joelixblas v1.1

Installation Guide and Reference Manual

Clelia Albrecht, Felix Boes, Johannes Holke

3. April 2017

Inhaltsverzeichnis

1 Einleitung **4**

2 Kompilieren und Linken **5**
2.1 Kompilieren 5
2.2 Linken 5

3 Das CSR-Format **6**

4 Fehlercodes und Fehlermanagement **8**

5 `joelixblas` Funktionen **9**
5.1 `include/joelix_error.h` 9
5.1.1 Joelix_Fehler 9
5.2 Dokumentation der Funktionen 10
5.2.1 joelix_fehler_beschreibung 10
5.3 Variablen-Dokumentation 10
5.3.1 joelix_fehler_code 10
5.4 `include/vektor.h` 10
5.4.1 Joelix_Vektor 10
5.4.2 joelix_vektor_init 10
5.4.3 joelix_vektor_loeschen 10
5.4.4 joelix_vektor_null 11
5.4.5 joelix_vektor_laenge 11
5.4.6 joelix_vektor_geti 11
5.4.7 joelix_vektor_seti 11
5.4.8 joelix_vektor_copy 12
5.4.9 joelix_vektor_ax 13
5.4.10 joelix_vektor_axpy 13
5.4.11 joelix_vektor_dot 13
5.4.12 joelix_vektor_print 13
5.4.13 joelix_vektor_print_tofile 14
5.5 `include/matrix.h` 15
5.5.1 Joelix_sMatrix 15
5.5.2 joelix_smatrix_init 15
5.5.3 joelix_smatrix_loeschen 15
5.5.4 joelix_smatrix_fuelleZeile 15
5.5.5 joelix_smatrix_aendernneintrag 16
5.5.6 joelix_smatrix_get_spalten 16
5.5.7 joelix_smatrix_get_zeilen 16
5.5.8 joelix_smatrix_print 17
5.5.9 joelix_smatvec 17

6 Ein einfaches Beispiel **18**
6.1 Vektor erstellen 18
6.2 Matrix-Vektor-Produkt 19

1 Einleitung

Das vorliegende Handbuch beschreibt `joelixblas`, eine Software-Bibliothek für grundlegende lineare Algebra Operationen, welche unter der *GNU General Public License, Version 3* veröffentlicht wurde.

`joelixblas` wurde entwickelt für den **C**-Programmierkurs für Fortgeschrittene von Clelia Albrecht, Felix Boes und Johannes Holke und hat zum Ziel, die Implementation eines Poisson-Lösers zu vereinfachen und dabei den Umgang mit externen Bibliotheken zu lehren. Zur optimalen Nutzung der Bibliothek wird daher der Besuch der begleitenden Vorlesung oder zumindest das dazu gehörige Skript [ABH17] empfohlen.

Weitere Informationen zu diesem Software-Projekt (einschließlich der `joelixblas` Bibliothek selbst) finden sich unter `http://github.com/joelix/joelixblas`.

Dieses Manual soll die Nutzung von `joelixblas` erleichtern. Dazu ist es wie folgt aufgebaut:

Hinweise zum Kompilieren und Linken der Bibliothek und genauere Beschreibung wie man sie für eigene Programme nutzbar machen kann finden sich in **Abschnitt 2**.

Die Bibliothek stellt dem Nutzer Routinen zum Umgang mit Vektoren und Matrizen zur Verfügung. Dabei erwarten die bereitgestellten Funktionen die Matrizen im speichersparenden *Comparsed Sparse Row* (CSR) Format, welches in **Abschnitt 3** genauer erklärt wird.

Zur einfacheren Fehlerbehandlung geben die Funktionen einen Fehlercode zurück. Um diesen effektiv interpretieren zu können werden sie in **Abschnitt 4** erklärt. Dem Nutzer wird die Lektüre dieses Kapitels ans Herz gelegt, da Fehlerbehandlungen in eigenen Programmen somit sehr erleichtert werden.

