Séance 5 : Régression Non Linéaire avec R - Modèle d'équations différentielles

Exercice 1 *On veut résoudre numériquement l'équation différentielle* $y'(t) = 2y(t)$ *avec condition initiale* $y(0) = 1$ *sur [0,5]. La résolution d'équations différentielles sous R nécessite l'installation du package "deSolve".*

1. *Charger le package deSolve.*

2. *Définir la fonction* f *et la tester en calculant* $f(0, 1, 2)$.

```
> library(deSolve)
> f<- function(t, y, a){
+ out=a*y
+ return(list(out))}
```

La variable t n'apparaît pas dans la fonction mais il faut absolument la mettre dans la définition de f pour résoudre l'équation différentielle (Help ode).

3. *Rappeler la solution de l'équation différentielle* $y' = 2y$ *avec condition initiale* $y(0) = 1$ *? (à la main).*

4. *Définir le vecteur "temps" suite de 0 à 5 par pas=0.02. Définir y0 égal à la valeur 1 (la condition initiale* $y(0) = 1$*).*

```
> temps=seq(0,5,0.02)
> y0=1
```

5. *Résoudre l'équation différentielle numériquement avec la commande ode (Ordinary Differential Equation).*

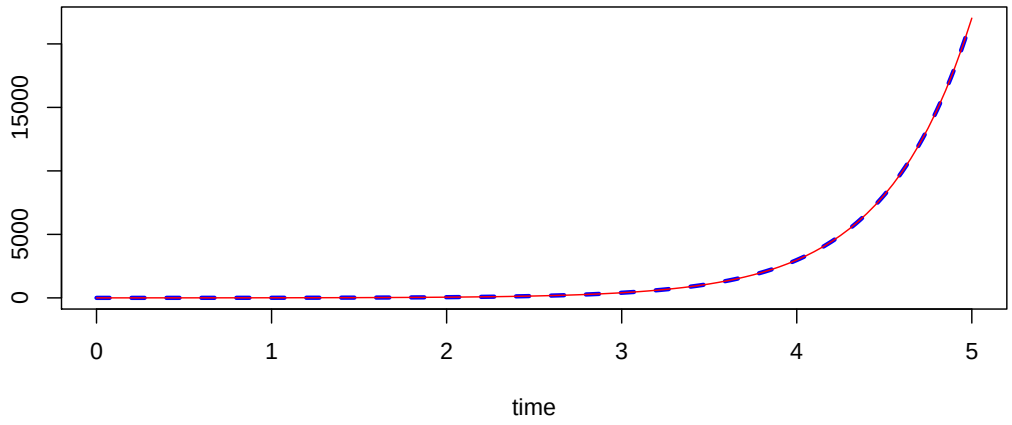
```
> sol <- ode(y = y0, times = temps, func = f, parms = 2)
> # ode=commande résolvant l'ED
> # Elle prend en paramètres: y, times, func, parms.
> # y=y0 condition initiale
> # times =temps (abscisse) pour lesquels on va calculer la solution approchée
> # y(0), y(0.02), y(0.04),...,y(5)
> # func=f telle que y'=f(t,y,a)
> # parms=paramètre(s) éventuel(s) de la fonction (ici le paramètre a de f prendra la valeur 2)
```

6. *Afficher la matrice sol (que contient-elle ?)*

```
> print(sol[1:4,])# les 4 premieres lignes

     time        1
[1,] 0.00 1.000000
[2,] 0.02 1.040812
[3,] 0.04 1.083289
[4,] 0.06 1.127498
```

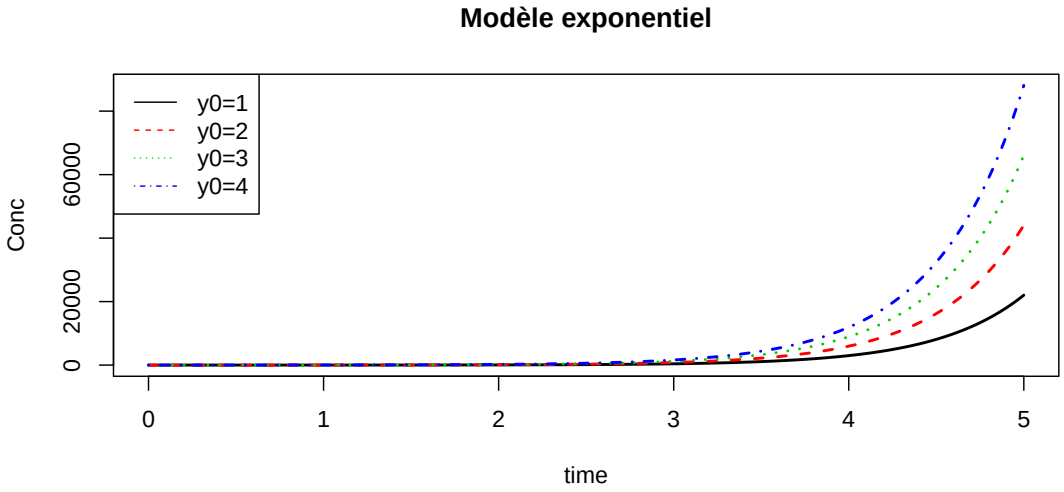
7. *Tracer la solution sol en fonction de temps (en pointillé et en bleu et l'option lwd=3) et la solution exacte de l'équation différentielle (trait plein et en rouge).*

```
> plot(sol,lty=2,col="blue",lwd=3)#il faudrait ajouter un titre, des labels aux axes,...
> curve(exp(2*x),0,5,add=TRUE,col="red",lty=1)
```



