Algorithmes en seconde : correction des exercices

Jean-Pierre Branchard - Lycée Monge, Chambéry

2 juillet 2009

Première partie

Les suites en javascript

En fait, j'ai traduit improprement le mot anglais "array" par "suite", parce que c'est le terme mathématiquement le plus adéquat.

On peut définir une suite en donnant la liste de ses éléments entre crochets :

```
var u=[0,1,4,9,16,25]
```

Si vous ne voulez pas des crochets (par crainte que vos élèves confondent les suites avec des intervalles), il est possible d'initialiser la suite avec le constructeur :

```
var u=new Array(0,1,4,9,16,25) ;
```

Avec les notations mathématiques usuelles, on a ainsi

$$u_0 = 0 \quad u_1 = 1 \quad u_2 = 4 \ldots$$

En syntaxe javascript, cela se traduira par

$$u[0] == 0 \quad u[1] == 1 \quad u[2] == 4$$

Quand on travaille sur une suite, on dispose de la propriété "length" pour connaître le nombre d'éléments de celle-ci. Ainsi l'instruction *afficher(u.length)* donnera 6.

On peut ajouter un élément à une suite avec la méthode *push*(élément). Par exemple :

```
u.push(36) ;
```

On aurait pu ainsi initialiser la suite avec une boucle et la méthode *push* :

```
var u=new Array() ;  //initialise une suite vide
for (var i=0 ; i<6 ; i++)
u.push(i*i) ;
```

On efface le dernier élément d'une suite avec la méthode *pop*(), par exemple u.pop() si la suite se nomme *u*.

Dans leurs versions les plus récentes, les navigateurs "Firefox" et "Opera" fournissent des méthodes puissantes pour manipuler les suites, on pourra par exemple initialiser un point A en écrivant :

```
var [xA, yA]=[2,-5] ;
```

alors que les autres navigateurs exigeront :

```
var xA=2 ;
var yA=-5 ;
```

De même, on pourra échanger les valeurs de deux variables x et y en écrivant :

```
[y,x]=[x,y] ;
```

Alors qu'avec les autres navigateurs, il faudra passer par un laborieux :

```
var temp ;
temp=x ;
x=y ;
y=temp ;
```

Deuxième partie

Correction des exercices

1 Additionner deux fractions

1.1 Algorithme

1. Saisie des fractions $\frac{a}{b}$ et $\frac{c}{d}$
2. Calculer le numérateur $ad + bc$ et le dénominateur bd de la somme
3. Afficher le résultat.

1.2 Programme

```
var a=demander(“a/b, entrer a :”) ;
var b=demander(“a/b, entrer b :”) ;
var c=demander(“c/b, entrer c :”) ;
var d=demander(“c/d, entrer d :”) ;
var num=a*d+b*c ;
var deno=b*d ;
afficher("somme non simplifiée : ",num,"/",deno) ;
```

2 Recherche de $[a\,;\,b] \cap [c\,;\,d]$

2.1 Algorithme

1. Demander les deux intervalles $[a\,;\,b]$ et $[c\,;\,d]$
2. Si $b < c$ ou bien $a > d$, l'intersection est vide
3. Sinon, calculer deux variables : m (maximum de a et de c) et M (minimum de b et de d) et afficher l'intervalle $[m\,;\,M]$

2.2 Programme

```
var a=demander("Intervalle [a,b], entrer a :") ;
var b=demander("Intervalle [a,b], entrer b :") ;
var c=demander("Intervalle [c,d], entrer c :") ;
var d=demander("Intervalle [c,d], entrer d :") ;
var m,M ;
if (b<c || d<a)
  afficher("ensemble vide") ;
else {
  m=Math.max(a,c) ;
  M=Math.min(b,d) ;
  afficher("[",m,"; ",M,"]") ;
}
```

3 Tableau de valeurs de la fonction $x \longmapsto \sqrt{x}$

3.1 Algorithme

On fera varier x entre 0 et 100

1. Construire un tableau à 2 colonnes
2. Définir une variable x et l'initialiser à zéro
3. Ajouter une ligne au tableau avec x et $\sqrt{x}$
4. Ajouter 5 à x
5. Si $x \leq 100$, retourner à l'étape 3