Die Dokumentation der `joelixblas` Funktionen, die dem Nutzer zur Verfügung gestellt werden findet sich in **Abschnitt 5**.

Um den Einstieg zu erleichtern befindet sich am Schluss dieses Handbuchs, in **Abschnitt 6** ein kleines Programmbeispiel.

Literatur

[ABH17] Clelia Albrecht, Felix Boes, and Johannes Holke. *C-Programmierkurs für Fortgeschrittene.* selfpublished, 2017.

2 Kompilieren und Linken

2.1 Kompilieren

Der Nutzer sollte die Datei `joelixblas-1.1.tar.gz` von der Seite `http://github.com/joelix/joelixblas/releases` herunterladen und in den Ordner `joelixblas-1.1` entpacken. Nun, sollte man im Terminal in diesen Ordner navigiern und `make` ausführen:

```
1 $ cd joelixblas-1.1
2 $ make
```

Danach sollten sich zusätzlich zu weiteren Dateien diese drei Unterordner in dem Ordner befinden: `build`, `joelixblas` und `lib`.

Dann hat die Installation geklappt.

2.2 Linken

Die headerdateien, die ein User in sein Programm einfügen kann, liegen im Unterordner `joelixblas/include` und die kompilierte Bibliothek im Unterordner `lib`. Ein Programm, welches gegen `joelixblas` linkt sollte deshalb mit den Kompilierflags

- `-I/path/to/joelixblas-1.1/joelixblas/include`
- `-L/path/to/joelixblas-1.1/lib`
- `-ljoelixblas`

kompiliert werden. `/path/to/joelixblas-1.1` ist hierbei Platzhalter für den eigentlichen Dateipfad zum Bibliotheksordner und ist vom User entsprechend anzupassen.

Ein konkretes Beispiel wird in Kapitel 6 besprochen.

3 Das CSR-Format

An dieser Stelle beschreiben wir das von der `joelixblas` Bibliothek benutzte Datenformat um Matrizen abzuspeichern. Die Erklärungen an dieser Stelle beschreiben allgemein das CSR-Format und spiegeln nicht die konkrete Implementation wieder.

Die `joelixblas` Bibliothek benutzt das sogenannte CSR-Format für Matrizen. Dieses Format ist besonders gut geeignet, um Matrizen mit vielen 0 Einträgen abzuspeichern, da für diese Einträge kein Speicher verwendet wird.

In der Mathematik tauchen solche Matrizen häufig auf, zum Beispiel als Adjazenmatrix eines Graphen mit einer großen Knotenanzahl N, in dem jeder Knoten mit höchstens $n < N$ anderen Knoten verbunden ist. Ein anderes Beispiel sind Bandmatrizen, diese tauchen oft in der numerischen Mathematik auf. Bandmatrizen sind Matrizen in denen nur einigen Diagonalen von 0 verschiedenen Werte haben. Ein weiteres Beispiel sind die Randmatrizen eines simplizialen Komplex, die in der algebraischen Topologie eine fundamentale Bedeutung spielen.

Sei im folgenden M eine $n \times m$ Matrix mit genau N von 0 verschiedenen Einträgen. Um M im CSR-Format zu speichern, definieren wir die drei Arrays `werte`, `spalten` und `zeilen`. Die Arrays `werte` und `spalten` haben Länge N und `zeilen` hat Länge $n + 1$.

In dem Array `werte` speichern wir die Werte der nicht-null Einträge in lexikographischer Ordnung, d.h. Eintrag a_{ij} kommt vor $b_{i'j'}$ genau dann, wenn $i < i'$ erfüllt ist oder aber $i = i'$ und $j < j'$ gleichzeitig gilt. Insbesondere speichern wir also zuerst die Werte aus Zeile 1, dann die aus Zeile 2 und so weiter.

In dem Array `spalten` speichern wir in Index i den Spaltenindex des i-ten nicht-null Eintrags.

