



Modélisation et simulation discrètes
Laboratoire n° 1

Espérance de vie d'un système

Daniel Lifschitz et Nicolas Seriot,
IL2005b

20 décembre 2004

Résumé

Ce laboratoire concerne la mesure de la durée de vie d'un réseau — dont les composantes tombent en panne selon une variable aléatoire — par une simulation discrète de type Monte-Carlo.

Dans ce rapport, nous analysons les performances de certaines politiques de réparation des composantes ; nous donnons et discutons les résultats de nos simulations ; nous proposons et analysons une politique de maintenance, plus efficace que celle proposée.

Table des matières

1	Introduction	3
2	Le programme de simulation	3
2.1	Schéma de la simulation	3
2.2	La politique 3	3
2.2.1	Critiques et améliorations possibles...	7
3	Mesures et discussions	8
3.1	data1, politique 1, espérance du système (point 4)	8
3.2	data1, politique 2, espérance du système en fonction de la durée de réparation (point 5)	9
3.3	data2-3, politique 1, espérance du système (point 6)	10
3.4	data2-3, politique 2, espérance du système en fonction de la durée de réparation (point 7)	10
4	Comparaison des politiques 2 et 3 (point 8)	13
4.1	Gains réalisés	13
4.2	Convergence	14
4.3	Recherche du point d'intersection	15
4.4	Une espérance de vie infinie	16
5	Conclusion	17
6	Annexes	19
6.1	Les jeux de données	19
6.1.1	Le jeux de données data1	19
6.1.2	Le jeux de données data2	20
6.1.3	Le jeux de données data3	21
6.1.4	Le jeux de données data4	22
6.2	Tableaux	23
	Listings	29

1 Introduction

Ce laboratoire concerne la simulation d'un système stochastique, un réseau, formé de multiples composantes soumises à des pannes *cataleptiques* (soudaines et complètes). L'objectif est de mesurer et d'analyser l'espérance de vie de différents réseaux qui nous sont fournis, ainsi que d'étudier l'influence du travail d'un réparateur qui remet les composantes en état en un temps donné. Ce réparateur peut agir selon différentes politiques ; il s'agira d'inventer une politique plus efficace que celle qui nous est proposée et qui consiste à réparer la plus ancienne des composantes en panne. Cette simulation est discrète, de type Monte-Carlo.

Les réseaux qui nous sont donnés, `data1`, `data2` et `data3` sont reproduits en annexe, page 19.

2 Le programme de simulation

2.1 Schéma de la simulation

Dans son principe, le programme simule la durée de vie d'un réseau autant de fois que nécessaire pour obtenir des résultats suffisamment¹ significatifs. Pour ce faire, il simule aléatoirement l'apparition d'*événements*.

Ces événements sont *discrets* ; ils peuvent être, à choix², la panne d'une composante, sa réparation ou la fin de la simulation, qui survient quand le réseau ne fonctionne plus. À chacun de ces événements correspondent une composante et un temps. Les événements sont insérés et prélevés dans un *échancier*.

L'échancier fonctionne sur le principe d'une queue de priorité, dans laquelle on insère des événements qui vont se placer selon le temps auquel ils vont survenir ; ainsi on obtient toujours le prochain événement en prélevant l'élément du côté du temps le plus petit.

Une fois que l'on a retiré le prochain événement de l'échancier, il nous faut le traiter. Ce traitement se fait selon l'algorithme 1.

2.2 La politique 3

Trouver une politique de réparation efficace permettant d'augmenter sensiblement la durée de vie d'un réseau a été le point le plus difficile — et aussi celui qui nous a demandé le plus de temps — de ce laboratoire. En effet, que de frustrations, après avoir codé plusieurs dizaines ou centaines de lignes de code, en réalisant que le gain par rapport à la politique 2 ne se chiffrait qu'à quelques misérables pour-cents ou, pire encore, de constater que les résultats étaient inférieurs à celle-ci.

¹Le seuil de signification se détermine par la demi-largeur de l'intervalle de confiance à 99%, passée en paramètre sur la ligne de commande.

²Nous allons voir quel intérêt il peut y avoir à ajouter un type d'événement lors de la présentation de notre politique 3.

Algorithme 1 Traitement des événements

```
prélever le prochain événement
avancer l'horloge à la date de l'événement

si l'événement est une panne alors
    mettre la composante en panne
sinon si l'événement est la fin d'une réparation alors
    libérer le réparateur
    remettre la composante en service...
    ...et planifier sa prochaine panne
sinon si l'événement est la fin de la simulation alors
    arrêter la simulation
    retourner le temps courant
finsi

si le réseau ne fonctionne plus alors
    insérer la fin de la simulation dans l'échéancier, au temps présent
sinon
    si le réparateur est libre alors
        appliquer la politique de réparation choisie
    finsi
finsi
```

Parmi les pistes explorées, nous avons essayé de combiner différentes analyses et décisions comme la réparation des éléments de plus grande espérance, la détection de « composants couplés » (dont la panne de l'une fait que l'autre devient critique), l'affectation d'une priorité à chaque composante suite à l'analyse initiale du réseau ou encore la détermination de composants qu'il est inutile de réparer. Nous avons également tenté d'effectuer des simulations dans la simulation, que ce soit lors de l'analyse du réseau ou lors de chaque application de la politique 3.

Pour finir, nous avons imaginé une solution qui permet, dans certains cas, de quasiment tripler la durée de vie du réseau selon l'application de la politique 2. Pour réussir cette performance, notre politique 3 s'axe sur quatre points :

1. pour choisir quelle pièce réparer, nous essayons de deviner le futur en calculant, pour chaque pièce, les probabilités qu'ont les autres composantes du réseau de tomber en panne quand le réparateur sera à nouveau disponible ;
2. parmi toutes les pièces réparables du point 1, nous choisissons celle qui a la plus grande espérance ;
3. il est parfois nécessaire de ne pas réparer immédiatement une composante en panne ; il vaut mieux attendre le dernier moment pour la remettre en service ;
4. enfin, si on décide d'attendre avant de réparer une composante, il faut pouvoir donner l'ordre de commencer la réparation à n'importe quel moment, et non pas seulement lorsqu'une nouvelle composante tombe en panne ; faute de quoi, le réparateur risque d'être débordé.

Ces deux derniers points sont certainement les plus importants : c'est grâce à eux que nous pouvons optimiser le travail du réparateur. En effet, imaginons un réseau très simple, qui n'a que deux composantes e_1 et e_2 . Pour que le réseau continue de fonctionner, il suffit que e_1 ou e_2 soit en service. Si on applique la politique naïve qui consiste à réparer une composante dès qu'une panne survient, on rencontre assez vite le cas où l'autre composante tombe en panne pendant la réparation de la première (figure 1).

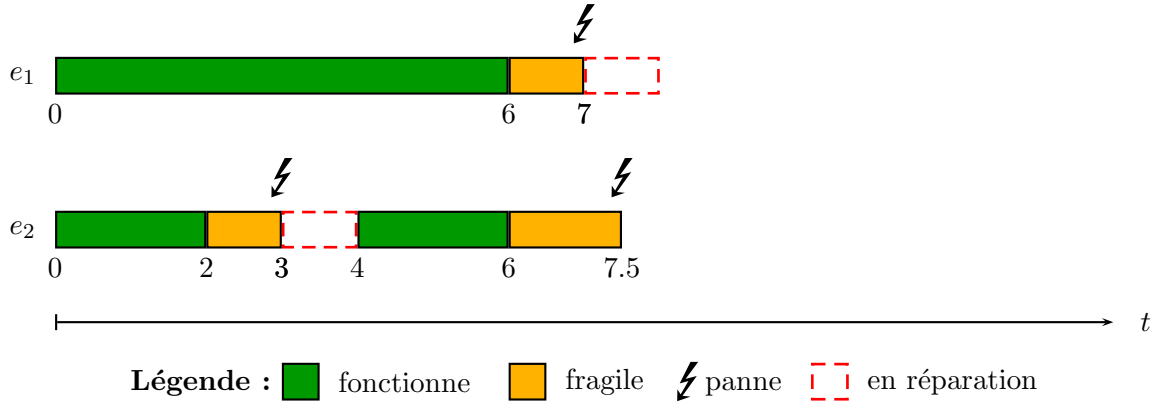


FIG. 1 – Politique de réparation naïve — les composantes sont réparées dès qu'elles tombent en panne.

Maintenant, posons d , la durée de réparation qui vaut 1 unité de temps, et t , le temps courant. Si la composante e_2 peut tomber en panne à partir de $t+2$, et que la composante e_1 peut tomber en panne dès $t+6$, l'idéal serait de retarder le début de la réparation de e_2 jusqu'au temps $t+5$, afin de la remettre en service juste avant que sa consœur ne soit susceptible de tomber en panne. Cette situation est illustrée à la figure 2, avec les mêmes paramètres que la figure 1.

Bien sûr, on ne sait pas quand e_1 va tomber en panne. Par contre, étant donné que son espérance est basée sur une loi triangulaire, on sait avec certitude jusqu'à quand elle va fonctionner. Il suffit alors de terminer sa réparation juste avant que la probabilité de panne de la composante e_2 soit supérieure à zéro.

C'est la raison pour laquelle nous avons ajouté un nouveau type d'événement nommé *réveil*. Cet événement est ajouté dans l'échéancier lorsqu'il existe une ou plusieurs composantes en panne, mais qu'il est préférable d'attendre avant d'entamer une réparation. Son échéance est basée sur une fraction de la durée d'une réparation, qui dépend de l'espérance moyenne de toutes les composantes du réseau.

La combinaison de l'événement *réveil* et de la technique de la « réparation reportée » s'avère très efficace, notamment pour des temps de réparation brefs, comme nous le verrons en chiffres au point 4 page 13. Le réveil permet de reporter la réparation, mais pas plus qu'il ne faut. En effet, sans réveil, on ne commence la réparation qu'au prochain événement qui, logiquement, ne peut être qu'une panne.

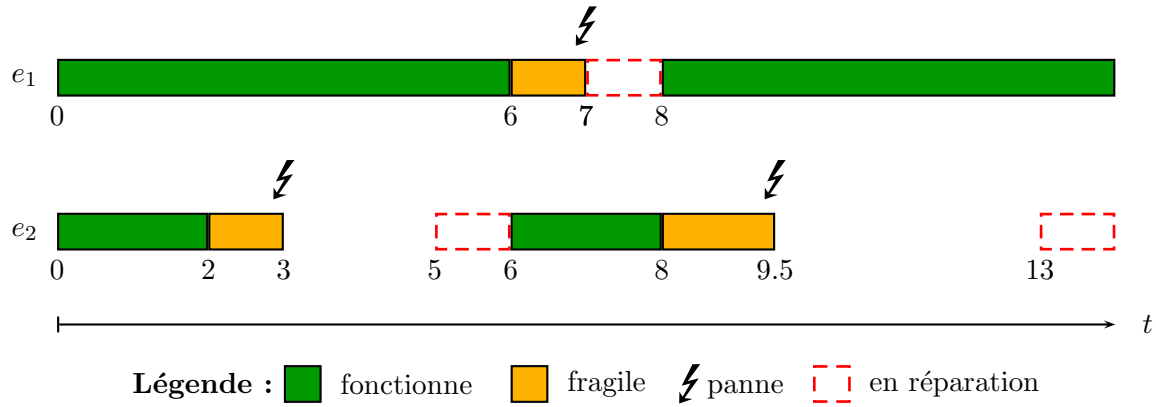


FIG. 2 – Politique de réparation efficace — la réparation est retardée pour que la composante « fraîche » prenne le relais de celle qui peut tomber en panne. On note au passage que dans la situation particulière qui est illustrée, cela permet d’entrer dans un « cycle vertueux » où la durée de vie du réseau est infinie (la situation au temps 6 est la même qu’au temps 14). Ce n’est bien sûr pas toujours le cas...

Nous avons mentionné que l’échéance du réveil était basée sur une fraction de la durée d’une réparation. Nous avons choisi cette formule car nous n’avions pas la possibilité de modifier la signature des fonctions dans le programme afin retourner un temps déterminé. Pour être précis, la fonction `reseau_conseil_3()` ne peut retourner qu’un entier qui est la composante à réparer. Nous avons donc choisi de retourner un entier négatif pour indiquer qu’il est nécessaire d’enclencher le réveil au temps :

$$\text{horloge} + \frac{-1}{\text{valeur de retour}}$$

La valeur de retour de `reseau_conseil_3()`, lorsqu’elle demande l’enclenchement du réveil est donnée par la formule suivante :

$$-\left(\frac{100 \cdot \text{durée d'une réparation}}{\text{moyenne des espérances}} + 1\right)$$

Le choix de prendre la durée d’une réparation comme base de calcul s’est imposé du fait que c’est la seule unité de temps que nous avons en commun entre `reseau.c` et `simulation.c`. Pour la moyenne des espérances, nous avons constaté que si le temps de réparation était trop long par rapport à la durée de vie du système, nous remettons en service les composantes trop tôt. Ne connaissant pas encore la durée de vie du système, nous avons donc choisi un indicateur de cette espérance, qui n’est autre que l’espérance moyenne de toutes les composantes du réseau.

