

LAP (Linear Assignment Problem)

Программная документация

1 LAP - Linear Assignment Problem / Линейная дискретная оптимизационная задача (задача о назначениях)	1
1.1 1. Brief / Обзор	1
1.2 2. References / Ссылки	1
1.3 3. Dependencies / Зависимости	2
1.4 4. Tests / Тесты	2
1.5 5. Time measurements tables for sparse matrices / Сводные таблицы замеров времени выполнения для разреженных матриц	6
1.6 6. Conclusion / Вывод	6
2 Алфавитный указатель групп	7
2.1 Группы	7
3 Алфавитный указатель пространств имен	9
3.1 Пространства имен	9
4 Алфавитный указатель классов	11
4.1 Классы	11
5 Группы	13
5.1 Spml	13
5.1.1 Подробное описание	13
6 Пространства имен	15
6.1 Пространство имен SPML	15
6.1.1 Подробное описание	15
6.2 Пространство имен SPML::Compare	15
6.2.1 Подробное описание	15
6.2.2 Функции	16
6.3 Пространство имен SPML::LAP	18
6.3.1 Подробное описание	19
6.3.2 Перечисления	19
6.3.3 Функции	19
6.4 Пространство имен SPML::Sparse	28
6.4.1 Подробное описание	29
6.4.2 Функции	29
7 Классы	37
7.1 Структура SPML::Sparse::CKeyCOO	37
7.1.1 Подробное описание	37
7.1.2 Конструктор(ы)	37
7.1.3 Методы	38
7.2 Структура SPML::Sparse::CMatrixCOO	38
7.2.1 Подробное описание	38
7.2.2 Данные класса	38
7.3 Структура SPML::Sparse::CMatrixCSC	39

7.3.1 Подробное описание	39
7.3.2 Данные класса	39
7.4 Структура SPML::Sparse::CMatrixCSR	40
7.4.1 Подробное описание	40
7.4.2 Данные класса	40
Предметный указатель	41

Раздел 1

LAP - Linear Assignment Problem / Линейная дискретная оптимизационная задача (задача о назначениях)

1.1 1. Brief / Обзор

Solving linear assignment problem using / Решение задачи о назначениях методами:

- Jonker-Volgenant-Castanon method (JVC) for dense and sparse (CSR - compressed sparse row) matrices / Метод Джонкера-Волгенанта-Кастаньона для плотных и разреженных матриц в CSR формате
- Mack method / Метод Мака
- Hungarian (Munkres) method / Венгерский алгоритм

1.2 2. References / Ссылки

Papers / Статьи:

- R.Jonker and A.Volgenant A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems Computing 38, 325-340 (1987)
- A.Volgenant Linear and Semi-Assignment Problems: A Core Oriented Approach
- Банди Б. Основы линейного программирования: Пер. с англ. - М.:Радио м связь, 1989, стр 113-123

Sites / Сайты:

- <http://www.assignmentproblems.com/linearAP.htm>
- <https://www.mathworks.com/matlabcentral/fileexchange/26836-lapjv-jonker-volgenant-algorithm-for-linear-assignment-problem-v3-0>

Repositories / Репозитории:

- <https://github.com/yongyanghai/LAPJV-algorithm-c>
- <https://github.com/RcppCore/rcpp-gallery/blob/gh-pages/src/2013-09-24-minimal-assignment.%C2%ABcpp>
- <https://github.com/fuglede/linearassignment>

1.3 3. Dependencies / Зависимости

Armadillo for matrices, Boost for testing / Armadillo для работы с матрицами, Boost для тестирования.

1.4 4. Tests / Тесты

- Comparison of calculation speed on dense and sparse matrices / Сравнение скорости работы на плотных и разреженных матрицах
- Simple assignment problem matrices are provided / Дополнительные тесты на простых матрицах
- test JVC algorithm for looping / Тест алгоритма JVCdense на зацикливание

(Sparsity is ~20% / В разреженной матрице ~20% назначенных ячеек)

Results for time measuring / Результаты замеров скорости работы:

<? <?

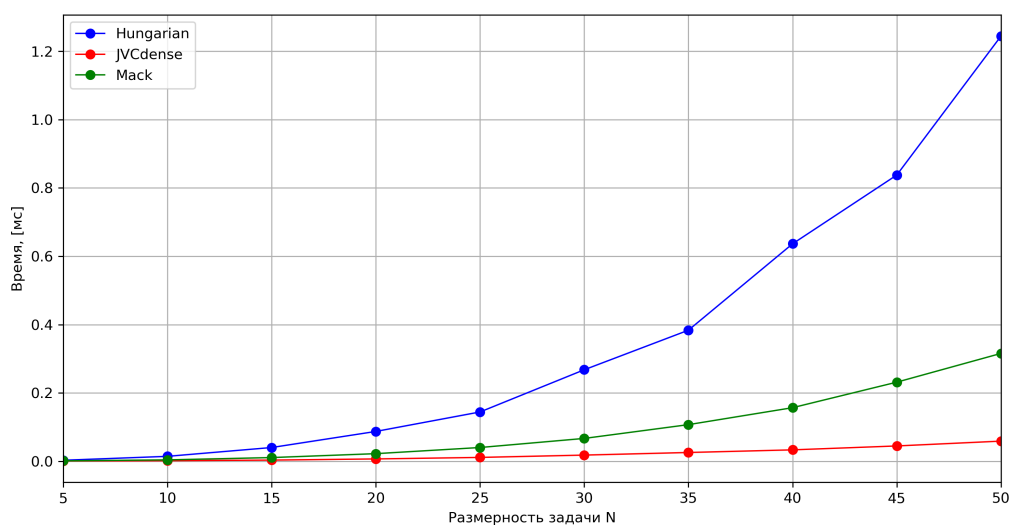


Рис. 1.1 ?>

Fig.1 - Execution time for dense matrices (small dimensions)

Рис.1 - Время выполнения на плотных матрицах (малые размерности)

<? <?

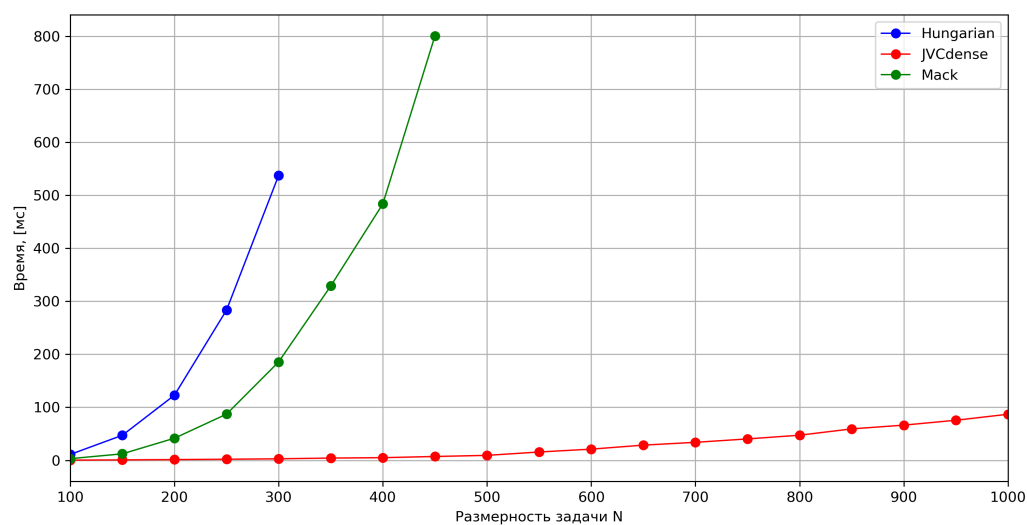


Рис. 1.2 ?>

Fig.2 - Execution time for dense matrices (large dimensions)

Рис.2 - Время выполнения на плотных матрицах (большие размерности)

<? <?

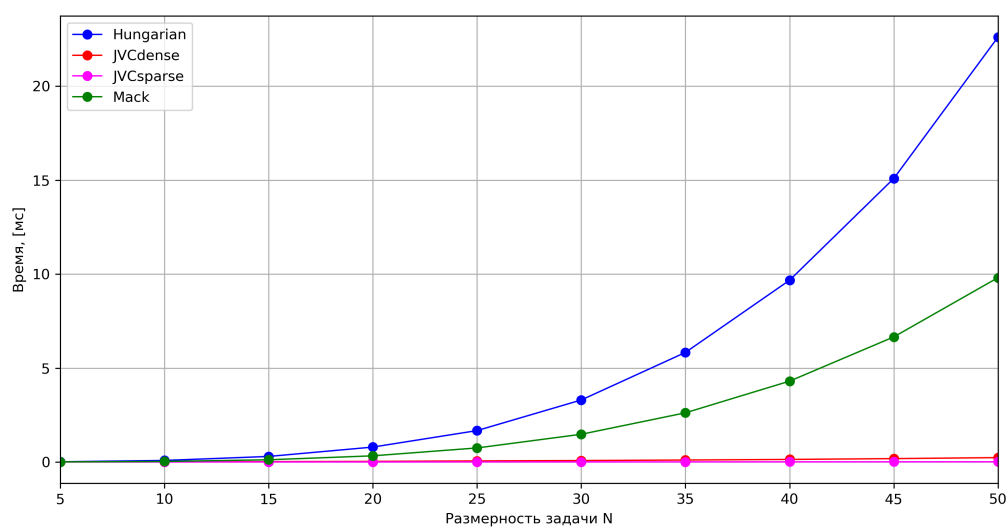


Рис. 1.3 ?>

Fig.3 - Execution time for sparse matrices (small dimensions)

Рис.3 - Время выполнения на разреженных матрицах (малые размерности)

<? <?

