Teaching linear algebra to C++

Guy Davidson

C++Russia 30/06/2020

@hatcat01

What to expect...

But first, our goals

Provide linear algebra vocabulary types

But first, our goals

Provide linear algebra vocabulary types

Parameterise orthogonal aspects of implementation

But first, our goals

Provide linear algebra vocabulary types

Parameterise orthogonal aspects of implementation

Defaults for the 90%, customisable for power users

But first, our goals

Provide linear algebra vocabulary types

Parameterise orthogonal aspects of implementation

Defaults for the 90%, customisable for power users

Element access, matrix arithmetic, fundamental operations

But first, our goals

Provide linear algebra vocabulary types

Parameterise orthogonal aspects of implementation

Defaults for the 90%, customisable for power users

Element access, matrix arithmetic, fundamental operations

Mixed precision and mixed representation expressions

What to expect...

0. Representing linear equations

1. I can do better than this
2. Everything you need to know about storage
3. The upsetting story of std::complex
4. Alternative algorithms
5. Assembling the API

Linear algebra 101

“The branch of mathematics concerning linear equations and linear functions, and their representation through matrices and vector spaces”

Linear algebra 101

“The branch of mathematics concerning linear equations and linear functions, and their representation through matrices and vector spaces”

$$a_1x_1 + a_2x_2 + \dots + a_nx_n = b$$

Linear algebra 101

“The branch of mathematics concerning linear equations and linear functions, and their representation through matrices and vector spaces”

$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$

Geometry

Linear algebra 101

"The branch of mathematics concerning linear equations and linear functions, and their representation through matrices and vector spaces"

$a_1x_1 + a_2x_2 + \dots + a_nx_n = b$

Geometry

Linear regression

Linear algebra 101

“The branch of mathematics concerning linear equations and linear functions, and their representation through matrices and vector spaces”

$a_1x_1 + a_2x_2 + … + a_nx_n = b$

Geometry

Linear regression

Simultaneous equations

Linear algebra 101

$(a_1, a_2 \dots a_n)$

Linear algebra 101

$(a_1, a_2 \dots a_n)$

$(a_1, a_2 \dots a_n) + (b_1, b_2 \dots b_n) = (a_1+b_1, a_2+b_2 \dots a_n+b_n)$

Linear algebra 101

$(a_1, a_2 \dots a_n)$

$(a_1, a_2 \dots a_n) + (b_1, b_2 \dots b_n) = (a_1+b_1, a_2+b_2 \dots a_n+b_n)$

$b * (a_1, a_2 \dots a_n) = (ba_1, ba_2 \dots ba_n)$

Linear algebra 101

$(a_1, a_2 … a_n)$

$(a_1, a_2 … a_n) + (b_1, b_2 … b_n) = (a_1+b_1, a_2+b_2 … a_n+b_n)$

$b * (a_1, a_2 … a_n) = (ba_1, ba_2 … ba_n)$

$$(a_1, a_2, a_3) . \begin{matrix}(b_1)\\(b_2)\\(b_3)\end{matrix} = a_1b_1 + a_2b_2 + a_3b_3$$

Linear algebra 101

$(a_{11}, \dots a_{1n})$
$(a_{21}, \dots a_{2n})$
$(a_{31}, \dots a_{3n})$

Linear algebra 101

$$\begin{pmatrix} a_{11}, & \dots & a_{1n} \\ a_{21}, & \dots & a_{2n} \\ a_{31}, & \dots & a_{3n} \end{pmatrix}$$

$$\begin{pmatrix} a_{11}, & \dots & a_{1n} \\ a_{21}, & \dots & a_{2n} \\ a_{31}, & \dots & a_{3n} \end{pmatrix} + \begin{pmatrix} b_{11}, & \dots & b_{1n} \\ b_{21}, & \dots & b_{2n} \\ b_{31}, & \dots & b_{3n} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11}, & \dots & a_{1n}+b_{1n} \\ a_{21}+b_{21}, & \dots & a_{2n}+b_{2n} \\ a_{31}+b_{31}, & \dots & a_{3n}+b_{3n} \end{pmatrix}$$

Linear algebra 101

$$b * \begin{matrix}(a_{11}, \dots a_{1n}) \\ (a_{21}, \dots a_{2n}) \\ (a_{31}, \dots a_{3n})\end{matrix} = \begin{matrix}(ba_{11}, \dots ba_{1n}) \\ (ba_{21}, \dots ba_{2n}) \\ (ba_{31}, \dots ba_{3n})\end{matrix}$$

Linear algebra 101

$$b * \begin{pmatrix} (a_{11}, \dots a_{1n}) \\ (a_{21}, \dots a_{2n}) \\ (a_{31}, \dots a_{3n}) \end{pmatrix} = \begin{pmatrix} (ba_{11}, \dots ba_{1n}) \\ (ba_{21}, \dots ba_{2n}) \\ (ba_{31}, \dots ba_{3n}) \end{pmatrix}$$

$$\begin{pmatrix} (a_{11}, \dots a_{1n}) \\ (a_{21}, \dots a_{2n}) \\ (a_{31}, \dots a_{3n}) \end{pmatrix} * \begin{pmatrix} (b_{11}, b_{12}, b_{13}) \\ (\dots) \\ (b_{n1}, b_{n2}, b_{n3}) \end{pmatrix} = \begin{pmatrix} (a_1.b_1, a_1.b_2, a_1.b_3) \\ (a_2.b_1, a_2.b_2, a_2.b_3) \\ (a_3.b_1, a_3.b_2, a_3.b_3) \end{pmatrix}$$

