Standardising a linear algebra library

Guy Davidson

Meeting C++ 15/11/2018

@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

Solve common least-squares and eigenvalue problems

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

Solve common least-squares and eigenvalue problems

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"

$a_1x_1 + a_2x_2 + … + a_nx_n = b$

Simultaneous equations

Linear algebra 101

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

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

Simultaneous equations

Geometry

Linear algebra 101

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

$a_1x_1 + a_2x_2 + … + a_nx_n = b$

Simultaneous equations

Geometry

Linear regression

Linear algebra 101

$(a_1, a_2 \dots a_n)$

Linear algebra 101

$(a_1, a_2 \ldots a_n)$

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

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

$(a_1, a_2 \dots a_n)$

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

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

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

Linear algebra 101

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

Linear algebra 101

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

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

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

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

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

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

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

Linear algebra 101

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

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)

(x) = A⁻¹*(e)
(y)       (f)
```

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

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

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

Linear algebra 101

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

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

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

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

Linear algebra 101

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

$|A| = -7$

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

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

Linear algebra 101

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

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

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

Linear algebra 101

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

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

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

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

Linear algebra 101

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

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

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

```
(x) = ((-2 * 8) + (-3 * 3)) / -7
(y)   ((-1 * 8) + ( 2 * 3)) / -7
```

@hatcat01

Linear algebra 101

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

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

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

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

Linear algebra 101

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

Linear algebra 101

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

$a_1x_1 + a_2x_2 = b$

Linear algebra 101

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

$a_1x_1 + a_2x_2 = b$

$ax + by = c$

@hatcat01

Linear algebra 101

$a_1x_1 + a_2x_2 + … + a_nx_n = b$

$a_1x_1 + a_2x_2 = b$

$ax + by = c$

$by = -ax + c$

@hatcat01

Linear algebra 101

$a_1x_1 + a_2x_2 + … + a_nx_n = b$

$a_1x_1 + a_2x_2 = b$

$ax + by = c$

$by = -ax + c$

$y = mx + c$

Linear algebra 101

Translate

```
(x, y) + (a, b) = (x+a, y+b)
```

Linear algebra 101

Scale

```
(x, y) * 2 = (2x, 2y)

(x, y) * (2 0) = (2x, 2y)
         (0 2)
```

Linear algebra 101

Shear

```
(x, y) * (1 4) = (x, 4x + y)
         (0 1)
```

Linear algebra 101

Reflect

```
(x, y) * (-1 0) = (-x, y)
         ( 0 1)
```

Linear algebra 101

Rotate

```
(x, y) * (cos a -sin a)
         (sin a  cos a)

= (x*cos a + y*sin a,
         -x*sin a + y*cos a)
```

Linear algebra 101

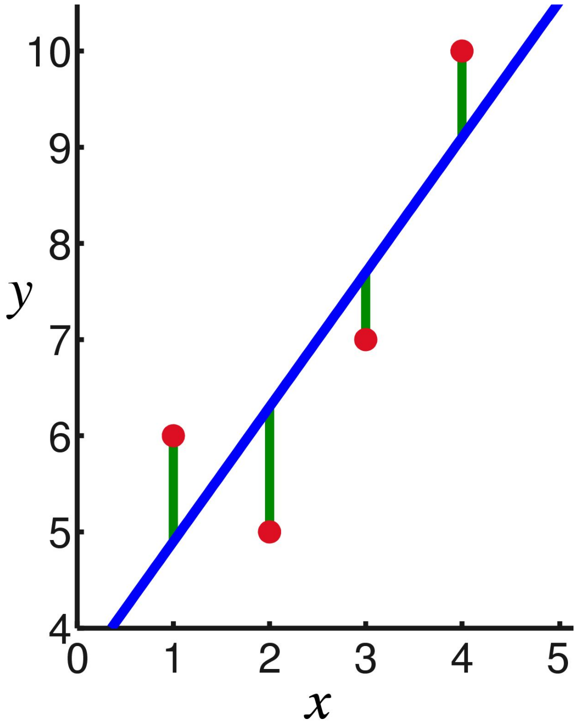
Reflect and shear

