Simulation numérique avec Julia

Mickael Bestard, Emmanuel Franck

Université de Strasbourg
IRMA

23 Juin 2021

Introduction

- Julia : un langage **haut niveau** créé pour la **performance**
- Récent (2012) mais déjà largement utilisé dans le monde industriel
  - Calcul de risques (Aviva, assurance, UK)
  - Modèles économiques (Federal reserve Bank of New York) → *"about 10 times faster than its previous MATLAB implementation"*
  - Sélectionné par la Climate Modeling Alliance pour créer un modèle climatique à l'échelle de la Terre
  - Utilisé par la NASA et l'INPE (Brésil) pour planifier des missions spatiales
- utilisé à l'IRMA depuis quelques années : retour d'expérience

Contents

Julia et la simulation numérique

Préliminaire sur les types

Il y a 2 grandes familles de types en Julia :

Les types abstraits

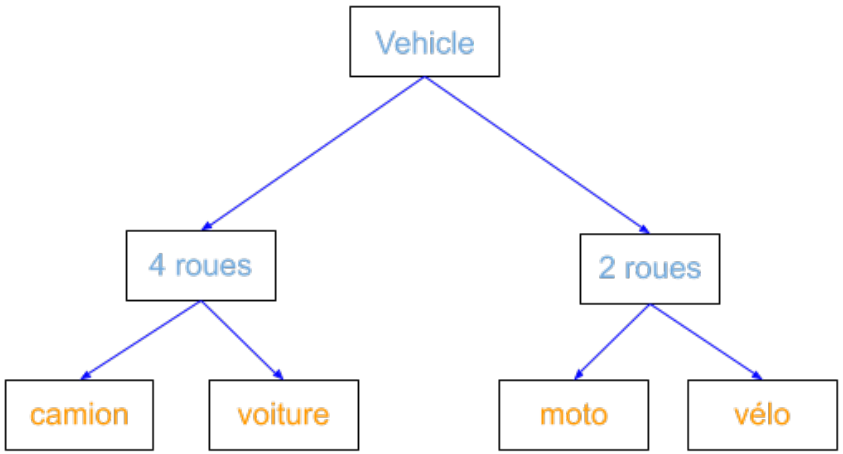
- servent à organiser le code via l'héritage
- on ne peut pas créer d'objet à partir d'eux

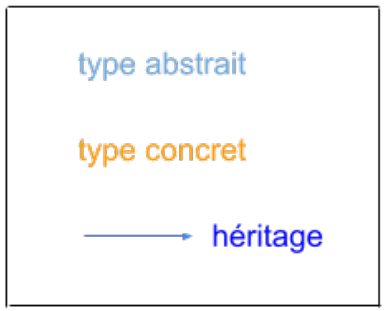
Les types concrets

- aucun type ne peut en hériter
- but principal : créer des objets

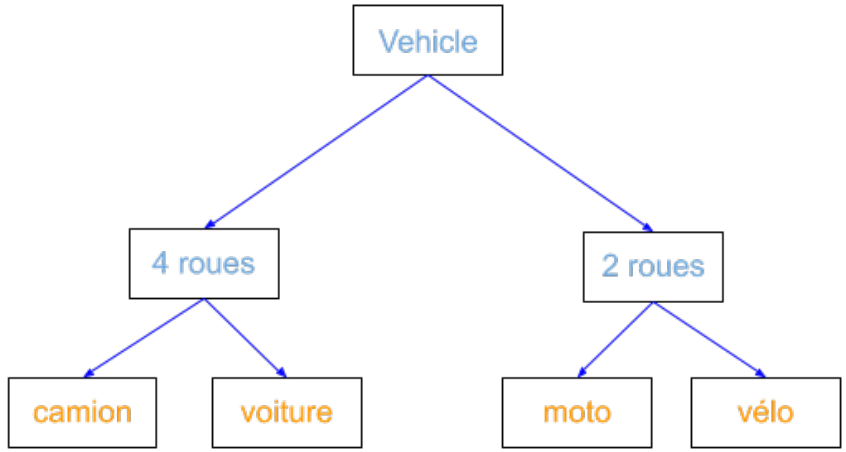
Syntaxe :

```
(1) abstract type TypeAbstrait end
(2) struct TypeConcret <: TypeAbstrait
(3) end
```





```
abstract type Vehicle end
abstract type 4roues <:Vehicle end
abstract type 2roues <:Vehicle end
struct camion <:4roues end
struct voiture <:4roues end
struct moto <:2roues end
struct velo <:2roues end
```



On peut également appeler un objet comme si c'était une fonction :

```
function(objet::Type)(x,y)
   # code utilisant x, y ainsi que les attributs de l'objet
end
```

Exemple simplifié :

```
Euler=Model(Mh,3)
for n=1:Nt
 Euler()
end
plot(Euler.x, Euler.solutions)
```

Exemple "fil rouge" : schéma aux Volumes Finis générique. On cherche à résoudre :

Equation de conservation

$$\begin{cases} \partial_t \rho(t,x) + \partial_x f(\rho(t,x)) = 0, & (t,x) \in [0,T] \times [a,b] \\ \rho(0,x) = \rho_0(x), & x \in [a,b] \end{cases} \tag{1}$$

- Maillage : $a = x_{1/2} < \cdots < x_{i-1/2} < x_{i+1/2} < \cdots < x_{N_c+1/2} = b$
- Volumes : $V_i := [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$

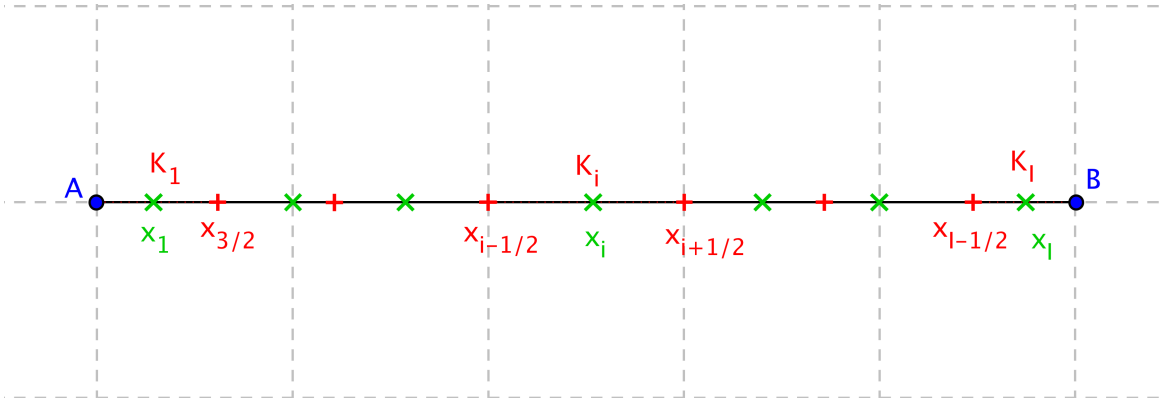


Figure: Grille Volumes Finis 1D

- Inconnue numérique **constante** par maille: $\rho_i^n := \rho_i(t_n) \approx \frac{1}{\Delta x_i} \int_{V_i} \rho(t_n, x)\, \mathrm{d}x$