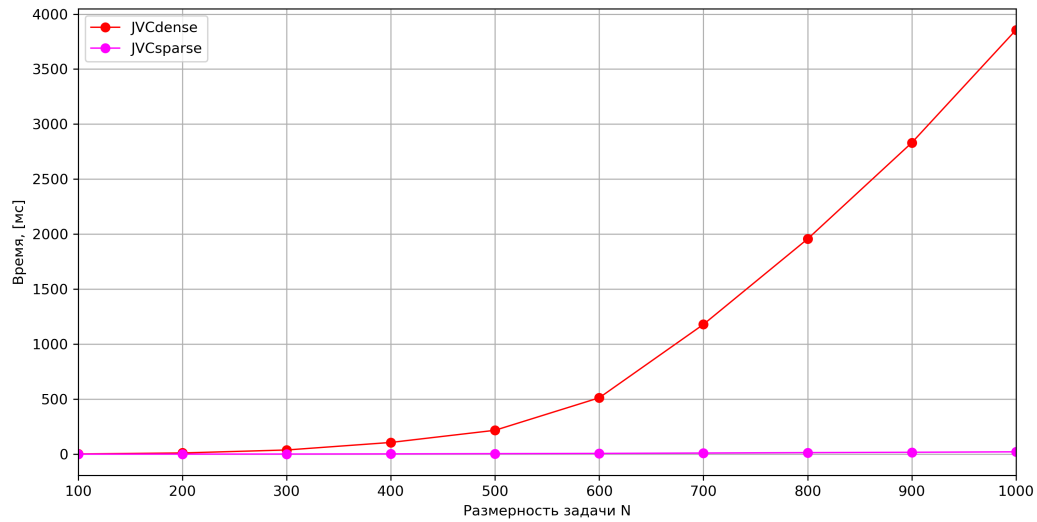


Рис. 1.4 ?>

Fig.4 - Execution time for sparse matrices (large dimensions)

Рис.4 - Время выполнения на разреженных матрицах (большие размерности)

<? <?

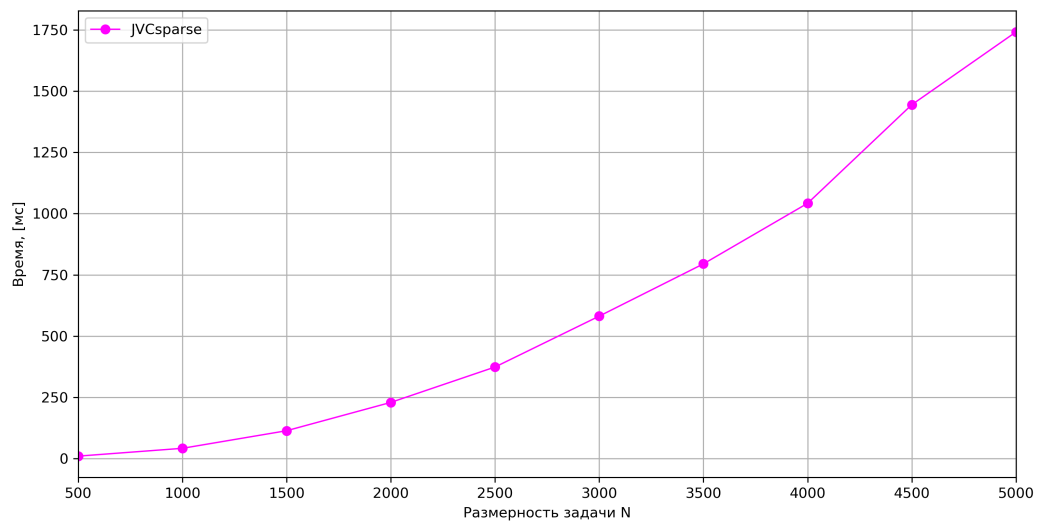


Рис. 1.5 ?>

Fig.5 - Execution time for sparse matrices (large dimensions) for JVCsparse

Рис.5 - Время выполнения на разреженных матрицах (большие размерности) для JVCsparse

Additional graphs pics for methods of sequential extremum for dense and sparse matrices in dense and COOrdinated formats / Дополнительные графики для методов последовательного выбора экстремума для плотных и разреженных матриц в обычном плотном виде и COO-формате.

<? <?

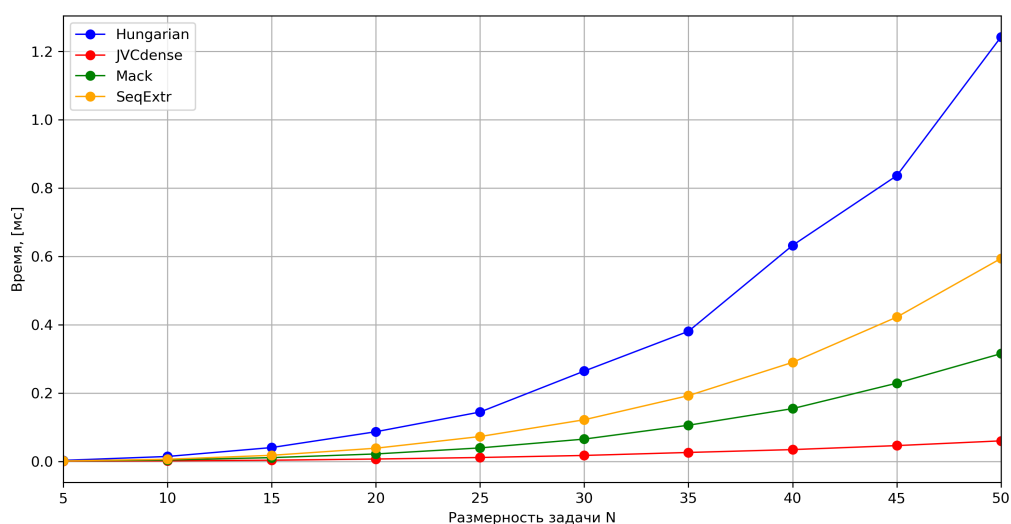


Рис. 1.6 ?>

Fig.6 - Execution time for dense matrices (small dimensions)

Рис.6 - Время выполнения на плотных матрицах (малые размерности)

<? <?

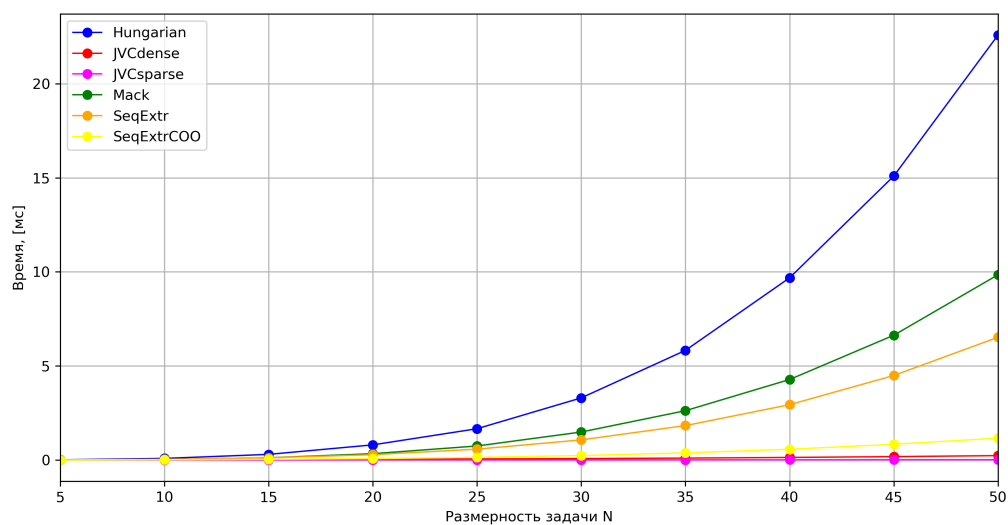


Рис. 1.7 ?>

Fig.7 - Execution time for sparse matrices (small dimensions)

Рис.7 - Время выполнения на разреженных матрицах (малые размерности)

1.5 5. Time measurements tables for sparse matrices / Сводные таблицы замеров времени выполнения для разреженных матриц

Table 1 - Execution time, milliseconds / Таблица 1 - Время выполнения, миллисекунды

N	5	10	25	50	100	150	200	250	500	1000
Hungarian	0.013	0.095	1.931	24.129	253.↵ 358	1772.↵ 925	6687.↵ 237	-	-	-
Mack	0.005	0.037	0.801	10.185	135.↵ 415	627.↵ 683	1815.↵ 623	-	-	-
Seq↵ Extr	0.006	0.035	0.577	6.556	88.152	433.↵ 079	1343.↵ 761	-	-	-
Seq↵ Extr↵ COO	0.003	0.015	0.152	1.216	15.856	83.929	323.↵ 072	-	-	-
JVCdense	0.003	0.009	0.052	0.220	1.400	4.562	10.246	20.598	232.↵ 334	3876.↵ 377
JVCsparse	0.001	0.002	0.005	0.013	0.049	0.132	0.261	0.535	4.481	21.902

