Génération de trajectoires automobiles sur circuit

Pierre Gueguen
n° SCEI : 13094

MPI 2024

Comment générer des trajectoires optimales dans le cadre spécifique des séances de qualifications à l'aide d'un algorithme génétique ?

I – Représentations numériques

II – Algorithme génétique

III – Premiers résultats et critiques

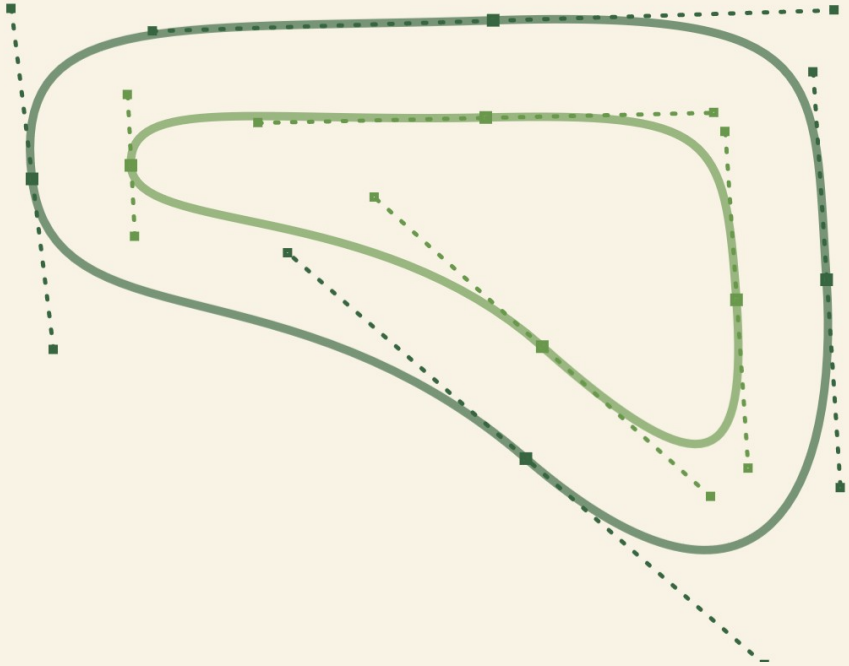
IV – Profil de vitesse

V – Conclusions

I – Représentations numériques

a) Circuit

1.



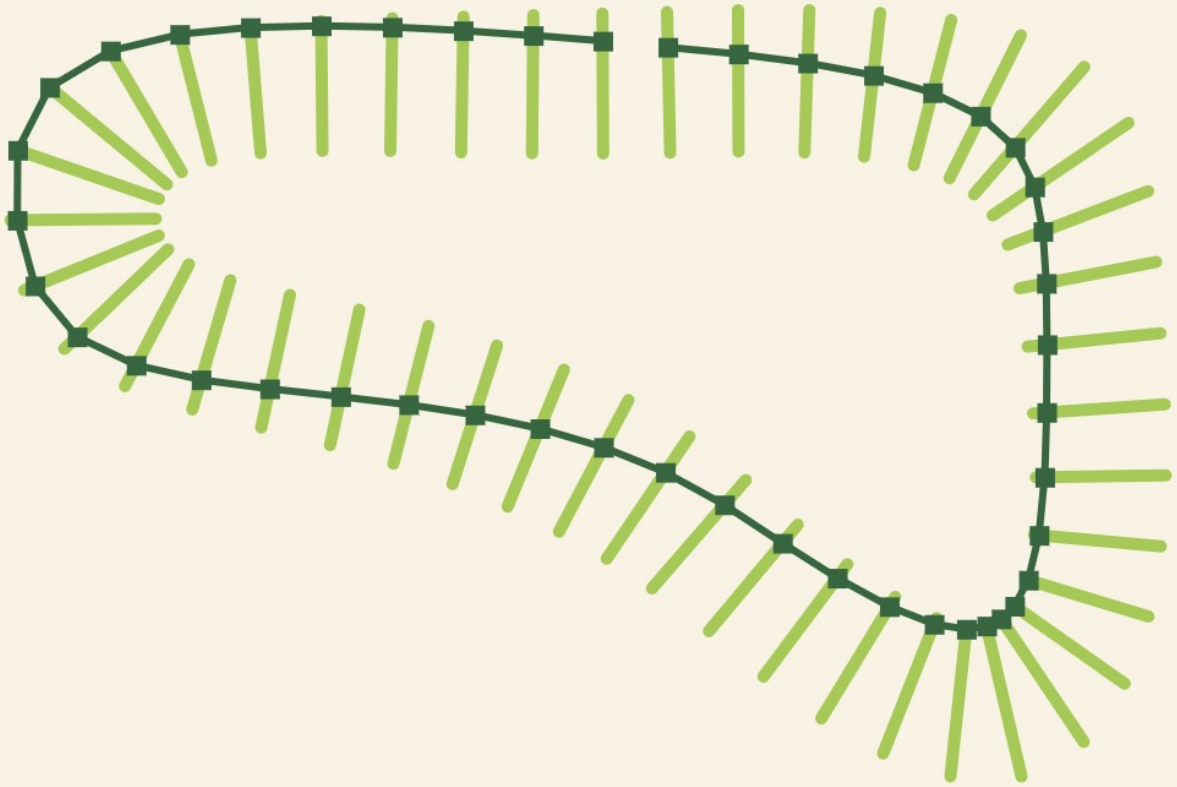
Pointage des deux bords de piste “à la main”

2.

