§4 Bsp: Selbstdef. numerischer Matrixtyp

Leitideen: *C++ stellt in der STL einen für numerische Operationen optimierten Vektordatentyp (valarray) bereit, allerdings keinen entsprechenden Matrixtyp.*

Es soll skizziert werden, wie sich damit ein für numerische Zwecke besser geeigneter Matrixtyp als `vector<vector<double>>` *aufbauen lässt.*

- Für Numerik optimierte Vektoren I,II
- Matrixtyp mit Elementzugriff per [ ][ ] (ohne Slices)
- Hilfsklasse slice_array
- Anmerkungen zum selbstdefinierten Matrixtyp

Für Numerik optimierte Vektoren

Eigenschaften

- Vereinb.: `valarray<T> a(n); // n Komponenten`
  `valarray<T> a(t,n);`
  umgekehrt wie bei `vector<T>`!
- Headerdatei: `<valarray>`
- Starke Einschränkungen beim Komponententyp T:
  Eingebaute arithmetische Typen, Zeiger, `complex` (STL)
  [und valarrays dieser Typen] sind zulässig
- Viele arithmetische und logische Vektoroperationen
  (Durchführung komponentenweise), z.B.
  `a+=b c=a*b c=a|b a<=b c=pow(a,b) a.sum()`
- a.resize(n) *löscht* Komponentenwerte (!!)
- Keine Iteratoren vorhanden
- `a[i]`: i ganzzahlig und vorzeichenlos wie bei C-Vektoren
- Dichte Speicherung ähnlich wie bei C-Vektoren:
  `&a[i+j]=&a[i]+j` , allerdings im allg. `&a`≠`&a[0]`

Für Numerik optimierte Vektoren II

Indexmengen

- Valarray: kein Matrixtyp, jedoch Operatoren und Funktionen vorhanden, mit denen effiziente Realisierung möglich
- Hierzu erforderlich: Operatoren zur Teilvektorbildung – Auswahl über verallgemeinerte Indizes (Indexmengenbeschreibungen)
  1. `slice` arithmetische Folge, z.B. für Matrixzeilen
  2. `gslice` arithmetisches Gitter, z.B. für Untermatrizen
  3. `valarray<bool>` Bitmaske
  4. `valarray<size_t>` indirekte Adressierung

Bsp. für slice: [`slice s(i0,n,h)` beschreibt $\{i_0+kh\colon k=0,\ldots,n-1\}$]

$(a_{ij})_{\substack{i=0,\ldots,m-1\\ j=0,\ldots,n-1}} \mathrel{\hat{=}} (v[i\cdot n+j])_{\substack{i=0,\ldots,m-1\\ j=0,\ldots,n-1}}$ (bei zeilenweiser Speich.)

Indizes i-te Zeile: $(i\cdot n+j)_{j=0,\ldots,n-1} \rightarrow$ `slice(i*n,n,1)`

Indizes j-te Spalte: $(i\cdot n+j)_{i=0,\ldots,m-1} \rightarrow$ `slice(j,m,n)`

Matrixtyp mit Elementzugriff per [ ][ ] (ohne Slices)

- $A \in \mathbb{R}^{m \times n}$, Speicherung auf valarray `v` mit $m \cdot n$ Komp.
- Indexabbildung: $(i, j) \to i \cdot n + j$ (zeilenweise Speicherung)
- Zugriff `a[i][j]` $\to$ Zeile `a[i]` mit folg. Indexzugriff `[j]`
- Wegen `a[i][j]` $\equiv$ `v[i*n+j]` muss die Klasse `zeile` die Komponenten `i`, `n` und einen Verweis auf `v` enthalten (Referenz wg. Effizienz) .
  Zusätzlich: `T& operator[](int j)`
- Entspr. für Klasse `konstzeile` konst. Ref. auf `v` notw.
  Zusätzlich: `T  operator[](int j) const`
- Die Klasse `matrix` muss `v`, `m`, `n` enthalten. Außerdem:
  ```
  zeile<T>      operator[](int i)
  konstzeile<T> operator[](int i) const
  ```
- Reihenfolge der Klassenvereinb.: `zeile`, `konstzeile`, `matrix` (Vereinb. von `matrix` zuerst auch möglich, aber erheblich komplizierter: Vorwärtsdekl., teilweise Dekl. in den Klassen und Def. außerhalb)

Hilfsklasse slice_array

- `a valarray`, `s slice` $\Rightarrow$ `a[s] slice_array (s_a)`
  Erforderlich, weil `a[s]` $=$ `b` möglich sein soll.
- Vorhanden: Arithmetische Zuweisungsoperatoren, z.B.
  `s_a=b`, `s_a+=b`, d.h. `a[s]=b`, `a[s]+=b`
- Typumwandlung `slice_array` $\rightarrow$ `valarray` public,
  daher sind z.B. zulässig
  `a=s_b`, `a+=s_b`, d.h. `a=b[s]`, `a+=b[s]`
  `s_a+=t_b`, d.h. `a[s]+=b[t]`
- Nicht vorhanden: binäre arith. Operatoren,
  daher sind z.B. unzulässig:
  `s_a=t_b+u_c`, d.h. `a[s]=b[t]+c[u]`
  Umgehungsmöglichkeit: Typumwandlungen in valarray
  oder geeignete valarray-Temporärvariablen

Anmerkungen zum selbstdef. Matrixtyp

- $A \in \mathbb{R}^{m \times n}$, `a[i][j]`$\hat{=}a_{ij}$, `a[i]`$\hat{=}(a_{ij})_{j=0,\ldots,n-1}$ (Zeile i) naheliegend: `a[i]` $\rightarrow$ `v[slice(i*n,n,1)]` slice_array! Indexoperator `[]` in Matrixklasse entsprechend definieren!
- *Ansatz:*
```
slice_array<T> operator[](int i)
    { return v[slice(i*n,n,1)]; }
```
Reicht nicht aus! (U.a. kein Indexop. in `slice_array`)
- *Abhilfe:* Selbstdef. Klassen `myslice_array` und `mycslice_array` als Ersatz für nichtkonstante und konstante Slice_arrays. Öffentliche Konstruktoren.
- Daten in `myslice_array`: Slice `s` und Referenz auf Valarray `a` (Effizienz)
- Daten in `matrix`: `m`, `n`, `v` (Valarray)
- Indexoperator `[]` in `matrix`: Liefert `myslice_array`
- Indexoperator `[]` in `myslice_array`: Liefert `T&`
- Klasse ähnlich zu Stroustrup Abschnitt 22.4.6
- Verschiedene Open-Source-Matrixklassen verfügbar, z.B. `ublas` auf `www.boost.org` und `eigen`