```
(x, y) * (-1 0) * (1 4)
         ( 0 1)   (0 1)

(x, y) * (-1 -4)
         ( 0  1)
```

Linear algebra 101

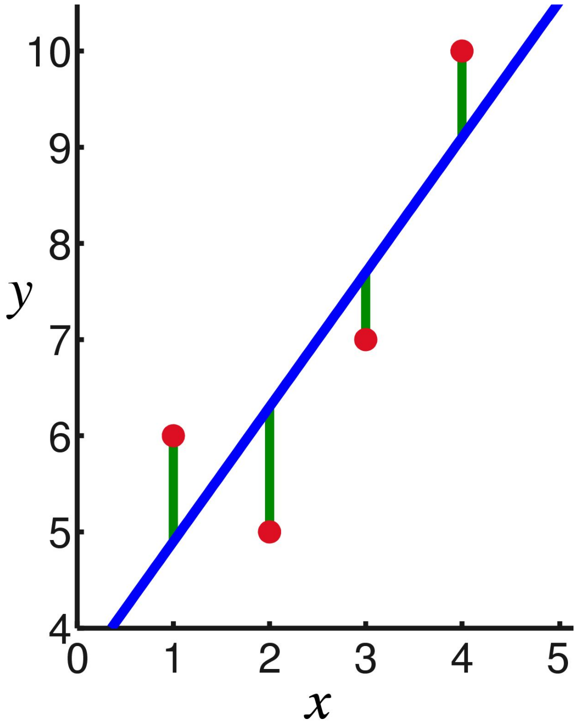
Linear regression



Linear algebra 101

Linear regression

Given x, what is y?

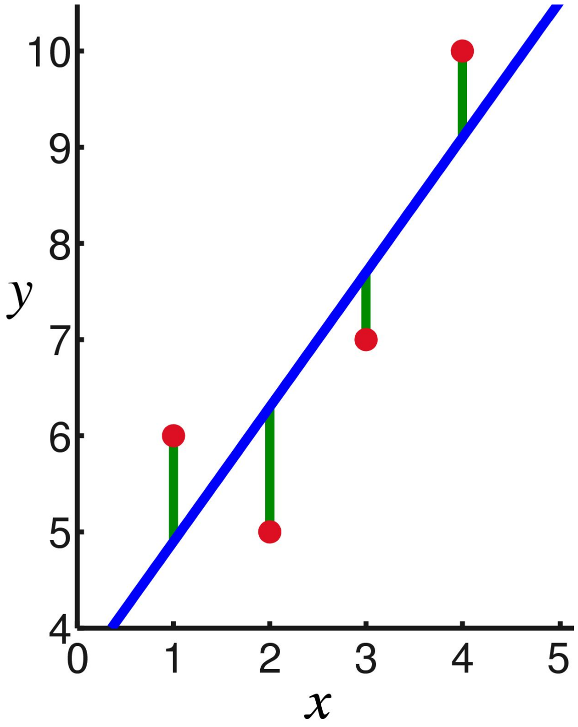


Linear algebra 101

Linear regression

Given x, what is y?

Predictive model



Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \dots x_{ip}\}^n_{i=1}$

Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \dots x_{ip}\}^n_{i=1}$

$y_i = \beta_0 1 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i , i = 1,\dots,n$

Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \ldots x_{ip}\}^n_{i=1}$

$y_i = \beta_0 1 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i, \; i = 1,\ldots,n$

$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$

Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \dots x_{ip}\}^n_{i=1}$

$y_i = \beta_0 1 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i, \ i = 1,\dots,n$

$\mathbf{y} = \mathbf{X\beta} + \boldsymbol{\varepsilon}$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \dots x_{ip}\}^n_{i=1}$

$y_i = \beta_0 1 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i = x_i^T\beta + \varepsilon_i \;, \; i = 1,\dots,n$

$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \dots \\ x_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix}$$

Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \ldots x_{ip}\}^n_{i=1}$

$y_i = \beta_0 1 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i = x_i^T\beta + \varepsilon_i, \quad i = 1,\ldots,n$

$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \ldots \\ x_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1p} \\ 1 & x_{21} & \ldots & x_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n1} & \ldots & x_{np} \end{pmatrix} \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \ldots \\ \beta_p \end{pmatrix}$$

Linear algebra 101

$\{y_i, x_{i1}, x_{i2}, \dots x_{ip}\}^n_{i=1}$

$y_i = \beta_0 1 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i \ , \ i = 1,\dots,n$

$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \dots \\ x_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix} \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{pmatrix} \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \dots \\ \varepsilon_n \end{pmatrix}$$

@hatcat01

Linear algebra 101

Linear algebra 101

$$h_i = \beta_1 t_i + \beta_2 t_i^2 + \varepsilon_i$$

Linear algebra 101

$h_i = \beta_1 t_i + \beta_2 t_i^2 + \varepsilon_i$

$\mathbf{x}_i = (x_{i1}, x_{i2}) = (t_i, t_i^2)$

Linear algebra 101

$h_i = \beta_1 t_i + \beta_2 t_i^2 + \varepsilon_i$

$\mathbf{x}_i = (x_{i1}, x_{i2}) = (t_i, t_i^2)$

$h_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i$

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

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

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