8. *Tester et comprendre les commandes ci-dessous*

```
> sol <- ode(y = c(1,2,3,4), times = temps, func = f, parms = 2)
> matplot(sol[ , 1], sol[ , 2:5], type = "l", xlab = "time", ylab = "Conc",
+         main = "Modèle exponentiel", lwd = 2)
> legend("topleft", c("y0=1", "y0=2","y0=3","y0=4"), col = 1:4, lty = 1:4)
```



Exercice 2 Modélisation de la croissance d'un champignon

On étudie la croissance d'un champignon microscopique que l'on trouve dans les sols du genre fusarium [1]. *Les données expérimentales sont les suivantes (t représente le temps et y la densité de champignons):*

t_i	0	2	4	6	9	11	13	16	18	20	23	27	31
y_i	0.79	4.89	5.12	6.7	27.3	34	37.6	64.9	71.9	85.2	95	98	97.4

On suppose que la fonction y suit une croissance dite logistique c'est-à-dire vérifie l'équation différentielle

$$\begin{cases} y'(t) = ry(t)(1 - y(t)/K) \\ y(0) = y_0 \end{cases}$$

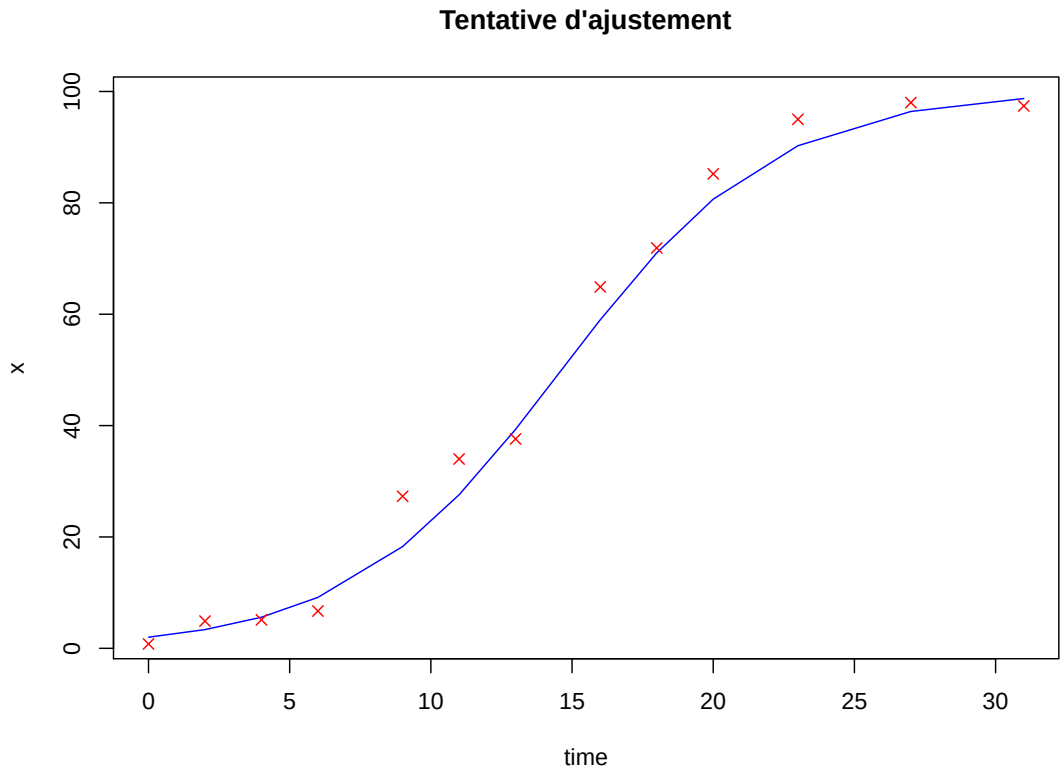
Sans calculer la solution explicite de l'équation différentielle, on veut déterminer les paramètres (y_0, r, K) *connaissant seulement le modèle théorique vérifié par* $y'(t)$.

[1] extrait du livre de A. Pavé, Modélisation en biologie et en écologie, 1997

1. *Télécharger et lire le fichier de données champignon.csv. Tracer le nuage de points* (t_i, y_i) *(marqueur "triangle").*

2. *Résoudre numériquement l'équation différentielle c'est-à-dire évaluer* $y_{th}(t_i)$ *pour* $i = 1, \cdots, 13$ *(valeur de* t_i *du tableau de données) en prenant pour* $(y_0, r, K) = (2, 0.266, K = 100)$.

