REGARDS CROISÉS SUR L'ALGORITHMIQUE ET LA PROGRAMMATION

PRÉSENTATION DE L'ESPRIT ET DU CONTENU QUE JE COMPTE METTRE DANS CETTE RUBRIQUE

Dans cette nouvelle rubrique je me propose de présenter des exemples d'activités **courtes** d'algorithmiques et de programmation au niveau du lycée. Elle accompagne l'apparition d'exercices d'algorithmique dans les épreuves du bac S et ES.

Si l'on veut faire de l'algorithmique autre chose qu'un simple exercice de bac (à l'instar de ceux du bac L, qui fournissent un bon fond), de quelques fractions de points, ce qui aura pour conséquence inéluctable de reléguer son enseignement au strict minimum piloté par ce nombre décimal de l'intervalle [0 ; 1], il nous semble qu'il faille proposer quelque chose de plus enrichissant et pour les élèves et pour les enseignants.

L'objectif principal est d'offrir aux enseignants de lycée un éventail d'exemples variés, tant du point de vue des mathématiques abordées que des langages utilisés, et suffisamment simples pour être réalisables par les élèves sans technicité excessive au niveau programmation.

Afin de ne pas trop me disperser, j'ai choisi de me limiter principalement au logiciel et langage de programmation **R** langage objet et fonctionnel moderne spécialisé dans les méthodes statistiques et doté d'un moteur graphique puissant.

L'algorithmique et les programmes qui la mettent en œuvre permettent :

1° D'illustrer divers aspects et notions nouvelles, et/ou délicates du cours de mathématique. **(Illustration Prolongement Cours)**

2° De servir d'outil pour surmonter certains obstacles didactiques. **(Obstacles Didactique)**

3° De servir de pourvoyeur de conjectures. **(Conjectures)**

4° De faire de l'algorithmique un véritable outil de résolution (numérique) de problème, certains élèves étant capables de trouver des solutions "algorithmiques" à des problèmes dont la solution mathématique classique leur échappe, (exemple de la simulation en probabilité). C'est aussi l'occasion de s'intéresser à des problèmes dont la solution mathématique est hors de portée des programmes du lycée. **(Résolution Problème)**

Les activités que je présente proviennent :

- Des propositions de corrigés de certains algorithmes du bac, avec des prolongements permettant des enrichissements tant du point de vue mathématique que du point de vue algorithmique et programmation.
- D'activités glanées dans les manuels de la seconde à la terminale, avec leurs enrichissements.
- D'activités un peu plus longues, pouvant donner lieu à un travail de groupe, sur plusieurs séances et faire l'objet de recherches.

Je m'efforcerai de favoriser autant l'aspect numérique que l'aspect graphique des problèmes traités, tirant bénéfice des excellentes efficacité et qualité du moteur graphique de **R** et des capacité graphiques des autres langages.

Non seulement l'illustration graphique constitue souvent une bonne approche didactique, mais, faisant appel à des algorithmes différents, elle nécessite de réinvestir la géométrie et l'analyse, en variant et enrichissant ainsi les activités.

Dans les textes des codes des programmes, les fonctions **R** sont en bordeaux, la couleur orange indique des lignes de commandes, le vert, les commentaires, le bleu, les fonctions créés lors des activités et le noir, les lignes de codes de ces fonctions.

2013 JUIN POLYNÉSIE Exercice 1_2° .

● Résumé de l'activité :

Je propose la mise en œuvre mot à mot de l'algorithme de l'énoncé puis huit prolongements concernant l'intégration, les petite et grande somme de Darboux (l'énoncé concerne la grande somme de Darboux) et comment les utiliser pour faire de l'intégration numérique. **Le premier** prolongement concerne l'utilisation des liste pour remplacer les boucles pour. **Le deuxième** fournit un algorithme permettant de tracer les rectangles sur le graphique représentatif de la fonction sur l'intervalle concerné. Dans **le troisième**, je montre comment prolonger l'algorithme pour calculer la petite somme de Darboux, dans le cas de la fonction de l'énoncé, positive, strictement croissante entre 0 et 1. **Le quatrième** montre comment atteindre une certaine précision pour le calcul de l'intégrale. **Le cinquième** aborde le prolongement pour un intervalle [a ; b] sur lequel la fonction est positive et strictement croissante. **Les sixième septième et huitième** prolongements abordent l'adaptation de l'algorithme à une fonction (intégrable au sens de Riemann) positive sur un intervalle [a ; b]. Il faut tenir compte du fait que la fonction n'est plus forcément monotone. Il faut donc que l'algorithme trouve le minimum et le maximum sur chacun des intervalles déterminés par la subdivision choisie.