```
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& vec) : elems(vec.size()) {
  for (size_t i = 0; i != vec.size(); ++i)
    elems[i] = vec[i];
}
```

Prior art

```
template <typename E1, typename E2>
class vector_sum : public vector_expression<vector_sum<E1, E2>> {
public:
  vector_sum(E1 const& u, E2 const& v) : _u(u), _v(v) {}
  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>
vector_sum<E1,E2> operator+(E1 const& u, E2 const& v) {
  return vector_sum<E1, E2>(u, v); }
```

Prior art

```
template <typename E1, typename E2>
vector_sum<E1,E2> 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>
vector_sum<E1,E2> 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] = vec[i];
```

Prior art

```
template <typename E1, typename E2>
vector_sum<E1,E2> 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] = vec[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 <typename scalar, size_t row_count, size_t column_count>
class fixed_size_matrix
{
  public:
    constexpr fixed_size_matrix() noexcept;
    constexpr fixed_size_matrix(std::initializer_list<scalar> &&) noexcept;
    constexpr scalar& operator()(size_t, size_t);
    constexpr scalar operator()(size_t, size_t) const;
  private:
    scalar e[row_count * col_count];
};

operator[](std::pair<size_t, size_t>); // To be implemented
```

Storage

```
template<typename mdspan>
class matrix_view
{
  public:
    using scalar = mdspan::element_type;
    constexpr matrix_view(mdspan) noexcept;
    constexpr scalar operator()(size_t, size_t) const;
    constexpr size_t columns() const noexcept;
    constexpr size_t rows() const noexcept;
  private:
    mdspan m_span;
};
```

Storage

```
template <typename scalar, typename allocator>
class dynamic_size_matrix
{
  public:
    constexpr dynamic_size_matrix() noexcept;
    constexpr dynamic_size_matrix(std::initializer_list<scalar> &&) noexcept;
    constexpr scalar& operator()(size_t, size_t);
    constexpr scalar operator()(size_t, size_t) const;
    constexpr size_t columns() const noexcept;
    constexpr size_t rows() const noexcept;
```

Storage

```
    constexpr size_t column_capacity() const noexcept;
    constexpr size_t row_capacity() const noexcept;
    void reserve (size_t, size_t);
    void resize (size_t, size_t);

  private:
    unique_ptr<scalar> e;
    size_t m_rows;
    size_t m_cols;
    size_t m_row_capacity;
    size_t m_column_capacity;
};
```

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;
```

@hatcat01

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 = 11.6666666666666
```

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 = 11.66666666666666
auto a = 7. * 5.f / 3;        // double a = 11.66666666666666
```

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 = 11.666666666666666
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;
```

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 T1, class T2> using element_promotion_t =
typename element_promotion<T1, T2>::type;
```

