Optimisation des flux d'un réseau métabolique à l'état stationnaire

1 Graphe en 3D, courbes de niveaux d'une fonction de deux variables

On peut tracer le graphe d'une fonction f de deux variables x et y avec z=f(x,y) (persp). On peut aussi tracer ses courbes de niveaux (contour).

Exercice 1.1 *1. Définir la fonction* $f(x_1, x_2) = x_1^2 + x_2^2$ *en ouvrant un script et en tapant les lignes*

```
> carre2D=function(x1,x2){return(x1^2+x2^2)}
```

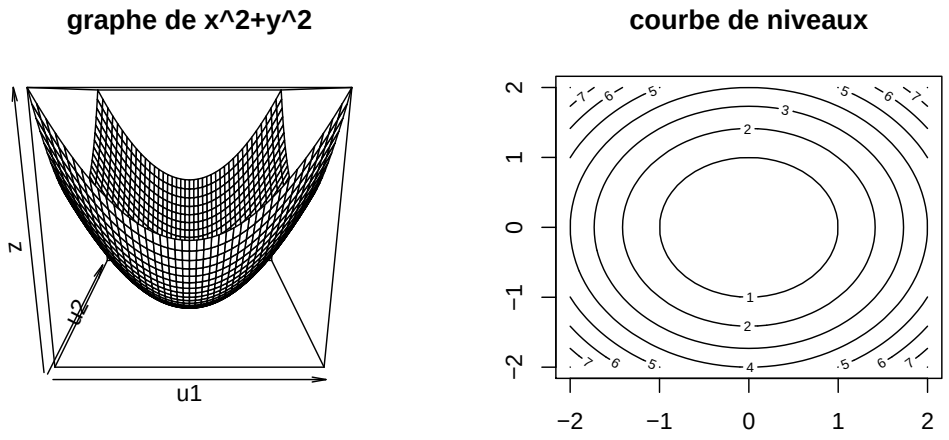
2. Déterminer à la main le minimum de la fonction carre2D.
Il suffit de déterminer l'unique point critique (qui annule les dérivées partielles) de la fonction: c'est $(0, 0)$.

3. Définir les vecteurs u1 et u2 comme suite de -2 à 2 par pas de 0.1

```
> u1<- seq(-2, 2, 0.1)
> u2 <- seq(-2, 2, 0.1)
```

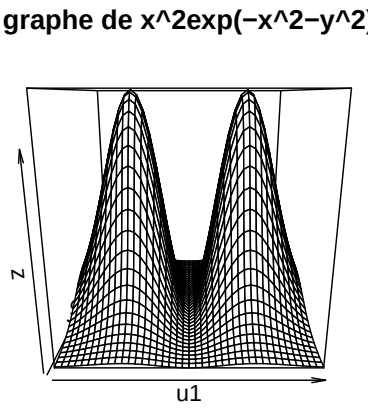
4. Calculer et tracer la fonction carre en tout point de la grille (u1,u2) et les courbes de niveaux correspondantes

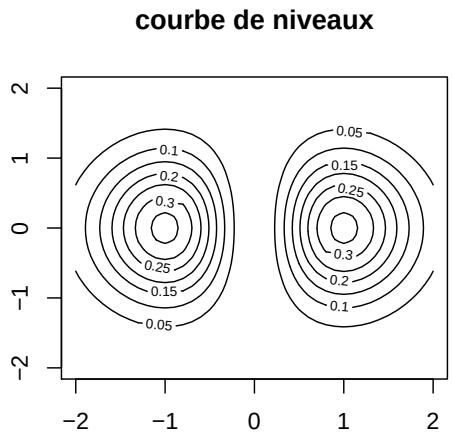
```
> par(mfrow = c(1,2))# pour avoir deux graphiques côte à côte
> z<-outer(u1, u2, carre2D)# z=carre2D(u1,u2)
> persp(u1, u2, z,main="graphe de x^2+y^2")# graphe 3D de carre2D
> contour(u1,u2,z, main="courbe de niveaux")# lignes de niveaux
```



Exercice 1.2 *Mêmes questions avec la fonction* $f(x_1, x_2) = x_1^2 exp(-x_1^2 - x_2^2)$

```
> Montagne2=function(x1,x2){return(x1^2*exp(-x1^2-x2^2))}
> par(mfrow = c(1,2))# pour avoir deux graphiques côte à côte
> z<-outer(u1, u2, Montagne2)
> persp(u1, u2, z,main="graphe de x^2exp(-x^2-y^2)")# graphe 3D de Montagne2
> contour(u1,u2,z, main="courbe de niveaux")# lignes de niveaux
```





2 Optimisation

2.1 Optimisation sans contraintes

On peut faire de l'optimisation de fonctions de plusieurs variables avec le logiciel R. Commençons par de l'optimisation sans contrainte.

Exercice 2.1 *1. Définir pour $x = (x_1, x_2)$, la fonction $carre(x) = x_1^2 + x_2^2$ en ouvrant un script et en tapant les lignes (il s'agit de la même fonction que précédemment mais l'arguement d'entrée est un vecteur)*

```
> carre=function(x){return(x[1]^2+x[2]^2)}
```

2. Définir le vecteur $x0 = (1, 1)$. Tester votre fonction carre en calculant $carre(x0)$.

```
> # initialisation de x0
> x0<-c(1,1)
> carre(x0)

[1] 2
```

3. Que fait la fonction R "optim" (help)?
La fonction optim permet de chercher le minimum d'une fonction. La méthode de recherche du minimum est une méthode dite itérative: on construit une suite $x_k = (x_{k,1}, x_{k,2})$ qui va converger (si la méthode fonctionne) vers le minimum de la fonction. x_0 est le premier terme de la suite (l'initialisation).