● **Proposition de programmes** mettant en œuvre les algorithmes de l'énoncé tels quels.

f est la fonction définie sur R par f(x) = (x + 2) × e^{-x}.

L'algorithme ci-dessous permet d'obtenir une valeur approchée de l'aire du domaine $\mathcal{D}$ en ajoutant les aires des quatre rectangles précédents :

Variables :	k est un nombre entier
	S est un nombre réel
Initialisation :	Affecter à S la valeur 0
Traitement :	Pour k variant de 0 à 3
	Affecter à S la valeur $S + \frac{1}{4} f\left(\frac{k}{4}\right)$
	Fin Pour
Sortie :	Afficher S

Donner une valeur approchée à 10^{-3} près du résultat affiché par cet algorithme.

b. Dans cette question, N est un nombre entier strictement supérieur à 1. On découpe l'intervalle $[0\,;1]$ en N intervalles de même longueur. Sur chacun de ces intervalles, on construit un rectangle en procédant de la même manière qu'à la question 2.a.
Modifier l'algorithme précédent afin qu'il affiche en sortie la somme des aires des N rectangles ainsi construits.

```
# Exercice 1_2°_a
# Mise en œuvre "mot à mot" de l'algorithme
# Lignes de commandes R
# On peut ainsi avoir accès au contenu de
# toutes les variables, ce qui est
# nécessaire lorsque l'on veut rechercher
# une erreur ou contrôler le contenu
# des variables.
# Avec R il n'y a pas besoin de déclarer
# les variables. R les "crée" lors des
# affectations.
# Création de la fonction support
f <- function(x){(x+2) * exp(-x)}
# Initialisation
S <- 0
# Traitement
for (k in 0:3) {S <- S + f(k / 4) * 1 / 4}
# Sortie
cat("S =", S, "\n\n") # \n pour aller à la ligne

S = 1.641909
```

```
#-----------------------------
# Exercice 1_2°_b
# La subdivision est la suite
# k/N, k entier de 0 à N - 1
# Les rectangles correspondants
# ont pour largeur 1 / N et
# pour hauteur f(k / N)
# L'aire est donc f(k / N) * 1 / N

# Lignes de commandes R
f <- function(x){(x+2) * exp(-x)}
N <- 10
S <- 0
for (k in 0:(N - 1)) {
   S <- S + f(k / N) * 1 / N
}
cat("S =", S, "\n\n")

S = 1.573521
```

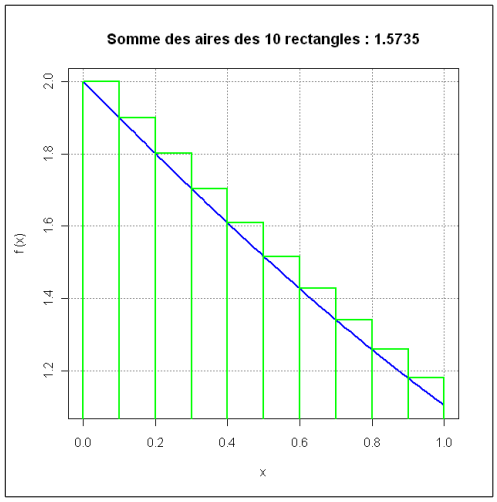
2013 JUIN POLYNÉSIE Exercice 1_2° Prolongements 1/6