Linear algebra 101

$$b * \begin{matrix}(a_{11}, \dots a_{1n}) \\ (a_{21}, \dots a_{2n}) \\ (a_{31}, \dots a_{3n})\end{matrix} = \begin{matrix}(ba_{11}, \dots ba_{1n}) \\ (ba_{21}, \dots ba_{2n}) \\ (ba_{31}, \dots ba_{3n})\end{matrix}$$

$$\begin{matrix}(a_{11}, \dots a_{1n}) \\ (a_{21}, \dots a_{2n}) \\ (a_{31}, \dots a_{3n})\end{matrix} * \begin{matrix}(b_{11}, b_{12}, b_{13}) \\ (\quad \dots \quad) \\ (b_{n1}, b_{n2}, b_{n3})\end{matrix} = \begin{matrix}(a_1.b_1, a_1.b_2, a_1.b_3) \\ (a_2.b_1, a_2.b_2, a_2.b_3) \\ (a_3.b_1, a_3.b_2, a_3.b_3)\end{matrix}$$

A*B != B*A

@hatcat01

Linear algebra 101

$$A = \begin{pmatrix} a_{11}, & a_{12}, & \ldots & a_{1n} \\ a_{21}, & a_{22}, & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1}, & a_{n2}, & \ldots & a_{nn} \end{pmatrix}$$

Linear algebra 101

$$A = \begin{pmatrix} a_{11}, & a_{12}, & \ldots & a_{1n} \\ a_{21}, & a_{22}, & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1}, & a_{n2}, & \ldots & a_{nn} \end{pmatrix} \quad I = \begin{pmatrix} 1, & 0, & \ldots & 0 \\ 0, & 1, & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0, & 0, & \ldots & 1 \end{pmatrix}$$

Linear algebra 101

$$A = \begin{pmatrix} a_{11}, & a_{12}, & \dots & a_{1n} \\ a_{21}, & a_{22}, & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1}, & a_{n2}, & \dots & a_{nn} \end{pmatrix} \quad I = \begin{pmatrix} 1, & 0, & \dots & 0 \\ 0, & 1, & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0, & 0, & \dots & 1 \end{pmatrix}$$

Determinant of A = |A|

Linear algebra 101

$$A = \begin{pmatrix} a_{11}, & a_{12}, & \dots & a_{1n} \\ a_{21}, & a_{22}, & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1}, & a_{n2}, & \dots & a_{nn} \end{pmatrix} \quad I = \begin{pmatrix} 1, & 0, & \dots & 0 \\ 0, & 1, & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0, & 0, & \dots & 1 \end{pmatrix}$$

Determinant of A = $|A|$

Inverse of A = A^{-1}

Linear algebra 101

$$A = \begin{pmatrix} a_{11}, & a_{12}, & \dots & a_{1n} \\ a_{21}, & a_{22}, & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1}, & a_{n2}, & \dots & a_{nn} \end{pmatrix} \quad I = \begin{pmatrix} 1, & 0, & \dots & 0 \\ 0, & 1, & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0, & 0, & \dots & 1 \end{pmatrix}$$

Determinant of A = |A|

Inverse of A = A^{-1}

$A^{-1} * A = A * A^{-1} = I$

Linear algebra 101

`operator+(), operator-()`

`operator*(), operator/()`

`operator*() overload`

~~`operator++(), operator--()`~~

~~`operator<(), operator>()`~~

Linear algebra 101

$$ax + by = e$$
$$cx + dy = f$$

Linear algebra 101

```
ax + by = e
cx + dy = f

(a b)*(x) = (e)
(c d) (y)   (f)
```

Linear algebra 101

```
ax + by = e
cx + dy = f

(a b)*(x) = (e)
(c d) (y)   (f)

A*(x) = (e)
  (y)   (f)
```

Linear algebra 101

```
ax + by = e
cx + dy = f

(a b)*(x) = (e)
(c d) (y)   (f)

A*(x) = (e)
  (y)   (f)
```

$$\begin{pmatrix} x \\ y \end{pmatrix} = A^{-1} * \begin{pmatrix} e \\ f \end{pmatrix}$$

Linear algebra 101

```
2x + 3y =  8
 x - 2y = -3
```

Linear algebra 101

```
2x + 3y =  8      A = (2  3)
 x - 2y = -3          (1 -2)
```

Linear algebra 101

```
2x + 3y =  8      A = (2  3)
 x - 2y = -3          (1 -2)
```

$|A|^{-1}$ * classical adjoint(A)

Linear algebra 101

```
2x + 3y =  8
 x - 2y = -3
```

```
A = (2  3)
    (1 -2)
```

$|A|^{-1}$ * classical adjoint(A)

```
|A| = (2 * -2) - (1 * 3)
    = -7
```

Linear algebra 101

```
2x + 3y =  8
 x - 2y = -3
```

```
A = (2  3)
    (1 -2)
```

$|A|^{-1}$ * classical adjoint(A)

```
|A| = (2 * -2) - (1 * 3)
    = -7
```

```
classical adjoint A = (-2 -3)
                      (-1  2)
```