Table 2 - Increasing execution time relative to JVCsparse (times) / Таблица 2 - Возрастание времени выполнения относительно метода JVCsparse (разы)

N	5	10	25	50	100	150	200	250	500	1000
Hungarian	10.914	46.775	407.↵ 213	1884.↵ 278	5146.↵ 364	13436.↵ 377	25614.↵ 081	-	-	-
Mack	4.437	17.944	168.↵ 873	795.↵ 364	2750.↵ 645	4756.↵ 991	6954.↵ 369	-	-	-
Seq↵ Extr	4.789	17.179	121.↵ 615	511.↵ 956	1790.↵ 602	3282.↵ 153	5147.↵ 001	-	-	-
Seq↵ Extr↵ COO	2.848	7.351	32.091	94.928	322.↵ 073	636.↵ 071	1237.↵ 459	-	-	-
JVCdense	2.246	4.249	10.926	17.158	28.438	34.577	39.247	38.476	51.850	176.↵ 988

1.6 6. Conclusion / Вывод

JVCsparse is the fastest method from considered (for sparse matrices), cause it works with compact CSR storage and uses fast JVC algorithm. JVCdense is the fastest for dense. Method of sequential extremum is non-optimal and it's usage is not recommended. / JVCsparse самый быстрый метод среди рассмотренных (для разреженных матриц), поскольку работает с матрицами, хранящимися в компактном CSR формате и использует быстрый JVC алгоритм. Для плотных матриц самый быстрый JVCdense. Метод последовательного выбора экстремума неоптимален и его использование не рекомендуется.

Раздел 2

Алфавитный указатель групп

2.1 Группы

Полный список групп.

Spm1 13

Раздел 3

Алфавитный указатель пространств имен

3.1 Пространства имен

Полный список пространств имен.

SPML	Специальная библиотека программных модулей (СБ ПМ)	15
SPML::Compare	Сравнение чисел	15
SPML::LAP	Решение задачи о назначениях	18
SPML::Sparse	Разреженные матрицы	28

Раздел 4

Алфавитный указатель классов

4.1 Классы

Классы с их кратким описанием.

SPML::Sparse::CKeyCOO	
Ключ элемента A_{ij} матрицы A в COO формате (Coordinate list)	37
SPML::Sparse::CMatrixCOO	
Структура хранения матрицы в COO формате (Coordinate list)	38
SPML::Sparse::CMatrixCSC	
Структура хранения матрицы в CSC формате (Compressed Sparse Column Yale format)	39
SPML::Sparse::CMatrixCSR	
Структура хранения матрицы в CSR формате (Compressed Sparse Row Yale format)	40

Раздел 5

Группы

5.1 Spml

Пространства имен

- namespace [SPML](#)

Специальная библиотека программных модулей (СБ ПМ)

5.1.1 Подробное описание

Раздел 6

Пространства имен

6.1 Пространство имен SPML

Специальная библиотека программных модулей (СБ ПМ)

Пространства имен

- namespace [Compare](#)
Сравнение чисел
- namespace [LAP](#)
Решение задачи о назначениях
- namespace [Sparse](#)
Разреженные матрицы

6.1.1 Подробное описание

Специальная библиотека программных модулей (СБ ПМ)

6.2 Пространство имен SPML::Compare

Сравнение чисел

Функции

- bool [AreEqualAbs](#) (float first, float second, const float &eps=EPS_F)
Сравнение двух действительных чисел (по абсолютной разнице)
- bool [AreEqualAbs](#) (double first, double second, const double &eps=EPS_D)
Сравнение двух действительных чисел (по абсолютной разнице)
- bool [AreEqualRel](#) (float first, float second, const float &eps=EPS_REL)
Сравнение двух действительных чисел (по относительной разнице)
- bool [AreEqualRel](#) (double first, double second, const double &eps=EPS_REL)
Сравнение двух действительных чисел (по относительной разнице)
- bool [IsZeroAbs](#) (float value, const float &eps=EPS_F)
Проверка действительного числа на равенство нулю (по абсолютной разнице)
- bool [IsZeroAbs](#) (double value, const double &eps=EPS_D)
Проверка действительного числа на равенство нулю (по абсолютной разнице)

6.2.1 Подробное описание

Сравнение чисел

6.2.2 Функции

6.2.2.1 AreEqualAbs() [1/2]

```
bool SPML::Compare::AreEqualAbs (
    double first,
    double second,
    const double & eps = EPS_D ) [inline]
```

Сравнение двух действительных чисел (по абсолютной разнице)

Возвращает результат: $\text{abs}(\text{first} - \text{second}) < \text{eps}$

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле [compare.h](#) строка 48

6.2.2.2 AreEqualAbs() [2/2]

```
bool SPML::Compare::AreEqualAbs (
    float first,
    float second,
    const float & eps = EPS_F ) [inline]
```

Сравнение двух действительных чисел (по абсолютной разнице)

Возвращает результат: $\text{abs}(\text{first} - \text{second}) < \text{eps}$

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле [compare.h](#) строка 35

6.2.2.3 AreEqualRel() [1/2]

```
bool SPML::Compare::AreEqualRel (
    double first,
    double second,
    const double & eps = EPS_REL ) [inline]
```

Сравнение двух действительных чисел (по относительной разнице)

Возвращает результат: $(\text{abs}((\text{first} - \text{second}) / \text{first}) < \text{eps}) \&\& (\text{abs}((\text{first} - \text{second}) / \text{second}) < \text{eps})$

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- относительная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле [compare.h](#) строка 76

6.2.2.4 AreEqualRel() [2/2]

```
bool SPML::Compare::AreEqualRel (
    float first,
    float second,
    const float & eps = EPS_REL ) [inline]
```

Сравнение двух действительных чисел (по относительной разнице)

Возвращает результат: $(\text{abs}(\text{first} - \text{second}) / \text{first}) < \text{eps}) \&\& (\text{abs}(\text{first} - \text{second}) / \text{second}) < \text{eps})$

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- относительная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле [compare.h](#) строка 62

6.2.2.5 IsZeroAbs() [1/2]

```
bool SPML::Compare::IsZeroAbs (
    double value,
    const double & eps = EPS_D ) [inline]
```

Проверка действительного числа на равенство нулю (по абсолютной разнице)

Возвращает результат: $\text{abs}(\text{value}) < \text{eps}$

Аргументы

in	value	- проверяемое число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле [compare.h](#) строка 102

6.2.2.6 IsZeroAbs() [2/2]

```
bool SPML::Compare::IsZeroAbs (
    float value,
    const float & eps = EPS_F ) [inline]
```

Проверка действительного числа на равенство нулю (по абсолютной разнице)

Возвращает результат: `abs(value) < eps`

Аргументы

in	value	- проверяемое число
in	eps	- абсолютная точность сравнения

Возвращает

`true` - если разница меньше точности, иначе `false`

См. определение в файле [compare.h](#) строка 90

6.3 Пространство имен SPML::LAP

Решение задачи о назначениях

Перечисления

- enum [TSearchParam](#) { [SP_Min](#) , [SP_Max](#) }
- Критерий поиска - минимум/максимум для задачи о назначениях

Функции

- void [SequentialExtremum](#) (const arma::mat &assigncost, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Процедура последовательного поиска экстремума по строкам/столбцам матрицы ценностей
- void [SequentialExtremum](#) (const [Sparse::CMatrixCOO](#) &assigncost, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Процедура последовательного поиска экстремума по строкам/столбцам матрицы ценностей
- void [JVCdense](#) (const arma::mat &assigncost, int dim, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Метод Джонкера-Волгенанта-Кастаньона (Jonker-Volgenant-Castanon) решения задачи о назначениях для плотных матриц
- int [JVCsparse](#) (const std::vector< double > &cc, const std::vector< int > &kk, const std::vector< int > &first, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Метод Джонкера-Волгенанта-Кастаньона (Jonker-Volgenant-Castanon) решения задачи о назначениях для разреженных матриц
- int [JVCsparse](#) (const [Sparse::CMatrixCSR](#) &csr, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Метод Джонкера-Волгенанта-Кастаньона (Jonker-Volgenant-Castanon) решения задачи о назначениях для разреженных матриц
- void [Mack](#) (const arma::mat &assigncost, int dim, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Метод Мака решения задачи о назначениях
- void [Hungarian](#) (const arma::mat &assigncost, int dim, [TSearchParam](#) sp, double maxcost, double resolution, arma::ivec &rowsol, double &lapcost)
Венгерский метод решения задачи о назначениях (Метод Мункреса)
- void [hungarian_step_1](#) (unsigned int &step, arma::mat &cost, const unsigned int &N)

