Dokumentacja iteracyjnego solvera rzadkich układów równań.

Projekt Języki i Metody Programowania I, Wydział Elektryczny

Artur Skonecki
Prowadzący: dr inż. Jacek Starzyński

Spis treści

1 Wstęp

1.1 Opis Problemu

Macierze rzadkie często mają rozmiary, które uniemożliwiają lub czynią niepraktycznym rozwiązywanie ich przez algorytmy bezpośrednie. Metoda gradientów sprzężonych umożliwia rozwiązywanie układów równań liniowych. Ponieważ jest algorytmem iteracyjnym, metoda gradientów sprzężonych może by zastosowana do układów o rzadkich macierzach.

W celu zmniejszenia rozmiarów pamięci zajmowanej przez macierze , korzystnie jest zastosować algorytmy i struktury danych, które wykorzystają strukturę macierzy rzadkich. W prezentowanym programie wykorzystywany jest `schemat Yale`. W schemacie Yale macierz przechowywana jest w 3 wektorach: **a, ia, ja**.

Element	Rozmiar	Opis
a	ilość elementów niezerowych	wszystkie niezerowe wartości macierzy
ia	liczba kolumn + 1	indeks pierwszego niezerowego elementu w wierszu
ja	ilość elementów niezerowych	pozycja kolumn niezerowych elementów

1.2 Narzędzia

Funkcja i program testowy zostały napisane w języku `C` w standardzie **c89**. Wykorzystywano kompilator **gcc**, debugger **gdb** oraz testowano wycieki pamięci przy użyciu **valgrind** i **electric-fence**. Program budowano przy użyciu **GNU Make**. Dokumentacja zastała napisana w formacie *LaTeX*.

1.3 Format danych

Jeżeli macierz nie jest kwadratowa to program testowy zakończy swoje działanie. Program wczytując macierze bierze pod uwagę tylko cyfry nad diagonalną, macierz pod diagonalną jest tworzona przez lustrzane odbicie. Wczytana macierz jest następnie zapisywana w formacie Yale. Program wczytuje dane w następującym formacie:

Przykładowa macierz.

```
 6x6 [
3 0 0 1 0 3
0 3 1 4 5 6
0 0 3 0 0 0
0 0 0 3 0 0
0 0 0 0 3 1
0 0 0 0 0 3
]
```

2 Specyfikacja funkcjonalna

2.1 Solver

```
NAZWA
      solve

SKŁADNIA

      #include "solver.h"
      vec_t solve(yale_t a, vec_t b, vec_t x0,
                  int max_iter, double precision);

OPIS
      Funkcja rozwiązuje iteracyjnie rzadki, symetryczny układ równań
      liniowych w postaci A*x=b używając zaimplementowanej metody
      (metoda gradientow sprzężonych(conjugate gradients method)).

      Macierze powinny by zapisane wg. schematu Yale.
      Dokładniejsze informacji o YSM znajdują się w Dokumentaji Głównej.

      Funkcja zwraca wskaźnik na wektor wynikowy
      lub NULL w przypadku niepowodzenia.

ARGUMENTY:
      yale_t a          - macierz zapisana w YSM
      vec_t b           - wektor
      vec_t x0          - wektor startowy
      int  max_iter     - maksymalna liczba iteracjii
      double precision  - dokładność do jakiej poszukiwane jest rozwiązanie

PRZYKŁADY
      #include "matrix.h"
      #include "yale.h"
      #include "vec.h"
      #include "solver.h"
      matrix_t m = matrix_gen_sym(10 , 0.05, -10, 10, 1);
      yale_t y=make_yale(m);
      vec_t b=vec_gen_sym(10, -10, 10, 1);
      vec_t x0= make_vec(10);

      vec_t x = solve(a,b,x,0,0,0);

      print_vec(stdout, x);
      free_mat(m); free_yale(y); free_vec(b); free_vec(x0); free_vec(x);
```