Nous pouvons maintenant donner l’algorithme général de la politique 3 (algorithme n° 2). Celui-ci mérite encore une explication concernant l’indice de la boucle `for`. Cette valeur détermine jusqu’à quelle échéance nous allons explorer « le futur ». L’indice maximal doit être suffisamment grand pour permettre au réparateur de terminer la réparation de toutes les composantes en panne avant que le réseau ne soit plus opérationnel, raison pour laquelle l’indice maximal dépend du nombre de pièces qui ne fonctionnent pas.

Algorithme 2 Politique 3

```
déterminer le nombre de composantes en panne ;
si il n'y pas de composante en panne alors
    ne rien faire jusqu'au prochain événement ;
sinon
    pour  $i$  allant de 1 à 2 + le nombre de composantes en panne faire
        pour chaque composante en service faire
            déterminer sa probabilité de tomber en panne au temps horloge + ( $i \cdot$  durée
            d'une réparation) ;
        fin pour
        pour chaque composante ayant une probabilité de panne  $> 0$ , dans l'ordre dé-
        croissant faire
            simuler sa panne ;
            si le réseau est en panne alors
                déterminer dans les composantes réellement en panne, laquelle permet de
                remettre le réseau en service ;
                réparer cette composante ;
                le réparateur est occupé ;
            fin politique 3 ;
        finsi
    fin pour
fin pour
    enclencher le réveil ;
finsi
```

Contrairement à ce qui est mentionné dans l'algorithme présenté, l'indice minimal de la boucle n'est pas toujours de 1 dans notre programme. En effet, si entre deux appels successifs de la fonction `reseau_conseil_3()` aucun changement n'est intervenu sur le réseau, il est possible de reprendre les calculs aux temps où nous nous étions arrêtés ; il est ainsi possible de réduire considérablement les temps de calcul.

2.2.1 Critiques et améliorations possibles...

Nous soupçonnons que notre politique 3 est peut-être un peu trop « procédurale » et que l'on doit pouvoir regarder le problème d'un point de vue plus mathématique. Nous pensons qu'avec un œil de mathématicien nous pourrions éventuellement reformuler le problème en quelque chose que l'on sait résoudre, ou dont on sait approcher la résolution.

Peut-être aussi qu'il serait possible d'améliorer notre politique 3 en gardant une trace du passé, en profitant des données acquises lors de précédentes expériences. En effet, c'est bien ce que l'on ferait dans la réalité. Nous avons là une information dont nous disposons mais que nous n'utilisons pas.

3 Mesures et discussions

3.1 data1, politique 1, espérance du système (point 4)

On demande, pour le jeu de données `data1` et la politique 1, de construire les intervalles de confiance (avec seuil de confiance à 99%) pour l'espérance de vie du système, pour des demi-largeurs maximales égales à 0.1, 0.09, ..., 0.01 unités de temps.

La figure 3 présente un graphique indiquant l'évolution du nombre de mesures nécessaires en fonction de l'intervalle de confiance désiré ainsi que les valeurs numériques pour les demi-largeurs demandées.

On constate que la relation entre la demi-largeur et le nombre de mesures nécessaires n'est pas linéaire. Comme nous l'avons vu au cours, la largeur de l'intervalle de confiance obtenu est proportionnelle à $\frac{S_X}{\sqrt{n}}$. Si tel est le cas, cela veut dire qu'il existe une fonction f qui, une fois élevée au carré nous donne le nombre d'expériences nécessaires pour un intervalle de confiance donné. Par régression linéaire, nous avons trouvé une fonction qui colle au modèle. Le résultat est présenté à la figure 4.

demi-largeur	nbre de mesures
0.10	116
0.09	161
0.08	217
0.07	283
0.06	375
0.05	556
0.04	854
0.03	1521
0.02	3328
0.01	13431

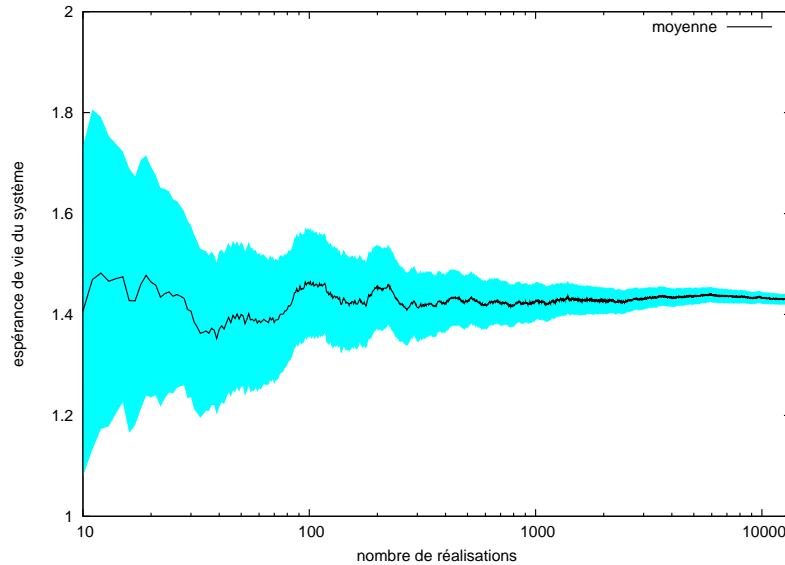


FIG. 3 – `data1`, politique 1 — Évolution du nombre de réalisations nécessaires en fonction de la précision requise (échelle horizontale logarithmique).

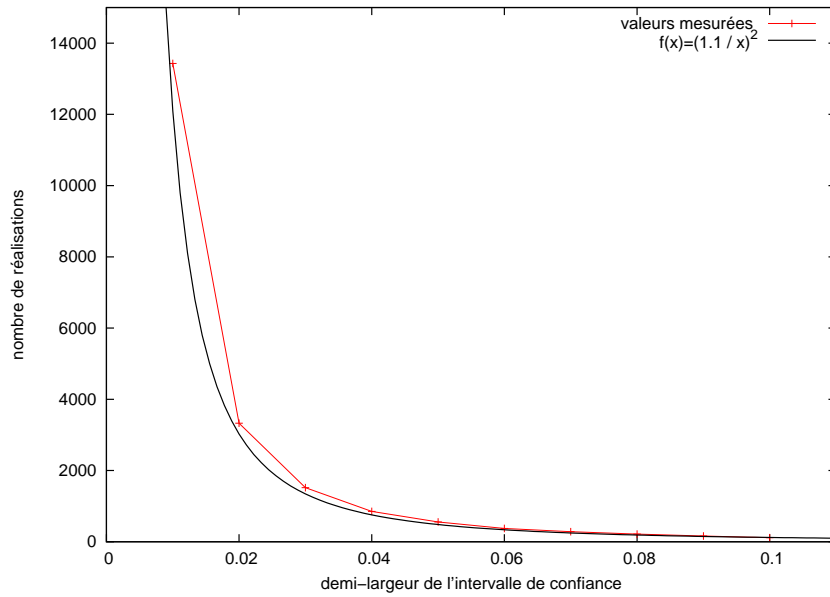


FIG. 4 – data1, politique 1 — Évolution du nombre de réalisations nécessaires en fonction de la précision requise.

3.2 data1, politique 2, espérance du système en fonction de la durée de réparation (point 5)

On demande, pour le jeu de données data1 et la politique 2, d'analyser l'espérance de vie du système en fonction de la durée d d'une réparation (en fixant la demi-largeur maximale des intervalles de confiance à 0.01 unités de temps), ceci en partant de $d = 1.0$ en faisant tendre cette durée vers zéro. Une représentation graphique de ces données est présentée à la figure 5, alors que le tableau des mesures est donné en annexe à la page 23.

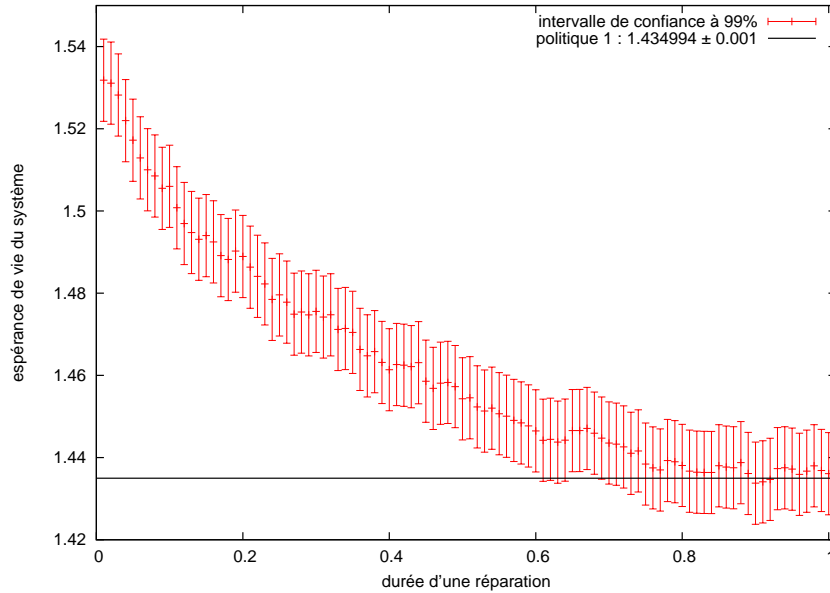


FIG. 5 – data1, politique 2 — Évolution de l’espérance de vie en fonction de la durée de réparation. La demi-largeur de l’espérance de vie avec la politique 1 est si faible qu’elle n’est pas représentée.

On observe ce que l’on comprend bien intuitivement, à savoir que l’espérance de vie du réseau est inversement proportionnelle à la durée de réparation. L’espérance de vie chute très vite quand la durée de réparation augmente de près de zéro jusqu’à 0.2. Ensuite, entre 0.8 et 1.0, la durée de vie se stabilise ; elle tend vers $1.434994(\pm 0.001)$ — toujours avec seuil de confiance à 99%. Cette durée de vie est celle constatée lors de l’application de la politique 1, c’est-à-dire sans rien réparer. On peut donc conclure qu’au delà d’une durée de réparation de 0.8 unités de temps, le réparateur n’a pratiquement pas d’influence sur la durée de vie du réseau.

3.3 data2-3, politique 1, espérance du système (point 6)

On demande, pour les jeux de données data2 et data3, avec la politique 1, de construire un intervalle de confiance à 99% de demi-largeur maximale égale à 0.01 pour l’espérance de vie du système. Ces intervalles sont donnés à la figure 6.

On peut simplement dire que, de par les différences de nature entre ces deux réseaux, la construction de notre intervalle de confiance est nettement plus longue et demande un plus grand nombre d’expériences dans le cas du réseau data3.

3.4 data2-3, politique 2, espérance du système en fonction de la durée de réparation (point 7)

On demande, pour les jeux de données data2 et data3, avec la politique 2, d’analyser l’espérance de vie du système en fonction de la durée d d’une réparation. Pour chaque expérience, nous avons fixé nous-mêmes la demi-largeur souhaitée des intervalles de confiance, en fonction des résultats déjà obtenus. Des représentations graphiques des

```
./Lab01 data2 1 1 0.01
nombre de realisations : 11078
moyenne : 6.384080
intervalle de confiance a 99% : [6.374081, 6.394080]

./Lab01 data3 1 1 0.01
nombre de realisations : 283235
moyenne : 46.658005
intervalle de confiance a 99% : [46.648005, 46.668005]
```

FIG. 6 – Construction d’intervalles de confiance pour data2 et data3.

données relatives à data2 et data3 sont présentées respectivement aux figures 7 et 8 (échelles logarithmiques). Les mesures numériques sont données en annexe aux pages 24 et 25.

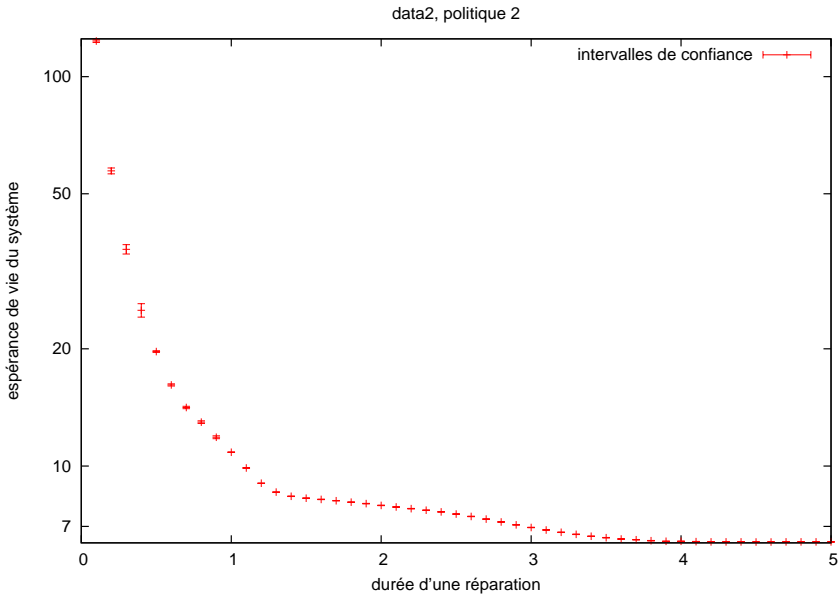


FIG. 7 – data2, politique 2 — Évolution de l’espérance de vie en fonction de la durée d’une réparation (échelle logarithmique).

