HDNUM

- C++ kennt keine Matrizen und Vektoren, ...
- Wir haben C++ erweitert um die **Heidelberg Educational Numerics Library**, kurz **HDNum**.
- Alle in der Vorlesung behandelten Beispiele sind dort enthalten.

Herunterladen von HDNUM

1. Einloggen
2. Erzeuge neues Verzeichnis mit

   ```
   $ mkdir kurs
   ```
3. Wechsle in das Verzeichnis mit

   ```
   $ cd kurs
   ```
4. Gehe zur Webseite `http://conan.iwr.uni-heidelberg.de/teaching/numerik0_ws2015/`
5. Klicke auf HDNum Version 0.23 inkl. Dokumentation und bestätige
6. Kopiere Datei `hdnum.tar` in das Verzeichnis:

   ```
   $ cp ~/Downloads/hdnum.tar .
   ```
7. Entpacken der Datei mit

   ```
   $ tar xvf hdnum.tar
   ```
8. Wechsle in das Verzeichnis

   ```
   $ cd hdnum/examples
   ```
9. Anzeigen der Dateien mittels `$ ls`

Bestandteile von HDNUM

- Vektoren
- Matrizen
- Timer

Vektoren in C++

- Die Klasse `std::vector` aus der Standard Template Library ist der komfortabelste Weg in C++ ein Feld von Werte anzulegen.
- Ein solcher Vektor ist eine angeordnete Menge von Werten. Die Elemente können über einen Index in eckigen Klammern angesprochen werden. Der erste Index ist *Null*.
- Um einen Standardvektor verwenden zu können muss die Headerdatei `vector` mit `#include <vector>` eingebunden werden.
- Vektoren werden wie normale Variablen angelegt. Der Variablentyp ist `std::vector<typ>`, wobei `typ` der Variablentyp der Elemente ist (Vektoren sind also eher Schablonen für Felder).

```
std::vector<int> intVector;
```

- Die Länge des Vektors (die Anzahl der Elemente) kann in runden Klammern nach dem Variablennamen angegeben werden.

```
std::vector<int> intVector(7);
```

Verwendung von `std::vector`

- Nach der Größe lässt sich ein Defaultwert für die Elemente angeben. Andernfalls ist der Wert der Elemente nicht definiert.

```
std::vector<int> intVector(7,0);
```

- Vektoren können als Kopie eines existierenden Vektors angelegt werden:

```
std::vector<int> intVector(7,0);
std::vector<int> secondVector(intVector);
```

- Die einzelnen Elemente werden durch Angabe des Index in eckigen Klammern nach dem Variablennamen ausgewählt. Der Index des ersten Elements ist `0`, der Index des letzten Elements ist `size-1`

```
intVector[1] = 3; // setzt den Wert des zweiten Elements auf 3
```

Verwendung der Methoden von `std::vector`

Vektoren haben spezielle Funktionen (Methoden), die mittels Variablenname gefolgt von einem Punkt und dem Namen der Funktion aufgerufen werden, z.B.

```
int size = intVector.size();    // size() liefert die Länge
                                // des Vektors zurück
```

Methodenname	Zweck
`size()`	gibt Länge des Vektors zurück
`resize(int newSize)`	ändert die Länge des Vektors. Zusätzliche Elemente werden nicht initialisiert. Ist der neue Vektor kürzer, wird der Rest abgeschnitten.
`front()`	liefert eine Referenz auf das erste Element
`back()`	liefert eine Referenz auf das letzte Element
`push_back(value)`	fügt ein Element am Ende hinzu (und erhöht die Länge um eins)
`clear()`	Löscht alle Elemente (Länge ist anschließend Null)

HDNUM Vektor

- `std::vector` sieht keine mathematischen Operationen mit Vektoren vor. Wir haben deshalb eine erweiterte Klasse `hdnum::Vector` geschaffen.
- Sie steht nach Einbinden des Headers `hdnum.hh` mit `#include "hdnum.hh"` zur Verfügung.
- Beim Übersetzen des Programms muss der Pfad zu dem Verzeichnis `hdnum` hinter der Option `-I` angeben werden, damit der Compiler die Headerdateien findet, z.B. zur Übersetzung des Programms `vektoren.cc` im Unterverzeichnis `examples` von `hdnum`

```
g++ -I.. -o vektoren vektoren.cc
```

HDNUM Vektor

```
// vektoren.cc
#include <iostream>     // notwendig zur Ausgabe
#include "hdnum.hh"     // hdnum header

template<class T>
void product (hdnum::Vector<T> &x)
{
  for (int i=1; i<x.size(); i=i+1)
    x[i] = x[i]*x[i-1];
}

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

HDNUM Vektor

```
int main ()
{
  // Konstruktion
  hdnum::Vector<float> x(10);          // Vektor mit 10 Elementen
  hdnum::Vector<double> y(10,3.14);    // 10 Elemente initialisiert
  hdnum::Vector<float> a;              // ein leerer Vektor
  x.resize(117);                  // vergrößern, Daten gelöscht!
  x.resize(23,2.71);              // verkleinern geht auch

  // Zugriff auf Vektorelemente
  for (int i=0; i<x.size(); i=i+1)
  x[i] = i;                      // Zugriff auf Elemente

