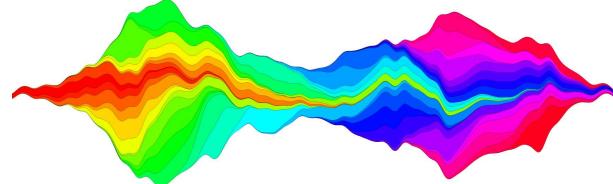


### 3. Introduction à l'étude de données avec

graphiques, paramètres de description statistiques, quelques tests

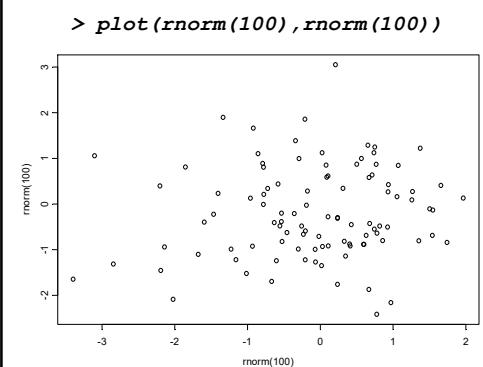
73

#### 3.1 Les graphiques dans



74

#### La fenêtre graphique et la fonction `plot()`



La fonction `rnorm()` simule des variables aléatoires normalement distribuées.

Consultez `?rnorm`, mais aussi `?runif`, `?rexp`, `?binom`, ...

75

#### Flexibilité de la fonction `plot()`

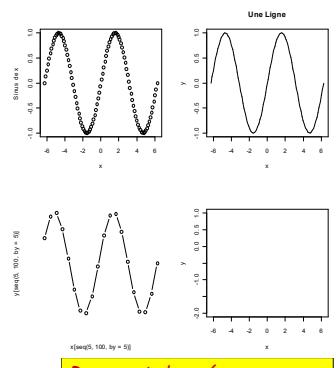
```
x <- seq(-2*pi, 2*pi, length=100)
y <- sin(x)

par(mfrow=c(2,2))
plot(x,y,xlab="x",
      ylab="Sinus de x")

plot(x,y,type= "l",
      main="Une Ligne")

plot(x[seq(5,100,by=5)],
      y[seq(5,100,by=5)],
      type= "b",axes=F)

plot(x,y,type="n",
      ylim=c(-2,1)
      par(mfrow=c(1,1))
```



R permet de créer une multitude de graphique !

76

## D'autres fonctions à utiliser après `plot()`

```
axis(1,at=c(2,4,5) #Détails d'un axe ("ticks", légende, ...)
legend("A","B","C")
lines(x,y,...) #Pour ajouter des lignes
abline(h=0,v=c(2,3)) #Ajouter une horizontale à 0 et deux verticales à 2 et 3
abline(lsfit(x,y)) #Ajoute une droite de régression
abline(0,1) #Ajoute une ligne de pente 1 et "intercept" 0
```

*Graphiques R très personnalisables !  
De très nombreux tuto sur internet :*

*Exemples :*

<http://www.duclert.org/Aide-memoire-R/Graphiques/Parametres-des-graphes.php>  
<http://www.harding.edu/fmccown/r/>

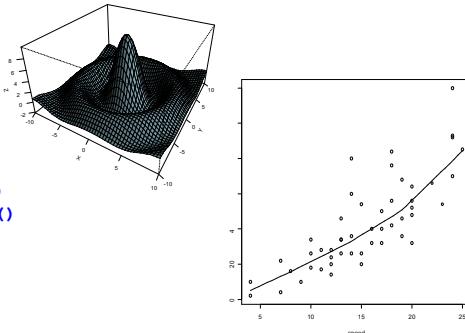
77

## Autres fonctions graphiques

Voir les aides "on-line" des fonctions suivantes pour des exemples:

```
barplot()
image()
hist()
pairs()
persp()
piechart()
polygon()

library(modreg)
scatter.smooth()
```



78

## Graphiques : gestion des paramètres

- Sauver les paramètres avant de les modifier : "touche panique"
- Stocker tous les paramètres par défaut  
`old.par <- par(no.readonly = TRUE)`
- Dessiner un graphique en modifiant les paramètres  
`par(bg="aliceblue", col="red")`  
`plot(.....)`
- Initialiser les paramètres aux valeurs par défaut  
`par(old.par)`

→ De nombreux exemples de graphiques sont visibles à cette adresse : <http://addictedor.free.fr/graphiques/>

79

## Graphiques : exercice

- Les données

```
Bodywt<-c( 10, 207, 62, 6.8, 52.2) ; Brainwt<-c(115, 406, 1320, 179, 440)
Noms<-c("Potar monkey", "Gorilla", "Human", "Rhesus Monkey", "Chimp")
```

- Couleur de fond du graphique

```
par(bg="aliceblue", col="red")
```

- Le titre et les labels

```
titre<-"Poids du cerveau / poids du corps"
```

```
labelX<-"Poids du corps"
```

```
labelY<-"Poids du cerveau"
```

- Le graphique (pch=type de point)

```
plot(Bodywt, Brainwt, xlim=c(5,250), main=titre, xlab=labelX, ylab=labelY,
      pch=16)
```

```
text(Bodywt, Brainwt, labels=Noms, adj=0)
```

- Les couleurs disponibles : `colors()`

**exercice** : Comment rendre ce graphique le plus laid possible ?

Modifiez les couleurs, les tailles de caractères, ...

80

## Découper la fenêtre graphique

- La fenêtre graphique en 4 parties  
`par(mfcol=c(2,2))`

1	3
2	4

- La fenêtre graphique en 6 parties  
`par(mfcol=c(3,2))`

1	4
2	5
3	6

- La fenêtre graphique en 6 parties  
`par(mfcol=c(2,3))`

1	3	5
2	4	6

81

## Découper la fenêtre graphique : un exemple

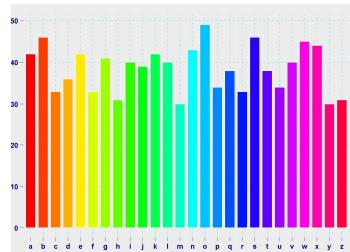
- Description de la variable rendement

- voir page 49 pour importer les données `bledur`
- taper en comprenant les lignes de codes suivantes (p64 à 76)

```
par(mfcol=c(2,2))
boxplot(bledur$RDT)
hist(bledur$RDT)
plot(bledur$RDT,bledur$PLM)
pie(summary(as.factor(bledur$VRT)))
par(mfcol=c(1,1))
```

82

## 3.2. Décrire des variables



83

## Rapide inspection d'un fichier

```
bledur<-read.table("~/bledur.txt", header=T, na.string="M", dec=",", sep=" ")
dim(bledur) ; dimnames(bledur)
# structure
str(bledur)

# n premières lignes
head(bledur, n=5)

# n dernières lignes
tail(bledur, n=5)

# retrouver des elts d'un tableau
bledur[1,1] ; bledur[9,6] ; bledur[,1] ; bledur[1,]
bledur[1:5,4:6] ; bledur[c(1,3),c(3,5)]

# exclure des éléments
bledur[-1,] ; bledur[,-3]
```

84

## Décrire des variables quantitatives

```
summary(bledur[, c(2,3,5,6,7,9,10,11)])
maliste<-c(2,3,5,6,7,9,10,11) # liste des données quantitatives
summary(bledur[, maliste])
• Un paramètre statistique pour chaque niveau d'une variable qualitative :
aggregate(bledur[,maliste], list("variete"=bledur$VRTC), mean)
aggregate(bledur[,maliste], list("variete"=bledur$VRTC, "zone"=bledur$ZON), mean)
variete zone RDT PLM ARG LIM SAB PGM MST AZP
1 1 1 11.40 103.40 38.52 41.50 19.98 38.00 35.82 3.36
2 2 1 8.45 128.40 27.69 52.46 20.85 39.52 28.64 3.45
3 3 1 13.66 87.00 27.05 53.30 19.65 42.65 35.36 3.16
4 1 2 9.93 101.00 21.22 32.96 45.82 39.87 31.63 3.18
5 2 2 12.76 139.50 24.25 33.20 42.55 32.00 37.97 3.66
6 3 2 13.07 117.50 20.95 39.60 39.45 43.80 31.17 2.91
7 1 3 11.29 137.25 17.10 18.54 64.36 40.58 33.99 3.27
8 2 3 8.61 160.78 20.44 29.43 50.12 30.77 28.42 2.87
9 3 3 15.10 117.00 31.10 47.70 21.20 30.70 41.30 3.84
• On peut calculer : mean, sd, max, min, median, sum, ...