@hatcat01

Linear algebra 101

```
2x + 3y =  8
 x - 2y = -3
```

$$A = \begin{pmatrix} 2 & 3 \\ 1 & -2 \end{pmatrix}$$

$$|A| = -7$$

$$\text{classical adjoint } A = \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix}$$

$$A^{-1} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix}$$

Linear algebra 101

```
2x + 3y =  8        A = (2  3)
 x - 2y = -3            (1 -2)

                    A^-1 = -7^-1 * (-2 -3)
                                   (-1  2)
```

$$A^{-1} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix}$$

Linear algebra 101

```
2x + 3y =  8        A = (2  3)
 x - 2y = -3            (1 -2)
```

$$A^{-1} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix} * \begin{pmatrix} 8 \\ 3 \end{pmatrix}$$

Linear algebra 101

$$2x + 3y = 8$$
$$x - 2y = -3$$

$$A = \begin{pmatrix} 2 & 3 \\ 1 & -2 \end{pmatrix}$$

$$A^{-1} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix} * \begin{pmatrix} 8 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{matrix} ((-2 * 8) + (-3 * 3)) / -7 \\ ((-1 * 8) + (\ 2 * 3)) / -7 \end{matrix}$$

Linear algebra 101

$$\begin{aligned} 2x + 3y &= 8 \\ x - 2y &= -3 \end{aligned}$$

$$A = \begin{pmatrix} 2 & 3 \\ 1 & -2 \end{pmatrix}$$

$$A^{-1} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = -7^{-1} * \begin{pmatrix} -2 & -3 \\ -1 & 2 \end{pmatrix} * \begin{pmatrix} 8 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

What to expect...

0. Representing linear equations
1. **I can do better than this**
2. Everything you need to know about storage
3. The upsetting story of std::complex
4. Alternative algorithms
5. Assembling the API

Prior art

Fixed point, 80286 (no maths coprocessor)

Prior art

Fixed point, 80286 (no maths coprocessor)

Floating point, 80486

Prior art

Fixed point, 80286 (no maths coprocessor)

Floating point, 80486

SSE2, Pentium IV

Prior art

Fixed point, 80286 (no maths coprocessor)

Floating point, 80486

SSE2, Pentium IV

AVX, 2011 (Sandy Bridge?)

Prior art

Optimisations available through specialisation

Prior art

Optimisations available through specialisation

Matrix size

Prior art

Optimisations available through specialisation

Matrix size

float

Prior art

Optimisations available through specialisation

Matrix size

float

SIMD instruction set

Prior art

Optimisations available through specialisation

Matrix size

float

SIMD instruction set

Cache line size

Prior art

Optimisations available through specialisation

Matrix size

float

SIMD instruction set

Cache line size

Dense

Prior art

BLAS (Basic Linear Algebra Subprograms)

Prior art

BLAS (Basic Linear Algebra Subprograms)

BLAS++

Prior art

BLAS (Basic Linear Algebra Subprograms)

BLAS++

```
void blas::axpy(int64_t n, float alpha,
                float const* x, int64_t incx,
                float* y, int64_t incy);
```

Prior art

BLAS (Basic Linear Algebra Subprograms)

BLAS++

```
void blas::axpy(int64_t n, float alpha,
                float const* x, int64_t incx,
                float* y, int64_t incy);
```

Boost.uBLAS

Prior art

asum	vector 1 norm (sum)
axpy	add vectors
copy	copy vector
dot	dot product
dotu	dot product, unconjugated
iamax	max element
nrm2	vector 2 norm
rot	apply Givens plane rotation
rotg	generate Givens plane rotation
rotm	apply modified Givens plane rotation
rotmg	generate modified Givens plane rotation
scal	scale vector
swap	swap vectors

Prior art

asum	**gemv**	general matrix-vector multiply
axpy	**ger**	general matrix rank 1 update
copy	**hemv**	hermitian matrix-vector multiply
dot	**her**	hermitian rank 1 update
dotu	**her2**	hermitian rank 2 update
iamax	**symv**	symmetric matrix-vector multiply
nrm2	**syr**	symmetric rank 1 update
rot	**syr2**	symmetric rank 2 update
rotg	**trmv**	triangular matrix-vector multiply
rotm	**trsv**	triangular matrix-vector solve
rotmg		
scal		
swap		

Prior art

asum	gemv	**gemm**	general matrix multiply: C = AB + C
axpy	ger	**hemm**	hermitian matrix multiply
copy	hemv	**herk**	hermitian rank k update
dot	her	**her2k**	hermitian rank 2k update
dotu	her2	**symm**	symmetric matrix multiply
iamax	symv	**syrk**	symmetric rank k update
nrm2	syr	**syr2k**	symmetric rank 2k update
rot	syr2	**trmm**	triangular matrix multiply
rotg	trmv	**trsm**	triangular solve matrix
rotm	trsv		
rotmg			
scal			
swap			

Prior Art

asum	gemv	**gemm**	general matrix multiply: C = AB + C
axpy	ger	**hemm**	hermitian matrix multiply
copy	hemv	**herk**	hermitian rank k update
dot	her	**her2k**	hermitian rank 2k update
dotu	her2	**symm**	symmetric matrix multiply
iamax	symv	**syrk**	symmetric rank k update
nrm2	syr	**syr2k**	symmetric rank 2k update
rot	syr2	**trmm**	triangular matrix multiply
rotg	trmv	**trsm**	triangular solve matrix
rotm	trsv		
rotmg			
scal			
swap			

https://wg21.link/P1673

P1673R2: A free function linear algebra interface based on the BLAS

Prior art

Eigen

Prior art

Eigen

Matrix and vector class templates

Prior art

Eigen

Matrix and vector class templates

Dynamic or static sizes

Prior art

Eigen

Matrix and vector class templates

Dynamic or static sizes

Span option via Eigen::Map

Quiz time

How many member functions does string have which are NOT special functions?

