

# ECN 6338 Cours 1

## Introduction

William McCausland

2025-09-04

## Quelques observations

- ▶ L'économie et la recherche des implications des modèles.
- ▶ La difficulté d'exprimer ces implications en forme analytique.
- ▶ L'importance de l'optimisation et des espérances dans les problèmes d'agents économiques.
- ▶ L'importance de l'optimisation et des espérances dans les problèmes économétriques.
- ▶ L'apport de l'analyse numérique en microéconomie, macroéconomie et économétrie.
- ▶ Le site web [QuantEcon](#) donne une bonne idée de la diversité d'applications.

# L'optimisation

## Par les agents des modèles économiques

1. maximisation de l'utilité (avec ou sans contraintes)
  - a. choix de panier dans les modèles statiques
  - b. choix de travail, de loisir, de consommation, de placements dans les modèles dynamiques en macro et finance
  - c. choix d'action dans les jeux (enchères, jeux de négociation, jeux de signalisation)
2. maximisation du profit (choix des quantités de production)
  - a. choix de panier d'intrants dans les modèles statiques
  - b. choix d'investissement, de niveau de R&D, dans les modèles dynamiques en macro
  - c. choix d'action dans les jeux
    - i. jeux d'oligopole : cournot, bertrand, stackelberg
    - ii. jeux en organisation industrielle : d'entrée

# L'optimisation (suite)

## Par les économètres

### 1. Estimation par Extremum

- a. maximum de vraisemblance
- b. moindres carrés non-linéaire
- c. méthode des moments généralisée
- d. régression quantile

# L'intégration

## Par les agents

1. Évaluation de l'espérance de l'utilité, du profit
2. Évaluation de l'utilité dans les modèles macro en temps continu
3. Évaluation des fonctions d'enchère

## Par les économètres

1. Inférence bayésienne : calcul des moyennes *a posteriori*
2. Simulation Monte Carlo : des estimateurs, du couvrage, ...
3. Simulation bootstrap

# D'autres opérations

D'autres opérations numériques jouent souvent un rôle de soutien

- ▶ Résolution de systèmes d'équation
  - ▶ reliée à l'optimisation, à la recherche des racines
  - ▶ recherche d'un équilibre
- ▶ Approximation de fonctions
- ▶ Résolution d'équations différentielles
- ▶ Simulation de variables aléatoires
  - ▶ pour l'intégration (méthodes Monte Carlo)
  - ▶ pour l'optimisation (recuit simulé, simulated annealing)

# Ce cours, relatif au livre classique de Judd

Relatif au livre de Judd, je mets un accent sur

- ▶ l'économétrie (cependant, ce n'est pas un cours d'économétrie)
  - ▶ exemples dans le domaine de choix discret
  - ▶ maximum de vraisemblance
  - ▶ inférence bayésienne
- ▶ la simulation
  - ▶ intégration par simulation (utile en grandes dimensions)
  - ▶ optimisation par recuit simulé
  - ▶ applications en inférence bayésienne

Je mets moins d'emphasis sur l'optimisation dynamique. Le but ici est de présenter les cas les plus simples pour vous préparer pour la matière plus avancée.

# Évaluation

Type	Date	Pondération
~10 interrogations de 10 minutes	Au début des classes	20%
Quatres exercices computationnelles	26 septembre, 17 octobre, 14 novembre, 5 décembre	40%
Examen final	20 avril	40%



# Documents et Communication

## Site Github du cours

1. Diapos (code source en [R Markdown](#), pdf)
2. Démonstrations (en [R](#))
3. Lectures, exercices
4. Devoirs avec computation
5. Liens vers les enregistrements des cours à distance
6. README.md comme page d'accueil

## Site StudiUM du cours

1. Messages aux étudiants
2. Documents avec droit d'auteur
3. Téléversement de vos devoirs computationnels

# Logiciels (pour les travaux pratiques, votre choix)

## R

- ▶ graticiel, accent sur la statistique, beaucoup d'applications
- ▶ utilisé pour les démonstrations du cours
- ▶ recommandé, introduction pendant la première séance TP