Julia et la simulation numérique

Code générique d'une méthode aux volumes finis

On moyenne l'équation (1) sur un volume V_i :

$$\frac{1}{\Delta x_i}\int_{V_i}\partial_t\rho(t,x)\,\mathrm{d}x+\frac{1}{\Delta x_i}\int_{V_i}\partial_x f(\rho(t,x))\,\mathrm{d}x=0$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{\Delta x_i}\int_{V_i}\rho(t,x)\,\mathrm{d}x\right)+\frac{1}{\Delta x_i}\left(f(\rho(t,x_{i+\frac{1}{2}}))-f(\rho(t,x_{i-\frac{1}{2}}))\right)=0$$

Euler explicite : $\dfrac{\mathrm{d}\rho_i}{\mathrm{d}t}(t_n)\approx(\rho_i^{n+1}-\rho_i^n)/\Delta t$

$$\boxed{\rho_i^{n+1}=\rho_i^n-\frac{\Delta t}{\Delta x_i}\left(F_{i+\frac{1}{2}}^n-F_{i-\frac{1}{2}}^n\right)}$$

Julia et la simulation numérique

Code générique d'une méthode aux volumes finis

On moyenne l'équation (1) sur un volume V_i :

$$\frac{1}{\Delta x_i}\int_{V_i}\partial_t\rho(t,x)\,\mathrm{d}x+\frac{1}{\Delta x_i}\int_{V_i}\partial_x f(\rho(t,x))\,\mathrm{d}x=0$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{\Delta x_i}\int_{V_i}\rho(t,x)\,\mathrm{d}x\right)+\frac{1}{\Delta x_i}\left(f(\rho(t,x_{i+\frac{1}{2}}))-f(\rho(t,x_{i-\frac{1}{2}}))\right)=0$$

Euler explicite : $\dfrac{\mathrm{d}\rho_i}{\mathrm{d}t}(t_n)\approx(\rho_i^{n+1}-\rho_i^n)/\Delta t$

$$\boxed{\rho_i^{n+1}=\rho_i^n-\frac{\Delta t}{\Delta x_i}\left(F_{i+\frac{1}{2}}^n-F_{i-\frac{1}{2}}^n\right)}$$

NB: $f(\rho(t_n,x_{i+\frac{1}{2}}))$ n'est pas défini (multivalué) !

On introduit le *flux numérique* : $F_{i+\frac{1}{2}}^n=F(\rho_i^n,\rho_{i+1}^n)\simeq f(\rho(t_n,x_{i+\frac{1}{2}}))$.

Local Lax-Friedrichs

$F(\rho_L,\rho_R)=\frac{f(\rho_L)+f(\rho_R)}{2}-\frac{\lambda}{2}(\rho_R-\rho_L)$, avec $\lambda:=max\{|f'(\rho_L)|,|f'(\rho_R)|\}$

Julia et la simulation numérique

Code générique d'une méthode aux volumes finis

On moyenne l'équation (1) sur un volume V_i :

$$\frac{1}{\Delta x_i}\int_{V_i}\partial_t\rho(t,x)\,\mathrm{d}x+\frac{1}{\Delta x_i}\int_{V_i}\partial_x f(\rho(t,x))\,\mathrm{d}x=0$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{\Delta x_i}\int_{V_i}\rho(t,x)\,\mathrm{d}x\right)+\frac{1}{\Delta x_i}\left(f(\rho(t,x_{i+\frac{1}{2}}))-f(\rho(t,x_{i-\frac{1}{2}}))\right)=0$$

Euler explicite : $\dfrac{\mathrm{d}\rho_i}{\mathrm{d}t}(t_n)\approx(\rho_i^{n+1}-\rho_i^n)/\Delta t$

$$\boxed{\rho_i^{n+1}=\rho_i^n-\frac{\Delta t}{\Delta x_i}\left(F_{i+\frac{1}{2}}^n-F_{i-\frac{1}{2}}^n\right)}$$

NB: $f(\rho(t_n,x_{i+\frac{1}{2}}))$ n'est pas défini (multivalué) !

On introduit le *flux numérique* : $F_{i+\frac{1}{2}}^n=F(\rho_i^n,\rho_{i+1}^n)\simeq f(\rho(t_n,x_{i+\frac{1}{2}}))$.

Flux numérique et performances du code

Le flux numérique peut être quelconque et souvent délicat à vectoriser. Il est plus simple de ne faire que des boucles `for` et Julia s'avère alors bien plus performant que Python.

Julia et la simulation numérique

Code générique d'une méthode aux volumes finis

```
export Model, NumFlux

abstract type NumFlux end

struct NullFlux <: NumFlux
end

function (self::NullFlux)(x,vL,vR)
end

struct NullNcFlux <: NumFlux
end

function (self::NullNcFlux)(x,vL,vC,vR)
end

struct Limiting <: NumFlux
end

function (self::Limiting)(x,vC,vL,vR)
end
```

```
mutable struct Model

    ntvar     :: Int64
    npvar     :: Int64
    mh        :: Mesh
    var       :: Array{Float64, 2}
    flux      :: Array{Float64, 2}
    var_bcl   :: Vector{Float64}
    var_bcr   :: Vector{Float64}
    ncterm    :: Int64
    numflux   :: NumFlux
    ncnumflux :: NumFlux
    limiting  :: NumFlux
    theta     :: Float64
    eps       :: Float64
    order     :: Int64

    function Model(mh, ntvar)

        npvar     = 3
        var       = zeros(Float64,(mh.Nc,ntvar))
        flux      = zeros(Float64,(mh.Nc,ntvar))
        var_bcl   = zeros(Float64,ntvar)
        var_bcr   = zeros(Float64,ntvar)
        ncterm    = 0
        numflux   = NullFlux()
        ncnumflux = NullNcFlux()
        limiting  = Limiting()
        theta     = 0.0
        eps       = 0.0
        order     = 1

        new( ntvar, npvar, mh, var, flux,  var_bcl,
             var_bcr, ncterm, numflux, ncnumflux, limiting, theta, eps, order)
    end
end
```

Julia et la simulation numérique

Code générique d'une méthode aux volumes finis

```
function (self::Model)() # generic_vf_op() is replaced by model()

        x  = self.mh.centers[1]
        Fl = self.numflux(x,self.var_bcl,self.var_bcl,self.var[1,:])
        Fr = self.numflux(x,self.var[1,:],self.var_bcl,self.var[2,:])

        self.flux[1,:] = (Fr-Fl)

        for i in 2:self.mh.Nc-1
           x  = self.mh.centers[i]
           Fl = self.numflux(x,self.var[i,:],self.var[i-1,:])
           Fr = self.numflux(x,self.var[i+1,:],self.var[i,:])
           self.flux[i,:] = (Fr-Fl)
        end

        x  = self.mh.centers[end]
        Fl = self.numflux(x,self.var[end,:],self.var[end-1,:])
        Fr = self.numflux(x,self.var_bcr,self.var[end,:])

        self.flux[end,:] = (Fr-Fl)
    end
end
```

Figure: Intégrateur Volumes Finis générique

- Toute instance de la classe `Model` devient un intégrateur FV
- On n'a qu'un seul intégrateur générique à coder
- Il suffit de l'appeler depuis le modèle voulu pour changer de cas-test

Exemple simplifié :

```
Euler=Model(Mh,3)
for n=1:Nt
 Euler()
end
plot(Euler.x, Euler.solutions)
```

Julia et la simulation numérique

Comparaison Julia / Python

- Équation d'Euler : $\partial_t \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho u \\ \rho u^2 + p \\ E u + p u \end{pmatrix} = 0$
- Discrétisation spatiale : Volumes Finis (ordre 1)
- Discrétisation temporelle : Euler explicite (CFL = 0.5)