2.2 Program Testujący

```
NAZWA
      solver - program testowy dla funkcji rozwiązującej
      iteracyjnie rzadki symetryczny układ równań liniowych

SKŁADNIA
      solver [ opcje ] ...

OPIS
      Program testujący wczytuje lub generuje następujące dane:

            A - macierz rzadka (opcja -m), x - wektor wynikowy (opcja -x),
            x0 - wektor startowy (opcja -0), b = A * x

      Następnie program przekazuje dane A,b,x0 funkcji solve,
      która rozwiązuje układ równań A*x=b i zwraca wynik.
      Program testowy porównuje wynik funkcji solve i wynik prawidłowy
      i informuje użytkownika o wynikach.

      Dane, które nie zostaną określone przez użytkownika
      są generowane automatycznie. W razie niezgodności rozmiarów
      macierzy i wektorów program przerywa działanie.

      Macierze przechowywane są w pamięci w sposób efektywny - wg. schematu Yale
      Dokładniejsze informacje o YSM i o formacie danych wejściowych
      znajdują się w Dokumentacji Głównej programu.

OPCJE
      FLAGI:
      -h --help             - wyświetl krótką pomoc
      -v --verbose          - wyświetlaj więcej informacji
      -q --quiet            - wyświetlaj minimalną ilość informacji
      -a --auto             - ignoruj dane wejściowe,
                              generuj automatycznie wszystkie dane
      -n --nice             - losuj macierze z liczbami o wartościach całkowitych

      ARGUMENTY:
      -s --size <int>       - ustaw rozmiar losowanej macierzy
      -r --sparsity <double> - modyfikuj "rzadkość" macierzy
      -o --file <filename>  - drukuj wyjście do pliku
      -m --m <filename>     - macierz wejściowa
      -x --mx <filename>    - macierz wynikowa
      -0 --mx0<filename>    - macierz startowa
      -y --max <double>     - maksymalna losowana liczba
      -z --min <double>     - minimalna losowana liczba
      -c --check <double>   - dokładność z jaką program porównuje
                              wektor wejściowy i wyjściowy
      -p --precision <double> - dokładność metody rozwiązującej
      -i --max-iter <int>   - maksymalna liczba iteracji
      -d --diagonal <double>  - wartości jakimi wypełniana jest diagonalna
```