4. Tester l'instruction et afficher solution. A quoi correspondent les variables par, value, counts et convergence ? (Help)

```
> solution <- optim(x0, carre)#optim avec algo par défaut Nelder-Mead
> print(solution)

£par
[1] 3.754010e-05 5.179101e-05

£value
[1] 4.091568e-09

£counts
function gradient
      63       NA
```

```
£convergence
[1] 0

£message
NULL
```

La fonction optim utilise par défaut l'algorithme de Nelder-Mead pour chercher le minimum et renvoie

- *par une valeur approchée du minimum de la fonction carré qui est* $(0; 0)$,
- *value la valeur de la fonction carre* $(0; 0) = 0$.
- *counts un vecteur de deux éléments contenant le nombre de fois où carre et son gradient ont été appelés par l'algorithme.*
- *convergence un code: 0 signifie que l'algorithme a bien fonctionné. Pour plus de détails, voir l'aide optim.*

Le résultat obtenu n'est pas très précis et nécessite beaucoup d'itération mais l'algorithme utilisé (Nelder-Mead) fonctionne même avec des fonctions qui ne sont pas dérivables.

5. *Notre fonction étant dérivable, on peut changer l'algorithme de recherche du minimum en spécifiant une autre méthode. Tester et commenter*

```
> solution_meilleure <-optim(x0,carre, NULL, method = "BFGS")# avec la méthode BFGS
> print(solution_meilleure)

£par
[1] -4.263536e-16 -4.263536e-16

£value
[1] 9.087931e-30

£counts
function gradient
       8        3

£convergence
[1] 0

£message
NULL
```

Le résultat est plus précis (plus proche 0) et la convergence plus rapide avec l'algorithme "BFGS" (quasi-Newton).

6. *Quel est le gradient de la fonction carre (à la main)? Définir la fonction gcarre, gradient de la fonction carre (arguments d'entrée x et de sortie un vecteur). Tester et commenter*

```
> gcarre=function(x){ #fonction calculant le vecteur gradient de carre
+ dx=c(2*x[1],2*x[2])
+ return(dx)}

> solution_encore_meilleure <-optim(x0, carre, gcarre, method = "BFGS")# en fournissant le gradient
> print(solution_encore_meilleure)

£par
[1] -2.220446e-16 -2.220446e-16

£value
[1] 2.46519e-32

£counts
```

```
function gradient
       4        3

£convergence
[1] 0

£message
NULL
```

Si on fournit le gradient de la fonction à l'algorithme, la convergence est encore plus rapide. En effet l'algorithme utilise les dérivées de la fonction, si on ne les lui fournit pas explicitement, il les calcule de façon approchée (taux de variation dans les deux directions) et cela nécessite donc un peu plus de temps de calcul.

2.2 Optimisation avec contraintes

Ecriture:

- Tout problème de minimisation d'une fonction $f(x)$ peut s'écrire comme problème de maximisation de $-f(x)$.
- Toute contrainte "$\geq$" peut s'écrire comme une contrainte "$\leq$":

$$a \geq b \Leftrightarrow -a \leq -b$$

- Une contrainte d'égalité peut toujours s'écrire comme deux contraintes d'inégalités:

$$a = b \Leftrightarrow \left\{ \begin{array}{l} a \geq b \\ a \leq b \end{array} \right. \Leftrightarrow \left\{ \begin{array}{r} a \geq b \\ -a \geq -b \end{array} \right.$$

2.2.1 Optimisation d'un problème linéaire avec contraintes

Définition 2.1 *On appelle programme (ou problème) linéaire avec n variables $(x_1, \cdots, x_n)$ et m contraintes:*

$min \sum_{i=1}^{n} c_i x_i$

$$\text{sous les contraintes} \left\{ \begin{array}{r} \sum a_{ji} x_i \leq b_j \\ x_i \in \mathbb{R} \\ i = 1, \cdots, n \\ j = 1, \cdots, m \end{array} \right.$$

Linéarité: la fonction (objectif) à minimiser et les contraintes sont des fonctions linéaires de la variable x. c_i, a_{ji} et b_j sont des constantes fixées.
Exemple de problèmes non linéaires:

- *si la fonction objectif est $x_1^2 + x_2^2$, ce n'est pas une fonction linéaire.*
- *si x_1 doit être entier, le problème n'est pas un programme linéaire*
- *si une contrainte est $x_1 x_2 = 1$.*

Continuité: les variables x_i sont des réels.

Définition 2.2 *On appelle programme linéaire sous forme normale avec n variables $(x_1, \cdots, x_n)$ et m contraintes:*

$min \sum_{i=1}^{n} c_i x_i$

$$\text{sous les contraintes} \left\{ \begin{array}{r} \sum a_{ji} x_i \leq b_j \\ \mathbf{x_i \geq 0} \\ i = 1, \cdots, n \\ j = 1, \cdots, m \end{array} \right.$$

Si on a une variable $x_i \in \mathbb{R}$ (non nécessairement positive), on introduit $x_i^+ \geq 0$ et $x_i^- \geq 0$ et on pose $x_i = x_i^+ - x_i^-$.

