Fiche TD avec le logiciel : tdr7f

Cible de jeu de fléchettes

J.R. Lobry

Table des matières

1 Introduction

Une cible de jeu de fléchettes permet d'illustrer un certain nombre de points de probabilités ainsi que des propriétés des estimateurs de façon intuitive.

2 Code de la fonction `butt()`

Voici le code de la fonction `butt()`[1] permettant de représenter une cible de jeu de fléchettes sur un périphérique graphique :

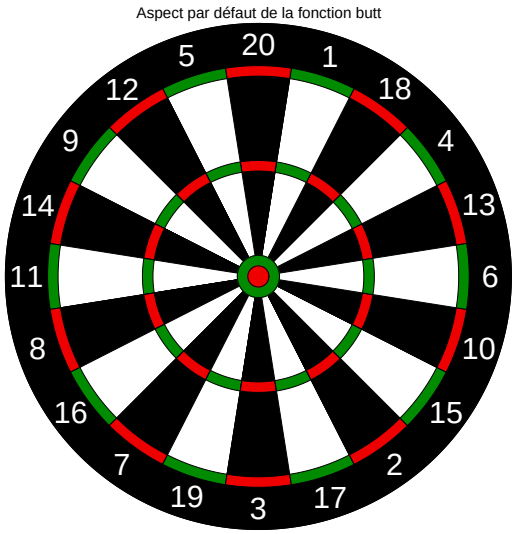
```
butt <- function(r = c(1.2, 1, 0.95, 0.55, 0.5, 0.1, 0.05), col = c("black",
    "red2", "green4", "white", "white", "black"), labels = c(13,
    4, 18, 1, 20, 5, 12, 9, 14, 11, 8, 16, 7, 19, 3, 17, 2, 15,
    10, 6), cex = 2, add = FALSE, isec = NULL, rsec = NULL, coladd = "yellow",
    legend = NULL) {
    old.par <- par(no.readonly = TRUE)
    on.exit(par(old.par))
    par(mar = c(0, 0, 0, 0) + 0.1)
    arc.circle <- function(xc = 0, yc = 0, radius = 1, rint = 0,
        from = 0, to = 2 * pi, n = 360, col = "blue", border = "black",
        ...) {
        theta <- seq(from = from, to = to, length = n * (to - from)/(2 *
            pi))
        x <- radius * cos(theta) + xc
        y <- radius * sin(theta) + yc
        xi <- rint * cos(theta) + xc
        yi <- rint * sin(theta) + yc
        if (abs(to - from) >= 2 * pi) {
            polygon(c(x, rev(xi)), c(y, rev(yi)), col = col, border = col,
                ...)
            lines(x, y, col = border)
            lines(xi, yi, col = border)
        }
        else {
            polygon(c(x, rev(xi)), c(y, rev(yi)), col = col, border = border,
                ...)
        }
        invisible(list(x = x, y = y))
    }
    if (!add) {
        plot.new()
        plot.window(xlim = c(-r[1], r[1]), ylim = c(-r[1], r[1]),
            asp = 1)
        arc.circle(radius = r[1], col = col[1], border = col[6])
        for (i in 0:19) {
            from <- i * pi/10 + pi/20
            to <- (i + 1) * pi/10 + pi/20
            arc.circle(radius = r[2], from = from, to = to, col = ifelse(i%%2 ==
                0, col[2], col[3]), border = col[6])
            arc.circle(radius = r[3], from = from, to = to, col = ifelse(i%%2 ==
                0, col[1], col[4]), border = col[6])
            arc.circle(radius = r[4], from = from, to = to, col = ifelse(i%%2 ==
                0, col[2], col[3]), border = col[6])
            arc.circle(radius = r[5], from = from, to = to, col = ifelse(i%%2 ==
                0, col[1], col[4]), border = col[6])
            theta <- (from + to)/2
            radius <- (r[1] + r[2])/2
            text(radius * cos(theta), radius * sin(theta), labels[i +
                1], col = col[5], cex = cex, xpd = NA)
        }
        arc.circle(radius = r[6], col = col[3], border = col[6])
        arc.circle(radius = r[7], col = col[2], border = col[6])
    }
    else {
        plot.window(xlim = c(-r[1], r[1]), ylim = c(-r[1], r[1]),
            asp = 1)
        for (i in 0:19) {
            from <- i * pi/10 + pi/20
            to <- (i + 1) * pi/10 + pi/20
```

[1] butt est en anglais l'ancien nom pour *dartboard*. Butt dérive du vieux français *butte* pour cible, que l'on retrouve de nos jours dans le mot *but* avec un sens très voisin.

```
            if (any(rsec == 1 & isec == labels[i + 1]))
                arc.circle(radius = r[2], rint = r[3], from = from,
                  to = to, col = coladd, xpd = NA, border = col[6])
            if (any(rsec == 2 & isec == labels[i + 1]))
                arc.circle(radius = r[3], rint = r[4], from = from,
                  to = to, col = coladd, xpd = NA, border = col[6])
            if (any(rsec == 3 & isec == labels[i + 1]))
                arc.circle(radius = r[4], rint = r[5], from = from,
                  to = to, col = coladd, xpd = NA, border = col[6])
            if (any(rsec == 4 & isec == labels[i + 1]))
                arc.circle(radius = r[5], rint = r[6], from = from,
                  to = to, col = coladd, xpd = NA, border = col[6])
        }
        if (any(rsec == 5))
            arc.circle(radius = r[6], rint = r[7], col = coladd,
                xpd = NA, border = col[6])
        if (any(rsec == 6))
            arc.circle(radius = r[7], col = coladd, xpd = NA, border = col[6])
    }
    if (!is.null(legend)) {
        mtext(legend, line = -1, xpd = NA)
    }
    invisible(par(no.readonly = TRUE))
}
```

Par défaut, l'appel de cette fonction donnera le résultat suivant :

```
butt(legend = "Aspect par défaut de la fonction butt")
```

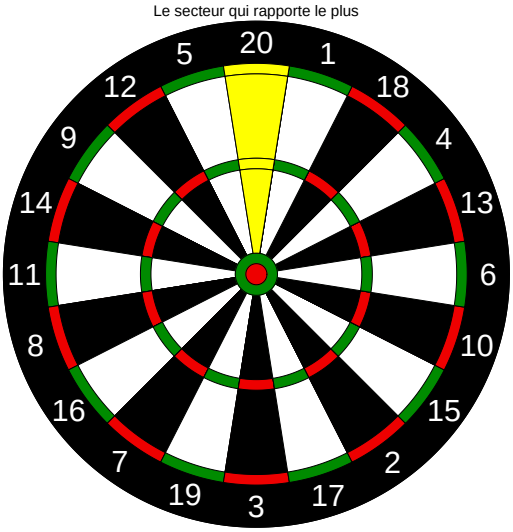


3 Les régles du jeu

La cible est divisée en 20 secteurs, ce qui est sans doute une relique de de l'usage d'un système vicésimal (de base 20), dont on trouve toujours des manifestations en français hexagonal[2]. Le secteur qui rapporte le plus de points est le secteur 20, ci dessous en jaune :