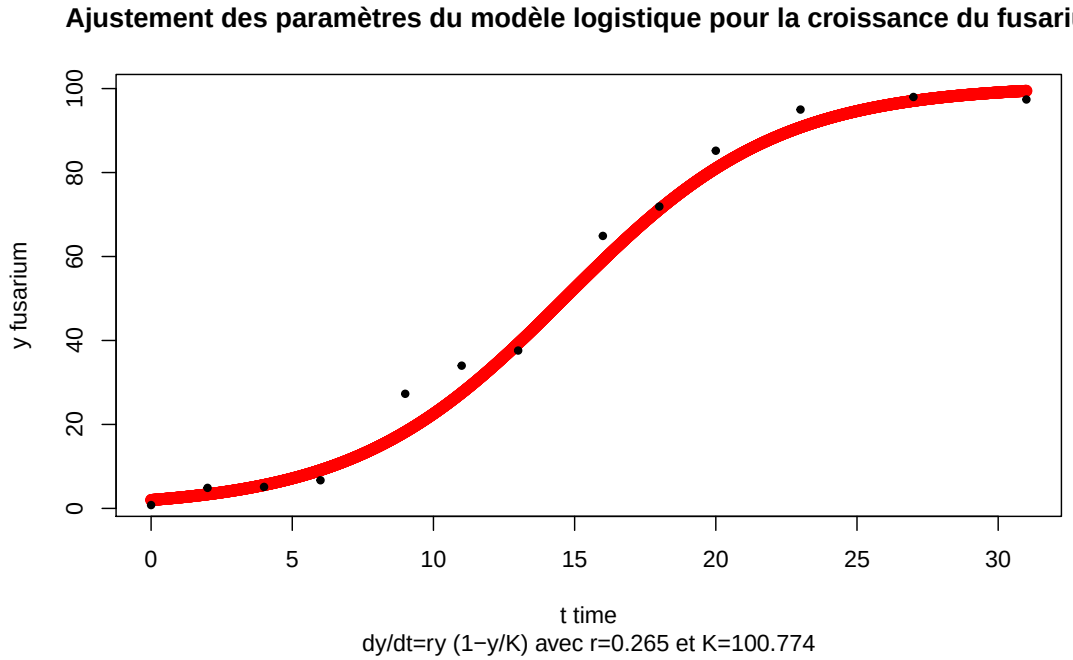
```
> #Croissance d'un champignon x'(t)=f(x(t))
> # la croissance est supposée ici logistique
>
> #librairie pour résoudre les ED
> library(deSolve)
> #on définit la fonction logistic ici (avec deux paramètres r et K)
> logistic<-function(t,x,parms){
+    dx=x*parms["r"]*(1-x/parms["K"])
+   return(list(dx))
+ }
> #on définit les paramètres pour la simulation du système
> parms<-c(r=0.266, K=100,x0=2)
> #temps des observations et les mesures
> tobs<-c(0,2,4,6,9,11,13,16,18,20,23,27,31)
> xobs<-c(0.79,4.89,5.12,6.70,27.3,34,37.6,64.9,71.9,85.2,95,98,97.4)
> # condition initiale
> x0<- xstart <-(x = 2)
> #on résout le système x'(t)=f(x(t)) avec la fonction ode
> # cela calcule x(ti) pour les valeurs ti fournies
> # ode(condition initiale, ti, fonction f, paramètres éventuels de f)
> out<-as.data.frame(ode(c(x=x0), tobs, logistic, parms))
> # out contient une matrice (ti, x(ti))
>
> # toujours regarder le résultat pour détecter les incohérences
> plot(out, type="l",col='blue',main="Tentative d'ajustement")
> points(tobs,xobs,col="red",pch=4)#ajout des points exp.
```



3. *Chercher les paramètres (y_0, r, K) tels que les points $(t_i, y_{th}(t_i))$ approche au mieux au sens des moindres carrés le nuage de points expérimentaux (t_i, y_i). On utilisera la fonction nls. Tracer la solution de l'équation différentielle.*

```
> #on ajuste le modèle sur les données xobs par les moindres carrés
> # pour cela on utilise la fonction nls, il faut faire attention aux paramètres initiaux
> # ode(c(x=x0),times,logistic,c(r=r, K=K)) renvoie les couples (ti,x(ti))
> # ode(c(x=x0),times,logistic,c(r=r, K=K))[,2] renvoie les x(ti) seulement
> fit<-nls(xobs~ode(c(x=x0),tobs,logistic,c(r=r, K=K))[,2],start=list(r=0.2,K=100,x0=2),
+          control=list(minFactor=1e-20,maxiter=150))
> summary(fit)

Formula: xobs ~ ode(c(x = x0), tobs, logistic, c(r = r, K = K))[, 2]

Parameters:
    Estimate Std. Error t value Pr(>|t|)
r    0.26545    0.02149  12.350 2.23e-07 ***
K  100.77394    2.84022  35.481 7.51e-12 ***
x0   2.37310    0.64768   3.664  0.00436 **
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.614 on 10 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 3.388e-06

> # Vérification: un graphique
> parms<-coef(fit)
> # ici on résout à nouveau l'ED pour davantage de valeurs de times
```

```
> out<-as.data.frame(ode(c(x=x0),times=seq(min(tobs),max(tobs),0.01), logistic, parms))
> # la courbe théorique
> eq=paste0("dy/dt=ry (1-y/K) avec r=",round(parms[1],3)," et K=",round(parms[2],3))
> plot(out,col='red',main="Ajustement des paramètres du modèle logistique pour la croissance du fusarium"
+      ,xlab="t time",ylab="y fusarium",sub=eq)
> # les points expérimentaux
> points(tobs,xobs,pch=20)
>
```



Exercice 3 Procédé en batch dans un fermenteur: Modèle de Monod[2]

On veut modéliser l'évolution des concentrations de biomasse x et de substrat s sous les conditions "batch" par les équations différentielles suivantes :

$$\begin{cases} \dfrac{dx}{dt} = \mu(s)x & x(0) = x_0 \\ \dfrac{ds}{dt} = -\dfrac{\mu(s)}{Y_{x/s}}x & s(0) = s_0 \end{cases}$$

où $Y_{x/s}$ le rendement instantané est supposé constant au cours du temps et (x_0, s_0) sont les concentrations initiales (des constantes strictement positives). $\mu(s)$ le taux de croissance de la biomasse est supposé être la fonction de Monod suivante $\mu(s) = \dfrac{\mu_0 s}{k_s + s}$

1. *Télécharger RMonod.csv sur Moodle et le lire dans la data frame donnees. Tracer le nuage de points expérimentaux (s_i, x_i) (de donnees).*

2. *On rappelle que $s = \dfrac{x_m - x}{Y_{x/s}}$ (*). Déterminer le rendement $Y_{x/s}$.*

3. *Détermination des paramètres μ_0, k_s et x_m:*
   On suppose maintenant que $Y_{x/s}$ est fixé à la valeur déterminée précédemment (on pourrait aussi décider de fixer x_m mais ici on décide de le laisser libre).