3.2 Programme

```
var t=new tableau("x","radical(x)") ;
for (x=0 ; x<101 ; x=x+5)
    t.ajouter(x,Math.sqrt(x)) ;
```

4 Jeu de hasard

4.1 Algorithme

1. Tirer au sort un entier naturel n entre 1 et 1000
2. Demander au joueur quel est le nombre n.
3. Si ce nombre est égal à n, informer le joueur qu'il a trouvé n
4. Si ce nombre est différent de n, informer le joueur s'il a entré un nombre trop petit ou trop grand et retourner à l'étape 2.

4.2 Programme

```
var n=Math.floor(Math.random()*1001) ;
var p=demander("entrer le nombre") ;
while(p!=n) {
 if (p<n){
   afficher(p,"<n") ;
   p=demander(p,"trop petit. Essayer un autre") ;
 }
 else {
   afficher("n<",p) ;
   p=demander(p,"trop grand. Essayer un autre") ;
 }
}
afficher("Bravo ! Le nombre est ",p) ;
```

5 Calcul de la factorielle d'un entier naturel

5.1 Algorithme

1. Demander le nombre n dont on doit calculer la factorielle
2. Initialiser une variable nommée "factorielle" en lui donnant la valeur 1
3. Si $n = 0$, passer directement à l'étape 8
4. Sinon, pour tout naturel i tel que $1 \leq i \leq n$, multiplier *factorielle* par i
5. Afficher la factorielle

5.2 Programme

```
var n=demander("Entrer n") ;
var factorielle=1 ;
if (n>0) {
  var i ;
  for (i=1 ; i<=n ; i++)
    factorielle=factorielle*i ;
}
afficher("factorielle(",n,")=",factorielle) ;
```

6 Fréquence du 6 quand on lance un dé

On lance 1000 fois un dé non truqué et on calcule la fréquence du 6 :

6.1 Algorithme

Il faut d'abord disposer d'une fonction nommée "lanceLeDe" qui simule le lancer d'un dé non truqué et retourne le résultat.

L'algorithme est alors le suivant :

1. Créer une variable i et une variable *total* et leur donner la valeur 0
2. Pour tous les naturels i tels que $0 \leq i < 1000$, faire :
   (a) Lancer le dé : si le résultat est 6, ajouter une unité à *total*
   (b) Ajouter une unité à i
3. Afficher la fréquence $\frac{total}{1000}$

6.2 Programme

```
function lanceLeDe(){
  return Math.floor(Math.random()*6+1) ;
}
var i ;
var total=0 ;
for (i=0 ; i<1000 ; i++)
  if (lanceLeDe()==6)
     total=total+1 ;
afficher(total/1000) ;
```

7 Algorithme de Babylone

7.1 Algorithme

1. Demander le nombre A dont il faut encadrer la racine carrée
2. Initialiser deux variables x et y en leur donnant respectivement les valeurs 1 et A
3. Tant que la distance entre x et y est supérieure à la précision recherchée, répéter les étapes (a) et (b) :
   (a) Affecter à x la valeur $\frac{x+y}{2}$
   (b) Affecter à y la valeur $\frac{A}{x}$
4. Afficher l'encadrement de $\sqrt{A}$ par x et y

7.2 Programme

Pour encadrer $\sqrt{A}$ à 10^{-3} près

```
var A=demander("A") ;
var x=1 ;
var y=A ;
while(Math.abs(x-y)>0.001){
  afficher("La racine de ", A, " est comprise entre ", x," et ", y) ;
  x=(x+y)/2 ;
  y=A/x ;
}
afficher("La racine de ", A, " est comprise entre ", x," et ", y) ;
```