In dem Array `zeilen` speichern wir in Index i die Gesamtzahl an nicht-null Einträgen in allen Spalten vor Spalte i.

Beispiel 1. Wir betrachten die 5×5 Matrix M:

$$M = \begin{pmatrix} 2 & 0 & 0 & 1 & 0 \\ 0 & -7/10 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 42 & 17 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{1}$$

Für M sehen die obigen Arrays des CSR-Formates wie folgt aus.

Listing 1: Das CSR-Format für M

```
1 werte = {2, 1, -0.7, 1, 3, 42, 17, -1}
2 spalten = {0, 3, 1, 2, 4, 1, 2, 0}
3 zeilen = {0, 2, 5, 5, 7, 8}
```

Wir sehen zum Beispiel, dass der dritte nicht-null Eintrag `werte`$[2] = -0.7$ ist und in Spalte `spalten`$[2] = 1$ zu finden ist.

Da `zeilen` an der Stelle i die Anzahl an nicht-null Werten in allen Zeilen vor Zeile i speichert, ist `werte[zeilen[i]]` der Wert des ersten nicht-null Eintrags in Zeile i. Der zugehörige Spaltenindex ist `spalten[zeilen[i]]`. Will man also bei einer im CSR-Format gegebenen Matrix den Eintrag a_{ij} auslesen, so startet man mit $k = \texttt{zeilen}[i]$ und erhöht k bis $\texttt{spalten}[k] \geq j$. Falls nun $k > j$, dann ist $a_{ij} = 0$, falls $k = j$, dann ist $a_{ij} = \texttt{werte}[k]$.

Beispiel 2. Wir beschreiben hier kurz die Matrix-Vektor-Multiplikation mit Hilfe des CSR-Formates. Gegeben sei ein $n \times n$ Matrix M im CSR-Format und ein Vektor x als n-dimensionales Array. Wir berechen $Mx = b$.

Listing 2: Matrix-Vektor Multiplikation

```
1   int i, k;
2   for (i = 0;i < n;i++) {
3     /* Schleife ueber alle Zeilen der Matrix */
4     x[i] = 0;
5     for (k = M->zeilen[i];k < M->zeilen[i+1];k++) {
6       /* Schleife ueber alle nicht-null Eintrage dieser Zeile */
7       /* Der Wert an Stelle k steht in Spalte spalten[k] und muss
8          dehalb  mit dem Wert in Zeile spalten[k] von x multi-
9          pliziert werden. */
10      x[i] += M->werte[k] * x[M->spalten[k]];
11    }
12  }
```

Wir sehen in Bespiel 2, dass die Matrix-Vektor-Multiplikation Laufzeit $\mathcal{O}(N)$ hat, im Gegensatz zu $\mathcal{O}(nm)$ bei der Matrix-Vektor-Multiplikation mit einer als $n \times m$ Array gespeicherten Matrix.

Benutzt man die `joelixblas` Bibliothek, so wir eine Matrix durch ein Objekt vom Datentyp `Joelix_sMatrix` repräsentiert. Um eine solche Matrix zu erstellen, ruft man zuerst die Funktion `joelix_smatrix_init` auf und danach für jede Zeile mit nicht-nulleinträgen die Funktion `joelix_smatrix_fuelleZeile`.

4 Fehlercodes und Fehlermanagement

Fast jede Funktion in `joelixblas` gibt einen Fehlercode zurück, welcher angibt, ob die Funktion fehlerfrei gelaufen ist, oder ob es bei der Ausführung zu einem Fehler kam. **Falls eine Funktion nicht fehlerfrei gelaufen ist, so liegt es in der Verantwortung des Users hierauf zu reagieren.**

Der Datentyp `Joelix_Fehler` dient zum abspeichern des Fehlercodes und ist der Rückgabetyp der meisten Funktionen. `joelixblas` stellt zwei Möglichkeiten zur Verfügung, um den von einer Funktion zurückgegebenen Fehlercode abzufragen.

