Devoir 2

Corrigé

Exercice I

1. On a

$$f(x) = \frac{1}{2}\langle Ax - b, Ax - b\rangle$$

ainsi, pour tout $k \in [\![1, N]\!]$ on a

$$\frac{\partial f(x)}{\partial x_k} = \frac{1}{2}\left\langle \frac{\partial}{\partial x_k}(Ax - b), Ax - b\right\rangle + \frac{1}{2}\left\langle Ax - b, \frac{\partial}{\partial x_k}(Ax - b)\right\rangle = \left\langle \frac{\partial}{\partial x_k}(Ax), Ax - b\right\rangle$$

Pour $i \in [\![1, N]\!]$ Notons $(Ax)_i$ la i-ème composante du vecteur Ax. On a

$$(Ax)_i = \sum_{j=1}^{N} A_{ij}x_j$$

ainsi

$$\frac{\partial}{\partial x_k}((Ax)_i) = \frac{\partial}{\partial x_k}\sum_{j=1}^{N} A_{ij}x_j = \sum_{j=1}^{N} A_{ij}\frac{\partial}{\partial x_k}x_j = \sum_{j=1}^{N} A_{ij}\delta_j^k = A_{ik}$$

En notant A_k la k-ième colonne de A, il vient

$$\frac{\partial f(x)}{\partial x_k} = \langle A_k, Ax - b\rangle$$

ainsi

$$\nabla f(x) = \begin{pmatrix} \langle A_1, Ax - b\rangle \\ \vdots \\ \langle A_N, Ax - b\rangle \end{pmatrix} = {}^tA(Ax - B) = A(Ax - B)$$

De plus, pour tout $l \in [\![1, N]\!]$ on a

$$\frac{\partial^2 f(x)}{\partial x_k \partial x_l} = \frac{\partial}{\partial x_l}\left(\langle A_k, Ax - b\rangle\right) = \left\langle A_k, \frac{\partial}{\partial x_l}(Ax)\right\rangle = \langle A_k, A_l\rangle$$

ainsi

$$\mathrm{H}f(x) = \begin{pmatrix} \langle A_1, A_1\rangle & \cdots & \langle A_1, A_N\rangle \\ \vdots & & \vdots \\ \langle A_N, A_1\rangle & \cdots & \langle A_N, A_N\rangle \end{pmatrix} = {}^tAA = A^2$$

2. La matrice A est non-singulière. Posons $x^* = A^{-1}b$ alors $f(x^*) = \frac{1}{2}\|AA^{-1}b - b\|^2 = 0$ ainsi pour tout $x \in \mathbb{R}^N$ on a $f(x) \geq f(x^*)$. Par suite x^* est le minimum global de f. D'autre part, la fonction f admet un point critique en x si et seulement si $\nabla f(x) = 0$ c'est -à-dire $A(Ax - b) = 0$. La matrice A étant non singulière cela équivaut à $x = A^{-1}b$. Il y a donc un point critique unique : x^*.

3. On a

$$\|Ax-b\|^2 = \langle Ax-b, Ax-b\rangle = \|Ax\|^2 - 2\langle Ax, b\rangle + \|b\|^2$$

La matrice A est symétrique réelle, il existe une base orthonormale de vecteurs propres $u_1, \dots u_N$. Soit $\lambda_1, \dots, \lambda_N$ les valeurs propres associées. Notons $\alpha = \min\{|\lambda_1|, \dots, |\lambda_N|\}$. La matrice A est inversible donc aucune de ses valeurs propres n'est nulle, donc $a > 0$. On a

$$\begin{aligned}
\|Ax\|^2 &= \left\| \sum_{i=1}^N A(\langle x, u_i\rangle u_i) \right\|^2 \\
&= \left\| \sum_{i=1}^N \langle x, u_i\rangle \lambda_i u_i \right\|^2 \\
&= \left\langle \sum_{i=1}^N \langle x, u_i\rangle \lambda_i u_i, \sum_{i=j}^N \langle x, u_i\rangle \lambda_i u_i \right\rangle \\
&= \sum_{i=1}^N \sum_{j=1}^N \langle \langle x, u_i\rangle \lambda_i u_i, \langle x, u_j\rangle \lambda_j u_j\rangle \\
&= \sum_{i=1}^N \sum_{j=1}^N \langle x, u_i\rangle \lambda_i \langle x, u_j\rangle \lambda_j \langle u_i, u_j\rangle \\
&= \sum_{i=1}^N \sum_{j=1}^N \langle x, u_i\rangle \lambda_i \langle x, u_j\rangle \lambda_j \delta_i^j \\
&= \sum_{i=1}^N \| \langle x, u_i\rangle \lambda_i u_i\|^2 \\
&= \sum_{i=1}^N \lambda_i^2 \| \langle x, u_i\rangle u_i\|^2 \\
&\geq \alpha^2 \sum_{i=1}^N \| \langle x, u_i\rangle u_i\|^2 \\
&\geq \alpha^2 \|x\|^2
\end{aligned}$$