Promotion and conversion

```
template<class T> struct is_complex
  : public false_type {};
```

Promotion and conversion

```
template<class T> struct is_complex
  : public false_type {};

template<class T> struct is_complex<std::complex<T>>
  : public std::bool_constant<std::is_arithmetic_v<T>> {};
```

Promotion and conversion

```
template<class T> struct is_complex
  : public false_type {};

template<class T> struct is_complex<std::complex<T>>
  : public std::bool_constant<std::is_arithmetic_v<T>> {};

template<class T>
inline constexpr bool is_complex_v = is_complex<T>::value;
```

Promotion and conversion

```
template<class T> struct is_matrix_element
  : public std::bool_constant<std::is_arithmetic_v<T> || is_complex_v<T>> {};
```

Promotion and conversion

```
template<class T> struct is_matrix_element
  : public std::bool_constant<std::is_arithmetic_v<T> || is_complex_v<T>> {};

template<class T>
inline constexpr bool is_matrix_element_v = is_matrix_element<T>::value;
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion_helper {
  static_assert(std::is_arithmetic_v<T1> && std::is_arithmetic_v<T2>);
  using type = decltype(T1() * T2());
};
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion_helper {
  static_assert(std::is_arithmetic_v<T1> && std::is_arithmetic_v<T2>);
  using type = decltype(T1() * T2());
};

template<class T1, class T2>
using element_promotion_helper_t =
  typename element_promotion_helper<T1, T2>::type;
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion_helper {
  static_assert(std::is_arithmetic_v<T1> && std::is_arithmetic_v<T2>);
  using type = decltype(T1() * T2());
};

template<class T1, class T2>
using element_promotion_helper_t =
  typename element_promotion_helper<T1, T2>::type;

template<class T1, class T2>
struct element_promotion {
  using type = element_promotion_helper_t<T1, T2>;
};
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion<T1, std::complex<T2>> {
  static_assert(std::is_same_v<T1, T2>);
  using type = std::complex<element_promotion_helper_t<T1, T2>>;
};
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion<T1, std::complex<T2>> {
  static_assert(std::is_same_v<T1, T2>);
  using type = std::complex<element_promotion_helper_t<T1, T2>>;
};

template<class T1, class T2>
struct element_promotion<std::complex<T1>, T2> {
  static_assert(std::is_same_v<T1, T2>);
  using type = std::complex<element_promotion_helper_t<T1, T2>>;
};
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion<std::complex<T1>, std::complex<T2>> {
  static_assert(std::is_same_v<T1, T2>);
  using type = std::complex<element_promotion_helper_t<T1, T2>>;
};
```

Promotion and conversion

```
template<class T1, class T2>
struct element_promotion<std::complex<T1>, std::complex<T2>> {
  static_assert(std::is_same_v<T1, T2>);
  using type = std::complex<element_promotion_helper_t<T1, T2>>;
};

template<class T1, class T2>
using element_promotion_t = typename element_promotion<T1, T2>::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
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
auto g = ds * fs;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
auto g = ds * fs; // ds
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
auto g = ds * fs; // ds
auto h = ds * mv;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
auto g = ds * fs; // ds
auto h = ds * mv; // ds
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
auto g = ds * fs; // ds
auto h = ds * mv; // ds
auto i = ds * ds;
```

Promotion and conversion

```
auto fs = matrix<fixed_size_matrix<float, 3, 3>>{};
auto mv = matrix<matrix_view<float, 3, 3>>{mdspan(blah)};
auto ds = matrix<dynamic_size_matrix<float>>{};

auto a = fs * fs; // fs
auto b = fs * mv; // fs
auto c = fs * ds; // ds
auto d = mv * fs; // fs
auto e = mv * mv; // fs
auto f = mv * ds; // ds
auto g = ds * fs; // ds
auto h = ds * mv; // ds
auto i = ds * ds; // ds
```