- void [hungarian_step_2](#) (unsigned int &step, const arma::mat &cost, arma::umat &indM, arma::ivec &rcov, arma::ivec &ccov, const unsigned int &N)
- void [hungarian_step_3](#) (unsigned int &step, const arma::umat &indM, arma::ivec &ccov, const unsigned int &N)
- void [hungarian_find_noncovered_zero](#) (int &row, int &col, const arma::mat &cost, const arma::ivec &rcov, const arma::ivec &ccov, const unsigned int &N)
- bool [hungarian_star_in_row](#) (int &row, const arma::umat &indM, const unsigned int &N)
- void [hungarian_find_star_in_row](#) (const int &row, int &col, const arma::umat &indM, const unsigned int &N)
- void [hungarian_step_4](#) (unsigned int &step, const arma::mat &cost, arma::umat &indM, arma::ivec &rcov, arma::ivec &ccov, int &rpath_0, int &cpath_0, const unsigned int &N)
- void [hungarian_find_star_in_col](#) (const int &col, int &row, const arma::umat &indM, const unsigned int &N)
- void [hungarian_find_prime_in_row](#) (const int &row, int &col, const arma::umat &indM, const unsigned int &N)
- void [hungarian_augment_path](#) (const int &path_count, arma::umat &indM, const arma::imat &path)
- void [hungarian_clear_covers](#) (arma::ivec &rcov, arma::ivec &ccov)
- void [hungarian_erase_primes](#) (arma::umat &indM, const unsigned int &N)
- void [hungarian_step_5](#) (unsigned int &step, arma::umat &indM, arma::ivec &rcov, arma::ivec &ccov, arma::imat &path, int &rpath_0, int &cpath_0, const unsigned int &N)
- void [hungarian_find_smallest](#) (double &minval, const arma::mat &cost, const arma::ivec &rcov, const arma::ivec &ccov, const unsigned int &N)
- void [hungarian_step_6](#) (unsigned int &step, arma::mat &cost, const arma::ivec &rcov, const arma::ivec &ccov, const unsigned int &N)
- void [updateDual](#) (int nc, arma::vec &d, arma::vec &v, arma::ivec &todo, int last, double min_)
- void [updateAssignments](#) (arma::ivec &lab, arma::ivec &y, arma::ivec &x, int j, int i0)
- int [solveForOneL](#) (std::vector< double > &cc_, const std::vector< int > &kk, const std::vector< int > &first, int l, int nc, arma::vec &d, arma::ivec &ok, arma::ivec &free, arma::vec &v, arma::ivec &lab, arma::ivec &todo, arma::ivec &y, arma::ivec &x, int td1, double resolution, double maxcost, bool &fail)

6.3.1 Подробное описание

Решение задачи о назначениях

6.3.2 Перечисления

6.3.2.1 TSearchParam

enum [SPML::LAP::TSearchParam](#)

Критерий поиска - минимум/максимум для задачи о назначениях

Элементы перечислений

SP_Min	Поиск минимума
SP_Max	Поиск максимума

См. определение в файле [lap.h](#) строка [32](#)

6.3.3 Функции

6.3.3.1 Hungarian()

```
void SPML::LAP::Hungarian (
    const arma::mat & assigncost,
    int dim,
    TSearchParam sp,
    double maxcost,
    double resolution,
    arma::ivec & rowsol,
    double & lapcost )
```

Венгерский метод решения задачи о назначениях (Метод Мункреса)

Источник: <https://github.com/RcppCore/rcpp-gallery/blob/gh-pages/src/2013-09-24-minimal-assignment.cpp>

Аргументы

in	assigncost	- квадратная матрица ценности, размер [dim,dim]
in	dim	- порядок квадратной матрицы ценности и размерность результата res соответственно
in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

At last, we must create a function that enables us to jump around the different steps of the algorithm. The following code shows the main function of the algorithm. It defines also the important variables to be passed to the different steps.

См. определение в файле [lap_hungarian.cpp](#) строка 384

6.3.3.2 hungarian_augment_path()

```
void SPML::LAP::hungarian_augment_path (
    const int & path_count,
    arma::umat & indM,
    const arma::imat & path )
```

In addition we need a function to augment the path, one to clear the covers from rows and one to erase the primed zeros from the indicator matrix indM.

См. определение в файле [lap_hungarian.cpp](#) строка 261

6.3.3.3 hungarian_clear_covers()

```
void SPML::LAP::hungarian_clear_covers (
    arma::ivec & rcov,
    arma::ivec & ccov )
```

См. определение в файле [lap_hungarian.cpp](#) строка 273

6.3.3.4 hungarian_erase_primes()

```
void SPML::LAP::hungarian_erase_primes (
    arma::umat & indM,
    const unsigned int & N )
```

См. определение в файле [lap_hungarian.cpp](#) строка 279

6.3.3.5 `hungarian_find_noncovered_zero()`

```
void SPML::LAP::hungarian_find_noncovered_zero (
    int & row,
    int & col,
    const arma::mat & cost,
    const arma::ivec & rcov,
    const arma::ivec & ccov,
    const unsigned int & N )
```

We cover a column by looking for 1s in the indicator matrix `indM` (See step 2 for assuring that these are indeed only starred zeros).

Step 4 finds noncovered zeros and primes them. If there are zeros in a row and none of them is starred, prime them. For this task we program a helper function to keep the code more readable and reusable. The helper function searches for noncovered zeros.

См. определение в файле [lap_hungarian.cpp](#) строка 110

6.3.3.6 `hungarian_find_prime_in_row()`

```
void SPML::LAP::hungarian_find_prime_in_row (
    const int & row,
    int & col,
    const arma::umat & indM,
    const unsigned int & N )
```

Then we need a function to find a primed zero in a row. Note, that these tasks are easily performed by searching the indicator matrix `indM`.

См. определение в файле [lap_hungarian.cpp](#) строка 247

6.3.3.7 `hungarian_find_smallest()`

```
void SPML::LAP::hungarian_find_smallest (
    double & minval,
    const arma::mat & cost,
    const arma::ivec & rcov,
    const arma::ivec & ccov,
    const unsigned int & N )
```

Recall, if step 4 was successful in uncovering all columns and covering all rows with a primed zero, it then calls step 6. Step 6 takes the cover vectors `rcov` and `ccov` and looks in the uncovered region of the cost matrix for the smallest value. It then subtracts this value from each element in an uncovered column and adds it to each element in a covered row. After this transformation, the algorithm starts again at step 4. Our last helper function searches for the smallest value in the uncovered region of the cost matrix.

См. определение в файле [lap_hungarian.cpp](#) строка 342

6.3.3.8 `hungarian_find_star_in_col()`

```
void SPML::LAP::hungarian_find_star_in_col (
    const int & col,
    int & row,
    const arma::umat & indM,
    const unsigned int & N )
```

Notice the `rpath_0` and `cpath_0` variables. These integer variables store the first vertex for an augmenting path in step 5. If zeros could be primed we go further to step 5.

Step 5 constructs a path beginning at an uncovered primed zero (this is actually graph theory - alternating and augmenting paths) and alternating between starred and primed zeros. This path is continued until a primed zero with no starred zero in its column is found. Then, all starred zeros in this path are unstarred and all primed zeros are starred. All primes in the indicator matrix are erased and all rows are uncovered. Then return to step 3 to cover again columns.

Step 5 needs several helper functions. First, we need a function to find starred zeros in columns.

См. определение в файле [lap_hungarian.cpp](#) строка 232

6.3.3.9 `hungarian_find_star_in_row()`

```
void SPML::LAP::hungarian_find_star_in_row (
    const int & row,
    int & col,
    const arma::umat & indM,
    const unsigned int & N )
```

См. определение в файле [lap_hungarian.cpp](#) строка 165

6.3.3.10 `hungarian_star_in_row()`

```
bool SPML::LAP::hungarian_star_in_row (
    int & row,
    const arma::umat & indM,
    const unsigned int & N )
```

We can detect noncovered zeros by checking if the cost matrix contains at row r and column c a zero and row and column are not covered yet, i.e. $rcov(r) == 0$, $ccov(c) == 0$. This loop breaks, if we have found our first uncovered zero or no uncovered zero at all.

In step 4, if no uncovered zero is found we go to step 6. If instead an uncovered zero has been found, we set the indicator matrix at its position to 2. We then have to search for a starred zero in the row with the uncovered zero, uncover the column with the starred zero and cover the row with the starred zero. To indicate a starred zero in a row and to find it we create again two helper functions.

См. определение в файле [lap_hungarian.cpp](#) строка 153

6.3.3.11 `hungarian_step_1()`

```
void SPML::LAP::hungarian_step_1 (
    unsigned int & step,
    arma::mat & cost,
    const unsigned int & N )
```

См. определение в файле [lap_hungarian.cpp](#) строка 18

6.3.3.12 `hungarian_step_2()`

```
void SPML::LAP::hungarian_step_2 (
    unsigned int & step,
    const arma::mat & cost,
    arma::umat & indM,
    arma::ivec & rcov,
    arma::ivec & ccov,
    const unsigned int & N )
```

Note, that we use references for all function arguments. As we have to switch between the steps of the algorithm continuously, we always must be able to determine which step should be chosen next. Therefore we give a mutable unsigned integer step as an argument to each step function of the algorithm.

Inside the function we can easily access a whole row by Armadillo's `row()` method for matrices. In the second step, we then search for a zero in the modified cost matrix of step one.