  // Kopie und Zuweisung
  hdnum::Vector<float> z(x);           // Kopie erstellen
  z[2] = 1.24;                    // Wert verändern

  a = z;                          // hat Werte von z
  a[2] = -0.33;
  a = 5.4;                        // Zuweisung an alle Elemente

  hdnum::Vector<float> w(x);
```

HDNUM Vektor

```
w += z;                                  // w = w+z
w -= z;                                  // w = w-z
w *= 1.23;                               // skalare Multiplikation
w /= 1.23;                               // skalare Division
w.update(1.23,z);                        // w = w + a*z
x[0] = w*z;                              // skalare Multiplikation
std::cout << x.two_norm() << std::endl; // euklidische Norm

// Ausgabe
std::cout << w << std::endl;// schöne Ausgabe
w.iwidth(2);                             // Stellen in Indexausgabe
w.width(20);                             // Anzahl Stellen gesamt
w.precision(16);                         // Anzahl Nachkommastellen
std::cout << w << std::endl;// nun mit mehr Stellen

// Hilfsfunktionen
zero(w);                                 // das selbe wie w=0.0
fill(w,(float)1.0);                      // das selbe wie w=1.0
fill(w,(float)0.0,(float)0.1); // w[0]=0, w[1]=0.1, w[2]=0.2, ...
unitvector(w,2);                         // kartesischer Einheitsvektor
gnuplot("test.dat",w);                   // gnuplot Ausgabe: i w[i]
gnuplot("test2.dat",w,z);                // gnuplot Ausgabe: w[i] z[i]
```

```
  // Funktionsaufruf
  product(x);
  std::cout << "x=" << x << std::endl;
  std::cout << sum(x) << std::endl;
}
```

Beispielausgabe

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

[ 0] 1.204200077056848e+01
[ 1] 1.204200077056848e+01
[ 2] 1.204200077056848e+01
[ 3] 1.204200077056848e+01
```

HDNUM Matrix

- In C++ gibt es keine Standardtypen für Matrizen.
- Deshalb führt HDNUM auch einen Datentyp `DenseMatrix<typ>` ein.
- Er steht ebenfalls nach Einbinden des Headers `hdnum.hh` zur Verfügung.

HDNUM Matrix

```
// matrizen.cc
#include <iostream>     // notwendig zur Ausgabe
#include "hdnum.hh"     // hdnum header

// Beispiel wie man A und b für ein
// Gleichungssystem initialisieren könnte
template<class T>
void initialize (hdnum::DenseMatrix<T> &A, hdnum::Vector<T> &b)
{
  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;
  }
}
```

HDNUM Matrix

```
int main ()
{
  // Konstruktion
  hdnum::DenseMatrix<float> A;                // leere Matrix mit Größe 0
  hdnum::DenseMatrix<float> B(10,10);         // 10x10 Matrix uninitialis
  hdnum::DenseMatrix<float> C(10,10,0.0); // 10x10 Matrix initialisie

  // Zugriff auf Vektorelemente
  for (int i=0; i<B.rowsize(); ++i)
    for (int j=0; j<B.colsize(); ++j)
      B[i][j] = 0.0;                          // jetzt ist B initialisiert

  // Kopie und Zuweisung
  hdnum::DenseMatrix<float> D(B);     // D ist eine Kopie von B
  A = D;                                      // A ist nun identisch mit D!
  A[0][0] = 3.14;
  B[0][0] = 3.14;

  // Rechnen mit Matrizen und Vektoren
  A += B;                                     // A = A+B
  A -= B;                                     // A = A-B
  A *= 1.23;                                  // Multiplikation mit Skalar
  A /= 1.23;                                  // Division durch Skalar
  A.update(1.23,B);                           // A = A + s*B
```

HDNUM Matrix

```
  hdnum::Vector<float> x(10,1.0);     // make two vectors
  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

  // Ausgabe
  A.iwidth(2);                        // Stellen in Indexausgabe
  A.width(11);                        // Anzahl Stellen gesamt
  A.precision(4);                     // Anzahl Nachkommastellen
  std::cout << A << std::endl;// schöne Ausgabe

  // Hilfsfunktionen
  identity(A);                        // setze A auf Einheitsmatrix
  std::cout << A << std::endl;
  spd(A);                             // eine s.p.d. Matrix
  std::cout << A << std::endl;
  fill(x,(float)1,(float)1);
  vandermonde(A,x);                   // Vandermondematrix
  std::cout << A << std::endl;
}
```

Beispielausgabe

	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

HDNUM Timing

- Für Effizienzvergleiche ist es notwendig die Laufzeit numerischer Algorithmen zu messen.
- Dazu gibt es in HDNUM den Typ `hdnum::Timer` .
- Auch dafür ist das Einbinden des Headers `hdnum.hh` notwendig.

HDNUM Timing

```
// pendelmittimer.cc
#include <iostream> // header für Ein-/Ausgabe
#include <cmath>    // mathematische Funktionen
#include "hdnum.hh" // Zeitmessung
int main ()
{
  double l(1.34);   // Pendellänge in Meter
  double phi0(0.2); // Amplitude im Bogenmaß
  double dt(0.05);  // Zeitschritt in Sekunden
  double T(30.0);   // Ende in Sekunden
  hdnum::Timer zeit,zeitIter;
  for (double t=0.0; t<=T; t=t+dt)
  {
    zeitIter.reset();
    std::cout << t << " "
              << phi0*cos(sqrt(9.81/l)*t)
              << std::endl;
    std::cout << "Durchgang " << int(t/dt) << " brauchte "
              << std::scientific << zeitIter.elapsed()
              << " Sekunden" << std::endl;
  }
  std::cout << "Die Ausgabe aller Werte brauchte " << zeit.elapsed()
            << " Sekunden" << std::endl;
}
```