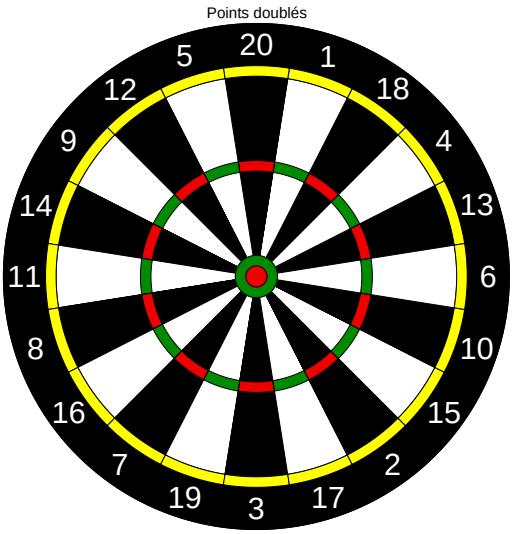
```
butt()
butt(add = TRUE, isec = rep(20, 4), rsec = 1:4, legend = "Le secteur qui rapporte le plus")
```

[2]par exemple quatre-vingt au lieu d'octante ou de huitante, les exemples sont nombreux.



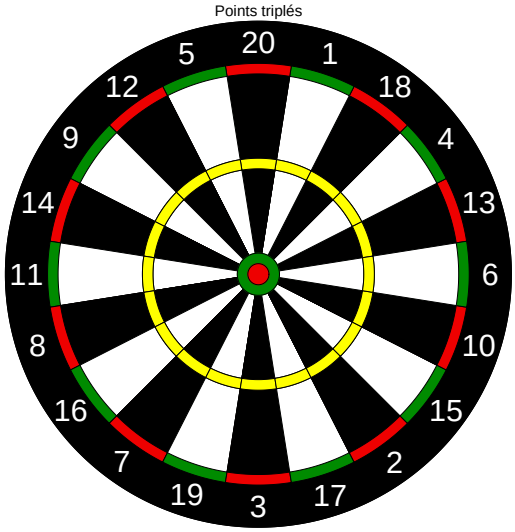
Sur l'anneau externe on double les points :

```
butt()
butt(add = TRUE, isec = 1:20, rsec = rep(1, 20), legend = "Points doublés")
```



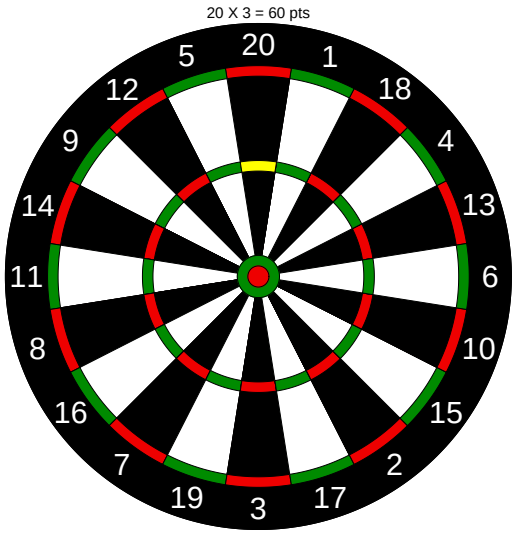
Sur l'anneau interne on triple les points :

```
butt()
butt(add = TRUE, isec = 1:20, rsec = rep(3, 20), legend = "Points triplés")
```



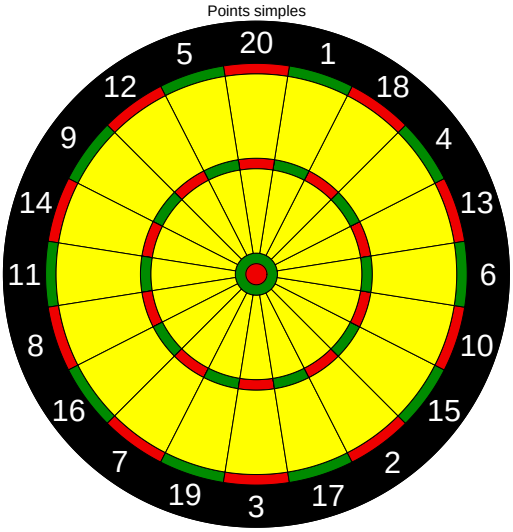
Le maximum de points possible est donc obtenu ici :

```
butt()
butt(add = TRUE, isec = 20, rsec = 3, legend = "20 X 3 = 60 pts")
```



Ailleurs on garde le nombre de points :

```
butt()
butt(add = TRUE, isec = rep(1:20, 2), rsec = rep(c(2, 4), each = 20),
    legend = "Points simples")
```

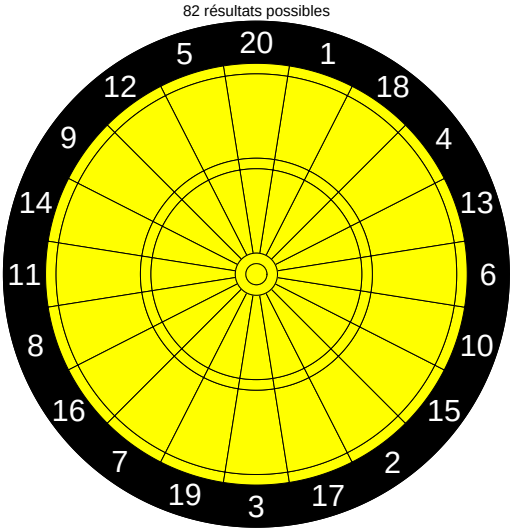


4 Probabilités

4.1 Théorie

L'univers des possibles, Ω, est l'ensemble des $20 \times 4 + 2 = 82$ événements élémentaires possibles :

```
butt()
butt(add = TRUE, isec = rep(1:20, 6), rsec = rep(c(1:6), each = 20),
    legend = "82 résultats possibles")
```