См. определение в файле [lap_hungarian.cpp](#) строка 39

6.3.3.13 `hungarian_step_3()`

```
void SPML::LAP::hungarian_step_3 (
    unsigned int & step,
    const arma::umat & indM,
```

```
arma::ivec & ccov,
const unsigned int & N )
```

Only the first zero in a row is taken. Then, the indicator matrix indM indicates this zero by setting the corresponding element at (r, c) to 1. A unique zero - the only or first one in a column and row - is called starred zero. In step 2 we find such a starred zero.

Note, that we use here Armadillo's element access via the method at(), which makes no bound checks and improves performance.

Note Bene: This code is thoroughly debugged - never do this for fresh written code!

In step 3 we cover each column with a starred zero. If already N columns are covered all starred zeros describe a complete assignment - so, go to step 7 and finish. Otherwise go to step 4.

См. определение в файле [lap_hungarian.cpp](#) строка 77

6.3.3.14 hungarian_step_4()

```
void SPML::LAP::hungarian_step_4 (
    unsigned int & step,
    const arma::mat & cost,
    arma::umat & indM,
    arma::ivec & rcov,
    arma::ivec & ccov,
    int & rpath_0,
    int & cpath_0,
    const unsigned int & N )
```

We know that starred zeros are indicated by the indicator matrix containing an element equal to 1. Now, step 4.

См. определение в файле [lap_hungarian.cpp](#) строка 180

6.3.3.15 hungarian_step_5()

```
void SPML::LAP::hungarian_step_5 (
    unsigned int & step,
    arma::umat & indM,
    arma::ivec & rcov,
    arma::ivec & ccov,
    arma::imat & path,
    int & rpath_0,
    int & cpath_0,
    const unsigned int & N )
```

The function to augment the path gets an integer matrix path of dimension $2 * N \times 2$. In it all vertices between rows and columns are stored row-wise. Now, we can set the complete step 5:

См. определение в файле [lap_hungarian.cpp](#) строка 296

6.3.3.16 hungarian_step_6()

```
void SPML::LAP::hungarian_step_6 (
    unsigned int & step,
    arma::mat & cost,
    const arma::ivec & rcov,
    const arma::ivec & ccov,
    const unsigned int & N )
```

Step 6 looks as follows:

См. определение в файле [lap_hungarian.cpp](#) строка 359

6.3.3.17 JVCdense()

```
void SPML::LAP::JVCdense (
```

```

const arma::mat & assigncost,
int dim,
TSearchParam sp,
double maxcost,
double resolution,
arma::ivec & rowsol,
double & lapcost )

```

Метод Джонкера-Волгенанта-Кастаньона (Jonker-Volgenant-Castanon) решения задачи о назначениях для плотных матриц

Источники: 1) "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing 38, 325-340, 1987 by R. Jonker and A. Volgenant, University of Amsterdam. 2)

<https://github.com/yongyanghz/LAPJV-algorithm-c> 3) <https://www.mathworks.com/matlabcentral/fileexchange/26836-lapjv-jonker-volgenant-algorithm-for-linear-assignment-problem-v3-0>

Прим.

- Оригинальный код R. Jonker and A. Volgenant [1] для целых чисел адаптирован под вещественные

- Метод подразделен на 4 процедуры в соответствии с модификацией Castanon:
 - COLUMN REDUCTION
 - REDUCTION TRANSFER
 - AUGMENTING ROW REDUCTION - аукцион
 - AUGMENT SOLUTION FOR EACH FREE ROW на основе алгоритма Dijkstra
- Правки из [3] касающиеся точности сравнения вещественных чисел

Аргументы

in	assigncost	- квадратная матрица ценности, размер [dim,dim]
in	dim	- порядок квадратной матрицы ценности и размерность результата res соответственно
in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

См. определение в файле [lap_jvc_dense.cpp](#) строка 18

6.3.3.18 JVCsparse() [1/2]

```

int SPML::LAP::JVCsparse (
const Sparse::CMatrixCSR & csr,
TSearchParam sp,
double maxcost,
double resolution,
arma::ivec & rowsol,
double & lapcost )

```

Метод Джонкера-Волгенанта-Кастаньона (Jonker-Volgenant-Castanon) решения задачи о назначениях для разреженных матриц

Аргументы

in	csr	- матрица в CSR формате (Compressed Sparse Row Yale format)
----	-----	---

Аргументы

in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

Возвращает

0 - ОК, 1 - fail

См. определение в файле [lap_jvc_sparse.cpp](#) строка 388

6.3.3.19 JVCsparse() [2/2]

```
int SPML::LAP::JVCsparse (
    const std::vector< double > & cc,
    const std::vector< int > & kk,
    const std::vector< int > & first,
    TSearchParam sp,
    double maxcost,
    double resolution,
    arma::ivec & rowsol,
    double & lapcost )
```

Метод Джонкера-Волгенанта-Кастаньона (Jonker-Volgenant-Castanon) решения задачи о назначениях для разреженных матриц

Аргументы

in	cc	- вектор ненулевых элементов матрицы
in	kk	- вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов
in	first	- вектор начальных смещений в векторе cc
in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

Возвращает

0 - ОК, 1 - fail

См. определение в файле [lap_jvc_sparse.cpp](#) строка 147

6.3.3.20 Mack()

```
void SPML::LAP::Mack (
    const arma::mat & assigncost,
    int dim,
```

```

TSearchParam sp,
double maxcost,
double resolution,
arma::ivec & rowsol,
double & lapcost )

```

Метод Мака решения задачи о назначениях

Источник: Банди Б. Основы линейного программирования: Пер. с англ. - М.: Радио м связь, 1989, стр 113-123

Аргументы

in	assigncost	- квадратная матрица ценности, размер [dim,dim]
in	dim	- порядок квадратной матрицы ценности и размерность результата res соответственно
in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

См. определение в файле [lap_mack.cpp](#) строка 18

6.3.3.21 SequentialExtremum() [1/2]

```

void SPML::LAP::SequentialExtremum (
    const arma::mat & assigncost,
    TSearchParam sp,
    double maxcost,
    double resolution,
    arma::ivec & rowsol,
    double & lapcost )

```

Процедура последовательного поиска экстремума по строкам/столбцам матрицы ценностей

Неоптимальная процедура, сумма назначений будет неоптимальна, поскольку строки/столбцы с найденным экстремумом исключаются из анализа.

Аргументы

in	assigncost	- матрица ценности
in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

См. определение в файле [lap_seqextr.cpp](#) строка 18

6.3.3.22 SequentialExtremum() [2/2]

```

void SPML::LAP::SequentialExtremum (
    const Sparse::CMatrixCOO & assigncost,
    TSearchParam sp,

```

```
double maxcost,
double resolution,
arma::ivec & rowsol,
double & lapcost )
```

Процедура последовательного поиска экстремума по строкам/столбцам матрицы ценностей
Неоптимальная процедура, сумма назначений будет неоптимальна, поскольку строки/столбцы с найденным экстремумом исключаются из анализа.

Аргументы

in	assigncost	- матрица ценности в сжатом COO формате
in	sp	- критерий поиска (минимум/максимум)
in	maxcost	- модуль максимального элемента матрицы assigncost
in	resolution	- точность для сравнения двух вещественных чисел
out	rowsol	- результат задачи о назначениях, размерность [dim] (индекс макс/мин элемента в i-ой строке) rowsol[i] = j --> в i-ой строке выбран j-ый элемент
out	lapcost	- сумма назначенных элементов матрицы ценности assigncost

См. определение в файле [lap_seqextr.cpp](#) строка 84

6.3.3.23 solveForOneL()

```
int SPML::LAP::solveForOneL (
    std::vector< double > & cc_,
    const std::vector< int > & kk,
    const std::vector< int > & first,
    int l,
    int nc,
    arma::vec & d,
    arma::ivec & ok,
    arma::ivec & free,
    arma::vec & v,
    arma::ivec & lab,
    arma::ivec & todo,
    arma::ivec & y,
    arma::ivec & x,
    int td1,
    double resolution,
    double maxcost,
    bool & fail )
```

См. определение в файле [lap_jvc_sparse.cpp](#) строка 44

6.3.3.24 updateAssignments()

```
void SPML::LAP::updateAssignments (
    arma::ivec & lab,
    arma::ivec & y,
    arma::ivec & x,
    int j,
    int i0 )
```

См. определение в файле [lap_jvc_sparse.cpp](#) строка 27

6.3.3.25 updateDual()

```
void SPML::LAP::updateDual (
```

```

int nc,
arma::vec & d,
arma::vec & v,
arma::ivec & todo,
int last,
double min_ )

```

См. определение в файле `lap_jvc_sparse.cpp` строка 18

6.4 Пространство имен SPML::Sparse

Разреженные матрицы

Классы

- struct `CKeyCOO`
Ключ элемента A_{ij} матрицы A в COO формате (Coordinate list)
- struct `CMatrixCOO`
Структура хранения матрицы в COO формате (Coordinate list)
- struct `CMatrixCSC`
Структура хранения матрицы в CSC формате (Compressed [Sparse](#) Column Yale format)
- struct `CMatrixCSR`
Структура хранения матрицы в CSR формате (Compressed [Sparse](#) Row Yale format)