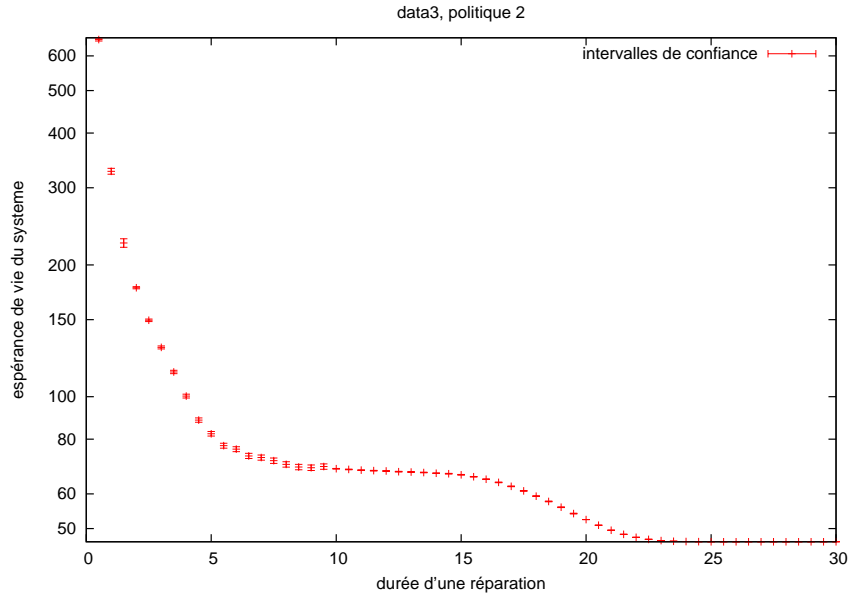


FIG. 8 – data3, politique 2 — Évolution de l’espérance de vie en fonction de la durée d’une réparation (échelle logarithmique).

Pour ne pas passer des heures à calculer des valeurs avec une précision superfétatoire, tout en conservant une bonne précision quand le temps de calcul n’était pas rédhibitoire, nous avons fixé les demi-largeurs selon la durée de réparation d (table 1). On voit sur le graphique qu’un intervalle de confiance plus large pour les durées de réparation les plus courtes ne gêne pas du tout la lecture.

data2	
durée d	demi-largeur
0.1–0.4	1.0
0.5–0.9	0.1
1.0–5.0	0.01

data3	
durée d	demi-largeur
0.5–1.5	5
2.0–9.5	1
10.0–19.5	0.1
20.0–30.0	0.01

TAB. 1 – Précision souhaitée en fonction de la durée de réparation.

Globalement, les graphiques obtenus confirment la « tendance » déjà observée au point 3.2 : un temps de réparation très court augmente très fortement la durée de vie du système (dès qu’une composante tombe en panne, elle est presque aussitôt réparée). Cependant, on observe un phénomène nouveau : on distingue comme un tassement, une zone dans laquelle, pour des durées de réparation moyennes, la durée de vie n’est pas affectée ou presque. Ce phénomène est rendu encore plus visible par l’utilisation d’une échelle logarithmique. Ces zones vont approximativement des durées de réparation 1.3 à 2.5 pour data2 et 8 à 16 pour data3.

4 Comparaison des politiques 2 et 3 (point 8)

Pour procéder à la comparaison des politiques 2 et 3 sur les fichiers `data2` et `data3`, nous avons effectué une série de simulations sur chacun des jeux de données en faisant varier la durée d'une réparation. Pour que les comparaisons soient le plus précises possible — tout en gardant des temps de calcul raisonnables — nous avons, pour chaque mesure, adapté la demi-largeur de l'intervalle de confiance afin qu'elle représente un demi pour-cent de l'espérance de vie du système. Les résultats numériques obtenus sont présentés en annexes aux pages 26, 27 et 28.

L'analyse des résultats porte sur les aspects suivants :

1. les gains obtenus sur l'espérance de vie du système lorsque la durée de réparation est petite ;
2. la convergence des politiques 2 et 3 lorsque la durée de réparation est grande ;
3. la recherche du point d'intersection entre les politiques 2 et 3.

Les sections suivantes présentent en détails chacun de ces points.

4.1 Gains réalisés

Les figures 9 et 10 présentent le graphique des gains obtenus avec le jeu de données `data2`, respectivement `data3`. On constate que pour des durées de réparation faibles, la politique 3 est nettement meilleure que la politique 2. Ensuite, au fur et à mesure que la durée de réparation augmente — et donc que l'espérance de vie du système chute — la différence entre les deux politiques s'amenuise jusqu'à ne représenter que quelques pour-cents de différence.

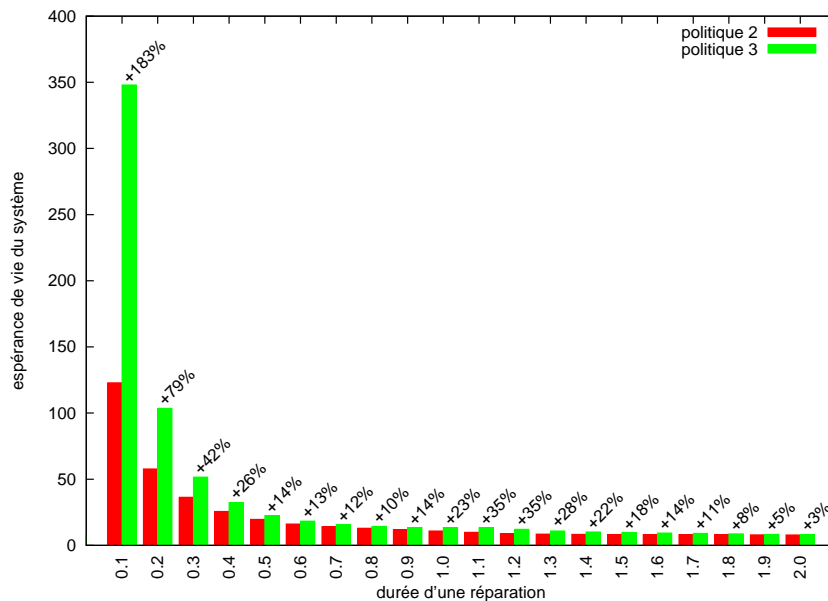


FIG. 9 – `data2`, comparaison des politiques 2 et 3 — Gains réalisés.

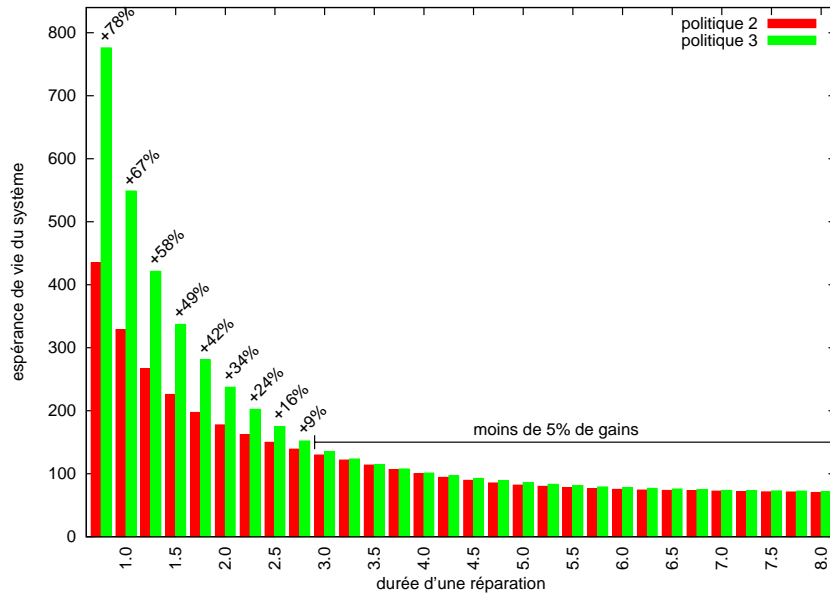


FIG. 10 – data3, comparaison des politiques 2 et 3 — Gains réalisés.

4.2 Convergence

La deuxième question que l'on peut se poser est de savoir si, à partir d'une certaine durée de réparation, les espérances de vie de chacun des deux systèmes sont identiques pour les deux politiques ou si, par malheur, la politique 3 se révèle moins performante. Les figures 11 et 12 permettent de lever le doute : la politique 3 n'est jamais moins bonne que la politique 2 !

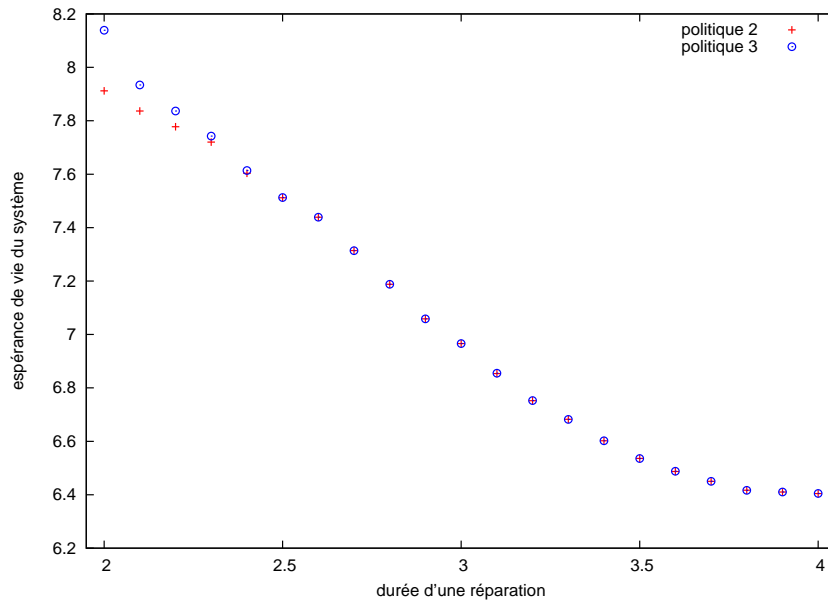


FIG. 11 – data2, comparaison des politiques 2 et 3 — Convergence des résultats.

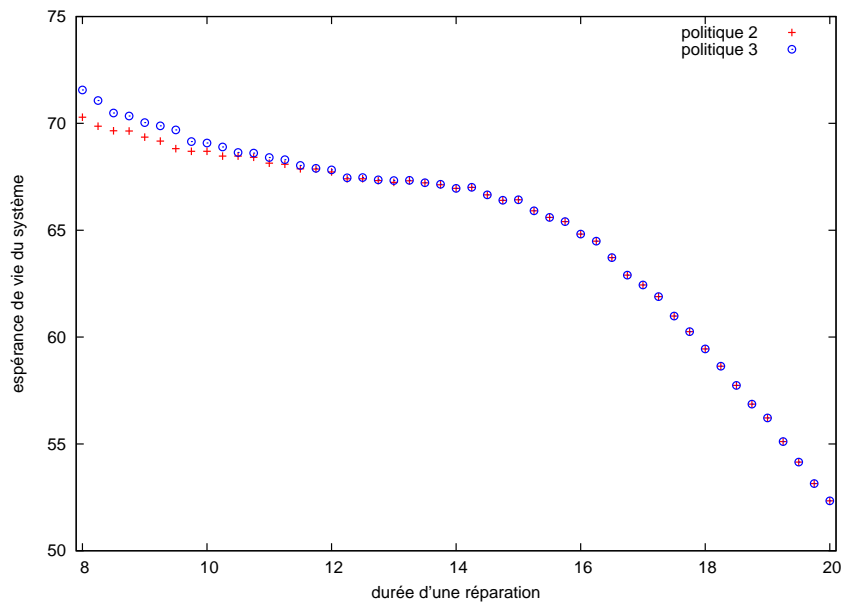


FIG. 12 – data3, comparaison des politiques 2 et 3 — Convergence des résultats.

4.3 Recherche du point d'intersection

Nous allons maintenant rechercher, pour chacun des fichiers de données, la durée de réparation à partir de laquelle l'espérance de vie des systèmes n'est plus significativement différente entre les politiques 2 et 3.

Pour le jeu de données `data2`, il est possible de constater visuellement sur la figure 13 que pour une durée de réparation inférieure ou égale à 2.1, la politique 3 est encore plus efficace, alors qu'à partir de 2.3, les performances ne sont plus significativement différentes. Pour trancher le cas de la valeur médiane, il est nécessaire d'effectuer un t test.

Selon la méthode proposée dans le cours (pages 133 et suivantes), nous obtenons l'intervalle de confiance $[-0.0761; 0.0328]$. L'intervalle contient 0 ; on peut donc conclure que, pour ces paramètres, les deux politiques ne sont pas significativement différentes.

Nous avons effectué la même opération pour le jeu de données `data3` (voir figure 14) pour une durée de réparation de 10.0. L'intervalle de confiance est le suivant : $[-0.6746; -0.0998]$; la politique 3 est donc plus performante.

