

Rapport de projet OS13

Analyse de politique de maintenance

TRAN QUOC NHAT HAN & ADRIEN WARTELLE

10 janvier 2019

Sommaire

| | | |
|----------|--|-----------|
| 1 | Maintenance basée sur l'âge | 1 |
| 1.1 | Rappel | 1 |
| 1.2 | Modéliser la durée de vie du système | 2 |
| 1.3 | La politique de maintenance basée sur l'âge | 5 |
| 2 | Maintenance basée sur dégradation | 6 |
| 2.1 | Rappel | 6 |
| 2.2 | Modéliser la dégradation du système | 7 |
| 2.3 | La politique de maintenance basée sur dégradation | 9 |
| 3 | Annexe | 13 |
| 3.1 | L'importation de données de pannes | 13 |
| 3.2 | Le premier histogramme de distribution de pannes | 13 |
| 3.3 | Estimer le mixage de la loi Exponentielle et Gamma | 13 |
| 3.4 | Optimiser le coût moyenne sur une durée de temps | 15 |
| 3.5 | Importer les valeurs de dégradation | 15 |
| 3.6 | Premiers traces de dégradation | 16 |
| 3.7 | Estimation de paramètres de dégradations | 16 |
| 3.8 | Optimisation de maintenance basée sur dégradations | 17 |

Résumé

Soient des données liées à la fonctionnement de système, nous déterminons un modèle approprié et puis choisir une politique de maintenance optimal.

1 Maintenance basée sur l'âge

1.1 Rappel

Considérons un système non maintenu. En l'observant, nous obtenons un liste des dates de panne, grâce auquel nous construirons une politique de remplacement systématique basée sur l'âge : *Nous remplaçons lorsque le système tombe en panne ou qu'il survit une durée t_0 .*

Le but est de minimiser le coût moyen cumulé.

$$\mathbb{E}(C) = \frac{\mathbb{E}(C(S))}{\mathbb{E}(S)} \quad (1)$$

Où S est la variable aléatoire représentant la date de remplacement et $C(S)$ est le coût de maintenance cumulé à l'instant S (sachant que $C(S)$ est $c_c(= 1200)$ si une maintenance corrective et $c_p(= 800)$ si préventive).

1.2 Modéliser la durée de vie du système

L'importation de données de `FailureTimes_5.csv` (l'annexe 3.1) expose les dates de pannes de l'ordre grandement variée (300 à 27000) (l'annexe 3.2).

Exponentiel des valeurs extrêmes résulteront *Inf*, ce qui est indésirable. Alors nous devons forcément les réduire en les divisant par un scalaire `scale`, prenons par exemple 1000. (Figure 1)

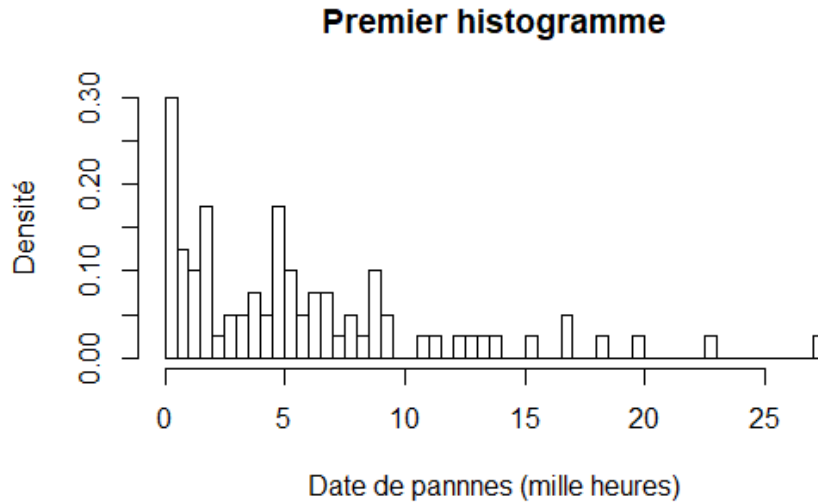


FIGURE 1 – Le premier histogramme de distribution de pannes

Les pannes se concentrent autour de 2 sommets, l'un à $[0; 0, 5]$ et l'autre à $[4, 5; 5]$. Ceci nous fait penser naturellement à un mixage de deux lois.

Nous pouvons remarquer que les valeurs sont positives (étant données que ce sont des temps) et que la distribution semble posséder deux parties importantes. Une dont le sommet se situe près de zéro et qui est suivi d'une pente forte, et l'autre sommet est à une valeur non nulle (5) que la distribution locale en forme de pic. Nous avons donc penser estimer un mixage de loi *Exponentielle* et *Gamma* afin de modéliser les données. En effet la première partie correspondrait à une loi exponentielle tandis que la seconde à une loi gamma (Et .

La fonction de densité avec le paramètre $\theta = (p_1, p_2, \lambda, \alpha, \beta)$:

$$\begin{aligned} f_{\theta}(x) &= p_1 f_1(x) + p_2 f_2(x) \\ &= p_1 \lambda e^{-\lambda x} + p_2 \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \end{aligned} \quad (2)$$

Où f_1, f_2 désignent respectivement $\exp(\lambda)$ et $\Gamma(\alpha, \beta)$; $p_1, p_2 > 0$; $p_1 + p_2 = 1$.

Nous allons utiliser l'algorithme EM, la méthode la plus efficace pour estimer le MLE de mixage fini.

Soit X la variable aléatoire de durée de vie du système. Soient (x_1, \dots, x_N) les observations.

Soit la matrice de probabilité d'appartenance (ζ_{ki}) : ζ_{ki} vaut la probabilité que x_i suive la loi f_k .

$$\zeta_{ki} = \frac{p_k f_k(x_i)}{p_1 f_1(x_i) + p_2 f_2(x_i)} \forall k = \overline{1, 2} \forall i = \overline{1, N} \quad (3)$$

La fonction de vraisemblance :

$$\ln \Lambda = \sum_{i=1}^N \ln f_{\theta}(x_i) = \sum_{i=1}^N \ln (p_1 f_1(x_i) + p_2 f_2(x_i)) \quad (4)$$

Nous cherchons à maximiser $\ln \Lambda$ en la dérivant selon λ, α, β .