Функции

- void `MatrixDenseToCOO` (const arma::mat &A, std::vector< double > &coo_val, std::vector< int > &coo_row, std::vector< int > &coo_col)
Преобразование плотной матрицы в COO формат (Coordinated list)
- void `MatrixDenseToCOO` (const arma::mat &A, `CMatrixCOO` &COO)
Преобразование плотной матрицы в COO формат (Coordinated list)
- void `MatrixCOOtoDense` (const std::vector< double > &coo_val, const std::vector< int > &coo_row, const std::vector< int > &coo_col, arma::mat &A)
Преобразование матрицы из COO формата (Coordinated list) в плотную матрицу
- void `MatrixCOOtoDense` (const `CMatrixCOO` &COO, arma::mat &A)
Преобразование матрицы из COO формата (Coordinated list) в плотную матрицу
- void `MatrixDenseToCSR` (const arma::mat &A, std::vector< double > &csr_val, std::vector< int > &csr_kk, std::vector< int > &csr_first)
Преобразование плотной матрицы в CSR формат (Compressed [Sparse](#) Row Yale format)
- void `MatrixDenseToCSR` (const arma::mat &A, `CMatrixCSR` &CSR)
Преобразование плотной матрицы в CSR формат (Compressed [Sparse](#) Row Yale format)
- void `MatrixCSRtoDense` (const std::vector< double > &csr_val, const std::vector< int > &csr_kk, const std::vector< int > &csr_first, arma::mat &A)
Преобразование матрицы из CSR формата (Compressed [Sparse](#) Row Yale format) в плотную матрицу
- void `MatrixCSRtoDense` (const `CMatrixCSR` &CSR, arma::mat &A)
Преобразование матрицы из CSR формата (Compressed [Sparse](#) Row Yale format) в плотную матрицу
- void `MatrixDenseToCSC` (const arma::mat &A, std::vector< double > &csc_val, std::vector< int > &csc_kk, std::vector< int > &csc_first)
Преобразование плотной матрицы в CSC формат (Compressed [Sparse](#) Column Yale format)
- void `MatrixDenseToCSC` (const arma::mat &A, `CMatrixCSC` &CSC)
Преобразование плотной матрицы в CSC формат (Compressed [Sparse](#) Column Yale format)
- void `MatrixCSCtoDense` (const std::vector< double > &csc_val, const std::vector< int > &csc_kk, const std::vector< int > &csc_first, arma::mat &A)
Преобразование матрицы из CSC формата (Compressed [Sparse](#) Column Yale format) в плотную матрицу

- void [MatrixCSCtoDense](#) (const [CMatrixCSC](#) &CSC, arma::mat &A)
Преобразование матрицы из CSC формата (Compressed [Sparse](#) Column Yale format) в плотную матрицу
- void [MatrixCOOtoCSR](#) (const std::vector< double > &coo_val, const std::vector< int > &coo_row, const std::vector< int > &coo_col, std::vector< double > &csr_val, std::vector< int > &csr_kk, std::vector< int > &csr_first, bool sorted=false)
Преобразование матрицы в COO формате (Coordinate list) в CSR формат (Compressed [Sparse](#) Row Yale format)
- void [MatrixCOOtoCSR](#) (const [CMatrixCOO](#) &COO, [CMatrixCSR](#) &CSR, bool sorted=false)
Преобразование матрицы в COO формате (Coordinate list) в CSR формат (Compressed [Sparse](#) Row Yale format)
- void [MatrixCOOtoCSC](#) (const std::vector< double > &coo_val, const std::vector< int > &coo_row, const std::vector< int > &coo_col, std::vector< double > &csc_val, std::vector< int > &csc_kk, std::vector< int > &csc_first, bool sorted=false)
Преобразование матрицы в COO формате (Coordinate list) в CSC формат (Compressed [Sparse](#) Column Yale format)
- void [MatrixCOOtoCSC](#) (const [CMatrixCOO](#) &COO, [CMatrixCSC](#) &CSC, bool sorted=false)
Преобразование матрицы в COO формате (Coordinate list) в CSC формат (Compressed [Sparse](#) Column Yale format)

6.4.1 Подробное описание

Разреженные матрицы

Решение задачи о назначениях

6.4.2 Функции

6.4.2.1 MatrixCOOtoCSC() [1/2]

```
void SPML::Sparse::MatrixCOOtoCSC (
    const CMatrixCOO & COO,
    CMatrixCSC & CSC,
    bool sorted = false )
```

Преобразование матрицы в COO формате (Coordinate list) в CSC формат (Compressed [Sparse](#) Column Yale format)

Аргументы

in	COO	- матрица в COO формате (Coordinate list)
out	CSC	- матрица в CSR формате (Compressed Sparse Column Yale format)
in	sorted	- признак отсортированности COO входа по столбцам (даёт ускорение в ~2 раза)

См. определение в файле [sparse.cpp](#) строка [328](#)

6.4.2.2 MatrixCOOtoCSC() [2/2]

```
void SPML::Sparse::MatrixCOOtoCSC (
    const std::vector< double > & coo_val,
    const std::vector< int > & coo_row,
    const std::vector< int > & coo_col,
    std::vector< double > & csc_val,
    std::vector< int > & csc_kk,
    std::vector< int > & csc_first,
    bool sorted = false )
```

Преобразование матрицы в COO формате (Coordinate list) в CSC формат (Compressed [Sparse](#) Column Yale format)

Аргументы

in	coo_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	coo_row	- индексы строк ненулевых элементов, размер nnz
in	coo_col	- индексы столбцов ненулевых элементов, размер nnz
in	csc_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	csc_kk	- вектор индексов строк ненулевых элементов, размер равен количеству ненулевых элементов nnz
in	csc_first	- вектор начальных смещений в векторе CSR, размер m+1
in	sorted	- признак отсортированности COO входа по столбцам (даёт ускорение в ~2 раза)

См. определение в файле [sparse.cpp](#) строка 256

6.4.2.3 MatrixCOOtoCSR() [1/2]

```
void SPML::Sparse::MatrixCOOtoCSR (
    const CMatrixCOO & COO,
    CMatrixCSR & CSR,
    bool sorted = false )
```

Преобразование матрицы в COO формате (Coordinate list) в CSR формат (Compressed [Sparse](#) Row Yale format)

Аргументы

in	COO	- матрица в COO формате (Coordinate list)
out	CSR	- матрица в CSR формате (Compressed Sparse Row Yale format)
in	sorted	- признак отсортированности COO входа по строкам (даёт ускорение в ~2 раза)

См. определение в файле [sparse.cpp](#) строка 251

6.4.2.4 MatrixCOOtoCSR() [2/2]

```
void SPML::Sparse::MatrixCOOtoCSR (
    const std::vector< double > & coo_val,
    const std::vector< int > & coo_row,
    const std::vector< int > & coo_col,
    std::vector< double > & csr_val,
    std::vector< int > & csr_kk,
    std::vector< int > & csr_first,
    bool sorted = false )
```

Преобразование матрицы в COO формате (Coordinate list) в CSR формат (Compressed [Sparse](#) Row Yale format)

Аргументы

in	coo_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	coo_row	- индексы строк ненулевых элементов, размер nnz

Аргументы

in	coo_col	- индексы столбцов ненулевых элементов, размер nnz
in	csr_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	csr_kk	- вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов nnz
in	csr_first	- вектор начальных смещений в векторе CSR, размер n+1
in	sorted	- признак отсортированности COO входа по строкам (даёт ускорение в ~2 раза)

См. определение в файле [sparse.cpp](#) строка 179

6.4.2.5 MatrixCOOtoDense() [1/2]

```
void SPML::Sparse::MatrixCOOtoDense (
    const CMatrixCOO & COO,
    arma::mat & A )
```

Преобразование матрицы из COO формата (Coordinated list) в плотную матрицу

Аргументы

in	COO	- структура хранения матрицы в COO формате (Coordinate list)
out	A	- плотная матрица, размер [n,m] (n - число строк, m - число столбцов)

См. определение в файле [sparse.cpp](#) строка 60

6.4.2.6 MatrixCOOtoDense() [2/2]

```
void SPML::Sparse::MatrixCOOtoDense (
    const std::vector< double > & coo_val,
    const std::vector< int > & coo_row,
    const std::vector< int > & coo_col,
    arma::mat & A )
```

Преобразование матрицы из COO формата (Coordinated list) в плотную матрицу

Аргументы

in	coo_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	coo_row	- индексы строк ненулевых элементов, размер nnz
in	coo_col	- индексы столбцов ненулевых элементов, размер nnz
out	A	- плотная матрица, размер [n,m] (n - число строк, m - число столбцов)

См. определение в файле [sparse.cpp](#) строка 48

6.4.2.7 MatrixCSCtoDense() [1/2]

```
void SPML::Sparse::MatrixCSCtoDense (
    const CMatrixCSC & CSC,
    arma::mat & A )
```

Преобразование матрицы из CSC формата (Compressed [Sparse](#) Column Yale format) в плотную матрицу

Аргументы

in	CSC	- структура хранения матрицы в CSC формате (Compressed Sparse Column Yale format)
out	A	- плотная матрица, размер [n,m] (n - число строк, m - число столбцов)