Die erste Möglichkeit ist, den Rückgabewer einer aufgerufenen Funktion auf `F_ERFOLG` zu überprüfen, wie hier anschaulich an der Funktion `joelix_vektor_init` gezeigt wird.

```
1  Joelix_Vektor V;
2  Joelix_Fehler fehlercode;
3
4  /* Erstelle einen Vektor der Laenge 10 */
5  fehlercode = joelix_vektor_init (&V, 10);
6  if (fehlercode == F_ERFOLG) {
7   /* Weiterer Programmcode bei erfolgreicher Initialisierung. */
8  }
9  else {
10  /* Fehlerbehandlung */
11 }
```

Als zweite Möglichkeit kann auch der Wert der Variable `joelix_fehler_code` ausgelesen werden. Hier speichert `joelixblas` immer den Fehlercode des letzten Funktionsaufrufs.

```
1  Joelix_Vektor V;
2
3  /* Erstelle einen Vektor der Laenge 10 */
4  joelix_vektor_init (&V, 10);
5  if (joelix_fehler_code == F_ERFOLG) {
6   /* Weiterer Programmcode bei erfolgreicher Initialisierung. */
7  }
8  else {
9   /* Fehlerbehandlung */
10 }
```

Um mehr Information über einen Fehler zu erfahren, kann die Funktion `joelix_fehler_beschreibung` genutzt werden. Diese gibt zu einer gegebenen `Joelix_Fehler` Variable einen String mit einer Beschreibung des Fehlers zurück.

Beispiel:

```
1  /* Gebe die Beschreibung des letzten Fehlers in der Konsole aus */
2  printf ("%s\n", joelix_fehler_beschreibung (joelix_fehler_code));
```

5 joelixblas Funktionen

5.1 include/joelix_error.h

Die Headerdatei include/joelix_error.h stellt die grundlegenden Funktionen zur Erkennung von Laufzeitfehlern zur Verfügung. Siehe dazu auch Kapitel 4.

5.1.1 enum Joelix_Fehler

Der Fehlertyp, welcher als Rückgabewert fast aller Funktionen dient.

Aufzählungswerte

F_ERFOLG Alles it gut, kein Fehler.

F_KEIN_SPEICHER Speicherfehler

F_FALSCHE_PARAMETER Parameterfehler

F_FALSCHER_INDEX Index ausserhalb des erlaubten Bereichs

F_FALSCHE_DIMENSIONEN_VEKTOR_VEKTOR Die zwei Input Vektoren haben verschiedene Laengen.

F_FALSCHE_DIMENSIONEN_VEKTOR_KOPIE Die zwei Input Vektoren haben verschiedene Laengen.

F_FALSCHE_DIMENSIONEN_MATRIX_VEKTOR Matrix und Vektor Dimensionen passen nicht zusammen.

F_FALSCHE_DIMENSIONEN_MATRIX_MATRIX Die zwei input Matrizen haben verschiedene Dimensionen.

F_FALSCHE_DIMENSIONEN_MATRIX_NICHT_QUADRATISCH Eine nicht-quadratische Matrix wurde uebergeben.

F_FALSCHE_ANZAHL_NICHT_NULL_WERTE Die falsche Anzahl an nicht-null Werten wurde uebergeben.

F_FILEIO_FEHLER Beim Oeffnen oder Schreiben oder Speichern einer Datei ist ein Fehler aufgetreten.

F_CG_TERMINIERT_NICHT Das CG-Verfahren ist nicht nach den maximal vorgegebenen Schritten konvergiert.

5.2 Dokumentation der Funktionen

5.2.1 const char* joelix_fehler_beschreibung (const Joelix_Fehler Fehler)

Gebe eine kurze Beschreibung eines Fehlers als string zurueck.
Parameter

in	*Fehler*	Der Fehler.

Rückgabe

Ein String, welcher eine Beschreibung des Fehlers enthaelt.

5.3 Variablen-Dokumentation

5.3.1 Joelix_Fehler joelix_fehler_code

Variable, welche immer den zuletzt erzeugten Fehlercode speichert.

