hdnum::Vector<T>

- `hdnum::Vector<T>` es una plantilla de Clase.
- Convierte cualquier (número) tipo de datos `T` en un vector.
- También es posible utilizar números complejos con mucha precisión.
- Los vectores se comportan como los conocemos por las matemáticas:
  - Tiene n componentes.
  - Inicia en el elemento 0 hasta el elemento $n-1$ numerados consecutivamente.
  - Adición y multiplicación por un escalar.
  - Producto escalar y norma Euclidiana.
  - Multiplicación vector-matriz.
- Los siguientes ejemplos se pueden encontrar en el archivo `vektoren.cc`

Construcción y Acceso

- Construcción con y sin Inicialización

```
hdnum::Vector<float> x(10);          // Vector con i
hdnum::Vector<double> y(10,3.14);    // Vector con i
hdnum::Vector<float> a;              // Vector sin e
```

- Vectores más específicos

```
hdnum::Vector<std::complex<double> >
  cx(7,std::complex<double>(1.0,3.0));
mpf_set_default_prec(1024); // Establece precision
mpf_class
hdnum::Vector<mpf_class> mx(7,mpf_class("4.44"));
```

- Acceso a un elemento

```
for (std::size_t i=0; i<x.size(); i=i+1)
  x[i] = i;                          // Acceso a cada eleme
```

- El objeto vectorial se elimina al final del ciclo for.

Copia y Asignación

- Constructor copia: crea una copia de otro vector

```
hdnum::Vector<float> z(x); // z es copia de x
```

- Asignación, ¡El tamaño también cambia!

```
b = z;                    // b copia los datos de z
a = 5.4;                  // asignacion a todos los elem
hdnum::Vector<double> w;    // Vector sin elementos
w.resize(x.size()); // Redimiensiona el vector
w = x;                    // Copia los elementos
```

- Extracto de vectores

```
hdnum::Vector<float> w(x.sub(7,3));// w es una cop
z = x.sub(3,4);                  // z es una copia de x
```

Cálculos y operaciones

- Operaciones de espacio vectorial y producto escalar

```
w += z;                  // w = w+z
w -= z;                  // w = w-z
w *= 1.23;               // Multiplicacion por escalar
w /= 1.23;               // Division por escalar
w.update(1.23,z);        // w = w + a*z
float s;
s = w*z;                 // Producto escalar
```

- Mostrando las salidas

```
std::cout << w << std::endl;// Salida por pantallo
w.iwidth(2);                // Digitos en la salio
w.width(20);                // Definiendo la canti
w.precision(16);            // Numero de posicione
std::cout << w << std::endl;// Imprimiendo w
std::cout <<cx << std::endl;// Imprimiendo comple
std::cout <<mx << std::endl;// Funciona para mpf_c
```

Visualización

```
[    0]      1.204200e+01
[    1]      1.204200e+01
[    2]      1.204200e+01
[    3]      1.204200e+01

[ 0]  1.20420007705688 48e+01
[ 1]  1.20420007705688 48e+01
[ 2]  1.20420007705688 48e+01
[ 3]  1.20420007705688 48e+01
```

Funciones auxiliares

```
float d = norm(w);                    // Norma Euclidea
d = w.two_norm();                     // La misma norma
zero(w);                              // Igual que w=0.0
fill(w,(float)1.0);                   // Igual que w=1.0
fill(w,(float)0.0,(float)0.1);        // w[0]=0, w[1]=0.1, w
unitvector(w,2);                      // Vector unitario cartes
gnuplot("test.dat",w);                // Salida a gnuplo: i w[i
gnuplot("test2.dat",w,z);             // Saida a gnuplot: w[i]
```

Funciones

- Ejemplo: Suma de todas las componentes

```
double sum (hdnum::Vector<double> x) {
  double s(0.0);
  for (std::size_t i=0; i<x.size(); i=i+1)
    s = s + x[i];
  return s;
}
```

- Versión mejorada de la función con **template** y uso por referencia

```
template<class T>
T sum (const hdnum::Vector<T>& x) {
  T s(0.0);
  for (std::size_t i=0; i<x.size(); i=i+1)
    s = s + x[i];
  return s;
}
```

- El uso por referencia es mejor para objetos grandes.