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

```
scalar_t inner_product(matrix_t const& lhs, matrix_t const& rhs) {
  return scalar_t(std::inner_product(lhs.cbegin(), lhs.cend(),
                                     rhs.cbegin(), scalar_t(0));
}
```

Operations

```
scalar_t modulus_squared(matrix_t const& mat) {
  return std::accumulate(mat.cbegin(), mat.cend(), scalar_t(0),
    [&](scalar_t tot, const auto& el) {
      return tot + (el * el); });
}
```

Operations

```
scalar_t modulus_squared(matrix_t const& mat) {
  return std::accumulate(mat.cbegin(), mat.cend(), scalar_t(0),
    [&](scalar_t tot, const auto& el) {
      return tot + (el * el); });
}

scalar_t modulus(matrix_t const& mat) {
  return std::sqrt(modulus_squared(mat));
}
```

Operations

```
matrix_t unit(matrix_t const& mat) {
  auto res(mat);
  auto mod(modulus(mat));
  std::transform(mat.cbegin(), mat.cend(), res.begin(),
    [&](const auto& el) {
      return el / mod;
    });
  return res;
}
```

Operations

```
matrix_t transpose(matrix_t const& mat) {
  auto res = matrix_t{};
  for (auto i = 0; i < mat::row(); ++i) {
    for (auto j = 0; j < mat::col(); ++j) {
      res._Data[i + j * mat::row()] = mat._Data[i * mat::col() + j];
    }
  }
  return res;
}
```

Operations

```
     (1 2 3)
M =  (4 5 6)
     (7 8 9)

submatrix(1,1) of M = (5 6)
                      (8 9)
```

Operations

```
auto submatrix(matrix_t const& mat, size_t i, size_t j) {
  auto l_in = mat.cbegin();
  auto res = submatrix_t::matrix_t;
  auto r_out = res.begin();
  for (auto r = 0U; r < mat.row(); ++r) {
    for (auto c = 0U; c < mat.col(); ++c) {
      if (r != i && c != j) *r_out = *l_in;
    }
    ++l_in;
    }
  }
  return res;
}
```

Operations

```
matrix_t inverse(matrix_t const& mat);

bool is_invertible(matrix_t const& mat);
```

Operations

```
matrix_t identity(size_t i) {
  auto res = matrix_t{};
  auto out = res.begin();
  auto x = res.row() + 1;
  for (auto y = 0; y != res.row() * res.row(); ++y, ++out) {
    *out = (x == res.row() + 1 ? 1 : 0;
    if (--x == 0) x = res.row() + 1;
  }
  return res;
}
```

Operations

```
scalar_t determinant(matrix_t const& mat) {
  if constexpr (mat::row() == 1) return mat._Data[0];
  else if constexpr (mat::row() == 2) return (mat._Data[0] * mat._Data[3]) -
                                             (mat._Data[1] * mat._Data[2]);
  else if constexpr (mat::row() > 2) {
    auto det = scalar_t{0};
    auto sign = scalar_t{1};
    for (auto f = 0; f < mat::row(); ++f) {
      auto cofactor = sign * mat._Data[f] * determinant(submatrix(mat, 0, f));
      det += cofactor;
      sign = -sign;
    }
  return det;
}}
```

@hatcat01

Operations

```
matrix_t classical_adjoint(matrix_t const& mat) {
  auto res = matrix_t{};
  for (auto i = 0; i < mat::row(); ++i) {
    auto sign = i % 2 == 0 ? scalar_t{1} : scalar_t{-1};
    for (auto j = 0; j < mat::row(); ++j) {
      auto det = determinant(submatrix(mat, i, j));
      res._Data[i * mat::row() + j] = sign * det;
      sign = -sign;
    }
  }
  return transpose(res);
}
```

Operations

```
template <typename Storage>
struct matrix_ops {
  using scalar_t = Storage::scalar_t;
  using matrix_t = Storage::matrix_t;
  template <class Ops2>
  using multiply_t = matrix_ops<
    typename Storage::template multiply_t<typename Ops2::matrix_t>>;

  static constexpr bool equal(matrix_t const& lhs, matrix_t const& rhs) noexcept;
  ...
  template <typename Ops2>
  static constexpr typename multiply_t<Ops2>::matrix_t matrix_multiply(
    matrix_t const& lhs, typename Ops2::matrix_t const& rhs) noexcept;
  ...
};
```