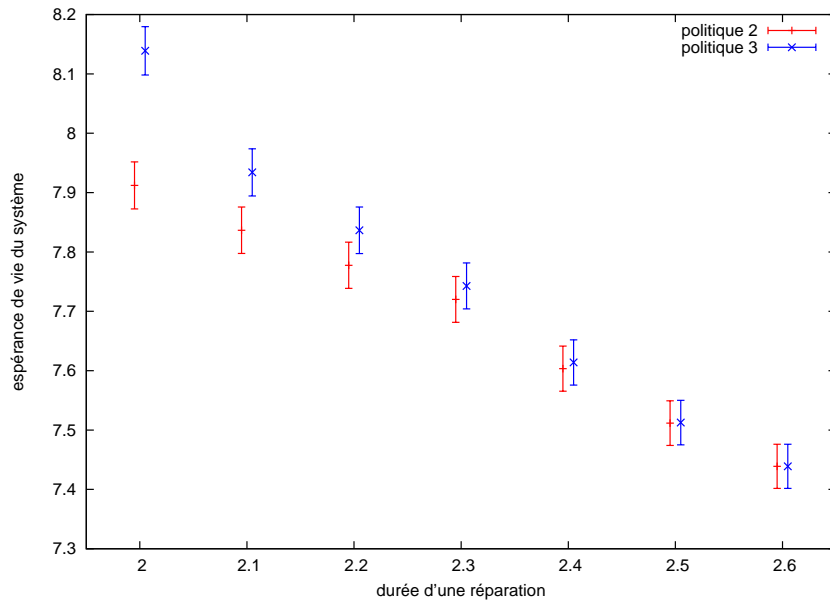


FIG. 13 – data2, comparaison des politiques 2 et 3 — Recherche de l'intersection.

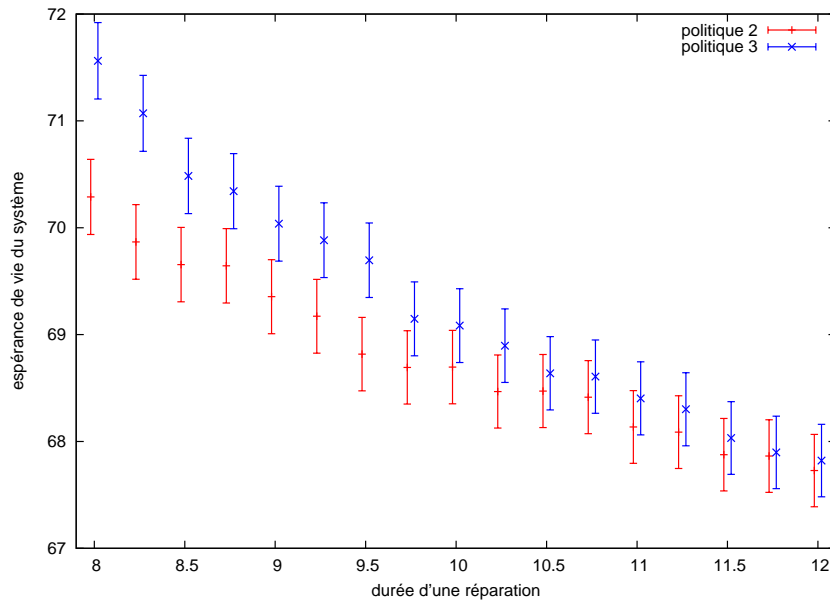


FIG. 14 – data3, comparaison des politiques 2 et 3 — Recherche de l'intersection.

4.4 Une espérance de vie infinie

Pour finir, nous avons construit un nouveau réseau data4 (voir 6.1.4 en page 22). Celui-ci ne comporte que trois composantes c_i . Ces composantes ont la particularité d'être associées à des lois triangulaires telles que :

$$\alpha_i > (\beta_i - \alpha_i) \quad \forall c_i$$

La figure 15 donne les résultats mesurés pour les politiques 1, 2 et 3 avec une durée de réparation de 1.

```
./Labo1 data4 1 1 1
statistiques affichage des resultats
nombre de realisations : 30
moyenne : 26.962581
variance : 0.787209
intervalle de confiance a 99% : 26.544651-27.380512

./Labo1 data4 1 2 1
statistiques affichage des resultats
nombre de realisations : 38893
moyenne : 101.119791
variance : 5842.855251
intervalle de confiance a 99% : 100.119799-102.119783

./Labo1 data4 1 3 1
^C
```

FIG. 15 – Politique 1, 2 et 3 pour le réseau data4 pour une durée de réparation de 1.

Pour la politique 3, nous avons dû interrompre le programme, car visiblement celui-ci se trouvait dans une boucle infinie. À l'aide de traces dans le programme, nous avons constaté que l'horloge de la simulation dépassait allègrement la valeur de 100 millions, sans qu'aucune panne complète du réseau ne soit survenue.

Sans pouvoir le prouver mathématiquement, nous constatons qu'en appliquant la politique 3 sur ce réseau, il existe une durée de réparation se situant entre 1 et 1.5, à partir de laquelle le réseau ne tombe plus jamais en panne.

5 Conclusion

Nous avons trouvé ce laboratoire tout à fait intéressant et ceci à plus d'un titre. En effet, il s'agit d'un laboratoire qui dépasse la simple application de techniques vues au cours; le fait qu'il n'y ait pas *une* réponse juste mais que le champ de recherche soit complètement ouvert a été quelques chose de très stimulant.

D'autre part, ce laboratoire nous a permis de mieux comprendre les éléments du cours comme les indicateurs statistiques, la notion de convergence, de demi-largeur, etc. Les particularités du passage de la théorie mathématique à la programmation d'une machine (échancier, formules itératives, ...) sont autant de raisons de l'intérêt que nous avons porté à ce laboratoire.

Le laboratoire s'est déroulé sur 8 semaines. Nous avons arrêté les améliorations successives apportées à notre politique 3 à 10 jours du rendu du travail, pour rédiger le rapport.

Comme d'autres groupes peut-être, il nous faut relever que le choix du langage n'est peut-être pas le plus pertinent. Si nous devions écrire nous-même un nouveau programme de simulation, nous choisirions certainement un langage de plus haut niveau que le C, comme Java ou Python par exemple. Nous trouvons que la syntaxe et l'insécurité du langage C favorisent les petites erreurs de codage et donc la perte de temps. De plus, elles n'aident pas à garder une vue d'ensemble sur ce que fait le programme.

Nous retenons encore de ce laboratoire l'importance de l'intuition. Nous avons l'habitude de comprendre et d'appliquer des algorithmes, mais trop peu d'occasions de développer nos propres algorithmes et encore moins de développer des algorithmes stochastiques. Nous avons pu nous rendre compte à quel point notre intuition est parfois trompeuse. Plus d'une fois nous avons été convaincus d'avoir trouvé une idée tout à fait géniale, pour déchanter une fois la politique programmée et les mesures relevées... En tous les cas nous savons maintenant mesurer et comparer d'un point de vue statistique différentes politiques et déterminer mathématiquement laquelle est la meilleure. Il reste que de l'intuition et de nombreux essais parfois hasardeux ont été nécessaires à l'élaboration de notre solution.

Enfin, ce laboratoire nous a montré les différentes étapes de la réalisation d'une simulation informatique discrète. Nous savons maintenant mettre en œuvre une simulation et avons déjà commencé à profiter de cette expérience pour évaluer des politiques de remplissage de sac au laboratoire de théorie de la complexité. Nous sommes également convaincus que la nécessité de la simulation se présentera encore à de nombreuses reprises dans nos études et nos vies professionnelles.

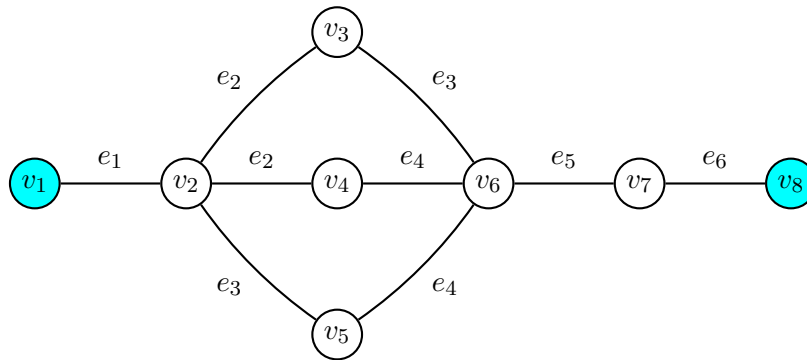
Références

- [1] Éric Thiémarc. *Transparents du cours « Modélisation et simulation discrètes »*. EIVD, 2004–2005.
- [2] Gnuplot, <http://gnuplot.sf.net/>.

6 Annexes

6.1 Les jeux de données

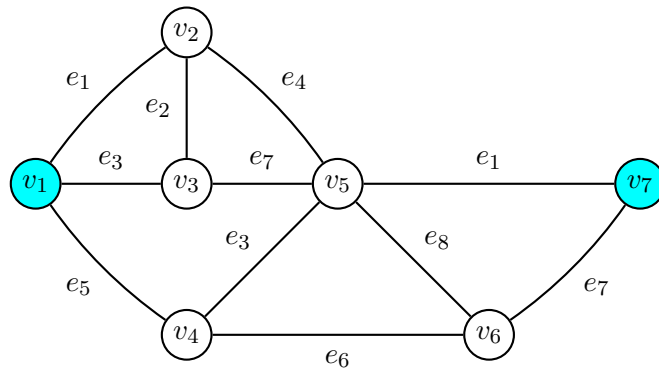
6.1.1 Le jeux de données **data1**



Composante	Loi triangulaire			Espérance
	α	γ	β	
e_1	0.2	2.0	3.0	1.73
e_2	0.5	2.0	3.5	2.00
e_3	0.5	2.0	3.5	2.00
e_4	0.5	2.0	3.5	2.00
e_5	0.8	3.2	5.0	3.00
e_6	0.4	2.5	4.0	2.30

FIG. 16 – data1 — graphe du réseau et lois des probabilités.

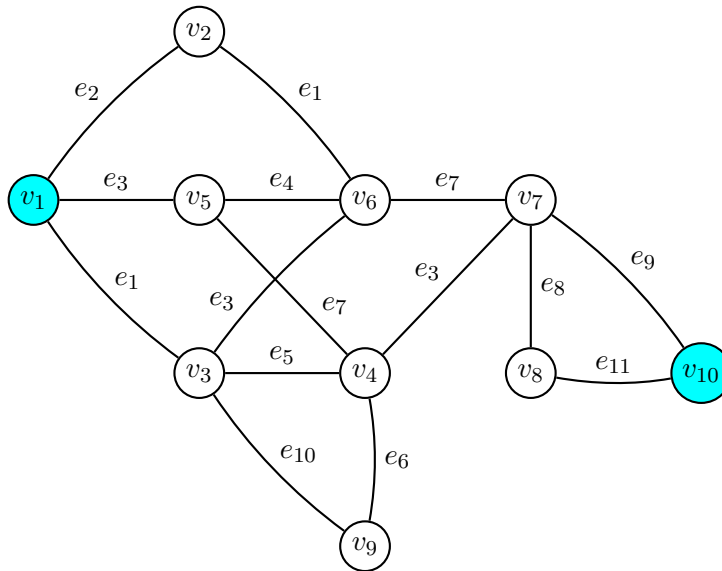
6.1.2 Le jeux de données **data2**



Composante	Loi triangulaire			Espérance
	α	γ	β	
e_1	5.5	7.0	8.0	6.83
e_2	3.5	4.0	4.5	4.00
e_3	9.5	12.0	20.5	14.00
e_4	2.5	4.0	4.5	3.67
e_5	5.8	5.9	8.0	6.56
e_6	10.5	12.5	14.0	12.33
e_7	3.0	3.5	5.0	3.83
e_8	5.0	6.0	7.0	6.00

FIG. 17 – data2 — graphe du réseau et lois des probabilités.

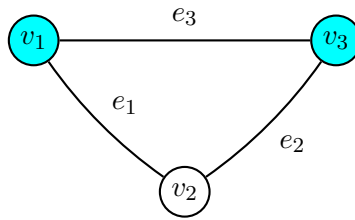
6.1.3 Le jeu de données data3



Composante	Loi triangulaire			Espérance
	α	γ	β	
e_1	44.5	47.0	50.0	47.16
e_2	75.0	76.0	97.0	82.67
e_3	24.5	27.3	32.5	28.10
e_4	62.5	64.0	64.5	63.67
e_5	45.8	57.2	68.0	57.00
e_6	30.5	68.5	94.0	64.33
e_7	33.5	67.0	85.0	61.83
e_8	86.0	87.0	88.0	87.00
e_9	75.0	76.0	97.0	82.67
e_{10}	240.0	250.0	260.0	250.00
e_{11}	240.0	250.0	260.0	250.00

FIG. 18 – data3 — graphe du réseau et lois des probabilités.

6.1.4 Le jeux de données **data4**



Composante	Loi triangulaire			Espérance
	α	γ	β	
e_1	8.0	10.0	12.0	10.0
e_2	12.0	14.0	16.0	14.0
e_3	25.0	27.0	29.0	27.0

FIG. 19 – data4 — graphe du réseau et lois des probabilités.