8 Statistiques (min, max, $\bar{x}$, σ)

8.1 Algorithme

1. Demander la série statistique, qui prendra la forme d'une suite x
2. Initialiser une variable n avec l'effectif de la série statistique (nombre d'éléments de la suite)
3. Initialiser deux variables *somme* et *sommeCarre* à zéro. Au final, ces variables contindront les valeurs $\sum_{i=0}^{n} x_i$ et $\sum_{i=0}^{n} x_i^2$.
4. Initialiser deux variables *min* et *max* avec la valeur x_0.
5. Pour tous les naturels i compris entre 0 et $n-1$, faire :
   (a) Si $x_i < min$ alors *min* prend la valeur x_i
   (b) Si $x_i > max$ alors *max* prend la valeur x_i
   (c) Ajouter x_i à *somme*
   (d) Ajouter x_i^2 à *sommeCarre*
   (e) Ajouter 1 à i
   (f) fait
6. Afficher le minimum, le maximum, la moyenne $m = \frac{somme}{n}$ et l'écart type $s = \sqrt{\frac{sommeCarre}{n} - m^2}$

8.2 Programme

```
var x=demander("Série statistique") ;
var n=x.length ; //effectif
var somme=0 ;
var sommeCarre=0 ;
var min=x[0] ;
var max=x[0] ;
for (var i=0 ; i<n ; i++){
  if (x[i]<min)
    min=x[i] ;
  if (x[i]>max)
    max=x[i] ;
  somme=somme+x[i] ;
  sommeCarre=sommeCarre+x[i]*x[i] ;
}
afficher("min : ",min) ;
afficher("max : ",max) ;
var m=somme/n ;
afficher("m : ",somme/n) ;
afficher("s : ",Math.sqrt(sommeCarre/n-m*m)) ;
```

Remarque : la ligne d'instruction

```
for (i=1 ; i<n ; i++)
```

peut être remplacée par

```
for (i in x)
```

9 Méthode du tri-bulle

9.1 Algorithme

1. Demander la suite de nombres u qu'il faudra ranger dans l'ordre croissant
2. Initialiser une variable booléienne *triEnCours* avec la valeur *true*. Elle prendra la valeur *false* lorsque le tri sera terminé.
3. Tant que *triEnCours* a la valeur *true*, accomplir les taches suivantes :
   (a) Donner à *triEncours* la valeur *false*
   (b) Pour tout entier i tel que $1 \leq i \leq n$, faire
      i. Si $u_{i-1} > u_i$, échanger u_{i-1} avec u_i : pour cela, utiliser une variable temporaire à laquelle on donne la valeur u_i, puis donner la valeur u_{i-1} à u_i, puis donner à u_{i-1} la valeur de la variable temporaire. Redonner enfin la valeur *true* à la variable *triEnCours*
      ii. Ajouter 1 à la variable i
      iii. Si i est inférieur au nombre d'élement de la suite (*u.length*), retourner à l'étape (c)
      iv. fait
4. Afficher la suite triée

9.2 Programme

```
var u=demander("Entrer la suite de nombres") ;
var triEnCours=true ;
var i ;
var temporaire ;
while (triEnCours) {
  triEnCours=false ;
  for (i=1 ; i<u.length ; i++) {
    if (u[i]<u[i-1]) {
      // on échange les deux termes
      temporaire=u[i] ;
      u[i]=u[i-1] ;
      u[i-1]=temporaire ;
      triEnCours=true ;
    }
  }
}
afficher(u) ;
```

Remarque : la ligne d'instruction

```
for (i=1 ; i<u.length ; i++)
```

peut être remplacée par

```
for (i in u)
```

10 Recherche de $\sqrt[3]{2}$ par dichotomie

10.1 Algorithme

1. Initialiser deux variables a et b en leur donnant respectivement les valeurs 1 et 2, ainsi $a < \sqrt[3]{2} < b$
2. Tant que b-a est supérieur à la précision recherchée, répéter les étapes (a) et (b) :
   (a) Poser $c = \frac{a+b}{2}$
   (b) Si $c^3 < 2$, donnez à a la valeur de c, sinon, donner à b la valeur de c
3. Afficher a et b.