Python

```python
def generic_vf_op(self):

    x = self.mh.centers[0]
    Fr = self.numflux(x,self.var[:,0],self.var[:,1])
    Fl = self.numflux(x,self.var_bcl,self.var[:,0])

    self.flux[:,0] = (Fr-Fl)

    for i in range(1,self.mh.Nc-1):
        x = self.mh.centers[i]
        Fr = self.numflux(x,self.var[:,i],self.var[:,i+1])
        Fl = self.numflux(x,self.var[:,i-1],self.var[:,i])
        self.flux[:,i] = (Fr-Fl)

    x = self.mh.centers[self.mh.Nc-1]
    Fr = self.numflux(x,self.var[:,self.mh.Nc-1],self.var_bcr)
    Fl = self.numflux(x,self.var[:,self.mh.Nc-2],self.var[:,self.mh.Nc-1])

    self.flux[:,self.mh.Nc-1] = (Fr-Fl)
```

Julia

```julia
function (self::Model)() # generic_vf_op() is replaced by model()

        x  = self.mh.centers[1]
        Fl = self.numflux(x,self.var_bcl,self.var_bcl,self.var[1,:])
        Fr = self.numflux(x,self.var[1,:],self.var_bcl,self.var[2,:])

        self.flux[1,:] = (Fr-Fl)

        for i in 2:self.mh.Nc-1
           x  = self.mh.centers[i]
           Fl = self.numflux(x,self.var[i,:],self.var[i-1,:])
           Fr = self.numflux(x,self.var[i+1,:],self.var[i,:])
           self.flux[i,:] = (Fr-Fl)
        end

        x  = self.mh.centers[end]
        Fl = self.numflux(x,self.var[end,:],self.var[end-1,:])
        Fr = self.numflux(x,self.var_bcr,self.var[end,:])

        self.flux[end,:] = (Fr-Fl)
    end
end
```

Julia et la simulation numérique

Comparaison Julia / Python

- Équation d'Euler : $\partial_t \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho u \\ \rho u^2 + p \\ Eu + pu \end{pmatrix} = 0$
- Discrétisation spatiale : Volumes Finis (ordre 1)
- Discrétisation temporelle : Euler explicite (CFL = 0.5)

Python

```
while time < Tf:

        dt = cfl*Mh.h/euler.reduction(local_lax_dt,0.0)
        diags_time[0,n_iter] = time
        diags_time[1,n_iter] = dt

        if time+ dt> Tf:
            dt = Tf-time

        euler.bc_neumann()
        euler.explicit_o1(euler.generic_vf_op,dt)

        n_iter = n_iter +1
        time = time + dt
        if n_iter % ntp == 0:
            ref = euler.diagnostic(gamma,1,time,sod)
```

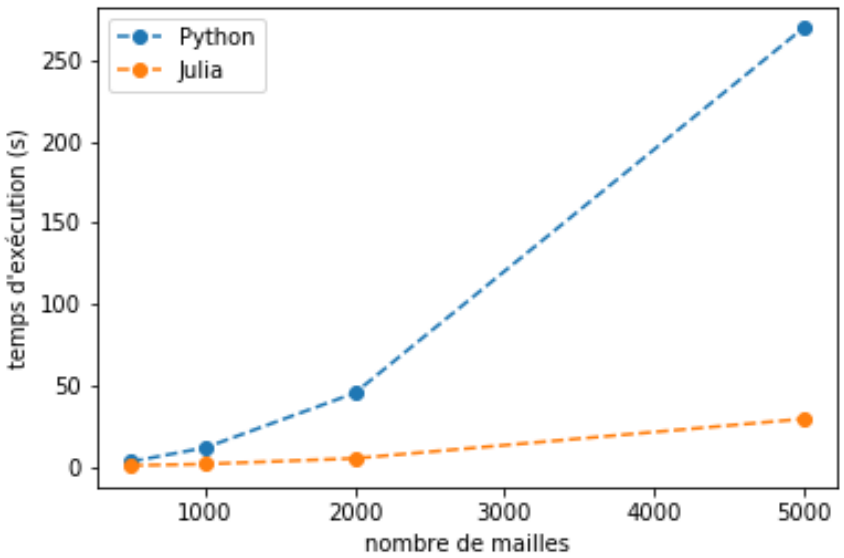
Julia

```
while time < Tf

    push!(times,time)
    push!(timesteps,dt)
    dt = cfl*Mh.h/reduction(euler, compute_cfl, 0.0)

    if time + dt >  Tf
        dt = Tf-time
    end

    bc_neumann(euler)
    vmax =explicit_o1(euler, dt, order)

    n_iter += 1
    time   += dt
    if n_iter % ntp == 0
        ref = diagnostic_euler(euler, gamma, pinf, 1, time, init_data)
    end
end
```

Julia et la simulation numérique

Comparaison Julia / Python

- Équation d'Euler : $\partial_t \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho u \\ \rho u^2 + p \\ E u + p u \end{pmatrix} = 0$
- Discrétisation spatiale : Volumes Finis (ordre 1)
- Discrétisation temporelle : Euler explicite (CFL = 0.5)



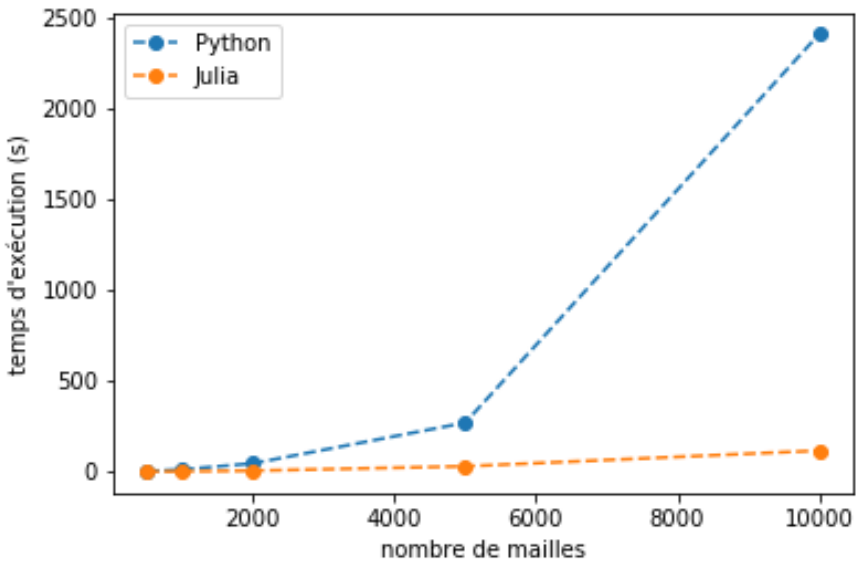
Temps d'exécution (en secondes)

$nb_{mailles}$	Python	Julia
500	3	0.6
1000	11.63	1.5
2000	45.3	5.
5000	270	29.3

Julia et la simulation numérique

Comparaison Julia / Python

- Équation d'Euler : $\partial_t \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho u \\ \rho u^2 + p \\ E u + p u \end{pmatrix} = 0$
- Discrétisation spatiale : Volumes Finis (ordre 1)
- Discrétisation temporelle : Euler explicite (CFL = 0.5)