Circuit representé par une succession de segments transversaux

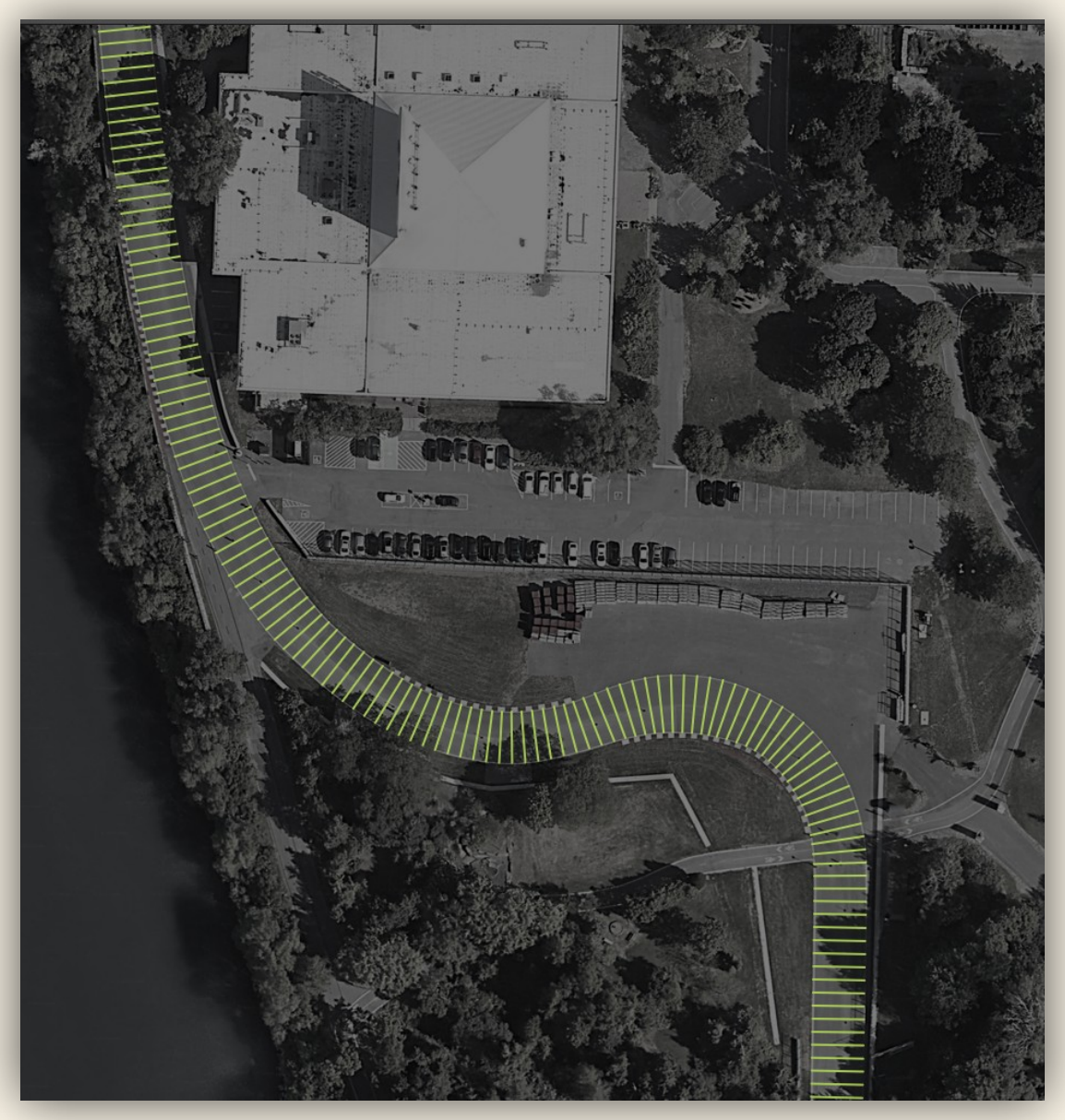
I – Représentations numériques

b) Trajectoires



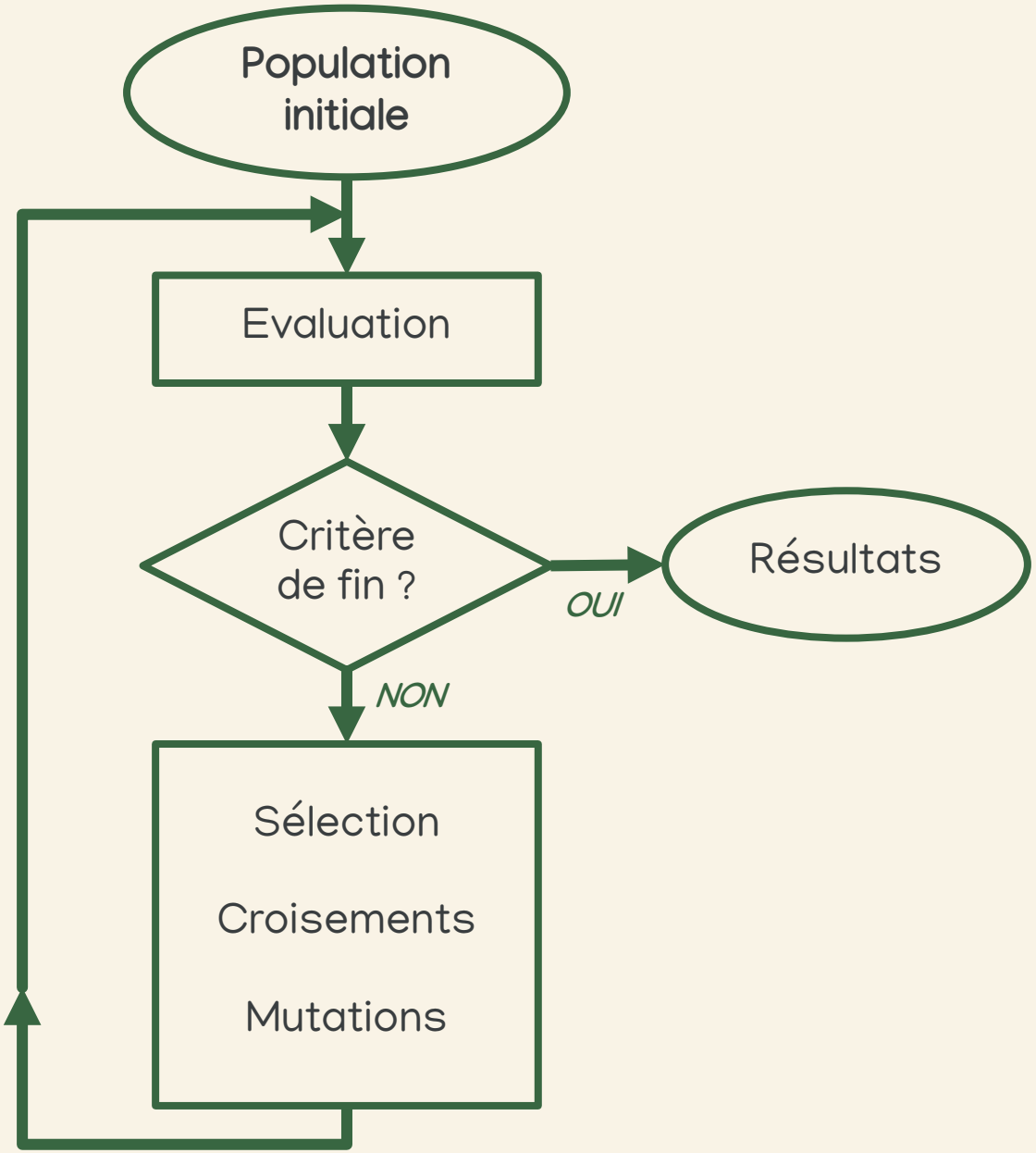
I – Représentations numériques

c) Circuit Villeneuve



II – Algorithme génétique

a) Principe