Prior art

Eigen

Matrix and vector class templates

Dynamic or static sizes

Span option via Eigen::Map

Member function API

Prior art

```
#include <iostream>
#include <Eigen/Dense>
using namespace Eigen;
using namespace std;

int main() {
  MatrixXd m = MatrixXd::Random(3,3);
  m = (m + MatrixXd::Constant(3,3,1.2)) * 50;
  cout << "m =" << endl << m << endl;
  VectorXd v(3);
  v << 1, 2, 3;
  cout << "m * v =" << endl << m * v << endl;
}
```

Prior art

Dlib

Prior art

Dlib

Expression templates

Prior art

Dlib

Expression templates

https://en.wikipedia.org/wiki/Expression_templates

Prior art

```
class row_vector {
  public:
    row_vector(size_t n) : elems(n)    {}
    double &operator[](size_t i)       { return elems[i]; }
    double operator[](size_t i) const  { return elems[i]; }
    size_t size()               const  { return elems.size(); }
  private:
    std::vector<float> elems;
};
```

Prior Art

```
row_vector operator+(row_vector const &u, row_vector const &v) {
  row_vector sum(u.size());
  for (size_t i = 0; i < u.size(); i++)
    sum[i] = u[i] + v[i];
  return sum;
}

auto a = row_vector(4);
auto b = row_vector(4);
auto c = row_vector(4);
...
auto d = a + b + c;
```

Prior art

Delayed evaluation

Prior art

Delayed evaluation

```
row_vector_sum operator+(...
```

Prior art

Delayed evaluation

`row_vector_sum operator+(...`

Expression trees

Prior art

Delayed evaluation

`row_vector_sum operator+(...`

Expression trees

Compile time evaluation

Prior art

```
template <typename E>
class vector_expression {
  public:
    double operator[](size_t i) const {
      return static_cast<E const&>(*this)[i];
    }
    size_t size() const {
      return static_cast<E const&>(*this).size();
    }
};
```

Prior art

```
row_vector(std::initializer_list<float>init) {
  for (auto i:init)
    elems.push_back(i);
}

template <typename E>
row_vector(vector_expression<E> const& exp) : elems(exp.size()) {
  for (size_t i = 0; i != exp.size(); ++i)
    elems[i] = exp[i];
}
```

Prior art

```
template <typename E1, typename E2>
class vector_sum : public vector_expression<vector_sum<E1, E2>> {
  public:
    vector_sum(E1 const& u_in, E2 const& v_in) : u(u_in), v(v_in) {}
    double operator[](size_t i) const { return u[i] + v[i]; }
    size_t size()               const { return v.size(); }
  private:
    E1 const& u;
    E2 const& v;
};
```

Prior art

```
template <typename E1, typename E2>
auto operator+(E1 const& u, E2 const& v)
{
  return vector_sum<E1, E2>(u, v);
}
```

Prior art

```
template <typename E1, typename E2>
auto operator+(E1 const& u, E2 const& v)
{
  return vector_sum<E1, E2>(u, v);
}

vector_sum<vector_sum<row_vector, row_vector>, row_vector> d = a + b + c;
```

Prior art

```
template <typename E1, typename E2>
auto operator+(E1 const& u, E2 const& v)
{
  return vector_sum<E1, E2>(u, v);
}

vector_sum<vector_sum<row_vector, row_vector>, row_vector> d = a + b + c;

elems[i] = exp[i];
```

Prior art

```
template <typename E1, typename E2>
auto operator+(E1 const& u, E2 const& v)
{
  return vector_sum<E1, E2>(u, v);
}

vector_sum<vector_sum<row_vector, row_vector>, row_vector> d = a + b + c;

elems[i] = exp[i];

elems[i] = a.elems[i] + b.elems[i] + c.elems[i];
```

What to expect...

0. Representing linear equations
1. I can do better than this
2. **Everything you need to know about storage**
3. The upsetting story of std::complex
4. Alternative algorithms
5. Assembling the API

Storage

Fixed size

Storage

Fixed size

Sparse

Storage

Fixed size

Sparse

Dynamic size

Storage

Fixed size

Sparse

Dynamic size

View

Storage

Cache lines

Storage

Cache lines

SIMD

Storage

Cache lines

SIMD

Paramaterise

Storage

```
template <class T, ptrdiff_t Rows, ptrdiff_t Cols, class Alloc, class Layout>
class matrix_storage_engine
{
  public:
    using reference        = element_type&;
    using const_reference  = element_type const&;
    using index_type       = ptrdiff_t;
    using index_tuple_type = tuple<index_type, index_type>;
    using span_type        = ...;  //- implementation-defined
    using const_span_type  = ...;  //- implementation-defined
```

Storage

```
constexpr matrix_storage_engine();
constexpr matrix_storage_engine(index_type rows, index_type cols);
template<class U>
constexpr matrix_storage_engine(
    std::initializer_list<<std::initializer_list<U>>);

constexpr index_type       columns()         const noexcept;
constexpr index_type       rows()            const noexcept;
constexpr index_tuple_type size()            const noexcept;
constexpr index_type       column_capacity() const noexcept;
constexpr index_type       row_capacity()    const noexcept;
constexpr index_tuple_type capacity()        const noexcept;
```

