MOwNiT Lab6 - sprawozdanie

Autor: Michał Flak

Polecenie

```
    Korzystając z przykładu napisz program, który:

1.      Jako parametr pobiera rozmiar układu równań n
2.      Generuje macierz układu A(nxn) i wektor wyrazów wolnych b(n)
3.      Rozwiązuje układ równań
4.      Sprawdza poprawność rozwiązania (tj., czy Ax=b)
5.      Mierzy czas dekompozycji macierzy - do mierzenia czasu można skorzystać z
  przykładowego programu dokonującego pomiaru
  czasu procesora spędzonego w danym fragmencie programu.
6.      Mierzy czas rozwiązywania układu równań

·  Zadanie domowe: Narysuj wykres zależności czasu dekompozycji i czasu
rozwiązywania układu od rozmiaru układu równań.
Wykonaj pomiary dla 10 wartości z przedziału od 10 do 1000.
```

Rozwiązanie

Analogicznie jak w przykładzie tworzymy macierz, wektor i rozwiązujemy za pomocą GSL (tworzymy też kopię pierwotnej macierzy, bo dekompozycja modyfikuje nam pierwotne dane). Zapisujemy czas wykonania w zmiennych (metody generowania danych oraz mierzenia czasu do wglądu w listingu kodu na końcu sprawozdania):

```
  double* a_data = generate_matrix(n);
  double* a_copy = copyVector(a_data, n*n);

  double* b_data = generate_vector(n);

  gsl_matrix_view m
    = gsl_matrix_view_array (a_data, n, n);
  gsl_matrix_view m_copy
    = gsl_matrix_view_array (a_copy, n, n);

  gsl_vector_view b
    = gsl_vector_view_array (b_data, n);

  gsl_vector *x = gsl_vector_alloc (n);

  int s;

  gsl_permutation * p = gsl_permutation_alloc (n);

  double decomptime = decomp (&m.matrix, p, &s);

  double solvetime = solve (&m.matrix, p, &b.vector, x);
```

Poprawność rozwiązania sprawdzamy mnożąc pierwotną macierz przez wektor wynikowy za pomocą `gsl_blas_dgemv`. Porównujemy otrzymany wektor z pierwotnym wektorem `b` do epsilonu=0.005:

```
int compare_vectors(gsl_vector*a, gsl_vector*b) {
  if(a->size != b->size) return 0;
```

```
  const double eps = 0.005;

  for(int i = 0; i < a->size; i++){
    if(fabs(gsl_vector_get(a,i)-gsl_vector_get(b,i)) > eps) {
      return 0;
    }
  }
  return 1;
}

int verify_correctness(int n, gsl_matrix *m, gsl_vector*b, gsl_vector *x) {

  gsl_vector *y = gsl_vector_alloc(n);
  int err = gsl_blas_dgemv( CblasNoTrans, 1.0, m, x, 1.0, y );

  return compare_vectors(y, b);
}
```

Wypisujemy wynik na wyjście standardowe w formacie:

```
Size: 700, DTime: 0.10751800000000000, STime: 0.00103000000000000, Correct: 1
```

Następującym skryptem uruchamiamy program dla 10 wartości w przedziale [10,1000]:

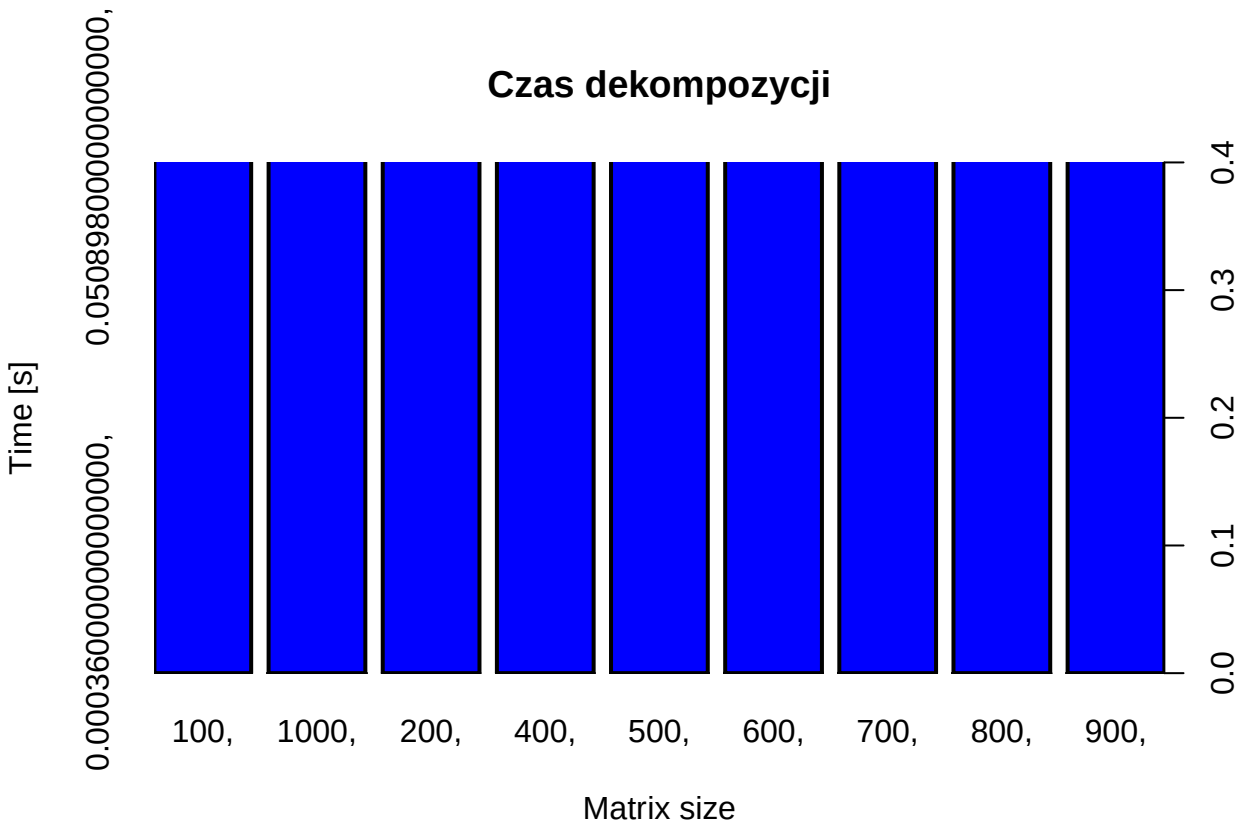
```
rm wyniki.txt
for c in {10, 100, 200, 400, 500, 600, 700, 800, 900, 1000}
do
    ./lab6 $c | tee -a wyniki.txt
done
```

Wyniki

Zawartość `wyniki.txt`:

```
Size: 100, DTime: 0.00036000000000000, STime: 0.00003400000000000, Correct: 1
Size: 200, DTime: 0.00272200000000000, STime: 0.00010400000000000, Correct: 1
Size: 400, DTime: 0.02155200000000000, STime: 0.00038700000000000, Correct: 1
Size: 500, DTime: 0.05089800000000000, STime: 0.00086500000000000, Correct: 1
Size: 600, DTime: 0.07247099999999999, STime: 0.00106600000000000, Correct: 1
Size: 700, DTime: 0.10751800000000000, STime: 0.00103000000000000, Correct: 1
Size: 800, DTime: 0.16430400000000001, STime: 0.00128000000000000, Correct: 1
Size: 900, DTime: 0.26297900000000002, STime: 0.00174800000000000, Correct: 1
Size: 1000, DTime: 0.34624500000000002, STime: 0.00233800000000000, Correct: 1

## Warning in .External.graphics(C_rect, as.double(xleft), as.double(ybottom), :
## parametr graficzny 'type' jest przestarzały
```