- **Premier prolongement :** un changement de stratégie (d'algorithme) va nous permettre d'éviter la boucle for qui ralentit le programme par rapport à la version ci-dessous : On va créer des listes et utiliser des opérateurs sur les listes.

```
N <- 10 ; f <- function(x){(x+2) * exp(-x)} # C'est la fonction support
sub_S <- seq(0, N - 1, 1) / N # Création d'une liste contenant le subdivision régulière d'ordre N
                              # sur [0 ; (N-1)/N]
S <- sum(f(sub_S) / N) # L'image de cette subdivision par f × 1/N donne la liste des aires de tous
                       # les rectangles. On fait la somme de cette liste pour avoir l'aire totale.
                       # La fonction sum(...) effectue la somme d'une liste.
cat("Grand_S :", S, "\n\n") # Affichage du résultat, précédé d'un message. \n provoque un retour
                            # à la ligne
```

- **Deuxième prolongement :** représentation graphique des rectangles de l'énoncé. Les rectangles sont tracés en reliant les points de leurs sommets avec la fonction **polygon**`(listeX, listeY, ...)`. `listeX, listeY` sont respectivement les listes des abscisses et des ordonnées des quatre sommets de chaque rectangle.

```
# Lignes de commandes R
# Création de la fonction support
f <- function(x){(x+2) * exp(-x)}
# Initialisations
N <- 10
S <- 0
# Traitement
for (k in 0:(N - 1)) {
   S <- S + f(k / N) / N
}
# Sortie
cat("S =", S, "\n\n")
# Prolongement : Illustration graphique
# Courbe représentative de f
plot(f, 0, 1, col = "blue", lwd = 2,
   main = paste("Somme des aires des", N,
      "rectangles :", round(S, 4)))
   grid(col = "grey50")
# Tracé des rectangles sur le graphique.
   for (k in 0:(N - 1)) {
      xpoly <- c(k / N, k / N, (k + 1) / N, (k + 1) / N)
      ypoly <- c(0, f(k / N), f(k / N), 0)
      polygon(xpoly, ypoly, border = "green", lwd = 2)
   }

# La fonction polygon(listeX, listeY, ...) trace un
# polygone en reliant les points dont les coordonnées
# cartésiennes sont contenues
# dans la listeX et la listeY.
```



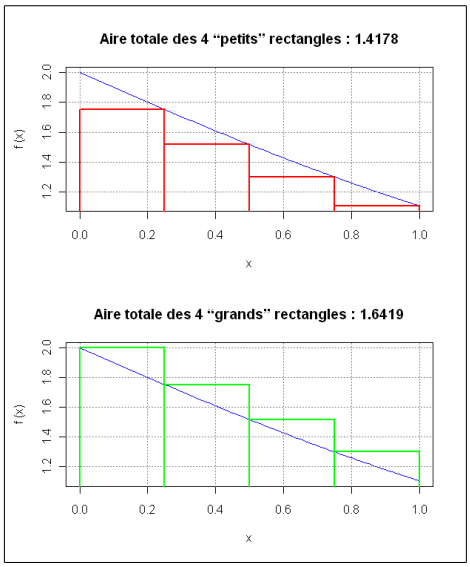
2013 JUIN POLYNÉSIE Exercice 1_2° Prolongements 2/6

- **Troisième prolongement :** L'exercice du bac calcule une grande somme de Darboux dans le cas d'une fonction (intégrable au sens de Riemann) positive décroissante sur l'intervalle [0 ; 1]. Cette grande somme consiste à prendre, comme hauteur du rectangle, le maximum de la fonction pour chaque intervalle défini par la subdivision. Nous prolongeons cette activité par le calcul de la petite somme de Darboux. Le calcul de la petite somme se fait comme la grande mais avec un décalage de 1 dans les indices.