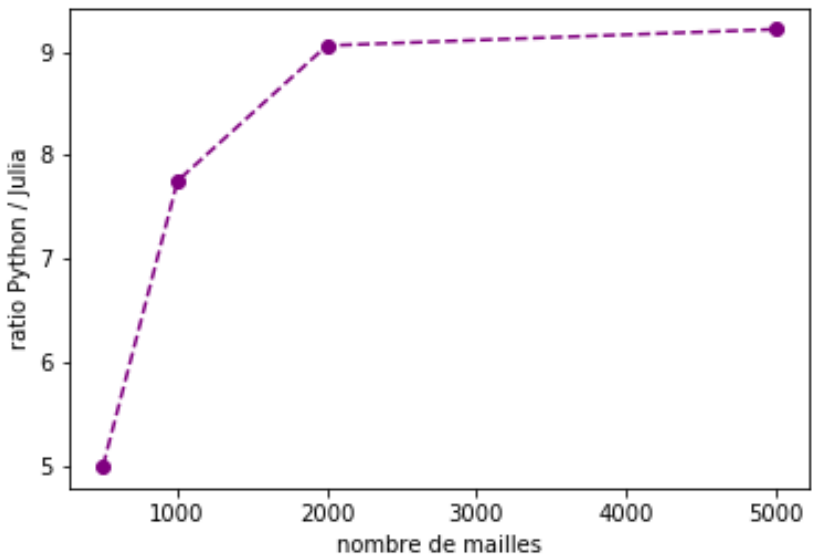
Temps d'exécution (en secondes)

$nb_{mailles}$	Python	Julia
500	3	0.6
1000	11.63	1.5
2000	45.3	5.
5000	270	29.3
10000	2415	116

Julia et la simulation numérique

Comparaison Julia / Python

- Équation d'Euler : $\partial_t \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho u \\ \rho u^2 + p \\ E u + p u \end{pmatrix} = 0$
- Discrétisation spatiale : Volumes Finis (ordre 1)
- Discrétisation temporelle : Euler explicite (CFL = 0.5)



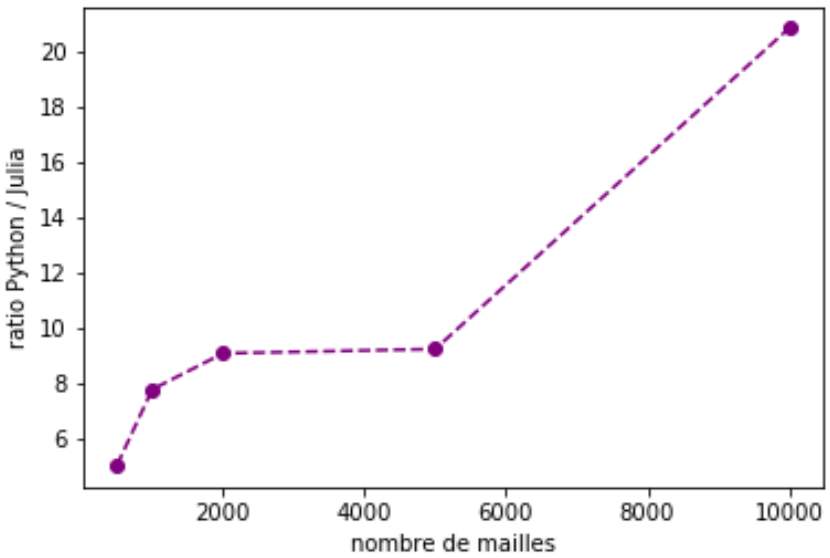
Rapports entre les temps d'exécutions

$nb_{mailles}$	Python / Julia
500	5
1000	7.75
2000	9.06
5000	9.21

Julia et la simulation numérique

Comparaison Julia / Python

- Équation d'Euler : $\partial_t \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho u \\ \rho u^2 + p \\ Eu + pu \end{pmatrix} = 0$
- Discrétisation spatiale : Volumes Finis (ordre 1)
- Discrétisation temporelle : Euler explicite (CFL = 0.5)



Rapports entre les temps d'exécutions

$nb_{mailles}$	Python / Julia
500	5
1000	7.75
2000	9.06
5000	9.21
10000	20.8

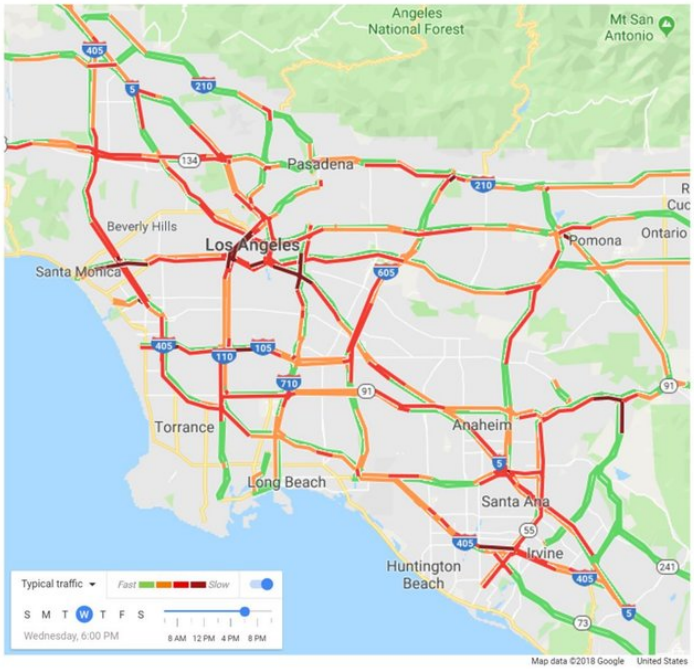
Simulation et contrôle d'un modèle fluide de trafic routier

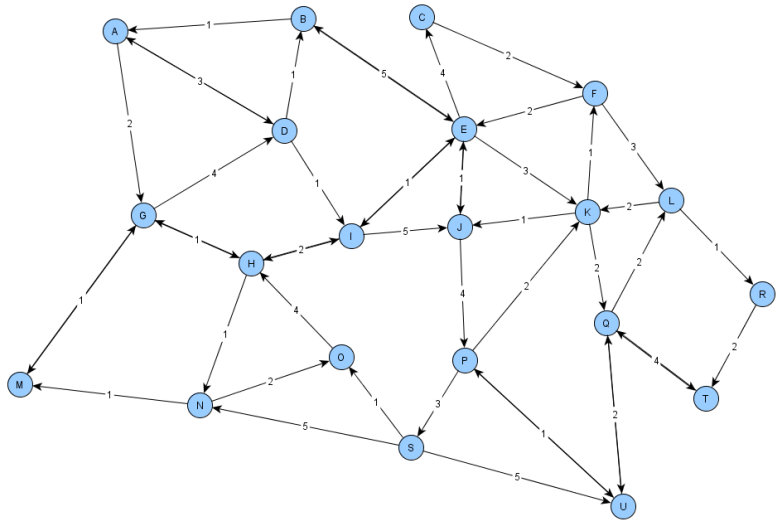
Comparaison Julia / Python

Objectif principal : outil d'aide à la décision dans un contexte de gestion de crise impliquant du trafic routier.

Travail en collaboration avec E.Franck, L.Navoret et Y.Privat.