## Python

- ▶ graticiel, général, beaucoup d'applications

## Julia

- ▶ graticiel, général, moins utilisé que les autres
- ▶ très rapide, élégant

## Matlab

- ▶ commercial mais disponible à l'université, général, beaucoup d'applications
- ▶ son importance diminue en faveur de R et python

# Notation pour les dérivées multivariées

- ▶ Soit  $x$  un vecteur  $n \times 1$ ,  $y = f(x)$  un vecteur  $m \times 1$ .
- ▶ La *matrice jacobienne* ( $m \times n$ ) contient toutes les dérivées de première ordre:

$$f_x = \frac{\partial y}{\partial x}, \quad \text{où} \quad \left[ \frac{\partial y}{\partial x} \right]_{ij} = \frac{\partial y_i}{\partial x_j}.$$

- ▶ Le *gradient* est un cas spécial du Jacobien où  $y$  est scalaire, un vecteur ligne  $1 \times n$ .
- ▶ La *matrice hessienne* ( $n \times n$ ) contient toutes les dérivées de deuxième ordre pour  $y$  scalaire:

$$f_{xx} = \frac{\partial}{\partial x} \left( \frac{\partial y}{\partial x} \right)^{\top} = \frac{\partial^2 y}{\partial x \partial x^{\top}}.$$

- ▶ La notation ci-haut suit la convention “numerator layout” [ici](#)

## Quelques propriétés des dérivées multivariées

À la même [page](#) il y a des tableaux de propriétés, telles que :

- Pour une matrice constante  $A$ ,  $m \times n$ ,

$$\frac{\partial Ax}{\partial x} = A.$$

- Règle du produit : pour les vecteurs  $u(x)$  et  $v(x)$ ,  $m \times 1$ ,

$$\frac{\partial u^T v}{\partial x} = u^T \frac{\partial v}{\partial x} + v^T \frac{\partial u}{\partial x}.$$

- Règle des fonctions composées, de la chaîne : pour  $z = g(y)$ ;  $y = f(x)$ ;  $x$ ,  $y$  et  $z$  multivariés,

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} = \frac{\partial g(y)}{\partial y} \frac{\partial f(x)}{\partial x}.$$

# Analyse de l'erreur

Deux sources d'erreur numérique :

- ▶ Précision finie des nombres réels
- ▶ Troncation de calculs séquentiels infinis

Les erreurs se propagent à travers les computations.

# La représentation virgule flottante

L'ordinateur représente un nombre réel  $x$  comme

$$x = \pm m \times 2^{\pm e},$$

où

- ▶  $m \in \mathbb{N}$  est la mantisse et
- ▶  $e \in \mathbb{N}$  est l'exposant.

Le nombre de bits pour représenter  $m$  détermine la précision numérique.

Le nombre de bits pour représenter  $e$  détermine les points de dépassement et sous-dépassement numérique (overflow/underflow).

## Quatre constantes mécanique

Pour une machine donnée, les constantes suivantes décrivent les points de dépassement et soupassement, ainsi que la précision.

Constante	description
<code>double.xmax</code>	$x > 0$ le plus grand distinct de $\infty$ .
<code>double.xmin</code>	$x > 0$ le plus petit distinct de 0.
<code>double.eps</code>	$x > 0$ le plus petit tel que $1 + x$ et 1 sont distincts.
<code>double.neg.eps</code>	$x > 0$ le plus petit tel que $1 - x$ et 1 sont distincts.

On appelle

- ▶ `double.xmax` l'infini de la machine,
- ▶ `double.eps` l'epsilon de la machine.

