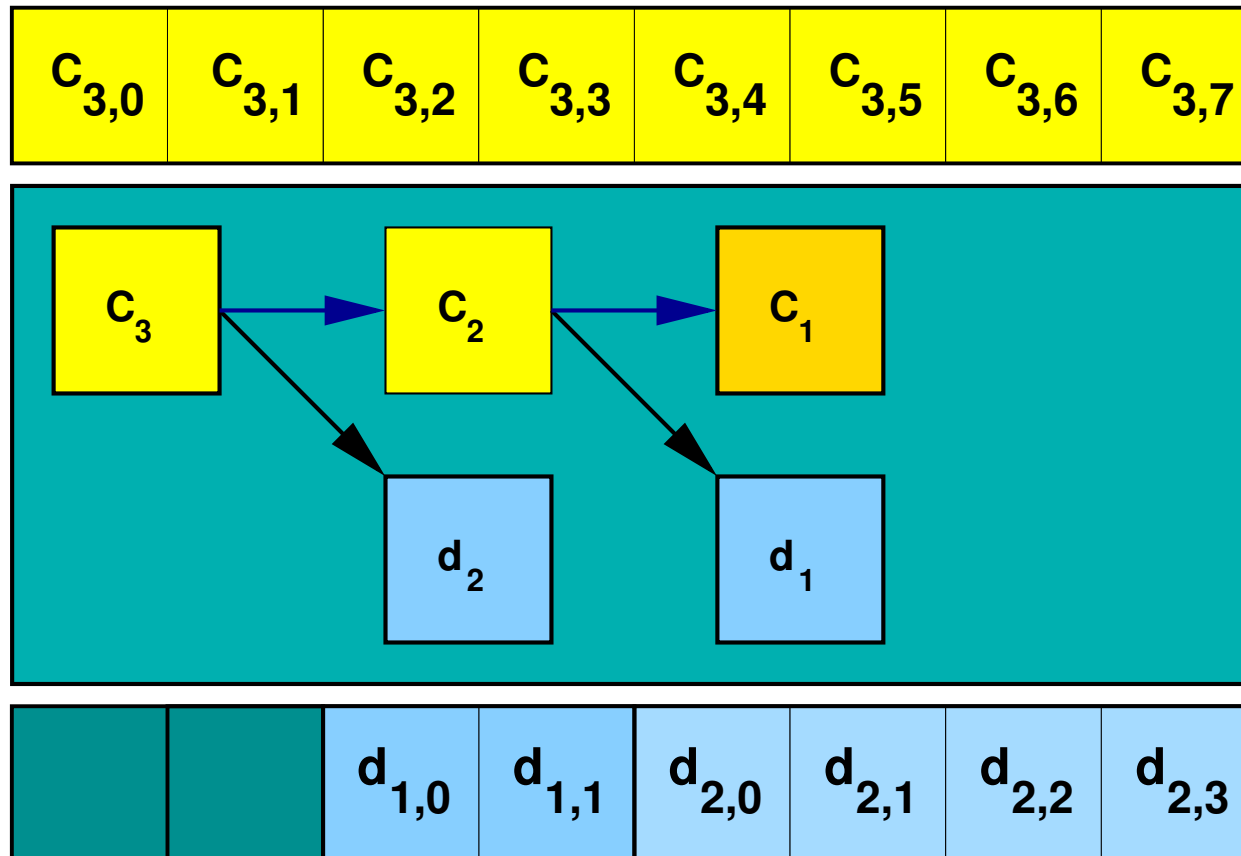
Example: Fast Wavelet Transform



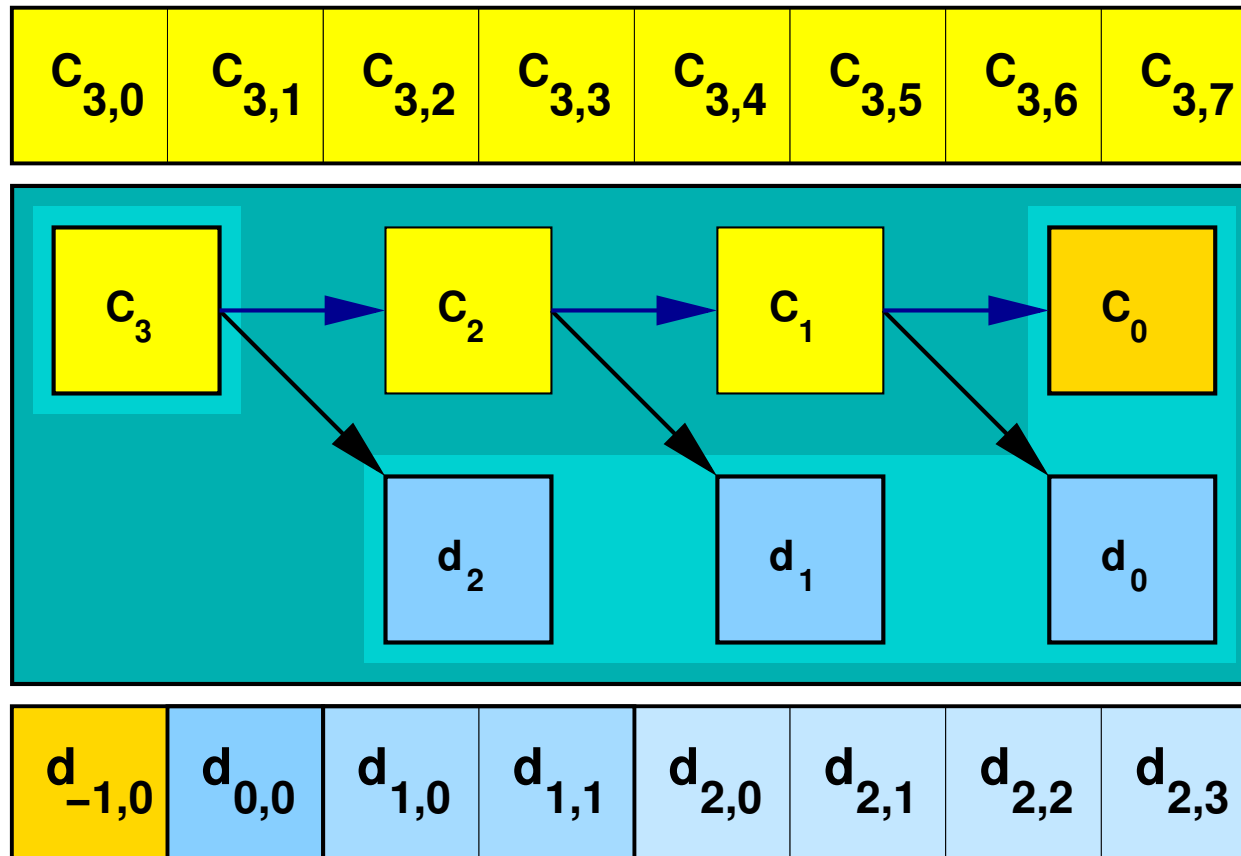
Example: Fast Wavelet Transform



Example: Fast Wavelet Transform



Example: Fast Wavelet Transform



Example: Fast Wavelet Transform

Fast Wavelet Transform:

~> **Input:** Vector indexed by a **scalar** l

~> **Output:** Vector indexed by a **tupel** (j, k)



Example: Fast Wavelet Transform

Fast Wavelet Transform:

↪ **Input:** Vector indexed by a **scalar** l

↪ **Output:** Vector indexed by a **tupel** (j, k)

```
VectorScalarIndex c(8);
```

```
c = 1, 2, 3, 4, 5, 6, 7, 8;
```

```
VectorMLIndex d = fwt(c);
```

```
for (MLIndex i(2,0); i<=MLIndex(3,0); ++i) {  
    std::cout << v(i) << std::endl;  
}
```

Flexibility: Realization



matrix/vector type	data representation
DenseMatrix	array
SparseMatrix	compressed row storage, jagged diagonal storage, ...
SparseVector	hash map, list,...



Flexibility: Realization



- Data representation of matrices:
 - encapsulated
 - exchangeable
- Matrix functionality:
 - Independent of data representation



Flexibility: Realization



- Data representation of matrices:
 - encapsulated
 - exchangeable
- Matrix functionality:
 - Independent of data representation

...

```
typedef CRSData<double, long> SparseData;  
typedef SparseMatrix<SparseData> DoubleSparseMatrix;
```

...



Efficiency



Measure of efficiency:

- handwritten, problem specific C-code
- other libraries

In our domain:

- fast access to matrix parts
- sparse matrices/vectors
- matrices of matrices

In general: open for further optimizations!



Efficiency



- fine-grained structure
 - ~> no virtual functions
 - ~> static polymorphism
- data sharing
 - ~> avoiding temporary objects
 - ~> avoiding copying
- iterators
 - ~> fast sequential access
 - ~> STL algorithms



Efficiency



```
Matrix A(10, 10);  
Vector v;  
...  