```

85

## Extraire une partie d'un data frame avec subset

```
subset(bledur, variete==1 & zone== 1)
Binary comparison operators
```

operator	description	usage
<	less than	x<y
>	greater than	x>y
<=	less than or equal	x<=y
>=	greater than or equal	x>=y
!=	different	x != y
&	AND (logical)	A & B
	OR (logical)	A   B
!	NOT (logical)	!A

86

## Autres manipulations :

```
split(); transform() ; with() ... cbind()

# splitting rows by variete
split(bledur, variete)
# transforming an existing column
bledur.variete<-transform(bledur,variete=as.factor(variete))

A useful solution to express conditions on data frame columns is to use
  with() function : with(data frame, expression to evaluate)
# frequency of variety
table(bledur$variety)

# same table using with()
with(bledur,table(variety))

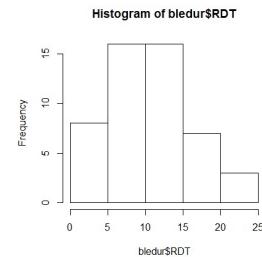
# plotting SAB PGM
plot(bledur$SAB,bledur$PGM)
#same plot using with()
with(bledur,plot(SAB,PGM))
```

87

## Les histogrammes (1)

### Les données quantitatives :

```
hist(bledur$RDT)
• On peut choisir le nombre de classes
hist(bledur$RDT, nclass=5)
• Pour avoir les bornes et les effectifs des classes
hist(bledur$RDT, nclass=5, plot=F)
• Choisir ce dont on a besoin :
a<- hist(bledur$RDT, nclass=5, plot=F)
names(a)
bornes<-a$breaks
[1] 0 5 10 15 20 25
```



88

## Les histogrammes (2)

- Attention, quelle différence entre ?

```
attach(bledur)
```

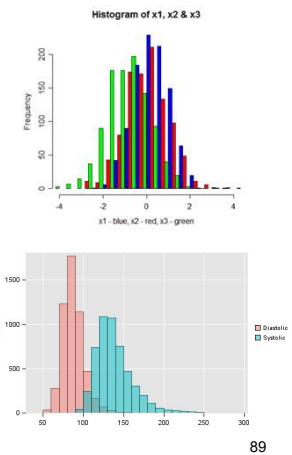
```
hist(VRT)
```

```
barplot(VRT)
```

```
barplot(summary(VRT))
```

```
barplot(summary(as.factor(VRT)))
```

```
detach(bledur)
```



89

## Les boîtes à moustaches

- Une boîte simple

```
boxplot(bledur$RDT)
```

- Une boîte par niveau de facteur

```
boxplot(split(bledur$RDT,bledur$ZON))
```

ou

```
boxplot(bledur$RDT~bledur$ZON) # résultat identique
```

- Pour rendre le graphique plus lisible, ajouter un titre

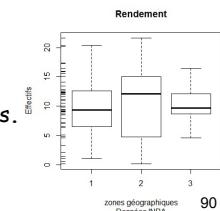
```
title("Rendement", sub="Données INRA", ylab= "Effectifs",
      xlab= "zones géographiques")
```

- Représenter les individus

```
rug(bledur$RDT, side=2)
```

- Exporter le graphique sous différents formats.

Menus File/Save as/...



90

## Les camemberts

- Les données doivent être préparées !

```
pie(bledur$VRT) # mauvais
```

```
pie(summary(as.factor(bledur$VRT))) # correct
```

- Attention à l'ordre d'apparition des niveaux de facteur

```
summary(as.factor(bledur$VRT))
```

- On ajoute un nom "explicite" pour chaque secteur

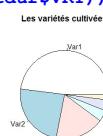
```
nom<-c("Var1","Var2","Var3","Var4","Var5","Var6")
```

```
pie(summary(as.factor(bledur$VRT)), label=nom)
```

- On ajoute un titre principal

```
titre<-"Les variétés cultivées"
```

```
pie(summary(as.factor(bledur$VRT)), label=nom, main=titre)
```



91

## Les nuages de points

- Simple graphique

```
plot(bledur$PLM,bledur$RDT)
```

- Graphique "commenté"

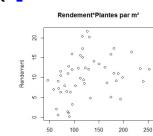
```
titre<-"Rendement*Plantes par m²"
x<-"Nombre de plants par m²" ; y<-"Rendement"
plot(bledur$PLM,bledur$RDT, main=titre, xlab=x,ylab=y)
```

- Graphique illustré

```
plot(bledur$PLM,bledur$RDT)
```

```
text(bledur$PLM,bledur$RDT,labels=bledur$ZON)
```

**Exercice :** les points et les numéros se superposent. Comment rendre les numéros plus lisibles ? ([?plot](#) et [?text](#))



92

## Graphique de toutes les variables quantitatives

- Vérifier la linéarité des relations entre variables avant de lancer une Analyse en Composantes Principales (ACP)

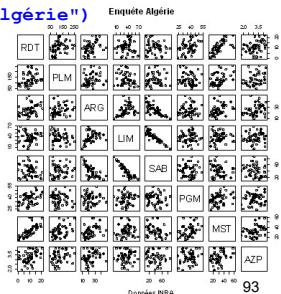
```
maliste<-c(2,3,5,6,7,9,10,11) # les variables quantitatives
pairs(bledur[,maliste])
title("Enquête Algérie ", sub="Données INRA")
pairs(bledur[,maliste], main="Enquête Algérie")
title(sub="Données INRA")
```

Graphe à améliorer !

Voir la library **ggairis**,  
**ggplot2** par ex pour de plus beaux graphes  
[www.statmethods.net/advgraphs/ggplot2.html](http://www.statmethods.net/advgraphs/ggplot2.html)

Ou

dans la library **psych** la fonction **pairs.panels**



## Les corrélations

- Choisir les données quantitatives

```
maliste<-c(2,3,5,6,7,9,10,11)
```

- Un tableau des corrélations

```
cor(bledur[,maliste])
```

Pour aller plus loin dans la description des données voir:

[AS-02-Outils-LB&RT-2p.pdf](#)

94

## Les fonctions graphiques avec ggplot2

Le package **ggplot2** :

- outil puissant de visualisation des données ; graphes mieux finalisés qu'avec les fonctions classiques sous R
- yntaxe spécifique et totalement différente de celles des graphes conventionnels sous R (de type « Grammar of Graphics »).
- voir l'antisèche de **ggplot2** en consultant sa « cheat sheet » dans l'onglet **Help** de **Rstudio**

95

## Les fonctions graphiques avec ggplot2

Premiers graphes avec **ggplot2**

Considérons le jeu de données **diamonds** prix et caractéristiques de 54000 diamants et travaillons sur un sous-échantillon de taille 5000 pour alléger les représentations graphiques:

```
> library(ggplot2)
> set.seed(1234)
> diamonds2<-diamonds[sample(nrow(diamonds),5000),]
> ggplot(diamonds2)+aes(x=cut)+geom_bar()
```

L'approche **ggplot2** consiste à segmenter les instructions. Pour un diagramme en barres, il faut spécifier :

- Le jeu de données : **diamonds2** => renseigné dans **ggplot**
- La variable à représenter : **cut** => renseigné dans **aes**
- Le type de graphe souhaité : **diagramme en barres** => renseigné dans **geom\_bar()**

```
> ggplot(diamonds2)+aes(x=price)+geom_histogram()
```

96

## Les fonctions graphiques avec ggplot2

### Premiers graphes avec ggplot2

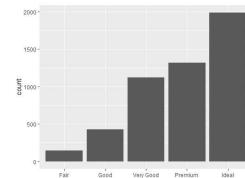
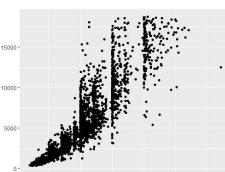


Diagramme en barres pour `cut` (gauche) et histogramme des effectifs pour `price` (droite)



```
> ggplot(diamonds2)+aes(x=carat,y=price)
+geom_point()
```

Nuage de points : prix d'un diamant en fonction de son nombre de carats

97

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot

Construction d'une représentation graphique `ggplot` à partir d'un ensemble d'éléments constituant la **grammaire de la syntaxe** :

- Data (`ggplot`) : jeu de données contenant les variables utilisées
- Aesthetics (`aes`) : variables à représenter
- Geometrics (`geom_...`) : type de représentation graphique souhaitée
- Statistics (`stat_...`) : les éventuelles transformations des données pour la représentation souhaitée
- Scales (`scale_...`) : contrôle du lien entre données et aesthetics (couleurs, gestion des axes, ...)