Storage

```
void    resize_columns(index_type cols);
void    reserve_columns(index_type colcap);
void    reshape_columns(index_type cols, index_type colcap);
void    resize_rows(index_type rows);
void    reserve_rows(index_type rowcap);
void    reshape_rows(index_type rows, index_type rowcap);
void    resize(index_type rows, index_type cols);
void    reserve(index_type rowcap, index_type colcap);
void    reshape(index_type rows, index_type cols,
                index_type rowcap, index_type colcap);
```

Storage

```
    constexpr reference         operator ()(index_type i, index_type j);
    constexpr const_reference   operator ()(index_type i, index_type j) const;
    constexpr span_type         span();
    constexpr const_span_type   span() const;
    constexpr void      swap(matrix_storage_engine& rhs) noexcept;
    constexpr void      swap_columns(index_type c1, index_type c2) noexcept;
    constexpr void      swap_rows(index_type r1, index_type r2) noexcept;
};
```

https://wg21.link/P0009
P0009R10: mdspan, a polymorphic multidimensional array reference

What to expect...

0. Representing linear equations
1. I can do better than this
2. Everything you need to know about storage
3. **The upsetting story of std::complex**
4. Alternative algorithms
5. Assembling the API

Quiz time

```
auto a = 7 * 5 / 3;
```

Quiz time

```
auto a = 7 * 5 / 3;          // int a = 11
```

Quiz time

```
auto a = 7 * 5 / 3;          // int a = 11

auto a = 7 * 5 / 3l;
```

Quiz time

```
auto a = 7 * 5 / 3;          // int a = 11

auto a = 7 * 5 / 3l;         // long a = 11l
```

Quiz time

```
auto a = 7 * 5 / 3;          // int a = 11

auto a = 7 * 5 / 3l;         // long a = 11l

auto a = 7 * 5 / -3ul;
```

Quiz time

```
auto a = 7 * 5 / 3;          // int a = 11
auto a = 7 * 5 / 3l;         // long a = 11l
auto a = 7 * 5 / -3ul;       // unsigned long a = 0ul
```

Quiz time

```
auto a = 7 * 5 / 3;          // int a = 11

auto a = 7 * 5 / 3l;         // long a = 11l

auto a = 7 * 5 / -3ul;       // unsigned long a = 0ul

long a = 7 * 5 / -3ul;
```

Quiz time

```
auto a = 7 * 5 / 3;         // int a = 11
auto a = 7 * 5 / 3l;        // long a = 11l
auto a = 7 * 5 / -3ul;      // unsigned long a = 0ul
long a = 7 * 5 / -3ul;      // long a = 0l
```

Quiz time

```
auto a = 7 * 5 / 3.;
```

Quiz time

```
auto a = 7 * 5 / 3.;          // double a = 11.666666666666666
```

Quiz time

```
auto a = 7 * 5 / 3.;          // double a = 11.666666666666666

auto a = 7. * 5.f / 3;
```

Quiz time

```
auto a = 7 * 5 / 3.;        // double a = 11.666666666666666

auto a = 7. * 5.f / 3;      // double a = 11.666666666666666
```

Quiz time

```
auto a = 7 * 5 / 3.;         // double a = 11.6666666666666

auto a = 7. * 5.f / 3;       // double a = 11.6666666666666

auto a = 7.f * 5.f / 3;
```

Quiz time

```
auto a = 7 * 5 / 3.;        // double a = 11.666666666666666
auto a = 7. * 5.f / 3;      // double a = 11.666666666666666
auto a = 7.f * 5.f / 3;     // float a = 11.666667f
```

Quiz time

```
auto a = 7 * 5 / 3.;        // double a = 11.666666666666666

auto a = 7. * 5.f / 3;      // double a = 11.666666666666666

auto a = 7.f * 5.f / 3;     // float a = 11.666667f

auto a = 7.f * 5.f / -3l;
```

Quiz time

```
auto a = 7 * 5 / 3.;        // double a = 11.666666666666666
auto a = 7. * 5.f / 3;      // double a = 11.666666666666666
auto a = 7.f * 5.f / 3;     // float a = 11.666667f
auto a = 7.f * 5.f / -3l;   // float a = -11.666667f
```

Quiz time

```
auto a = 7 * 5 / 3.;         // double a = 11.66666666666666
auto a = 7. * 5.f / 3;       // double a = 11.66666666666666
auto a = 7.f * 5.f / 3;      // float a = 11.666667f
auto a = 7.f * 5.f / -3l;    // float a = -11.666667f
auto a = 7.f * 5.f / -3ul;
```

Quiz time

```
auto a = 7 * 5 / 3.;        // double a = 11.666666666666666

auto a = 7. * 5.f / 3;      // double a = 11.666666666666666

auto a = 7.f * 5.f / 3;     // float a = 11.666667f

auto a = 7.f * 5.f / -3l;   // float a = -11.666667f

auto a = 7.f * 5.f / -3ul;  // float a =
                            // 0.0000000000000000018973538f
```

Promotion and conversion

Integral promotion

Promotion and conversion

Integral promotion

Floating point promotion

Promotion and conversion

Integral promotion

Floating point promotion

Integral conversions

Promotion and conversion

Integral promotion

Floating point promotion

Integral conversions

Floating-point conversions

Promotion and conversion

Integral promotion

Floating point promotion

Integral conversions

Floating-point conversions

Floating-integral conversions

Promotion and conversion

Integral promotion

Floating point promotion

Integral conversions

Floating-point conversions

Floating-integral conversions

(Search for integral promotion at cppreference.com)