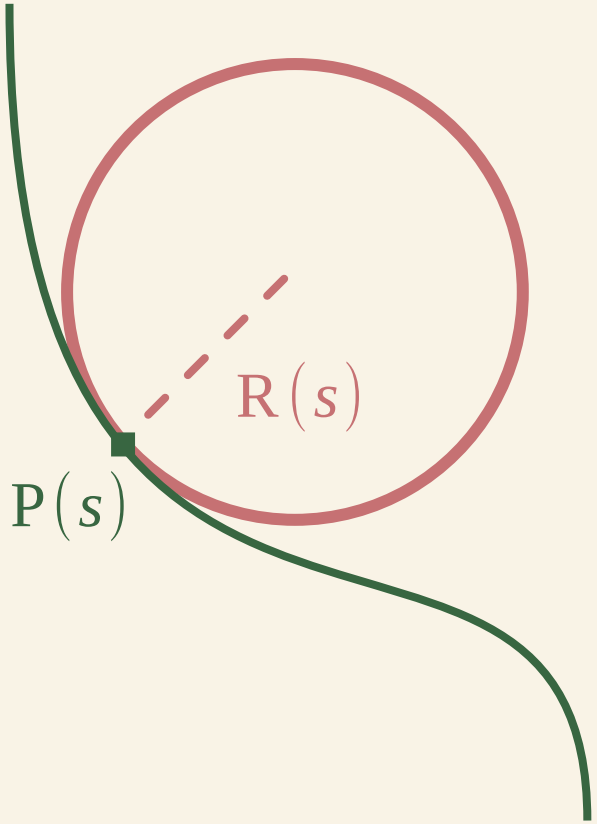


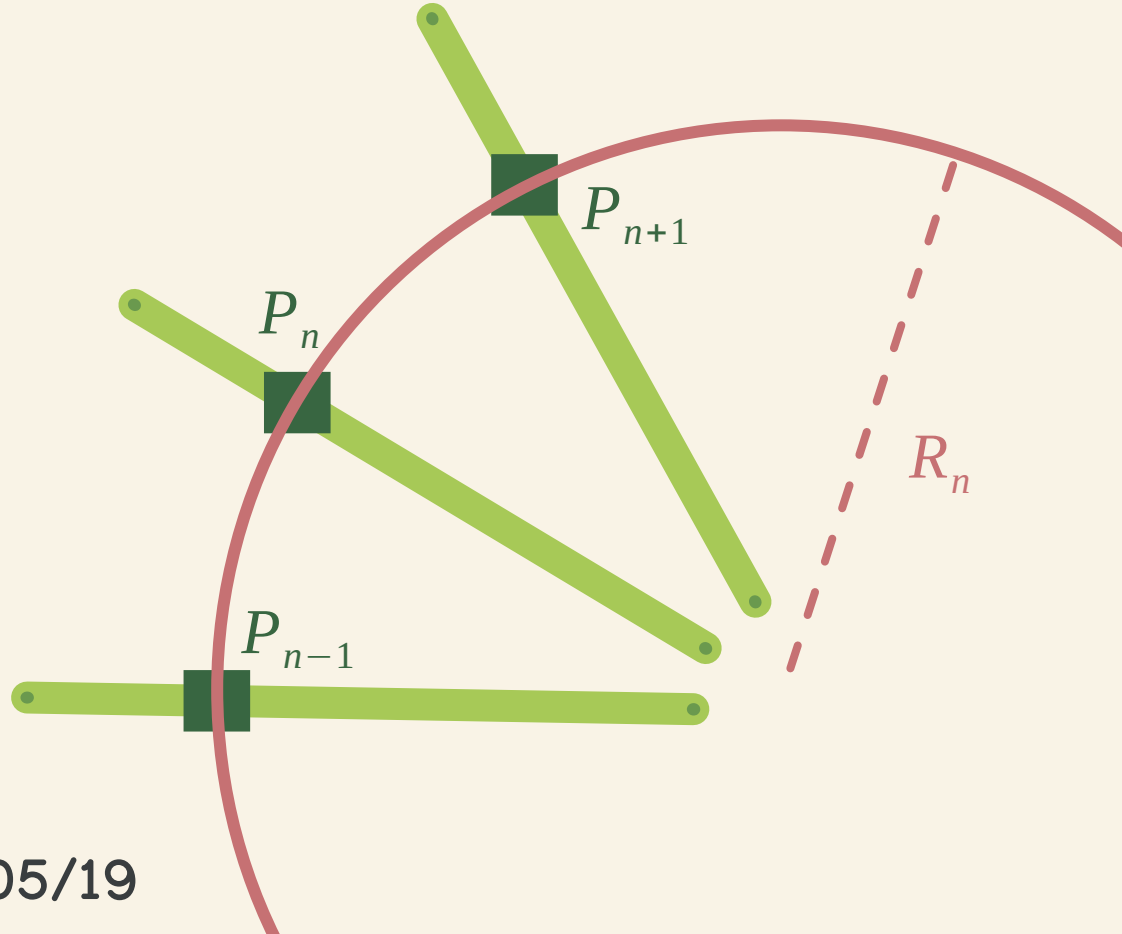
II – Algorithme génétique

b) Evaluation de la courbure

Continu : $\oint_{Traj} \frac{ds}{\mathrm{R}(s)}$

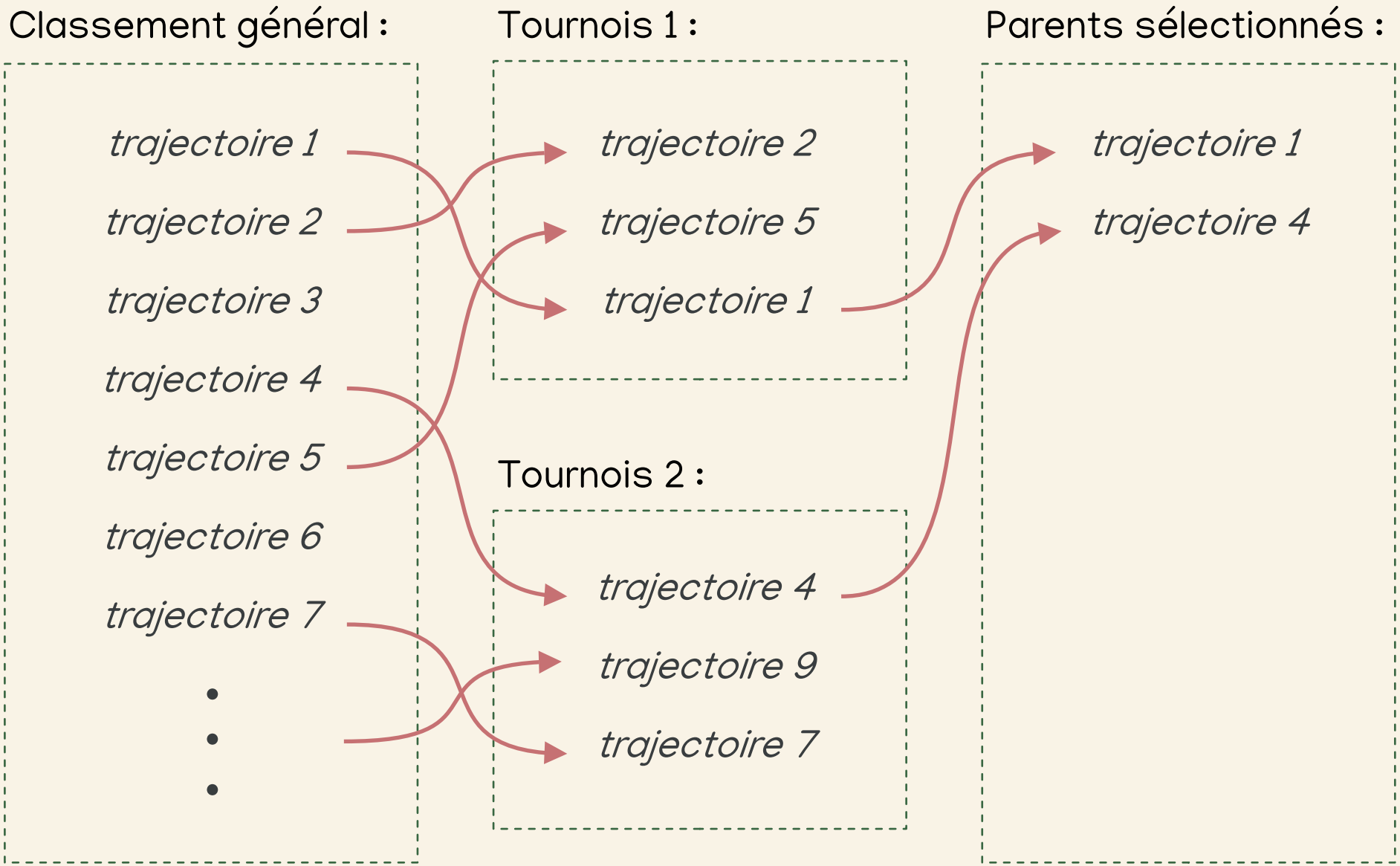
Discrétisation : $\sum_{Traj} \frac{\|P_n - P_{n+1}\|}{R_n}$