- **Prédire** le trafic routier via un modèle fluide
  B. Piccoli M. Garavello. *Traffic flow on networks*. Ed. by Applied Mathematics. American Institute of Mathematical Sciences, 2006
- **Contrôler** la circulation par des barrages aux jonctions
- **Désencombrer** des voies d'évacuation / intervention





Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$

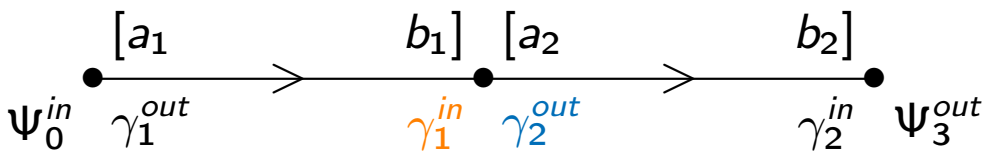
Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** $\rightarrow$ quid des jonctions ?

Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** $\rightarrow$ quid des jonctions ?
- Couplage des arêtes par CL de type Neumann

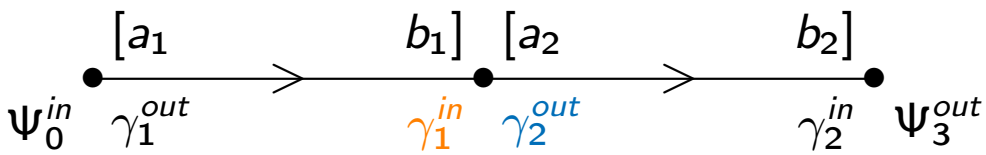
$$\max_{\gamma^{\text{in}} \in \Omega_{\boldsymbol{u}}} \sum_{i \in \text{ingoing}} \gamma_i^{\text{in}}$$



Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** $\rightarrow$ quid des jonctions ?
- Couplage des arêtes par CL de type Neumann

$$\max_{\gamma^{\text{in}} \in \Omega_{\boldsymbol{u}}} \sum_{i \in \text{ingoing}} \gamma_i^{\text{in}}$$

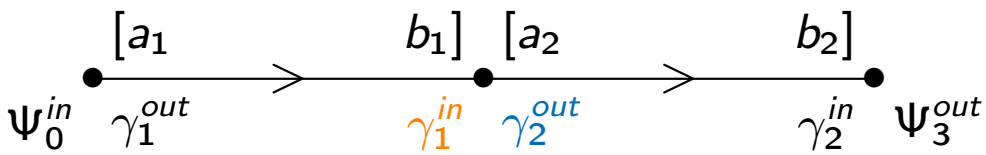


- $\gamma_i^{\text{in}}(t) = f(\rho_i(t, b_i))$: flux **entrant** à la jonction au temps t.

Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** $\rightarrow$ quid des jonctions ?
- Couplage des arêtes par CL de type Neumann

$$\max_{\gamma^{\text{in}} \in \Omega_{\boldsymbol{u}}} \sum_{i \in \text{ingoing}} \gamma_i^{\text{in}}$$

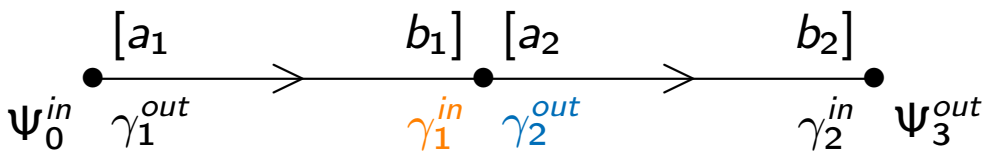


- $\gamma_i^{\text{in}}(t) = f(\rho_i(t, b_i))$: flux **entrant** à la jonction au temps t.
- $\gamma_j^{\text{out}}(t) = f(\rho_j(t, a_i))$: flux **sortant** de la jonction au temps t.

Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** $\rightarrow$ quid des jonctions ?
- Couplage des arêtes par CL de type Neumann

$$\max_{\gamma^{\text{in}} \in \Omega_{\boldsymbol{u}}} \sum_{i \in \text{ingoing}} \gamma_i^{\text{in}}$$

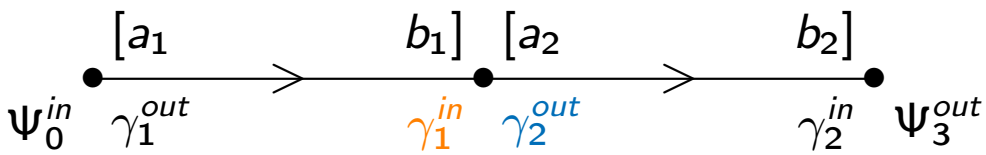


- $\gamma_i^{\text{in}}(t) = f(\rho_i(t, b_i))$: flux **entrant** à la jonction au temps t.
- $\gamma_j^{\text{out}}(t) = f(\rho_j(t, a_i))$: flux **sortant** de la jonction au temps t.
- $A = (\alpha_{ji})_{\substack{j \in \textbf{outgoing} \\ i \in \textbf{ingoing}}}$, avec α_{ji} le % de conducteurs faisant $i \rightarrow j$: $\boxed{\gamma^{\text{out}} = A\gamma^{\text{in}}}$

Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** → quid des jonctions ?
- Couplage des arêtes par CL de type Neumann

$$\max_{\gamma^{\text{in}} \in \Omega_{\boldsymbol{u}}} \sum_{i \in \text{ingoing}} \gamma_i^{\text{in}}$$



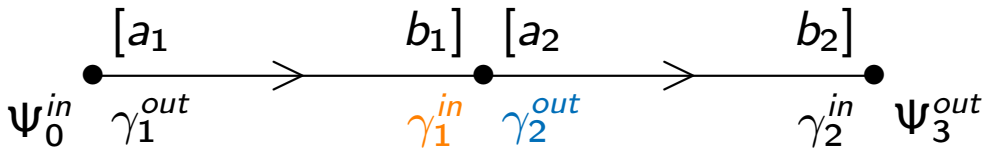
- $\gamma_i^{\text{in}}(t) = f(\rho_i(t, b_i))$: flux **entrant** à la jonction au temps t.
- $\gamma_j^{\text{out}}(t) = f(\rho_j(t, a_i))$: flux **sortant** de la jonction au temps t.
- $A = (\alpha_{ji})_{\substack{j \in \text{outgoing} \\ i \in \text{ingoing}}}$, avec α_{ji} le % de conducteurs faisant $i \to j$: $\boxed{\gamma^{\text{out}} = A\gamma^{\text{in}}}$

- $\boldsymbol{u} \in [0,1]^N$ le **contrôle** : blockage total ou partiel des routes sortantes des jonctions.