Pour λ :

$$\begin{aligned} \frac{\partial}{\partial \lambda} \ln \Lambda &= \sum_{i=1}^N \frac{p_1 e^{-\lambda x_i} - p_1 \lambda x_i e^{-\lambda x_i}}{p_1 f_1(x_i) + p_2 f_2(x_i)} \\ &= \sum_{i=1}^N \frac{p_1 f_1(x_i)}{p_1 f_1(x_i) + p_2 f_2(x_i)} \left(\frac{1}{\lambda} - x_i \right) \\ &= \frac{1}{\lambda} \sum_{i=1}^N \zeta_{1i} - \sum_{i=1}^N \zeta_{1i} x_i = 0 \\ \Leftrightarrow \lambda &= \frac{\sum_{i=1}^N \zeta_{1i}}{\sum_{i=1}^N \zeta_{1i} x_i} \end{aligned} \quad (5)$$

Pour β :

$$\begin{aligned} \frac{\partial}{\partial \beta} \ln \Lambda &= \sum_{i=1}^N \frac{p_2 x_i^{\alpha-1}}{\Gamma(\alpha)} \frac{\alpha \beta^{\alpha-1} e^{-\beta x_i} - \beta^{\alpha} x_i e^{-\beta x_i}}{p_1 f_1(x_i) + p_2 f_2(x_i)} \\ &= \sum_{i=1}^N \frac{p_2 f_2(x_i)}{p_1 f_1(x_i) + p_2 f_2(x_i)} \left(\frac{\alpha}{\beta} - x_i \right) \\ &= \frac{\alpha}{\beta} \sum_{i=1}^N \zeta_{2i} - \sum_{i=1}^N \zeta_{2i} x_i = 0 \\ \Leftrightarrow \beta &= \alpha \frac{\sum_{i=1}^N \zeta_{2i}}{\sum_{i=1}^N \zeta_{2i} x_i} \end{aligned} \quad (6)$$

Pour α :

$$\begin{aligned}
\frac{\partial}{\partial \alpha} \ln \Lambda &= \sum_{i=1}^N \frac{p_2 e^{-\beta x_i}}{p_1 f_1(x) + p_2 f_2(x)} \left(\frac{\beta (\ln \beta + \ln x_i) (\beta x_i)^{\alpha-1}}{\Gamma(\alpha)} - \beta^\alpha x_i^{\alpha-1} \frac{\Psi(\alpha)}{\Gamma(\alpha)} \right) \\
&= \sum_{i=1}^N \frac{p_2 f_2(x_i)}{p_1 f_1(x) + p_2 f_2(x)} (\ln \beta + \ln x_i - \Psi(\alpha)) \\
&= \left(\sum_{i=1}^N \zeta_{2i} \right) \ln \beta + \sum_{i=1}^N \zeta_{2i} \ln x_i - \Psi(\alpha) \left(\sum_{i=1}^N \zeta_{2i} \right) = 0 \\
&\Leftrightarrow 0 = \ln \alpha + \ln \frac{\sum_{i=1}^N \zeta_{2i}}{\sum_{i=1}^N \zeta_{2i} x_i} + \frac{\sum_{i=1}^N \zeta_{2i} \ln x_i}{\sum_{i=1}^N \zeta_{2i}} - \Psi(\alpha) \quad (\text{substitué par (6)}) \\
&\Leftrightarrow 0 = \ln \alpha - \Psi(\alpha) - c
\end{aligned}$$

Où $c = \ln \left(\frac{\sum_{i=1}^N \zeta_{2i} x_i}{\sum_{i=1}^N \zeta_{2i}} \right) - \frac{\sum_{i=1}^N \zeta_{2i} \ln(x_i)}{\sum_{i=1}^N \zeta_{2i}}$; Ψ est la fonction digamma.

Selon la méthode de Newton-Rashphon, nous pouvons résoudre α numériquement avec ce formul itératif :

$$\alpha_{r+1} = \alpha_r - \frac{\ln \alpha_r + \Psi(\alpha_r) - c}{\frac{1}{\alpha_r} - \Psi'(\alpha_r)}$$

[1] propose un autre formule convergeant plus vite :

$$\frac{1}{\alpha_{r+1}} = \frac{1}{\alpha_r} + \frac{\ln(\alpha_r) - \Psi(\alpha_r) - c}{\alpha_r^2 \left(\frac{1}{\alpha_r} - \Psi'(\alpha_r) \right)} \quad (7)$$

Avec Ψ' la fonction trigamma. L'itération part avec $\alpha_0 = \frac{0.5}{c}$.

Au final, pour p_k :

$$p_k = \frac{\sum_{i=1}^N \zeta_{ki}}{N} \forall k = \overline{1, 2} \quad (8)$$

Etant donné (3), (5), (6), (7) et (8), nous définissons l'algorithme EM :

1. **Initialisation** : Choisir θ_0 .
2. **Etape E** : Evaluer (ζ_{ki}) sachant θ_c en utilisant (3).
3. **Etape M** : Calculer θ_{c+1} à l'aide des équations (5), (6), (7) et (8).
Note : Pour α , l'itération se termine quand $|\alpha_{r+1} - \alpha_r| < \varepsilon_\alpha$ où ε_α est un réel positif fixé à l'initialisation.
4. **Evaluation** : Si $\|\theta_{c+1} - \theta_c\| < \varepsilon_\theta$ (ε_θ est un réel positif fixé à l'initialisation), l'algorithme s'arrête et $\theta = \theta_{vieux}$.
Sinon, reviens à l'étape E avec $\theta_{vieux} \leftarrow \theta_{nouveau}$.