Supposons que la probabilité de toucher une zone soit proportionnelle à sa surface. Par défaut on a un cercle de rayon unité, et donc de surface totale égale à $\pi r^2 = \pi$. On utilise les valeurs par défaut des rayons de la fonction `butt()` :

```
args(butt)
```

```
function (r = c(1.2, 1, 0.95, 0.55, 0.5, 0.1, 0.05), col = c("black",
    "red2", "green4", "white", "white", "black"), labels = c(13,
    4, 18, 1, 20, 5, 12, 9, 14, 11, 8, 16, 7, 19, 3, 17, 2, 15,
    10, 6), cex = 2, add = FALSE, isec = NULL, rsec = NULL, coladd = "yellow",
    legend = NULL)
NULL
```

La probabilité de toucher un secteur double, p_1, est :

$$p_1 = \frac{1}{20\pi}(\pi r_1^2 - \pi r_2^2) = \frac{1}{20}(r_1^2 - r_2^2) = \frac{1}{20}(1^2 - 0.95^2) = 0.004875 \quad (1)$$

La probabilité de toucher un secteur simple externe, p_2, est :

$$p_2 = \frac{1}{20}(r_2^2 - r_3^2) = \frac{1}{20}(0.95^2 - 0.55^2) = 0.03 \quad (2)$$

La probabilité de toucher un secteur triple, p_3, est :

$$p_3 = \frac{1}{20}(r_3^2 - r_4^2) = \frac{1}{20}(0.55^2 - 0.5^2) = 0.002625 \quad (3)$$

La probabilité de toucher un secteur simple interne, p_4, est :

$$p_4 = \frac{1}{20}(r_4^2 - r_5^2) = \frac{1}{20}(0.5^2 - 0.1^2) = 0.012 \quad (4)$$

La probabilité de toucher le simple mil, p_5, est :

$$p_5 = r_5^2 - r_6^2 = 0.1^2 - 0.05^2 = 0.0075 \quad (5)$$

La probabilité de toucher le double mil, p_6, est :

$$p_6 = r_6^2 = 0.05^2 = 0.0025 \quad (6)$$

La somme des probabilités élémentaires doit faire 1, vérifions :

```
p1 <- (1 - 0.95^2)/20
p2 <- (0.95^2 - 0.55^2)/20
p3 <- (0.55^2 - 0.5^2)/20
p4 <- (0.5^2 - 0.1^2)/20
p5 <- 0.1^2 - 0.05^2
p6 <- 0.05^2
sum(20 * (p1 + p2 + p3 + p4) + p5 + p6)
```

```
[1] 1
```

Soit X la variable aléatoire nombre de points. Les valeurs possibles pour les 82 événements élémentaires sont :

```
(X <- c(1:20, 2 * 1:20, 3 * 1:20, 25, 50))
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  2  4  6  8 10 12 14
[28] 16 18 20 22 24 26 28 30 32 34 36 38 40  3  6  9 12 15 18 21 24 27 30 33 36 39 42
[55] 45 48 51 54 57 60 25 50
```

Les probabilités associées sont :

```
(p <- rep(c(p2 + p4, p1, p3, p5, p6), c(20, 20, 20, 1, 1)))
```

```
 [1] 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000
[10] 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000 0.042000
[19] 0.042000 0.042000 0.004875 0.004875 0.004875 0.004875 0.004875 0.004875 0.004875
[28] 0.004875 0.004875 0.004875 0.004875 0.004875 0.004875 0.004875 0.004875 0.004875
[37] 0.004875 0.004875 0.004875 0.004875 0.002625 0.002625 0.002625 0.002625 0.002625
[46] 0.002625 0.002625 0.002625 0.002625 0.002625 0.002625 0.002625 0.002625 0.002625
[55] 0.002625 0.002625 0.002625 0.002625 0.002625 0.002625 0.007500 0.002500
```

Il nous faut maintenant regrouper les valeurs dupliquées :

```
(Xu <- unique(X))
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 22 24 26 28 30 32 34
[28] 36 38 40 21 27 33 39 42 45 48 51 54 57 60 25 50
```

Et sommer les probabilités correspondantes :

```
(pu <- sapply(Xu, function(x) sum(p[X == x])))
```

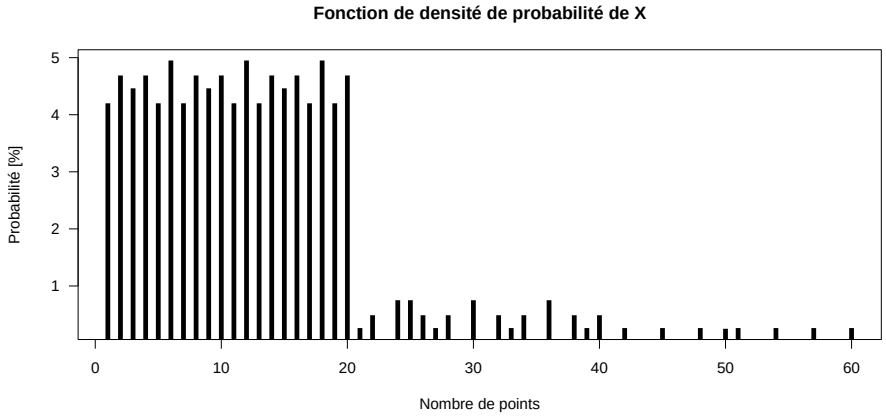
```
 [1] 0.042000 0.046875 0.044625 0.046875 0.042000 0.049500 0.042000 0.046875 0.044625
[10] 0.046875 0.042000 0.049500 0.042000 0.046875 0.044625 0.046875 0.042000 0.049500
[19] 0.042000 0.046875 0.004875 0.007500 0.004875 0.004875 0.007500 0.004875 0.004875
[28] 0.007500 0.004875 0.004875 0.002625 0.002625 0.002625 0.002625 0.002625 0.002625
[37] 0.002625 0.002625 0.002625 0.002625 0.002625 0.007500 0.002500
```

```
sum(pu)
```

```
[1] 1
```

Nous pouvons maintenant représenter la fonction de densité de probabilité de X :

```
plot(Xu, 100 * pu, type = "h", las = 1, lwd = 5, lend = "butt",
    xlab = "Nombre de points", ylab = "Probabilité [%]", main = "Fonction de densité de probabilité de X")
```



L'espérance de X, $E(X) = \sum_{i=1}^{n} p_i x_i$, vaut :

```
sum(pu * Xu)
```

```
[1] 12.83375
```

La variance de X, $V(X) = \sum_{i=1}^{n} p_i (x_i - \bar{x})^2$, vaut :