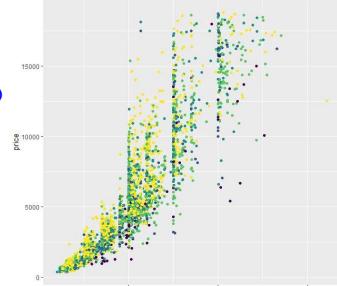
98

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : data et aesthetics

- Le jeu de données (data-frame ou data-table) est spécifié dans `ggplot`
- Les variables sont spécifiées dans la fonction `aes` - arguments `color`, `size`, `fill` pour définir couleur, taille à partir d'une variable du fichier

```
> ggplot(diamonds2)
+aes(x=carat,y=price, color=cut)
+geom_point()
```



99

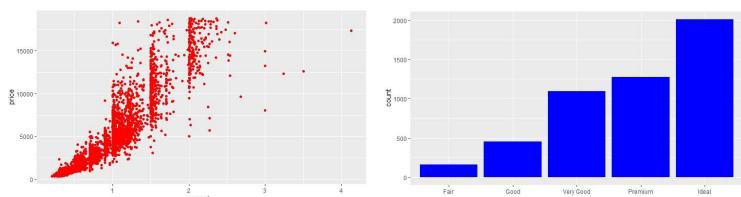
## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : geometrics

- la commande `ggplot(diamonds2)+aes(x=carat,y=price, color=cut)` ne renvoie pas de graphe !
- Les fonctions `geom_...` précisent le type de representation souhaitée. Elles admettent également des arguments permettant de modifier le graphe (couleur, taille de pts, épaisseur de traits, etc)

```
> ggplot(diamonds2)+aes(x=carat,y=price)+geom_point(color="red")
```

```
> ggplot(diamonds2)+aes(x=cut)+geom_bar(fill="blue")
```



100

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : geometrics

- Exemples de représentations :

Geom	Description	Aesthetics
geom_point()	Nuage de points	x, y, shape, fill
geom_line()	Ligne (ordonnées selon x)	x, y, linetype
geom_abline()	Droite	slope, intercept
geom_path()	Ligne (ordre original)	x, y, linetype
geom_text()	Texte	x, y, label, hjust, vjust
geom_rect()	Rectangle	xmin, xmax, ymin, ymax
geom_polygon()	Polygone	fill, linetype
geom_segment()	Segment	x, y, fill, linetype
geom_bar()	Diagramme en barres	x, fill, linetype, weight
geom_histogram()	Histogramme	x, fill, linetype, weight
geom_boxplot()	Boîtes à moustaches	x, y, fill, weight
geom_boxplot()	Boîtes à moustaches	x, y, fill, weight
geom_density()	Densité	x, y, fill, linetype
geom_contour()	Lignes de contour	x, y, fill, linetype
geom_smooth()	Lissage	x, y, fill, linetype
Tous		color, size, group

Table tirée de F. Huson (dir.) (2018, p77)

101

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : statistics

- Si la transformation est fonctionnelle, la spécifier dans **aes**

```
> D<- data.frame (X=seq(-2*pi,2*pi,by=0.01))
> ggplot(D)+aes (x=X,y=sin(X))+geom_line()
```



- Transformations plus complexes : l'argument **stat** dans **geom\_...** permet de gérer les graphes:

- Exemple : **geom\_histogram** : valeur par défaut **stat="bin"** permet de calculer 4 indicateurs. **count** nb de pts par classe ; **density** densité pour chaque classe; **ncount** équivaut à **count** avec valeur max à 1 ; **ndensity** équivaut à **density** avec valeur max à 1.
- Par défaut, **geom\_histogram** représente le 1<sup>er</sup> de ces indicateurs
- Si on souhaite représenter un autre indicateur préciser son nom dans **aes** en utilisant le format **.nom..**

```
ggplot(diamonds2)+aes (x=price)+geom_histogram(bins=40)
ggplot(diamonds2)+aes (x=price,y=..count..)+geom_histogram(bins=40) # idem
ggplot(diamonds2)+aes (x=price,y=..density..)+geom_histogram(bins=4)
```

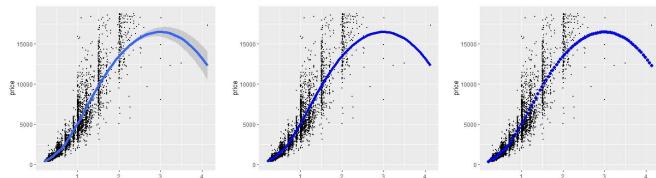
102

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : statistics

- Alternative représentations faisant intervenir des transformations des données, les fonctions : **stat\_...**

```
> ggplot(diamonds2)+aes (x=price,y=..count..)+stat_bin(bins=40) # idem
> ggplot(diamonds2)+aes (x=price,y=..density..)+stat_bin(bins=4)
  - stat_... Possède un argument geom qui change la représentation graphique
> ggplot(diamonds2)+aes (x=carat,y=price)+geom_point(size=0.5)
  +stat_smooth(method = "loess",size=2)
> ggplot(diamonds2)+aes (x=carat,y=price)+geom_point(size=0.5)
  +stat_smooth(method = "loess",geom="line",color="blue",size=2)
> ggplot(diamonds2)+aes (x=carat,y=price)+geom_point(size=0.5)
  +stat_smooth(method = "loess", geom="point",color="blue",size=2)
```



Lissage avec **stat\_smooth**

103

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : statistics

- Exemples de statistics

Stat	Description	Paramètres
stat_identity()	Aucune transformation	
stat_bin()	Comptage	binwidth, origin
stat_density()	Densité	adjust, kernel
stat_smooth()	Lissage	method, se
stat_boxplot()	Boxplot	coef

Table tirée de F. Huson (dir.) (2018, p80)

104

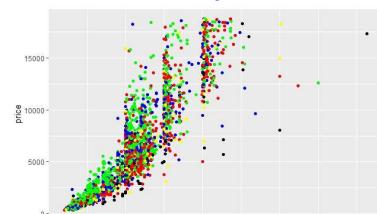
## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

#### Les fonctions `scale_...`

- contiennent tous les paramètres qui font le lien entre données et aesthetics
- permettent d'affiner le graphe
- commencent par `scale_...`; suivi du nom de l'aesthetics à modifier (`color_fill_x_...`) ; se terminent par le nom du scale : `manual`, `identity`, ...
- Exemple :

```
> ggplot(diamonds2)+aes(x=carat,y=price, color=cut)
+geom_point() +scale_colour_manual(values = c("Fair"="black", "Good"="yellow", "Very
Good"="blue", "Premium"="Red", "Ideal"="green"))
```



Changement des couleurs avec `scale_colour_manual`

105

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

#### • Principaux types de scales

Position	Taille	Forme	Couleur
<code>x</code> (ou <code>y</code> )			<code>colour</code> (ou <code>fill</code> )
<code>_continuous</code>	<code>_size_continuous</code>		<code>_colour_gradient</code>
<code>_date</code>	<code>_area</code>		<code>_colour_gradient2</code>
<code>_datetime</code>			<code>_colour_gradients</code>
<code>_discrete</code>	<code>_size_discrete</code>	<code>_shape_discrete</code>	<code>_colour_hex</code>
	<code>_size_identity</code>	<code>_shape_identity</code>	<code>_colour_brewer</code>
	<code>_size_manual</code>	<code>_shape_manual</code>	<code>_colour_grey</code>
			<code>_colour_identity</code>
			<code>_colour_manual</code>

Table tirée de F. Huson (dir.) (2018, p81)

106

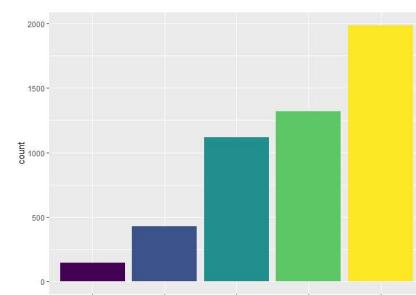
## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

#### • Exemples d'utilisation : couleurs d'un diagramme en barre

Barplot de `cut` en utilisant une couleur différente pour chaque barre

```
> p1<-ggplot(diamonds2)+aes(x = cut)+geom_bar(aes(fill=cut)) ; p1
```



107

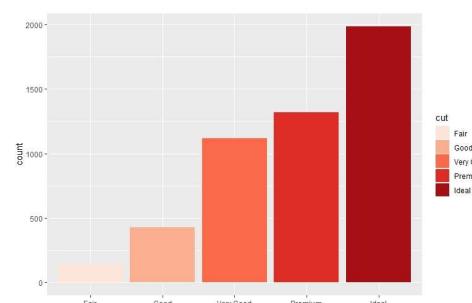
## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