5.4 include/vektor.h

Die Headerdatei include/vektor.h stellt grundlegende Funktionen für die Arbeit mit Vektoren zur Verfügung.

5.4.1 Joelix_Vektor

Der Datentyp fuer Vektoren.

5.4.2 Joelix_Fehler joelix_vektor_init (Joelix_Vektor * pVektor, int n)

Initialisiert einen Vektor der Laenge n und fuellt diesen mit Nullen auf.
Parameter

in,out	*pVektor*	Pointer auf den Vektor (vom Typ Joelix_Vektor) der initialisiert werden soll.
in	*n*	Laenge des zu erstellenden Vektors.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.3 Joelix_Fehler joelix_vektor_loeschen (Joelix_Vektor * pVektor)

Gibt den Speicher, der von einem Vektor benutzt wird, wieder frei.

Parameter

`in,out`	*pVektor*	Pointer auf einen von joelix_vektor_init initialiserter Vektor. Ist nach Ausführen der Funktion NULL.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.4 Joelix_Fehler joelix_vektor_null (Joelix_Vektor x)

Setze alle Eintraege eines Vektor auf den Wert 0.
Parameter

`in`	*x*	Ein mit joelix_vektor_init initialisierter Vektor.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.5 int joelix_vektor_laenge (Joelix_Vektor x)

Gebe die Laenge eines Vektors aus.
Parameter

`in`	*x*	Ein mit joelix_vektor_init initialisierter Vektor.

Rückgabe

Die Laenge des Vektors x oder -1, bei nicht initialisiertem Vektor.

5.4.6 Joelix_Fehler joelix_vektor_geti (Joelix_Vektor x, int i, double * wert)

Lese den i-ten Eintrag eines Vektors aus.
Parameter

`in`	*x*	Ein mit joelix_vektor_init initialisierter Vektor.
`in`	*i*	Ein Index innerhalb des Vektors, 0 <= i < n.
`in`	*wert*	Pointer auf einen allozierten double. Dieser double wird auf den i-ten Eintrag von x gesetzt.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.7 Joelix_Fehler joelix_vektor_seti (Joelix_Vektor x, int i, double wert)

Setze den i-ten Eintrag eines Vektors auf einen gegebenen Wert.

Parameter

`in,out`	*x*	Ein mit joelix_vektor_init initialisierter Vektor.
`in`	*i*	Ein Index innerhalb des Vektors, 0 <= i < n.
`in`	*wert*	Der Wert auf den der i-te Eintrag von x gesetzt werden soll.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.8 Joelix_Fehler joelix_vektor_copy (Joelix_Vektor y, Joelix_Vektor x)

Berechnet y = x.
Parameter

`in`	*x*	Ein mit joelix_vektor_init initialisierter Vektor.
`in,out`	*y*	Ein mit joelix_vektor_init initialisierter Vektor mit gleicher Laenge wie x.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.9 Joelix_Fehler joelix_vektor_ax (Joelix_Vektor x, double alpha)

Berechne x = alpha ∗ x fuer einen Vektor x und einen Skalar alpha.
Parameter

in	*alpha*	Skalar.
in,out	*x*	Ein mit joelix_vektor_init initialisierter Vektor.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.10 Joelix_Fehler joelix_vektor_axpy (Joelix_Vektor y, Joelix_Vektor x, double alpha)

Berechnet y = alpha ∗ x + y fuer Vektoren x und y und einen reellen Skalar alpha.
Parameter

in	*alpha*	Skalar.
in	*x*	Ein mit joelix_vektor_init initialisierter Vektor.
in,out	*y*	Ein mit joelix_vektor_init initialisierter Vektor mit gleicher Laenge wie x.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.11 Joelix_Fehler joelix_vektor_dot (double ∗ produkt, Joelix_Vektor x, Joelix_Vektor y)

Berechnet das Skalarprodukt zweier Vektoren.
Parameter

out	*produkt*	Ein Pointer auf eine initialisierte Double Variable.
in	*x*	Ein mit joelix_vektor_init initialisierter Vektor.
in	*y*	Ein mit joelix_vektor_init initialisierter Vektor mit gleicher Laenge wie x.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.12 Joelix_Fehler joelix_vektor_print (Joelix_Vektor x)

Gebe einen Vektor auf der Konsole aus.

