ECN 6338 Cours 1

Introduction

William McCausland

2022-01-07

Deux opérations clés de l'analyse numérique en économie

L'optimisation

Par les agents dans un modèle économique

- maximisation de l'utilité (choix des actions dans un jeu, des quantités de consommation)
- maximisation des profits (choix des quantités de production)

Par les économètres

- estimation économétrique par maximum de vraisemblance
- autres méthodes d'estimation par optimisation

L'intégration (très souvent le calcul d'une espérance)

Par les agents

- évaluer l'espérance de l'utilité, des profits

Par les économètres

- simulations Monte Carlo des estimateurs
- bootstrap
- inférence bayésienne

Autres opérations

Les autres opérations sont souvents les outils de soutien

- Résolution des systèmes d'équation
  - reliée à l'optimisation, à la recherche des racines
  - recherche d'un équilibre
- Approximation
- Résolution des équations différentielles
- Simulation

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)
  - applications en inférence bayésienne

Je mets moins d'emphase sur l'optimisation dynamique. Le but ici est de présenter les cas les plus simples et de vous guider vers la matière plus avancée.

Documents et Communication

Site Github du cours

1. Diapos (code source, pdf)
2. Démonstrations
3. Lectures, exercices
4. Devoirs avec computation

Site StudiUM du cours

1. Messages
2. Forums (possiblement)
3. Documents avec droit d'auteur
4. Chargement des devoirs

Logiciels (votre choix de quatre logiciels)

R

- graticiel, accent sur la statistique, beaucoup d'applications
- utilisé pour les démonstrations du cours

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
- toujours populaire mais 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:

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

- 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 quand y est 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 les suivantes:

- Pour les matrices A constantes, $m \times n$,

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

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

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

- Règle de la chaîne, des fonctions composées : pour $z = g(y)$,

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

Analyse de l'erreur

Deux sources d'erreur numérique :

- Précision finie des nombres réels
- Troncation de calculs infinis

Les deux sources d'erreur propagent

La représentation virgule flottante

L'ordinateur représente un nombre réel x comme

$$x = s \times m \times 2^e,$$

où

- $s \in \{-1, 1\}$ est la signe,
- $m \in \mathbb{R}_+$ est la mantisse et
- $e \in \{\ldots, -1, 0, 1, \ldots\}$ est l'exposant.

Le nombre d'octets utilisés pour représenter m gouverne la précision numérique.

Le nombre d'octets utilisés pour représenter e détermine les points de dépassement et soupassement numérique.

Quatre constantes méchanique

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

Constante	description
`double.xmax`	$x > 0$ le plus grand distinct de ∞.
`double.xmin`	$x > 0$ le plus petit distinct de 0.
`double.eps`	$x > 0$ le plus petit tel que $1 + x$ et 1 sont distincts.
`double.neg.eps`	$x > 0$ le plus petit tel que $1 - x$ et 1 sont distincts.