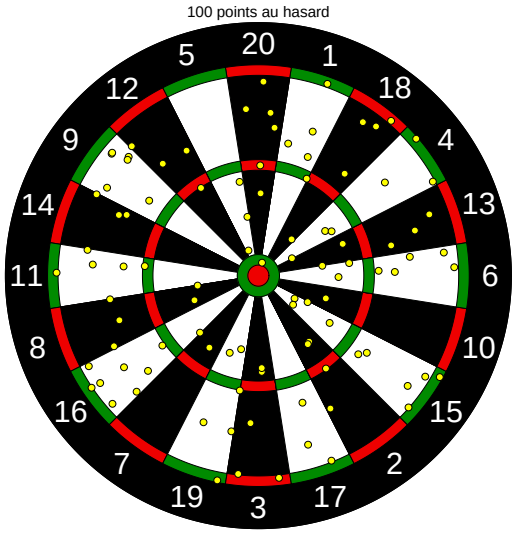
```
sum(pu * (Xu - sum(pu * Xu))^2)
```

```
[1] 90.54111
```

4.2 Vérification expérimentale

Nous avons besoin d'une fonction pour tirer des points avec une distribution uniforme dans le cercle. On procède ici par exclusion.

```
cunif <- function(n) {
    x <- runif(n, -1, 1)
    y <- runif(n, -1, 1)
    d <- x^2 + y^2
    nout <- sum(d > 1)
    while (nout > 0) {
        x[d > 1] <- runif(nout, -1, 1)
        y[d > 1] <- runif(nout, -1, 1)
        d[d > 1] <- x[d > 1]^2 + y[d > 1]^2
        nout <- sum(d > 1)
    }
    return(list(x = x, y = y))
}
par(butt(legend = "100 points au hasard"))
tmp <- cunif(100)
points(tmp$x, tmp$y, pch = 21, bg = "yellow")
```



Nous avons besoin d'une fonction pour compter les points, en voici une version vectorisée :

```
butt.points <- function(x, y, r = eval(as.list(args(butt))$r), labels = eval(as.list(args(butt))$labels)) {
    onepoint <- function(x, y) {
        dist <- sqrt(x^2 + y^2)
        theta <- atan2(y, x)
        sector <- 20 * theta/(2 * pi) - 1/2
        if (sector < 0)
            sector <- 20 + sector
        sector <- floor(sector)
        if (dist > r[2])
            return(0)
        if (dist > r[3])
            return(2 * labels[sector + 1])
        if (dist > r[4])
            return(labels[sector + 1])
        if (dist > r[5])
            return(3 * labels[sector + 1])
        if (dist > r[6])
            return(labels[sector + 1])
        if (dist > r[7])
            return(25)
```

```
        return(50)
    }
    mapply(onepoint, x, y)
}
```

Faisons 10000 lancers de fléchette et représentons la distribution du nombre de points.

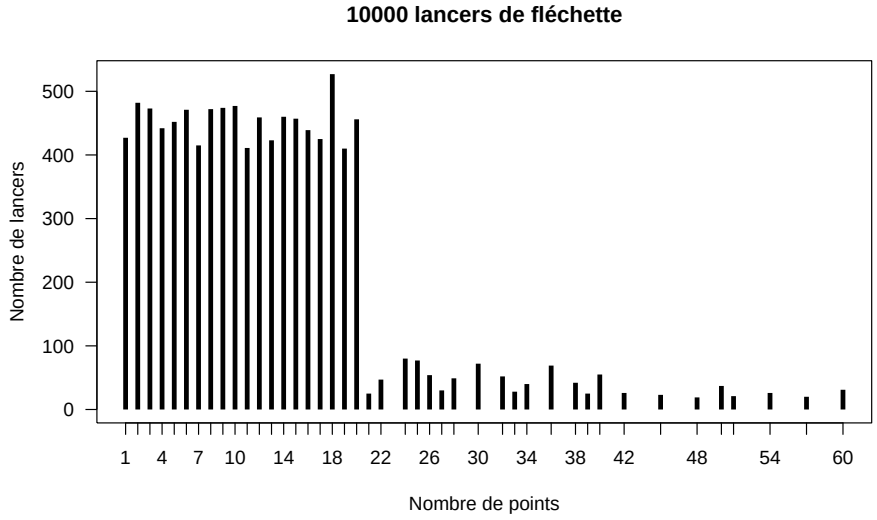
```
n <- 10000
res <- cunif(n)
pts <- butt.points(res$x, res$y)
mean(pts)
```

```
[1] 12.9376
```

```
var(pts)
```

```
[1] 93.50746
```

```
plot(table(pts), type = "h", lwd = 4, lend = "butt", xlim = c(1,
    60), las = 1, main = "10000 lancers de fléchette", xlab = "Nombre de points",
    ylab = "Nombre de lancers")
```



On retrouve bien les résultats théoriques.

5 Où viser ?

La distribution uniforme sur le cercle n'est pas très réaliste car quand on lance une fléchette on vise un certain point. Supposons que dans la vraie vie quand on tire une fléchette les coordonnées soient dans une loi normale bivariée sans corrélation entre les abscisses et les ordonnées :

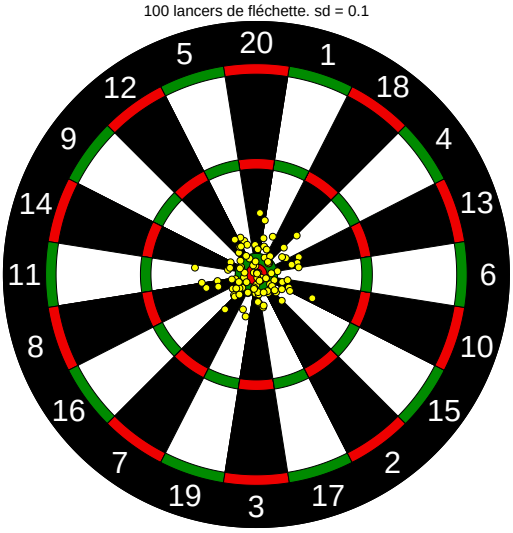
$$\phi(X,Y) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{\frac{-(X^2+Y^2)}{2\sigma^2}} \tag{7}$$

Par exemple :

```
par(butt(legend = "100 lancers de fléchette. sd = 0.1"))
sd <- 0.1
points(rnorm(100, sd = sd), rnorm(100, sd = sd), pch = 21, bg = "yellow")
```

Classe	Résultat	σ typique
Très bon	Au moins 30 % dans le simple ou double mil	0.5
Bon	10-30 % dans le simple ou double mil	1.0
Moyen	Au moins 85 % à l'intérieur du rayon interne du triple	1.6
Mauvais	70-85 % à l'intérieur du rayon interne du triple	2.1
Très Mauvais	Moins de 70 % à l'intérieur du rayon interne du triple	3.1

TAB. 1 – Les cinq classes de joueurs selon David Kohler [3]. Il faut lancer une cinquantaine de fléchettes en visant le double mil et noter les résultats. Les écart-types σ sont exprimés en pouces, soit 2.54 cm.