## Trouver ces constantes avec R

```
m = .Machine  
m$double.eps
```

```
## [1] 2.220446e-16
```

```
m$double.neg.eps
```

```
## [1] 1.110223e-16
```

```
m$double.xmin
```

```
## [1] 2.225074e-308
```

```
m$double.xmax
```

```
## [1] 1.797693e+308
```



## Propagation de l'erreur

- ▶ L'erreur relative du résultat d'un calcul peut être très différente de l'erreur relative des intrants.
- ▶ Supposez qu'on évalue la dérivée numérique suivante, pour approximer la dérivée de la fonction  $f(x) = e^x$  à  $x = 0$  :

$$d_h = \frac{f(0+h) - f(0-h)}{2h} = \frac{e^h - e^{-h}}{2h},$$

où  $h > 0$  est très petit.

- ▶ Mettons que les erreurs relatives maximales de  $e^h$  et  $e^{-h}$  sont  $\epsilon$ .
- ▶ Puisque  $e^x = 1$  à  $x = 0$ , les erreurs absolues sont pareilles.
- ▶ Par une expansion de Taylor,

$$d_h \approx \frac{2h \pm 2\epsilon}{2h} = 1 \pm \frac{\epsilon}{h} = f'(0) + \frac{\epsilon}{h}$$

- ▶ L'erreur (relative et absolue) du résultat peut être aussi grande que  $\epsilon/h$ .

## Expansions de Taylor et de Mercator de la fonction $\log x$

L'expansion de Taylor de  $\log x$  autour de  $x = 1$  :

$$\log x = \sum_{k=1}^{\infty} \frac{(-1)^k (x-1)^k}{k} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \dots$$

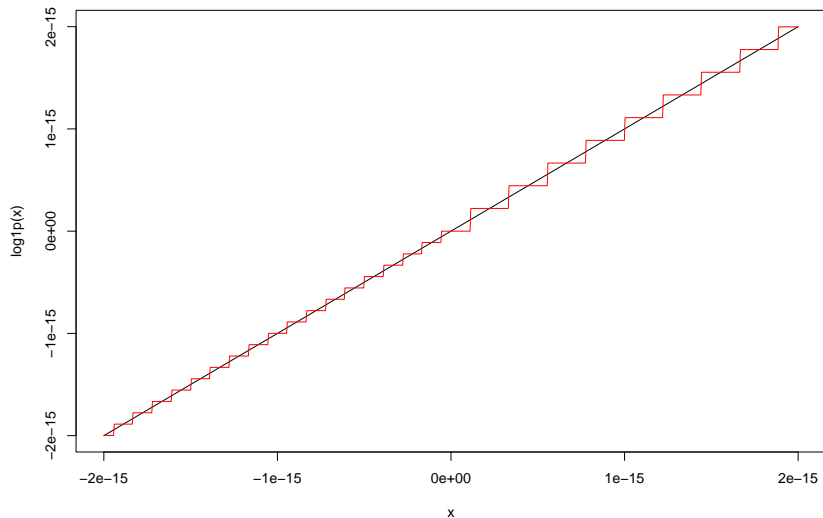
L'expansion de Mercator :

$$\log(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^k x^k}{k} = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots$$

- ▶ Si on veut évaluer  $\log(1+x)$  pour  $x$  petit, ne calcule pas  $1+x$  comme résultat intermédiaire.
- ▶ La fonction `log1p` en R (et autres langages) évalue la fonction  $f(x) = \log(1+x)$  directement avec l'expansion de Mercator.
- ▶ Exemple économique : pour le rendement net simple  $R$ , le rendement continu composé est  $\log(1+R)$ .

## La fonction log1p

```
x = seq(-2e-15, 2e-15, length.out=1000)
plot(x, log1p(x), 'l')
lines(x, log(1+x), col='red')
```



# Troncation mathématique

La deuxième source d'erreur est la troncation mathématique.