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

```
template <typename Storage>
struct my_matrix_ops {
  using scalar_t = Storage::scalar_t;
  using matrix_t = Storage::matrix_t;
  template <class Ops2>
  using multiply_t = matrix_ops<
    typename Storage::template multiply_t<typename Ops2::matrix_t>>;

  static constexpr bool equal(matrix_t const& lhs, matrix_t const& rhs) noexcept {
    return matrix_ops::equal(lhs, rhs);
  ...
  template <typename Ops2>
  static constexpr typename multiply_t<Ops2>::matrix_t matrix_multiply(
    matrix_t const& lhs, typename ops2::matrix_t const& rhs) noexcept;
  ...
};
```

@hatcat01

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

```
fixed_size_matrix<float, 3, 3>
```

Enter The Matrix

```
fixed_size_matrix<float, 3, 3>

matrix_ops<fixed_size_matrix<float, 3, 3>>
```

Enter The Matrix

```
fixed_size_matrix<float, 3, 3>

matrix_ops<fixed_size_matrix<float, 3, 3>>

template <typename REP> class matrix;
```

Enter The Matrix

```
fixed_size_matrix<float, 3, 3>

matrix_ops<fixed_size_matrix<float, 3, 3>>

template <typename REP> class matrix;

template <typename REP> class row_vector;
```

Enter The Matrix

```
fixed_size_matrix<float, 3, 3>

matrix_ops<fixed_size_matrix<float, 3, 3>>

template <typename REP> class matrix;

template <typename REP> class row_vector;

template <typename REP> class column_vector;
```

Enter The Matrix

```
template <typename REP> struct matrix {
  using scalar_t = typename REP::scalar_t;
  using matrix_t = typename REP::matrix_t;

  constexpr matrix() noexcept = default;
  constexpr matrix(matrix_t const&) noexcept = default;
  constexpr matrix(std::initializer_list<scalar_t>) noexcept;
  constexpr matrix(std::pair<size_t, size_t>) noexcept;

  constexpr matrix_t const& data() const noexcept;
  constexpr matrix_t& data() noexcept;
  constexpr scalar_t operator()(size_t, size_t) const;
  constexpr scalar_t& operator()(size_t, size_t);
```

Enter The Matrix

```
  constexpr bool operator==(matrix<REP> const& rhs) const noexcept;
  constexpr bool operator!=(matrix<REP> const& rhs) const noexcept;

  constexpr matrix<REP>& operator*=(scalar_t const& rhs) noexcept;
  constexpr matrix<REP>& operator/=(scalar_t const& rhs) noexcept;

  constexpr matrix<REP>& operator+=(matrix<REP> const& rhs) noexcept;
  constexpr matrix<REP>& operator-=(matrix<REP> const& rhs) noexcept;

  matrix_t _Data;
};
```

Enter The Matrix

```
template <typename REP> constexpr matrix<REP> operator*(
  matrix<REP> const&, typename matrix<REP>::scalar_t const&) noexcept;

template <typename REP> constexpr matrix<REP> operator*(
  typename matrix<REP>::scalar_t const&, matrix<REP> const&) noexcept;

template <typename REP> constexpr matrix<REP> operator/(
  matrix<REP> const&, typename matrix<REP>::scalar_t const&) noexcept;
```

Enter The Matrix

```
template <typename REP> constexpr auto transpose(
  matrix<REP> const&) noexcept;

template <typename REP> constexpr auto submatrix(
  matrix<REP> const&, size_t p, size_t q) noexcept;

template <typename REP> constexpr bool is_invertible(
  matrix<REP> const&) noexcept;

template <typename REP> constexpr bool is_identity(
  matrix<REP> const&) noexcept;
```