3 Specyfikacja implementacyjna

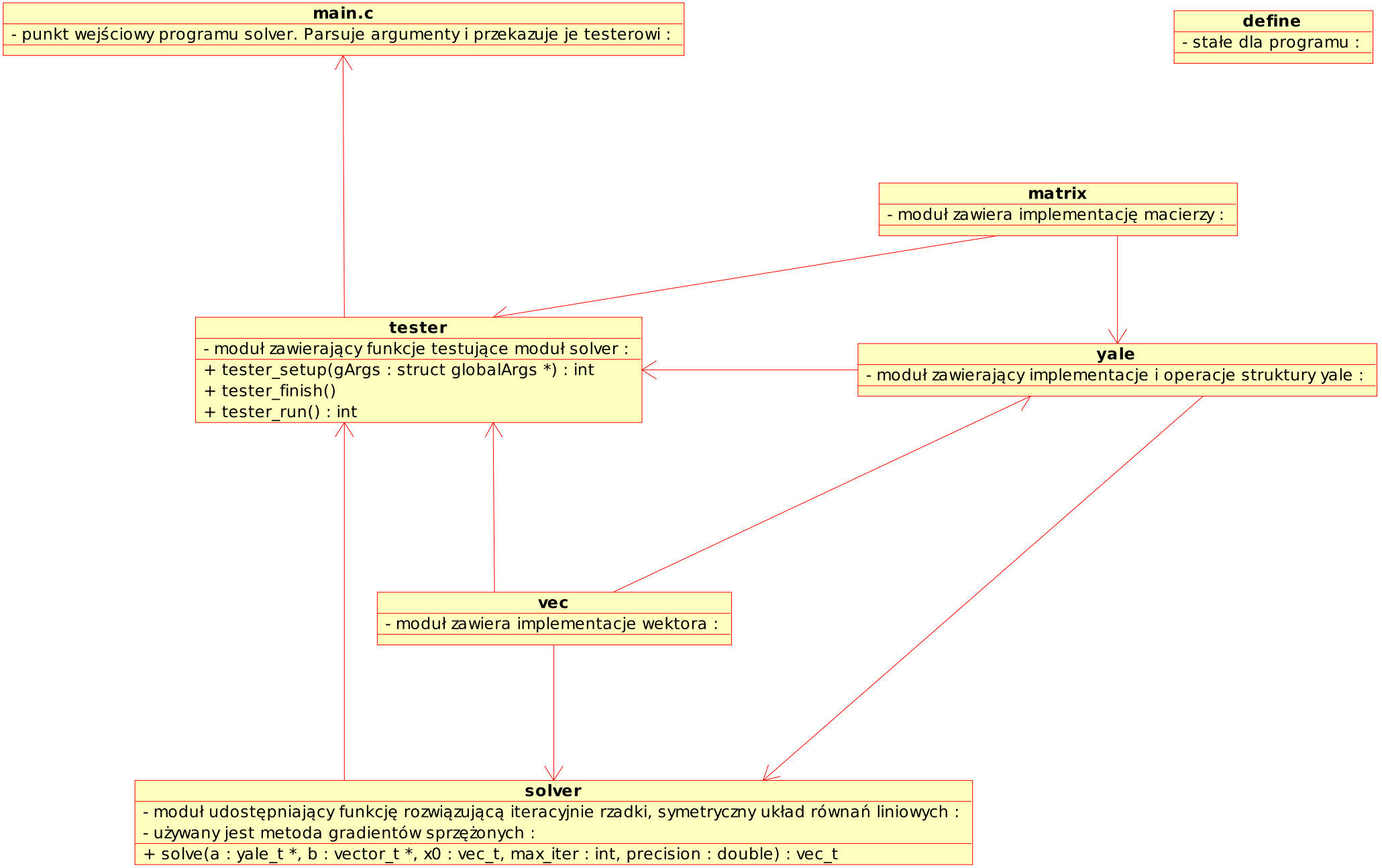
3.1 Opis ogólny

Krótki opis modułów.

MODUŁ	PLIKI	ZAWIERA	OPIS
solver	solver.c solver.h	yale vec	zawiera funkcję rozwiązującą
main	main	tester define	punkt wejściowy programu
tester	tester.c tester.h	yale vec solver matrix define	moduł testujący
vec	vec.c vec.h	define	implementacja wektora
yale	yale.c yale.c	matrix vec define	implementacja schematu Yale
matrix	matrix.c matrix.hee	define	implementacja macierzy
define	define.h	-	stałe używane w programie

Program korzysta z następujących systemowych plików nagłówkowych:

- stdlib.h
- stdio.h
- math.h
- time.h


main.c
- punkt wejściowy programu solver. Parsuje argumenty i przekazuje je testerowi :
define
- stałe dla programu :
matrix
- moduł zawiera implementację macierzy :
tester
- moduł zawierający funkcje testujące moduł solver :
+ tester_setup(gArgs : struct globalArgs *) : int
+ tester_finish()
+ tester_run() : int
yale
- moduł zawierający implementacje i operacje struktury yale :
vec
- moduł zawiera implementacje wektora :
solver
- moduł udostępniający funkcję rozwiązującą iteracyjnie rzadki, symetryczny układ równań liniowych :
- używany jest metoda gradientów sprzężonych :
+ solve(a : yale_t *, b : vector_t *, x0 : vec_t, max_iter : int, precision : double) : vec_t

3.2 Solver

3.2.1 Moduł solver

Moduł **solver** składa się z 2 plików: ***solver.c solver.h***
W **solver** znajduje się implementacja funkcji rozwiązującej iteracyjnie rzadkie układy liniowe. **solver** jest zależny od modułów *vec i yale*.

Moduł solver udostępnia 1 funkcję:

```
/* Funkcja rozwiązująca iteracyjnie rzadkie układy równań:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t solve (yale_t a, vec_t  b, vec_t  x0, int method, int max_iter, double precision)
```

argument	typ	opis
a	yale_t	macierz
b	vec_t	wektor
x0	vec_t	wektor startowy
max_iter	int	maksymalna liczba iteracjii
precision	double	dokładność wyniku

Funkcja **solve()** służy do rozwiązywania rzadkich układów liniowych. Wartość zwracana to wskaźnik na wektor wynikowy lub NULL w przypadku niepowodzenia. Zastosowano `algorytm gradientów sprzężonych` (*conjugate gradients method*).

3.2.2 Moduł vec

Moduł **vec** składa się z 2 plików: ***vec.c vec.h***
vec udostępnia prostą implementację wektora.
vec nie zależy od innych modułów. Potrzebuje jedynie plik *define.h*.