La valeur exacte de la fonction exponentielle est

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!},$$

mais on pratique il faut tronquer et utiliser un nombre fini  $N$  de termes :

$$\sum_{n=0}^N \frac{x^n}{n!}.$$

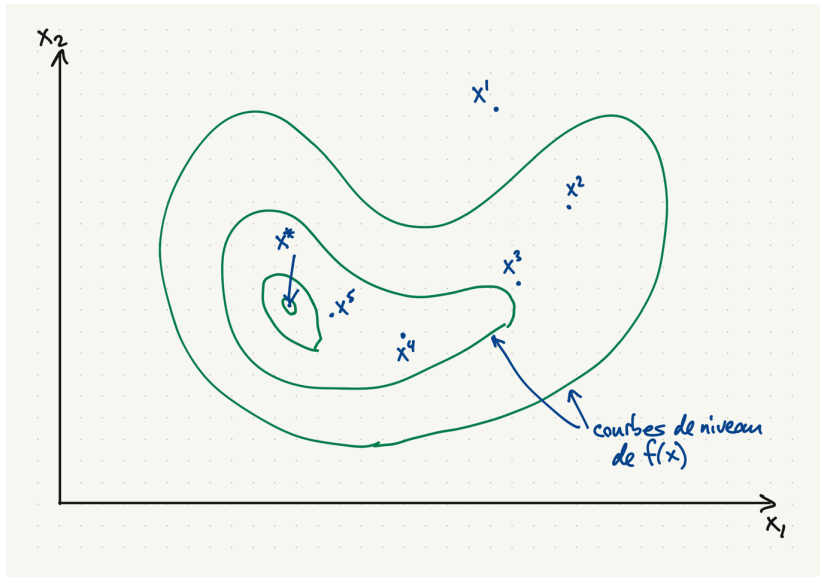
## Troncation mathématique, plus généralement

- ▶ Souvent un algorithme itératif génère une suite de vecteurs  $x^k$ ,  $k = 1, 2, \dots$ , qui converge au résultat voulu  $x^* \equiv \lim_{k \rightarrow \infty} x^k$ .
- ▶ Il faut accepter une valeur approximative  $x^k$ , pour  $k$  fini.
- ▶ Évaluer l'erreur  $\|x^k - x^*\|$  est infaisable.
- ▶ On peut utiliser  $\|x^{k+1} - x^k\|$ ,  $\dots$
- ▶  $\dots$  mais prudemment, parce que (par exemple) pour

$$x^k = \sum_{j=1}^k \frac{1}{j}$$

- ▶  $|x^k - x^{k-1}| = 1/k \rightarrow 0$ ,
- ▶ mais  $x^k$  diverge.

## Exemple, maximisation de $f(x)$



## Règles d'arrêt

- ▶ En pratique, on veut arrêter quand  $\|x^k - x^{k+1}\|$  est petit relatif et à  $\|x^k\|$  et à zéro.
- ▶ Par exemple, arrêter quand :

$$\frac{\|x^k - x^{k+1}\|}{1 + \|x^k\|} \leq \epsilon.$$

- ▶ Si on peut trouver un  $\beta < 1$  tel que

$$\forall k, \|x^{k+1} - x^*\| \leq \beta \|x^k - x^*\|,$$

on a une garantie que  $\|x^k - x^*\| \leq \|x^k - x^{k+1}\| / (1 - \beta)$ .

- ▶ Des fois, la convergence est *linéaire* : il existe  $\beta$  tel que

$$\lim_{k \rightarrow \infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} \leq \beta < 1,$$

auquel cas, on peut estimer  $\beta$  et espérer que cela marche.