#### • Exemples d'utilisation : couleurs d'un diagramme en barre

Barplot de `cut` en utilisant une couleur différente pour chaque barre avec une palette de couleurs prédéfinie

```
> p1+scale_fill_brewer(palette="Reds")
```



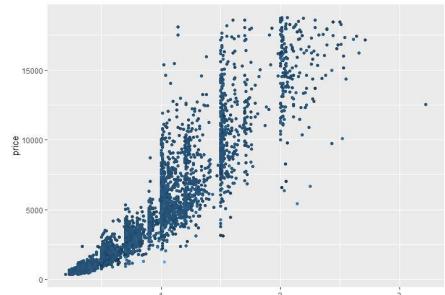
108

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

- Exemples d'utilisation :** dégradé de couleurs pour un nuage de points  
Représentation du nuage de points `carat*price` avec une échelle de couleur définie par la variable continue `depth`

```
> p2<-ggplot(diamonds2)+aes(x=carat,y=price)+geom_point(aes(color=depth)) ; p2
```



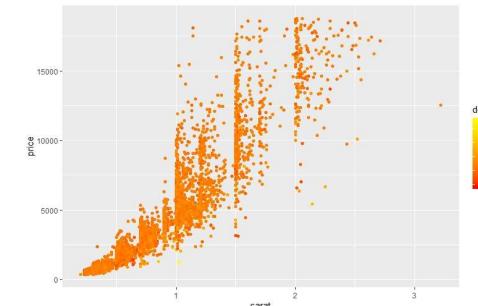
109

## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

- Exemples d'utilisation :** dégradé de couleurs pour un nuage de points  
Représentation du nuage de points `carat*price` avec une échelle de couleur définie par la variable continue `depth`, en définissant un nouveau dégradé

```
> p2+scale_color_gradient(low="red",high="yellow")
```



110

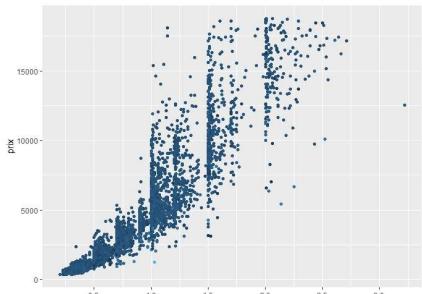
## Les fonctions graphiques avec ggplot2

### La grammaire ggplot : scales

- Exemples d'utilisation :** gestion des axes et des légendes

Modification de la graduation de l'axe des abscisses, du nom de l'axe des ordonnées et celui de la légende du graphe `p2`

```
> p2+scale_x_continuous(breaks = seq(0.5,3,by=0.5))+  
  scale_y_continuous(name="prix")+scale_color_gradient("Profondeur")
```



111

## Les fonctions graphiques avec ggplot2

### Group et facets

Avec le package `ggplot2`, deux façons de représenter des sous-groupes d'individus caractérisés par une ou plusieurs variables :

- représenter les sous-groupes sur un même graphe : utiliser l'argument `group` dans la fonction `aes`
- représenter les sous-groupes sur des graphes différents : utiliser les fonctions `facet_grid` et `facet_wrap`

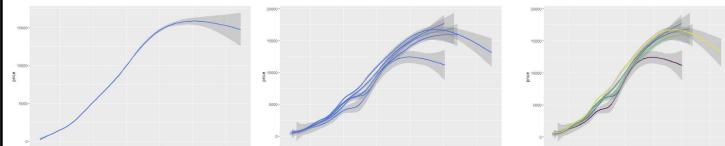
Exemple d'un lissage du nuage de points `carat*price` avec la fonction `geom_smooth`

112

## Les fonctions graphiques avec ggplot2

**Group :** sous-groupes sur un même graphe

```
> ggplot(diamonds2)+aes(x=carat,y=price)+geom_smooth()
> ggplot(diamonds2)+aes(x=carat,y=price,group=cut)+geom_smooth()
> ggplot(diamonds2)+aes(x=carat,y=price,group=cut,color=cut)+geom_smooth()
```



Lissage en fonction de la variable *cut*

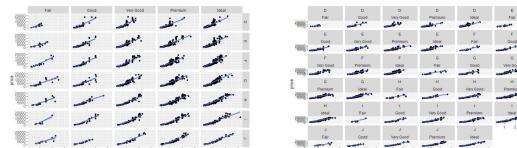
113

## Les fonctions graphiques avec ggplot2

**Facets :** sous-groupes sur des graphes différents

Les fonctions `facet_grid` et `facet_wrap` vont pour argument une formule de la forme `var1~var2`. Elles renvoient la représentation calculée sur les indiv appartenant à chaque croisement des modalités de `var1` et `var2`.

```
> ggplot(diamonds2)+aes(x=carat,y=price)+geom_point()
  +geom_smooth(method = "lm")+facet_grid(color~cut)
> ggplot(diamonds2)+aes(x=carat,y=price)+geom_point()
  +geom_smooth(method = "lm")+facet_wrap(color~cut)
```



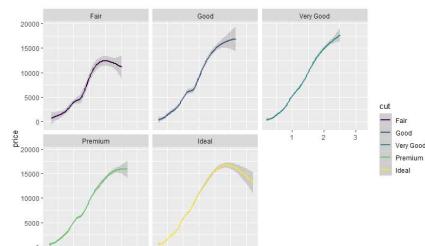
Nuages de points *price* vs *carat* et droite des moindres carrés en fonction des variables *cut* et *colour*

114

## Les fonctions graphiques avec ggplot2

**Facets :** sous-groupes sur des graphes différents

- `facet_grid` : présentation plus cohérente qd on est présence de 2 variables avec peu de modalités
- `facet_wrap` : qd on est en présence d'une variable ou qu'une variable admet bcp de modalités



Lissage en fonction de la variable *cut*

115

## Les fonctions graphiques avec ggplot2

**Compléments:** exportation

```
> monplot<-qplot(data=diamonds2,x=carat,y=price,
  geom=c("point", "smooth"),facets=color~cut)
> ggsave(mon_graph.pdf,plot=monplot,width=11,height=8)
```

116

## Les fonctions graphiques avec ggplot2

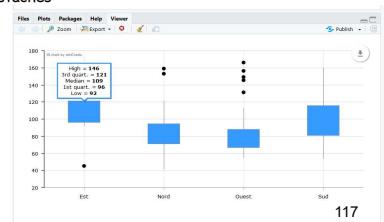
### Graphes interactifs

- le 1<sup>er</sup> arrivé : package `htmlwidgets`, voir la page <http://htmlwidgets.org>  
Les graphes s'intègrent facilement dans des documents RMarkdown ou dans les applications shiny et s'exportent en divers formats

- Exemple du package `rAmCharts` :

- reprend les principales fonctionnalités graphiques de base de R
- nom de la fonction « `am` » suivi du nom de la fonction R: `amBoxplot, amBorplot, amHist`
- Ex : Construction de boîtes à moustaches

```
> library(rAmCharts)
> ozone=read.table("ozone.txt",h=T)
> amBoxplot(maxO3~vent,
            data=ozone,export=TRUE)
```



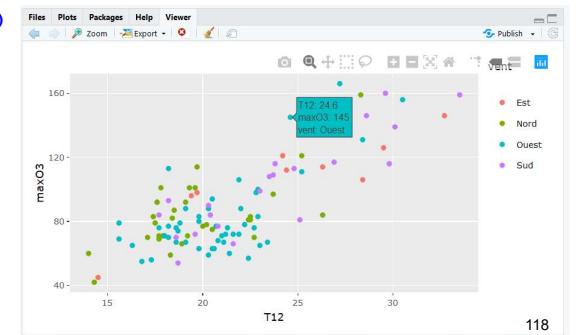
117

## Les fonctions graphiques avec ggplot2

### Graphes interactifs

Avec le package `ggplotly` () ; si on construit des graphes avec `ggplot2`

```
> ggplot(ozone) + aes(x=T12,y=maxO3,color=vent) +
  geom_point()
> library(plotly)
> ggplotly()
```



118

## Les fonctions graphiques avec ggplot2

### exercice : avec ggplot2

Considérer le jeu de données `mtcars` du package `datasets`

- Tracer l'histogramme de la variable `mpg`
- Tracer le diagramme en barres de la variable `cyl`
- Représenter le nuage de points `dispxmpg` en fonction des valeurs de la variable `cyl`. On représentera un nuage de points pour chaque valeur de `cyl`

119

## 4. Calcul matriciel sous R : pour aller plus loin

bases, opérations,  
décompositions

Gaston Sanchez ([gastonsanchez.com](http://gastonsanchez.com))

120

## Matrices sous R

### Basic functions in R for matrix objects

Function	Description
<code>matrix()</code>	create a matrix
<code>dim()</code>	dimension of a matrix
<code>nrow()</code>	number of rows
<code>ncol()</code>	number of columns
<code>as.matrix()</code>	convert into matrix
<code>is.matrix()</code>	test if the argument is a matrix

Bear in mind that R **can do some things that matrix algebra cannot**: row-column naming, handling **NA**'s, and recycling.