6.2 Tableaux

durée d'une réparation	espérance	durée d'une réparation	espérance	durée d'une réparation	espérance	durée d'une réparation	espérance
0.01	1.5318	0.26	1.4778	0.51	1.4545	0.76	1.4375
0.02	1.5311	0.27	1.4749	0.52	1.4523	0.77	1.4370
0.03	1.5282	0.28	1.4754	0.53	1.4513	0.78	1.4393
0.04	1.5220	0.29	1.4747	0.54	1.4520	0.79	1.4390
0.05	1.5172	0.30	1.4756	0.55	1.4507	0.80	1.4381
0.06	1.5129	0.31	1.4742	0.56	1.4501	0.81	1.4367
0.07	1.5100	0.32	1.4747	0.57	1.4491	0.82	1.4365
0.08	1.5085	0.33	1.4712	0.58	1.4484	0.83	1.4364
0.09	1.5055	0.34	1.4714	0.59	1.4477	0.84	1.4364
0.10	1.5060	0.35	1.4705	0.60	1.4465	0.85	1.4380
0.11	1.5008	0.36	1.4663	0.61	1.4442	0.86	1.4377
0.12	1.4969	0.37	1.4648	0.62	1.4445	0.87	1.4375
0.13	1.4948	0.38	1.4658	0.63	1.4438	0.88	1.4388
0.14	1.4931	0.39	1.4631	0.64	1.4442	0.89	1.4361
0.15	1.4940	0.40	1.4614	0.65	1.4466	0.90	1.4338
0.16	1.4925	0.41	1.4626	0.66	1.4466	0.91	1.4341
0.17	1.4891	0.42	1.4625	0.67	1.4471	0.92	1.4347
0.18	1.4882	0.43	1.4621	0.68	1.4459	0.93	1.4373
0.19	1.4902	0.44	1.4631	0.69	1.4447	0.94	1.4375
0.20	1.4889	0.45	1.4586	0.70	1.4435	0.95	1.4372
0.21	1.4863	0.46	1.4568	0.71	1.4433	0.96	1.4359
0.22	1.4841	0.47	1.4581	0.72	1.4426	0.97	1.4367
0.23	1.4823	0.48	1.4583	0.73	1.4410	0.98	1.4380
0.24	1.4785	0.49	1.4573	0.74	1.4416	0.99	1.4369
0.25	1.4796	0.50	1.4543	0.75	1.4384	1.00	1.4361

TAB. 2 – data1, politique 2, espérance de vie du système avec une demi-largeur de 0.01 unités de temps.

durée d'une réparation	demi- largeur	espérance	durée d'une réparation	demi- largeur	espérance
0.1	1.00	123.3756	2.6	0.01	7.4207
0.2	1.00	57.2556	2.7	0.01	7.3069
0.3	1.00	36.0244	2.8	0.01	7.1855
0.4	1.00	25.1144	2.9	0.01	7.0644
0.5	0.10	19.6776	3.0	0.01	6.9517
0.6	0.10	16.1523	3.1	0.01	6.8531
0.7	0.10	14.1414	3.2	0.01	6.7606
0.8	0.10	12.9677	3.3	0.01	6.6773
0.9	0.10	11.8540	3.4	0.01	6.6042
1.0	0.01	10.8517	3.5	0.01	6.5472
1.1	0.01	9.8866	3.6	0.01	6.4973
1.2	0.01	9.0317	3.7	0.01	6.4685
1.3	0.01	8.5706	3.8	0.01	6.4309
1.4	0.01	8.3614	3.9	0.01	6.4090
1.5	0.01	8.2706	4.0	0.01	6.4067
1.6	0.01	8.2088	4.1	0.01	6.3953
1.7	0.01	8.1485	4.2	0.01	6.3855
1.8	0.01	8.0761	4.3	0.01	6.3851
1.9	0.01	8.0069	4.4	0.01	6.3868
2.0	0.01	7.9240	4.5	0.01	6.3814
2.1	0.01	7.8489	4.6	0.01	6.3814
2.2	0.01	7.7739	4.7	0.01	6.3849
2.3	0.01	7.7013	4.8	0.01	6.3841
2.4	0.01	7.6242	4.9	0.01	6.3841
2.5	0.01	7.5281	5.0	0.01	6.3841

TAB. 3 – data2, politique 2, espérance de vie du système.

durée d'une réparation	demi- largeur	espérance	durée d'une réparation	demi- largeur	espérance
0.5	5.00	653.7049	15.5	0.10	65.6689
1.0	5.00	327.1473	16.0	0.10	64.8100
1.5	5.00	224.4076	16.5	0.10	63.7464
2.0	1.00	177.3523	17.0	0.10	62.4201
2.5	1.00	149.4469	17.5	0.10	60.9560
3.0	1.00	129.5955	18.0	0.10	59.2948
3.5	1.00	113.9244	18.5	0.10	57.6597
4.0	1.00	100.3460	19.0	0.10	55.9299
4.5	1.00	88.4542	19.5	0.10	54.0946
5.0	1.00	82.3160	20.0	0.01	52.4259
5.5	1.00	77.3009	20.5	0.01	50.8849
6.0	1.00	75.9012	21.0	0.01	49.5644
6.5	1.00	73.2580	21.5	0.01	48.5105
7.0	1.00	72.6221	22.0	0.01	47.7483
7.5	1.00	71.4439	22.5	0.01	47.2487
8.0	1.00	70.0586	23.0	0.01	46.9370
8.5	1.00	69.0922	23.5	0.01	46.7802
9.0	1.00	68.8373	24.0	0.01	46.7049
9.5	1.00	69.2734	24.5	0.01	46.6739
10.0	0.10	68.4906	25.0	0.01	46.6580
10.5	0.10	68.2241	25.5	0.01	46.6580
11.0	0.10	67.9736	26.0	0.01	46.6580
11.5	0.10	67.7953	26.5	0.01	46.6580
12.0	0.10	67.6668	27.0	0.01	46.6580
12.5	0.10	67.4407	27.5	0.01	46.6580
13.0	0.10	67.3155	28.0	0.01	46.6580
13.5	0.10	67.1469	28.5	0.01	46.6580
14.0	0.10	66.8902	29.0	0.01	46.6580
14.5	0.10	66.7255	29.5	0.01	46.6580
15.0	0.10	66.3502	30.0	0.01	46.6580

TAB. 4 – data3, politique 2, espérance de vie du système.

durée d'une réparation	politique 2		politique 3		gain obtenu ²
	demi-largeur ¹	espérance	demi-largeur ¹	espérance	
0.1	0.61450	122.90	1.73995	347.99	183%
0.2	0.28878	57.755	0.51790	103.58	79%
0.3	0.18187	36.373	0.25821	51.641	42%
0.4	0.12812	25.624	0.16145	32.290	26%
0.5	0.09838	19.675	0.11226	22.451	14%
0.6	0.08059	16.117	0.09140	18.279	13%
0.7	0.07054	14.107	0.07897	15.793	12%
0.8	0.06495	12.990	0.07168	14.336	10%
0.9	0.05926	11.852	0.06752	13.504	14%
1.0	0.05437	10.873	0.06690	13.380	23%
1.1	0.04953	9.9064	0.06694	13.388	35%
1.2	0.04500	9.0008	0.06078	12.156	35%
1.3	0.04270	8.5403	0.05465	10.930	28%
1.4	0.04183	8.3652	0.05096	10.192	22%
1.5	0.04129	8.2586	0.04857	9.7136	18%
1.6	0.04098	8.1960	0.04658	9.3165	14%
1.7	0.04080	8.1591	0.04515	9.0309	11%
1.8	0.04039	8.0777	0.04348	8.6952	8%
1.9	0.03990	7.9791	0.04195	8.3895	5%
2.0	0.03956	7.9122	0.04069	8.1389	3%
2.1	0.03918	7.8366	0.03967	7.9341	1%
2.2	0.03889	7.7775	0.03918	7.8365	1%
2.3	0.03860	7.7201	0.03871	7.7428	0%
2.4	0.03802	7.6035	0.03807	7.6139	0%
2.5	0.03756	7.5117	0.03756	7.5126	0%
2.6	0.03719	7.4389	0.03719	7.4388	0%
2.7	0.03657	7.3139	0.03657	7.3139	0%
2.8	0.03594	7.1879	0.03594	7.1879	0%
2.9	0.03529	7.0585	0.03529	7.0585	0%
3.0	0.03483	6.9659	0.03483	6.9659	0%
3.1	0.03427	6.8536	0.03427	6.8548	0%
3.2	0.03376	6.7522	0.03376	6.7527	0%
3.3	0.03341	6.6824	0.03341	6.6819	0%
3.4	0.03301	6.6021	0.03301	6.6022	0%
3.5	0.03268	6.5359	0.03268	6.5358	0%
3.6	0.03243	6.4869	0.03244	6.4880	0%
3.7	0.03225	6.4502	0.03225	6.4502	0%
3.8	0.03209	6.4170	0.03208	6.4165	0%
3.9	0.03205	6.4103	0.03205	6.4103	0%
4.0	0.03202	6.4046	0.03202	6.4048	0%

TAB. 5 – data2, mesures pour les comparaisons des politiques 2 et 3.

¹Elle correspond à 0.5% de l'espérance.

²Il s'agit du gain en pour-cent de l'espérance de la politique 3 sur la politique 2.

durée d'une réparation	politique 2		politique 3		gain obtenu ²
	demi-largeur ¹	espérance	demi-largeur ¹	espérance	
0.75	2.17520	435.04	3.87975	775.95	78%
1.00	1.64430	328.86	2.74335	548.67	67%
1.25	1.33290	266.58	2.10670	421.34	58%
1.50	1.12910	225.82	1.68610	337.22	49%
1.75	0.98650	197.30	1.40465	280.93	42%
2.00	0.88725	177.45	1.18640	237.28	34%
2.25	0.81145	162.29	1.01095	202.19	25%
2.50	0.74960	149.92	0.87310	174.62	16%
2.75	0.69530	139.06	0.76040	152.08	9%
3.00	0.64870	129.74	0.67640	135.28	4%
3.25	0.60850	121.70	0.61670	123.34	1%
3.50	0.56890	113.78	0.57240	114.48	1%
3.75	0.53245	106.49	0.53800	107.60	1%
4.00	0.50025	100.05	0.50605	101.21	1%
4.25	0.47099	94.198	0.48357	96.713	3%
4.50	0.44717	89.434	0.46255	92.509	3%
4.75	0.42625	85.250	0.44500	89.000	4%
5.00	0.40973	81.945	0.42897	85.794	5%
5.25	0.40030	80.059	0.41616	83.231	4%
5.50	0.39075	78.149	0.40619	81.238	4%
5.75	0.38254	76.508	0.39541	79.082	3%
6.00	0.37624	75.247	0.38835	77.669	3%
6.25	0.37028	74.055	0.38366	76.732	4%
6.50	0.36741	73.481	0.37915	75.830	3%
6.75	0.36458	72.915	0.37433	74.866	3%
7.00	0.36177	72.354	0.36729	73.457	2%
7.25	0.35738	71.476	0.36559	73.118	2%
7.50	0.35661	71.322	0.36404	72.807	2%
7.75	0.35497	70.993	0.36152	72.304	2%
8.00	0.35145	70.289	0.35781	71.562	2%
8.25	0.34934	69.867	0.35536	71.071	2%
8.50	0.34828	69.655	0.35243	70.485	1%
8.75	0.34822	69.644	0.35172	70.343	1%
9.00	0.34678	69.355	0.35019	70.038	1%
9.25	0.34586	69.172	0.34942	69.883	1%
9.50	0.34409	68.817	0.34848	69.696	1%
9.75	0.34347	68.693	0.34574	69.147	1%
10.00	0.34348	68.696	0.34542	69.084	1%
10.25	0.34234	68.467	0.34448	68.896	1%
10.50	0.34236	68.472	0.34319	68.638	0%
10.75	0.34207	68.414	0.34304	68.607	0%

TAB. 6 – data3, mesures pour les comparaisons des politiques 2 et 3 — première partie.

¹Elle correspond à 0.5% de l'espérance.