   (a) *Tracer le nuage de points (t_i, x_i).*

   (b) *Exprimer $\dfrac{dx}{dt}$ seulement en fonction de x. On simplifiera $\dfrac{dx}{dt}$ pour obtenir la fraction de la forme $\dfrac{dx}{dt} = \mu_0 x \dfrac{x_m - x}{C - x}$.*

[2] extrait en partie de URL :http://pbil.univ-lyon1.fr/R/pdf/tdr48.pdf

(c) Définir la fonction Monod

```
> monod<-function(t,x,parms){
+   s<-(parms["xm"]-x)/Y
+   dx=x*parms["mum"]*s/(parms["ks"]+s)
+   return(list(dx))
+ }
```

(d) Résoudre numériquement l'équation différentielle

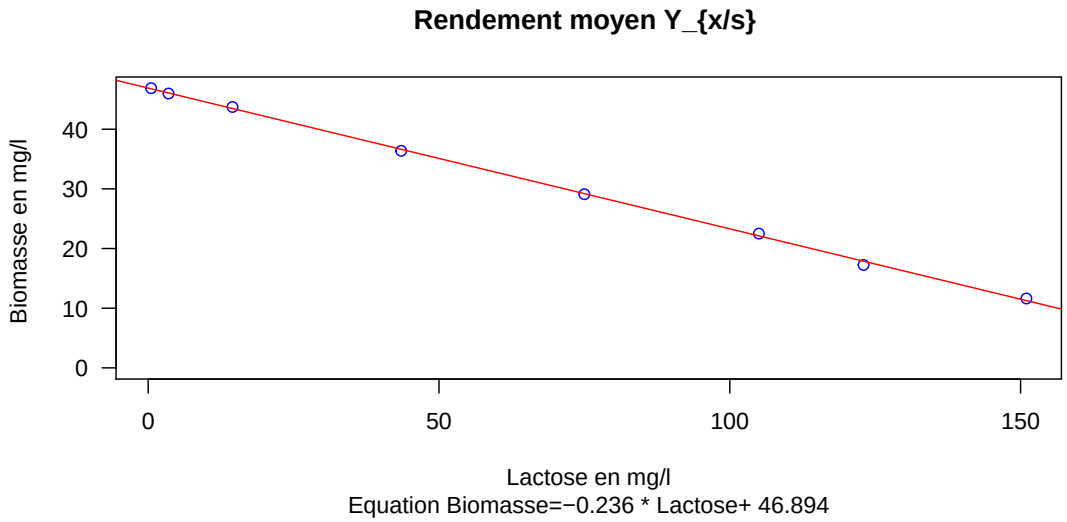
$$\begin{cases} s & = & \dfrac{x_m - x}{Y_{x/s}} \\ \dfrac{dx}{dt} & = & \dfrac{\mu_0 s}{K_s + s} x \quad x(0) = x_0 \end{cases}$$

pour les t_i correspond aux valeurs observées (dans le fichier) ie calculer $x_{th}(t_i)$. Pour x_0 et x_m prendre le minimum et le maximum observés. Pour les autres paramètres, on pourra prendre $\mu_0 = 0.85$ et $k_s = 20$.

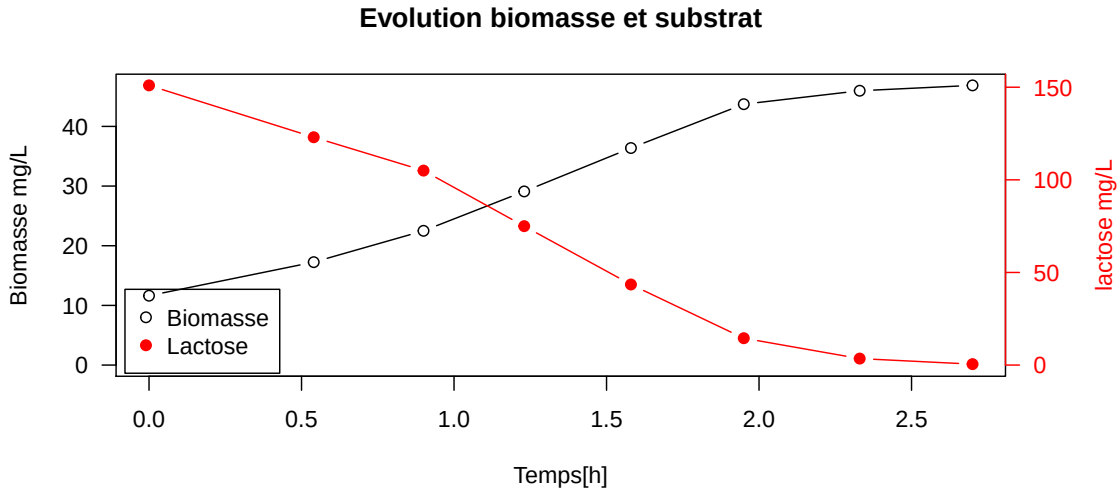
(e) Déterminer les paramètres (x_0, μ_0, k_s) tels que les points $(t_i, x_{th}(t_i))$ approche au mieux au sens des moindres carrés le nuage de points expérimentaux (t_i, x_i). On utilisera la fonction nls, on appelera fit le résultat. Tracer la solution de l'équation différentielle avec les valeurs des paramètres obtenus sur le graphe où figure le nuage de points. On mettra un titre, des labels pour les axes, une légende et des couleurs. On affichera en sous titre les valeurs des différents paramètres.