D'après David Kohler [3] c'est une assez bonne approximation de la réalité. Dans ses expériences conduites avec trois joueurs, il n'a pas réussi à mettre en évidence de corrélation entre les abscisses et les ordonnées, ni de différence entre la dispersion horizontale et verticale, même si les joueurs ont le sentiment d'être légèrement plus précis à l'horizontale qu'à la verticale.

L'écart type, σ, est une mesure de l'habileté d'un joueur : un joueur très habile aura une très faible dispersion de ses lancers et donc un σ faible ; un joueur médiocre aura une grande dispersion de ses lancers, et donc un σ élevé. Pour déterminer votre classe de joueur, David Kohler propose [3] de lancer une cinquantaine de fléchettes et de vous référer à la table 1.

Le code suivant permet de générer la figure 1 pour visualiser les résultats typiques pour les 5 classes de joueurs.

```
ntir <- 50
classn <- c("Très bon", "Bon", "Moyen", "Mauvais", "Très mauvais")
sdn <- c(0.5, 1, 1.6, 2.1, 3.1)/(6 + 5/8)
par(mfrow = c(2, 3), oma = c(0, 0, 1, 0))
for (i in 1:5) {
    par(butt(cex = 0.8, col = c(rep("white", 4), rep(grey(0.5),
        2)), legend = classn[i]))
    par(mfg = c((i - 1)%/%3 + 1, (i - 1)%%3 + 1))
    x <- rnorm(ntir, sd = sdn[i])
    y <- rnorm(ntir, sd = sdn[i])
    points(x, y, pch = 21, bg = "yellow", xpd = NA)
}
```

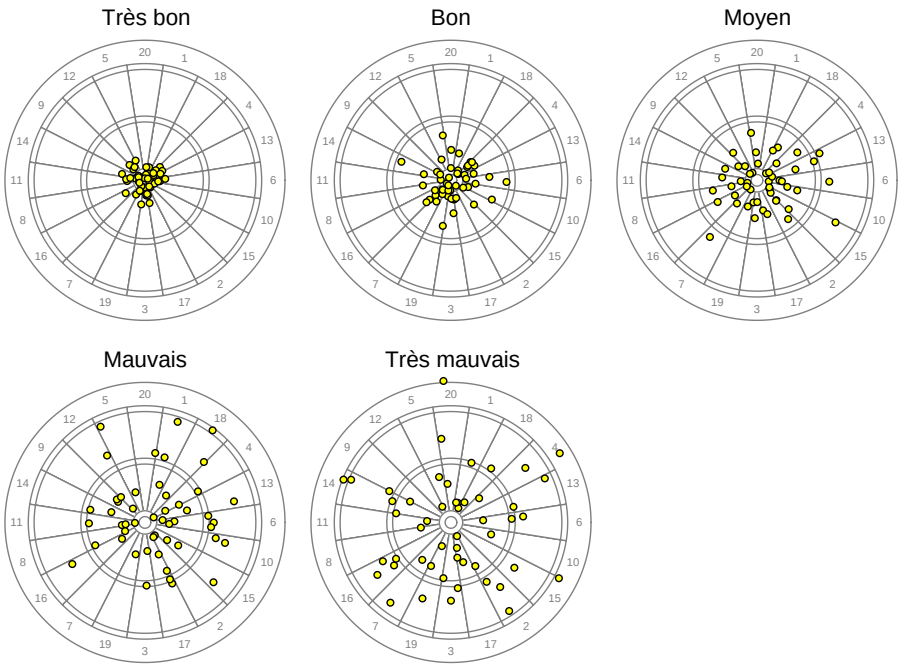


FIG. 1 – Les résultats typiques des joueurs visant le centre de la cible en fonction de leur niveau de jeu selon David Kohler [3].

Dans ces conditions, où avons nous intérêt de viser pour maximiser le nombre de points ? Le code suivant permet de générer la figure 2, attention, temps de calcul assez long.

```
ngrid <- 50
ntirs <- 1000
x <- seq(-1, 1, le = ngrid)
y <- seq(-1, 1, le = ngrid)
mycols <- apply(col2rgb(topo.colors(12)), 2, function(x) rgb(x[1]/255,
    x[2]/255, x[3]/255, 0.5))
par(mfrow = c(2, 3), oma = c(0, 0, 1, 0))
for (cls in 1:5) {
    par(butt(col = c("white", "white", "white", "white", grey(0.5),
        "black"), legend = classn[cls], cex = 0.8))
    par(mfg = c((cls - 1)%/%3 + 1, (cls - 1)%%3 + 1))
    resultat <- matrix(NA, ncol = ngrid, nrow = ngrid)
    for (i in seq_len(ngrid)) {
        for (j in seq_len(ngrid)) {
            tirx <- rnorm(ntirs, sd = sdn[cls], mean = x[i])
            tiry <- rnorm(ntirs, sd = sdn[cls], mean = y[j])
            pts <- sum(butt.points(tirx, tiry))/ntirs
            resultat[i, j] <- pts
        }
    }
    image(x, y, resultat, add = T, col = mycols)
    contour(x, y, resultat, add = T)
}
```