10.2 Programme

```
var a=1 ;
var b=2 ;
var c ;
while (b-a>0.000000001) {
  c=(a+b)/2 ;
  if (c*c*c<2)
    a=c ;
  else
    b=c ;
}
afficher (a,"<racine cubique de 2 <",b) ;
```

11 Algorithme d'Euclide

11.1 Algorithme

1. Demander les deux entiers dont on recherchera le PGCD.. On nomme respectivement a et b le plus petit et le plus grand d'entre eux.
2. Soit r le reste de la division euclidienne de b par a.
3. Tant que r est non nul, répéter les étapes (a) à (c) :
   (a) donner à a la valeur de b
   (b) donner à b la valeur de r
   (c) recalculer r
4. Afficher b.

11.2 Programme

Pour obtenir le pgcd de deux naturels non nuls :

```
var x=demander("entrer a") ;
var y=demander("entrer b") ;
var a=Math.max(x,y) ;
var b=Math.min(x,y) ;
afficher(a," ; ",b) ;
var r=a%b ; // reste de la div de a par b
while (r>0) {
  a=b ;
  b=r ;
  r=a%b
}
afficher("le pgcd est ",b) ;
```

12 Triplets de Pythagore

12.1 Algorithme

1. Poser $a = 2$
2. Poser $b = a + 1$
3. Poser $c = \sqrt{a^2 + b^2}$
4. Si c est entier, afficher a, b et c
5. Ajouter 1 à b
6. Si $b \leq 1000$, retourner à l'étape 3
7. Ajouter 1 à a
8. si $a < 1000$, retourner à l'étape 2

12.2 Programme

```
var a,b,c ;
for (a=2 ; a<1000 ; a++)
   for (b=a+1 ; b<1001 ; b++){
    c=Math.sqrt(a*a+b*b) ;
    if (c==Math.floor(c))
      afficher(a," ",b," ",c) ;
  }
```

13 Faire sortir toutes les faces d'un dé

13.1 Algorithme

1. Initialiser une suite nommée "faces" de 6 valeurs fausses. Au début de l'algorithme faces[0]=faces[1]=...=faces[5]=faux. Puis par la suite, faces[1] deviendra vraie (par exemple) lorsque la face 1 sortira.
2. Poser i=0 (nombre de lancers) et n=0 nombre de faces obtenues
3. tant que $n < 6$, répéter les étapes suivantes :
   (a) Lancer le dé : soit j la face obtenue
   (b) Vérifier si j sort pour la première fois (ce qui est le cas si faces[j] est fausse), si tel est le cas, augmenter n d'une unité et rendre faces[j] vraie
   (c) augmenter i d'une unité
4. Afficher i (nombre de lancer pour sortir les 6 faces)

13.2 Programme

Ce premier programme lance le dé jusqu'à ce que toutes les faces soient sorties :

```
// lance un dé numéroté de 0 à 5 et retourne la face obtenue
function lanceLeDe(){
 return Math.floor(6*Math.random()) ;
}
var i,j, n=0 ;
var faces=[false, false , false, false , false , false] ;
for (i=0 ; n<6 ; i=i+1) {
   j=lanceLeDe() ;
   if (!faces[j]) {
      faces[j]=true ;
      n=n+1 ;
   }
}
afficher(i) ;
```

Celui-ci réitère l'opération 1000 fois et calcule le nombre de lancers moyen :

```
// lance un dé numéroté de 0 à 5 et retourne la face obtenue
function lanceLeDe(){
 return Math.floor(6*Math.random()) ;
}
// lance le dé jusqu'à ce que toutes les faces soient sorties
// et retourne le nombre de lancers
function toutesLesFaces() {
  var i,j, n=0 ;
  var faces=[false, false , false, false , false , false] ;
  for (i=0 ; n<6 ;i++) {
       j=lanceLeDe() ;
       if ( !faces[j]) {
         faces[j]=true ;
           n=n+1 ;
       }
   }
 return i ;
}
// Réitère 1000 fois l'opération et calcule la moyenne :
var total=0 ;
for (k=0 ; k<1000 ; k++)
  total=total+toutesLesFaces() ;
afficher(total/1000) ;
```