См. определение в файле [sparse.cpp](#) строка 172

6.4.2.8 MatrixCSCtoDense() [2/2]

```
void SPML::Sparse::MatrixCSCtoDense (
    const std::vector< double > & csc_val,
    const std::vector< int > & csc_kk,
    const std::vector< int > & csc_first,
    arma::mat & A )
```

Преобразование матрицы из CSC формата (Compressed [Sparse](#) Column Yale format) в плотную матрицу

Аргументы

in	csc_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	csc_kk	- вектор индексов строк ненулевых элементов, размер равен количеству ненулевых элементов nnz
in	csc_first	- вектор начальных смещений в векторе CSR, размер m+1
out	A	- плотная матрица, размер [n,m] (n - число строк, m - число столбцов)

См. определение в файле [sparse.cpp](#) строка 156

6.4.2.9 MatrixCSRtoDense() [1/2]

```
void SPML::Sparse::MatrixCSRtoDense (
    const CMatrixCSR & CSR,
    arma::mat & A )
```

Преобразование матрицы из CSR формата (Compressed [Sparse](#) Row Yale format) в плотную матрицу

Аргументы

in	CSR	- структура хранения матрицы в CSR формате (Compressed Sparse Row Yale format)
out	A	- плотная матрица, размер [n,m] (n - число строк, m - число столбцов)

См. определение в файле [sparse.cpp](#) строка 116

6.4.2.10 MatrixCSRtoDense() [2/2]

```
void SPML::Sparse::MatrixCSRtoDense (
    const std::vector< double > & csr_val,
    const std::vector< int > & csr_kk,
    const std::vector< int > & csr_first,
    arma::mat & A )
```

Преобразование матрицы из CSR формата (Compressed [Sparse](#) Row Yale format) в плотную матрицу

Аргументы

in	csr_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
in	csr_kk	- вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов nnz
in	csr_first	- вектор начальных смещений в векторе CSR, размер n+1
out	A	- плотная матрица, размер [n,m] (n - число строк, m - число столбцов)

См. определение в файле [sparse.cpp](#) строка 100

6.4.2.11 MatrixDenseToCOO() [1/2]

```
void SPML::Sparse::MatrixDenseToCOO (
    const arma::mat & A,
    CMatrixCOO & COO )
```

Преобразование плотной матрицы в COO формат (Coordinated list)

Аргументы

in	A	- исходная матрица, размер [n,m] (n - число строк, m - число столбцов)
out	COO	- структура хранения матрицы в COO формате (Coordinate list)

См. определение в файле [sparse.cpp](#) строка 43

6.4.2.12 MatrixDenseToCOO() [2/2]

```
void SPML::Sparse::MatrixDenseToCOO (
    const arma::mat & A,
    std::vector< double > & coo_val,
    std::vector< int > & coo_row,
    std::vector< int > & coo_col )
```

Преобразование плотной матрицы в COO формат (Coordinated list)

Аргументы

in	A	- исходная матрица, размер [n,m] (n - число строк, m - число столбцов)
out	coo_val	- вектор ненулевых элементов матрицы A, размер nnz
out	coo_row	- индексы строк ненулевых элементов, размер nnz
out	coo_col	- индексы столбцов ненулевых элементов, размер nnz

См. определение в файле [sparse.cpp](#) строка 20

6.4.2.13 MatrixDenseToCSC() [1/2]

```
void SPML::Sparse::MatrixDenseToCSC (
    const arma::mat & A,
    CMatrixCSC & CSC )
```

Преобразование плотной матрицы в CSC формат (Compressed [Sparse](#) Column Yale format)

Аргументы

in	A	- исходная матрица, размер [n,m] (n - число строк, m - число столбцов)
----	---	--

Аргументы

out	CSC	- структура хранения матрицы в CSC формате (Compressed Sparse Column Yale format)
-----	-----	---

См. определение в файле [sparse.cpp](#) строка 151

6.4.2.14 MatrixDenseToCSC() [2/2]

```
void SPML::Sparse::MatrixDenseToCSC (
    const arma::mat & A,
    std::vector< double > & csc_val,
    std::vector< int > & csc_kk,
    std::vector< int > & csc_first )
```

Преобразование плотной матрицы в CSC формат (Compressed [Sparse](#) Column Yale format)

Данный способ хранения эффективен, если кол-во ненулевых элементов $NNZ < (m*(n-1)-1)/2$

Аргументы

in	A	- исходная матрица, размер [n,m] (n - число строк, m - число столбцов)
out	csc_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
out	csc_kk	- вектор индексов строк ненулевых элементов, размер равен количеству ненулевых элементов nnz
out	csc_first	- вектор начальных смещений в векторе CSR, размер m+1

См. определение в файле [sparse.cpp](#) строка 123

6.4.2.15 MatrixDenseToCSR() [1/2]

```
void SPML::Sparse::MatrixDenseToCSR (
    const arma::mat & A,
    CMatrixCSR & CSR )
```

Преобразование плотной матрицы в CSR формат (Compressed [Sparse](#) Row Yale format)

Аргументы

in	A	- исходная матрица, размер [n,m] (n - число строк, m - число столбцов)
out	CSR	- структура хранения матрицы в CSR формате (Compressed Sparse Row Yale format)

См. определение в файле [sparse.cpp](#) строка 95

6.4.2.16 MatrixDenseToCSR() [2/2]

```
void SPML::Sparse::MatrixDenseToCSR (
    const arma::mat & A,
    std::vector< double > & csr_val,
    std::vector< int > & csr_kk,
    std::vector< int > & csr_first )
```

Преобразование плотной матрицы в CSR формат (Compressed [Sparse](#) Row Yale format)

Данный способ хранения эффективен, если кол-во ненулевых элементов $NNZ < (m*(n-1)-1)/2$

Аргументы

in	A	- исходная матрица, размер [n,m] (n - число строк, m - число столбцов)
out	csr_val	- вектор ненулевых элементов матрицы A, размер равен количеству ненулевых элементов nnz
out	csr_kk	- вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов nnz
out	csr_first	- вектор начальных смещений в векторе CSR, размер n+1

См. определение в файле [sparse.cpp](#) строка [67](#)

Раздел 7

Классы

7.1 Структура SPML::Sparse::CKeyCOO

Ключ элемента A_{ij} матрицы A в COO формате (Coordinate list)
`#include <sparse.h>`

Открытые члены

- `int i () const`
i - индекс строки
- `int j () const`
j - индекс столбца
- `CKeyCOO ()`
Конструктор по умолчанию
- `CKeyCOO (int i, int j)`
Параметрический конструктор
- `bool operator< (CKeyCOO const &other) const`

7.1.1 Подробное описание

Ключ элемента A_{ij} матрицы A в COO формате (Coordinate list)

Внимание

Оператор `<` перегружен для случая построчного хранения

См. определение в файле [sparse.h](#) строка 218

7.1.2 Конструктор(ы)

7.1.2.1 CKeyCOO() [1/2]

`SPML::Sparse::CKeyCOO::CKeyCOO () [inline]`

Конструктор по умолчанию

См. определение в файле [sparse.h](#) строка 234

7.1.2.2 CKeyCOO() [2/2]

`SPML::Sparse::CKeyCOO::CKeyCOO (`

`int i,`

`int j) [inline]`

Параметрический конструктор

Аргументы

in	i	- индекс строки
in	j	- индекс столбца

См. определение в файле [sparse.h](#) строка 241

7.1.3 Методы

7.1.3.1 i()

```
int SPML::Sparse::CKeyCOO::i ( ) const [inline]
```

i - индекс строки

См. определение в файле [sparse.h](#) строка 224

7.1.3.2 j()

```
int SPML::Sparse::CKeyCOO::j ( ) const [inline]
```

j - индекс столбца

См. определение в файле [sparse.h](#) строка 229

7.1.3.3 operator<()

```
bool SPML::Sparse::CKeyCOO::operator< (
    CKeyCOO const & other ) const [inline]
```

См. определение в файле [sparse.h](#) строка 243

Объявления и описания членов структуры находятся в файле:

- [sparse.h](#)

7.2 Структура SPML::Sparse::CMatrixCOO

Структура хранения матрицы в COO формате (Coordinate list)

```
#include <sparse.h>
```

Открытые атрибуты

- `std::vector< double > coo_val`
Вектор ненулевых элементов матрицы $A[n,m]$ (n - число строк, m - число столбцов), размер nnz.
- `std::vector< int > coo_row`
Индексы строк ненулевых элементов
- `std::vector< int > coo_col`
Индексы столбцов ненулевых элементов

7.2.1 Подробное описание

Структура хранения матрицы в COO формате (Coordinate list)

Матрица $A[n,m]$ (n - число строк, m - число столбцов)

См. определение в файле [sparse.h](#) строка 32

7.2.2 Данные класса

7.2.2.1 coo_col

`std::vector<int> SPML::Sparse::CMatrixCOO::coo_col`

Индексы столбцов ненулевых элементов

См. определение в файле [sparse.h](#) строка 36

7.2.2.2 coo_row

`std::vector<int> SPML::Sparse::CMatrixCOO::coo_row`

Индексы строк ненулевых элементов

См. определение в файле [sparse.h](#) строка 35

7.2.2.3 coo_val

`std::vector<double> SPML::Sparse::CMatrixCOO::coo_val`

Вектор ненулевых элементов матрицы $A[n,m]$ (n - число строк, m - число столбцов), размер `nnz`.