D'autre part

$$\langle Ax, b\rangle \leq \|Ax\|\|b\| \leq |||A|||x\|\|b\|$$

donc

$$\|Ax-b\| \geq \alpha^2\|x\|^2 - 2|||A|||x\|\|b\| + \|b\|^2$$

d'où finalement

$$\frac{1}{2}\|Ax-b\| \geq \|x\|\left(\frac{1}{2}\alpha^2\|x\| - |||A|||b\|\right) + \frac{1}{2}\|b\|^2$$

ainsi

$$\lim_{\|x\|\to+\infty} \frac{1}{2}\|Ax-b\| = +\infty$$

donc f est coercive.

4. Soit x et y deux éléments de $\mathbb{R}^N$ alors

$$\|\nabla f(y) - \nabla f(x)\| = \|A(Ay-b) - A(Ax-b)\| = \|A^2(y-x)\| \leq |||A^2|||y-x\|$$

Posons alors $M = |||A^2|||$. Les valeurs propres de A^2 sont $\lambda_1^2, \dots, \lambda_N^2$, on a dont

$$M = \min\{\lambda_1^2, \dots, \lambda_N^2\}$$

Exercice II

1. Notons

$$\begin{array}{rcl} \varphi : \mathcal{M}_{N\times N}(\mathbb{R}) \times \mathcal{M}_{N\times N}(\mathbb{R}) & \to & \mathcal{M}_{N\times N}(\mathbb{R}) \\ (A, B) & \mapsto & A \cdot\cdot B \end{array}$$

On a

- Soit A et B dans $\mathcal{M}_{N\times N}(\mathbb{R})$, on a

$$\varphi(A, B) = A \cdot\cdot B = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}B_{ij} = \sum_{i=1}^{N}\sum_{j=1}^{N} B_{ij}A_{ij} = B \cdot\cdot A = \varphi(B, A)$$

  donc la fonction φ est symétrique.

- Soit A, A', B dans $\mathcal{M}_{N\times N}(\mathbb{R})$ et $\lambda \in \mathbb{R}$, on a

$$\varphi(A + \lambda A', B) = (A + \lambda A') \cdot\cdot B = \sum_{i=1}^{N}\sum_{j=1}^{N}(A_{ij} + \lambda A'_{ij})B_{ij}$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}B_{ij} + \lambda\sum_{i=1}^{N}\sum_{j=1}^{N} A'_{ij}B_{ij} = A \cdot\cdot B + \lambda A' \cdot\cdot B = \varphi(A, B) + \lambda\varphi(A', B)$$

  donc la fonction φ est bilinéaire.

- Soit A dans $\mathcal{M}_{N\times N}(\mathbb{R})$, on a

$$\varphi(A, A) = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}^2 \geq 0$$

  donc φ est positive.

- Soit A dans $\mathcal{M}_{N\times N}(\mathbb{R})$ telle que $\varphi(A, A) = 0$ ainsi

$$\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}^2 = 0$$

  donc pour tout $(i, j) \in [\![1, N]\!]^2$ on a $A_{ij} = 0$ ainsi $A = 0$. Par suite A est ́definie.

La fonction φ est bilinéaire sym etrique définie positive, c'est un produit scalaire.

2. L'expression de $\|\!| \cdot |\!\|_F$ en fonction des composantes de A résulte directement de la définition

$$|\!|\!| A |\!|\!|_F = \sqrt{\sum_{i=1}\sum_{j=1} A_{ij}^2}$$

3. Soit A une matrice symétrique réelle. Elle admet N valeurs propres $\lambda_1, \ldots, \lambda_N$. Notons ρ la plus grande de ces valeurs propres. On a

$$\|\!|A|\!\|^2 = \rho^2 \leq \sum_{i=1}^{N} \lambda_i^2$$

Or les valeurs propres de A^2 sont $\lambda_1^2, \ldots, \lambda_N^2$ et la somme des valeurs propres d'une matrice est la trace de cette matrice. Ainsi

$$\sum_{i=1}^{N} \lambda_i^2 = \mathrm{tr}(A^2) = A \cdot\cdot A = \|\!|A|\!\|_F^2$$

Il en résulte l'inégalité souhaitée: $\|\!|\cdot|\!\| \leq \|\!|\cdot|\!\|_F$.