Prédire

- Modèle LWR : $\begin{cases} \partial_t \rho + \partial_x f(\rho) = 0, & t \in (0,T),\ x \in (0,L) \\ \rho(t=0,x) = \rho_0(x), & t \in (0,T) \end{cases}$
- Construction d'un graphe avec **une équation par arête** $\rightarrow$ quid des jonctions ?
- Couplage des arêtes par CL de type Neumann

$$\max_{\gamma^{\text{in}} \in \Omega_{\boldsymbol{u}}} \sum_{i \in \text{ingoing}} \gamma_i^{\text{in}}$$



  - $\gamma_i^{\text{in}}(t) = f(\rho_i(t, b_i))$: flux **entrant** à la jonction au temps t.
  - $\gamma_j^{\text{out}}(t) = f(\rho_j(t, a_i))$: flux **sortant** de la jonction au temps t.
  - $A = (\alpha_{ji})_{\substack{j \in \text{outgoing} \\ i \in \text{ingoing}}}$, avec α_{ji} le % de conducteurs faisant $i \rightarrow j$: $\boxed{\gamma^{\text{out}} = A\gamma^{\text{in}}}$
- $\boldsymbol{u} \in [0,1]^N$ le **contrôle** : blockage total ou partiel des routes sortantes des jonctions.
- $\Omega_{\boldsymbol{u}} = \{\gamma^{\text{in}} \mid 0 \leq \gamma_i^{\text{in}} \leq \gamma_i^{\max} \text{ et } 0 \leq \gamma_j^{\text{out}} \leq (1 - \boldsymbol{u}_j)\gamma_j^{\max}\}$

Prédire

- Modèle LWR discrétisé par un schéma aux volumes finis :

$$\begin{cases} \frac{\mathrm{d}\hat{\boldsymbol{\rho}}}{\mathrm{d}t} = f^{\mathsf{FV}}(\hat{\boldsymbol{\rho}}, \boldsymbol{\gamma}), & t \in (0, T), \\ \boldsymbol{\gamma} = \phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u}), & t \in (0, T), \\ (\hat{\boldsymbol{\rho}}(t=0), \boldsymbol{\gamma}(t=0)) = (\hat{\boldsymbol{\rho}}_0, \boldsymbol{\gamma}_0)\,. \end{cases}$$

  - $\hat{\boldsymbol{\rho}} = \frac{1}{\Delta x} \int \boldsymbol{\rho}(t, x)\, \mathrm{d}x$ *(inconnue numérique)*
  - $\phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u})$ donne la solution du problème d'optimisation aux jonctions

Prédire

- Modèle LWR discrétisé par un schéma aux volumes finis :

$$\begin{cases} \frac{\mathrm{d}\hat{\boldsymbol{\rho}}}{\mathrm{d}t} = f^{\mathsf{FV}}(\hat{\boldsymbol{\rho}}, \boldsymbol{\gamma}), & t \in (0, T), \\ \boldsymbol{\gamma} = \phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u}), & t \in (0, T), \\ (\hat{\boldsymbol{\rho}}(t=0), \boldsymbol{\gamma}(t=0)) = (\hat{\boldsymbol{\rho}}_0, \boldsymbol{\gamma}_0)\,. \end{cases}$$

  - $\hat{\boldsymbol{\rho}} = \frac{1}{\Delta x} \int \boldsymbol{\rho}(t, x)\, \mathrm{d}x$ *(inconnue numérique)*
  - $\phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u})$ donne la solution du problème d'optimisation aux jonctions

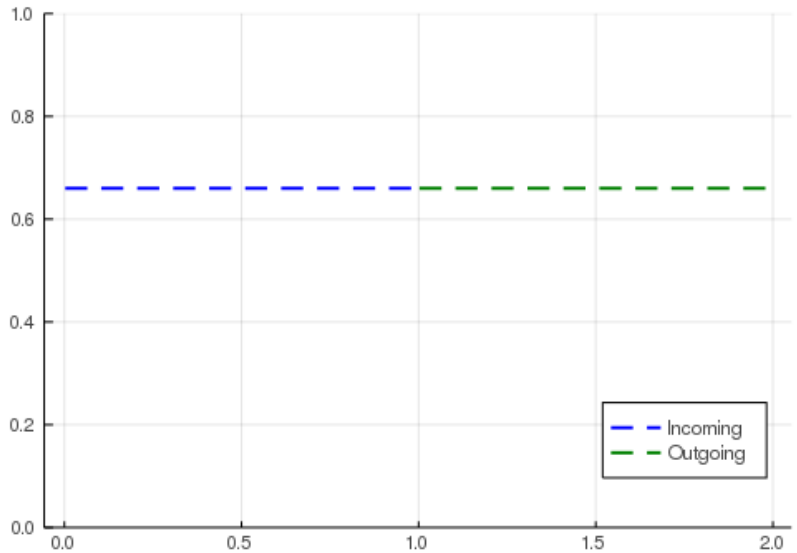


Figure: Une route large connectée à une route étroite

Prédire

- Modèle LWR discrétisé par un schéma aux volumes finis :

$$\begin{cases} \frac{\mathrm{d}\hat{\boldsymbol{\rho}}}{\mathrm{d}t} = f^{\mathsf{FV}}(\hat{\boldsymbol{\rho}}, \boldsymbol{\gamma}), & t \in (0, T), \\ \boldsymbol{\gamma} = \phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u}), & t \in (0, T), \\ (\hat{\boldsymbol{\rho}}(t=0), \boldsymbol{\gamma}(t=0)) = (\hat{\boldsymbol{\rho}}_0, \boldsymbol{\gamma}_0)\,. \end{cases}$$

  - $\hat{\boldsymbol{\rho}} = \frac{1}{\Delta x} \int \boldsymbol{\rho}(t, x)\, \mathrm{d}x$ *(inconnue numérique)*
  - $\phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u})$ donne la solution du problème d'optimisation aux jonctions

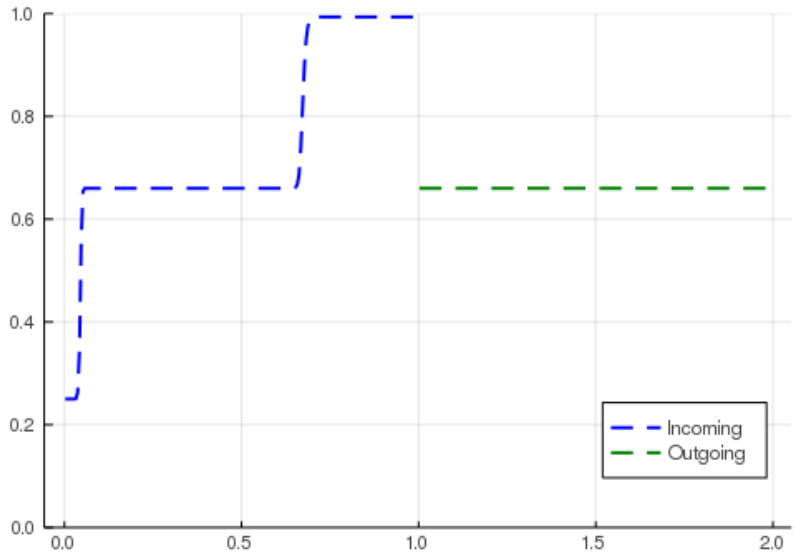


Figure: Création d'une congestion devant la jonction

Prédire

- Modèle LWR discrétisé par un schéma aux volumes finis :

$$\begin{cases} \frac{\mathrm{d}\hat{\boldsymbol{\rho}}}{\mathrm{d}t} = f^{\mathsf{FV}}(\hat{\boldsymbol{\rho}}, \boldsymbol{\gamma}), & t \in (0, T), \\ \boldsymbol{\gamma} = \phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u}), & t \in (0, T), \\ (\hat{\boldsymbol{\rho}}(t=0), \boldsymbol{\gamma}(t=0)) = (\hat{\boldsymbol{\rho}}_0, \boldsymbol{\gamma}_0)\,. \end{cases}$$

  - $\hat{\boldsymbol{\rho}} = \frac{1}{\Delta x} \int \boldsymbol{\rho}(t, x)\, \mathrm{d}x$ *(inconnue numérique)*
  - $\phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u})$ donne la solution du problème d'optimisation aux jonctions