121

## Matrix Recap

```
# matrix
A = matrix(1:12, nrow=4, ncol=3)

# add row names
rownames(A) = c("a", "b", "c", "d")

# add column names
colnames(A) = c("one", "two", "three")

A

##   one two three
## a  1   5   9
## b  2   6   10
## c  3   7   11
## d  4   8   12

# test class
is.matrix(A)

## [1] TRUE
```

```
# dimensions
dim(A)

## [1] 4 3

# number of rows
nrow(A)

## [1] 4

# number of columns
ncol(A)

## [1] 3

# first row
A[1,]

##   one two three
## 1   5   9
```

122

## Matrix Recap (con't)

*con't: abbreviation for "continued"*

### Recycling a vector into matrix

```
# vector
b = 1:3

# dimension
dim(b)

## NULL

# test class matrix?
is.matrix(b)

## [1] FALSE

# recycling
(B = matrix(b, nrow=4, ncol=3))

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    3    1
## [3,]    3    1    2
## [4,]
```

### Missing values

```
# test class matrix?
is.matrix(B)

## [1] TRUE

# missing values
B[1, 1] = NA
B[4, 3] = NA
B

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    3    1
## [3,]    3    1    2
## [4,]
```

123

## Matrix Transpose

The transpose of a  $n \times p$  matrix  $\mathbf{X}$  is the  $p \times n$  matrix  $\mathbf{X}'$ . In R the transpose is given by the function `t()`.

```
# matrix X
X = matrix(1:12, 4, 3)
X

##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]
```

```
# transpose of X
t(X)

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]
```

124

## From scalar to matrix

Scalar

Scalar into  $1 \times 1$  matrix

```
# scalar
x = 1

# dim
dim(x)
## NULL

# test if matrix
is.matrix(x)
## [1] FALSE

# convert to matrix
xx = matrix(x, 1, 1)
xx
##      [,1]
## [1,]    1

dim(xx)
## [1] 1 1
```

125

## Rectangular matrix

The general *shape* of a matrix is a **rectangular**  $n \times p$  matrix  
( $n$  number of rows,  $p$  number of columns)

```
# rectangular matrix
Rectangular = matrix(runif(15), nrow = 3, ncol = 5)
Rectangular
##      [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 0.2655  0.9082  0.9447  0.06179  0.6870
## [2,] 0.3721  0.2017  0.6608  0.20597  0.3841
## [3,] 0.5729  0.8984  0.6291  0.17656  0.7698

# dimensions
dim(Rectangular)
## [1] 3 5
```

126

## Square matrix

A matrix  $\mathbf{X}$  is **square** if the number of rows is equal to the number of columns (i.e.  $n = p$ )

```
# square matrix
Square = matrix(runif(9), nrow = 3, ncol = 3)
Square
##      [,1]     [,2]     [,3]
## [1,] 0.585800 0.2774  0.7244
## [2,] 0.008946 0.8136  0.9061
## [3,] 0.293740 0.2604  0.9490

# same number of rows and columns
dim(Square)
## [1] 3 3
```

127

## Symmetric matrix

A square matrix  $\mathbf{X}$  is **symmetric** if  $x_{ij} = x_{ji}$  for all  $i$  and  $j$ , that is if  $\mathbf{X} = \mathbf{X}'$ . A square matrix is **asymmetric** if it is not symmetric.

```
# square matrix
Symmetric = matrix(c(1, 2, 3, 2, 1, 4, 3, 4, 1), 3, 3)
Symmetric
##      [,1]     [,2]     [,3]
## [1,]     1      2      3
## [2,]     2      1      4
## [3,]     3      4      1

# transpose
t(Symmetric)

##      [,1]     [,2]     [,3]
## [1,]     1      2      3
## [2,]     2      1      4
## [3,]     3      4      1
```

128

## Diagonal matrix

A square matrix  $\mathbf{X}$  is **diagonal** if  $x_{ij} = 0$  for all  $i \neq j$ . Thus a diagonal matrix is symmetric.

In R we can create diagonal matrices with `diag()`

```
# diagonal matrix
Diagonal = diag(runif(3))
Diagonal

##      [,1]  [,2]  [,3]
## [1,] 0.07314 0.0000 0.000
## [2,] 0.00000 0.7547 0.000
## [3,] 0.00000 0.0000 0.286
```

We can also use `diag()` to extract the diagonal from a square matrix

```
# diagonal matrix
diag(Square)

## [1] 0.5858 0.8136 0.9490
```

129

## Identity matrix

A diagonal matrix is the **identity matrix** if all diagonal elements are equal to one.

We can also use `diag()` to create identity matrices:

```
# identity matrix
Identity = diag(1, 3, 3)
Identity

##      [,1]  [,2]  [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

130

## Upper Triangular matrix

A matrix is **upper triangular** if  $x_{ij} = 0$  for all  $i > j$

```
# upper triangular
Upper_Triang = matrix(c(1, 0, 0, 2, 4, 0, 3, 5, 6), 3, 3)
Upper_Triang

##      [,1]  [,2]  [,3]
## [1,]    1    2    3
## [2,]    0    4    5
## [3,]    0    0    6
```

131

## Upper Triangular matrix

A matrix is **upper triangular** if  $x_{ij} = 0$  for all  $i > j$

```
# upper triangular
Upper_Triang = matrix(c(1, 0, 0, 2, 4, 0, 3, 5, 6), 3, 3)
Upper_Triang

##      [,1]  [,2]  [,3]
## [1,]    1    2    3
## [2,]    0    4    5
## [3,]    0    0    6
```

132

## Opérations de base sur les matrices

133

## Matrix Addition

**A + B**

Matrix addition of two matrices **A + B** is defined when **A** and **B** have the same dimensions:

```
# matrix A (2,3)
A = matrix(1:6, 2, 3)

# matrix B (2, 3)
B = matrix(7:9, 2, 3)

A + B

##      [,1] [,2] [,3]
## [1,]    8   12   13
## [2,]   10   11   15
```

134

## Scalar Multiplication

**0.5 \* X**

We can multiply a matrix by a scalar using the usual product operator `*`, moreover it doesn't matter if we pre-multiply or post-multiply:

```
# matrix X (3,4)
X = matrix(1:3, 3, 4)

# (pre)multiply X by 0.5
(1/2) * X

##      [,1] [,2] [,3] [,4]
## [1,]  0.5  0.5  0.5  0.5
## [2,]  1.0  1.0  1.0  1.0
## [3,]  1.5  1.5  1.5  1.5

# matrix X (3,4)
X = matrix(1:3, 3, 4)

# (post)multiply X by 0.5
X * 0.5

##      [,1] [,2] [,3] [,4]
## [1,]  0.5  0.5  0.5  0.5
## [2,]  1.0  1.0  1.0  1.0
## [3,]  1.5  1.5  1.5  1.5
```

135

## Matrix-Matrix Multiplication

**A %\*% B**

The matrix product operator in R is `%*%`. We can multiply matrices **A** and **B** if the number of columns of **A** is equal to the number of rows of **B**

```
# matrix A (2,3)
A = matrix(1:6, 2, 3)

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6

# matrix B (3, 2)
B = matrix(7:9, 3, 2)

##      [,1] [,2]
## [1,]    7    7
## [2,]    8    8
## [3,]    9    9

# product AB (2, 2)
AB = A %*% B
AB

##      [,1] [,2]
## [1,]   76   76
## [2,]  100  100

# product BA (2, 2)
BA = B %*% A
BA

##      [,1] [,2]
## [1,]   21   49
## [2,]   24   56
## [3,]   27   63
```

136

## Matrix-Matrix Multiplication (con't)

$A \%*\% (B \%*\% C)$

Matrix multiplication is **associative**:  $A(BC) = (AB)C$   
 (Obviously, the dimensions must conform to the matrix product)

```
# associative
A %*% (B %*% AB)

##      [,1] [,2]
## [1,] 13376 13376
## [2,] 17600 17600
```

```
# associative
(A %*% B) %*% AB

##      [,1] [,2]
## [1,] 13376 13376
## [2,] 17600 17600
```

137

## Matrix-Matrix Multiplication (con't)

$A \%*\% (B + C)$

Matrix multiplication is **distributive over addition**:

$$\begin{aligned} A(B + C) &= (AB) + (AC) \\ (A + B)C &= (AC) + (BC) \end{aligned}$$

```
# distributive
D = t(A)
A %*% (B + D)

##      [,1] [,2]
## [1,] 111 120
## [2,] 144 156
```

```
# distributive
(A %*% B) + (A %*% D)

##      [,1] [,2]
## [1,] 111 120
## [2,] 144 156
```

138

## Cross Products

$t(X) \%*\% X, X \%*\% t(X)$

A very common type of products in multivariate data analysis are  $X'X$  and  $XX'$ , sometimes known as **crossproducts**.