²Il s'agit du gain en pour-cent de l'espérance de la politique 3 sur la politique 2.

durée d'une réparation	politique 2		politique 3		gain obtenu ²
	demi-largeur ¹	espérance	demi-largeur ¹	espérance	
11.00	0.34068	68.135	0.34202	68.403	0%
11.25	0.34044	68.088	0.34151	68.301	0%
11.50	0.33938	67.876	0.34017	68.033	0%
11.75	0.33932	67.863	0.33949	67.898	0%
12.00	0.33864	67.727	0.33911	67.821	0%
12.25	0.33705	67.409	0.33728	67.455	0%
12.50	0.33704	67.407	0.33732	67.464	0%
12.75	0.33662	67.324	0.33678	67.355	0%
13.00	0.33621	67.241	0.33663	67.326	0%
13.25	0.33656	67.311	0.33667	67.334	0%
13.50	0.33610	67.219	0.33612	67.223	0%
13.75	0.33564	67.127	0.33574	67.148	0%
14.00	0.33479	66.958	0.33479	66.957	0%
14.25	0.33503	67.005	0.33504	67.007	0%
14.50	0.33327	66.654	0.33327	66.654	0%
14.75	0.33200	66.399	0.33201	66.401	0%
15.00	0.33213	66.425	0.33213	66.425	0%
15.25	0.32956	65.911	0.32956	65.911	0%
15.50	0.32801	65.601	0.32801	65.601	0%
15.75	0.32703	65.405	0.32703	65.405	0%
16.00	0.32409	64.817	0.32409	64.817	0%
16.25	0.32244	64.487	0.32244	64.487	0%
16.50	0.31860	63.720	0.31860	63.720	0%
16.75	0.31451	62.901	0.31451	62.901	0%
17.00	0.31220	62.439	0.31220	62.439	0%
17.25	0.30947	61.893	0.30947	61.893	0%
17.50	0.30493	60.986	0.30493	60.986	0%
17.75	0.30126	60.252	0.30126	60.252	0%
18.00	0.29724	59.447	0.29724	59.447	0%
18.25	0.29318	58.635	0.29318	58.635	0%
18.50	0.28869	57.737	0.28869	57.737	0%
18.75	0.28432	56.864	0.28432	56.864	0%
19.00	0.28109	56.217	0.28109	56.217	0%
19.25	0.27556	55.111	0.27556	55.111	0%
19.50	0.27074	54.148	0.27074	54.148	0%
19.75	0.26573	53.146	0.26573	53.146	0%
20.00	0.26168	52.336	0.26168	52.336	0%

TAB. 7 – data3, mesures pour les comparaisons des politiques 2 et 3 — suite et fin.

¹Elle correspond à 0.5% de l'espérance.

²Il s'agit du gain en pour-cent de l'espérance de la politique 3 sur la politique 2.

Listings

1	echeancier.h	29
2	generateurs.c	31
3	reseau.c	33
4	simulation.c	42
5	statistiques.c	46

Listing 1 – echeancier.h

```
/*
/*
/* Fichier : echeancier.h
/*
5 /* Auteur : E. Thiémond, département E+I, EIVD
/* Version : Octobre 2004
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Définition de la structure des événements qui seront
10 /* stockés dans l'échéancier et manipulés par les
/* politiques du module simulation. Ce module contient
/* également les quatre routines de base de gestion
/* d'un échéancier : créer, vider, nouvel_evenement et
/* retire_prochain_evenement.
15 /*
/*
/*
/* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 10.11.2004
20 /* Modifs : ajout de l'événement réveil pour la politique 3
/*
/*
/* Consigne : A priori, ne rien changer à ce fichier...
25 /*
/*
/* ... dans tous les cas, la simulation des politiques
/* 1 et 2 ne nécessite aucun changement.
/*
/* Pour la politique 3, des ajouts (nouveau t_evnmnt ou
30 /* ou données supplémentaires dans echeancier_type_eve)
/* sont envisageables, quoique pas nécessaires...
/*
/*
/*
35 #ifndef ECHEANCIER_H
#define ECHEANCIER_H

/*
/* Types d'événements : une composante tombe en panne, la réparation
40 /* d'une composante s'achève, fin de simulation (panne de système).
/*
enum t_evnmnt {panne_composante, fin_reparation, fin_simul, reveil};

/*
45 /* Structure des événements : un type, une date, le sommet concerné.
/*
typedef struct
{
enum t_evnmnt type; /* Type de l'événement */
50 double date_evenement; /* Date de l'événement */
int composante; /* Composante (1 à nb_comp) concernée */
} echeancier_type_eve;

/*
55 /* Crée un échéancier vide
/*
void echeancier_cree(void);
```

```

60  /*****
    /* Vider l'échéancier */
    /*****
    void echeancier_vider(void);

65  /*****
    /* Insère un nouvel événement dans l'échéancier */
    /* Input : un événement */
    /*****
    void echeancier_nouvel_evenement(echeancier_type_eve evenement);

70  /*****
    /* Retire le prochain événement de l'échéancier */
    /* Output : le prochain événement */
    /*****
    echeancier_type_eve echeancier_retire_prochain_evenement(void);

75  #endif

```

Listing 2 – generateurs.c

```

/*****
/*
/* Fichier : generateurs.c
/*
5 /* Auteur : E. Thiémarc, département E+I, EIVD
/* Version : Octobre 2004
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Bibliothèque de générateurs de variables aléatoires
10 /*
/*****
/*
/* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 10.11.2004
15 /* Modifs : implémentation de:
/* - generateurs_triangulaire()
/*
/*****
/*
20 /* Consigne : compléter la fonction generateurs_triangulaire(...)
/*
/* La fonction generateurs_exponentielle(...) vous est
/* généreusement fournie en guise d'exemple.
/*
25 /*****

#include <stdlib.h>
#include <stdio.h>
30 #include <math.h>
#include "generateurs.h"

/*****
35 /* Affiche un message d'erreur car paramètres fournis non admissibles */
/*****
static void erreur_parametres(char* nom)
{
    fprintf(stderr, "Erreur : les paramètres fournis au générateur %s ne sont pas
    admissibles\n", nom);
40    exit(EXIT_FAILURE);
}

/*****
45 /* Génération d'une réalisation d'une loi exponentielle de paramètre
/* lambda à l'aide du générateur pseudo-aléatoire GenPseudo.
/*****
double generateurs_exponentielle(double (*GenPseudo)(void), double lambda)
{
50    if (lambda<=0)
        erreur_parametres("d'exponentielles");
    return -log(GenPseudo())/lambda;
}

55 /*****
/* Génération d'une réalisation d'une loi triangulaire de paramètres
/* (a,c,b) à l'aide du générateur pseudo-aléatoire GenPseudo et de la
/* méthode des fonctions inverses.
60 /*****
double generateurs_triangulaire(double (*GenPseudo)(void), double a, double c,
    double b)
{

65    double u = GenPseudo();

```

```
    if (u <= (c-a)/(b-a)) {
        return a + sqrt(u * (b-a) * (c-a));
    } else {
70     return b - sqrt((1-u) * (b-a) * (b-c));
    }
}
```

Listing 3 – reseau.c

```

/*****
/*
/* Fichier : reseau.c
/*
5 /* Auteur : E. Thiémarc, département E+I, EIVD
/* Version : Octobre 2004
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Module servant à stocker la structure du système, à
10 /* modifier l'état des composantes, à vérifier si le
/* système fonctionne et à fournir des informations aux
/* politiques_2_3 de maintenance du module simulation.
/*
/*****
15 /*
/* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 10.11.2004
/* Modifs : implémentation de:
/* - reseau_plus_ancienne_comp_en_panne
20 /* - reseau_analyse_3
/* - reseau_conseil_3
/*
/*****
25 /* Consigne : compléter ce fichier (fonctions et variables),
/* notamment, pour la politique numéro
/* 2, compléter reseau_plus_ancienne_comp_en_panne,
/* 3, compléter reseau_analyse_3 et reseau_conseil_3.
/*
30 /*****

#include <stdlib.h>
#include <stdio.h>
35 #include <float.h>
#include "reseau.h"
#include "MT19937.h"
#include "generateurs.h"

40
typedef struct {
    int comp; /* index de la composante */
    double prob; /* probabilité qu'elle tombe en panne à un temps t */
} t_tab_prob;
45
static int nb_comp; /* Nombre de composantes dans le système. */
/* Les composantes sont numérotées de 1 à nb_comp. */
static double *alpha; /* Premiers paramètres des lois triangulaires pour
/* la durée de fonctionnement des composantes. */
50 static double *gamma; /* Deuxièmes paramètres des lois triangulaires... */
static double *beta; /* Troisièmes paramètres des lois triangulaires... */
static int *fonctionne; /* Tableau indiquant, pour chaque composante, si
/* elle fonctionne ou non : 1 = OK , 0 = panne. */

55 static int nb_sommets; /* Nombre de sommets dans le graphe correspondant
/* Les sommets sont numérotés de 1 à nb_sommets. */
static int **graphe_de_fiabilite; /* Matrice sommet-sommet précisant le
/* numéro de l'éventuelle composante les reliant. */
static int *exploration; /* Tableau auxiliaire pour l'exploration du graphe*/
60 /* afin de détecter si le système fonctionne. */

static double *temps_dernier_ev; /* stocke la date du dernier
/* événement pour chaque composante. */

65 static double duree_reparation; /* duree d'une réparation */

static t_tab_prob *tab_prob; /* tableau contenant l'index des

```

```

70      /* composantes et la probabilité          */
      /* qu'elles tombes en panne à un temps t */
      /* Etant donné que ce tableau est        */
      /* destiné à être trier, il est neces-   */
      /* saire de connaitre l'index de la     */
      /* composante.                           */

75  static double *esperance;      /* tableau des espérances des composantes */

      static int reveil;          /* délai maximum avant de rappeler la fonction */
      /* reseau_conseil_3. cette valeur est un rapport */
      /* de la duree d'une réparation          */
80      /* délai de rappel = -1 * duree_reparation/reveil */

      static double horloge;     /* temps courant */

      int reseau_inchange;       /* flag indiquant que l'état du réseau est */
85      /* inchangé */

      int nbre_comp_en_panne = 0; /* nbre de composantes actuellement en panne */

      /******
90  /* Affiche un message d'erreur relatif à la lecture du fichier de données */
      /* Input : Nom du fichier, message d'erreur */
      /******
      static void message_erreur(char *nom_fichier, char *message)
      {
95      fprintf(stderr,"Erreur : le fichier \"%s\" %s\n", nom_fichier, message);
      exit(EXIT_FAILURE);
      }

100  /******
      /* Affiche un message d'erreur au niveau allocation dynamique de mémoire */
      /******
      static void erreur_allocation_memoire(void)
      {
105      fprintf(stderr,"Problème d'allocation mémoire pour les données\n");
      exit(EXIT_FAILURE);
      }

110  /******
      /* Lecture et vérification du fichier de données définissant le système. */
      /* Stockage dans les zones mémoire allouées. Initialise le générateur */
      /* pseudo-aléatoire et retourne le nombre de composantes du système. */
      /******
115  int reseau_lecture_donnees(char *nom_fichier)
      {
      int i,j;
      FILE *fichier_donnees;      /* Fichier à lire contenant les données */

120      unsigned long init[4]={0x123, 0x345, 0x567, 0x789}, length=4; /* Germe */
      init_by_array(init,length); /* Initialisation gén pseudo-aléatoire */

      printf("Importation des données à partir du fichier \"%s\"\n",nom_fichier);
      fichier_donnees=fopen(nom_fichier,"r");
125      if (fichier_donnees==NULL)
          message_erreur(nom_fichier,"n'existe pas");

      if ((fscanf(fichier_donnees,"%d",&nb_comp)==EOF) || (nb_comp<1))
          message_erreur(nom_fichier,"n'a pas le bon format. Il doit commencer par le
          nombre de composantes k>=1 dans le système.");
130      if ((fscanf(fichier_donnees,"%d",&nb_sommets)==EOF) || (nb_sommets<2))
          message_erreur(nom_fichier,"n'a pas le bon format. Le second paramètre doit
          être le nombre de sommets n>=2 dans le graphe de fiabilité.");

      alpha = (double *) calloc(nb_comp+1,sizeof(double));

```

```

135  gamma = (double *) calloc(nb_comp+1, sizeof(double));
      beta  = (double *) calloc(nb_comp+1, sizeof(double));
      fonctionne = (int *) calloc(nb_comp+1, sizeof(int));
      graphe_de_fiabilite = (int **) calloc(nb_sommets+1, sizeof(int *));
      exploration = (int *) calloc(nb_sommets+1, sizeof(int));
      temps_dernier_ev = (double *) calloc(nb_comp+1, sizeof(double));
140  if ((alpha==NULL) || (gamma==NULL) || (beta==NULL) || (fonctionne==NULL) || (
      graphe_de_fiabilite==NULL) || (exploration==NULL) || (temps_dernier_ev==NULL)
      )
      erreur_allocation_memoire();

      for (i=1; i<=nb_comp; i++)
          if ((fscanf(fichier_donnees, "%lf %lf %lf", &alpha[i], &gamma[i], &beta[i]) == EOF)
              || (alpha[i]<0.0) || (alpha[i]>gamma[i]) || (gamma[i]>beta[i]))
145  message_erreur(nom_fichier, "n'a pas le bon format. Pour chacune des
      composantes, trois paramètres 0<=alpha<=gamma<=beta doivent être spécifiés.")
      ;

      for (i=1; i<=nb_sommets; i++) {
          graphe_de_fiabilite[i] = (int *) calloc(nb_sommets+1, sizeof(int));
          if (graphe_de_fiabilite[i]==NULL)
              erreur_allocation_memoire();
150  for (j=1; j<=i; j++)
              if ((fscanf(fichier_donnees, "%d", &graphe_de_fiabilite[i][j]) == EOF) || (
                  graphe_de_fiabilite[i][j]<0) || (graphe_de_fiabilite[i][j]>nb_comp))
                  message_erreur(nom_fichier, "n'a pas le bon format. Pour chaque
                  sommet i (entre 1 et nb_sommets), on doit spécifier 0 ou un numéro de
                  composante (entre 1 et nb_comp), ceci pour chaque sommet j entre 1 et i.");
      }

      for (i=2; i<=nb_sommets; i++)
155  for (j=1; j<=i; j++)
          /* Pour le même prix on définit le symétrique de la matrice (a_{i,j}) de l'
          énoncé */
          graphe_de_fiabilite[j][i] = graphe_de_fiabilite[i][j];

      fclose(fichier_donnees);
160  return nb_comp;
      }

/*****/
165  /* Génère la durée de fonctionnement d'une composante. */
/*****/
double reseau_duree_fonct_comp(int comp)
{
170  if ((comp<1) || (comp>nb_comp))
      {
          fprintf(stderr, "Erreur: l'index %d ne désigne pas une composante\n", comp);
          exit(EXIT_FAILURE);
      }

      return generateurs_triangulaire(genrand_real3, alpha[comp],
175  gamma[comp], beta[comp]);
}

/*****/
180  /* Nous sommes à une certaine date et une composante tombe en panne. */
/*****/
void reseau_comp_en_panne(int comp, double temps)
{
185  if ((comp<1) || (comp>nb_comp))
      {
          fprintf(stderr, "Erreur: l'index %d ne désigne pas une composante\n", comp);
          exit(EXIT_FAILURE);
      }

190  if (fonctionne[comp]==0)
      {