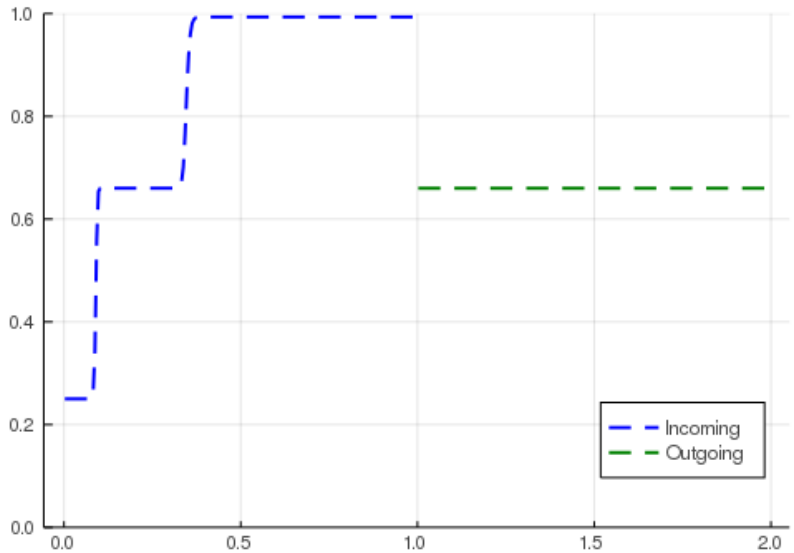


Figure: Propagation d'une onde de choc

Prédire

- Modèle LWR discrétisé par un schéma aux volumes finis :

$$\begin{cases} \frac{\mathrm{d}\hat{\boldsymbol{\rho}}}{\mathrm{d}t} = f^{\mathsf{FV}}(\hat{\boldsymbol{\rho}}, \boldsymbol{\gamma}), & t \in (0, T), \\ \boldsymbol{\gamma} = \phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u}), & t \in (0, T), \\ (\hat{\boldsymbol{\rho}}(t=0), \boldsymbol{\gamma}(t=0)) = (\hat{\boldsymbol{\rho}}_0, \boldsymbol{\gamma}_0)\,. \end{cases}$$

  - $\hat{\boldsymbol{\rho}} = \frac{1}{\Delta x} \int \boldsymbol{\rho}(t, x)\, \mathrm{d}x$ *(inconnue numérique)*
  - $\phi(\hat{\boldsymbol{\rho}}, \boldsymbol{u})$ donne la solution du problème d'optimisation aux jonctions

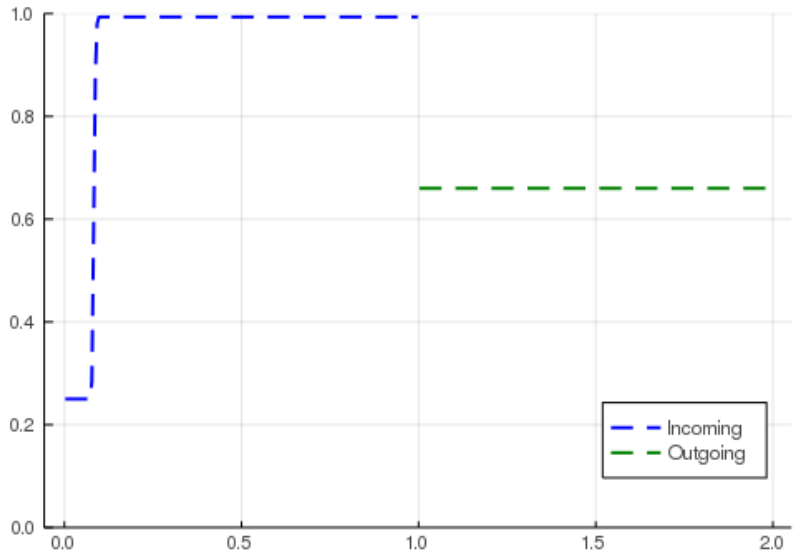


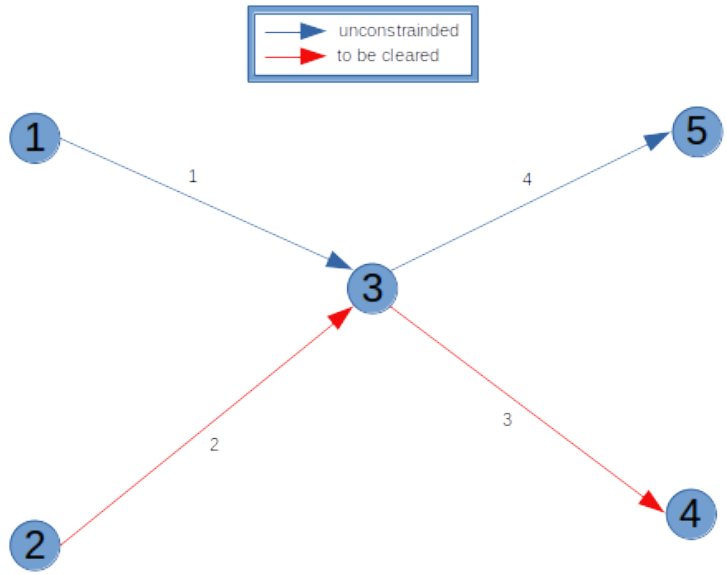
Figure: Propagation d'une onde de choc

Contrôler

- Critère : on veut minimiser la densité en temps final le long d'un chemin particulier

$$J(\boldsymbol{u}) = \sum_{k \in \text{chemin}} \hat{\rho}_k(T;\ u_k)$$

- Contrôle : valeurs admissibles pour les CL $\rightarrow$ dépendance indirecte vis-à-vis de ρ
  - Méthode adjointe (problème dual) double le coût du calcul
  - Descente de gradient : différentiation automatique