Promotion and conversion

Promotion:

float->double, int->long, widening representation

Promotion and conversion

Promotion:

float->double, int->long, widening representation

Conversion:

integral->floating point, changing representation

Promotion and conversion

Promotion:

float->double, int->long, widening representation

Conversion:

integral->floating point, changing representation

ftol()

Promotion and conversion

Promotion:

float->double, int->long, widening representation

Conversion:

integral->floating point, changing representation

ftol()

```
int a = b * 3.5;
```

Promotion and conversion

```
(3 5 5)   (1.0 3.3 6.8)   (4.0 8.3 11.8)
(4 4 3) + (3.0 2.5 7.3) = (7.0 6.5 10.3)
(1 0 1)   (2.1 4.8 4.4)   (3.1 4.8  5.4)
```

Promotion and conversion

```
(3 5 5)   (1.0 3.3 6.8)   (4.0 8.3 11.8)
(4 4 3) + (3.0 2.5 7.3) = (7.0 6.5 10.3)
(1 0 1)   (2.1 4.8 4.4)   (3.1 4.8  5.4)
```

```
template<class... T> struct std::common_type;
```

Quiz time

```
auto a = complex<int>(7, 0) * complex<int>(5, 0) / complex<int>(3, 0);
```

Quiz time

```
auto a = complex<int>(7, 0) * complex<int>(5, 0) / complex<int>(3, 0);
// complex<int> a = {17,0}
```

Quiz time

```
auto a = complex<int>(7, 0) * complex<int>(5, 0) / complex<int>(3, 0);
// complex<int> a = {17,0}

auto a = complex<int>(7.0, 0.0) * complex<int>(5, 0) / complex<int>(3.0, 0.0);
```

Quiz time

```
auto a = complex<int>(7, 0) * complex<int>(5, 0) / complex<int>(3, 0);
// complex<int> a = {17,0}

auto a = complex<int>(7.0, 0.0) * complex<int>(5, 0) / complex<int>(3.0, 0.0);
// complex<int> a = {17,0}
```

Quiz time

```
auto a = complex<int>(7, 0) * complex<int>(5, 0) / complex<int>(3, 0);
// complex<int> a = {17,0}

auto a = complex<int>(7.0, 0.0) * complex<int>(5, 0) / complex<int>(3.0, 0.0);
// complex<int> a = {17,0}

auto a = complex<float>(7.0, 0.0) * complex<float>(5, 0)
                     / complex<float>(3.0, 0.0);
```

Quiz time

```
auto a = complex<int>(7, 0) * complex<int>(5, 0) / complex<int>(3, 0);
// complex<int> a = {17,0}

auto a = complex<int>(7.0, 0.0) * complex<int>(5, 0) / complex<int>(3.0, 0.0);
// complex<int> a = {17,0}

auto a = complex<float>(7.0, 0.0) * complex<float>(5, 0)
                     / complex<float>(3.0, 0.0);
// complex<float> a = {11.6666667f, 0.0f}
```

Quiz time

```
auto a = complex<float>(7.0, 0.0) * complex<int>(5, 0)
                     / complex<float>(3.0, 0.0);
```

Quiz time

```
auto a = complex<float>(7.0, 0.0) * complex<int>(5, 0)
                     / complex<float>(3.0, 0.0);
// malformed
```

Quiz time

```
auto a = complex<float>(7.0, 0.0) * complex<int>(5, 0)
                     / complex<float>(3.0, 0.0);
// malformed

auto a = complex<float>(7.0f, 0.0f) * complex<double>(5.0, 0.0)
                     / complex<float>(3.0f, 0.0f);
```

Quiz time

```
auto a = complex<float>(7.0, 0.0) * complex<int>(5, 0)
                     / complex<float>(3.0, 0.0);
// malformed

auto a = complex<float>(7.0f, 0.0f) * complex<double>(5.0, 0.0)
                     / complex<float>(3.0f, 0.0f);
// malformed
```

What to expect...

0. Representing linear equations
1. I can do better than this
2. Everything you need to know about storage
3. The upsetting story of std::complex
4. **Alternative algorithms**
5. Assembling the API

Operations

```
(2 2) * (1 4) = ((2*1)+(2*2) (2*4)+(2*1)) = ( 6 10)
(3 4)   (2 1)   ((3*1)+(4*2) (3*4)+(4*1))   (11 16)
```

Operations

```
(2 2) * (1 4) = ((2*1)+(2*2) (2*4)+(2*1)) = ( 6 10)
(3 4)   (2 1)   ((3*1)+(4*2) (3*4)+(4*1))   (11 16)

(2 2) * (0 4) = (0 (2*4)+(2*1)) = (0 10)
(3 4)   (0 1)   (0 (3*4)+(4*1))   (0 16)
```

Operations

Element promotion

Operations

Element promotion

Engine promotion

Operations

Element promotion

Engine promotion

Arithmetic promotion

Operations

Multiplication

Operations

Multiplication

$O(n^3)$

Operations

Multiplication

$O(n^3)$

Strassen - $O(n^{2.807})$

Operations

Multiplication

$O(n^3)$

Strassen - $O(n^{2.807})$

Best result - $O(n^{2.3728639})$

Operations

```
struct matrix_operation_traits {
    //- Addition
    //

    template<class T1, class T2>
    using addition_element_traits = matrix_addition_element_traits<T1, T2>;

    template<class OTR, class ET1, class ET2>
    using addition_engine_traits = matrix_addition_engine_traits<OTR, ET1, ET2>;

    template<class OTR, class OP1, class OP2>
    using addition_arithmetic_traits = matrix_addition_arithmetic_traits<OTR, OP1, OP2>;
```