```

```

        fprintf(stderr, "Erreur: la composante %d est déjà en panne !!!\n",comp);
        exit(EXIT_FAILURE);
195     }

        fonctionne[comp] = 0;

        temps_dernier_ev[comp] = temps;
200     reseau_inchange = 0;

    }

205     /* Une certaine composante est remise en état. */
    /* ***** */
    void reseau_comp_fonctionne(int comp, double temps)
210 {

        if ((comp<1) || (comp>nb_comp))
        {
            fprintf(stderr, "Erreur: l'index %d ne désigne pas une composante\n",comp);
215             exit(EXIT_FAILURE);
        }

        fonctionne[comp] = 1;

220     temps_dernier_ev[comp] = temps;

        reseau_inchange = 0;
    }

225     /* Retourne l'index de la composante qui est depuis le plus longtemps */
    /* en panne ou 0 si elles fonctionnent toutes. */
    /* ***** */
230     int reseau_plus_ancienne_comp_en_panne(void)
    {

        int index = 0;          /* index à retourner, 0 si tout fonctionne */
        double plus_ancien_temps = 0.0; /* temps de la plus ancienne comp */
235         int i;                /* indice de boucle */

        /* parcours le tableau des composantes */
        for (i = 1; i <= nb_comp; i++) {
            if (!fonctionne[i]) { /* une en panne */
240                 /* si c'est la première qu'on trouve ou que son temps est plus
ancien, on la garde */
                if (index == 0 || (temps_dernier_ev[i] < plus_ancien_temps)) {
                    index = i;
                    plus_ancien_temps = temps_dernier_ev[i];
245             }
            }
        }
        return index;
    }
250 }

    /* Retourne 0 si la composante est en panne, 1 si elle fonctionne. */
    /* ***** */
255     int reseau_comp_etat(int comp)
    {
        if ((comp<1) || (comp>nb_comp))
        {
            fprintf(stderr,
260                 "Erreur : l'index %d ne désigne pas une composante\n",comp);

```

```

        exit(EXIT_FAILURE);
    }
    return fonctionne[comp];
}
265

/*****/
/* Explore les sommets atteignables à partir d'un sommet donné. */
/*****/
270 void visite_sommet(int sommet)
{
    int i;

    exploration[sommet] = 1; /* Sommet traité */
275 for (i=1;i<=nb_sommets;i++)
    if (exploration[i]==0) /* Visiter les voisins pas encore atteints */
        if (graphe_de_fiabilite[sommet][i]>0) /* et atteignables par des
                                                arêtes correspondant */
            if (fonctionne[graphe_de_fiabilite[sommet][i]]==1) /* à des composantes
280                 qui fonctionnent */
                visite_sommet(i);
}

285 /*****/
/* Retourne 1 si le réseau est fonctionnel, 0 s'il est en panne. */
/*****/
int reseau_systeme_fonctionne(void)
{
290     int i;

    for (i=1;i<=nb_sommets;i++)
        exploration[i]=0;
    visite_sommet(1);
295
    if (exploration[nb_sommets]==1)
        return 1;
    else
        return 0;
300 }

/*****/
/* Retourne la probabilitié que la composante 'comp' tombe en panne */
/* au temps passé en paramètre. */
/*****/
305 double probabilite_panne(int comp, double temps) {

    double t = temps - temps_dernier_ev[comp];

310     if (t < alpha[comp])
        return 0.0;

    if (t > beta[comp])
        return 1.0;
315
    if (t < gamma[comp])
        return (t-alpha[comp]) * (t-alpha[comp]) /
            ((beta[comp]-alpha[comp]) * (gamma[comp]-alpha[comp]));
    else
320     return 1 - ((beta[comp]-t) * (beta[comp]-t) /
            ((beta[comp]-alpha[comp]) * (beta[comp]-gamma[comp])));
}

/*****/
325 /* Effectue le tri du tableau tab_prob par ordre décroissant des */
/* probabilités. Cet algorithme s'effectue en n**2, ce qui est très */
/* efficace pour des petits réseaux. Si les réseaux venaient à être */
/* grand, il faudrait le réécrire en n*log(n) */

```

```

/*****/
330 void trier_tab_prob() {
    int i, j;          /* indice de boucle */
    int max;          /* probabilité maximum */
    t_tab_prob temp;  /* tampon d'échange */

335     for (i=1; i<=nb_comp-1; i++) {
        max = i;
        for (j=i+1; j<=nb_comp; j++) {
            max = tab_prob[j].prob > tab_prob[max].prob ? j : max;
340         }
        temp = tab_prob[i];
        tab_prob[i] = tab_prob[max];
        tab_prob[max] = temp;
    }
345 }

/*****/
/* Retourne l'index de la composante en panne qui a la plus grande */
/* espérance, ou 0 si elles fonctionnent toutes. */
/*****/
350 int plus_grande_esperance_en_panne(void) {
    int i;
    double esperance_max = 0.0;
    int indice = 0;

355     for (i=1; i<=nb_comp; i++) {
        if (!fonctionne[i] && esperance[i] > esperance_max) {
            esperance_max = esperance[i];
            indice = i;
        }
360     }
    return indice;
}

/*****/
365 /* Retourne la composante à réparer lorsque le réseau est en panne */
/*****/
int composante_a_reparer() {
    int i, j;
    int indice = 0;
370     double esperance_max = 0.0;

    /* On effectue le parcours du tableau des probabilités depuis
       la fin car les composantes réellement en panne sont marqués
       par une probabilité de -1.0, et vu que le tableau est trié
375     par ordre décroissant, on trouve toutes les composantes
       à la fin du tableau */
    for (i=nb_comp; i>0 && tab_prob[i].prob < 0.0; i--) {
        fonctionne[tab_prob[i].comp] = 1; /* on simule sa réparation */
        if (reseau_systeme_fonctionne()) { /* et on teste le reseau */
            /* on a trouvé une pièce qui permet de réparer le réseau :-*/
            /* pour départager les ex aequo, on regarde celle qui à
               la plus grande espérance */
            if (esperance[tab_prob[i].comp] > esperance_max) {
385                 esperance_max = esperance[tab_prob[i].comp];
                indice = tab_prob[i].comp;
            }
        }
        /* ne pas oublier de remettre la composante en panne */
        fonctionne[tab_prob[i].comp] = 0;
390     }

    /* si on a trouvé une composante à réparer, on la retourne */
    if (indice != 0) {
395         return indice;
    }
}

```

```

/* on effectue la même chose, mais en mettant cette fois deux
   composante en service */
400 for (i=nb_comp; i>1 && tab_prob[i].prob < 0.0; i--) {
   fonctionne[tab_prob[i].comp] = 1;

   /* pour la deuxième composante, on peut également simuler
      la réparation d'une pièce qui n'est pas réellement en panne */
405   for (j=i-1; j>0; j--) {
      if (!fonctionne[tab_prob[j].comp]) {
         fonctionne[tab_prob[j].comp] = 1;
         if (reseau_systeme_fonctionne()) {
            /* on a trouvé un couple de composantes qui permet de
               réparer le réseau */
410            /* on le garde la première composante si son espérance
               est plus grande que les autres */
            if (esperance[tab_prob[i].comp]>esperance_max) {
               esperance_max = esperance[tab_prob[i].comp];
               indice = tab_prob[i].comp;
415            }

            /* on garde la seconde composante si c'est une composante
               qui est réellement en panne et que son espérance
               est plus grande que les autres */
420            if (tab_prob[j].prob < 0.0
                && esperance[tab_prob[j].comp]>esperance_max) {
               esperance_max = esperance[tab_prob[j].comp];
               indice = tab_prob[j].comp;
425            }
            }
            /* on remet la composante en panne */
            fonctionne[tab_prob[j].comp] = 0;
         }
      }
430   /* et celle-ci aussi */
   fonctionne[tab_prob[i].comp] = 0;
}
return indice;
}
435
/*****
/* Cette fonction met en panne l'une après l'autres les composantes
/* ayant une probabilité de tomber en panne et détermine si le
/* réseau continue de fonctionner. La fonction retourne 0 si les
440 /* réseau continue de fonctionner ou la composante à réparer.
/*****
int reseau_simulation_panne(void) {

   int i;           /* indice de boucle */
445   int panne_trouvee = 0; /* flag indiquant que le réseau est en panne */
   int indice = 0;  /* indice de la composante à réparer */

   /* on met en panne les composantes ayant un probabilité > 0.001
      dans l'ordre décroissant de leur probabilité */
450   for (i=1; i<=nb_comp && tab_prob[i].prob > 0.001 && !panne_trouvee; i++) {
      fonctionne[tab_prob[i].comp] = 0;
      /* test si le réseau est en panne */
      panne_trouvee = !reseau_systeme_fonctionne();
455   }

   /* On détermine la composante qu'il faut réparer afin que le
      réseau refonctionne */
   indice = composante_a_reparer();

460   /* On remet le réseau dans l'état dans lequel il se trouvait
      à l'entrée de la fonction */
   for (i=1; i<=nb_comp && tab_prob[i].prob > 0.001; i++) {
      fonctionne[tab_prob[i].comp] = 1;
   }
}

```

```

465     /* si aucune panne de réseau n'a pu être simulée, on retourne 0 */
    if (!panne_trouvee) {
        return 0;
    }
470     /* dans le cas contraire, on retourne celle devant être réparée
        afin que le réseau refonctionne. Si on en connaît aucune,
        on retourne celle ayant la plus grande espérance */
    return indice == 0 ? plus_grande_espérance_en_panne() : indice;
475 }

/*****
/* Cette fonction est destinée à retourner un conseil à la politique */
480 /* 3, à savoir le numéro de la composante à réparer. */
/* Cette fonction peut accéder à l'état actuel des composantes, aux */
/* paramètres alpha[], beta[], gamma[] associés aux composantes, à la */
/* structure du graphe de fiabilité (dans graphe_de_fiabilite[][]), */
/* ainsi qu'aux résultats de l'analyse effectuée lors de l'appel à la */
485 /* fonction reseau_analyse(void). Vous pouvez bien sûr aussi tenir */
/* compte de ce qui a eu lieu dans le passé (date de la dernière */
/* réparation de chaque composante... pour autant qu'elle soit */
/* stockée quelque part), mais il est par contre interdit de tricher: */
/* cette fonction n'a pas accès au futur, comme par exemple aux dates */
490 /* des prochaines pannes planifiées dans l'échéancier... */
/*****/
int reseau_conseil_3(double temps)
{
    int i, j; /* indices de boucles */
495    int composante_a_reparer = 0; /* no de la composante à réparer */
    int i_min = 1; /* premier indice de boucle */

    /* Afin d'éviter de re-calculer des probabilités de pannes
        qui ont déjà été effectuées, on décale le premier indice
500 de la boucle si l'état du réseau n'a pas changé depuis
        le dernier appel de la fonction et que le délai entre
        les 2 appels de la fonction <= à la durée de réparation */
    if (reseau_inchange && horloge + duree_reparation >= temps) {
        i_min = 2 + nbre_comp_en_panne;
505    } else { /* l'état du réseau a changé (ou est trop ancien) */
        reseau_inchange = 1; /* mise à jour du flag */

        /* calcul du nombre de composante en panne */
        nbre_comp_en_panne = 0;
510        for (i=1; i<=nb_comp;i++) {
            if (!fonctionne[i]) {
                nbre_comp_en_panne++;
            }
        }
515    }

    /* Si aucune composante n'est en panne, on retourne 0 */
    if (nbre_comp_en_panne == 0)
        return 0;

520    /* sauvegarde du temps */
    horloge = temps;

    /* On va maintenant regarder dans le futur en avançant dans le temps par
525 unité de durée de réparation jusqu'à ce que l'on trouve une composante
        à réparer ou qu'aucune réparation ne soit nécessaire dans un délai
        permettant théoriquement de réparer toutes les composantes en panne */
    for (i=i_min; i<=2 + nbre_comp_en_panne && composante_a_reparer == 0; i++) {

530        /* Calcul de la probabilité de panne de toutes les composantes au
            temps i*durée d'une réparation */
        for (j=1; j<=nb_comp; j++) {