Simulation et contrôle d'un modèle fluide de trafic routier

Problème de contrôle

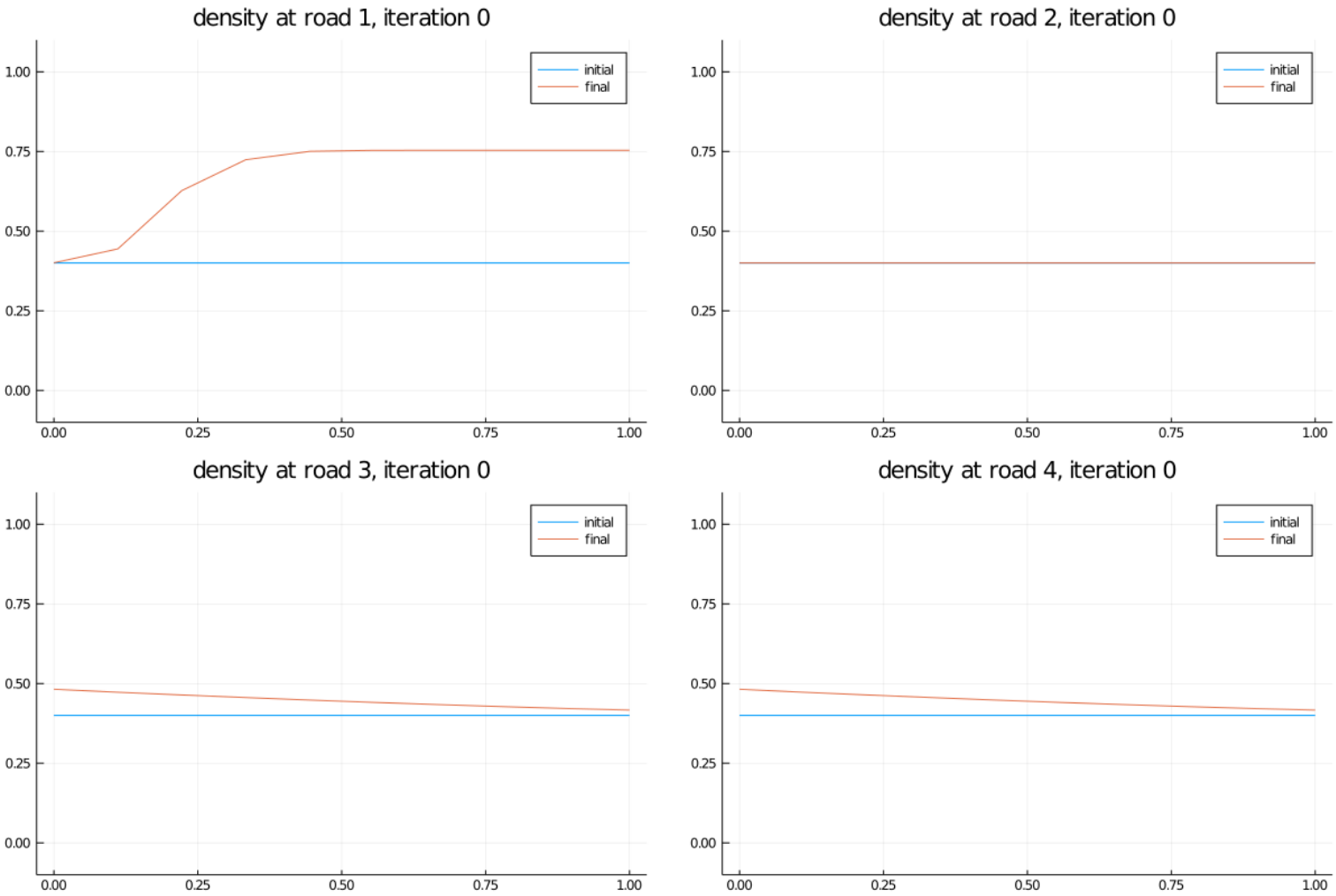


Figure: Without control : min(J) = 8.46

Simulation et contrôle d'un modèle fluide de trafic routier

Problème de contrôle

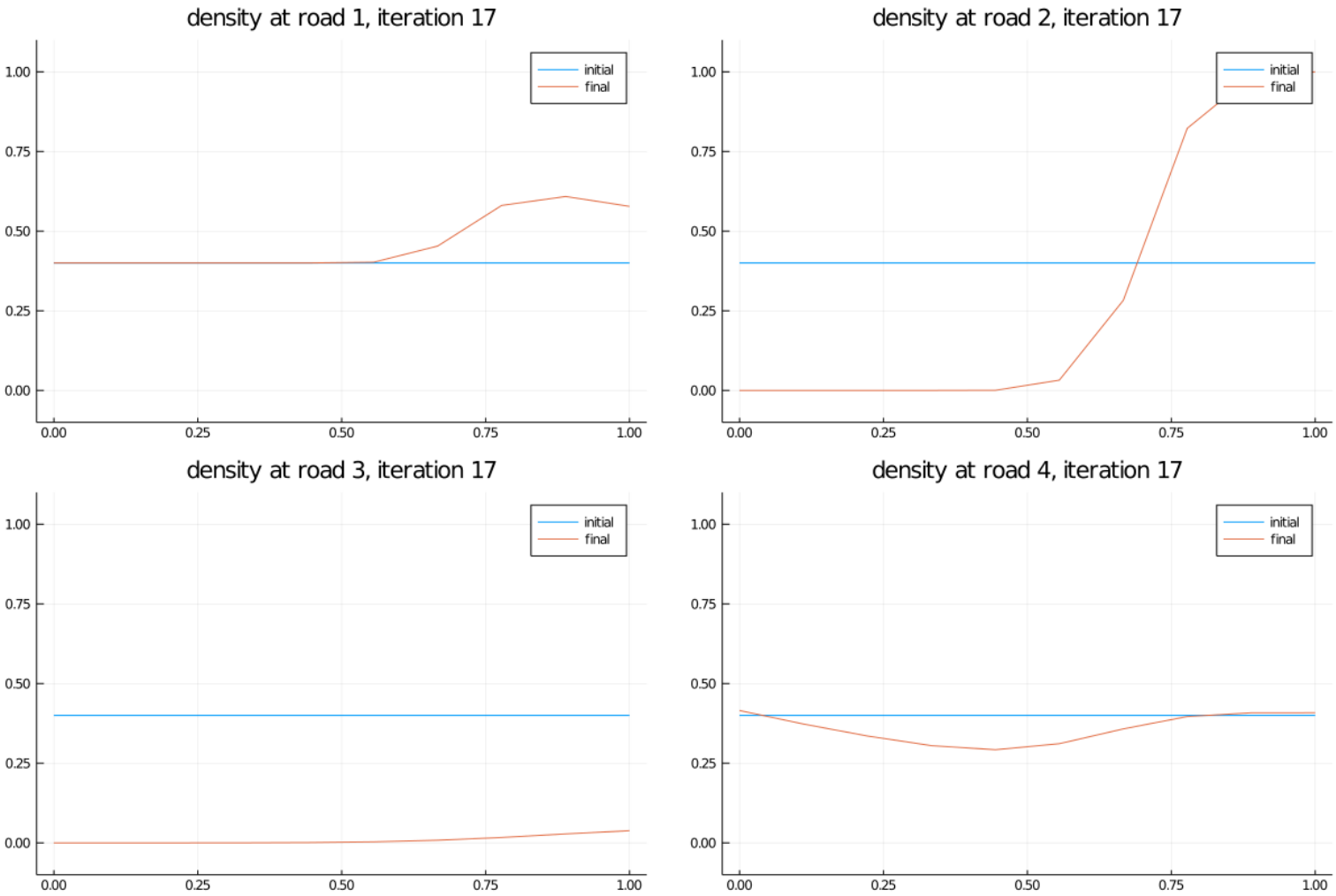


Figure: With control : min(J) = 3.22

Pour 10 mailles par route : $\sim 4.5 \times T$ secondes / itération

Conclusion

Avantages

- Facile à prendre en main (syntaxe proche Python/Matlab)
- Performances (langage compilé)
- Souple (compilation à la volée $\rightarrow$ REPL)
- Communauté active (forums, congrès, $\cdots$)

Conclusion

Avantages

- Facile à prendre en main (syntaxe proche Python/Matlab)
- Performances (langage compilé)
- Souple (compilation à la volée $\rightarrow$ REPL)
- Communauté active (forums, congrès, $\cdots$)

Inconvénients

- Langage encore récent $\rightarrow$ communauté moins grande que C/C++/Python/$\cdots$
- Perte drastique de simplicité si debug technique $\rightarrow$ besoin d'ingés au labo

Conclusion

Avantages

- Facile à prendre en main (syntaxe proche Python/Matlab)
- Performances (langage compilé)
- Souple (compilation à la volée $\rightarrow$ REPL)
- Communauté active (forums, congrès, $\cdots$)

Inconvénients

- Langage encore récent $\rightarrow$ communauté moins grande que C/C++/Python/$\cdots$
- Perte drastique de simplicité si debug technique $\rightarrow$ besoin d'ingés au labo

---

Merci pour votre attention !