v.is( A(_(2,2), _(1,10)) );  
  
double norm = 0;  
for (long i=v.minIndex(); i<=v.maxIndex(); ++i) {  
    norm += v(i) * v(i);  
}
```



Efficiency



```
Matrix A(10, 10);  
Vector v;  
...  
v.is( A(_(2,2), _(1,10)) );  
  
double norm = 0;  
for (Vector::Iterator it=v.begin(); it!=v.end(); ++it) {  
    norm += (*it) * (*it);  
}
```



Efficiency & Usability



- sparse matrices: element access (read/write)

```
std::cerr << v(5) << std::endl; //read  
v(5) = 10;                       //write
```

- separate **read** from **write** operations

```
T &  
operator(const Index &i);
```

```
const T &  
operator(const Index &i) const;
```

Problem: see example above!



Efficiency & Usability



Solution: set and get methods

read	write
<pre>const T & get(Index i) const;</pre>	<pre>void set(Index i, const T &);</pre>

$\rightsquigarrow v(10) += 5 \Rightarrow v.set(10, v.get(10) + 5)$

Problem:

- creates a temporary object
- copies an object
- usability



Efficiency & Usability



Solution: return proxy objects

```
Proxy<T>
```

```
operator(const Index &i);
```

```
const Proxy<T>
```

```
operator(const Index &i) const;
```

Efficiency & Usability:

- **exactly** as efficient as `get` and `set` methods
- direct data access
- invisible for the user: `v(10) += 5;`



Efficiency & Usability



```
template<class T, class Index>
class Proxy
{
    public:
        ...
        operator const T &() const;    //read access

        Proxy<T, Index> &
        operator=(T c);                //write access

    private:
        Vector<T, Index> &v_;
        const Index &index_;
};
```



Maintainability



- coding style
- code localization
- flat interfaces
 - uniform
 - minimal: `norm(v)` is not a vector class method
 - avoid friend functions
- documentation **within** the source code
 - for developer
 - for user





`www.mathematik.uni-ulm.de/numerik`