Operations

```
//- Subtraction
//

template<class T1, class T2>
using subtraction_element_traits = matrix_subtraction_element_traits<T1, T2>;

template<class OTR, class ET1, class ET2>
using subtraction_engine_traits = matrix_subtraction_engine_traits<OTR, ET1, ET2>;

template<class OTR, class OP1, class OP2>
using subtraction_arithmetic_traits =
    matrix_subtraction_arithmetic_traits<OTR, OP1, OP2>;
```

Operations

```
//- Multiplication
//

template<class T1, class T2>
using multiplication_element_traits = matrix_multiplication_element_traits<T1, T2>;

template<class OTR, class ET1, class ET2>
using multiplication_engine_traits =
    matrix_multiplication_engine_traits<OTR, ET1, ET2>;

template<class OTR, class OP1, class OP2>
using multiplication_arithmetic_traits =
    matrix_multiplication_arithmetic_traits<OTR, OP1, OP2>;
```

Operations

```
    //- Scalar Division
    //

    template<class T1, class T2>
    using division_element_traits = matrix_division_element_traits<T1, T2>;

    template<class OTR, class T1, class T2>
    using division_engine_traits = matrix_division_engine_traits<OTR, T1, T2>;

    template<class OTR, class T1, class T2>
    using division_arithmetic_traits = matrix_division_arithmetic_traits<OTR, T1, T2>;
};
```

Operations

```
//- Addition operators
//

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator +(vector<ET1, OT1> const& v1, vector<ET2, OT2> const& v2);

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator +(matrix<ET1, OT1> const& m1, matrix<ET2, OT2> const& m2);
```

Operations

```
//- Subtraction operators
//

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator -(vector<ET1, OT1> const& v1, vector<ET2, OT2> const& v2);

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator -(matrix<ET1, OT1> const& m1, matrix<ET2, OT2> const& m2);
```

Operations

```
//- Scalar multiplication operators
//

template<class ET1, class OT1, class S2>
constexpr auto   operator *(vector<ET1, OT1> const& v1, S2 const& s2);

template<class S1, class ET2, class OT2>
constexpr auto   operator *(S1 const& s1, vector<ET2, OT2> const& v2);

template<class ET1, class OT1, class S2>
constexpr auto   operator *(matrix<ET1, OT1> const& m1, S2 const& s2);

template<class S1, class ET2, class OT2>
constexpr auto   operator *(S1 const& s1, matrix<ET2, OT2> const& m2);
```

Operations

```
//- Scalar division operators
//

template<class ET1, class OT1, class S2>
constexpr auto   operator /(vector<ET1, OT1> const& v1, S2 const& s2);

template<class ET1, class OT1, class S2>
constexpr auto   operator /(matrix<ET1, OT1> const& m1, S2 const& s2);
```

Operations

```
//- Vector and matrix division operators
//

template<class ET1, class OT1, class ET2, class OT2>
auto  operator /(vector<ET1, OT1> const& v1, vector<ET2, OT2> const& v2) = delete;

template<class ET1, class OT1, class ET2, class OT2>
auto  operator /(vector<ET1, OT1> const& v1, matrix<ET2, OT2> const& v2) = delete;

template<class ET1, class OT1, class ET2, class OT2>
auto  operator /(matrix<ET1, OT1> const& v1, vector<ET2, OT2> const& v2) = delete;

template<class ET1, class OT1, class ET2, class OT2>
auto  operator /(matrix<ET1, OT1> const& v1, matrix<ET2, OT2> const& v2) = delete;
```

Operations

```
//- Vector and matrix multiplication operators
//

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator *(vector<ET1, OT1> const& v1, matrix<ET2, OT2> const& m2);

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator *(matrix<ET1, OT1> const& m1, vector<ET2, OT2> const& v2);

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator *(vector<ET1, OT1> const& v1, vector<ET2, OT2> const& v2);

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto   operator *(matrix<ET1, OT1> const& m1, matrix<ET2, OT2> const& m2);
```

Operations

```
//- Related free functions.
//

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto  inner_product(vector<ET1, OT1> const& v1, vector<ET2, OT2> const& v2);

template<class ET1, class OT1, class ET2, class OT2>
constexpr auto  outer_product(vector<ET1, OT1> const& v1, vector<ET2, OT2> const& v2);
```

What to expect...

0. Representing linear equations
1. I can do better than this
2. Everything you need to know about storage
3. The upsetting story of std::complex
4. Alternative algorithms
5. **Assembling the API**

Enter The Matrix

```
matrix_storage_engine<float, extents<3, 3>>
```

Enter The Matrix

```
matrix_storage_engine<float, extents<3, 3>>

matrix_operation_traits
```

Enter The Matrix

```
matrix_storage_engine<float, extents<3, 3>>

matrix_operation_traits

template<class ET, class OT = matrix_operation_traits>
class matrix
```

Enter The Matrix

```
matrix_storage_engine<float, extents<3, 3>>

matrix_operation_traits

template<class ET, class OT = matrix_operation_traits>
class matrix

template<class ET, class OT = matrix_operation_traits>
class vector
```