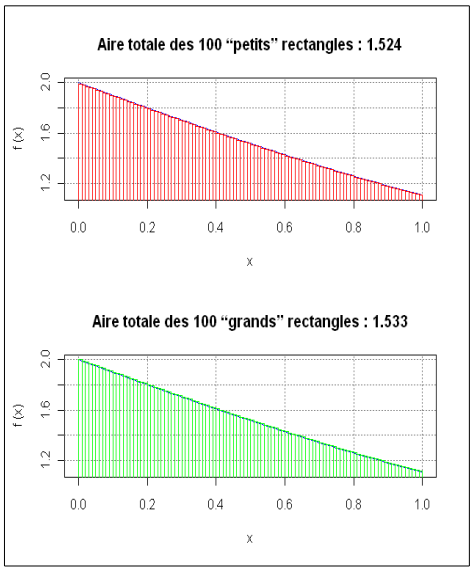
La représentation graphique permet de donner leur sens aux deux sommes de Darboux. Lorsque N grandit, on remarque que les deux sommes se rapprochent et que les deux graphiques se ressemblent.

```
# La version R_fonction avec représentation
# graphique des N rectangles, "grands" et "petits".
# L'algorithme construit une fonction R
f <- function(x){(x+2) * exp(-x)}
riemannD0_1 <- function(N = 4){
   Grand_S <- 0 ; Petit_s <- 0
   for (i in 0:(N - 1)) {
      Petit_s <- Petit_s + f((i + 1) / N) * 1 / N
      Grand_S <- Grand_S + f(i / N) * 1 / N
   }
   # Affichage des résultats
   cat("Grand_S :", Grand_S, "\nPetit_s :",
       Petit_s, "\n\n")
   # Les représentations graphiques
   par(mfrow = c(2, 1))
   # Les rectangles petit s
   plot(f, 0, 1, col = "blue",
        main = paste("Aire totale des", N,
                     ""petits" rectangles :",
          round(Petit_s, 4)))
      grid(col = "grey50")
      for (i in 0:(N - 1)) {
         lines(c(i / N, i / N, (i + 1) / N, (i + 1) / N),
               c(0, f((i + 1) / N), f((i + 1) / N), 0),
            col = "red")
      }
   # Les rectangles grand S
   plot(f, 0, 1, col = "blue",
        main = paste("Aire totale des", N,
                     ""grands" rectangles :",
          round(Grand_S, 4)))
      grid(col = "grey50")
      for (i in 0:(N - 1)) {
         lines(c(i / N, i / N, (i + 1) / N, (i + 1) / N),
               c(0, f(i / N), f(i / N), 0),
            col = "green")
      }
}

# La fonction lines(listeX, listeY, …) trace des
# segments de droite entre les points dont les
# coordonnées cartésiennes sont contenues
# dans la listeX et la listeY.
# C'est une alternative à la fonction polygon(...)
#
```

```
> riemannD0_1()
Grand_S : 1.641909
Petit_s : 1.417819
```




```
> riemannD0_1(N = 100)
Grand_S : 1.532966
Petit_s : 1.524003
```



2013 JUIN POLYNÉSIE Exercice 1_2° Prolongements 3/6

- **Quatrième prolongement :** Il s'agit d'atteindre une précision donnée pour le calcul d'une valeur approchée de l'intégrale recherchée. Le critère d'arrêt choisi (parmi d'autres) est la différence entre petite somme et grande somme.
  Le calcul des sommes et effectué à partir de la somme de listes plutôt qu'avec une boucle for, qui est plus lente (cf. premier prolongement).
  On peut travailler sur d'autres critères d'arrêt.

QUATRIÈME PROLONGEMENT :

Un critère d'arrêt est la différence entre grande Somme et petite somme de Darboux. Il s'agit de déterminer la plus petite valeur de N telle que la différence entre Grand_S et Petit_s soit inférieure ou égale à une "précision" donnée.

```
FONCTION R
f <- function(x){(x+2) * exp(-x)}

poly2013Prol3 <- function(e = 2){
   N <- 1 ; Petit_s <- f(1) ; Grand_S <- f(0)
   while (Grand_S - Petit_s > 10^-e) {
      N <- N + 1
      subGrand_S <- seq(0, N - 1, 1) / N
      subPetit_s <- seq(1, N, 1) / N
      Grand_S <- sum(f(subGrand_S) / N)
      Petit_s <- sum(f(subPetit_s) / N)
   }
   cat("N :", N, "\n\n")
   cat("Grand_S :", Grand_S, "\nPetit_s :", Petit_s, "\n\n")
}
```