Exemple 2.1 *On peut résoudre avec le logiciel R le problème linéaire normal suivant*

$min(x_1 + x_2 + x_3)$

$$\text{sous les contraintes} \left\{ \begin{array}{r} x_1 - x_2 \leq 1 \\ x_1 + x_3 \geq 3 \\ x_1 - x_2 - x_3 = 1 \\ x_1, x_2, x_3 \geq 0 \end{array} \right.$$

On définit les coefficients c_i, a_{ij} et b_j:

- *On définit $c = (1, 1, 1)$ ainsi $x_1 + x_2 + x_3 = \sum_{i=1}^{3} c_i x_i$,*
- *pour que $\sum a_{1i} x_i \leq b_1$ et $\sum a_{2i} x_i \geq b_2$ et $\sum a_{3i} x_i = b_3$), on écrit* $\begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & 1 \\ 1 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} \leq \\ \geq \\ = \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$

```
> library(lpSolve)# chargement libraire lpSolve
> f.obj=c(1,1,1)# fonction objectif
> f.con=matrix(c(1,-1,0,1,0,1,1,-1,-1),nrow=3,byrow=TRUE) # contraintes
> f.dir=c("<=",">=","=")# type contrainte
> f.rhs=c(1,3,1)# second membre
> sol=lp("min",f.obj,f.con,f.dir,f.rhs)# resolution
> print(sol$solution)# affichage solution
```

```
[1] 3 2 0
```

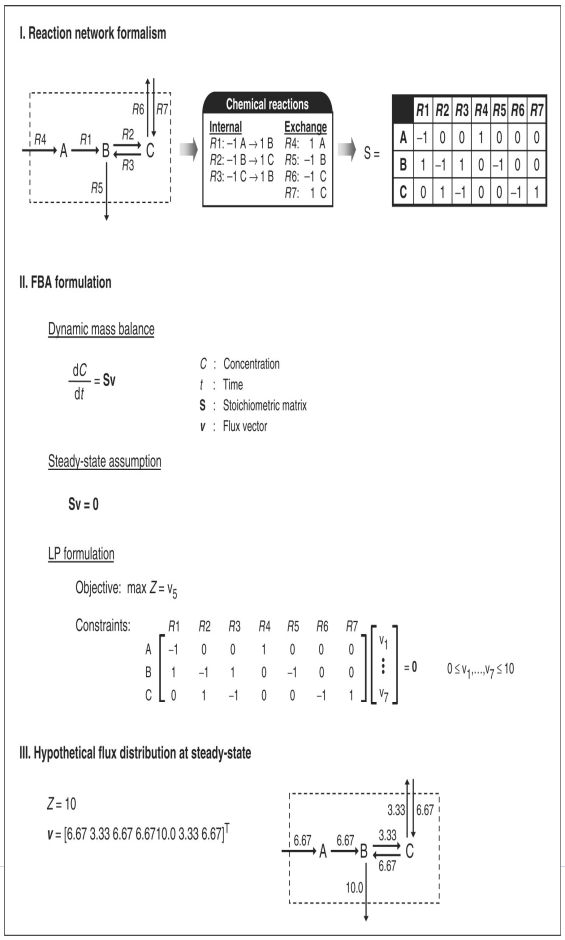
Exercice 2.2 *Résoudre le problème minimum de $x_1 + x_2$ en ajoutant la contrainte $x_2 - x_1 \geq 1$ et $x_1, x_2 \geq 0$ en utilisant le package lpSolve. On définit les coefficients c_i, a_{ij} et b_j:*

- *On définit $c = (1, 1)$ ainsi $x_1 + x_2 = \sum_{i=1}^{2} c_i x_i$,*
- *pour que $x_2 - x_1 \geq 1$, on écrit $\begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} \geq \end{pmatrix} \begin{pmatrix} 1 \end{pmatrix}$*

```
> library(lpSolve)# chargement libraire lpSolve
> # min (x1+x2) sc x2-x1>=1
> library(lpSolve)
> f.obj=c(1,1)# fonction objectif
> f.con=matrix(c(-1,1),nrow=1,byrow=TRUE)# matrice des contraintes
> f.dir=c(">=")# contrainte inégalite
> f.rhs=c(1)#second membre
> sol=lp("min",f.obj,f.con,f.dir,f.rhs)#resolution
> print(sol$solution)
```

```
[1] 0 1
```

Exercice 2.3 *Résoudre et commenter l'encadré ci-dessous.*



```
> f.obj=c(0,0,0,0,1,0,0)# fonction objectif (max v5)
> S = matrix(c(-1,1,0,0,-1,1,0,1,-1,1,0,0,0,-1,0,0,0,-1,0,0,1),
+              nrow  = 3)# matrice de stoechiométrie
> m=7
> Id<-matrix(0,m,m)
> diag(Id)<-rep(1,m)# Id=matrice Identité
> f.con=matrix(c(t(S),Id),nrow=10,byrow=TRUE)# matrice des contraintes SV=0 et Id V<=10
> f.dir=c("=","=","=","<=","<=","<=","<=","<=","<=","<=")# type de contraintes
> f.rhs=c(rep(0,3),rep(10,7))# second membre
> sol=lp("max",f.obj,f.con,f.dir,f.rhs)# resolution
> print(sol$solution)

[1] 10  0  0 10 10  0  0
```