4. Considérons les hypothèses et notations de l'exercice I. Nous avions $M = \|\!|A^2|\!\|$. En vertu de la question précédente

$$M \leq \|\!|A^2|\!\|_F = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}^4}$$

Exercice III

On crée un type pour les vecteurs et pour les matrices symétriques. Dans le programme qui suit on tient compte du caractère symétrique de la matrice pour ne stocker que la partie supérieure. Il est possible de réaliser un type matrices simple et de l'utiliser tel quel. On remarquera qu'il y a quatre fonctions permettant d'accéder au type vecteur et quatre fonctions permettant d'accéder au type matrice. En cas de changement de ces types, il suffit de changer les quatre fonctions en question et pas le reste du programme.

```
#include <stdio.h>
#include <stdlib.h>

/* Dimension du systeme */

#define N 5


/*
 * Definition du type vecteur de R^N (vector)
 * Un simple tableau suffit
 * La i-eme composante du vecteur sera stockee dans la case i-1
 */

typedef double* vector;


/*
 * Defintion du type matrices symetriques reelles NxN (matrix)
 * Afin d'eviter de stocker plus d'elements que necessaire on ne
 * stocke que la moitie de la matrice dans un tableau unidimensionel
 * Par exemple dans le cas 5x5, il y a 15 elements a stocker et non
 * 25 en raison de la symetrie, la matrice
```

```
 *
 * [ a b c d e ]
 * [ b f g h i ]
 * [ c g j k l ]
 * [ d h k m n ]
 * [ e i l n o ]
 *
 *  est stockee [ a b c d e f g h i j k l m n o ].
 */

typedef double* matrix;


/*
 * Fonctions pour acceder au type vector
 */


/* Reservation de l'espace memoire pour un vecteur */
vector vector_new ()
{
  return((double*)malloc(N*sizeof(double)));
}


/* Affecation de x a la i-eme composante de v */
void vector_set_elt (vector v, int i, double x)
{
  v[i-1]=x;
}


/* Retourne la i-eme composante de v */
double vector_get_elt (vector v, int i)
{
  return(v[i-1]);
}


/* Liberation de l'espace memoire de v */
void vector_free (vector v)
{
  free(v);
}


/*
 * Fonctions pour acceder au type matrix
 */


/* Reservation de l'espace memoire pour une matrice */
matrix matrix_new()
{
```

```
  /* Une matrice NxN necessite N+(N-1)+...+1/N(N+1)/2 cases memoire */
  return((double*)malloc(N*(N+1)/2*sizeof(double)));
}


/* Affecte x a la composante (i,j) de la matrice A */
void matrix_set_elt (matrix A, int i, int j, double x)
{
  /*
   * Premier cas i<=j
   *
   * Au dessus de la ligne i, on a stocke N+(N-1)+...+(N-(i-2)) elements
   * N+(N-1)+...+(N-(i-2)) = (i-1)*N-(1+2+...+(i-2))
   *
   * Sur la ligne i, les i-1 premiers elements de la ligne ne sont pas stockes
   * L'element de la colonne j correspond est donc le j-(i-1) eme element a
   * stocker
   *
   * La composante (i,j) a est donc le (i-1)*N-(1+2+...+(i-2))+j-(i-1) eme
   * element du tableau unidimensionel.
   *
   * (i-1)*N-(1+2+...+(i-2))+j-(i-1)
   * = (i-1)*N-(1+2+...+(i-2)+(i-1))+j
   * = (i-1)*N-i*(i-1)/2+j
   *
   * le n-eme element d'un tableau T est T[n-1], ainsi la composante (i,j)
   * de A est A[(i-1)*N-i*(i-1)/2+j-1]
   */
  if (i<=j) A[(i-1)*N-(i)*(i-1)/2+j-1]=x;
  /*
   * Deuxieme cas i>j
   *
   * Si i<j on exploite la symetrie de la matrice
   */
  else matrix_set_elt(A,j,i,x);
}


/* Retourne la composante (i,j) de A */
double matrix_get_elt (matrix A, int i, int j)
{
  /* Voir la fonction matrix_set_elt pour une explication du sotckage */
  if (i<=j) return(A[(i-1)*N-(i)*(i-1)/2+j-1]);
  else return(matrix_get_elt(A,j,i));
}


/* Liberation de l'espace memoire de A */
void matrix_free (matrix A)
{
  free(A);
}
```

```
/*
 * Fonctions relatives aux types vector et matrix
 */


/* Affichage de la matrice A */
void matrix_print (matrix A)
{
  int i,j;
  for (i=1;i<=N;i++) {
    printf("[");
    for (j=1;j<=N;j++)
      printf("%6.2f ",matrix_get_elt(A,i,j));
    printf("]\n");
  }
}


/* Affichage du vecteur v */
void vector_print (vector v)
{
  int i;
  printf("[");
  for (i=1;i<=N;i++)
    printf("%6.2f ",vector_get_elt(v,i));
  printf("]");
}


/* Prend deux vecteurs u et v en argument et retourne u-v */