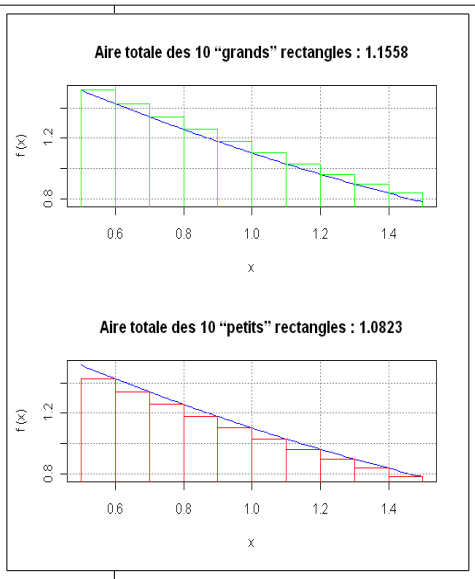
```
> poly2013Prol3(2)
N : 90

Grand_S : 1.533465
Petit_s : 1.523505
```

```
> poly2013Prol3(3)
N : 897

Grand_S : 1.528982
Petit_s : 1.527983

> poly2013Prol3(4)
N : 8964

Grand_S : 1.528532
Petit_s : 1.528432
```

2013 JUIN POLYNÉSIE Exercice 1_2° Prolongements 4/6

- **Cinquième prolongement :** Il s'agit maintenant de travailler sur **un intervalle [a ; b]** du domaine de définition sur lequel la fonction (intégrable au sens de Riemann) est positive décroissante.
  Le principal changement d'algorithme se passe au niveau du calcul de la subdivision. Son pas est (b – a) / N et la suite est a + i * (b – a) / N, avec i entier entre 0 et (N – 1) pour la grande somme et avec i entre 1 et N pour la petite somme. L'aire d'un rectangle est de la forme f(a + i * (b – a) / N) * (b – a) / N.

```
f <- function(x){(x+2) * exp(-x)}


riemannD <- function(a =0, b = 1, N = 4){
   sub <- (b - a) / N
   Grand_S <- 0 ; Petit_s <- 0
   for (i in 0:(N - 1)) {
      Grand_S <- Grand_S + f(a + i * sub) * sub
      Petit_s <- Petit_s + f(a + (i + 1) * sub) * sub
   }
   # Affichage des résultats
   cat("Grand_S :", Grand_S, "\nPetit_s :", Petit_s,
      "\nS - s :", Grand_S - Petit_s, "\n\n")
   # Les représentations graphiques
   par(mfrow = c(2, 1))
   # Les rectangles grand S
   plot(f, a, b, col = "blue",
         main = paste("Aire totale des", N,
            "“grands” rectangles :", round(Grand_S, 4)))
      grid(col = "grey50")
      for (i in 0:(N - 1)) {
         lines(c(a + i * sub, a + i * sub, a + (i + 1) * sub, a + (i + 1) * sub),
               c(0, f(a + i * sub), f(a + i * sub), 0), col = "green")
      }
   # Les rectangles petit s
   plot(f, a, b, col = "blue",
         main = paste("Aire totale des", N, "“petits” rectangles :",
            round(Petit_s, 4)))
      grid(col = "grey50")
      for (i in 0:(N - 1)) {
         lines(c(a + i * sub, a + i * sub, a + (i + 1) * sub, a + (i + 1) * sub),
               c(0, f(a + (i + 1) * sub), f(a + (i + 1) * sub), 0), col = "red")
      }
}
```