Ce problème est mal posé. En effet, il existe une infinité de solutions vérifiant les contrainte, par exemple, $v = (10, 0, 0, 10, 10, 0, 0)$ *ou encore* $v = (7, 2, 5, 7, 10, 4, 7)$. *Dans l'encadré, le titre du graphe est d'ailleurs "hypothetical flux distribution at steady state".*

2.2.2 Optimisation d'un problème quadratique avec contraintes

Exercice 2.4 *On cherche maintenant à résoudre le problème précédent minimum de carre$(x) = x_1^2 + x_2^2$ en ajoutant la contrainte $x_2 - x_1 \geq 1$. Il existe une fonction ContrOptim prédéfinie sous R pour résoudre les problèmes d'optimisation avec contraintes d'inégalités. Il suffit d'écrire les contraintes sous forme matricielle*

1. *Ecrire $x_2 - x_1 \geq 1$ sous forme matricielle $A \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq b$. Définir A et b.*

```
> # avec contrainte y-x>=1 sous forme matricielle
> A      <- matrix(c(-1,1), 1,2)
> b      <- c(1)
```

2. *Tester et commenter*

```
> sol_ineq=constrOptim(c(1,3), carre, gcarre, ui = A, ci = b)
> print(sol_ineq)

£par
[1] -0.5  0.5

£value
[1] 0.5

£counts
function gradient
     191       41

£convergence
[1] 0

£message
NULL

£outer.iterations
[1] 3

£barrier.value
[1] 1e-04
```

Le logiciel renvoie

- *par une valeur approchée du minimum de la fonction carré sous la contrainte $x_2 - x_1 \geq 1$ qui est donc $(-0,5; 0,5)$,*
- *value la valeur de la fonction est carre$(-0,5; 0,5) = 0,5$.*
- *counts un vecteur de deux éléments contenant le nombre de fois où carre et son gradient ont été appelés par l'algorithme.*
- *convergence un code: 0 signifie que l'algorithme a bien fonctionné. Pour plus de détails, voir l'aide optim ou ConstrOptim.*

3. *Tester la commande*

```
> constrOptim(c(1,2), carre, gcarre, ui = A, ci = b)#
```

Que signifie la réponse du logiciel ? La fonction constrOptim cherche à déterminer le minimum sous contrainte de manière itérative c'est-à-dire en construisant d'une suite de valeurs $x_n = (x_{1,n}, x_{2,n})$. Le vecteur c(1,2) est la valeur initiale de la suite x_0, valeur initiale qui doit vérifier les contraintes.

L'algorithme de constrOptim a besoin d'une initialisation à l'intérieur de l'ensemble des contraintes. Parfois l'ensemble des contraintes est d'intérieur vide et la fonction contrOptim ne peut donc pas être utilisée. Si la fonction à optimiser est quadratique définie positive(pouvant s'écrire ${}^t xAx$ avec A matrice définie positive), on peut utiliser la fonction solve.QP du package quadprog.

Exercice 2.5 *On cherche maintenant à résoudre le problème précédent minimum de* $carre(x) = x_1^2 + x_2^2$ *en ajoutant la contrainte* $x_2 - x_1 = 1$. *Une contrainte d'égalité peut toujours s'écrire comme deux contraintes d'inégalités, car* $a = b \Leftrightarrow \left\{ \begin{array}{l} a \geq b \\ a \leq b \end{array} \right. \Leftrightarrow \left\{ \begin{array}{r} a \geq b \\ -a \geq -b \end{array} \right.$.
Donc on a

$$x_2 - x_1 = 1 \Leftrightarrow \left\{ \begin{array}{l} x_2 - x_1 \geq 1 \\ {}_2 - x_1 \leq 1 \end{array} \right. \Leftrightarrow \left\{ \begin{array}{r} x_2 - x_1 \geq 1 \\ -x_2 + x_1 \geq -1 \end{array} \right.$$

Mais ici l'algorithme a besoin d'une initialisation à l'intérieur de l'ensemble des contraintes. Notre ensemble est ici d'intérieur vide et la fonction contrOptim ne peut donc pas être utilisée.

Pour chercher le minimum de notre fonction, on va utiliser la fonction solve.QP. Elle nécessite le package quadprog.

1. *Charger le package quadprog et afficher l'aide de solve.QP*

```
> library(quadprog)
```

2. *Définir les matrices et vecteurs suivants*

$$D = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad d = \begin{pmatrix} 0 \\ 0 \end{pmatrix} A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

```
> D<-matrix(c(1,0,0,1),ncol=2)
> d<-matrix(c(0,0),ncol=1)
> A<-matrix(c(1,-1,-1,1),ncol=2)
> b<-matrix(c(-1,1),ncol=1)
```

3. *Calculer à la main* ${}^t xDx - {}^t dx$. *Comparer avec la fonction carre.*

$${}^t xDx - {}^t dx = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1^2 + x_2^2$$

On retrouve la fonction carré.