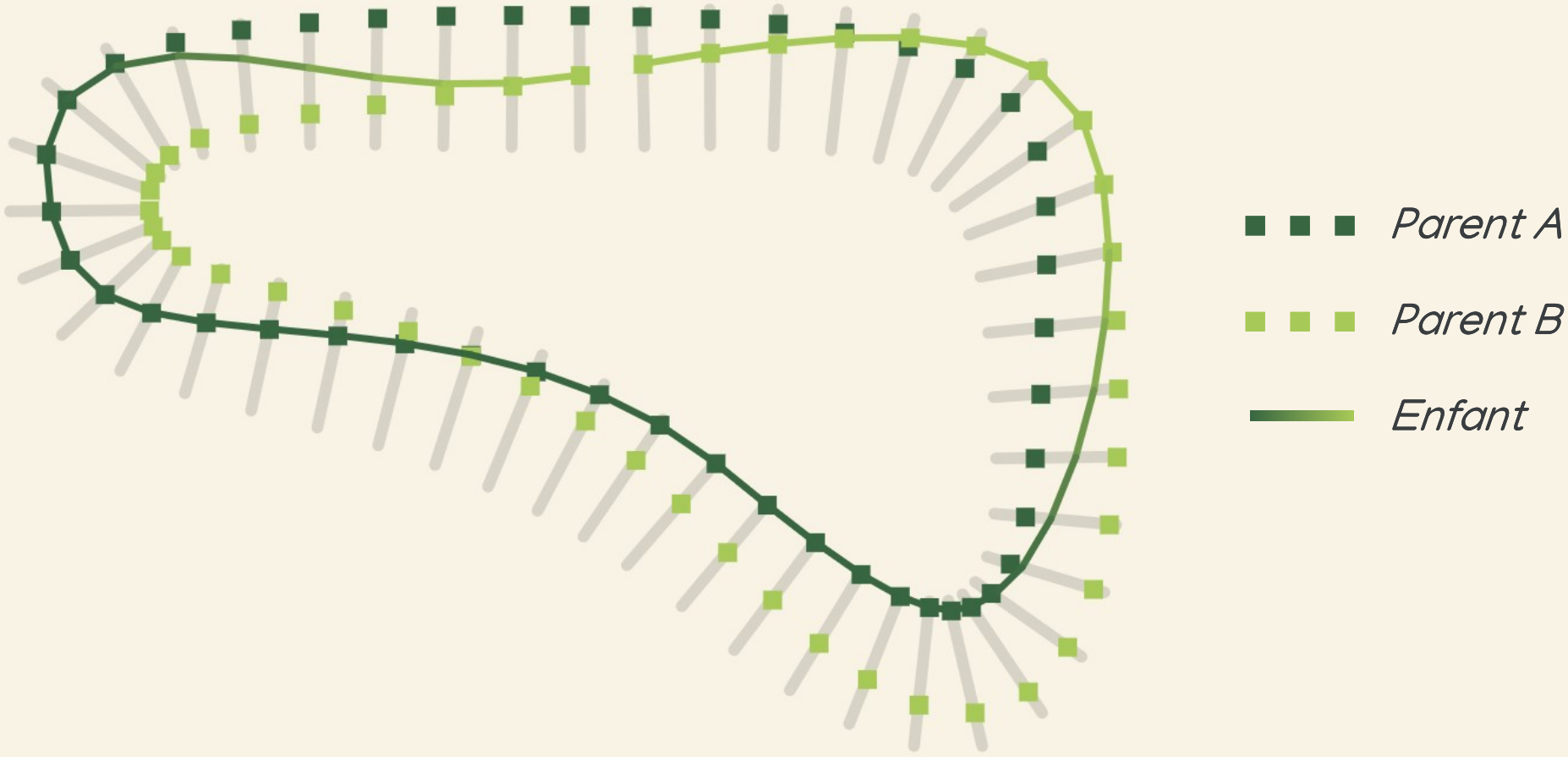
II – Algorithme génétique

c) Selection par tournois



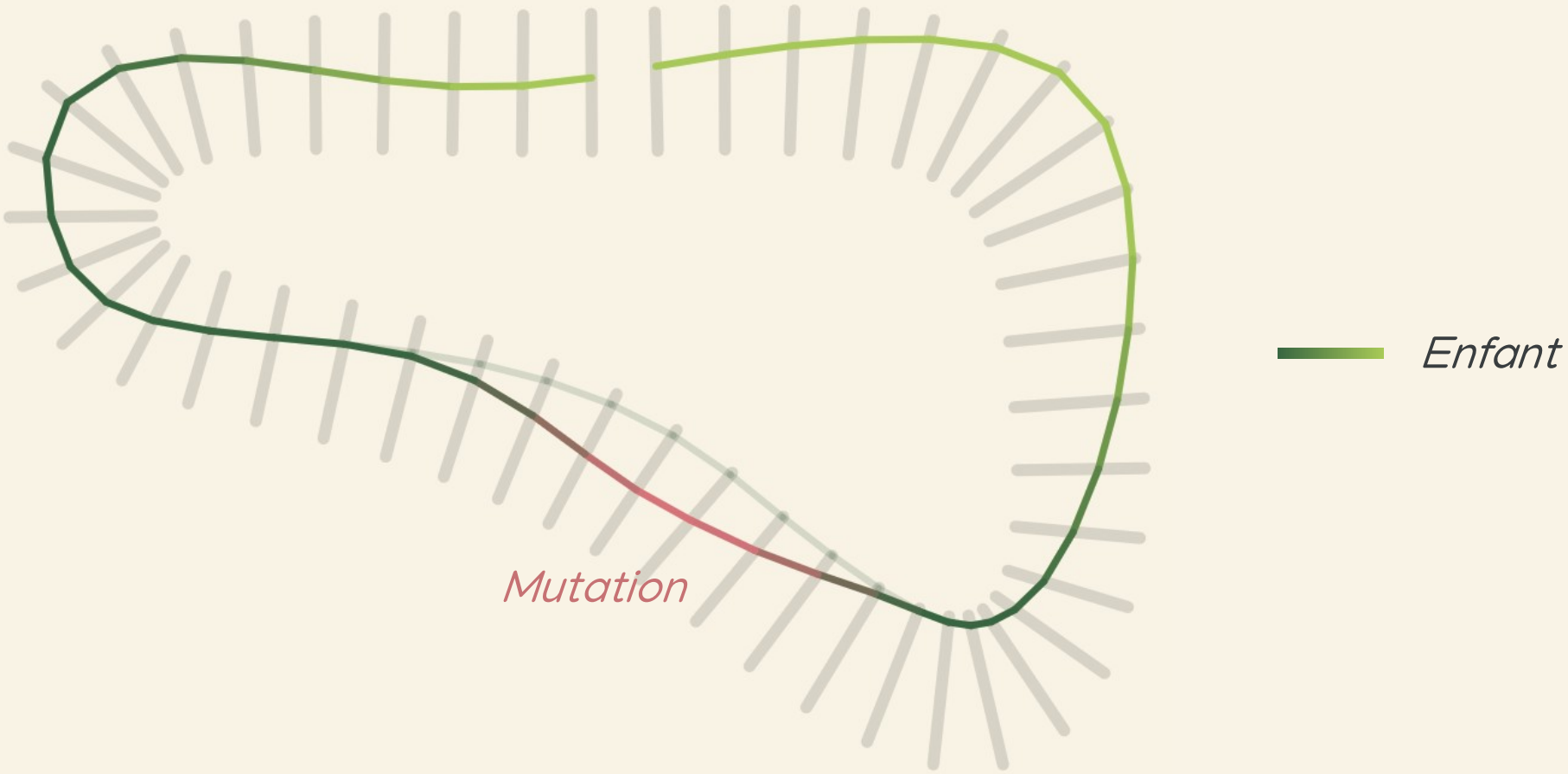
II – Algorithme génétique

d) Croisement