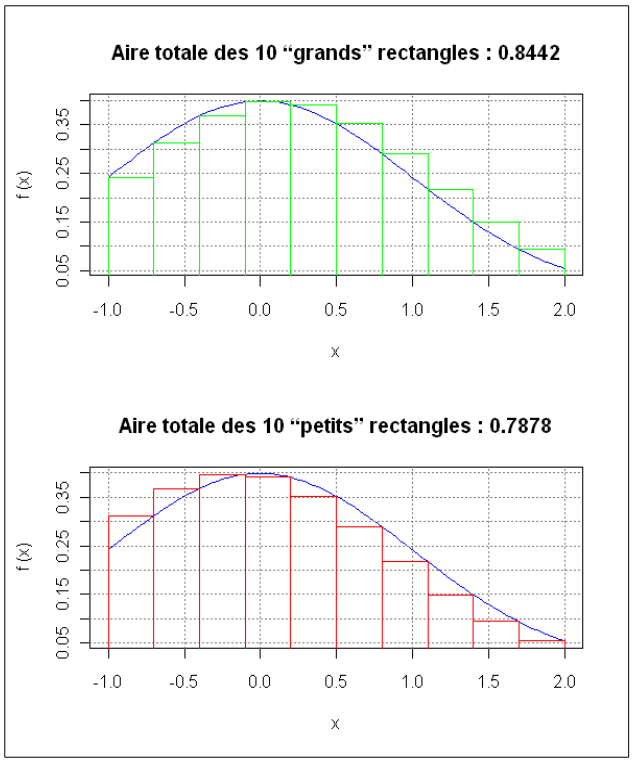

```
> riemannD(a = .5, b = 1.5,
N = 10)
Grand_S : 1.155834
Petit_s : 1.082296
S - s : 0.07353711
```

AUTRES CONTRIBUTIONS

2013 JUIN POLYNÉSIE Exercice 1_2° Prolongements 5/6

- **Sixième prolongement :** Comment adapter cet algorithme pour qu'il s'applique à une fonction (intégrable au sens de Riemann) positive croissante sur un intervalle [a ; b] ? Il suffit simplement d'intervertir petite somme et grande somme.(Une proposition de correction, par une fonction **R,** figure dans le fichier **"2013JuinPolynesieS.R"**).
- **Septième prolongement :** Comment adapter cet algorithme pour qu'il s'applique à une fonction positive monotone sur un intervalle [a ; b] ? On peut détecter, en début d'algorithme, si la fonction est croissante ou décroissante et puis utiliser les algorithmes précédents. Mais ce cas peut être traité d'une façon plus générale, par un algorithme différent, dans le prolongement suivant.
- **Huitième prolongement :** Et si l'on a **une fonction positive, non monotone** sur un intervalle [a ; b] ? Piste de recherche : voici ce que donne l'application de la fonction `riemannD(...)` au cas d'une fonction positive, non monotone, par exemple f(x) = $\frac{1}{\sqrt{(2\times\pi)}}\times(\exp(\frac{-x^2}{2}))$ sur l'intervalle [− 1 ; 2] :

```
> f <- function(x){exp(-x^2 / 2) / sqrt(2 * pi)}

> riemannD(a = -1, b = 2, N = 10)
Grand_S : 0.8441627
Petit_s : 0.7877688
S - s : 0.05639393

Après avoir commenté les résultats et graphiques
obtenus avec cette fonction R, que proposez-vous
pour construire un algorithme permettant de
calculer et d'illustrer les petites et grandes
sommes dans le cas d'une fonction positive non
monotone sur un intervalle [a ; b] ?
```



Une proposition d'algorithme figure page suivante.

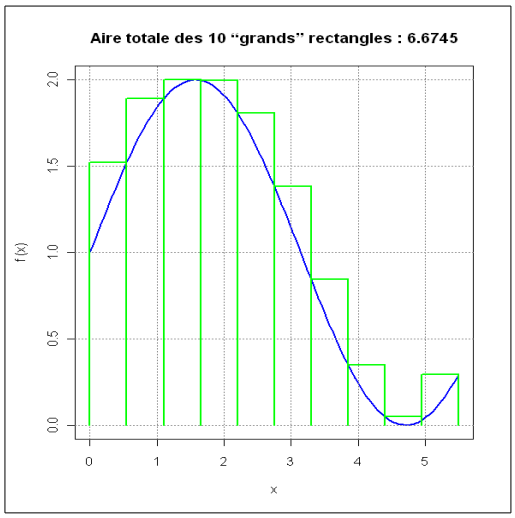
2013 JUIN POLYNÉSIE Exercice 1_2° Prolongements 6/6