14 Crible d'Erathostène

14.1 Algorithme

1. Initialiser une suite vide p de nombre premiers
2. Pour chaque entier i compris entre 2 et 100, faire :
   (a) initialiser une variable *prem* en lui donnant la valeur vraie, elle deviendra fausse si i n'est pas un nombre premier.
   (b) Pour chaque nombre premier p_j, tester si i est un mutiple de p_j, si c'est le cas, donner à prem la valeur fausse et arrêter les tests.
   (c) si prem est restée vraie, ajouter i à la suite p de nombres premiers.
3. Afficher la suite p

14.2 Programme

```
var p=[ ] ;
var prem=false ;
var i,j ;
for (i=2 ; i<100 ; i++) {
  // teste si i est premier
  prem=true ;
  // l'instruction suivante peur être remplacée par :  for (j in p)
  for (j=0 ; j<p.length ; j++)
    if (i%p[j]==0) {
      prem=false ;
      break ; //casse la boucle si i n'est pas premier
    }
  if (prem)
   p.push(i) ;
 }
// Affiche les nombres premiers
for (i=1 ; i<p.length ; i++)
  afficher(p[i]) ;
```

Ce programme n'est pas très performant, puisque il teste la divisibilité de chaque naturel par tous les nombres premiers précédemment trouvés. Or, pour savoir si 37 par exemple est un nombre premier, il n'est pas nécessaire de tester la

divisibilité par 31 : on pourrait s'arrêter au plus grand nombre premier inférieur à $\sqrt{37}$. Mais l'extraction de la racine carrée consomme aussi du temps de calcul ... pour éviter cela, on peut tenir en permanence à jour deux variables :

- Une variable "der" qui indique quel est le rang du dernier nombre premier à tester.
- Une variable "carre", égale au carré de ce nombre premier

D'où l'algorithme modifié (ça se complique !) :

```
var p=[2] ;
var prem=false ;
var der=0 ;
var carre=4 ;
var i=2,j,k ;
var N=10000 ;
while(i<=N) {
  while (i<carre) {
    // teste si i est premier
    prem=true ;
    for (j=0 ; j<=der ; j++) {
      if (i%p[j]==0) {
       prem=false ;
        break ;
      }
    }
   if (prem)
      p.push(i) ;
    i++ ; //synonyme de i=i+1
  }
 // incrémente der et carre
 der++ ;
 carre=Math.min(N+1,p[der]*p[der]) ;
}
// Affiche les nombres premiers
for (i=1 ; i<p.length ; i++)
  afficher(p[i]) ;
```

On peut se demander si on a vraiment gagné en vitesse d'exécution. Il est possible de mesurer la rapidité du programme (en ms) à l'aide du jeu d'intructions suivant :

```
var t0=new Date() ;
// insérer ici le programme dont on veut calculer le temps d'exécution
// (...)
var t1=new Date() ;
afficher("Durée : ",t1.getTime()-t0.getTime()) ;
```

On a mesuré ainsi le temps de recherche des nombres premiers inférieurs à 30 000 hors affichage. Avec le premier algorithme, les durées vont de 5622 ms à 5687 ms (10 essais) alors qu'avec le second, elles vont de 275 à 313 ms.

Avec un peu d'imagination, on peut sûrement faire encore mieux. par exemple, dans le second algorithme, initialisez i avec i=3 et remplacez l'instruction i++ par i=i+2... Les durées vont maintenant de 249 à 286 ms (est-ce plus rapide ?).

Pédagogiquement, il serait intéressant de trouver des problèmes un peu plus faciles, pour lesquelles plusieurs algorithmes peuvent être imaginés et de proposer aux élèves un concours pour trouver le plus rapide.