См. определение в файле [sparse.h](#) строка 34

Объявления и описания членов структуры находятся в файле:

- [sparse.h](#)

7.3 Структура SPML::Sparse::CMatrixCSC

Структура хранения матрицы в CSC формате (Compressed [Sparse](#) Column Yale format)

`#include <sparse.h>`

Открытые атрибуты

- `std::vector< double > csc_val`
Вектор ненулевых элементов матрицы $A[n,m]$ (n - число строк, m - число столбцов), размер `nnz`.
- `std::vector< int > csc_kk`
Вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов `nnz`.
- `std::vector< int > csc_first`
вектор начальных смещений в векторе CSC, размер `m+1`

7.3.1 Подробное описание

Структура хранения матрицы в CSC формате (Compressed [Sparse](#) Column Yale format)

Матрица $A[n,m]$ (n - число строк, m - число столбцов)

См. определение в файле [sparse.h](#) строка 54

7.3.2 Данные класса

7.3.2.1 csc_first

`std::vector<int> SPML::Sparse::CMatrixCSC::csc_first`

вектор начальных смещений в векторе CSC, размер `m+1`

См. определение в файле [sparse.h](#) строка 58

7.3.2.2 csc_kk

`std::vector<int> SPML::Sparse::CMatrixCSC::csc_kk`

Вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов `nnz`.

См. определение в файле [sparse.h](#) строка 57

7.3.2.3 csc_val

`std::vector<double> SPML::Sparse::CMatrixCSC::csc_val`

Вектор ненулевых элементов матрицы $A[n,m]$ (n - число строк, m - число столбцов), размер `nnz`.

См. определение в файле [sparse.h](#) строка 56

Объявления и описания членов структуры находятся в файле:

- `sparse.h`

7.4 Структура SPML::Sparse::CMatrixCSR

Структура хранения матрицы в CSR формате (Compressed [Sparse](#) Row Yale format)

`#include <sparse.h>`

Открытые атрибуты

- `std::vector< double > csr_val`
Вектор ненулевых элементов матрицы $A[n,m]$ (n - число строк, m - число столбцов), размер `nnz`.
- `std::vector< int > csr_kk`
Вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов `nnz`.
- `std::vector< int > csr_first`
Вектор начальных смещений в векторе CSR, размер `n+1`.

7.4.1 Подробное описание

Структура хранения матрицы в CSR формате (Compressed [Sparse](#) Row Yale format)

Матрица $A[n,m]$ (n - число строк, m - число столбцов)

См. определение в файле [sparse.h](#) строка 43

7.4.2 Данные класса

7.4.2.1 csr_first

`std::vector<int> SPML::Sparse::CMatrixCSR::csr_first`

Вектор начальных смещений в векторе CSR, размер `n+1`.

См. определение в файле [sparse.h](#) строка 47

7.4.2.2 csr_kk

`std::vector<int> SPML::Sparse::CMatrixCSR::csr_kk`

Вектор индексов колонок ненулевых элементов, размер равен количеству ненулевых элементов `nnz`.

См. определение в файле [sparse.h](#) строка 46

7.4.2.3 csr_val

`std::vector<double> SPML::Sparse::CMatrixCSR::csr_val`

Вектор ненулевых элементов матрицы $A[n,m]$ (n - число строк, m - число столбцов), размер `nnz`.

См. определение в файле [sparse.h](#) строка 45

Объявления и описания членов структуры находятся в файле:

- `sparse.h`

Предметный указатель

- AreEqualAbs
 - SPML::Compare, [16](#)
- AreEqualRel
 - SPML::Compare, [16](#), [17](#)
- CKeyCOO
 - SPML::Sparse::CKeyCOO, [37](#)
- coo_col
 - SPML::Sparse::CMatrixCOO, [38](#)
- coo_row
 - SPML::Sparse::CMatrixCOO, [39](#)
- coo_val
 - SPML::Sparse::CMatrixCOO, [39](#)
- csc_first
 - SPML::Sparse::CMatrixCSC, [39](#)
- csc_kk
 - SPML::Sparse::CMatrixCSC, [39](#)
- csc_val
 - SPML::Sparse::CMatrixCSC, [39](#)
- csr_first
 - SPML::Sparse::CMatrixCSR, [40](#)
- csr_kk
 - SPML::Sparse::CMatrixCSR, [40](#)
- csr_val
 - SPML::Sparse::CMatrixCSR, [40](#)
- Hungarian
 - SPML::LAP, [19](#)
- hungarian_augment_path
 - SPML::LAP, [20](#)
- hungarian_clear_covers
 - SPML::LAP, [20](#)
- hungarian_erase_primes
 - SPML::LAP, [20](#)
- hungarian_find_noncovered_zero
 - SPML::LAP, [20](#)
- hungarian_find_prime_in_row
 - SPML::LAP, [21](#)
- hungarian_find_smallest
 - SPML::LAP, [21](#)
- hungarian_find_star_in_col
 - SPML::LAP, [21](#)
- hungarian_find_star_in_row
 - SPML::LAP, [22](#)
- hungarian_star_in_row
 - SPML::LAP, [22](#)
- hungarian_step_1
 - SPML::LAP, [22](#)
- hungarian_step_2
 - SPML::LAP, [22](#)
- hungarian_step_3
 - SPML::LAP, [22](#)
- hungarian_step_4
 - SPML::LAP, [23](#)
- hungarian_step_5
 - SPML::LAP, [23](#)
- hungarian_step_6
 - SPML::LAP, [23](#)
- i
 - SPML::Sparse::CKeyCOO, [38](#)
- IsZeroAbs
 - SPML::Compare, [17](#)
- j
 - SPML::Sparse::CKeyCOO, [38](#)
- JVCdense
 - SPML::LAP, [23](#)
- JVCsparse
 - SPML::LAP, [24](#), [25](#)
- Mack
 - SPML::LAP, [25](#)
- MatrixCOOtoCSC
 - SPML::Sparse, [29](#)
- MatrixCOOtoCSR
 - SPML::Sparse, [30](#)
- MatrixCOOtoDense
 - SPML::Sparse, [31](#)
- MatrixCSCtoDense
 - SPML::Sparse, [31](#), [32](#)
- MatrixCSRtoDense
 - SPML::Sparse, [32](#)
- MatrixDenseToCOO
 - SPML::Sparse, [33](#)
- MatrixDenseToCSC
 - SPML::Sparse, [33](#), [34](#)
- MatrixDenseToCSR
 - SPML::Sparse, [34](#)
- operator<
 - SPML::Sparse::CKeyCOO, [38](#)
- SequentialExtremum
 - SPML::LAP, [26](#)
- solveForOneL
 - SPML::LAP, [27](#)
- SP_Max
 - SPML::LAP, [19](#)

- SP_Min
 - SPML::LAP, 19
- SPML, 15
- Spml, 13
- SPML::Compare, 15
 - AreEqualAbs, 16
 - AreEqualRel, 16, 17
 - IsZeroAbs, 17
- SPML::LAP, 18
 - Hungarian, 19
 - hungarian_augment_path, 20
 - hungarian_clear_covers, 20
 - hungarian_erase_primes, 20
 - hungarian_find_noncovered_zero, 20
 - hungarian_find_prime_in_row, 21
 - hungarian_find_smallest, 21
 - hungarian_find_star_in_col, 21
 - hungarian_find_star_in_row, 22
 - hungarian_star_in_row, 22
 - hungarian_step_1, 22
 - hungarian_step_2, 22
 - hungarian_step_3, 22
 - hungarian_step_4, 23
 - hungarian_step_5, 23
 - hungarian_step_6, 23
 - JVCdense, 23
 - JVCsparse, 24, 25
 - Mack, 25
 - SequentialExtremum, 26
 - solveForOneL, 27
 - SP_Max, 19
 - SP_Min, 19
 - TSearchParam, 19
 - updateAssignments, 27
 - updateDual, 27
- SPML::Sparse, 28
 - MatrixCOOtoCSC, 29
 - MatrixCOOtoCSR, 30
 - MatrixCOOtoDense, 31
 - MatrixCSCtoDense, 31, 32
 - MatrixCSRtoDense, 32
 - MatrixDenseToCOO, 33
 - MatrixDenseToCSC, 33, 34
 - MatrixDenseToCSR, 34
- SPML::Sparse::CKeyCOO, 37
 - CKeyCOO, 37
 - i, 38
 - j, 38
 - operator<, 38
- SPML::Sparse::CMatrixCOO, 38
 - coo_col, 38
 - coo_row, 39
 - coo_val, 39
- SPML::Sparse::CMatrixCSC, 39
 - csc_first, 39
 - csc_kk, 39
 - csc_val, 39
- SPML::Sparse::CMatrixCSR, 40
 - csr_first, 40
 - csr_kk, 40
 - csr_val, 40
- TSearchParam
 - SPML::LAP, 19
- updateAssignments
 - SPML::LAP, 27
- updateDual
 - SPML::LAP, 27