Parameter

`in`	x	Ein mit joelix_vektor_init initialisierter Vektor.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.4.13 Joelix_Fehler joelix_vektor_print_tofile (Joelix_Vektor x, char * filename)

Gebe einen Vektor in eine Datei aus.
Parameter

`in`	x	Ein mit joelix_vektor_init initialisierter Vektor.
`in`	*filename*	Der Name der Outputdatei. Die Datei wird ueberschrieben, falls sie existiert.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.5 include/matrix.h

In der matrix.h Headerdatei werden die Funktionen und Datenstrukturen definiert, um mit Matrizen zu arbeiten.

5.5.1 Joelix_sMatrix

Der Datentyp fuer Matrizen.

5.5.2 Joelix_Fehler joelix_smatrix_init (Joelix_sMatrix * pMatrix, int nzeilen, int nspalten, int nnichtnull)

Initialisiert eine sparse Matrix mit einer gegebenen Anzahl an nicht-null Eintraegen.
Parameter

in	*pMatrix*	Pointer auf die Matrix (vom Typ Joelix_sMatrix) die initialisiert werden soll.
in	*nzeilen*	Die Anzahl an Zeilen der Matrix.
in	*nspalten*	Die Anzahl an Spalten der Matrix.
in	*nnichtnull*	Die Anzahl an nicht-null Eintraegen der Matrix.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.5.3 Joelix_Fehler joelix_smatrix_loeschen (Joelix_sMatrix * pM)

Gibt den Speicher, der von einer Matrix benutzt wird, wieder frei.
Parameter

in,out	*pM*	Pointer auf eine von smatrix_neu erzeugte Matrix. Ist nach Ausführen der Funktion NULL.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.5.4 Joelix_Fehler joelix_smatrix_fuelleZeile (Joelix_sMatrix M, int zeile, int znichtnull, double * werte, int * spalten)

Befuelle eine Zeile einer sparse Matrix mit Eintraegen. Nachdem die Matrix mit joelix_smatrix_init initialisiert wurde, muss diese Funktion fuer jede Zeile mit nicht-null Eintraegen aufgerufen werden. Die Aufrufe muessen in aufsteigender Reihenfolge des Zeilenindex erfolgen.

Parameter

in	*M*	Eine mit joelix_sMatrix_init initialisierte Matrix.
in	*zeile*	Der Index der zu befuellenden Zeile.
in	*znichtnull*	Die Anzahl der nicht-null Eintraege in dieser Zeile.
in	*werte*	Ein Array der Laenge znichtnull mit den Werten der nicht-null Eintraege.
in	*spalten*	Ein Array der Laenge ynichtnull mit den Spaltenindices der nicht-null Eintraege.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode. Warnung: Es wird nicht ueberprueft, ob die Spaltenindices alle innerhalb der zulaessigen Grenzen liegen (0 <= j < Anzahl_Spalten).

5.5.5 Joelix_Fehler joelix_smatrix_aendernneintrag (Joelix_sMatrix M, int zeile, int spalte, double wert)

Ermoeglicht es einen nichtnull Eintrag, der mit joelix_smatrix_fuelleZeile gesetzt wurde, zu veraendern.
Parameter

in	*M*	Eine mit joelix_sMatrix_init initialisierte Matrix.
in	*zeile*	Der Zeilenindex des zu aendernden Eintrags.
in	*spalte*	Der Spaltenindex des zu aendernden Eintrags.
in	*wert*	Der neue Wert.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode. Diese Funktion darf nur aufgerufen werden, wenn vorher ein Aufruf von joelix_smatrix_fuelleZeile geschehen ist, bei deim zeile und spalte mit den Werten hier uebereinstimmen.