Avant de lancer l'algorithme nous essayons d'obtenir un ensemble de paramètres initiaux θ_0 qui soient cohérent avec la distribution des données. Nous avons choisi

$$(p_{10}, p_{20}, \lambda_0, \alpha_0, \beta_0) = (0.5; 0.5; 1; 10; 2)$$

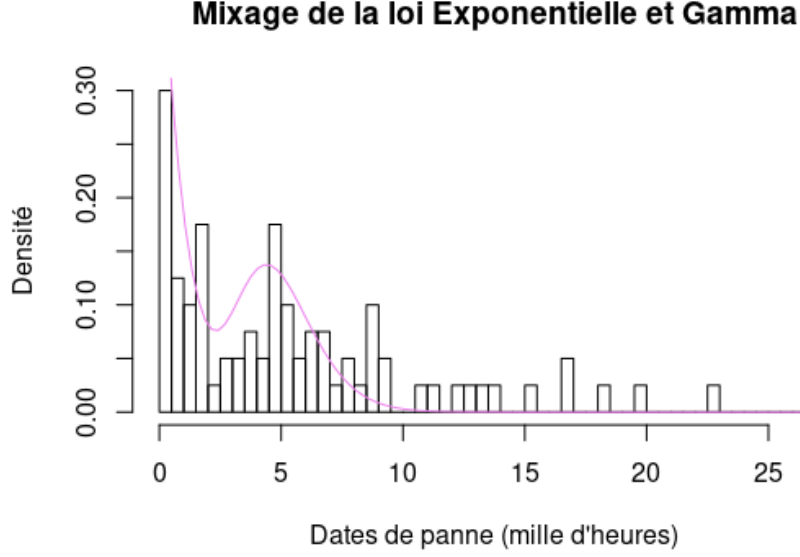


FIGURE 2 – Mixage de la loi Exponentielle et Gamma : paramètres initiaux

Après l'utilisation de l'algorithme EM, nous avons obtenu le résultat :

$$(p_1, p_2, \lambda, \alpha, \beta) = (0.2194518; 0.7805482; 1.56738; 1.665659; 0.2332427)$$

D'où nous traçons la fonction de densité f_θ trouvé (figure 3) et réalisons un test de Kolmogorov-Smirnov qui donne $p\text{-value} = 0,9663111$ signifiant 96,63% de nous tromper si nous rejetons ce modèle. Nous l'acceptons alors, quoiqu'il ne génère pas 2 sommets comme la remarque initiale. Le code est trouvable à l'annexe 3.3.

1.3 La politique de maintenance basée sur l'âge

Avec la fonction f_θ trouvée, nous construirons la politique optimale.

Nous avons par définition : $S = \min(X, t_0)$.

Autrement dit, $S = X\mathbb{I}_{\{X < t_0\}} + t_0\mathbb{I}_{\{X \geq t_0\}}$.

Traduit au coût : $C(S) = C_c\mathbb{I}_{\{X < t_0\}} + C_p\mathbb{I}_{\{X \geq t_0\}}$, avec C_c, C_p les coûts de maintenances correctives et préventives respectivement.

Le coût moyen :

$$\begin{aligned} \mathbb{E}(C(S)) &= c_c\mathbb{E}(\mathbb{I}_{\{X < t_0\}}) + c_p\mathbb{E}(\mathbb{I}_{\{X \geq t_0\}}) \\ &= c_cP(X < t_0) + c_pP(X \geq t_0) \\ &= c_cF_\theta(t_0) + c_p(1 - F_\theta(t_0)) \\ &= (c_c - c_p)F_\theta(t_0) + c_p \end{aligned} \tag{9}$$

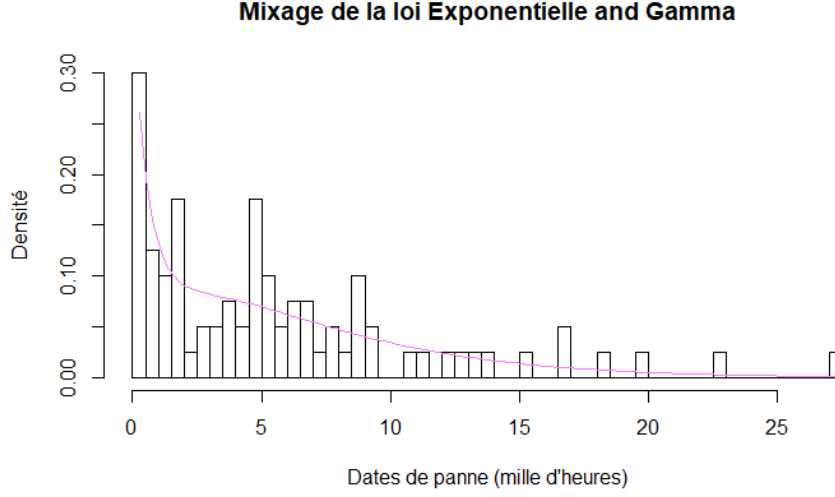


FIGURE 3 – Mixage de la loi Exponentielle et Gamma : paramètres finaux

La durée moyenne :

$$\begin{aligned}
 \mathbb{E}(S) &= \mathbb{E}(X \mathbb{I}_{\{X < t_0\}}) + t_0 \mathbb{E}(\mathbb{I}_{\{X \geq t_0\}}) \\
 &= \int_0^{t_0} x f_{\theta}(x) dx + t_0 P(X \geq t_0) \\
 &= x F_{\theta}(x) \Big|_0^{t_0} - \int_0^{t_0} F_{\theta}(x) dx + t_0 (1 - F_{\theta}(t_0)) \\
 &= t_0 - \int_0^{t_0} F_{\theta}(x) dx
 \end{aligned} \tag{10}$$

De (9) et (10), nous détaillons le coût moyen sur une durée de temps (1) :

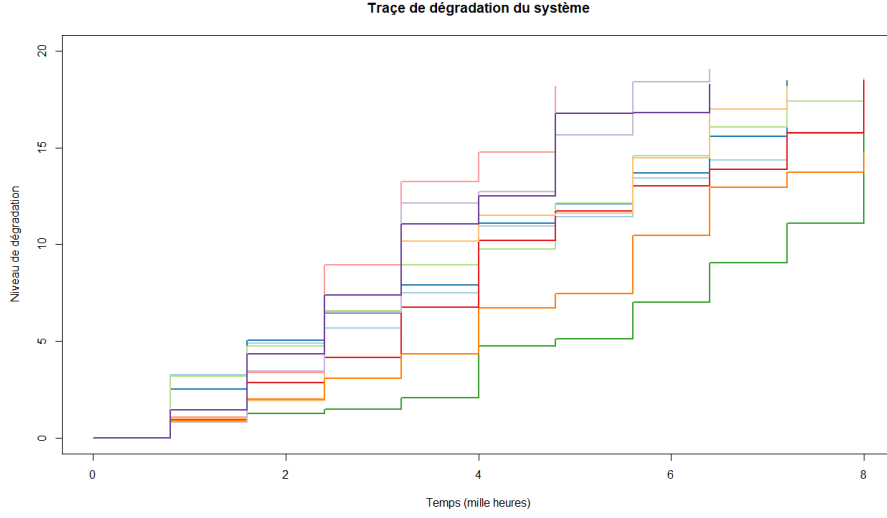
$$\mathbb{E}(C) = \frac{\mathbb{E}(C(S))}{\mathbb{E}(S)} = \frac{(c_c - c_p) F_{\theta}(t_0) + c_p}{t_0 - \int_0^{t_0} F_{\theta}(x) dx} \tag{11}$$

L'annexe (3.4) montrer comment chercher l'optimum numériquement. La valeur minimum est $t_0 = 27,29639$ (mille heures), correspondant à un coût moyen de 210,6402. Nous constatons que t_0^{min} est très proche du maximum de durée de vie, indiquant que l'optimisation de t_0 est inutile car le système ne vieillit pas.