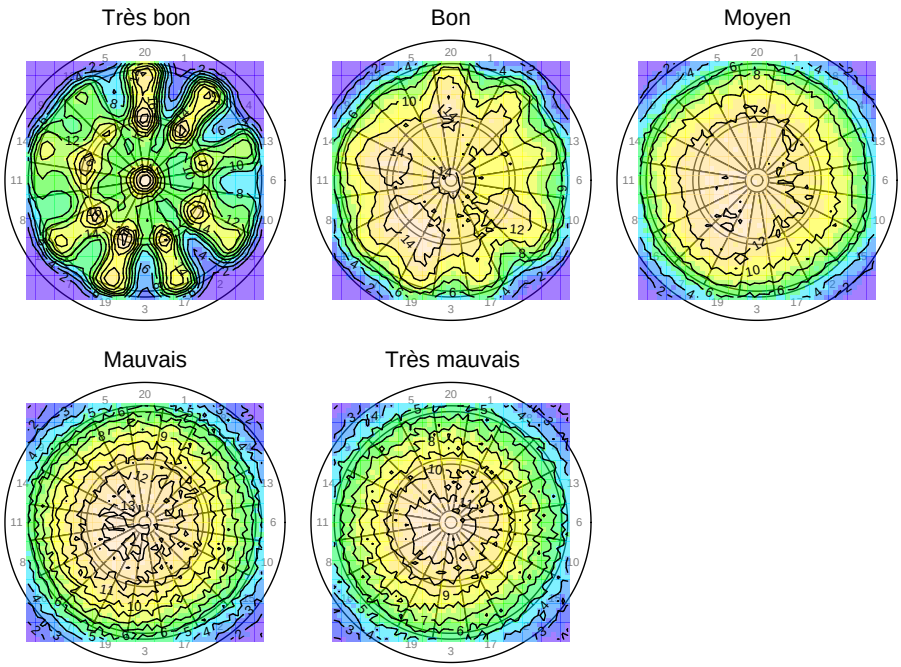


FIG. 2 – Où viser en fonction de son niveau de jeu pour maximiser l'expérance du nombre de points. Un très bon joueur visera le triple 20. Un bon joueur a intérêt à viser le triple 7 pour tomber dans une zone assez large où il fait en moyenne plus de 14 points. Un joueur moyen à intérêt à viser légèrement à gauche pour espérer plus de 12 points. Les mauvais et très mauvais joueurs ont intérêt à viser le centre de la cible.

6 Annexe : les paramètes de la fonction `butt()`

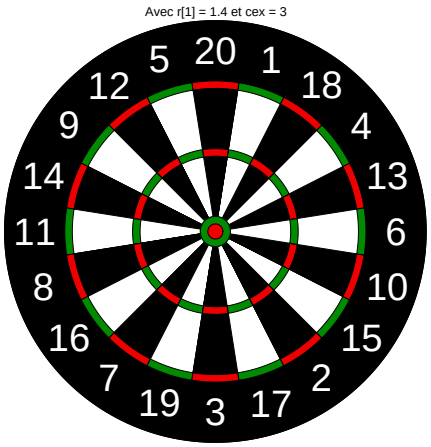
6.1 Les valeurs du vecteur de paramètres `r`

Je n'ai pas trouvé de norme officielle pour les dimensions exactes de la cible, les valeurs par défaut ont été ajustée par la méthode pifométrique [5] pour correspondre à l'aspect habituel d'une cible de jeu de fléchettes. Elles sont néamoins ajustables avec le paramètre `r` de la fonction `butt()`. C'est un vecteur de 7 rayons dont la signification est illustrée ci-après.

6.1.1 `r[1]`

Le paramètre `r[1]` contrôle l'anneau extérieur qui ne sert qu'à afficher le nom des secteurs et n'a pas d'incidence sur les probabilités. Il peut être utile en conjonction avec le paramètre `cex` pour augmenter la lisibilité du nom des secteurs, par exemple :

```
butt(r = c(1.4, 1, 0.95, 0.55, 0.5, 0.1, 0.05), cex = 3, legend = "Avec r[1] = 1.4 et cex = 3")
```



6.1.2 `r[2]`

Le paramètre `r[2]` contrôle le rayon maximum de l'univers des possibles, il vaut par défaut 1. Il n'y a pas de raison de le modifier sauf à introduire une mesure physique de la dimension de la cible. Pensez à modifier proportionellement tous les rayons en ce cas. Par exemple, les dimension données par David Kohler [3] sont indiquées dans la figure 3. Voici comment les reproduire :

```
rpc <- c(8, 6 + 5/8, 6 + 1/4, 4 + 1/8, 3 + 3/4, 5/8, 1/4)
butt(r = rpc, legend = "La cible de David Kohler")
```

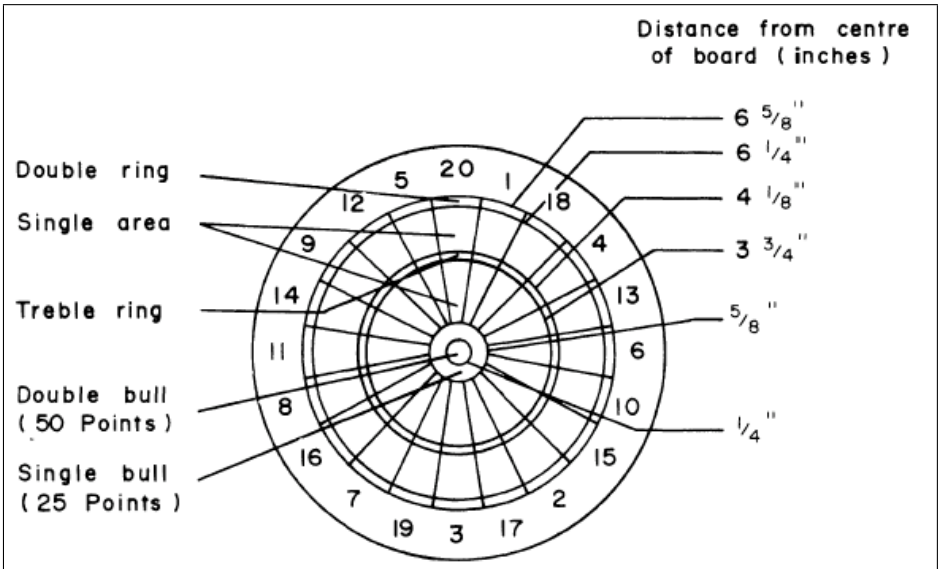
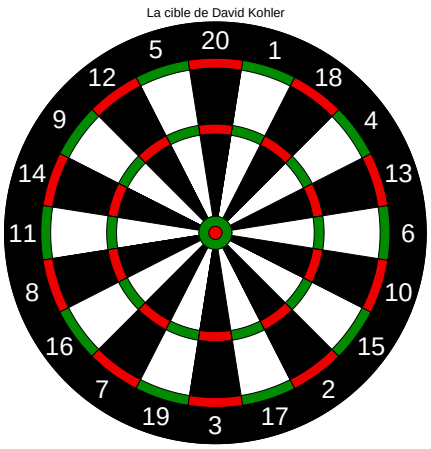


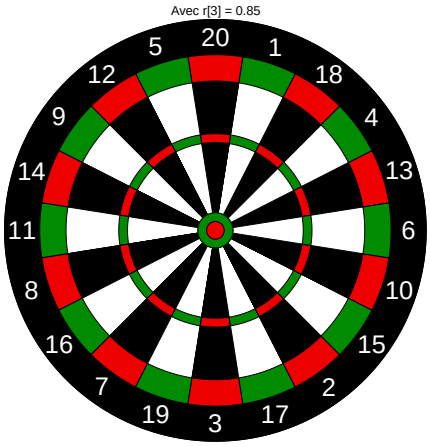
FIG. 3 – Les dimensions de la cible données par David Kohler [3]. Les valeurs sont exprimées en pouces, 1 pouce vaut 2.54 cm. Cette figure n'est pas à l'échelle.