vector vector_difference(vector u, vector v)
{
  int i;
  vector diff=vector_new();
  for (i=1;i<=N;i++)
    vector_set_elt(diff,i,vector_get_elt(u,i)-vector_get_elt(v,i));
  return diff;
}


/* Prend en argument un vecteur u et un scalaire r et retourne ru */
vector vector_real_product(double r, vector u)
{
  int i;
  vector res=vector_new();
  for (i=1;i<=N;i++)
    vector_set_elt(res,i,vector_get_elt(u,i)*r);
  return res;
}


/* Prend en argument une matrice M et un vecteur v et retourne Mv */
vector matrix_vector_product (matrix M, vector v)
{
```

```
  int i,j;
  double val, *prod=matrix_new();
  for (i=1;i<=N;i++) {
    val=0;
    /* Formule du produit matrice vecteur */
    for (j=1;j<=N;j++)
      val+=matrix_get_elt(M,i,j)*vector_get_elt(v,j);
    vector_set_elt(prod,i,val);
  }
  return prod;
}


/* Prend deux vecteurs u et v en argument et retourne le produit scalaire u.v */
double vector_inner_product (vector u, vector v)
{
  int i;
  double val=0;
  for (i=1;i<=N;i++)
    val+=vector_get_elt(u,i)*vector_get_elt(v,i);
  return(val);
}


/*
 * Mise en place de l'algorithme du gradient pour resoudre Ax=b
 */


/* Fonction dont on cherche le minimum ; A, b et x sont ceux de l'ennonce */
double f (matrix A, vector b, vector x)
{
  vector Ax, Ax_moins_b;
  double norme_sq; /* carre de la norme */
  Ax=matrix_vector_product(A,x);
  Ax_moins_b=vector_difference(Ax,b);
  norme_sq=vector_inner_product(Ax_moins_b,Ax_moins_b);
  vector_free(Ax);
  vector_free(Ax_moins_b);
  return(norme_sq/2);
}


/* Gradient de la fonction dont on cherche le minimum */
vector gradiant (matrix A, vector b, vector x)
{
  vector Ax, Ax_moins_b, grad;
  Ax=matrix_vector_product(A,x);
  Ax_moins_b=vector_difference(Ax,b);
  grad=matrix_vector_product(A,Ax_moins_b);
  vector_free(Ax);
  vector_free(Ax_moins_b);
  return(grad);
}
```

```
int main() {
  double epsilon=1e-6;
  double e, majorantM=0, alpha;
  vector U, Unext, grad, alpha_grad, b;
  matrix A;
  int i, j, n=0;

  /* Creation de la matrice A */
  A=matrix_new();
  matrix_set_elt(A,1,1,2);
  matrix_set_elt(A,1,2,-1);
  matrix_set_elt(A,1,3,1);
  matrix_set_elt(A,1,4,0);
  matrix_set_elt(A,1,5,1);
  matrix_set_elt(A,2,2,0);
  matrix_set_elt(A,2,3,1);
  matrix_set_elt(A,2,4,0);
  matrix_set_elt(A,2,5,-1);
  matrix_set_elt(A,3,3,2);
  matrix_set_elt(A,3,4,1);
  matrix_set_elt(A,3,5,3);
  matrix_set_elt(A,4,4,0);
  matrix_set_elt(A,4,5,1);
  matrix_set_elt(A,5,5,2);

  /* Creation du vecteur b */
  b=vector_new();
  for (i=1;i<=N;i++)
    vector_set_elt(b,i,i*i);

  /* Calcul d'un majorant de M par la methode exposee dans l'exercice II */
  for(i=1;i<=N;i++)
    for(j=1;j<=N;j++) {
      e=matrix_get_elt(A,i,j);
      e*=e; /* e^2 */
      e*=e; /* e^4 */
      majorantM+=e;
    }

  /* Calcul du pas alpha (cf devoir 1) */
  alpha=1/majorantM;

  /* Creation de U0 : le premier element de la suite Un */
  U=vector_new();
  for (i=1;i<=N;i++)
    vector_set_elt(U,i,0);
  printf("U0\t=\t");
  vector_print(U);
  printf("\t\tf=%e\n",f(A,b,U));

  /* Le critere d'arret choisi est f(A,b,U)<=epsilon */
  while((f(A,b,U)>epsilon)&&(n<1e6)) {
```

```
    n++;
    grad=gradiant(A,b,U); /* gradient f en U */
    alpha_grad=vector_real_product(alpha,grad); /* alpha * gradient f en U */
    Unext=vector_difference(U,alpha_grad); /* Calcul de U(n+1) */
    vector_free(U);
    vector_free(grad);
    vector_free(alpha_grad);
    U=Unext;
    /* Tous les 1000 termes on affiche la valeur de U et de f */
    if (n%1000==0) {
      printf("U%d\t=\t",n);
      vector_print(U);
      printf("\t\tf=%e\n",f(A,b,U));
    }
  }

  /* Affichage des resultats */
  printf("\nValeur de alpha utilisee : %e\n\n",alpha);
  printf("Solution de Ax=b trouve apres %d iterations :\n\n",n);
  printf("Matrice A =\n");
  matrix_print(A);
  printf("\nSecond membre b = ");
  vector_print(b);
  printf("\n\nSolution x = ");
  vector_print(U);
  printf("\n");

  matrix_free(A);
  vector_free(b);
  vector_free(U);

  return 0;
}
```