```
# X'X
t(X) %*% X

##      [,1] [,2] [,3] [,4]
## [1,]    14   14   14   14
## [2,]    14   14   14   14
## [3,]    14   14   14   14
## [4,]    14   14   14   14
```

```
# XX'
X %*% t(X)

##      [,1] [,2] [,3]
## [1,]    4    8   12
## [2,]    8   16   24
## [3,]   12   24   36
```

139

## Cross Products (con't)

In R we have the functions `crossprod()` and `tcrossprod()` which are formally equivalent to:

- ▶ `crossprod(X, X) ≡ t(X) %*% X`
- ▶ `tcrossprod(X, X) ≡ X %*% t(X)`

```
# X'X
crossprod(X, X)

##      [,1] [,2] [,3] [,4]
## [1,]    14   14   14   14
## [2,]    14   14   14   14
## [3,]    14   14   14   14
## [4,]    14   14   14   14
```

```
# XX'
tcrossprod(X, X)

##      [,1] [,2] [,3]
## [1,]    4    8   12
## [2,]    8   16   24
## [3,]   12   24   36
```

However, `crossprod()` and `tcrossprod()` are usually **slightly faster** than using `t()` and `%*%`

140

## Matrix-Vector Multiplication

We can **post-multiply** an  $n \times p$  matrix  $\mathbf{X}$  with a vector  $\mathbf{b}$  with  $p$  elements. This means making **linear combinations** (weighted sums) of the columns of  $\mathbf{X}$ :

```
# matrix X (4,3)
(X = matrix(1:12, 3, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

# vector b (length 4)
(b = seq(0.25, 1, by = 0.25))

## [1] 0.25 0.50 0.75 1.00
```

```
# product: Xb
X %*% b

##      [,1]
## [1,] 17.5
## [2,] 20.0
## [3,] 22.5
```

141

## Vector-Matrix Multiplication

We can **pre-multiply** a vector  $\mathbf{a}$  (with  $n$  elements) with an  $n \times p$  matrix  $\mathbf{X}$ . This means making **linear combinations** (weighted sums) of the rows of  $\mathbf{X}$ :

```
# matrix X (4,3)
(X = matrix(1:12, 3, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

# vector a (length 3)
(a = 1:3)

## [1] 1 2 3
```

```
# product: a'X
a %*% X

##      [,1] [,2] [,3] [,4]
## [1,] 14   32   50   68
```

142

## Matrix and Vector Multiplications

Notice that when we use the product operator `%*%` R is smart enough to use the convention that vectors are  $n \times 1$  matrices.

Notice also that if we ask for a **vector-matrix multiplication**, we can use both formulas:

1. `a %*% X`
2. `t(a) %*% X`

(R will reformat the  $n$  vector as an  $n \times 1$  matrix first)

143

## Trace, rang, déterminant, inverse, ... d'une matrice

144

## Trace

The **trace** of a *square* matrix  $\mathbf{X}$  is the sum of its diagonal elements. In R we can use the function `diag()` to extract the elements in the diagonal of a matrix, and then add them with `sum()`

```
# matrix X
set.seed(5)
(X = matrix(sample(1:20, size = 16, replace = TRUE), 4, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    5    3   20    7
## [2,]   14   15    3   12
## [3,]   19   11    6    6
## [4,]    6   17   10    5

# trace of X
sum(diag(X))

## [1] 31
```

145

## Rank

### Matrix Rank

One of the most important concepts in matrix algebra is that of **rank**. Roughly speaking, the rank of a matrix  $\mathbf{X}$  is the dimensionality of the row-column spaces encoded by  $\mathbf{X}$ .

### Nondegenerateness

The **rank** of a matrix  $\mathbf{X}$  is a measure of the *nondegenerateness* of the system of linear equations encoded by  $\mathbf{X}$ .

146

## Ranks

### Column Rank

The **column rank** of a matrix  $\mathbf{X}$  is the maximum number of *linearly independent columns* of  $\mathbf{X}$ . It is the dimension of the column space of  $\mathbf{X}$ .

### Row Rank

The **row rank** of a matrix  $\mathbf{X}$  is the maximum number of *linearly independent rows* of  $\mathbf{X}$ . It is the dimension of the row space of  $\mathbf{X}$ .

### Rank

The column rank is equal to the row rank.

147

## Rank in R

### How to find the rank?

A common approach to find the rank of a matrix is to reduce it to a simpler form. One option in R is to use the `qr()` function which returns the rank (among other results)

```
# matrix X
set.seed(5)
(X = matrix(sample(1:20, size = 16, replace = TRUE), 4, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    5    3   20    7
## [2,]   14   15    3   12
## [3,]   19   11    6    6
## [4,]    6   17   10    5

# rank of X
Xqr = qr(X)
Xqr$rank

## [1] 4
```

148

## Determinant

### Determinant

The determinant is a real number associated with every square matrix. It tells us things about a square matrix that are useful in systems of linear equations, and helps us find the inverse.

### Geometric Interpretation

When we regard an  $n \times n$  square matrix  $\mathbf{X}$  as a linear transformation, the determinant gives the area or volume magnification factor (in the the  $n$ -dimensional space) associated to  $\mathbf{X}$

149

## Determinant in R

In R, we can use the function `det()` to calculate the determinant of a square matrix:

```
# matrix X
set.seed(5)
(X = matrix(sample(1:20, size = 16, replace = TRUE), 4, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    5    3   20    7
## [2,]   14   15    3   12
## [3,]   19   11    6    6
## [4,]    6   17   10    5

# determinant of X
det(X)

## [1] 35991
```

150

## Matrix Inverse

### Inverse

The inverse of a square  $n \times n$  matrix  $\mathbf{X}$  is that unique  $n \times n$  matrix  $\mathbf{X}^{-1}$  whose elements are such that:

$$\mathbf{X}\mathbf{X}^{-1} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$$

### Existence

The inverse of a matrix exists if, and only if,  $|\mathbf{X}|$  is non-zero. In other words, if  $\mathbf{X}$  is *non-singular*

151

## Inverse in R

In R, we can use the function `solve()` to calculate the inverse of a square matrix:

```
# matrix X
set.seed(3)
(X = matrix(sample(1:10, size = 9, replace = TRUE), 3, 3))

##      [,1] [,2] [,3]
## [1,]    2    4    2
## [2,]    9    7    3
## [3,]    4    7    6

# inverse of X
(Xinv = solve(X))

##      [,1]     [,2]     [,3]
## [1,] -0.375  0.17857  0.03571
## [2,]  0.750 -0.07143 -0.21429
## [3,] -0.625 -0.03571  0.39286
```

152

## Inverse in R (con't)

We can test that  $\mathbf{X}\mathbf{X}^{-1} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$

```
# identity
round(Xinv %*% X, 3)

##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1

# identity
round(X %*% Xinv, 3)

##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

153

## Décomposition d'une matrice

154

## Introduction (cont.)

### Importance

Matrix decompositions make it easier to study the properties of matrices. Likewise, many computation tasks become easier with decompositions.

### Typical Applications

- ▶ solving systems of linear equations
- ▶ inverting a matrix
- ▶ analyzing numerical stability of a system
- ▶ understanding the structure of data
- ▶ finding basis for column space (or row space) of a matrix
- ▶ etc

155

## Some Assumptions

### Real Matrices

For this lecture all matrices will be assumed to be **real matrices**, i.e. matrices containing elements in the set of Real numbers.

### Rectangular $n > p$

Also for the sake of simplicity, we'll assume **rectangular matrices as "tall"** matrices, that is, more rows than columns.

