logo.png

Figure 1: logo uca

PRÉP'ISIMA

ADRESSE: campus Cézeaux, Clermont-Ferrand
INTERNET: www.uca.fr et www.isima.fr

isima.png

Figure 2: logo ISIMA

Algorithmique Numérique

latex.png

Figure 3: logo

SABIER Corentin et CHASSAGNOL Rémi
année 2020-2021

Professeur: CHORFI amina

24 octobre 2020

Table des matières

Algorithmiques directes

Gauss

Cholesky

0.0.1 Principe

Le principe de cette méthode est de trouver une matrice R telles que $A = R^T * R$ où R est une matrice triangulaire supérieure et R^T est la matrice transposée de R. Le système $Ax = b$ peut donc se diviser en deux sous-systèmes $R^T y = b$ et $Rx = y$. Cette méthode est légèrement plus rapide que celle de Gauss avec une complexité de $\frac{n^3}{3}$ au lieu de $\frac{2n^3}{3}$, cependant elle nécessite que la matrice soit définie positive.

0.0.2 L'Algorithme

L'algorithme qui permet de trouver la matrice R est le suivant:

```
Pour i allant de 1 a n:
    s = a_ii − Σ_{j=1}^{i−1} r_ji^2
    Si s ⩽ 0 alors:
        arr t(la matrice n'est pas d finie positive)
    Sinon
        r_ii = √s
        Pour j allant de i+1 a n:
            r_ij = (a_ij − Σ_{k=1}^{i−1}) / r_ii
```

Ensuite il suffit de résoudre les deux sous-systèmes ce qui ce fait facilement car la matrice est triangulaire.

0.0.3 Implémentation en C

Dans la fonction *main* on initialise un matrice de taille LEN qui est une constante pré-processeur. Les valeurs dans la matrice dépendent de la constante EXAMPLE qui permet de déterminer l'initialisation de la matrice via une fonction qui se situe dans le fichier *matrix.c*. Il y a en tout 8 matrices de testes qui sont celle demandé dans la fiche de TP (chaque fonction remplit le tableau en fonction de la définition de la matrice).

```c
  float *solus = malloc(LEN * sizeof(float));
  if (solus == NULL)
    exit(EXIT_FAILURE);
  float *b = malloc(LEN * sizeof(float));
  if (b == NULL)
    exit(EXIT_FAILURE);
  for (i = 0; i < LEN; i++) {
    b[i] = 1;
  }
  float **matrix = create_mat(LEN);
  mat(matrix, LEN, EXAMPLE);
  show(matrix, LEN);
  solver_cholesky(matrix, solus, b, LEN);
```

Ensuite on calcule la matrice R (le tableau à deux dimensions r) à l'aide de la fonction nommée MAKE_R qui suit l'algorithme ci-dessus.

```c
void make_R(float **matrix, float **r, int n) {
  int i, j, k;    // loop var
  float sum = 0; // Used to calculate the sum of r_ki*r_kj
  float s;
  for (i = 0; i < n; i++) {
    s = matrix[i][i];
    for (j = 0; j < i; j++) {
      s -= r[j][i] * r[j][i];
    }
    if (s <= 0) {
```

```
        printf("La matrice n'est pas d finie positive\n");
        exit(EXIT_FAILURE);
      } else {
        r[i][i] = sqrtf(s);
        for (j = (i + 1); j < n; j++) {
          sum = 0;
          for (k = 0; k < i; k++) {
            sum += r[k][i] * r[k][j];
          }
          r[i][j] = (matrix[i][j] - sum) / r[i][i];
        }
      }
    }
  }
```

Enfin la fonction SOLVER_CHOLESKY résout les deux sous-systèmes de la même manière qu'avec *Gauss* sauf que les a_{ii} ne sont pas forcement égaux à 1 d'où la division pas a_{ii}. De plus, dans le premier sous-système la matrice R^T est triangulaire inférieure, la résolution se fait donc en commencent par calculer la valeur de la première inconnue.

```
void solver_cholesky(float **matrix, float *solus_x, float *b, int n) {
  int i, j;
  float *solus_y = malloc(n * sizeof(float));
  /*Creation of the matrix  R:*/
  float **r = create_mat(n);
  mat_0(r, n);
  make_R(matrix, r, n);
  /*Resolution de R^T * y = b*/
  transpose(r, n);
  for (i = 0; i < n; i++) {
    for (j = 0; j < i; j++) {
      b[i] -= solus_y[j] * r[i][j];
    }
    solus_y[i] = b[i] / r[i][i];
  }
  /*Solving R * x = y*/
  transpose(r, n);
  for (i = (n - 1); i >= 0; i--) {
    for (j = (n - 1); j > i; j--) {
      solus_y[i] -= solus_x[j] * r[i][j];
    }
    solus_x[i] = solus_y[i] / r[i][i];
  }
  /*FREE*/
  free(solus_y);
  for (i = 0; i < n; i++) {
    free(r[i]);
  }
  free(r);
}
```

algorithmes itératifs

Jacobi

Gauss-seidel