4. *La commande*

```
solve.QP(D,d,t(A),b)
```

détermine le minimum de $f(x)$ *où* $f(x) = \frac{1}{2} {}^t xDx - {}^t dx$ *sous la contrainte* $Ax \geq b$.
Déterminer le minimum de $carre(x) = x_1^2 + x_2^2$ *sous la contrainte* $x_2 + x_1 = 1$.

```
> sol=solve.QP(2*D,d,t(A),b)
> print(sol)

£solution
[1] -0.5  0.5

£value
[1] 0.5

£unconstrained.solution
[1] 0 0

£iterations
[1] 2 0

£Lagrangian
```

```
[1] 0 1

$iact
[1] 2
```

Exercice 2.6 *Déterminer le minimum de* $f(x) = x_1^2 + x_2^2 + x_3^2$ *sous les contraintes* $x_2 + x_1 = 1$ *et* $x_1 \leq x_3$ *en utilisant solve.QP*
Soit D la matrice identité en dimension 3 et d le vecteur nul colonne de dimension 3. On peut écrire f

$$ {}^t x D x - {}^t d x = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = x_1^2 + x_2^2 + x_3^2 $$

Ecrivons les contraintes sous forme d'inégalités

$$ \begin{cases} x_1 + x_2 = 1 \\ x_1 \leq x_3 \end{cases} \Leftrightarrow \begin{cases} x_1 + x_2 \geq 1 \\ x_1 + x_2 \leq 1 \\ x_1 - x_3 \leq 0 \end{cases} \Leftrightarrow \begin{cases} x_1 + x_2 \geq 1 \\ -x_1 - x_2 \geq -1 \\ -x_1 + x_3 \geq 0 \end{cases} $$

puis sous forme matricielle $Ax \geq b$ *en posant* $A = \begin{pmatrix} 1 & 1 & 0 \\ -1 & -1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ $b = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$.

```
> D<-matrix(c(1,0,0,0,1,0,0,0,1),ncol=3)
> d<-matrix(c(0,0,0),ncol=1)
> A<-matrix(c(1,-1,-1,1,-1,0,0,0,1),ncol=3)
> b<-matrix(c(1,-1,0),ncol=1)
> sol=solve.QP(2*D,d,t(A),b)
> print(sol)

$solution
[1] 0.3333333 0.6666667 0.3333333

$value
[1] 0.6666667

$unconstrained.solution
[1] 0 0 0

$iterations
[1] 3 0

$Lagrangian
[1] 1.3333333 0.0000000 0.6666667

$iact
[1] 1 3
```

Le résultat est donc $x = (1/3, 2/3, 1/3)$.

Exercice 2.7 *Soit la fonction* $f(x) = x_1^2 - 2x_2^2 + 3x_3^2 + 4x_1x_2 + 6x_1x_3$. *Déterminer la matrice A (symétrique) telle que* $f(x) = {}^t(x)Ax$. *Est-elle définie positive ?*

$$ A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & -2 & 0 \\ 3 & 0 & 3 \end{pmatrix} $$

A n'est pas définie positive ($f(0,1,0) < 0$).

2.3 Optimisation des flux d'un réseau métabolique à l'état stationnaire

Exercice 2.8 *On considère le réseau métabolique 2. Sa matrice de stoechiométrie est*

```
> S = matrix(c(1,-1,0,-1,0,0,0,1,-1,0,1,0,0,0,0,1,-1,-1),
+   nrow  = 3,
+   byrow = TRUE)# matrice de stoechiométrie
> rg=qr(S)$rank # rang de S
> p=dim(S)[2]# nombre de colonnes de S
> dk=p-rg #dimension du noyau
```

1. *Résoudre le problème suivant:*
   $min||V||^2$
   tel que $SV = 0$
   sous les contraintes $v_1 = 1$
   $v_i \geq 0$ *pour* $1 \leq i \leq 6$.

   (a) *Existence et unicité de la solution: Avant tout calcul d'optimisation, il faut se poser la question de l'existence et de l'unicité de la solution.*
   Ici le noyau de S *est de dimension 3. On fixe* v_1 *alors l'ensemble des contraintes* $E = \{V \in \mathbb{R}^6; v_1 = 1; SV = 0\}$ *n'est pas vide. De* $SV = 0$ *et* $v_1 = 1$ *on tire* $S_c v_c = -S_m v_1$ *où* $v_c =^t (v_2, v_3, v_4, v_5, v_6)$ *et*

$$S_c = \begin{pmatrix} -1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 \end{pmatrix} \quad S_m = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

```
> Sc=S[,-1]# Sc :suppression de colonne 1 de  S
> bm=-S[,1]; # second membre -Sm*v1
> m=5#nombre d'inconnues
> Sa=matrix(c(Sc,bm),ncol=m+1)# matrice augmentée
> rgSc=qr(Sc)$rank# rang de Sc
> # si rg(Sc)=rg(Sa) alors il existe des solutions sinon pas de solution
> if (rgSc==qr(Sa)$rank) print("rang Sm et Sm augmentée égaux, existence de solutions") else ("aucun
[1] "rang Sm et Sm augmentée égaux, existence de solutions"
> if (m==rgSc) print("unique solution") else print("infinité de solutions vérifiant les contraintes"
[1] "infinité de solutions vérifiant les contraintes"
```

   D'après le théorème de Rouché-Fontené, il existe une infinité de solutions v_c *vérifiant le système.*
   On cherche parmi ces solutions celle(s) qui minimise(nt) la fonction $f(v_c) = \sum_{i=2\cdots 6} v_i^2$. *Cette fonction est strictement convexe, cela prouve l'unicité de la solution.*