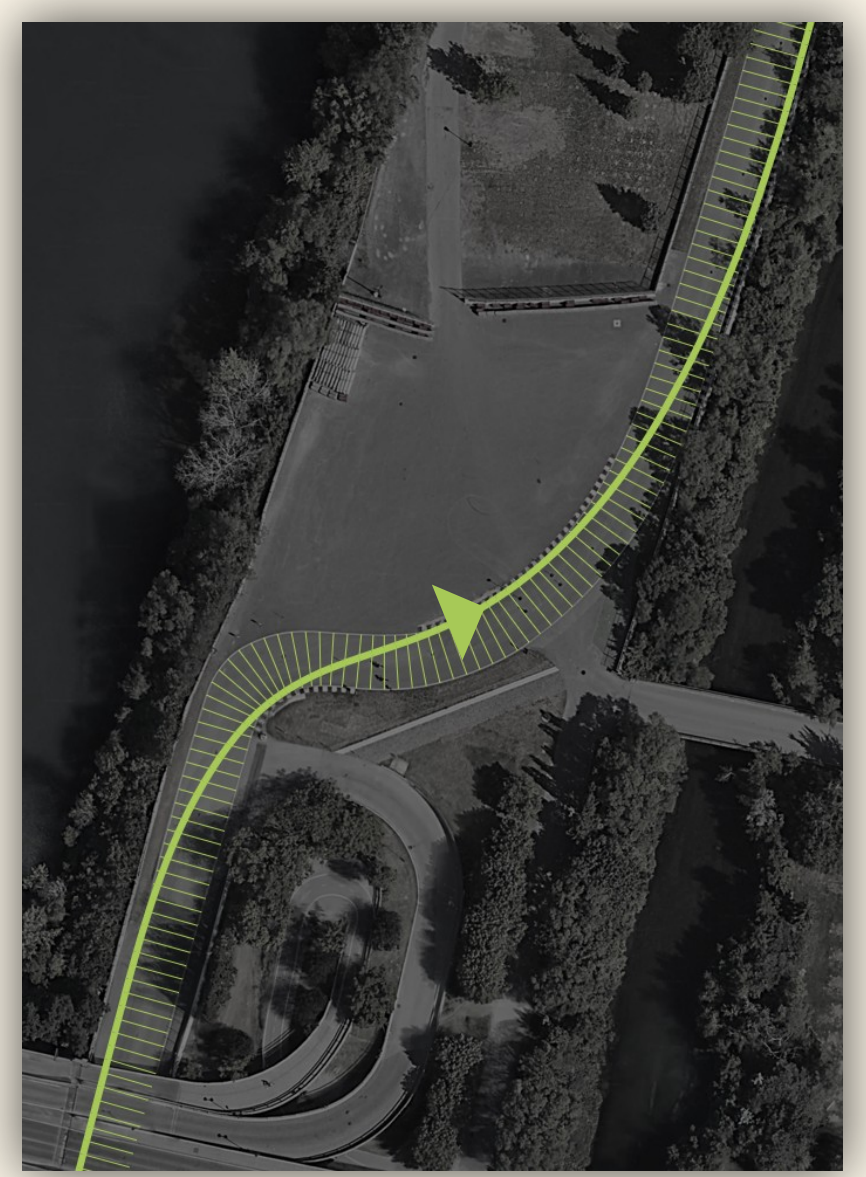
II – Algorithme génétique

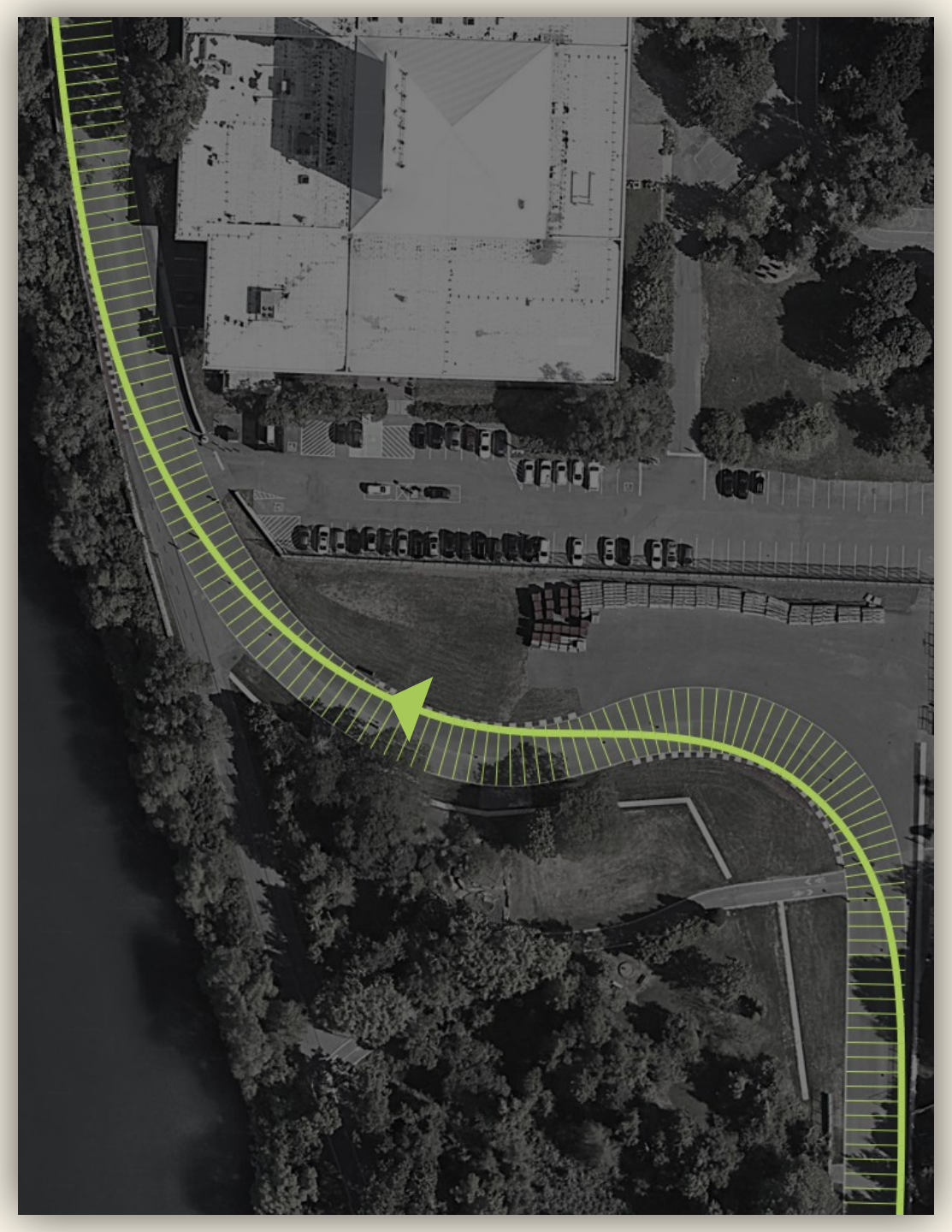
e) Mutations



III – Premiers résultats

a) Trajectoire “géométrique”