Le résultat du programme est le suivant :

```
U0      =       [  0.00   0.00   0.00   0.00   0.00 ]           f=4.895000e+02
U1000   =       [-10.11  -9.39   4.06  -4.12   8.60 ]           f=8.108164e+00
U2000   =       [-12.68 -11.87   3.23  -6.81  11.44 ]           f=1.099862e+00
U3000   =       [-13.54 -12.64   3.03  -8.04  12.44 ]           f=1.629868e-01
U4000   =       [-13.83 -12.88   2.99  -8.59  12.79 ]           f=2.580466e-02
U5000   =       [-13.94 -12.96   2.99  -8.82  12.92 ]           f=4.266598e-03
U6000   =       [-13.98 -12.99   2.99  -8.92  12.97 ]           f=7.240343e-04
U7000   =       [-13.99 -13.00   3.00  -8.97  12.99 ]           f=1.246937e-04
U8000   =       [-14.00 -13.00   3.00  -8.99  13.00 ]           f=2.164967e-05
U9000   =       [-14.00 -13.00   3.00  -8.99  13.00 ]           f=3.775374e-06

Valeur de alpha utilisee : 4.464286e-03

Solution de Ax=b trouve apres 9762 iterations :

Matrice A =
[  2.00  -1.00   1.00   0.00   1.00 ]
[ -1.00   0.00   1.00   0.00  -1.00 ]
```

```
[  1.00   1.00   2.00   1.00   3.00 ]
[  0.00   0.00   1.00   0.00   1.00 ]
[  1.00  -1.00   3.00   1.00   2.00 ]

Second membre b = [  1.00   4.00   9.00  16.00  25.00 ]

Solution x = [-14.00 -13.00   3.00  -9.00  13.00 ]
```

La solution cherchée est donc

$$\begin{cases} x_1 &= -14 \\ x_2 &= -13 \\ x_3 &= 3 \\ x_4 &= -9 \\ x_5 &= 13 \end{cases}$$

On remarquera le nombre élevé d'itérations nécessaire pour obtenir la solution est relativement élevé (9762). Il y a deux raisons à cela.

- La norme $\|\!| \cdot |\!\|_F$ appelée norme de Frobenius est une majoration très large de la norme $\|\!| \cdot |\!\|$ appelée norme spectrale. Le pas α trouvé est donc beaucoup plus petit que ce qu'il est nécessaire d'avoir pour garantir la convergence. Le calcul de la plus grande valeur propre de A^2 donne $\|\!|A|\!\| \simeq 34.57943877$ ce qui conduit à $\alpha \simeq 2.891892e-02$ c'est-à-dire un pas environ 15 plus grand. Le nombre d'itérations avec ce pas est 1504..
- La méthode du gradient fonctionne mal lorsque la plus grande et la plus petite propre de la matrice hessienne A^2 sont très différentes. Il existe des méthodes plus adaptées dites de gradient conjugués pour lesquelles la relation de récurrence $U_{n+1} = U_n - \alpha \nabla f(U_n)$ est remplacée par $U_{n+1} = U_n + \alpha d_n$ où le vecteur $d_0 = -\nabla f(U_0)$ et $d_n = -\nabla f(U_n) + \text{correction}$.