On appele

- `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
```

Propogation de l'erreur

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

$$d_h = \frac{e^h - e^{-h}}{2h},$$

où $h > 0$ est très petit.

- Mettons que l'erreurs relatives maximales de e^h et e^{-h} sont ϵ.
- Puisque $e^x = 1$ à $x = 0$, les erreurs abolues sont pareilles.
- Par une expansion Taylor,

$$d_h \approx \frac{2h \pm 2\epsilon}{2h} = 1 \pm \frac{\epsilon}{h}$$

- L'erreur relative du résultat peut être aussi grand que ϵ/h.

Expansions de Taylor et de Mercator de la fonction $\ln x$

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

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

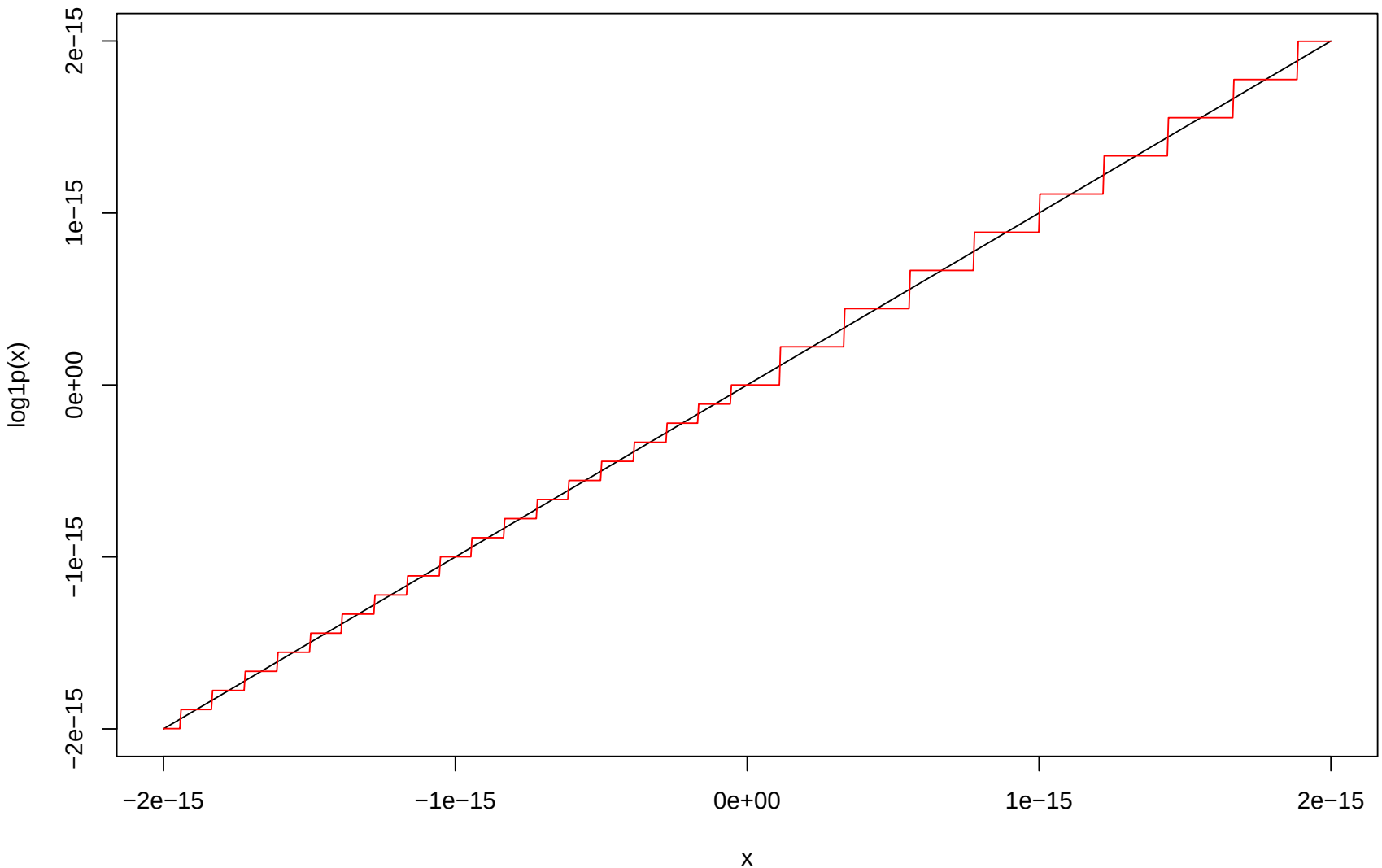
L'expansion de Mercator :

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

- Si on veux évaluer $\ln(1+x)$ quand x est petit et disponsible, on ne veut pas calculer $1+x$ comme résultat intermédiaire.
- La fonction `log1p` en R (et autres langages) évalue la fonction $f(x) = \ln(1+x)$ directement.

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

Une autre source d'erreur est la troncation

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

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

Le point plus général :

- un algorithme iteratif génére une suite de valeurs intermédiares x_N qui converge au résultat voulu $x^* = \lim_{N\to\infty} x_N$
- il faut accepter une valeur approximé x_N, $N < \infty$.

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 $|f(n)| \leq Mg(n)$.
- 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, 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 $n \times 1$,
- $O(n^3)$: pour multiplier deux matrices $n \times n$ (méthode évidente)
- $O(n^{2.81})$: pour multiplier deux matrices $n \times n$ (Algorithme de Strassen)

L'importance d'algorithmes d'ordre polynomial

- Soit n le nombre de scalaires dans l'intrant du problème.
- Distinction entre la complexité d'un algorithme est la complexité d'un problème.
- Un algorithme est d'ordre polynomial si le nombre d'opérations est $O(g(n))$, pour une polynôme $g(n)$.
- Il y a une distinction importante entre les problèmes "faisables" (pour lesquels il y a un algorithme polynomial connue pour le résoudre) et les problèmes "infaisables".
- Fonctions à sens unique et la cryptographie.
- Si un algorithme est polynomial, il est habituellement facile à prouver qu'il l'est.
  - Les polynômes sont stables pour l'addition, la multiplication et la composition : si $p(n)$ et $q(n)$ sont polynomiales, $p(n) + q(n)$, $p(n)q(n)$ et $p(q(n))$ le sont aussi.
  - (boucles, invocation des fonctions, etc.)

P, NP et NP Complet

- Une réduction : problème A n'est pas plus difficile que B ($A \leq_P B$) si A peut être réduit à B—résolu par un algorithme pour résoudre B plus un nombre d'opérations supplémentaires d'ordre polynomial.
- Classes de problèmes :
  - P : problèmes de décision (oui/no) résoluble en temps polynôme.
  - NP : problèmes de décision (oui/non) ou une réponse affirmative peut être prouvé correcte en temps polynôme.
  - NP-complet : les problèmes en NP les plus difficile (une classe d'équivalence avec plusieurs exemples connus)
  - NP-difficile : les problèmes qui sont au moins aussi difficile que les problèmes dans NP-complet.
- P=NP? est une question non-résolue : il n'y a pas d'algorithmes polynômes connus pour résoudre les problèmes en NP.

Quelques problèmes

(polynomial, NP-complet, NP-difficile)

Problème du voyageur de commerce (Travelling Salesman)

- Trouvez le trajet le plus court qui relie un ensemble de villes.
- Y a-t-il un trajet plus court que L qui relie les villes?

Optimisation linéaire en nombres entiers (Integer programming)

- Trouvez la solution optimale ou montrez qu'il n'y a pas de solution.
- (Cas spécial de variables binaires) Y a-t-il une solution faisable?

Optimisation linéaire (Linear programming)

- Trouvez la solution optimale ou montrez qu'il n'y a pas de solution.

L'évaluation des polynomes avec la méthode de Horner

Le problème est l'évaluation du polynôme $a_0 + a_1 x + \cdots + a_n x^n$.

Trois solutions

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

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

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

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

Parallelisme

Vous allez vous habituer à reconnaître deux types de problème où vous pouvez profitez des processeurs en parallèle.

Problèmes

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 indépendent
4. Évaluation d'une fonction de vraisemblance
5. Multiplication des matrices

Problèmes sans l'embarras du parallelisme

1. Méthodes iteratives d'optimisation
2. Méthodes iteratives pour trouver un point fixe
3. Simulation Markov chain Monte Carlo

Problèmes SIMD (Single Instruction, Multiple Data)