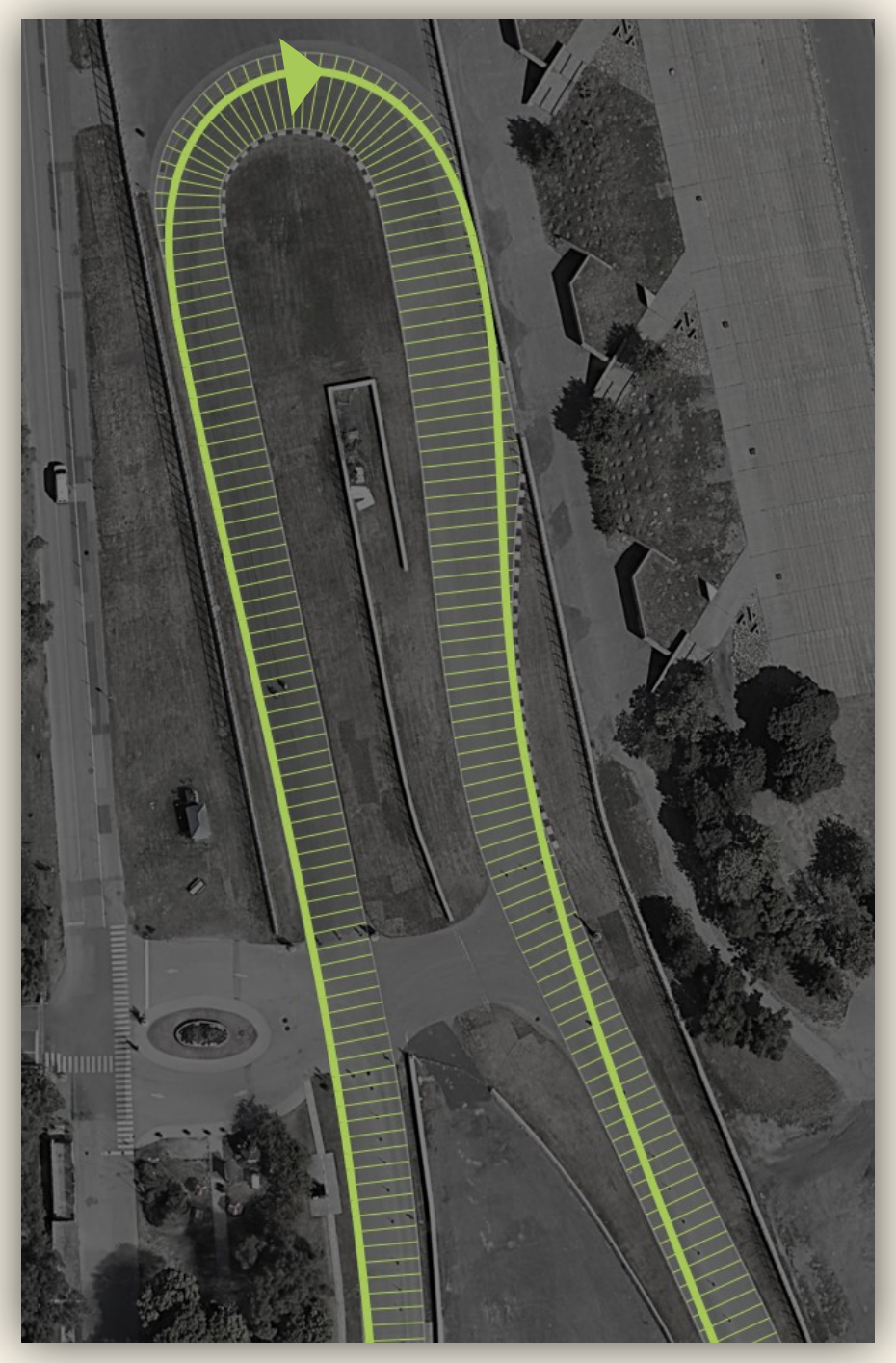


III – Premiers résultats

b) Limite de la fonction d'évaluation

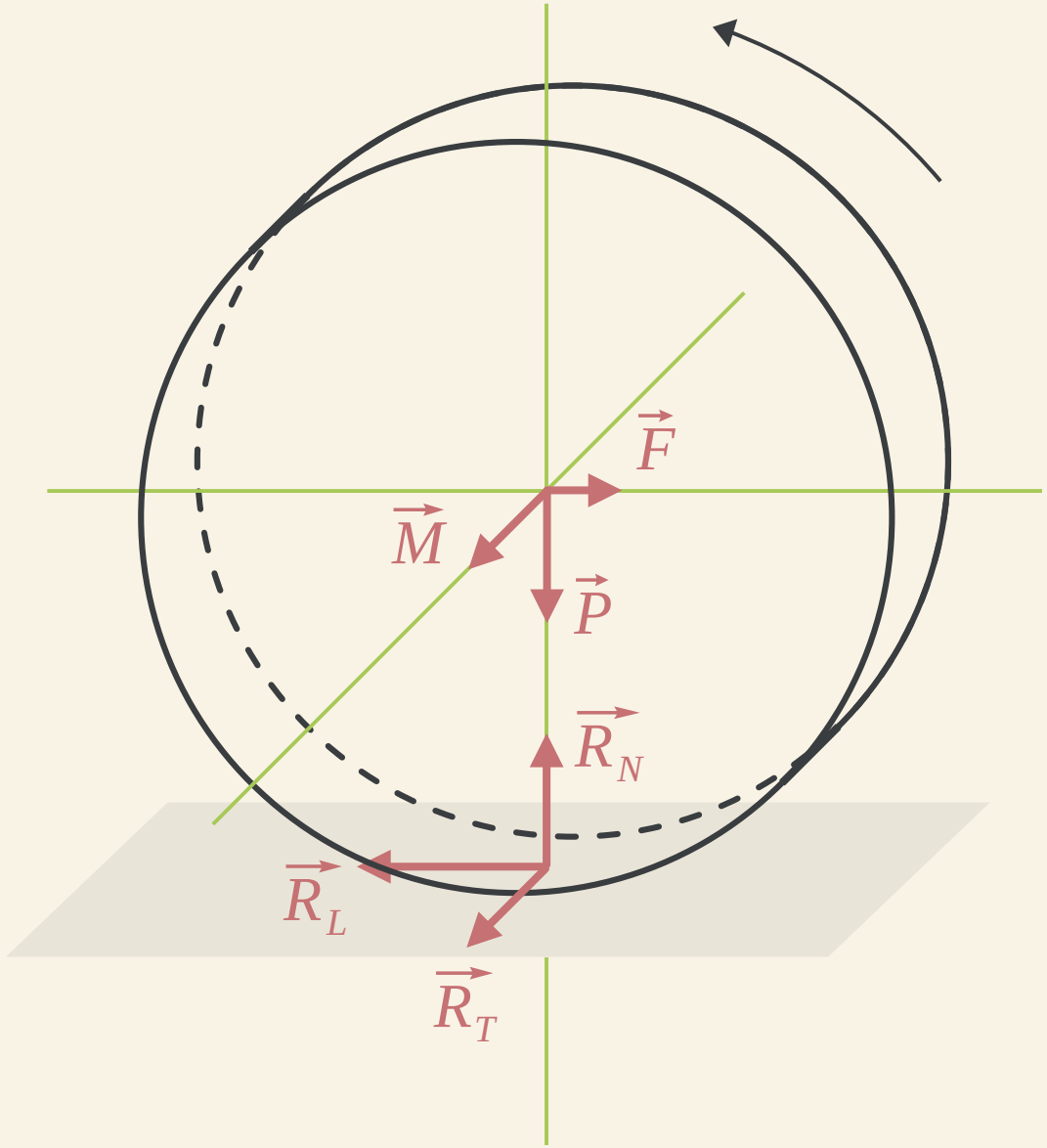
Evaluation : $\sum_{Traj} \frac{\|P_n - P_{n+1}\|}{R_n}$

Les portions en ligne droite ne "comptent pas" car la courbure y est nulle ...



IV – Profil de vitesse

a) Modélisation d'un véhicule