6.1.3 `r[3]`

Le paramètre `r[3]` contrôle l'épaisseur de l'anneau des doubles, par exemple :

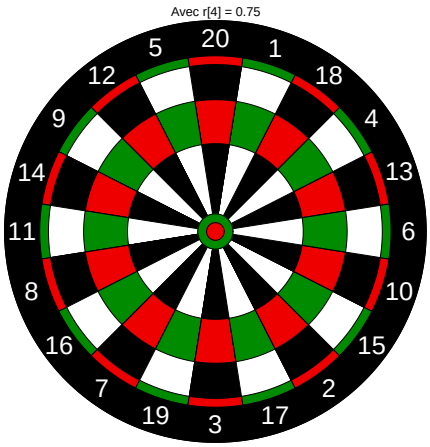
```
butt(r = c(1.2, 1, 0.85, 0.55, 0.5, 0.1, 0.05), legend = "Avec r[3] = 0.85")
```



6.1.4 `r[4]`

Le paramètre `r[4]` contrôle le rayon externe de l'anneau des triples. Par exemple :

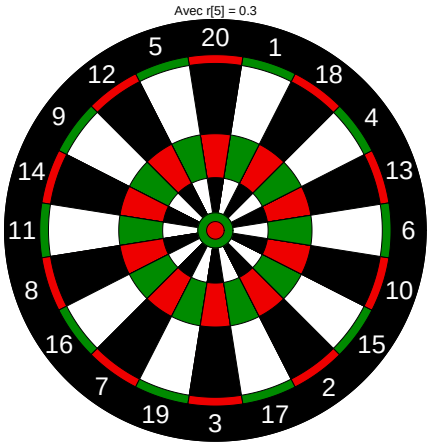
```
butt(r = c(1.2, 1, 0.95, 0.75, 0.5, 0.1, 0.05), legend = "Avec r[4] = 0.75")
```



6.1.5 `r[5]`

Le paramètre `r[5]` contrôle le rayon interne de l'anneau des triples. Par exemple :

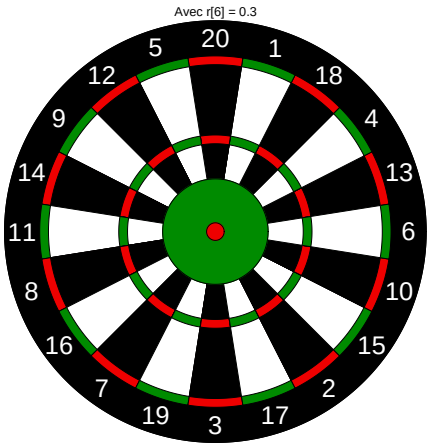
```
butt(r = c(1.2, 1, 0.95, 0.55, 0.3, 0.1, 0.05), legend = "Avec r[5] = 0.3")
```



6.1.6 `r[6]`

Le paramètre `r[6]` contrôle le rayon du simple mil. Par exemple :

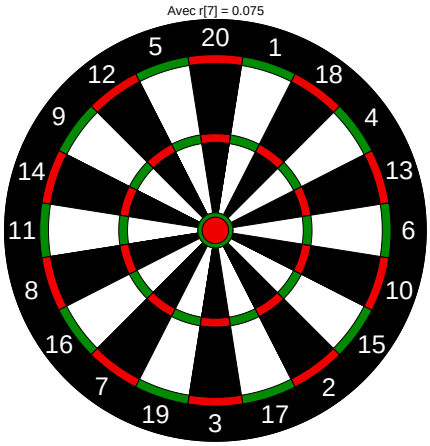
```
butt(r = c(1.2, 1, 0.95, 0.55, 0.5, 0.3, 0.05), legend = "Avec r[6] = 0.3")
```



6.1.7 `r[7]`

Le paramètre `r[7]` contrôle le rayon du double mil. Par exemple :

```
butt(r = c(1.2, 1, 0.95, 0.55, 0.5, 0.1, 0.075), legend = "Avec r[7] = 0.075")
```

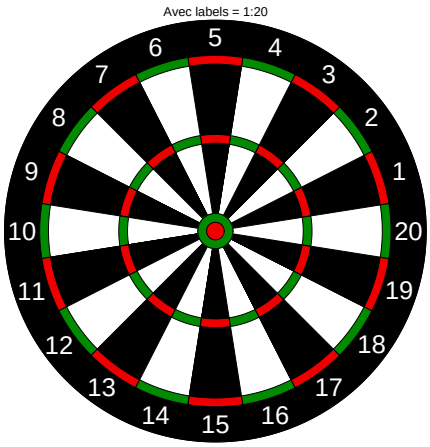


6.2 Les étiquettes du vecteur de paramètres `labels`

La numérotation des secteurs est visiblement faite pour maximiser la variance des points : le secteur 20 est à coté du secteur 1 par exemple. L'origine de cette nuémrotation n'est pas très claire. D'après Eiselt et Laporte [2] qui s'appuyent sur la version de 1990 du *Guinness Book of Records*, elle serait due à un dénomé Brian Gamlin qui aurait fixé les règles en 1896. La disposition des secteurs n'est pas optimale pour maximiser la variance, mais néamoins proche de l'optimum [6, 4, 1].

La numérotation des secteurs est controlée par l'argument `labels` dont voici l'ordre trigonométrique naturel :

```
butt(labels = 1:20, legend = "Avec labels = 1:20")
```



Notez le décalage de $\frac{\pi}{20}$ pour que le secteur du haut soit centré autour de la ligne des ordonnées.