Enter The Matrix

```
template <typename REP> constexpr matrix<REP> operator+(
  matrix<REP> const&, matrix<REP> const&) noexcept;

template <typename REP> constexpr matrix<REP> operator-(
  matrix<REP> const&, matrix<REP> const&) noexcept;

template <typename REP1, typename REP2> constexpr auto operator*(
  matrix<REP1> const&, matrix<REP2> const&) noexcept;
```

Enter The Matrix

```
template <typename REP> constexpr matrix<REP> identity() noexcept;

template <typename REP> constexpr typename REP::scalar_t determinant(
  matrix<REP> const&);

template <typename REP> constexpr auto classical_adjoint(
  matrix<REP> const&);

template <typename REP> constexpr matrix<REP> inverse(
  matrix<REP> const&);
```

Enter The Matrix

```
template <typename REP> constexpr typename REP::scalar_t inner_product(
  row_vector<REP> const&, column_vector<REP> const&) noexcept;

template <typename REP> constexpr typename REP::scalar_t modulus(
  row_vector<REP> const&) noexcept;

template <typename REP> constexpr typename REP::scalar_t modulus_squared(
  row_vector<REP> const&) noexcept;

template <typename REP> constexpr row_vector<REP> unit(
  row_vector<REP> const&) noexcept;
```

Enter The Matrix

```
auto f_33 = matrix<matrix_ops<fixed_size_matrix<float, 3, 3>>>{};

auto f_13 = row_vector<matrix_ops<fixed_size_matrix<float, 1, 3>>>{};

auto f_31 = column_vector<matrix_ops<fixed_size_matrix<float, 3, 1>>>{};
```

Enter The Matrix

Enter The Matrix

```
matrix
```

Enter The Matrix

```
matrix

row_vector
```

Enter The Matrix

`matrix`

`row_vector`

`column_vector`

Enter The Matrix

`matrix`

`row_vector`

`column_vector`

`matrix_ops`

Enter The Matrix

`matrix`

`row_vector`

`column_vector`

`matrix_ops`

`fixed_size_matrix`

Enter The Matrix

```
matrix

row_vector

column_vector

matrix_ops

fixed_size_matrix

dynamic_size_matrix
```

Enter The Matrix

```
matrix

row_vector

column_vector

matrix_ops

fixed_size_matrix

dynamic_size_matrix

matrix_view
```

Enter The Matrix

```
auto f_33 = matrix<matrix_ops<fixed_size_matrix<float, 3, 3>>>{};

auto f_13 = row_vector<matrix_ops<fixed_size_matrix<float, 1, 3>>>{};

auto f_31 = column_vector<matrix_ops<fixed_size_matrix<float, 3, 1>>>{};
```

Enter The Matrix

```
template <size_t M, size_t N>
using matrix_impl = std::matrix_ops<
                      std::fixed_size_matrix<
                        float, M, N>>;
```

Enter The Matrix

```
template <size_t M, size_t N>
using matrix_impl = std::matrix_ops<
                      std::fixed_size_matrix<
                        float, M, N>>;

auto m = std::matrix<matrix_impl<3, 3>>{};
```

Enter The Matrix

```
template <size_t M, size_t N>
using matrix_impl = std::matrix_ops<
                      std::fixed_size_matrix<
                        float, M, N>>;

auto m = std::matrix<matrix_impl<3, 3>>{};

using float_33 = std::matrix<
                   std::matrix_ops<
                     std::fixed_size_matrix<
                       float, 3, 3>>>;
```

Enter The Matrix

```
template <size_t M, size_t N>
using matrix_impl = std::matrix_ops<
                      std::fixed_size_matrix<
                        float, M, N>>;

auto m = std::matrix<matrix_impl<3, 3>>{};

using float_33 = std::matrix<
                   std::matrix_ops<
                     std::fixed_size_matrix<
                       float, 3, 3>>>;

auto m = float_33{};
```

In summary...

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

https://groups.google.com/a/isocpp.org/forum/#!forum/sg14

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

Solve common least-squares and eigenvalue problems

Mixed precision and mixed representation expressions

THANK YOU!

Ask me two questions...