vec składa się z następujących elementów:

```
typedef struct
{
    int n;      /* liczba elementow */
    double *p;  /* tablica elementow */
}*vec_t;

/* Funkcja porównuje dwa wektory:
   zwraca TRUE lub FALSE gdy wektory są inne */
bool vec_cmp (vec_t  a, vec_t  b, double precision);

/* Funkcja generująca losowa wektor:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t vec_gen (int size, double ra, double rb, int pattern);

/* Funkcja wylicza normę wektora */
double vec_norm (vec_t  a);

/* Funkcja kopiująca wektor:
```

```
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t vec_copy (vec_t  a);

/* Funkcja wczytująca wektor ze strumienia:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t wczytaj_vec (FILE *);

/* Funkcja wczytująca wektor ze pliku:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t wczytaj_vec_plik (char *);

/* Funkcja mnożąca 2 wektory */
double vec_mul (vec_t  a, vec_t  b);

/* Funkcja mnożąca wektor przez liczbę:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t vec_mul_val (vec_t  a, double val);

/* Funkcja dodająca wektor do wektora:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t vec_add (vec_t  a, vec_t  b);

/* Funkcja odejmująca wektor od wektora:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t vec_sub (vec_t  a, vec_t  b);

/* Funkcja pobiera z wektora element o indeksie i */
double vec_get (vec_t  a, int i);

/* Funkcja zapisuje wartość do wektora w indeksie i:
   zwraca TRUE dla niepowodzenia  */
int vec_put (vec_t  a, int i, double x);

/* Funkcja mnożąca wektor przez liczbę:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
vec_t make_vec (int);

/*Funkcja uwalnia pamięć zaalokowaną dla wektora  */
void free_vec (vec_t  a);

/*Funkcja drukuje wektor do strumienia */
void print_vec (FILE * out, vec_t  a);
```

3.2.3 Moduł yale

Moduł **yale** składa się z 2 plików: ***tester.c tester.h***
yale udostępnia minimalną implementację schematu Yale.
yale jest zależny od modułów *vec i matrix*.

yale składa się z następujących elementów.

```
typedef struct yale
{
    double *a;  /* tablica z elementami niezerowymi */
    int *ia;    /* tablica z indeksami pierwszego niezerowego elementu w wierszu */
    int *ja;    /* tablica z kolumnami wszystkich niezerowych elementow*/
    int n;      /* liczba elementow niezerowych */
    int cols;   /* liczba kolumn */
} *yale_t;

/* Funkcja mnoży macierz yale przez wektor:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia*/
vec_t yale_mul_vec (yale_t y, vec_t  x0);

/* Funkcja konwertuje macierz matrix_t do yale_t:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia  */
yale_t make_yale (matrix_t  m);

/* Funkcja uwalnia pamięć zarezerwowaną dla struktury yale */
void free_yale (yale_t);

/* Funkcja alokuje pamięć dla struktury yale:
   zwraca wskaźnik na wektor lub NULL w razie niepowodzenia */
yale_t alloc_yale (int n, int ia_n);

/* Funkcja drukuje struktury yale w czytelny dla użytkownika sposób */
void print_yale (FILE * f, const yale_t y);
```

3.3 Program Testujący

3.3.1 Moduł tester

Moduł **tester** składa się z 2 plików: ***tester.c tester.h***
Zadaniem **tester**'a jest testowanie funkcji modułu **solver**.
tester jest zależny od modułów *vec, yale, matrix, solver*.
Udostępnia 3 funkcje i definicję struktury **globalArgs**, w której przechowywane są wszystkie parametry niezbędne **tester**'owi:

```
struct globalArgs
{
    /* FLAGI */
    bool verbose;
    bool silent;
    bool nice_matrix;   /* losuj liczby całkowite */
    bool show_help;
    bool m_static;      /* nie inicjuj generatora liczb losowych*/
    bool matrix_auto;   /* ignoruj dane wejściowe */

    char *outFile;      /* plik wyjściowy */
    char *matrix;       /* plik z macierza */
    char *matrix_x;     /* plik z wektorem wynikowym */
    char *matrix_x0;    /* plik z wektorem startowym */
    int m_size;         /* rozmiar losowanej macierzy */
    double m_sparsity;  /* rzadkość losowanej macierzy */
    double check;       /* dokładność porównywania macierzy
                           wynikowej i wzoru */
    double precision;   /* dokładność do jakiej funkcji rozwiązującej */
    int max_iter;       /* maksymalna liczba iteracji */
    double range_min;   /* maksymalna losowana liczba */
    double range_max;   /* minimalna losowana liczba  */
    double diagonal;    /* wartost jakimi wypełniana jest diagonalna */
};

/* Funkcja wczytuje dane na podstawie struktury globalArgs:
   w razie niepowodzenia zwraca prawdę */
int tester_setup (struct globalArgs *gArgs);

/* Funkcja uwalnia zasoby testera */
void tester_finish ();

/* Funkcja uruchamia tester:
   w razie niepowodzenia zwraca prawdę */
int tester_run ();
```

3.3.2 Moduł matrix

Moduł **matrix** składa się z 2 plików: ***matrix.c matrix.h***
Moduł **matrix** jest elementem pomocniczym używanym przez tester w celu wczytania lub losowego wygenerowania macierzy testowej.
matrix nie zależy od innych modułów. Potrzebuje jedynie plik *define.h*.