```

```

        tab_prob[j].comp = j;
        if (fonctionne[j]) {
535         tab_prob[j].prob =
            probabilite_panne(tab_prob[j].comp, temps + i*duree_reparation);
        } else { /* la composante est déjà en panne, on l'indique par -1 */
            tab_prob[j].prob = -1.0;
        }
540     }

    /* On trie le tableau des probabilités de la plus probable à la moins */
    trier_tab_prob();

545     /* On simule les pannes possibles, afin de définir la composante
        à réparer */
    composante_a_reparer = reseau_simulation_panne();
}

550     /* si on ne trouve aucune composante à réparer, on retourne une valeur
        négative afin de demander le réveil. */
    return composante_a_reparer == 0 ? reveil : composante_a_reparer;
}

555     /*****
    /* Retourne la moyenne de l'espérance de toutes les composantes. */
    /*****
560     double moyenne_esperance(void){

        int i;
        double somme_esperance = 0.0;

        for (i=1; i<=nb_comp; i++) {
565             somme_esperance += (alpha[i] + gamma[i] + beta[i]) / 3.0;
        }
        return somme_esperance / (double)nb_comp;
    }

570     /*****
    /* Analyse de la structure du système en vue de la politique 3. */
    /* N'est appelée qu'une fois, juste après la lecture des données, */
    /* en lui passant en paramètre la durée d'une réparation. */
    /*****
575     void reseau_analyse_3(double duree)
    {
        int i; /* indice de boucle */

        /* initialisation des variables */
580         tab_prob = (t_tab_prob *) calloc(nb_comp+1, sizeof(t_tab_prob));
        esperance = (double *) calloc(nb_comp+1, sizeof(double));

        if ((tab_prob==NULL) || (esperance==NULL)) {
585             erreur_allocation_memoire();
        }

        /* sauvegarde de la durée de réparation */
        duree_reparation = duree;
590

        /* valeur de retour de reseau_conseil_3 lorsqu'il est préférable
            d'attendre avant de réparer une composante en panne */
        reveil = - (duree_reparation * 100 / moyenne_esperance() + 1);

595         /* initialisation du tableau des espérances */
        for (i=1; i<=nb_comp; i++) {
            esperance[i] = (alpha[i]+gamma[i]+beta[i])/3.0;
        }
    }
}

```

Listing 4 – simulation.c

```

/*****
/*
/* Fichier : simulation.c
/*
5 /* Auteur : E. Thiémarc, département E+I, EIVD
/* Version : Octobre 2004
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Module permettant de simuler une réalisation de la
10 /* durée de vie du système lorsqu'une des 3 politiques
/* de maintenance préconisées est appliquée.
/* Retourne la durée de vie du système obtenue.
/*
/*****
15 /*
/* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 01.11.04
/* modifs : implémentation de:
/* - politique_2()
20 /* - politique_3()
/* - traiter_evenement()
/*
/*****
25 /* Consigne : Compléter la fonctions traiter_evenement(), ainsi
/* que les politiques politique_2() et politique_3().
/* Au besoin introduire les éventuelles fonctions et
/* variables auxiliaires supplémentaires nécessaires.
/*
30 /*****

#include <stdlib.h>
#include <stdio.h>
35 #include "simulation.h"
#include "echeancier.h"
#include "reseau.h"

static int nb_comp; /* Nombre de composantes dans le système
*/
40 static void (*politique_reparation)(void); /* Pointeur sur politique à appliquer
*/
static double duree_d_une_reparation;
static double horloge;

static int reparateur_libre;
45

/*****
/* Planifie la prochaine panne (durée triangulairement distribuée
/* à partir de la date courante) d'une composante donnée et envoie à
50 /* l'échéancier l'événement correspondant.
/*****
static void planifie_panne_comp(int comp)
{
echeancier_type_eve evenement;
55
if (reseau_comp_etat(comp)==0)
{
fprintf(stderr,"Erreur : la composante %d est déjà en panne... pas 2 fois\n
",comp);
exit(EXIT_FAILURE);
60 }
evenement.type = panne_composante;
evenement.date_evenement = horloge+reseau_duree_fonct_comp(comp);
evenement.composante = comp;
echeancier_nouvel_evenement(evenement);

```

```

65 }

/*****/
/* Selon la politique_1, on n'effectue aucune réparation. */
70 /*****/
void politique_1(void)
{
    /* Ne rien faire */
}
75

/*****/
/* Planification de la prochaine réparation selon la politique "on */
/* commence par le relais qui est depuis le plus de temps en panne". */
80 /*****/
void politique_2(void)
{
    int comp_a_reparer; /* composante à réparer */
    echeancier_type_eve evt; /* evenement dans l'échéancier */
85
    /* si le réparateur n'est pas libre, on ne fait rien */
    if (!reparateur_libre) {
        return;
    }
90
    /* si aucune composante n'est en panne, on quitte la fonction */
    if ((comp_a_reparer = reseau_plus_ancienne_comp_en_panne()) == 0) {
        return;
    }
95
    /* on ajoute la fin de la réparation de la composante
       dans l'échéancier*/
    evt.type = fin_reparation;
    evt.date_evenement = horloge + duree_d_une_reparation;
100 evt.composante = comp_a_reparer;
    echeancier_nouvel_evenement(evt);

    reparateur_libre = 0; /* le reparateur est occupé */
}
105

/*****/
/* Planification du prochain événement (réparation ou réveil) selon la */
/* politique 3. */
/*****/
110 void politique_3(void)
{
    int comp_a_reparer; /* composante à réparer */
    echeancier_type_eve evt; /* événement dans l'échéancier */

115 /* si le réparateur n'est pas libre, on ne fait rien */
    if (!reparateur_libre) {
        return;
    }

120 /* demande de conseil sur la pièce à réparer */
    comp_a_reparer = reseau_conseil_3(horloge);

    /* s'il n'y a aucune composante en panne, on ne fait rien */
    if (comp_a_reparer == 0) {
125 return;
    }

    /* une valeur négative indique qu'il ne faut rien faire pour
       l'instant. */
130 if (comp_a_reparer < 0) {
    evt.type = reveil;
}

```

```

    evt.date_evenement = horloge + (duree_d_une_reparation / (-1*comp_a_reparer
));
    echeancier_nouvel_evenement(evt);
} else {
135   evt.type = fin_reparation;
      evt.date_evenement = horloge + duree_d_une_reparation;
      evt.composante = comp_a_reparer;
      echeancier_nouvel_evenement(evt);

140   reparateur_libre = 0; /* le reparateur est occupé */
}
}

145  /*****
/* Traiter le prochain événement et réagir si nécessaire en appliquant */
/* la politique de maintenance choisie. */
*****/
void traiter_evenement(echeancier_type_eve evenement)
150  {

    if (evenement.date_evenement < horloge)
    {
155      fprintf(stderr, "Erreur : chronologie non respectée !\n");
      exit(EXIT_FAILURE);
    }

    /* On met l'horloge à la date de l'événement */
    horloge = evenement.date_evenement;

160   /* On traite l'événement */
    switch(evenement.type)
    {
        case panne_composante : /* Mettre la composante en question en panne. */
165      reseau_comp_en_panne(evenement.composante, evenement.date_evenement);
        break;

        case fin_reparation :
            /* Mettre la composante en question en service */
170      reseau_comp_fonctionne(evenement.composante, evenement.date_evenement);

            /* Planifier la prochaine panne de la composante en question */
            planifie_panne_comp(evenement.composante);

175      /* le réparateur est de nouveau disponible */
            reparateur_libre = 1;
            break;

        case reveil :
180      break;

        default :
            fprintf(stderr, "Erreur : événement de type inapproprié\n");
            exit(EXIT_FAILURE);
185    }

    /* si le système ne fonctionne plus, la simulation s'arrête */
    if(!reseau_systeme_fonctionne()) {
        evenement.type = fin_simul;
190      /* on réinsère cet événement dans l'échéancier */
        echeancier_nouvel_evenement(evenement);
    } else {
        /* on applique la politique de réparation */
        politique_reparation();
195    }
}
}

```

```

/*****
200 /* Initialisation complète de la simulation : lecture du fichier de */
/* données (et création du problème) et création de l'échéancier. */
/*****/
void simulation_initialisation(char *nom_fichier, double duree_reparation, int
politique)
{
205 echeancier_cree(); /* Création d'un échéancier vide */

nb_comp = reseau_lecture_donnees(nom_fichier); /* Instancie le problème */
/* à partir du fichier et retourne le nombre de composantes du système */

210 switch(politique) /* Laquelle des 3 politiques de maintenance appliquer */
{
case 1 :
politique_reparation = politique_1;
break;
215 case 2 :
politique_reparation = politique_2;
break;
case 3 :
politique_reparation = politique_3;
220 reseau_analyse_3(duree_reparation); /* analyse de la structure du système
... */
break;
default :
fprintf(stderr,"Erreur : la politique %d n'a pas été définie\n",politique);
225 exit(EXIT_FAILURE);
}

if (duree_reparation>0) /* Quelle est la durée (constante) d'une réparation */
duree_d_une_reparation = duree_reparation;
else
230 {
fprintf(stderr,"Erreur : la durée %f d'une réparation est positive\n",
duree_reparation);
exit(EXIT_FAILURE);
}
}
235

/*****
/* Effectue une simulation et retourne la durée de vie obtenue. */
/*****/
240 double simulation_generer_une_duree_de_vie(void)
{
int i;
echeancier_type_eve prochain_evenement;

245 horloge = 0.0; /* Initialisation de l'horloge au temps t=0 */
reparateur_libre = 1; /* le réparateur est disponible */
echeancier_vider(); /* Vider l'échéancier */
for (i=1;i<=nb_comp;i++)
{
/* Toute composante... */
250 reseau_comp_fonctionne(i, horloge); /* est remise en état */
planifie_panne_comp(i); /* et sa première panne est planifiée. */
}

while ((prochain_evenement = echeancier_retire_prochain_evenement()).type !=
fin_simul)
255 traiter_evenement(prochain_evenement);

return prochain_evenement.date_evenement;
}

```

Listing 5 – statistiques.c

```

/*****
/*
/* Fichier : statistiques.c
/*
5 /* Auteur : E. Thiémarc, département E+I, EIVD
/* Version : Octobre 2004
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Ce module sert à calculer un intervalle de confiance
10 /* (seuil de confiance à 99%) de demi-largeur prescrite
/* à partir de mesures i.i.d. qui lui sont envoyées.
/*
/*****
/*
15 /* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 10.11.2004
/* modifs : Implémentation du calcul des statistiques
/*
/*****
20 /*
/* Consigne : à compléter conformément aux indications.
/*
/*****

25 #include <stdio.h>
#include <math.h>

static int n; /* nombre de réalisation */
30 static double moyenne; /* moyenne de la durée de vie du réseau */
static double ecart_type; /* écart type de la moyenne */
static double variance; /* variance des réalisation */

/* caractères affichés indiquant que le programme calcul... */
35 const char roue[] = {'-', '\\', '|', '/' };

/*****
/* Initialisation du module statistique.
40 /*****
void statistiques_initialisation(void)
{
n = 0;
moyenne = 0.0;
45 variance = 0.0;
ecart_type = 0.0;

printf("\nSimulation en cours %c", roue[0]);
fflush(stdout);
50 }

/*****
/* Envoie au module le résultat d'une nouvelle expérience et met à
55 /* jour les estimateurs statistiques en conséquence.
/*****
void statistiques_nouvelle_mesure(double mesure)
{
60 double moyenne_nouvelle = moyenne + ((mesure - moyenne) / (n+1));

/* Calcul de la variance possible uniquement dès de la deuxième itération */
if (n) {
variance = ((n-1)/(double)n) * variance + (n+1)
65 * pow(moyenne_nouvelle-moyenne, 2.0);
}
moyenne = moyenne_nouvelle;

```

```

n++;
/* 2.58 pour un i.c. à 99%, 1.96 pour un i.c. à 95% */
70  ecart_type = 2.58 * sqrt(variance) / sqrt(n);
}

/*****
75  /* Retourne 0 lorsqu'il a été possible de construire un intervalle de */
/* de demi-largeur suffisamment petite et 1 s'il faut continuer. */
/*****
int statistiques_condition_non_replie(double demilargeur)
{
80  /* on affiche régulièrement un caractère de progression */
  if (n % 101 == 4){
    printf("%c%c", 8, roue[n%4]);
    fflush(stdout);
85  }

  /* on continue tant qu'on a moins de 30 expériences */
  return (n < 30 || ecart_type > demilargeur);
90 }

/*****
/* Affiche les résultats : nombre de mesures, moyenne, demi-largeur */
/* et intervalle de confiance avec seuil de confiance à 99%. */
95  /*****
void statistiques_affiche_resultats(void)
{
  fprintf(stdout, "%c \n\nstatistiques affichage des resultats\n", 8);

100  fprintf(stdout, "nombre de realisations : %d\n", n);
  fprintf(stdout, "moyenne : %f\n", moyenne);
  fprintf(stdout, "variance : %f\n", variance);

  fprintf(stdout, "intervalle de confiance a 99%% : %f-%f\n",
105  moyenne - ecart_type, moyenne + ecart_type);
}

```