# Analyse (de la complexité) d'algorithmes

## Notation $O(\cdot)$

- ▶ Pour des fonctions  $f$  et  $g$  sur  $\mathbb{N}$ , on écrit  $f(n) = O(g(n))$  s'il existe  $M > 0$  tel que  $|f(n)| \leq Mg(n)$  pour chaque  $n$ .
- ▶ Par exemple  $f(n) = 6n^2 + 8n + 2 = O(n^2)$

## La complexité de certains algorithmes

- ▶  $O(1)$  : nombre d'opérations pour trouver le  $i$ -ième élément dans un  $n$ -vecteur;
- ▶  $O(\log n)$  : nombre de comparaisons pour trouver un élément donnée dans un  $n$ -vecteur trié, par recherche binaire;
- ▶  $O(n)$  : pour trouver un élément donnée dans un  $n$ -vecteur, par recherche exhaustive;
- ▶  $O(n^2)$  : nombre de multiplications scalaires pour multiplier une matrice  $n \times n$  et un vecteur  $n \times 1$ ;
- ▶  $O(n^3)$  : multiplier deux matrices  $n \times n$  (méthode évidente);
- ▶  $O(n^{2.81})$  : même chose, algorithme de Strassen



# L'évaluation des polynomes avec la méthode de Horner

Trois méthodes pour évaluer  $a_0 + a_1x + \cdots + a_nx^n$  :

1. Évaluation naïve,  $O(n^2)$  multiplications,  $O(1)$  registres :

$$a_0 + a_1 * x + a_2 * x * x + a_3 * x * x * x + \cdots$$

2. Meilleure, avec  $2n$  multiplications,  $O(n)$  registres :
  - a. Calculer  $x^i = x^{i-1} * x$ ,  $i = 2, \dots, n$ .
  - b. Calculer  $a_0 + a_1 * x + \cdots + a_n * x^n$ .
3. La méthode de Horner,  $n$  multiplications,  $O(1)$  registres :

$$a_0 + x * (a_1 + x * (a_2 + x * (a_3 + \cdots + x * (a_{n-1} + x * a_n) \dots)))$$

# Parallélisme

Deux types de problème où vous pouvez profiter des processeurs en parallèle :

1. problèmes avec l'embaras du parallélisme (**embarrassingly parallel problems**) où il n'y a pas de communication entre processeurs avant la fin des computations.
2. problèmes SIMD (single instruction multiple data)

Questions pratiques en commun :

- ▶ Les tâches individuelles doivent être suffisamment grandes relatif aux coûts fixes de communication entre processeurs.
- ▶ Ces couts fixes varient beaucoup :
  - ▶ coeurs multiples d'un processeur
  - ▶ processeurs multiples d'une machine
  - ▶ machines multiples d'un cluster

# L'embarras du parallelisme

## Problèmes avec l'embarras du parallelisme

1. Évaluation d'une fonction sur une grille de points
2. Intégration numérique
3. Simulation Monte Carlo indépendant
4. Évaluation d'une fonction de log vraisemblance (souvent)

$$L(\theta; y) = \sum_{t=1}^T \log f(y_t | \theta).$$

5. Multiplication des matrices

## Problèmes sans l'embarras du parallelisme

1. Méthodes itératives d'optimisation
2. Méthodes itératives pour trouver un point fixe
3. Simulation Markov chain Monte Carlo (MCMC)

# SIMD (Single Instruction, Multiple Data)

- ▶ Les GPUs (processeurs graphiques) peuvent exécuter les mêmes instructions pour plusieurs vecteurs différents de données.
- ▶ Convenable pour les problèmes où les boucles locales ont le même nombre d'itérations.
- ▶ Quand la structure de contrôle (control flow) est variable (if-else, do-while, etc.) les programmes marchent mais avec gaspillage.
- ▶ Prenons encore l'évaluation d'une fonction de log vraisemblance

$$L(\theta; y) = \sum_{t=1}^T \log f(y_t | \theta).$$

- ▶ Si l'évaluation de  $\log f(y_t | \theta)$  utilise les mêmes instructions, peu importe la valeur de  $y_t$ , le problème est disposé à SIMD.
- ▶ Si la suite des instructions dépend de  $y_t$ , SIMD est moins intéressant.