hdnum::DenseMatrix<T>

- `hdnum::DenseMatrix<T>` Es una plantilla o Template de clase.
- Convierte un elemento tipo `T` en una matriz.
- También es posible incluir complejos y su precisión.
- Las matrices son como las conocemos en matemáticas:
  - Dimensión de $m \times n$.
  - Inicia en elemento 0 hasta $m - 1$ o, hasta $n - 1$ numerados consecutivamente.
  - El conjunto de las matrices $m \times n$ constituyen un espacio vectorial.
  - Multiplicación de vectores y de matrices.
- Los siguientes ejemplos se pueden encontrar en el archivo `matrizen.cc`

Construcción y acceso

- Construcción con y sin inicialización

```
hdnum::DenseMatrix<float> B(10,10);      // 10x10 l
hdnum::DenseMatrix<float> C(10,10,0.0); // 10x10 l
```

- Acceso a elementos

```
for (int i=0; i<B.rowsize(); ++i)
  for (int j=0; j<B.colsize(); ++j)
    B[i][j] = 0.0;                  // Ahora la matriz B á
```

- El objeto matriz se elimina al final del ciclo.

Copia y asignación

- Constructor copia: Crea una matriz como copia de otra

```
hdnum::DenseMatrix<float> D(B); // D copia de B
```

- Asignación y tamaño de las copias

```
hdnum::DenseMatrix<float> A(B.rowsize(),B.colsize(
A = B;                    // Copia
```

- Extractos de matrices (Submatrices)

```
hdnum::DenseMatrix<float> F(A.sub(1,2,3,4));// 3x4
Submatriz (1,2)
```

Calculando con matrices

- Operaciones de espacio vectorial: Ojo, las matrices deben tener el tamaño correcto!)

```
A += B;                 // A = A+B
A -= B;                 // A = A-B
A *= 1.23;              // óMultiplicacin por escalar
A /= 1.23;              // óDivisin por escalar
A.update(1.23,B); // A = A + s*B
```

- Matrices, Vectores y multiplicación de matrices

```
hdnum::Vector<float> x(10,1.0); // Construimos dos
hdnum::Vector<float> y(10,2.0);
A.mv(y,x);                      // y = A*x
A.umv(y,x);                     // y = y + A*x
A.umv(y,(float)-1.0,x);         // y = y + s*A*x
C.mm(A,B);                      // C = A*B
C.umm(A,B);                     // C = C + A*B
A.sc(x,1);                      // Hace x la primera colu
A.sr(x,1);                      // Hace x la primera fila
float d=A.norm_infty();         // Halla la norma de la s
d=A.norm_1();                   // Halla la norma de la s
```

Funciones de salida auxiliares

- Salida de matrices

```
std::cout << A.sub(0,0,3,3) << std::endl;// Impres
A.iwidth(2);                             // Digitos en la salid
A.width(10);                             // Numero total de dig
A.precision(4);                          // Posiciones decimale
std::cout << A << std::endl;// Nueva publicacion
```

- Algunas funciones auxiliares

```
identity(A);
spd(A);
fill(x,(float)1,(float)1);
vandermonde(A,x);
```

Salida de muestra

```
                0             1             2             3
0   4.0000e+00 -1.0000e+00 -2.5000e-01 -1.1111e-01
1  -1.0000e+00  4.0000e+00 -1.0000e+00 -2.5000e-01
2  -2.5000e-01 -1.0000e+00  4.0000e+00 -1.0000e+00
3  -1.1111e-01 -2.5000e-01 -1.0000e+00  4.0000e+00
```

Funciones con argumentos matriciales

Ejemplo de una función de una matriz A y de un vector b inicializados.

```
template<class T>
void initialize (hdnum::DenseMatrix<T>& A, hdnum::Vect
{
  if (A.rowsize()!=A.colsize() || A.rowsize()==0)
    HDNUM_ERROR("need␣square␣and␣nonempty␣matrix");
  if (A.rowsize()!=b.size())
    HDNUM_ERROR("b␣must␣have␣same␣size␣as␣A");
  for (int i=0; i<A.rowsize(); ++i)
    {
      b[i] = 1.0;
      for (int j=0; j<A.colsize(); ++j)
        if (j<=i) A[i][j]=1.0; else A[i][j]=0.0;
    }
}
```