On modélise une voiture par une unique roue de masse m et de moment d'inertie J

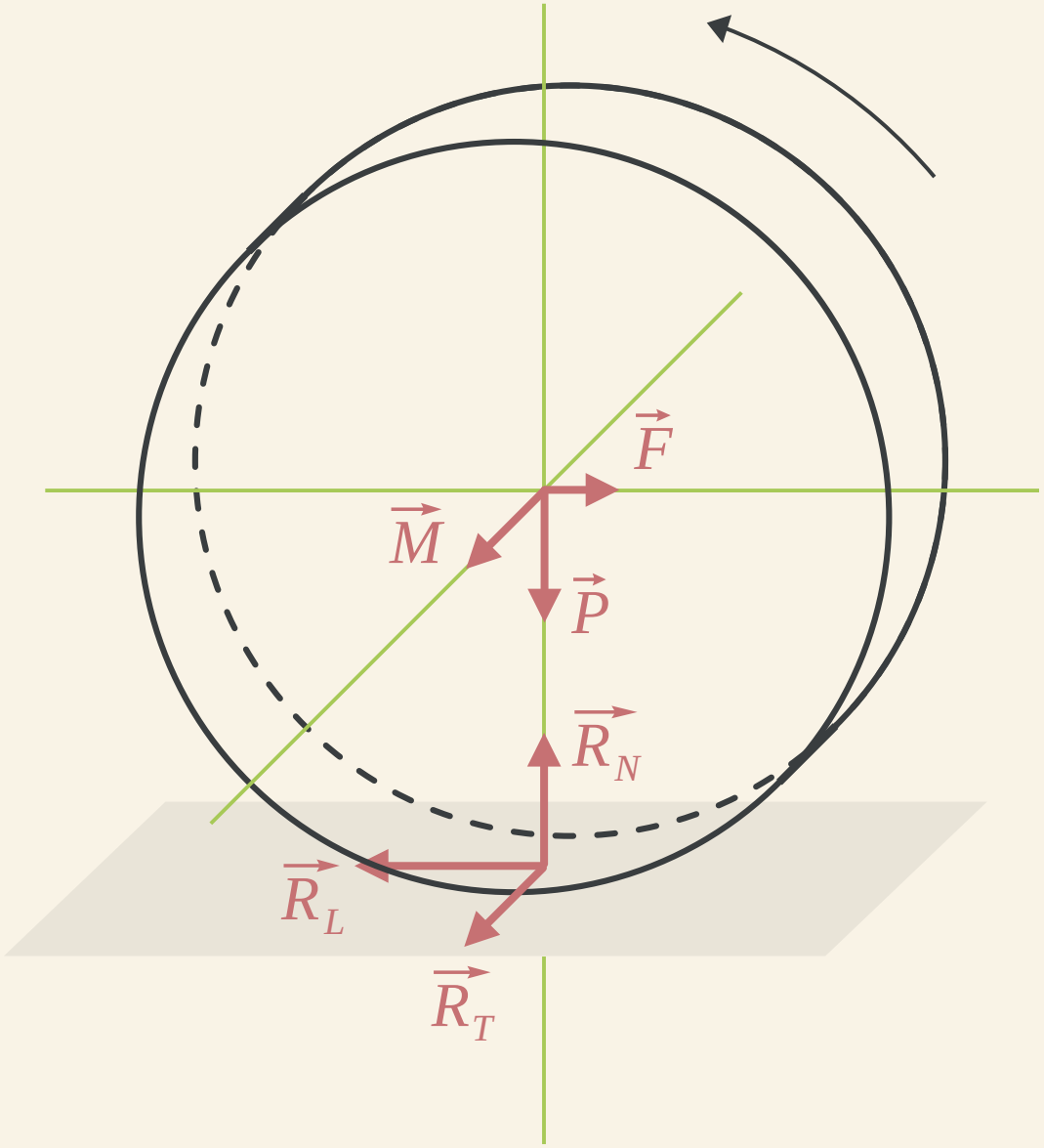
Réferentiel : la route

Bilan des actions :

- $\vec{P}$ *le poids*
- $\vec{F}$ *les frottements de l'air*
- $\vec{R}_N$ $\vec{R}_T$ $\vec{R}_L$ *les réactions normale, transversale et longitudinale de la route sur le système*
- $\vec{M}$ *le moment du couple engendré par le moteur et les freins*