5.5.6 int joelix_smatrix_get_spalten (Joelix_sMatrix M)

Fordere die Anzahl der Spalten an.
Parameter

in	*M*	Eine mit joelix_sMatrix_init initialisierte Matrix.

Rückgabe

Anzahl der Spalten oder -1 bei Fehler.

5.5.7 int joelix_smatrix_get_zeilen (Joelix_sMatrix M)

Fordere die Anzahl der Zeilen an.

Parameter

`in`	M	Eine mit joelix_sMatrix_init initialisierte Matrix.

Rückgabe

Anzahl der Zeilen oder -1 bei Fehler.

5.5.8 Joelix_Fehler joelix_smatrix_print (const Joelix_sMatrix M)

Gebe eine sparse matrix auf der Konsole aus.
Parameter

`in`	M	Eine mit joelix_smatrix_neu erstellte Matrix.

Rückgabe

F_ERFOLG bei Erfolg, sonst ein anderer Fehlercode.

5.5.9 Joelix_Fehler joelix_smatvec (Joelix_Vektor b, Joelix_sMatrix M, Joelix_Vektor x)

Berechnet b = Mx als Matrix-Vektor Multiplikation.
Parameter

`in`	M	Eine mit joelix_smatrix_neu erstellte Matrix.
`in`	x	Ein mit joelix_vektor_neu erstellter Vektor. (input)
`in,out`	b	Ein mit joelix_vektor_neu erstellter Vektor. (output)

Rückgabe

b b = Mx wird inplace berechnet. Warnung: b und x muessen verschiedene Vektoren sein.

6 Ein einfaches Beispiel

Wir beschreiben an dieser Stelle, wie ein Programm vernünftig gegen die Bibliothek gelinkt wird und kompiliert werden kann.

6.1 Vektor erstellen

Unser Testprogramm sei wie folgt, quasi ein „Hello World“ für die `joelixblas` Bibliothek.

```
1  #include <vektor.h>
2  #include <joelix_error.h>
3
4  int main () {
5    Joelix_Vektor V;
6
7    /* Initialisiere V als Vektor der Laenge 4 */
8    joelix_vektor_init (&V, 4);
9    if (joelix_fehler_code == F_ERFOLG) {
10     /* Falls das geklappt hat, gebe Vektor aus und
11      * gebe dann den Speicher wieder frei. */
12     joelix_vektor_print (V);
13     joelix_vektor_loeschen (&V);
14   }
15   else {
16     printf ("Es␣gab␣einen␣Fehler:␣%s\n",
17             joelix_fehler_beschreibung (joelix_fehler_code));
18   }
19   return 0;
20 }
```

Wir nehmen an, dass die Quelldatei den Namen `joelix_hwelt.c` hat und in dem Ordner `source` liegt, welcher im selben Ordner wie der Ordner `joelixblas-1.1` liegt. Man kann das Programm natürlich an einem beliebigen anderen Ordner abspeichern und kompilieren, dann muss man nur daran denken, die Dateipfade im Folgenden auch anzupassen.

Um dieses Quelldatei zu kompilieren müssen wir `gcc` mitteilen, dass gegen die `joelixblas` Bibliothek gelinkt werden soll. Dies geschieht mit `-ljoelixblas`. Dazu benötigt `gcc` noch die Information, wo diese Bibliothek zu finden ist. Dies geschieht, wie oben beschrieben, mit dem Befehl `-Llibpfad`. `libpfad` ist der Pfad zu dem Ordner mit der installierten Bibliothek, in unserem Fall also `../joelixblas-1.1/lib/`. Als letztes benötigt `gcc` die Information, an welcher Stelle nach dem Header `<vektor.h>` gesucht werden soll[1]. Dies geschieht mit dem Befehl `-Iincpfad`, wobei `incpfad` der Pfad zu dem Ordner mit den Headerdateien ist. In unserem Fall ist dies `../joelix_install/joelixblas/include`.