Enter The Matrix

```
matrix_storage_engine<float, extents<3, 3>>

matrix_operation_traits

template<class ET, class OT = matrix_operation_traits>
class matrix

template<class ET, class OT = matrix_operation_traits>
class vector

namespace std::math
```

Enter The Matrix

```
template<class ET, class OT = matrix_operation_traits>
class matrix {
  public:
    //- Types
    //
    using engine_type            = ET;
    using element_type           = typename engine_type::element_type;
    using value_type             = typename engine_type::value_type;
    using reference              = typename engine_type::reference;
    using const_reference        = typename engine_type::const_reference;
    using difference_type        = typename engine_type::difference_type;
    using index_type             = typename engine_type::index_type;
    using index_tuple_type       = typename engine_type::index_tuple_type;
```

Enter The Matrix

```
using span_type              = ...;  //- implementation-defined
using const_span_type        = ...;  //- implementation-defined
using const_negation_type    =
    matrix<matrix_negation_engine<engine_type>, OT>;
using const_transpose_type =
    matrix<matrix_transpose_engine<engine_type>, OT>;
using const_hermitian_type =
    matrix<matrix_hermitian_engine<engine_type>, OT>;
using submatrix_type         =
    matrix<matrix_subset_engine<engine_type, ...>, OT>;
using const_submatrix_type =
    matrix<matrix_subset_engine<engine_type, ...>, OT>;
```

Enter The Matrix

```
using column_type           =
    vector<matrix_column_engine<engine_type, ...>, OT>;
using const_column_type     =
    vector<matrix_column_engine<engine_type, ...>, OT>;
using row_type              =
    vector<matrix_row_engine<engine_type, ...>, OT>;
using const_row_type        =
    vector<matrix_row_engine<engine_type, ...>, OT>;
```

Enter The Matrix

```
//- Construct/copy/destroy
//
constexpr matrix() = default;
constexpr matrix(matrix&&) noexcept = default;
constexpr matrix(matrix const&) = default;
template<class ET2, class OT2>
    constexpr matrix(matrix<ET2, OT2> const& rhs);
constexpr matrix(initializer_list<initializer_list<U>> rhs);
explicit constexpr matrix(index_tuple_type size);
constexpr matrix(index_type rows, index_type cols);
constexpr matrix(index_tuple_type size, index_tuple_type cap);
~matrix() noexcept = default;
```

Enter The Matrix

```
constexpr matrix&   operator =(matrix&&) noexcept = default;
constexpr matrix&   operator =(matrix const&) = default;

template<class ET2, class OT2>
constexpr matrix&   operator =(matrix<ET2, OT2> const& rhs);

template<class U>
constexpr matrix&   operator =(initializer_list<initializer_list<U>> rhs);
```

Enter The Matrix

```
//- Capacity
//
constexpr index_type        columns() const noexcept;
constexpr index_type        rows() const noexcept;
constexpr index_tuple_type  size() const noexcept;
constexpr index_type        column_capacity() const noexcept;
constexpr index_type        row_capacity() const noexcept;
constexpr index_tuple_type  capacity() const noexcept;
```

Enter The Matrix

```
void    resize_columns(index_type cols);
void    reserve_columns(index_type colcap);
void    reshape_columns(index_type cols, index_type colcap);
void    resize_rows(index_type rows);
void    reserve_rows(index_type rowcap);
void    reshape_rows(index_type rows, index_type rowcap);
void    resize(index_type rows, index_type cols);
void    reserve(index_type rowcap, index_type colcap);
void    reshape(index_type rows, index_type cols,
                index_type rowcap, index_type colcap);
```

Enter The Matrix

```
//- Element access
//
constexpr reference                operator ()(index_type i, index_type j);
constexpr const_reference          operator ()
                                       (index_type i, index_type j) const;
constexpr const_negation_type      operator -() const noexcept;
constexpr const_transpose_type     t() const noexcept;
constexpr const_hermitian_type     h() const;
constexpr submatrix_type           submatrix(index_type ri, index_type rn,
                               index_type ci, index_type cn) noexcept;
constexpr const_submatrix_type     submatrix(index_type ri, index_type rn,
                               index_type ci, index_type cn) const noexcept;
```

Enter The Matrix

```
constexpr column_type          column(index_type j) noexcept;
constexpr const_column_type    column(index_type j) const noexcept;
constexpr row_type             row(index_type i) noexcept;
constexpr const_row_type       row(index_type i) const noexcept;
```

Enter The Matrix

```
//- Data access
//
constexpr engine_type&              engine() noexcept;
constexpr engine_type const&        engine() const noexcept;
constexpr span_type                 span() noexcept;
constexpr const_span_type           span() const noexcept;
```

Enter The Matrix

```
    //- Modifiers
    //
    constexpr void      swap(matrix& rhs) noexcept;
    constexpr void      swap_columns(index_type c1, index_type c2) noexcept;
    constexpr void      swap_rows(index_type r1, index_type r2) noexcept;
};
```

Enter The Matrix

Enter The Matrix

```
matrix
```

Enter The Matrix

matrix

vector

Enter The Matrix

`matrix`

`vector`

`matrix_operation_traits`

Enter The Matrix

`matrix`

`vector`

`matrix_operation_traits`

`matrix_storage_engine`

A reminder: our goals

Provide linear algebra vocabulary types

Parameterise orthogonal aspects of implementation

Defaults for the 90%, customisable for power users

Element access, matrix arithmetic, fundamental operations

Mixed precision and mixed representation expressions

THANK YOU!
ASK ME TWO QUESTIONS...