15 Représenter graphiquement la fonction racine carrée

15.1 Algorithme

1. Construire un nouveau repère et placer le style au point (0, 0)
2. Poser x=0
3. Pour chaque valeur de x comprise entre 0 et 10 avec un pas de 0.1, tracer un segment vers le point $(x\,;\,\sqrt{x})$

15.2 Programme

(on ne pourra pas exécuter ce programme sous M$ Internet Explorer)

```
var r=new repere(-1,-1,10,10,200, 200) ;
r.allerEn(0,0) ;
for (var x=0 ; x<10 ; x=x+0.1)
  r.ligneVers(x,Math.sqrt(x)) ;
```

16 Illustation de de 1+2+...+n

16.1 Algorithme

1. Demander la variable n
2. Construire un repère allant de (0 ; 0) jusqu'au point (n+1,n+1).
3. Construisons d'abord la suite des rectangles rouges : choisir la couleur de fond rouge ("red") puis, pour chaque entier i compris entre 1 et n, tracer le rectangle allant du point (0, 0) au point (i+1, i).
4. Construisons ensuite la suite des rectangles bleus : choisir la couleur de fond bleue puis, pour chaque entier i compris entre n+1 et 2, tracer le rectangle allant du point (i-1, i-1) au point (i, n+1).

16.2 Programme

(on ne pourra pas exécuter ce programme sous M$ Internet Explorer)

```
var n=demander("Entrer n") ;
var r=new repere(0,0,n+1,n+1,200,200) ;
r.couleurFond("red") ;
var i,j ;
for (i=1 ; i<=n ; i++)
   r.rectPlein(i,0,i+1,i) ;
r.couleurFond("blue") ;
for (i=n+1 ; i>1 ; i--)
  r.rectPlein(i-1, i-1, i, n+1) ;
```

17 Évolution d'une fréquence

17.1 Algorithme

1. Définir une fonction nommée "pointe()", qui prend la valeur 1 si la punaise tombe sur la pointe (p=2/3) et la valeur 0 sinon.
2. Définir un repère
3. Positionner le stylo en (0,0)
4. Initialiser une variable n à 0 (nombre de pointes)
5. Pour tous les entiers i compris entre 1 et 500, faire :
   (a) lancer la punaise, si elle tombe sur la pointe, ajouter 1 à n
   (b) tracer le segment allant jusqu'au point (i, n/i)

On peut modifier l'algorithme pour éviter que la courbe passe par l'origine (j'ai la flemme) et le compléter pour lui faire tracer l'asymptote d'équation $y = \frac{2}{3}$.

17.2 Programme

(on ne pourra pas exécuter ce programme sous M$ Internet Explorer)

```
//Probabilité
var p=2/3 ;
// simulation de la punaise
function pointe(){
  return (Math.random()<p) ;
}
//représentation graphique
var r=new repere(0,0,500,1,500,200) ;
var n=0 ;
var i ; r.couleurTrait("blue") ;
r.allerEn(0,0) ;
for (i=1 ; i<501 ; i++){
  if (pointe())
   n++ ;
  r.ligneVers(i,n/i) ;
}
// Asymptote
r.couleurTrait("red") ;
r.allerEn(0,p) ;
r.ligneVers(5000,p) ;
```

18 Estimation d'une aire (méthode de Monte-Carlo)

18.1 Algorithme

1. Construire un repère allant du point (0, 0) au point (2, 4).
2. poser $n = 0$
3. Répéter 100 000 fois les instructions suivantes :
   (a) tirer au sort un réel x entre 0 et 2
   (b) tirer au sort un réel y entre 0 et 4
   (c) si $y < x^2$, tracer le point (x, y) en vert et ajouter 1 à n, sinon le tracer en rouge
4. n est maintenant égal au nombre de points verts. Afficher $\frac{8n}{100000}$, qui est une estimation de l'aire recherchée

18.2 Programme

(on ne pourra pas exécuter ce programme sous M$ Internet Explorer)

```
var r=new repere(0,0,2,4,500,500) ;
var i,x,y,couleur ;
var n=0 ;
for (i=1 ; i<100000 ; i++) {
  x=Math.random()*2 ;
  y=Math.random()*4 ;
  if (y<x*x){
    couleur="green" ;
    n++ ;
  }
  else
    couleur="red" ;
  r.couleurFond(couleur) ;
  r.point(x,y) ;
}
afficher("Aire proche de ",8*n/100000) ;
```