Program

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_spline.h>
#include <gsl/gsl_interp.h>
#include <gsl/gsl_chebyshev.h>
#include <gsl/gsl_fit.h>
#include <gsl/gsl_integration.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_blas.h>

double* generate_matrix(int n) {
  double* matrix = (double*)malloc(n*n*sizeof(double));

  srand(2137);
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
      //printf("x: %d, y: %d\n", i,j);
      matrix[i*n+j] = ((double)rand())/RAND_MAX;
    }
```

```
  }

  return matrix;
}

double* generate_vector(int n) {
  double* vector = malloc(n*sizeof(double));
  for(int i = 0; i < n; i++) {
    vector[i] = i+1;
  }

  return vector;
}

double* copyVector(double* src, int size)
{
    double* vec = calloc(sizeof(double), size);
    for (int i = 0; i < size; ++i){
        vec[i] = src[i];
    }
    return vec;
}

int compare_vectors(gsl_vector*a, gsl_vector*b) {
  if(a->size != b->size) return 0;

  const double eps = 0.005;

  for(int i = 0; i < a->size; i++){
    if(fabs(gsl_vector_get(a,i)-gsl_vector_get(b,i)) > eps) {
      return 0;
    }
  }
  return 1;
}



int print_matrix(FILE *f, const gsl_matrix *m)
{
        int status, n = 0;

        for (size_t i = 0; i < m->size1; i++) {
                for (size_t j = 0; j < m->size2; j++) {
                        if ((status = fprintf(f, "%g, ", gsl_matrix_get(m, i, j))) < 0)
                                return -1;
                        n += status;
                }

                if ((status = fprintf(f, "\n")) < 0)
                        return -1;
                n += status;
        }
```

```
        return n;
}

int print_vector(const gsl_vector *v){
  for(int i = 0; i < v->size; i++){
    printf("%g, ", gsl_vector_get(v, i));
  }
  printf("\n");
}

int verify_correctness(int n, gsl_matrix *m, gsl_vector*b, gsl_vector *x) {

  gsl_vector *y = gsl_vector_alloc(n);
  int err = gsl_blas_dgemv( CblasNoTrans, 1.0, m, x, 1.0, y );

  return compare_vectors(y, b);
}

double decomp(gsl_matrix *A, gsl_permutation* p, int *signum)
{
    clock_t start = clock();
    gsl_linalg_LU_decomp(A, p, signum);
    clock_t end = clock();

    double t = ((double) (end - start)) / CLOCKS_PER_SEC;
    return t;
}

double solve(const gsl_matrix* LU, const gsl_permutation* p, const gsl_vector* b, gsl_vector* x)
{
    clock_t start = clock();
    gsl_linalg_LU_solve(LU, p, b, x);
    clock_t end = clock();

    double t = ((double) (end - start)) / CLOCKS_PER_SEC;
    return t;
}

int calculate(int n){
  double* a_data = generate_matrix(n);
  double* a_copy = copyVector(a_data, n*n);

  double* b_data = generate_vector(n);

  gsl_matrix_view m
    = gsl_matrix_view_array (a_data, n, n);
  gsl_matrix_view m_copy
    = gsl_matrix_view_array (a_copy, n, n);

  gsl_vector_view b
    = gsl_vector_view_array (b_data, n);

  gsl_vector *x = gsl_vector_alloc (n);
```

```
  int s;

  gsl_permutation * p = gsl_permutation_alloc (n);

  double decomptime = decomp (&m.matrix, p, &s);

  double solvetime = solve (&m.matrix, p, &b.vector, x);

  printf("Size: %d, DTime: %.17f, STime: %.17f, Correct: %d\n",
    n,
    decomptime,
    solvetime,
    verify_correctness(n, &(m_copy.matrix), &(b.vector), x)
  );
  gsl_permutation_free (p);
  gsl_vector_free (x);
}


int main(int argc, char const *argv[])
{
  if(argc != 2) return -1;
  int n = atoi(argv[1]);

  calculate(n);

  return 0;
}
```