   (b) *Résoudre le problème avec le logiciel R. Nous allons utiliser la fonction solve.QP de R qui déterminer* x *le minimum de* $f(x) = 1/2t(x) * D * x - t(d)x$ *sous les contraintes exprimées par* $t(A)x \geq b$. v_1 *étant fixé, notre vecteur inconnu est* x *est* $(v_2, \cdots, v_6)$. *On renomme* $(v_2, \cdots, v_6)$ *par* $x_1, \cdots, x_5$). *Les contraintes:*
   Sous forme d'inégalités, cela donne $S_c x \geq -S_m v_1$ *et* $-S_c x \geq S_m v_1$.
   On ajoute aussi la positivité de toutes réactions, matricellement cela s'écrit $I_5 x \geq 0_{\mathbb{R}^5}$ *(I_5 est la matrice identité d'ordre 5).*
   La fonction à minimiser
   $f(x) = \sum_{i=1,\cdots 5} x_i^2 = t(x) I_5 x$.

```
> # Optimisation min norm(V) avec constraintes  SV=0 and v1=1
> #vi >= 0  (>= signifie supérieur ou égal)
> require(quadprog)
> m=5# nombre d'inconnues
> # fonction à minimiser f(x)=sum xi^2 où x=(v2,...,v6)
> D<-matrix(0,m,m)
> diag(D)<-2*rep(1,m)# D=2*matrice Identité
> d=rep(0,m)# d=vecteur nul
> # ainsi f(x)=1/2 t(x)*D*x -t(d)x=sum xi^2
```

```
>
> # Les  contraintes écrites sous la forme Ax geq b
> # etat stationnaire et v1=1: Sc x<=bm et - Sc x>=-bm avec bm=-Sm v1
> # positivité des vitesses: Identité*V>=0
> # matrice des contraintes
> #  (Sc )
> #A=(-Sc)
> #  ( D)
> A<- matrix(c(t(Sc),t(-Sc),D),ncol=m,byrow=TRUE)
> bm=-S[,1]; # second membre -Sm*v1
> b<- c(bm,-bm,rep(0,m));# second membre:  bm+(-bm)+5 zeros
> #résolution attention: formalisme attendu par solve.QP contraintes t(A)x geq b
> sol  <- solve.QP(D, d, t(A), b, meq=0)
> print(sol)

$solution
[1] 5.000000e-01 5.000000e-01 5.000000e-01 2.775558e-17
[5] 5.000000e-01

$value
[1] 1

$unconstrained.solution
[1] 0 0 0 0 0

$iterations
[1] 4 0

$Lagrangian
 [1] 0 0 0 2 1 1 0 0 0 0 0

$iact
[1] 4 5 6
```

La solution affichée correspond à $v_c = (v_2, \cdots, v_6)$.

(c) *En se ramenerant à un problème à deux variables* (v_2, v_5)*, résoudre le problème à la main.*

On exprime les contraintes en fonction de (v_2, v_5)*: De* $SV = 0$ *et* $v_1 = 1$*, on tire* $\begin{cases} v_4 = 1 - v_2 \\ v_3 = v_2 + v_5 \\ v_6 = 1 - v_2 - v_5 \end{cases}$ *.*
$f(v) = \sum_{i=1\cdots6} v_i^2 = 1^2 + v_2^2 + (v_2 + v_5)^2 + (1 - v_2)^2 + v_5^2 + (1 - v_2 - v_5)^2 = g(v_2, v_5)$*. On a ainsi g, fonction de deux variables, dont on sait calculer l'unique point critique* $\frac{\partial g}{\partial v_2} = 8v_2 + 4v_5 - 4 = 0$ *et* $\frac{\partial g}{\partial v_5} = 4v_2 + 6v_5 - 2 = 0$*. L'unique point critique* $(v_2 = \frac{1}{2}, v_5 = 0)$ *est le minimum de g, fonction strictement convexe (sa Hessienne est toujours définie positive). A l'aide du système, on tire* $v_3 = \frac{1}{2}, v_4 = \frac{1}{2}, v_6 = \frac{1}{2}$.

2. *Le problème suivant:*
   max v_3
   tel que $SV = 0$
   sous les contraintes $v_1 = 1$ *admet-il une unique solution ?*
   $v_i \geq 0$ *pour* $1 \leq i \leq 6$

   Avant tout calcul d'optimisation, il faut se poser la question de l'existence et de l'unicité de la solution.

   D'après le théorème de Rouché-Fontené, il existe une infinité de solutions v_c *vérifiant le système* $S_c v_c = S_m v_1$.
   On cherche parmi ces solutions celle(s) qui maximise(nt) la fonction $f(v) = v_3$ *ou qui minimise(nt)* $f(v) = -v_3$.
   Cette fonction n'est pas strictement convexe, donc rien ne prouve l'unicité: ici, il existe une infinité de solutions: $V = t(1, 1, 1, 0, 0, 0)$, $V = t(1, 0, 1, 1, 1, 0)$,...
   On dit que le problème est mal posé. Une implémentation avec un logiciel pourra dans le meilleur des cas trouver une solution parmi l'infinité de solutions.