156

## Matrix Considerations

### Focus

We will focus on **general rectangular matrices** and on **symmetric matrices** because of their relevance for multivariate statistics.

### Specifically

- ▶ For **symmetric matrices** we'll consider **Cholesky**, **LU** and **EVD** decompositions.
- ▶ For **general rectangular matrices** we'll consider **QR** and **SVD** decompositions.

157

## Matrix Considerations (con't)

### Symmetric Matrices

For symmetric matrices  $\mathbf{X}$  we typically look at decompositions of the form  $\mathbf{X} = \mathbf{A}\mathbf{A}'$  or  $\mathbf{X} = \mathbf{B}\mathbf{B}'$

### General Rectangular Matrices

For rectangular matrices  $\mathbf{X}$  we look at decompositions of the form  $\mathbf{X} = \mathbf{AB}$  or  $\mathbf{X} = \mathbf{ABC}$

158

## Important Concepts

### Full Rank

If  $\mathbf{X}$  is an  $n \times p$  matrix, then  $\mathbf{X}$  is of **full rank** if  $\mathbf{X}\mathbf{b} = \mathbf{0}$  only if  $\mathbf{b} = \mathbf{0}$ .

### Linear Independence

We also say the columns of  $\mathbf{X}$  are *linearly independent*. Full row rank is defined in a similar way.

159

## Important Concepts (con't)

### Full Rank Decomposition

If  $\mathbf{X}$  can be written as the product  $\mathbf{AB}$ , with  $\mathbf{A}$  an  $n \times k$  matrix of full rank, and  $\mathbf{B}$  a  $k \times p$  matrix of full rank, then  $\mathbf{X}$  is said to have rank  $k$ .

The decomposition  $\mathbf{X} = \mathbf{AB}$  is a *full rank* decomposition

160

## Cholesky Decomposition

### Cholesky Decomposition

A **real symmetric** matrix (*p.d.* or *p.s.d.*)  $\mathbf{X}$  can be decomposed as:

$$\mathbf{X} = \mathbf{L}\mathbf{L}'$$

where  $\mathbf{L}$  is a lower triangular matrix with real diagonal entries.

**Equivalently**

The Cholesky decomposition of  $\mathbf{X}$  can also be expressed as  $\mathbf{X} = \mathbf{R}'\mathbf{R}$  with  $\mathbf{R}$  an upper triangular matrix.

161

## Cholesky Decomposition (con't)

### For Positive Definite Matrices

If  $\mathbf{X}$  is a *positive definite* matrix then  $\mathbf{L}$  is a lower triangular matrix with real and **positive** diagonal entries.

### For Positive Semi-Definite Matrices

If  $\mathbf{X}$  is a *positive semi-definite* matrix then  $\mathbf{L}$  is a lower triangular matrix with real and **non-negative** diagonal entries.

162

## Cholesky Decomposition

$$\mathbf{X} = \mathbf{L}\mathbf{L}'$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pp} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{p1} & l_{p2} & \cdots & l_{pp} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{p1} \\ 0 & l_{22} & \cdots & l_{p2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{pp} \end{bmatrix}$$

$$\mathbf{X} = \mathbf{R}'\mathbf{R}$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pp} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 & \cdots & 0 \\ r_{12} & r_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ r_{1p} & r_{2p} & \cdots & r_{pp} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ 0 & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{pp} \end{bmatrix}$$

163

## Cholesky Decomposition in R

### chol() function

In R we have the function `chol()` that performs the Cholesky decomposition.

### chol() output

The output of `chol()` is the matrix  $\mathbf{R}$  (upper triangular) such that  $\mathbf{X} = \mathbf{R}'\mathbf{R}$ . If the argument `pivot = TRUE`, then "pivot" and "rank" are also returned.

164

## Cholesky Decomposition in R

### Apply chol()

```
# X matrix (positive semi-definite)
set.seed(11)
Y = matrix(rnorm(20), 5, 4)
(X = t(Y) %*% Y)

## [1,]   [1]   [2]   [3]   [4]
## [1,] 5.896 -1.4814  1.8063 -1.3254
## [2,] -1.4814  4.0264  0.5179  0.9336
## [3,]  1.8063  0.5179  4.5617 -0.3928
## [4,] -1.3254  0.9336 -0.3928  1.6186

# Cholesky decomposition
R_chol = chol(X)
R_chol

## [1,]   [1]   [2]   [3]   [4]
## [1,] 2.428 -0.6101 0.7439 -0.54587
## [2,]  0.000  1.9113 0.5084  0.31423
## [3,]  0.000  0.0000 1.9364 -0.07565
## [4,]  0.000  0.0000 0.0000 1.10281
```

165

Note that R returns **R**, that is an *upper triangular matrix*

```
# X = R'R
t(R_chol) %*% R_chol

## [1,]   [1]   [2]   [3]   [4]
## [1,] 5.896 -1.4814  1.8063 -1.3254
## [2,] -1.4814  4.0264  0.5179  0.9336
## [3,]  1.8063  0.5179  4.5617 -0.3928
## [4,] -1.3254  0.9336 -0.3928  1.6186
```

## LU Decomposition

### LU

An  $n \times n$  square matrix **X** can be decomposed, with proper row and/or column orderings or permutations, as

$$\mathbf{X} = \mathbf{L}\mathbf{U}$$

where

- ▶ **L** is an  $n \times n$  *Lower triangular matrix*
- ▶ **U** is an  $n \times n$  *Upper triangular matrix*

*Without a proper ordering or permutations in **X**, the factorization may fail to materialize*

166

## LU Decomposition (con't)

### LU

$$\mathbf{X} = \mathbf{L}\mathbf{U}$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

167

## Eigen-Value Decomposition

### EVD

An  $n \times n$  **symmetric matrix** **X** can be decomposed as:

$$\mathbf{X} = \mathbf{U}\Lambda\mathbf{U}'$$

where

- ▶ **U** is a  $n \times p$  column **orthonormal** matrix containing the eigen-vectors of **X**
- ▶ **Λ** is a  $p \times p$  **diagonal** matrix containing the eigen-values of **X**

168

## Eigen-Value Decomposition

EVD

$$\mathbf{X} = \mathbf{U}\Lambda\mathbf{U}'$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{1p} \\ u_{21} & \cdots & u_{2p} \\ \vdots & \ddots & \vdots \\ u_{n1} & \cdots & u_{np} \end{bmatrix} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ 0 & \cdots & \lambda_p \end{bmatrix} \begin{bmatrix} u_{11} & \cdots & u_{n1} \\ u_{12} & \cdots & u_{n2} \\ \vdots & \ddots & \vdots \\ u_{1p} & \cdots & u_{np} \end{bmatrix}$$

Likewise

$$\mathbf{X} = \sum_{k=1}^p \lambda_k \mathbf{u}_k \mathbf{u}_k'$$

169

## EVD in R

### eigen() function

R provides the function `eigen()` to perform a eigen-value decomposition of a given matrix

### eigen() output

A list with the following components

`values` a vector containing the eigen-values

`vectors` a matrix whose columns contain the eigen-vectors

170

## EVD example in R

```
# X matrix
set.seed(22)
Y = matrix(rnorm(20), 5, 4)
X = t(Y) %*% Y

# eigen-value decomposition
EVD = eigen(X)

# elements returned by svd()
names(EVD)

## [1] "values" "vectors"

# vector of singular values
(lambda = EVD$values)

## [1] 15.615  4.090  2.175  0.187
```

```
# matrix of eigen-vectors
(U = EVD$vectors)

## [,1]   [,2]   [,3]   [,4]
## [1,]  0.5708  0.7407 -0.3363  0.1043
## [2,] -0.2742  0.5295  0.76797  0.2338
## [3,]  0.2772 -0.3206  0.0462  0.9046
## [4,]  0.7226 -0.2612  0.54181 -0.3408

# U orthonormal (U'U = I)
t(U) %*% U
```

171

## EVD example in R (con't)

```
# X equals U L U'
U %*% diag(lambda) %*% t(U)

## [,1]   [,2]   [,3]   [,4]
## [1,]  7.584 -1.401  1.485  5.244
## [2,] -1.401  3.614 -1.767 -2.769
## [3,]  1.485 -1.767  1.778  3.466
## [4,]  5.244 -2.769  3.466  9.092

# compare to X
X

## [,1]   [,2]   [,3]   [,4]
## [1,]  7.584 -1.401  1.485  5.244
## [2,] -1.401  3.614 -1.767 -2.769
## [3,]  1.485 -1.767  1.778  3.466
## [4,]  5.244 -2.769  3.466  9.092
```

172

## QR Decomposition

### QR

An  $n \times p$  matrix  $\mathbf{X}$  can be decomposed as

$$\mathbf{X} = \mathbf{Q}\mathbf{R}$$

where

- ▶  $\mathbf{Q}$  is an  $n \times p$  *orthonormal* matrix
- ▶  $\mathbf{R}$  is an *upper triangular* matrix

173

## QR Decomposition (con't)

### QR

$$\mathbf{X} = \mathbf{Q}\mathbf{R}$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1p} \\ q_{21} & q_{22} & \cdots & q_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \cdots & q_{np} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ 0 & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{pp} \end{bmatrix}$$

### Q basis

If  $\mathbf{X}$  has  $k$  linearly independent columns, then the first  $k$  columns of  $\mathbf{Q}$  form an orthonormal basis for the column space of  $\mathbf{X}$

174

## QR Decomposition in R

### QR in R

In R we can use the function `qr()` to compute the QR decomposition of a matrix.