19 Diagramme en bâtons

19.1 Algorithme

1. Demander deux suites : x (liste des valeurs) et y (liste des effectifs)
2. Initialiser les variables $xMin$, $xMax$ en leur donnant la valeur de x_0 et $nMax$ en lui donnant la valeur 0
3. Calcul de $xMin$ et de $xMax$: Pour chaque valeur x_i, comparer $xMin$, $xMax$, et x_i. Si $xMin > x_i$, poser $xMin = x_i$ et si $xMax < x_i$, poser $xMax = x_i$.
4. De même, calculer $nMax$ (effectif maximum)
5. Définir un repère allant du point (xMin-1, -1) au point (xMax+1, nMax+1)
6. Pour chaque valeur x_i, poser le stylo au point (x_i, 0) et tracer un segment vers le point (x_i, n_i)

19.2 Programme

(on ne pourra pas exécuter ce programme sous M$ Internet Explorer)

```
var x=demander("Liste des valeurs") ;
var n=demander("Liste des effectifs") ;
var nMax=n[0], xMax=x[0], xMin=x[0], i ;
for (i=0 ; i<x.length ; i++){
  if (x[i]>xMax)
    xMax=x[i] ;
  else if (x[i]<xMin)
    xMin=x[i] ;
}
for (i=0 ; i<n.length ; i++){
  if (n[i]>nMax)
    nMax=n[i] ;
}
var r=new repere(xMin-1, -1 , xMax+1, nMax+1, 200, 200) ;
r.couleurTrait("blue") ;
for (i=0 ; i<x.length ; i++) {
  r.allerEn(x[i],0) ;
  r.ligneVers(x[i],n[i]) ;
}
```

Variante :

```
var x=demander("Liste des valeurs") ;
var n=demander("Liste des effectifs") ;
var nMax=n[0], xMax=x[0], xMin=x[0], i ;
for (i in x){
  if (x[i]>xMax)
    xMax=x[i] ;
  else if (x[i]<xMin)
    xMin=x[i] ;
}
for (i in n){
  if (n[i]>nMax)
    nMax=n[i] ;
}
var r=new repere(xMin-1, -1 , xMax+1, nMax+1, 200, 200) ;
r.couleurTrait("blue") ;
for (i in x) {
  r.allerEn(x[i],0) ;
  r.ligneVers(x[i],n[i]) ;
}
```

20 Histogramme d'une série statistique continue

20.1 Algorithme

1. Demander deux suites : x (liste des bornes) et y (liste des effectifs) : il faudra une borne de plus que d'effectifs.
2. Initialiser une variable h en lui donnant la valeur 0 : h esprésentera la hauteur du graphique
3. Définir un repère allant du point $(x_0,\, 0)$ au point $(x_n,\, h)$
4. Calcul de h : chaque rectangle a une aire égale à y_i et une largeur égale à $(x_{i+1} - x_i)$ sa hauteur est donc $\frac{y_i}{x_{i+1}-x_i}$. chercher la hauteur maximale ainsi obtenue
5. Pour chaque valeur de i, tracer le rectangle allant du point $(x_i,\, 0)$ au point $\left(x_{i+1}\,,\, \frac{y_i}{x_{i+1}-x_i}\right)$

20.2 Programme

(on ne pourra pas exécuter ce programme sous M$ Internet Explorer)

```
var x=demander("entrer les bornes des intervalles") ;
var n=demander("entrer les effectifs") ;
// il faudra une borne de plus que d'effectifs !
var h=0 ;
var i ;
// Calcul des hauteurs
for (i=0 ; i<n.length ; i++) {
  if (h<n[i]/(x[i+1]-x[i]))
    h=n[i]/(x[i+1]-x[i]) ;
 }
var r=new repere(x[0]-1,0,x[x.length-1]+1,h+1,300,300) ;
r.couleurTrait("blue") ;
for (i=0 ; i<n.length ; i++) {
  h=n[i]/(x[i+1]-x[i]) ;
  r.rectCreux(x[i],0,x[i+1],h) ;
}
```