De $SV = 0$ et $v_1 = 1$, on tire

$$\begin{cases} v_4 = 1 - v_2 \\ v_3 = v_2 + v_5 \\ v_6 = 1 - v_2 - v_5 \end{cases}$$

On cherche à maximiser $v_3 = v_2 + v_5$ tout en gardant toutes les vitesses positives. Le problème se ramène à
$max v_2 + v_5$
avec les contraintes
$0 \leq v_2 \leq 1$
$0 \leq v_2 + v_5 \leq 1$

Par conséquent, toute solution s'écrit $V = t(1, a, 1, 1-a, 1-a, 0)$ avec $a \in [0; 1]$.

```
> library(lpSolve)
> S = matrix(c(1,-1,0,-1,0,0,0,1,-1,0,1,0,0,0,0,1,-1,-1),
+   nrow  = 3,
+   byrow = TRUE)# matrice de stoechiométrie
> f.obj=c(0,0,1,0,0,0)
> f.con=matrix(c(t(S),c(1,rep(0,5))),nrow=4,byrow=TRUE)
> f.dir=c("=","=","=","=")
> f.rhs=c(rep(0,3),1)
> sol=lp("max",f.obj,f.con,f.dir,f.rhs)
> print(sol$solution)

[1] 1 1 1 0 0 0
```

Exercice 2.9 *Reprendre le réseau métabolique suivant*

$$\begin{array}{lrcl} \text{R1}: & \text{A}_{\text{ext}} & \longrightarrow & \text{A} \\ \text{R2}: & \text{B}_{\text{ext}} & \longleftrightarrow & \text{B} \\ \text{R3}: & \text{P} & \longrightarrow & \text{P}_{\text{ext}} \\ \text{R4}: & \text{A} & \longrightarrow & \text{B} \\ \text{R5}: & \text{A} & \longrightarrow & \text{C} \\ \text{R6}: & \text{A} & \longrightarrow & \text{D} \\ \text{R7}: & \text{B} & \longleftrightarrow & \text{C} \\ \text{R8}: & 2\text{B} & \longrightarrow & \text{P} \\ \text{R9}: & \text{C} + \text{D} & \longrightarrow & \text{P} \end{array}$$

1. *Lire le fichier RM1.csv dans la variable s. Transformer la data frame s en matrice S*

```
> s <- read.csv2("RM1.csv")
> S<-as.matrix(s,nrow=5,ncol=9)
```

2. *Résoudre le problème suivant:*
$min||V||^2$
tel que $SV = 0$
sous les contraintes $v_1 = v_2 = 1$.
$v_i \geq 0$ *pour* $1 \leq i \leq 9$ *et* $i \neq 7$.
Vérifier que le problème admet une solution puis le résoudre avec R. On verifiera ses résultats (à la main) en exprimant la fonction objectif en fonction des variables (v_7, v_9) puis on cherchera son minimum.

Plusieurs écritures du problème sont possibles, en voici une: de $SV = 0$ et $v_1 = v_2 = 1$, on tire $S_c v_c = -S_m v_m$. On vérifie que ce système admet bien une infinité de solutions.
En effet, si le système n'admet aucune solution, on s'arrête: il n'y a pas de solution au problème de minimisation ! Si le système $S_c v_c = -S_m v_m$ n'admet qu'une unique solution, on s'arrête aussi: la solution de $S_c v_c = -S_m v_m$ est la solution du problème de minimisation (la région admissible ne contient qu'un élément c'est donc le plus petit!).

```
> # Optimisation min norm(V) avec constraintes  SV=0 and v1=v2=1
> #vi >= 0 excepté v7 (=> signifie supérieur ou égal)
> # etat stationnaire et v1=1 v2=1: Sc x >=bm et -ScVc>=-bm
```

```
> # avec bm=-SmVm
> Sc=S[,-c(1,2)]# Sc :suppression des colonnes 1 et 2 de S
> Sm=S[,c(1,2)]# Sm colonnes 1 et 2 de S
> Vm=matrix(c(1,1),ncol=1);bm=-Sm%*%Vm
> # test si l'ensemble Sc x =bm n'est pas vide: existe-t-il des solutions ?
> rgSc=qr(Sc)$rank#rang de Sc
> m=7
> Sa=matrix(c(Sc,bm),ncol=m+1)# matrice augmentéé
> #Si rg(Sc)=rg(Sa) alors il existe des solutions sinon pas de solution
> if (qr(Sa)$rank==rgSc) print ("il existe des solutions") else print("pas de solution")

[1] "il existe des solutions"
```

Le rang de S_c est 5 et le nombre de variables est 7, donc le système $S_c v_c = -S_m v_m$ admet une infinité de solutions. Cet ensemble est notre région admissible qui est non vide (on admet qu'il est convexe et que la fonction à minimiser est strictement convexe, le problème est donc bien posé ie admet une unique solution). On va donc chercher le vecteur v_c de norme minimum dans cet ensemble avec solve.QP:

Notre inconnue est maintenant $v_c = t(v_3, v_4, v_5, v_6, v_7, v_8, v_9)$ qui est de dimension 7. On veut minimiser $f(v_c) = \sum_{i=3}^{9} v_i^2$. Les contraintes s'écrivent:

$$\begin{cases} S_c v_c \geq -S_m v_m \\ -S_c v_c \geq S_m v_m \\ v_i \geq 0 \quad \textit{pour } i \neq 7 \end{cases}$$

On définit donc la matrice et le second membre des contraintes

$$A = \begin{pmatrix} S_c \\ -S_c \\ \tilde{I}_7 \end{pmatrix} \quad b = \begin{pmatrix} 0_{\mathbb{R}^17} \end{pmatrix}$$

où $\tilde{I}_7$ est la matrice identité de dimension 7 sauf la 5ème ligne qui est composée uniquement de 0.