### `qr()` output

A list with the following elements:

- `qr` a matrix with the same dimensions as the input `x`
- `qraux` a vector of length `ncol(x)` with info about `Q`
- `rank` the rank of `x`
- `pivot` information of the pivoting strategy

175

## QR Decomposition in R

```
# matrix
set.seed(33)
(X = matrix(rnorm(15), 5, 3))

##          [,1]     [,2]     [,3]
## [1,] -0.1359  0.4986  0.16735
## [2,] -0.0408 -0.7552 -0.02928
## [3,]  1.0105  0.7786  1.87585
## [4,] -0.1583  0.7546  0.24463
## [5,] -2.1566 -1.0995  0.70215

# QR decomposition
QR = qr(X)
QR

## $qr
##          [,1]     [,2]     [,3]
## [1,]  2.39112  1.2554  0.13427
## [2,]  0.01706  1.2758  0.63494
## [3,] -0.42262 -0.3731 -1.91816
## [4,]  0.06619 -0.6286 -0.01121
## [5,]  0.90194  0.3557  0.42923
##
## $rank
## [1] 3
##
## $qraux
## [1] 1.057 1.582 1.903
##
## $pivot
## [1] 1 2 3
##
## attr(,"class")
## [1] "qr"
```

176

## QR Decomposition in R (con't)

Use `qr.Q()` to extract the **Q** matrix of a QR decomposition

```
# Q matrix
qr.Q(QR)

##      [,1]     [,2]     [,3]
## [1,] -0.05685  0.44678  0.05667
## [2,] -0.01706 -0.57518 -0.17632
## [3,]  0.42262  0.19442 -0.88400
## [4,] -0.06619  0.65658  0.08517
## [5,] -0.90194  0.02564 -0.42070
```

Use `qr.R()` to extract the **R** matrix of a QR decomposition

```
# R matrix
qr.R(QR)

##      [,1]     [,2]     [,3]
## [1,]  2.391  1.256  0.1343
## [2,]  0.000  1.276  0.6349
## [3,]  0.000  0.000 -1.9182
```

177

## Singular Value Decomposition

### SVD

An  $n \times p$  matrix **X** can be decomposed as:

$$\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}'$$

where

- ▶ **U** is a  $n \times p$  column **orthonormal** matrix containing the left singular vectors *ie*  $\mathbf{X}\mathbf{X}'$  eigen vectors
- ▶  **$\Lambda$**  is a  $p \times p$  **diagonal** matrix containing the singular values of **X**
- ▶ **V** is a  $p \times p$  column **orthonormal** matrix containing the right singular vectors *ie*  $\mathbf{X}'\mathbf{X}$  eigen vectors

178

## Singular Value Decomposition

### SVD

$$\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}'$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{1p} \\ u_{21} & \cdots & u_{2p} \\ \vdots & \ddots & \vdots \\ u_{n1} & \cdots & u_{np} \end{bmatrix} \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_p \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{p1} \\ v_{12} & \cdots & v_{p2} \\ \vdots & \ddots & \vdots \\ v_{1p} & \cdots & v_{pp} \end{bmatrix}$$

Likewise

$$\mathbf{X} = \sum_{k=1}^p \lambda_k \mathbf{u}_k \mathbf{v}_k'$$

179

## svd() in R

### svd() function

R provides the function `svd()` to perform a singular value decomposition of a given matrix

### svd() output

A list with the following components

- d** a vector containing the singular values
- u** a matrix whose columns contain the left singular vectors
- v** a vector whose columns contain the right singular vectors

180

## SVD example in R

```
# X matrix
set.seed(22)
X = matrix(rnorm(20), 5, 4)

# singular value decomposition
SVD = svd(X)

# elements returned by svd()
names(SVD)

## [1] "d" "u" "v"

# vector of singular values
(d = SVD$d)

## [1] 3.9516 2.0224 1.4748 0.4324
```

```
# matrix of left singular vectors
(U = SVD$u)
```

```
## [,1]      [,2]      [,3]      [,4]
## [1,] -0.4281 -0.53913 -0.7233  0.009794
## [2,]  0.5269 -0.76863  0.2860  0.056100
## [3,]  0.5753  0.05000 -0.4421  0.131072
## [4,]  0.2215  0.05273 -0.1702 -0.951234
## [5,] -0.4021 -0.33655  0.4131 -0.273371
```

```
# matrix of right singular vectors
(V = SVD$v)
```

```
## [,1]      [,2]      [,3]      [,4]
## [1,]  0.5708 -0.7407  0.33863  0.1043
## [2,] -0.2742 -0.5295 -0.76797  0.2338
## [3,]  0.2772  0.3206 -0.04462  0.9046
## [4,]  0.7226  0.2612 -0.54181 -0.3408
```

181

## SVD example in R (con't)

```
# U orthonormal (U'U = I)
t(U) %*% U

## [,1]      [,2]      [,3]      [,4]
## [1,] 1.000e+00  1.388e-16  2.776e-17  0.000e+00
## [2,] 1.388e-16  1.000e+00 -2.776e-17 -8.327e-17
## [3,] 2.776e-17 -2.776e-17  1.000e+00  5.551e-17
## [4,] 0.000e+00 -8.327e-17  5.551e-17  1.000e+00
```

```
# V orthonormal (V'V = I)
```

```
t(V) %*% V

## [,1]      [,2]      [,3]      [,4]
## [1,] 1.000e+00 -1.100e-16 -5.551e-17  1.110e-16
## [2,] -1.100e-16  1.000e+00  8.327e-17  1.943e-16
## [3,] -5.551e-17  8.327e-17  1.000e+00 -8.327e-17
## [4,] 1.110e-16  1.943e-16 -8.327e-17  1.000e+00
```

```
# X equals U D V'
U %*% diag(d) %*% t(V)

## [,1]      [,2]      [,3]      [,4]
## [1,] -0.5121  1.85809 -0.76391 -0.9222
## [2,]  2.4852 -0.06603  0.08196  0.8616
## [3,]  1.0078 -0.16276  0.74303  2.0029
## [4,]  0.2928 -0.19986 -0.08402  0.9366
## [5,] -0.2090  0.30056 -0.79289 -1.6157
```

```
# compare to X
X

## [,1]      [,2]      [,3]      [,4]
## [1,] -0.5121  1.85809 -0.76391 -0.9222
## [2,]  2.4852 -0.06603  0.08196  0.8616
## [3,]  1.0078 -0.16276  0.74303  2.0029
## [4,]  0.2928 -0.19986 -0.08402  0.9366
## [5,] -0.2090  0.30056 -0.79289 -1.6157
```

182

## Summary

### Matrix Decompositions

Matrix	Decomposition
Symmetric	Cholesky
Symmetric	LU
Symmetric	Spectral (EVD)
Rectangular	QR
Rectangular	Singular Value Decomposition (SVD)

183

## Références

- L. Bellanger, R. Tomassone, *Exploration de données et méthodes statistiques : Data analysis & Data mining avec R. Collection Références Sciences*, Editions Ellipses, Paris, 2014.
- M. Baumgartner. Introduction à R, 2001.
- A. Bouchier. Formation au logiciel R, 2011.
- F. Huson (dir.). *R pour la statistique et la science des données*, PUR, Rennes, 2018.
- W. N. Venables, B. D. Ripley . *Modern Applied Statistics with S-PLUS*, 2nd edition, Springer, New York, 1997

### Sur R:

- R Foundation, <http://www.r-project.org>
- <http://cran.r-project.org/>
- Nombreux documents en ligne
- <http://gastonsanchez.com/teaching/>

Non traité : programmer sous R

184