(f) Télécharger sur Moodle le fichier CourbeNiveau et le copier dans votre programme. Le tester.

```
> # Les données de Monod (1941)
> #tobs<- c(0,0.54,0.9,1.23,1.58,1.95,2.33,2.7)#temps
> #xobs <- c(15.5,23,30,38.8,48.5,58.3,61.3,62.5)#Biomasse
> #sobs <- c(151,123,105,75,43.5,14.5,3.5,0.5)#substrat
> donnees <- read.csv2("RMonod.csv")
> attach(donnees)
> #
> tobs<-TIME
> xobs <-Biomasse*0.75 #conversion DO en mg/l
> sobs <-Substrat
> # ****************************
> # détermination du rendement moyen Y_{x/s}
> rendement<-lm(xobs~sobs,data=donnees)
> Y<--rendement$coefficients[2]
> xm=rendement$coefficients[1]
> # graphe du nuage des points (sobs,xobs)
> eq=paste0("Equation Biomasse=",-round(Y,3)," * Lactose+ ",round(xm,3))
> plot(sobs,xobs,las=1,type="p",xlab="Lactose en mg/l",ylab="Biomasse en mg/l ",ylim=c(0,max(xobs)),col="blue
> abline(xm,-Y,col="red")# droite de régression
> # une régression bien choisie permet d'en déduire une estimation du rendement Yxs.
> # rendement Y fixé (à partir de la régression linéaire de (sobs,xobs) à faire au préalable)
>
```

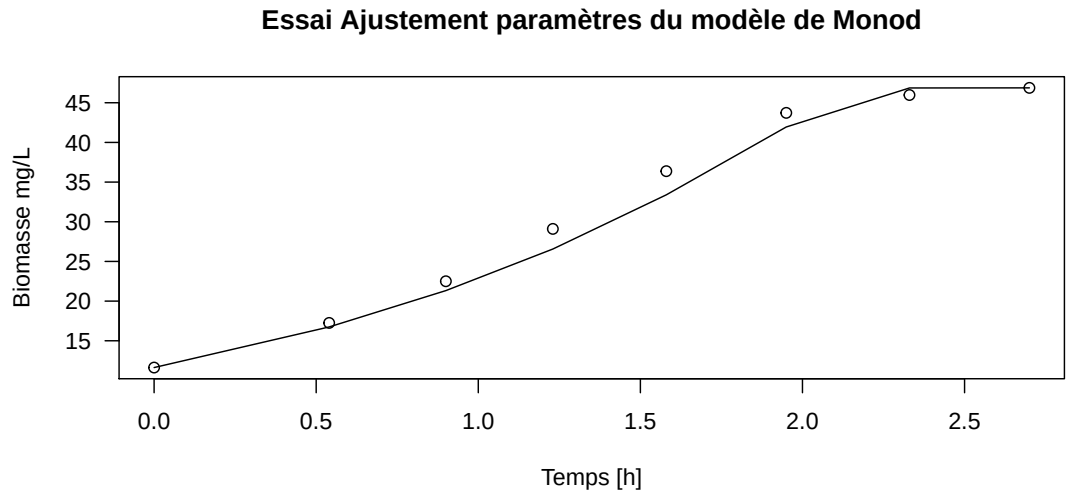



```
> #*******************************
> # graphe de croissance de la Biomasse et consommation Substrat au cours du temps
> # avec deux échelles différentes
> par(mar=c(5,4,4,4)+0.1)
> # nuage (t,x)
> plot(tobs,xobs,las=1,type="b",xlab="Temps[h]",ylab="Biomasse mg/L",
+      ylim=c(0,max(xobs)),main="Evolution biomasse et substrat")
> #calcul de l'échelle pour (t,s)
> yscale<-max(xobs)/max(sobs)
> # nuage (t,s)
> lines(tobs,yscale*sobs,type="b",pch=19,col="red")
> # axes et légendes
> axis(side=4,at=yscale*pretty(sobs),labels=pretty(sobs),las=1, col="red",col.axis="red")
> mtext(text=paste("lactose mg/L"),line=3,side=4,col="red")
> legend("bottomleft",inset=0.01,legend=c("Biomasse","Lactose"),pch=c(1,19),col=c("black","red"))
```