2 Maintenance basée sur dégradation

2.1 Rappel

En observant multiples systèmes identiques, nous effectuons des mesures de dégradation sur des intervalles de temps réguliers tout au long de leurs durée de vie.



La valeur limite de dégradation est $L = 20$. C'est-à-dire lorsque le niveau de dégradation dépasse L , le système tombe en panne et nous ne pourrions plus le mesurer.

On souhaite de mettre en place une politique de maintenance conditionnelle, basée sur un seuil M inférieur à L et l'intervalle de temps ΔT entre les inspections répétées. Appellons X_t le niveau de dégradation à l'instant t (instant d'une inspection).

- Si $X_t < M$, nous laissons le système tel quel.
- Si $M \leq X_t < L$, un remplacement préventif est réalisé au coût c_p . Et puis X_t est remis à 0.
- Si $X_t \geq L$, un remplacement correctif est fait au coût c_c . Et puis X_t est remis à 0.

Le but est minimiser le coût moyen sur une durée de temps (1) en bien choisissant le seuil M et l'intervalle d'inspection ΔT .

2.2 Modéliser la dégradation du système

Soient les données de `DegradLevel_2.csv`, nous traçons leurs processus de dégrader (figure (2.2)). (Les annexes (3.5) et (3.6))

Comme les temps d'inspection sont de l'ordre millier, il vaudrait de les diviser par un scalaire (par exemple, $scale = 1000$) afin d'assurer la précision de calcul numérique.

Nous voyons les accroissements positifs, suggérant un modèle de processus Gamma.

Soit $X(t)$ la variable aléatoire de dégradation du système. Supposons que $X(t) - X(s) \sim \Gamma(a(t-s), b) \forall t > s > 0$. Nous allons estimer les paramètres a, b en modélisant la distribution des incréments entre deux moments successifs.

Fixons $t - s = \delta = 0.8$, car les mesures données sont effectués au bout de chaque intervalle de 0.8.

L'histogramme des incréments :

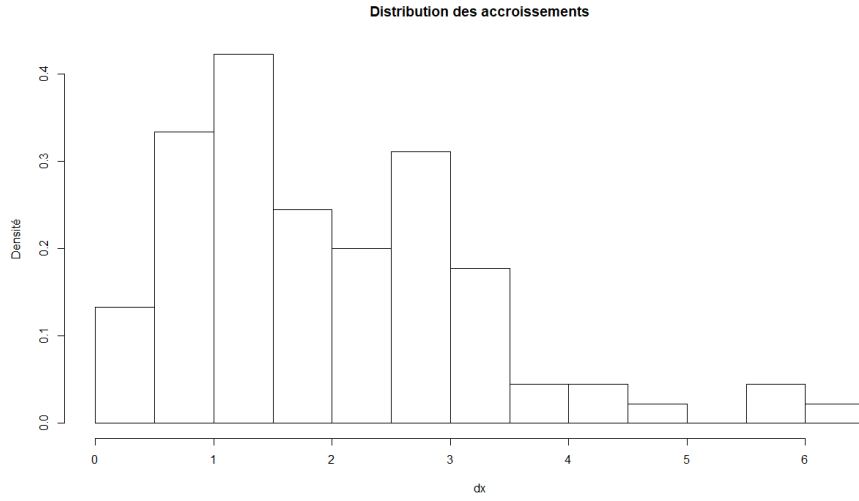


FIGURE 4 – Histogramme des incréments de l'intervalle $\delta = 0.8$

A l'aide du librairie MASS : $(a; b) = (\frac{\alpha}{\delta}; \beta) = (2.843101; 1.140354)$ où (α, β) sont les paramètres estimés par MASS. Le code est mis à l'annexe (3.7).

Un test rapide de Kolmogorov-Smirnov nous donne $p - value = 0.8651934$, indiquant le modèle est acceptable.

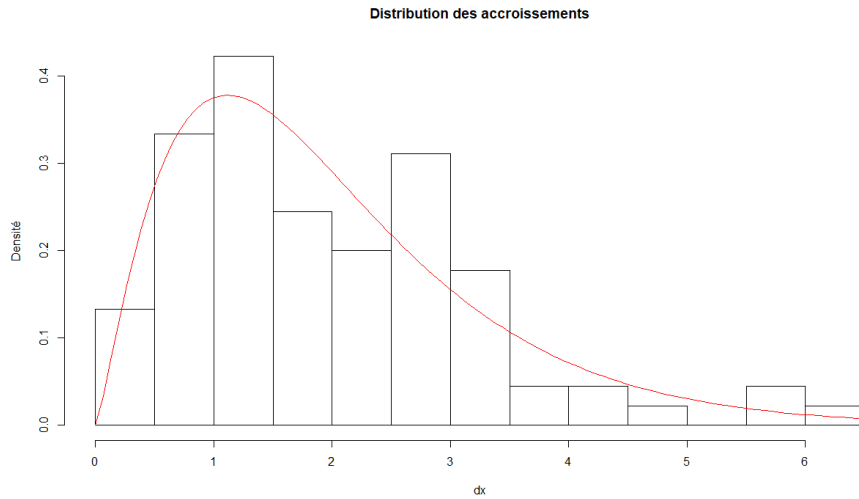


FIGURE 5 – Histogramme et la courbe de densité estimé des incréments de l'intervalle $\delta = 0.8$