Elementy modułu **matrix** potrzebne testerowi:

```
typedef struct
{
    int rn; /* liczba kolumn */
    int cn; /* liczba wierszy
    double **p; /* 2 wymiarowa tablica z elemntami */
}*matrix_t;

/* Utworz nową macierz */
/* zwraca macierz lub NULL dla niepowodzenia */
matrix_t make_matrix (int, int);

/* Pobierz wartosc elementu */
double matrix_get (matrix_t  a, int i, int j);

/* wygeneruj losowo symetryczną macierz */
/* zwraca macierz lub NULL dla niepowodzenia */
matrix_t matrix_gen_sym (int size, double sparsity, double ra, double rb, int pattern);

/* zsymetryzuj macierz względem diagonalnej */
void matrix_symmetrize (matrix_t  a);

/* wczytaj macierz z pliku */
/* zwraca macierz lub NULL dla niepowodzenia */
matrix_t wczytaj_matrix_plik (char *);

/* wczytaj macierz ze strumienia */
/* zwraca macierz lub NULL dla niepowodzenia */
matrix_t wczytaj_matrix (FILE *);

/* drukuj macierz */
void print_matrix (FILE * out, matrix_t  a);

/* uwolniej zasoby macierzy */
void free_matrix (matrix_t  a);
```

3.3.3 Moduł main

Moduł **main** składa się z 1 pliku: ***main.c***
Zadaniem **main** jest parsowanie argumentów lini poleceń przy użyciu biblioteki **getopt** i odpowiednie zapisanie ich w strukturze **globalArgs**, która zostaję następnie przekazana modułowi **tester**. **main** jest zależny od *tester*.
main nie udostępnia żadnych funkcji.

4 Testy

4.1 Kompilacja

System operacyjny: Linux 2.6.24-21 Kompilator: gcc 4.2.4

4.2 Wykorzystanie pamięci systemowej.

Program testujący i funkcja **solve()** zostały przetestowane przy użyciu programu **valgrind** i b**iblioteki electric-fence**. Testy nie wykazały utraty pamięci.

4.3 Program Testujący

Funkcje matematyczne programu testującego zostały przetestowane przy użyciu programu **matlab**.

4.4 Solver

Przykładowe testy wykonywane na module **solver**:

```
$ ./solver --verbose --sparsity 0.4 -min 0 --max 20  --diagonal 3.0
# losuj i rozwiązuj układ o rzadkości 0.4, liczbach z zakresu 0 20,
# wypełnij diagonalną 3.0, wyświetlaj maksymalną ilość informacji

$ ./solver -m data/m30x30 -x data/v30 --precision 1e-15
# wczytaj macierz i wektor wzorcowy, precyzja = 1e-15

$ ./solver --size 1000 --max-iter 2000 --sparsity 0.05 --quiet
# losuj macierz o rozmiarze 1000x1000, maksymalna liczba iteracji = 2000,
# rzadkość macierzy = 0.05, wyświetlaj minimalną ilość informacji

$ ./s ---nice --min -2000 --max 2000 --size 100 --diagonal 100
# losuj liczby całkowite z zakresu -2000 2000,
# rozmiar macierzy = 100, wypełnij diagonalną 100
```

Przy dużych macierzach, macierzach o dużych liczbach i macierzach o małej rzadkości koniecznie jest zwiększenie liczby maksymalnej iteracji(domyślna 1000).

5 Bibliografia

[1] http://en.wikipedia.org/wiki/Sparse_matrix

[2] http://en.wikipedia.org/wiki/Conjugate_gradient_method

[3] http://www.inf.uni-konstanz.de/cgip/lehre/na_08/index.shtml.de

[4] http://www.icm.edu.pl/kdm/Metoda_gradient%C3%B3w_sprz%C4%99%C5%BConych_CG

[5] http://icis.pcz.pl/~roman/mat_dyd/zast_prz_rown/zrown.html