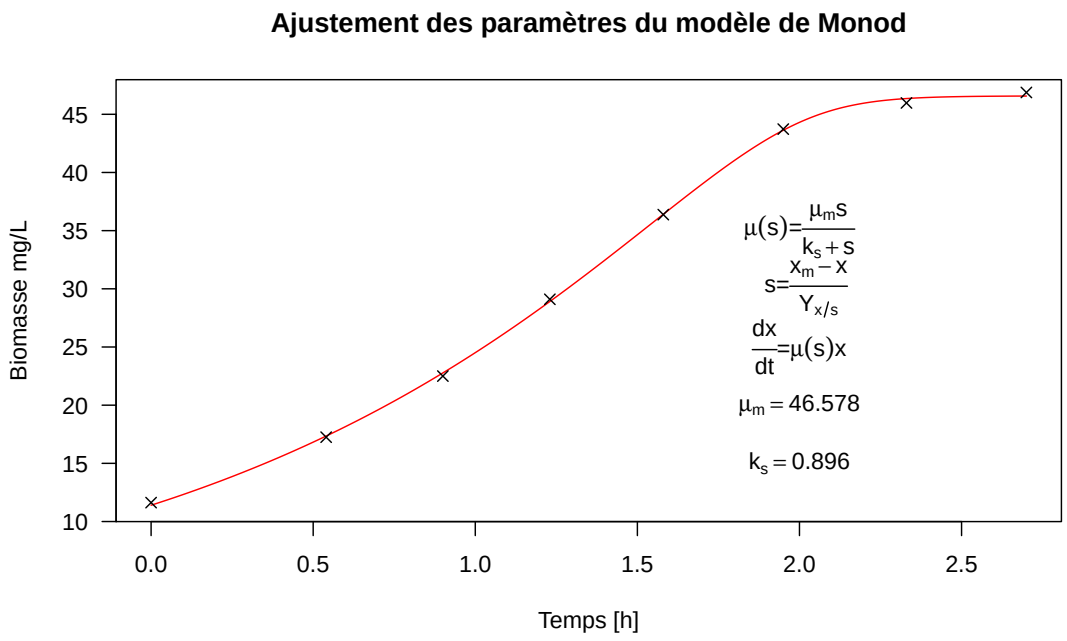

```
> #Intégration du modèle
> #librairie pour résoudre les ED
> library(deSolve)
> #définition de la fonction Monod (avec 3 paramètres: xm, mum, ks)
```

```
> monod<-function(t,x,parms){
+   s<-(parms["xm"]-x)/Y
+   dx=x*parms["mum"]*s/(parms["ks"]+s)
+   return(list(dx))
+ }
> # Un essai: les paramètres (qu'il vous faudra "fitter") sont ici fixés
> # mu=0.7 et ks=5
> # x0,xm sont fixés à partir des données expérimentales
> x0<-min(xobs) #x0=plus petite valeur expérimentale
> xmax<-max(xobs) #xmax=plus grande valeur
> # Résolution de l'ED: x'=f(x) par appel de la fonction lsoda
> # ode(condition initiale, valeurs ti où l'on va calculer y(ti), fonction f, paramètres éventuels de f)
> # ode renvoie une matrice [ti y(ti)]
> theo<-ode(c(x=x0),times=tobs,func=monod,parms=c(xm=xmax,mum=0.7,ks=5))
> # graphe de la solution (ti,y(ti))
> plot(theo,type="l",xlab="Temps [h]",ylab="Biomasse mg/L",las=1,main="Essai Ajustement paramètres du modèle
> points(tobs,xobs)
```




```
> ##############################################################################################################
> # Détermination des paramètres par la methode des moindres carres: appel de nls
> # Nonlinear Least Square
> initialisation=list(xm=xmax,mum=0.7,ks=5,x0=x0)
> fit<-nls(xobs~ode(c(x=x0),tobs,monod,c(xm=xm,mum=mum,ks=ks))[,2],start=initialisation,
+          control=list(minFactor=1e-20,maxiter=150))
> summary(fit)

Formula: xobs ~ ode(c(x = x0), tobs, monod, c(xm = xm, mum = mum, ks = ks))[,
    2]

Parameters:
    Estimate Std. Error t value Pr(>|t|)
xm  46.57768    0.26600 175.107 6.38e-09 ***
mum  0.89639    0.05353  16.746 7.45e-05 ***
ks  20.97950    5.37317   3.904   0.0175 *
x0  11.40114    0.24178  47.156 1.21e-06 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.3228 on 4 degrees of freedom

Number of iterations to convergence: 8
Achieved convergence tolerance: 9.956e-07

> # calcul de [ti y(ti)] pour les paramètres trouvés par nls
> theo<-ode(c(x=coef(fit)[4]),times=seq(min(tobs),max(tobs),0.01),func=monod,parms=coef(fit)[1:3])
> # graphe de la solution (ti,y(ti))
> plot(theo,type="l",xlab="Temps [h]",ylab="Biomasse mg/L",las=1,main="Ajustement des paramètres du modèle de
> text(2, 35, expression(paste(mu(s),"=",frac(mu[m]*s, k[s]+s) )))
> text(2, 30, expression( paste("  s=",frac(x[m] -x, Y[x/s]) )))
> text(2, 25, expression(paste(frac(dx,dt),"=",mu(s)*x) ))
> text(2, 20, bquote(mu[m] ==  .(round(coef(fit)[1],3))))
> text(2, 15, bquote(k[s] ==  .(round(coef(fit)[2],3))))
> points(tobs,xobs,pch=4)
```