On définit D la matrice identité de dimension 7 et d la matrice colonne nulle de dimension 7. On va minimiser la fonction $f(x) = \frac{1}{2}{}^t x D x - {}^t dx$ où $x = t(x_1, \cdots, x_7)$ correspond à $v_c = t(v_3, v_4, v_5, v_6, v_7, v_8, v_9)$.

```
> require(quadprog)
> # fonction à minimiser f(x)=sum xi^2  où x=(v3,...,v9)
> #Les  contraintes écrites sous la forme Ax >= b
> m=7 # nombre d'inconnues
> D<-matrix(0,m,m)
> diag(D)<-rep(1,m)# D=matrice Identité 7x7
> d=rep(0,m)# d=vecteur nul
> # ainsi f(x)=1/2 t(x)*D*x -t(d)x=sum xi^2
>
> # positivité des vitesses: Identité*V>=0 sauf ligne 5 (correspondant à v7)
> Itilde=D;Itilde[5,5]=0# Itilde=matrice identité 7x7 avec l'élement (5,5) remplacé par 0
> # matrice des contraintes
> #  (Sc )
> #A=(-Sc)
> #  ( Itilde)
> A<- matrix(c(t(Sc),t(-Sc),Itilde),ncol=m,byrow=TRUE)
> # second membre:  bm+(-bm)+7 zeros
> b<- c(bm,-bm,rep(0,m));
> #résolution attention: formalisme attendu par solve.QP contraintes t(A)x geq b
> sol  <- solve.QP(2*D, d, t(A), b, meq=0)
> print(sol)

£solution
[1] 1.00000000 0.26666667 0.33333333 0.40000000 0.06666667
```

```
[6] 0.60000000 0.40000000

$value
[1] 1.866667

$unconstrained.solution
[1] 0 0 0 0 0 0 0

$iterations
[1] 6 0

$Lagrangian
 [1] 0.000000 0.000000 0.000000 0.000000 0.000000 2.133333
 [7] 1.600000 1.466667 1.333333 2.000000 0.000000 0.000000
[13] 0.000000 0.000000 0.000000 0.000000 0.000000

$iact
[1]  6  7  8 10  9
```

A la main, de $SV = 0$ *et* $v_1 = v_2 = 1$*, on tire*

$$\begin{cases} v_6 = v_9 \\ v_5 = v_9 - v_7 \\ v_4 = 1 - 2v_9 + v_7 \\ v_8 = 1 - v_9 \\ v_3 = 1 \end{cases}$$

donc $f(v) = \sum_{i=1\cdots 9} v_i^2 = 3 + (1 - 2v_9 + v_7)^2 + (v_9 - v_7)^2 + v_9^2 + v_7^2 + (1 - v_9)^2 + v_9^2$ *dont les dérivées partielles* $\frac{\partial f}{\partial v_7} = 6v_7 - 6v_9 + 2$ *et* $\frac{\partial f}{\partial v_9} = -6v_7 + 16v_9 - 6$ *s'annulent en* $(v_7 = \frac{1}{15}, v_9 = \frac{2}{5})$*, unique minimum (la hessienne est définie positive). Avec le système, on retrouve* $V = t(1, 1, 1, \frac{4}{15}, \frac{1}{3}, \frac{2}{5}, \frac{1}{15}, \frac{3}{5}, \frac{2}{5})$.

3. *Peut-on résoudre le problème suivant ?:*
   max v_3
   tel que $SV = 0$
   sous les contraintes $v_1 = v_2 = 1$.
   $v_i \geq 0$ *pour* $1 \leq i \leq 9$ *et* $i \neq 7$.

Oui, on peut le résoudre mais ce problème n'est pas bien posé: il admet une infinité de solutions. En utilisant le package lpSolve, on peut trouver une solution: on commence par dupliquer les réactions réversibles pour obtenir un problème où toutes les variables cherchées sont positives.

```
> s <- read.csv2("RM1.csv")
> S<-as.matrix(s,nrow=5,ncol=9)
> # on duplique les vitesses réversibles
> # v1,v2+,v2-,v3,v4,v5,v6,v7+,v7-,v8,v9
> # nommer w1 à w11
> S2=matrix(c(S[,1:2],-S[,2],S[,3:7],-S[,7],S[,8:9]),ncol=11)# nouvelle matrice de stoechio
> f.obj=c(rep(0,3),1,rep(0,7))# max de w4 (ancien v3)
> con1=c(1,rep(0,10))# contrainte sur w1
> con2=c(0,1, rep(0,9))# contrainte sur w2
> f.con=matrix(c(t(S2),con1,con2),nrow=7,byrow=TRUE)#matrice des contraintes
> f.dir=c(rep("=",7))# contraintes d'égalité
> f.rhs=c(rep(0,5),1,1)# second membre
> sol=lp("max",f.obj,f.con,f.dir,f.rhs)#résolution
> print(sol$solution)# (sous forme w)

 [1] 1 1 0 1 1 0 0 0 0 1 0
```

```
> V=c(sol$solution[1],sol$solution[2]-sol$solution[3],sol$solution[4:7],sol$solution[8]-sol$solution[9],sol$s
> print(V)# retour à V : v2=w2-w3 et v7=w8-w9

[1] 1 1 1 1 0 0 0 1 0
```