D'autant plus, si nous refaisons les calculs ci-dessus avec $\delta = 1.6, 2.4, 3.2, etc.$, nous voyons les valeurs de a et b ne varient pas trop. Alors, nous choisissons le couple (a, b) avec p le plus grand. (δ trop grand réduira nombreux de données, résultant une perte importante de précision)

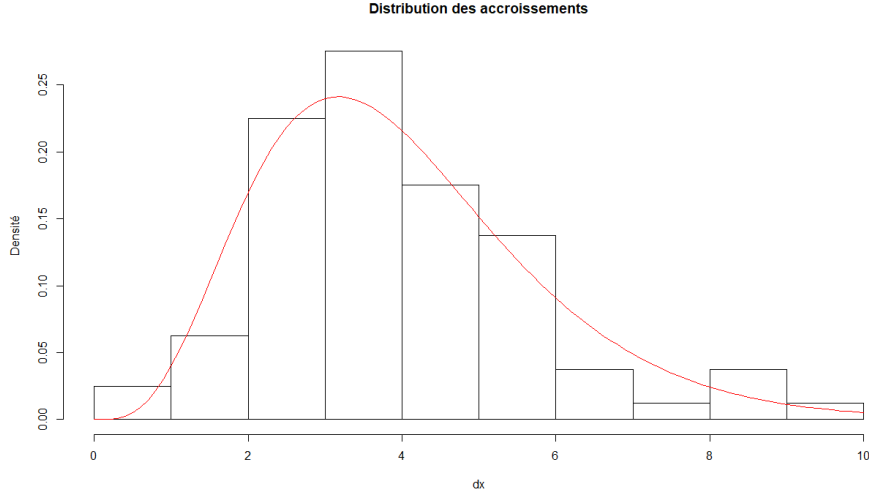


FIGURE 6 – Histogramme et la courbe de densité estimé des incréments de l'intervalle $\delta = 1.6$

| δ | a | b | p |
|----------|-----------|-----------|-----------|
| 0.8 | 2.8431009 | 1.1403540 | 0.8651934 |
| 1.6 | 3.0207719 | 1.2091646 | 0.9919939 |
| 2.4 | 3.0187497 | 1.1907719 | 0.1279252 |
| 3.2 | 2.997763 | 1.185547 | 0.300610 |
| 4.0 | 3.5036580 | 1.4061063 | 0.6307951 |

TABLE 1 – Calcul (a, b) avec différents δ

Au final, $X(t) - X(s) \sim \Gamma(a(t-s), b) \forall t > s > 0$ avec

$$(a, b) = (3.0207719; 1.2091646)$$

2.3 La politique de maintenance basée sur dégradation

Cette politique, outre que $c_c = 1200$ et $c_p = 800$, introduit ainsi le coût d'inspection répétitive $c_i = 10$.

Soit S la variable aléatoire représentant la date de remplacement ; $C(S)$ est le coût de maintenance cumulé à l'instant S , et $N(S)$ le nombre d'inspections depuis la dernière remplacement jusqu'à S .

$$C(S) = c_i N(S) + c_p \mathbb{I}_{\{L > X_{N(S)} \geq M\}} + c_c \mathbb{I}_{\{X_{N(S)} \geq L\}} \quad (12)$$

Posons $F_t = \Gamma(at, b)$ et les instants d'inspections $t_1, t_2, \dots, t_{N(S)}$ avec $t_{j+1} - t_j = \Delta T$. Puisque $t_0 = 0$, nous aurons $t_k = k\Delta T$.

Comme le calcul analytique de $C(S)$ et $N(S)$ devient grossier, nous passons à la méthode Monté Carlo, qui nous permet d'estimer la vraie valeur en répéter un nombre suffisant de simulations.

Le code implémenté se situe à l'annexe 3.8.

Algorithme 1 : Mesurer la durée de vie d'un processus

Entrées : Le processus p , l'intervalle d'inspection ΔT , le seuil M

Output : La durée de vie S

```
1  $len \leftarrow$  longueur de  $p$ ;  
2 si  $p_{len} < M$  alors  
   | // La simulation n'est pas suffisante longue  
3   | retourner  $\emptyset$   
4 sinon  
5   |  $n_s \leftarrow$  le dernier index de  $p$  tel que  $p_{n_s} < M$ ;  
6   | retourner  $n_s + 1$   
7 fin
```

Algorithme 2 : Mesurer le coût de maintenance d'un processus

Entrées : Le processus p , le seuil M

Output : Le coût de maintenance $C(S)$

```
1  $cout \leftarrow 0$ ;  
2  $len \leftarrow$  longueur de  $p$ ;  
3 si  $p_{len} \geq M$  alors  
4   |  $n_s \leftarrow$  le dernier index de  $p$  tel que  $p_{n_s} < M$ ;  
   | //  $n_s$  correspond le nombre d'inspection  
5   |  $cout \leftarrow cout + c_i * n_s$ ;  
   | // A l'inspection suivante  
6   | si  $p_{n_s+1} \geq L$  alors  
   | | // Le système est tombé en panne  
   | |  $cout \leftarrow cout + c_c$ ;  
7   | sinon  
   | | // Le système fonctionne encore  
   | |  $cout \leftarrow cout + c_p$ ;  
8   | fin  
9   | fin  
10 fin  
11 fin  
   | // Le coût est zéro si le niveau de dégradation ne dépasse  
   | pas encore  $M$   
12 retourner  $cout$ 
```

Bien que le résultat n'est pas stable, nous pouvons arriver à une valeur approximative après plusieurs fois d'appliquer l'algorithme 4 sur l'intervalle $[0.1, 4.1] \times [1, 20]$:

$$(\Delta T^*, M^*, o^*) = (0.1133949, 17.83996, 30.63433)$$

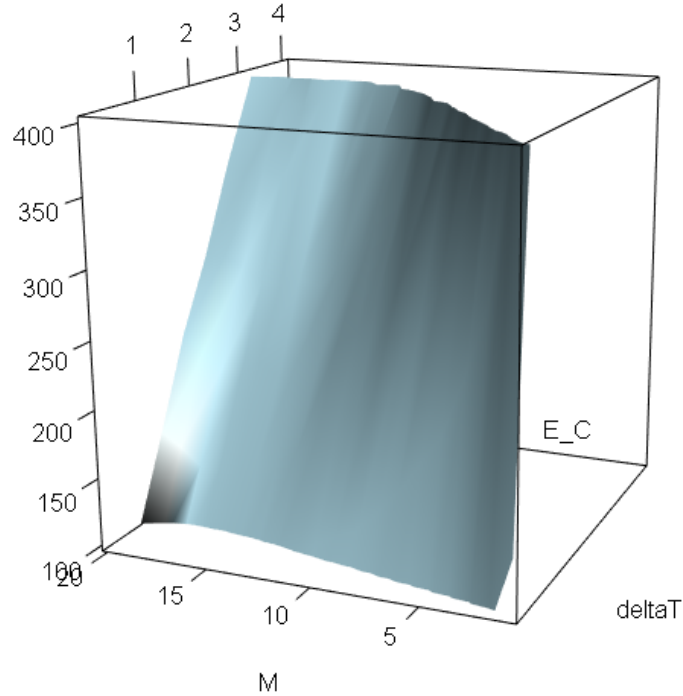
Algorithme 3 : Simulation de processus et calculer $\widehat{\mathbb{E}(C)}$

Entrées : L'intervalle d'inspection ΔT , le seuil M , le nombre de simulation $nSim$, le nombre d'inspection maximale $nStep$

Output : La valeur approximative $\widehat{\mathbb{E}(C)}$

- 1 Générer la matrice de l'incrément $simStep$ de taille $nSim \times nStep$ aux valeurs aléatoires selon la loi $\Gamma(a\Delta T, b)$;
 - 2 Calculer la matrice de processus $simProc$ de taille $nSim \times nStep$ en faisant la somme cumulative de $simStep$ horizontalement;
 - 3 $simTime \leftarrow$ les durées de vie de processus en appliquant l'algorithme (1);
 - 4 $simCost \leftarrow$ les coûts de processus en appliquant l'algorithme (2);
 - 5 $meanSimCost \leftarrow$ la moyenne de tous les coûts non nuls de $simCost$;
 - 6 $meanSimTime \leftarrow$ la moyenne de durée de vie non nuls de $simTime$;
 - 7 $meanCostTime \leftarrow \frac{meanSimCost}{meanSimTime}$;
// Si $meanSimCost$ ou $meanSimTime$ est NaN :
 $meanCostTime \leftarrow MAX_DOUBLE$
 - 8 retourner $meanCostTime$
-

FIGURE 7 – Représentation graphique de $E(C)$



Algorithme 4 : Optimiser la politique de maintenance basée sur dé-gradation

Entrées : $[L_{\Delta T}, U_{\Delta T}]$ l'intervalle de recherche de ΔT , $[L_M, U_M]$ l'intervalle de recherche de M , τ tolérance, $d_{\Delta T}$ nombre de points à évaluer pour ΔT , d_M nombre de points à évaluer pour M , $e_{\Delta T}$ ratio de réduction sur l'intervalle de ΔT , e_M ratio de réduction sur l'intervalle de M

Output : ΔT^* , M^* et o^* valeur objective optimale

// Suivre le principe de diachotomie

```

1 tant que  $\left(\frac{L_{\Delta T} - U_{\Delta T}}{d_{\Delta T}}\right)^2 + \left(\frac{L_M - U_M}{d_M}\right)^2 \geq \tau$  faire
    // Evaluer
2    $I_{\Delta T} \leftarrow$  l'ensemble de  $d_{\Delta T}$  points égaux-distances de  $[L_{\Delta T}, U_{\Delta T}]$ ;
3    $I_M \leftarrow$  l'ensemble de  $d_M$  points égaux-distances de  $[L_M, U_M]$ ;
4    $Z \leftarrow$  résultats d'appliquer l'algorithme 3 pour chaque point sur la
      grille  $I_{\Delta T} \times I_M$ ;
    // Chercher le minimum
5    $(i^*, j^*) \leftarrow$  l'indice de premier minimum de  $Z$ ;
6    $\Delta T^* \leftarrow$  l'élément  $i^*$ -ième de  $I_{\Delta T}$ ;
7    $M^* \leftarrow$  l'élément  $j^*$ -ième de  $I_M$ ;
8    $o^* \leftarrow Z_{i^*, j^*}$ ;
    // Ajuster la nouvelle zone de recherche au tour du
      minimum actuel
9    $l_{\Delta T} \leftarrow L_{\Delta T} - U_{\Delta T}$ ;
10   $l_M \leftarrow L_M - U_M$ ;
11   $L_{\Delta T} \leftarrow \min\left(\Delta T^* + \frac{l_{\Delta T}}{d_{\Delta T}}, L_{\Delta T}\right)$ ;
12   $U_{\Delta T} \leftarrow \max\left(\Delta T^* - \frac{l_{\Delta T}}{d_{\Delta T}}, U_{\Delta T}\right)$ ;
13   $L_M \leftarrow \min\left(M^* + \frac{l_M}{d_M}, L_M\right)$ ;
14   $U_M \leftarrow \max\left(M^* - \frac{l_M}{d_M}, U_M\right)$ ;
15 fin
16 retourner  $(\Delta T^*, M^*, o^*)$ 

```

3 Annexe

3.1 L'importation de données de pannes

```
1 pannes = read.csv(  
2   file = "FailureTimes_5.csv",  
3   header = TRUE,  
4   sep = ",",  
5   dec = ".",  
6   colClasses = c("NULL", NA)  
7 )  
8 scale = 1000  
9 data = pannes$Heures / scale  
10 N = length(data)
```

3.2 Le premier histogramme de distribution de pannes

```
1 hist(  
2   data,  
3   breaks = 40,  
4   probability = TRUE,  
5   xlab = "Date de pannes (mille heures)",  
6   ylab = "Densité",  
7   main = "Premier histogramme"  
8 )
```