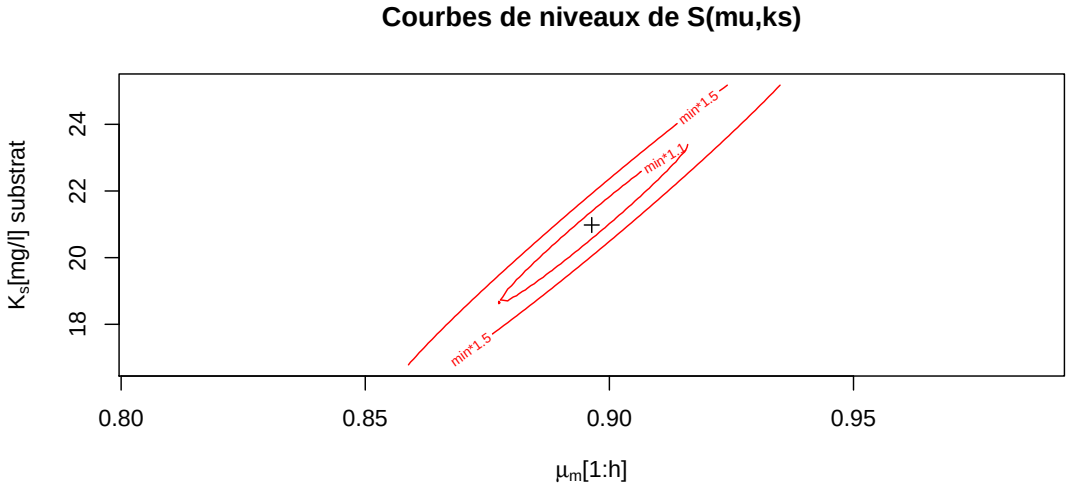



```
> # Courbes de niveaux de S en fonction de mu et ks
> ecart<-function(p){
+   # fonction S(mu,ks,x0) des écarts aux carré
+     # calcul de la solution theorique aux points t=tobs (pour calculer l'écart avec xobs valeurs observées)
+       theo<-ode(y=c(x=p[4]),times=tobs,func=monod,parms=c(xm=p[1],mum=p[2],ks=p[3]))
+     xtheo=theo[,2]
+     S=sum((xobs-xtheo)^2)
+     return(S)
+   }
> p=as.numeric(coef(fit))
>   # calcul de S(mu,ks,x0) sur une grille de (nbpoints,nbpoints)
>     mumin=p[2]*0.9
>     mumax=p[2]*1.1
>     ksmin=p[3]*0.8
>     ksmax=p[3]*1.2
>     nbpoints=100
>     x0=as.numeric(p[4])
>     xm=as.numeric(p[1])
```

```
>     museq<-seq(from =mumin, to =mumax, length=nbpoints)
>     ksseq<-seq(from =ksmin, to =ksmax, length=nbpoints)
>     # valeur mini de S
>     min=ecart(p)
>     print(min)
```

```
[1] 0.4168425
```

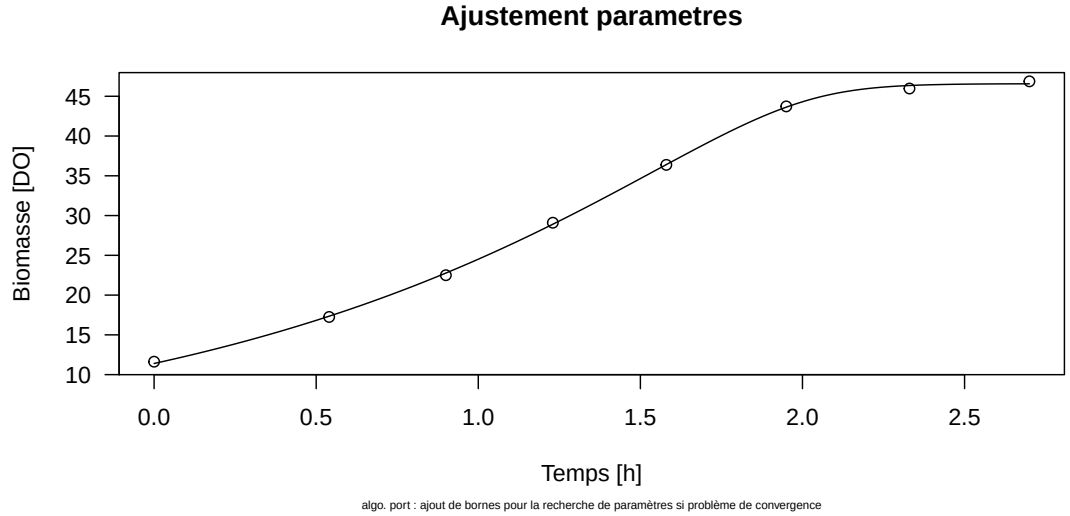
```
>     seuil=min*1.05
>     # grille de valeurs p=(mu,ks)
>     ecartgrid<-matrix(nrow=length(museq),ncol=length(ksseq))
>     # tracé dans le plan (ks,mu) de valeur p minimisant ecart(p)
>     for (i in 1:nbpoints){
+       for (j in 1:nbpoints){
+         #calcul ecart(p de la grille)
+         ecartgrid[i,j]<-ecart(c(xm,museq[i],ksseq[j],x0))
+         }
+     }
>     # courbes de niveaux de S
> contour(museq,ksseq,ecartgrid,col="red",levels=c(min*1.1,min*1.5),
+         xlab=expression(paste(mu[m],"[1:h]")),ylab=expression(paste(K[s],"[mg/l] substrat")),
+         main="Courbes de niveaux de S(mu,ks)",labels=c("min*1.1","min*1.5"))
>     points(coef(fit)[2],coef(fit)[3],pch=3)
```