Der komplette Kompilierbefehl sieht demnach so aus:

```
1  $ cd source
```

[1] Der aufmerksamen Leserin fällt auf, dass wir hier spitze Klammern '<' und '>' verwenden.

```
$ gcc -Wall -pedantic -o joelix_hwelt joelix_hwelt.c \
      -L../joelixblas-1.1/lib/ \
      -I../joelixblas-1.1/joelixblas/include/ \
      -ljoelixblas
```

Die Backslashes „\“ am Ende der Zeilen gebe dabei an, dass der Befehl in der nächsten Zeile fortgesetzt wird. Wenn man alles in eine Zeile schreibt, kann man sie also weglassen.

Das Programm sollte nun kompiliert sein und der Output sollte wie folgt aussehen:

```
$ ./joelix_hwelt
(0.000000, 0.000000, 0.000000, 0.000000)
```

6.2 Matrix-Vektor-Produkt

Wir diskutieren kurz ein etwas umfangreicheres Beispiel, welches ein Matrix-Vektor-Produkt berechnet. Die Matrix ist hierbei eine Diagonalmatrix mit Eintrag i in Zeile i.

```
#include <stdio.h>
#include <vektor.h>
#include <matrix.h>
#include <joelix_error.h>

void fehlerbehandlung (Joelix_Vektor V, Joelix_Vektor x,
                       Joelix_sMatrix M) {
    printf ("Fehler: %s\n",
            joelix_fehler_beschreibung (joelix_fehler_code));
    if (V != NULL)
      joelix_vektor_loeschen (&V);
    if (x != NULL)
      joelix_vektor_loeschen (&x);
    if (M != NULL)
      joelix_smatrix_loeschen (&M);
}

int main () {
  Joelix_Vektor  V = NULL;
  Joelix_Vektor  x = NULL;
  Joelix_sMatrix M = NULL;
  int            dim = 4;
  int            i;
  int            spalte;
  double         eintrag;

  /* Initialisiere V und x als Vektor der Laenge dim */
  joelix_vektor_init (&V, dim);
  if (joelix_fehler_code != F_ERFOLG) {
    fehlerbehandlung (V, x, M);
    return 0;
  }
  joelix_vektor_init (&x, dim);
```

```
34    if (joelix_fehler_code != F_ERFOLG) {
35      fehlerbehandlung (V, x, M);
36      return 0;
37    }
38    /* Erstelle Matrix mit dim Zeilen, dim Spalten und dim
39       nicht-null Eintraegen. */
40    joelix_smatrix_init (&M, dim, dim, dim);
41
42    if (joelix_fehler_code != F_ERFOLG) {
43      fehlerbehandlung (V, x, M);
44      return 0;
45    }
46
47    /* Fuelle die Matrix und den Vektor */
48    for (i = 0;i < dim;i++) {
49      eintrag = i+1;  /* Speicher Wert i+1 */
50      spalte = i;     /* In Spalte i */
51      joelix_smatrix_fuelleZeile (M, i, 1, &eintrag, &spalte);
52      if (joelix_fehler_code != F_ERFOLG) {
53        fehlerbehandlung (V, x, M);
54        return 0;
55      }
56      joelix_vektor_seti (V, i, 1);
57      if (joelix_fehler_code != F_ERFOLG) {
58        fehlerbehandlung (V, x, M);
59        return 0;
60      }
61    }
62    /* Matrix-Vektor-Produkt */
63    joelix_smatvec (x, M, V);
64    if (joelix_fehler_code != F_ERFOLG) {
65      fehlerbehandlung (V, x, M);
66      return 0;
67    }
68    joelix_vektor_print (x);
69
70    /* Speicher freigeben */
71    joelix_vektor_loeschen (&V);
72    joelix_vektor_loeschen (&x);
73    joelix_smatrix_loeschen (&M);
74    return 0;
75  }
```