3.3 Estimer le mixage de la loi Exponentielle et Gamma

```
1 # Fitting mixture of Exp and Gamma  
2 # Algorithm EM  
3 # Initialisation  
4 k = 2 # number of components  
5 p = c(0.5, 0.5)  
6 lambda = 1  
7 alpha = 10  
8 beta = 2  
9 f = list(  
10   '1' = function(x) {  
11     dexp(x, rate = lambda)  
12   },  
13   '2' = function(x) {  
14     dgamma(x, shape = alpha, rate = beta)  
15   }  
16 )  
17 # Illustration initial  
18 f_theta = function(x) {  
19   p[[1]] * f[[1]](x) + p[[2]] * f[[2]](x)  
20 }  
21 h_theta = hist(  
22   data,  
23   breaks = 40,  
24   probability = TRUE,  
25   main = "Mixage de la loi Exponentielle et Gamma",  
26   xlab = "Dates de panne (mille d'heures)",  
27   ylab = "Densité"  
28 )  
29 curve(  
30   f_theta(x),  
31   add = TRUE,  
32   col = "violet",  
33   from = min(h_theta$mids),  
34   to = max(h_theta$mids)  
35 )  
36 epsilon = list(  
37   alpha = 1e-4,  
38   theta = 1e-4
```

```

39 )
40 zeta = matrix(
41   0,
42   nrow = k,
43   ncol = N
44 )
45 # Norm
46 normVec = function(x) sqrt(sum(x^2))
47 # New value
48 p_new = p
49 alpha_new = alpha
50 beta_new = beta
51 lambda_new = lambda
52 repeat {
53   ## E Step
54   # Calculate each proba
55   for (l in 1:k) {
56     zeta[l,] = p[[l]] * f[[l]](data)
57   }
58   # Normalize proba
59   zeta = t(t(zeta) / rowSums(t(zeta)))
60   ## M step
61   # Lambda
62   lambda_new = sum(zeta[1,]) / sum(zeta[1,] * data)
63   # Alpha
64   c = log(sum(zeta[2,] * data) / sum(zeta[2,])) - sum(zeta[2,] * log(data))
65     / sum(zeta[2,])
66   alpha_new = 0.5 / c
67   alpha_temp = 0
68   repeat {
69     alpha_temp = 1 / (1 / alpha_new + (log(alpha_new) - digamma(alpha_new)
70       - c) / (alpha_new^2 * (1 / alpha_new - trigamma(alpha_new))))
71     if (abs(alpha_temp - alpha_new) < epsilon$alpha) {
72       break
73     } else {
74       alpha_new = alpha_temp
75     }
76   }
77   alpha_new = alpha_temp
78   # Beta
79   beta_new = alpha_new * sum(zeta[2,]) / sum(zeta[2,] * data)
80   # P
81   for (l in 1:k) {
82     p_new[[l]] = mean(zeta[l,])
83   }
84   ## Evaluation
85   if (normVec(c(alpha, beta, lambda, p[[1]], p[[2]]) - c(alpha_new, beta_
86     new, lambda_new, p_new[[1]], p_new[[2]])) < epsilon$theta) {
87     break
88   } else {
89     alpha = alpha_new
90     beta = beta_new
91     lambda = lambda_new
92     p = p_new
93   }
94 }
95 # Final value update
96 alpha = alpha_new
97 beta = beta_new
98 lambda = lambda_new
99 p = p_new
100 # Illustration final
101 f_theta = function(x) {
102   p[[1]] * f[[1]](x) + p[[2]] * f[[2]](x)
103 }
104 h_theta = hist(
105   data,
106   breaks = 40,
107   probability = TRUE,
108   main = "Mixage de la loi Exponentielle et Gamma",
109   xlab = "Dates de panne (mille d'heures)",
110   ylab = "Densité"
111 )
112 curve(

```

```

110 f_theta(x),
111 add = TRUE,
112 col = "violet",
113 from = min(h_theta$mids),
114 to = max(h_theta$mids)
115 )
116 # Kolmogorov-Smirnov test
117 F_theta = function(x) {
118   p[[1]] * pexp(x, rate = lambda) + p[[2]] * pgamma(x, shape = alpha, rate
119   = beta)
120 }
121 test = ks.test(data, F_theta, exact = TRUE)

```

3.4 Optimiser le coût moyenne sur une durée de temps

```

1 # Finding optimal t_0
2 # Given F_theta
3 c_c = 1200
4 c_p = 800
5 E_C_S = function(x) {
6   (c_c - c_p) * F_theta(x) + c_p
7 }
8 E_S = function(x) {
9   x - integrate(F_theta, 0, x)$value
10 }
11 E_C = function(x) {
12   E_C_S(x) / E_S(x)
13 }
14 o = optimize(
15   E_C,
16   c(min(data), max(data)),
17   tol = 1e-5
18 )
19 d = seq(
20   min(data),
21   max(data),
22   0.01
23 )
24 plot(
25   d,
26   lapply(
27     d,
28     E_C
29   ),
30   main = "Coût moyenne sur une durée de temps",
31   xlab = "t_0",
32   ylab = "",
33   type = "l"
34 )

```

3.5 Importer les valeurs de dégradation

```

1 # Import degradation
2 table = read.csv(
3   file = "DegradLevel_2.csv",
4   header = TRUE,
5   sep = ",",
6   dec = "."
7 )
8 scale = 1000
9 time = c(0, table$Temps / scale)
10 nbProcess = length(table) - 2
11 process = matrix(
12   ,
13   nrow = nbProcess,
14   ncol = 1 + length(table[[3]])
15 )
16 for (i in 1:nbProcess) {

```

```

17 |     process[i,] = c(0, table[[i + 2]]) # degrad = 0 at t = 0
18 | }

```

3.6 Premiers traces de dégradation

```

1 | # Plot process curves
2 | L = 20
3 | # Colormap
4 | library(RColorBrewer)
5 | color = brewer.pal(nbProcess, "Paired")
6 | # First process
7 | plot(
8 |     NULL,
9 |     type = "n",
10 |    main = "Trace de dégradation du système",
11 |    xlim = c(min(time), max(time)),
12 |    xlab = "Temps (mille heures)",
13 |    ylim = c(0, L),
14 |    ylab = "Niveau de dégradation"
15 | )
16 | for (i in 1:nbProcess) {
17 |   lines(
18 |     x = time,
19 |     y = process[i,],
20 |     type = "s",
21 |     col = color[[i]],
22 |     lwd = 2
23 |   )
24 | }