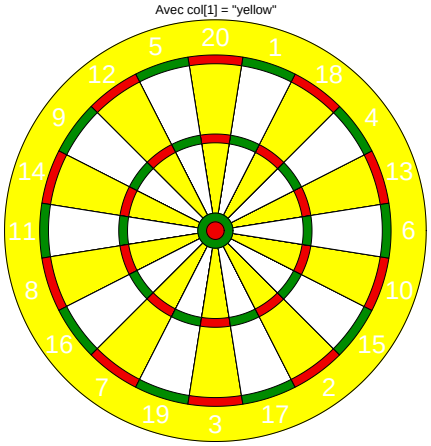
6.3 Les valeurs du vecteur de paramètres `col`

Je n'ai pas trouvé de norme officielle pour les couleurs de la cible. Elles sont modifiables avec l'argument `col` qui est un vecteur de 4 couleurs. Dans les exemples suivants la valeur modifiée est en jaune.

6.3.1 `col[1]`

Le paramètre `col[1]` contrôle la couleur de fond. Par exemple :

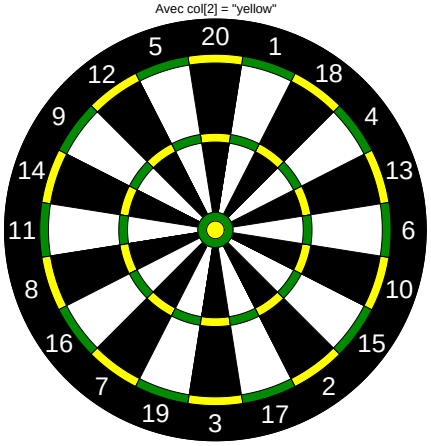
```
butt(col = c("yellow", "red2", "green4", "white", "white", "black"))
mtext("Avec col[1] = \"yellow\"", line = -1, outer = TRUE)
```



6.3.2 `col[2]`

Le paramètre `col[2]` contrôle la couleur du double mil et des anneaux double et triple du secteur 20 modulo 2. Par exemple :

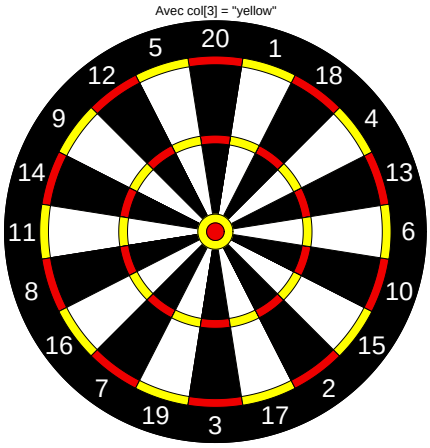
```
butt(col = c("black", "yellow", "green4", "white", "white", "black"))
mtext("Avec col[2] = \"yellow\"", line = -1, outer = TRUE)
```



6.3.3 `col[3]`

Le paramètre `col[3]` contrôle la couleur du simple mil et des anneaux double du secteur 1 modulo 2. Par exemple :

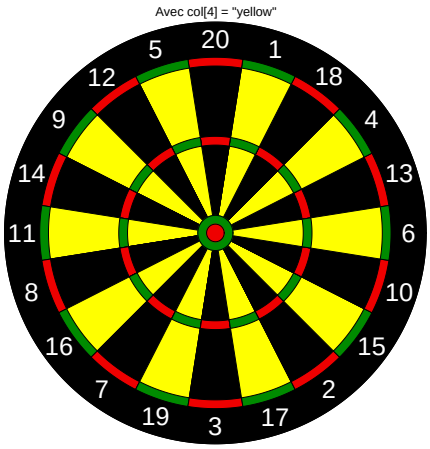
```
butt(col = c("black", "red2", "yellow", "white", "white", "black"))
mtext("Avec col[3] = \"yellow\"", line = -1, outer = TRUE)
```



6.3.4 `col[4]`

Le paramètre `col[4]` contrôle la couleur des anneaux simples, intérieurs et extérieurs, de secteur 1 modulo 2. Par exemple :

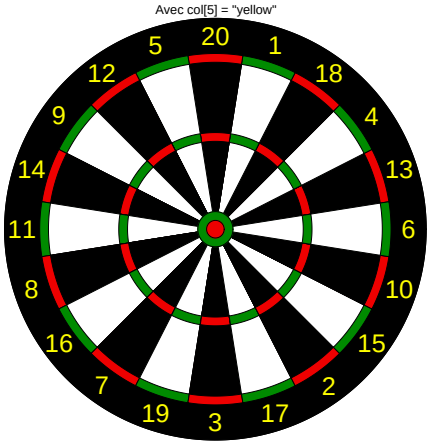
```
butt(col = c("black", "red2", "green4", "yellow", "white", "black"))
mtext("Avec col[4] = \"yellow\"", line = -1, outer = TRUE)
```



6.3.5 `col[5]`

Le paramètre `col[5]` contrôle la couleur des labels. Par exemple :

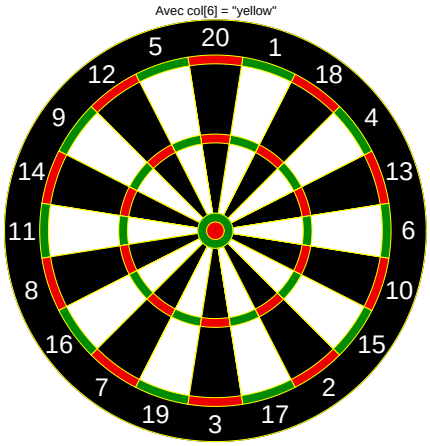
```
butt(col = c("black", "red2", "green4", "white", "yellow", "black"),
    legend = "Avec col[5] = \"yellow\"")
```



6.3.6 `col[6]`

Le paramètre `col[6]` contrôle la bordure des secteurs. Par exemple :

```
butt(col = c("black", "red2", "green4", "white", "white", "yellow"),
    legend = "Avec col[6] = \"yellow\"")
```



Références

[1] G.L. Cohen. Dartboard arrangements. *The Electronic Journal of Combinatorics*, 8 :R4, 2001.

[2] H.A. Eiselt and G. Laporte. A combinatorial optimization problem arising in dartboard design. *The Journal of the Operational Research Society*, 42 :113–118, 1991.

[3] D. Kohler. Optimal strategies for the game of darts. *The Journal of the Operational Research Society*, 33 :871–884, 1982.

[4] M.A.M. Lynch. Designing a dartboard - an application of graph modelling. *Teaching Mathematics and its Applications*, 16 :51–54, 1997.

[5] G. Pif. Sideral glasses to increase the accuracy of bull's-eyes fit method in non-linear regression. *Pif Gadjet*, 1 :1–48, 1969.

[6] K. Selkirk. Re-designing the dartboard. *The Mathematical Gazette*, 60 :171–178, 1976.