Parfois, le solveur peine à converger (les paramètres sont très corrélés, les courbes de niveaux ressemblent à des ellipses très "aplaties"). On peut aider le solveur en ajoutant des bornes (cela l'empêchera au moins de diverger).

```
> ##############################################################################################
> # bornes pour x0
> lx0<-x0*0.25
> ux0<-x0*2
> #bornes pour xm
> lxm<-xm*0.8
> uxm<-xm*1.5
> # mu, Ks, x0, xm sont bornés par lb et ub (lower bounds et upper bounds)
> lb=c(lxm,0.0001,18,lx0)
> ub=c(uxm,5,25,ux0)
> # dans ce cas, il faut changer d'algorithme pour nls: prendre "port", seul algo implémenté autorisant les b
> fit<-nls(xobs~lsoda(c(x=x0),tobs,monod,parms=c(xm=xm,mum=mum,ks=ks))[,2],
+          start=c(xm=xmax,mum=0.85,ks=20,x0=x0),control=list(tol = 1e-2,
+         minFactor=1e-20,maxiter=100),trace="TRUE",algorithm="port",lower =lb,upper = ub)
```

```
  0:     2.7972590:  46.8750 0.850000  20.0000  11.4011
  1:    0.24390426:  46.5539 0.868650  19.1540  11.5618
  2:    0.20940726:  46.5614 0.888949  20.2633  11.4264
  3:    0.20845120:  46.5744 0.895675  20.8921  11.4023
  4:    0.20842166:  46.5772 0.896272  20.9652  11.4014
  5:    0.20842125:  46.5776 0.896369  20.9771  11.4012
  6:    0.20842124:  46.5777 0.896386  20.9791  11.4011
  7:    0.20842124:  46.5777 0.896388  20.9794  11.4011
  8:    0.20842124:  46.5777 0.896389  20.9795  11.4011

> summary(fit)

Formula: xobs ~ lsoda(c(x = x0), tobs, monod, parms = c(xm = xm, mum = mum,
    ks = ks))[, 2]

Parameters:
    Estimate Std. Error t value Pr(>|t|)
xm  46.57768    0.26600 175.107 6.38e-09 ***
mum  0.89639    0.05353  16.746 7.45e-05 ***
ks  20.97946    5.37315   3.905   0.0175 *
x0  11.40114    0.24178  47.156 1.21e-06 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3228 on 4 degrees of freedom

Algorithm "port", convergence message: relative convergence (4)

> theo<-ode(c(x=coef(fit)[4]),times=seq(from=min(tobs), to =max(tobs),length=255),func=monod,parms=coef(fit))
> # graphe de la solution (ti,y(ti))
> plot(theo,type="l",xlab="Temps [h]",ylab="Biomasse [DO]",las=1,main="Ajustement parametres",sub="algo. port
> points(tobs,xobs)
```



Parfois cela ne suffit pas, on peut essayer de re-paramétrer le modèle pour décorréler les paramètres.

Exercice 4 *Etude de la sensibilité(exercice qui n'est pas au programme de l'examen)*
En pratique lorsque l'on veut déterminer des paramètres, il faut définir une stratégie d'échantillonnage. On commence en général par mener une expérience avec un échantillonnage régulier qui permet d'avoir une première estimation des paramètres. On peut lors d'une deuxième expérience affiner l'estimation en échantillonnant au voisinage des extrema des

fonctions de sensiblité. On veillera cependant à ne pas échantillonner dans la zone où les fonctions de sensibilité sont "linéaires" les unes par rapport aux autres.

On suppose que la fonction y suit une croissance dite logistique c'est-à-dire vérifie l'équation différentielle

$$\begin{cases} y'(t) = ry(t)(1 - y(t)/K) \\ y(0) = y_0 \end{cases}$$

Posons $f(y) = ry(1 - \frac{y}{K})$. *On a donc* $y'(t) = f(y(t))$.

1. *Déterminer les dérivées partielles de* f *par rapport à* y, r *et* K.
2. *On rappelle que* $\frac{\partial}{\partial a_i}(\frac{dy}{dt}) = \frac{\partial}{\partial a_i}(f(y)) = \frac{\partial f}{\partial a_i}(y) + \frac{\partial f}{\partial y}\frac{\partial y}{\partial a_i}$. *Calculer* $\frac{\partial}{\partial a_i}(\frac{dy}{dt})$ *pour* $a_i = y_0$, $a_i = r$, $a_i = K$.
3. *En posant* $y_2 = \frac{\partial y}{\partial y_0}$, *et en admettant que l'on peut intervertir l'ordre de dérivation, montrer que* y_2 *vérifie l' équation différentielle:*
$$\frac{dy_2}{dt} = r(1 - \frac{2y}{K})y_2$$
4. *Déterminer les équations différentielles vérifiées par* $y_3 = \frac{\partial y}{\partial r}$, $y_4 = \frac{\partial y}{\partial K}$.
5. *On donne les conditions initiales suivantes* $y(0) = 2$, $y_2(0) = 1$, $y_3(0) = 0$, $y_4(0) = 0$. *On donne* $r = 0.266$ *et* $K = 100$. *Résoudre le système différentiel formé par* $(y_1 = y, y_2, y_3, y_4)$ *avec la fonction ode (Remarque dans une fonction* $f(t, y, par)$, y *et par peuvent être des vecteurs).*
6. *Afficher sur une mettre fenêtre graphique l'évolution en fonction de* t *de* y_1, y_2, y_3 *et* y_4.
7. *Déterminer les meilleures zones d'échantillonnage pour estimer les différents paramètres.*