```

3.7 Estimation de paramètres de dégradations

```

1 | ## Estimate parameters for process
2 | lag = 2
3 | delta = 0.8 * lag
4 | d = t(diff(t(process), lag))
5 | # Concatenate into one vector
6 | increments = vector(
7 |   mode = "numeric",
8 |   length = length(d)
9 | )
10 | n = dim(d)[[1]]
11 | l = dim(d)[[2]]
12 | for (i in 1:n) {
13 |   increments[((i - 1) * l + 1):(i * l)] = d[i,]
14 | }
15 | # Filter out NA values
16 | increments = increments[!is.na(increments)]
17 | # Histogram
18 | hiso_degrad = hist(
19 |   increments,
20 |   breaks = 10,
21 |   probability = TRUE,
22 |   main = "Distribution des accroissements",
23 |   xlab = "dx",
24 |   ylab = "Densité"
25 | )
26 | # Estimate gamma distribution
27 | library(MASS)
28 | estim = fitdistr(
29 |   increments,
30 |   dgamma,
31 |   list(
32 |     shape = 1,
33 |     rate = 1
34 |   )
35 | )
36 | # Draw estimated density

```



```

37 a = estim$estimate[[1]] / delta
38 b = estim$estimate[[2]]
39 curve(
40   dgamma(x, a * delta ,b),
41   add = TRUE,
42   col = "red"
43 )
44 # Kolmogorov-Smirnov test
45 test = ks.test(increments, "pgamma", a * delta, b)
46 c(delta, a, b, test$p.value) # Print values
47 # PDF
48 f = function(x, t) {
49   dgamma(x, shape = a * t, rate = b)
50 }
51 # CDF
52 F = function(x, t) {
53   pgamma(x, shape = a * t, rate = b)
54 }
55 # Generator
56 rG = function(n, t) {
57   rgamma(n, shape = a * t, rate = b)
58 }

```

3.8 Optimisation de maintenance basée sur dégradations

```

1 # Monte carlo to optimize degradation
2 set.seed(2019)
3 L = 20
4 cost = c(10, 800, 1200) # Inspection, Preventive, Corrective
5 C_S = function(
6   p,
7   M
8 ) {
9   c = 0
10  p = p[!is.na(p)]
11  len = length(p)
12  if (p[[len]] >= M) {
13    n_s = length(p[p < M]) # Last index before >= M
14    c = c + cost[[1]] * n_s # Inspection
15    if (p[[n_s + 1]] >= L) {
16      c = c + cost[[3]] # Corrective
17    } else {
18      c = c + cost[[2]] # Preventive
19    }
20  }
21  return(c)
22 }
23 S = function(
24   p,
25   deltaT,
26   M
27 ) {
28   p = p[!is.na(p)]
29   len = length(p)
30   if (p[[len]] < M) return(0)
31   else return(length(p[p < M]) + 1)
32 }
33 E_C = function(
34   deltaT,
35   M,
36   nSim = 200,
37   nStep = 40,
38   silence = TRUE
39 ) {
40   simStep = matrix(
41     rG(nSim * nStep, deltaT),
42     nrow = nSim,
43     ncol = nStep
44   )
45   simProc = t(apply(
46     t(simStep),

```

```

47     2,
48     cumsum
49 ))
50 # Count cost
51 simCost = apply(
52     t(simProc),
53     2,
54     C_S,
55     M
56 )
57 # Count time
58 simTime = apply(
59     t(simProc),
60     2,
61     S,
62     deltaT,
63     M
64 )
65 meanSimCost = mean(simCost[simCost > 0])
66 meanSimTime = mean(simTime[simTime > 0])
67 meanCostTime = 0
68 if (is.nan(meanSimCost) || is.nan(meanSimTime)) meanCostTime = .Machine$
69     double.xmax
70 else meanCostTime = meanSimCost / meanSimTime
71 # For debug purpose
72 if (!silence)
73     cat(
74         "dT =", deltaT,
75         "M =", M,
76         "mCost =", meanSimCost,
77         "mTime =", meanSimTime,
78         "mCostTime =", meanCostTime,
79         "\n"
80     )
81 return(meanCostTime)
82 }
83 # Optimize
84 optim_degrad = function(
85     lower_deltaT,
86     upper_deltaT,
87     lower_M,
88     upper_M,
89     tol = 1e-2,
90     div_deltaT = 5,
91     div_M = 5,
92     cut_deltaT = 2,
93     cut_M = 2
94 ){
95     deltaT = 0
96     M = 0
97     obj = 0
98     while (((upper_deltaT - lower_deltaT) / div_deltaT)^2 + ((upper_M -
99         lower_M) / div_M)^2) >= tol^2) {
100         interval_deltaT = seq(lower_deltaT, upper_deltaT, length.out = div_
101             deltaT)
102         interval_M = seq(lower_M, upper_M, length.out = div_M)
103         cat("deltaT =", interval_deltaT,
104             "\nM =", interval_M,
105             "\n")
106         Z = outer(
107             interval_deltaT,
108             interval_M,
109             Vectorize(E_C)
110         )
111         # First minimum position
112         posMin = arrayInd(which.min(Z), dim(Z))
113         deltaT = interval_deltaT[[posMin[[1]]]]
114         M = interval_M[[posMin[[2]]]]
115         obj = Z[posMin]
116         cat("deltaT = ", deltaT,
117             "\nM = ", M,
118             "\nobj = ", obj,
119             "\n")
120     }
121     # Adjust zone of optimize

```

```

118     l_deltaT = upper_deltaT - lower_deltaT
119     l_M = upper_M - lower_M
120     upper_deltaT = min(deltaT + l_deltaT / cut_deltaT, upper_deltaT)
121     lower_deltaT = max(deltaT - l_deltaT / cut_deltaT, lower_deltaT)
122     upper_M = min(M + l_M / cut_M, upper_M)
123     lower_M = max(M - l_M / cut_M, lower_M)
124 }
125 return(list(
126     deltaT = deltaT,
127     M = M,
128     obj = obj
129 ))
130 }
131 # Illustration
132 interval_deltaT = seq(0.1, 4.1, 0.5)
133 interval_M = seq(1, 20, 1)
134 Z = outer(
135     interval_deltaT,
136     interval_M,
137     Vectorize(E_C)
138 )
139 library(rgl)
140 persp3d(
141     interval_deltaT,
142     interval_M,
143     Z,
144     xlab = "deltaT",
145     ylab = "M",
146     zlab = "E_C",
147     zlim = c(100, 400),
148     col = "lightblue"
149 )
150 rgl.snapshot(filename = "E_C_degrad.png")

```

Références

- [1] Minka, Thomas P. (2002). "Estimating a Gamma distribution"
<https://tminka.github.io/papers/minka-gamma.pdf>