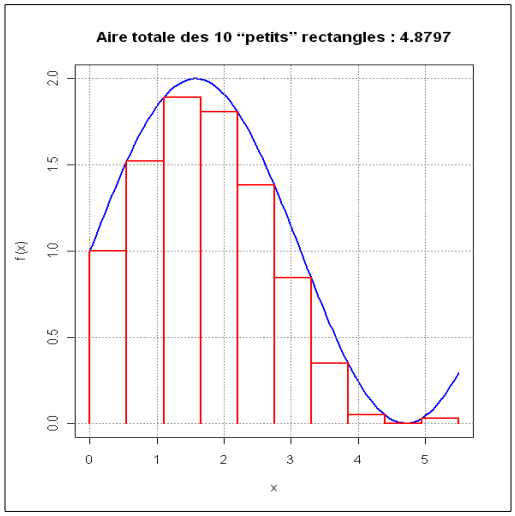
● Huitième prolongement (suite et fin) :

Pour chaque intervalle déterminé par la subdivision, il s'agit de trouver le minimum et le maximum de la fonction f, pour lesquels seront calculés la petite et la grande somme des aires des rectangles.

L'algorithme adopté pour déterminer ces optimum, consiste à découper chaque intervalle avec un subdivision et à déterminer le maximum et le minimum de l'image par f de ces subdivisions. Plus fine sera cette subdivision, plus précises seront les aires calculées.

```
# Fonction sinus(x) + 1 sur [0 ; 5.5]
f <- function(x){sin(x) + 1}
darboux <- function(a =0, b = 2 * pi, n = 10, dMn = 50){
   sub <- (b - a) / n
   Grand_S <- 0 ; Petit_s <- 0
   for (i in 0:(n - 1)) {
      Yi <- max(f(seq(a + i * sub, a + (i + 1) * sub, length = dMn)))
      yi <- min(f(seq(a + i * sub, a + (i + 1) * sub, length = dMn)))
      Grand_S <- Grand_S + Yi * sub
      Petit_s <- Petit_s + yi * sub
   }
   # Affichage des résultats
   cat("Grand_S :", Grand_S, "\nPetit_s :", Petit_s,
       "\nS - s :", Grand_S - Petit_s,
       "\nMoyenne des deux :", (Grand_S + Petit_s) / 2, "\n\n")
   # Les représentations graphiques
   # par(mfrow = c(2, 1))
   # Les rectangles grand S
   plot(f, a, b, col = "blue", lwd = 2,
        main = paste("Aire totale des", n, "“grands” rectangles :",
        round(Grand_S, 4)))
      grid(col = "grey50")
      for (i in 0:(n - 1)) {
         Yi <- max(f(seq(a + i * sub, a + (i + 1) * sub,
                         length = dMn)))
         lines(c(a + i * sub, a + i * sub,
                 a + (i + 1) * sub, a + (i + 1) * sub),
               c(0, Yi, Yi, 0),
               col = "green", lwd = 2)
      }
   # Les rectangles petit s
   plot(f, a, b, col = "blue", lwd = 2,
        main = paste("Aire totale des", n, "“petits” rectangles :",
        round(Petit_s, 4)))
      grid(col = "grey50")
      for (i in 0:(n - 1)) {
         yi <- min(f(seq(a + i * sub, a + (i + 1) * sub,
                         length = dMn)))
         lines(c(a + i * sub, a + i * sub,
                 a + (i + 1) * sub, a + (i + 1) * sub),
               c(0, yi, yi, 0),
               col = "red", lwd = 2)
      }
}
```





CONCLUSION

Voici comment, en huit étapes progressives, un exercice d'algorithmique de bac nous a mené vers les sommes de Darboux, calculées et illustrées grâce à la mise en œuvre d'algorithmes variés, réalisables par des élèves du lycée.

Ces algorithmes peuvent être utilisées pour faire des calculs approché d'intégrales, mais ce ne sont pas les outils couramment utilisés. On peut alors prolonger l'activité en présentant la méthode des trapèzes, et la méthode de Simpson qui est plus utilisée, qui est souvent celle qui est préprogrammée dans les calculettes.