IV – Profil de vitesse

b) Loi de Coulomb



Loi de Coulomb :

$$\|\vec{R_T}+\vec{R_L}\| \leqslant f_s \|\vec{R_N}\|$$

En étudiant d'abord l'inégalité plus large :

$$\|\vec{R_T}\| \leq f_s \|\vec{R_N}\|$$

On en déduit la vitesse maximale possible à chaque segment de la piste d'indice i:

$$v_i \leqslant \sqrt{\frac{f_s g}{|\kappa_i|}}$$

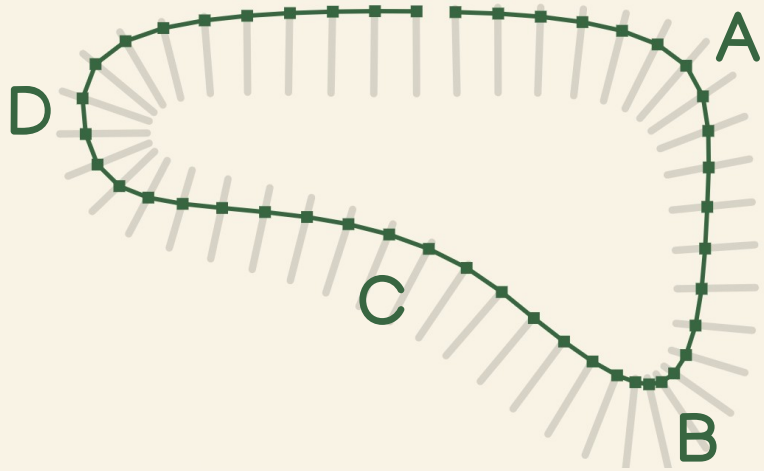
v_i *la vitesse au segment* i

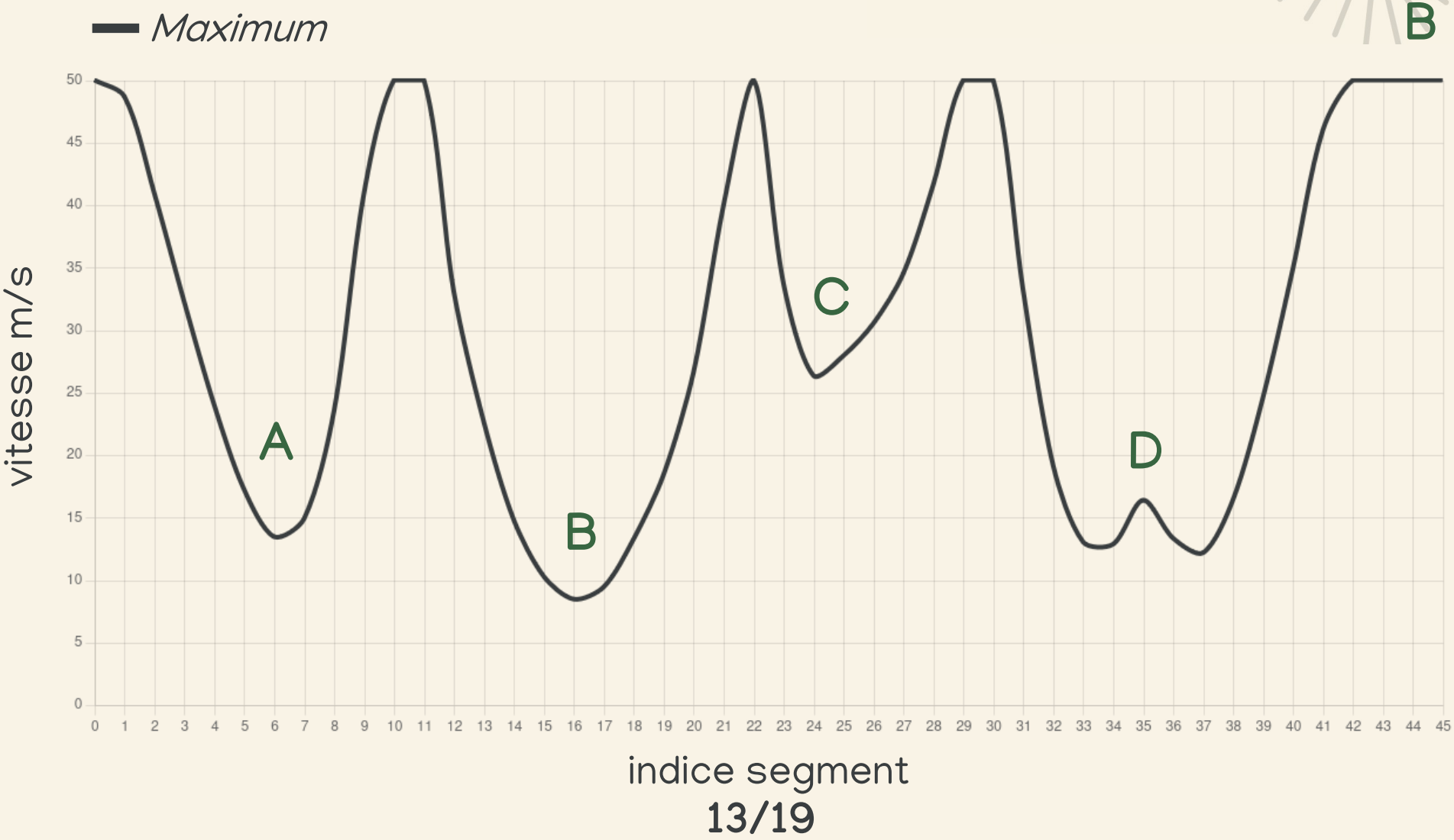
κ_i *la courbure au segement* i

g *la pesanteur*

IV – Profil de vitesse

c) Profil de vitesse





IV – Profil de vitesse

d) Freinage et accélération moteur

Inéquation au freinage :

$$v_{i-1} \leqslant v_i + \frac{\|P_{i-1} - P_i\|}{m v_i} \left(m \sqrt{f_s^2 g^2 - \kappa_i^2 v_i^4} + \alpha v_i^2 \right)$$

Inéquations à l'accélération :

$$v_{i+1} \leqslant v_i + \frac{\|P_{i+1} - P_i\|}{m v_i} \left(m \sqrt{f_s^2 g^2 - \kappa_i^2 v_i^4} - \alpha v_i^2 \right)$$

$$v_{i+1} \leqslant v_i + \frac{\|P_{i+1} - P_i\|}{m v_i} \left(\frac{mRC + \alpha J v_i^2}{J + mR^2} - \alpha v_i^2 \right)$$

P_i *position au segment* i

v_i *vitesse au segment* i

κ_i *courbure au segement* i

m *masse de la voiture*

f_s *coef. de frottement statique*

α *coef. de frottement de l'air*

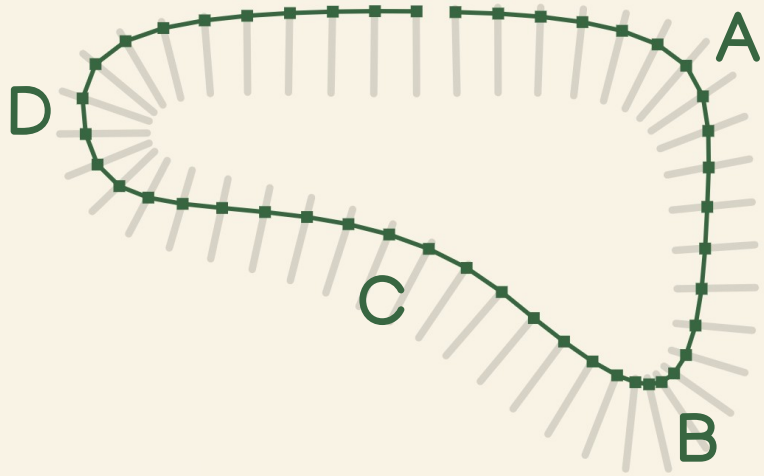
J *moment d'inertie de la roue*

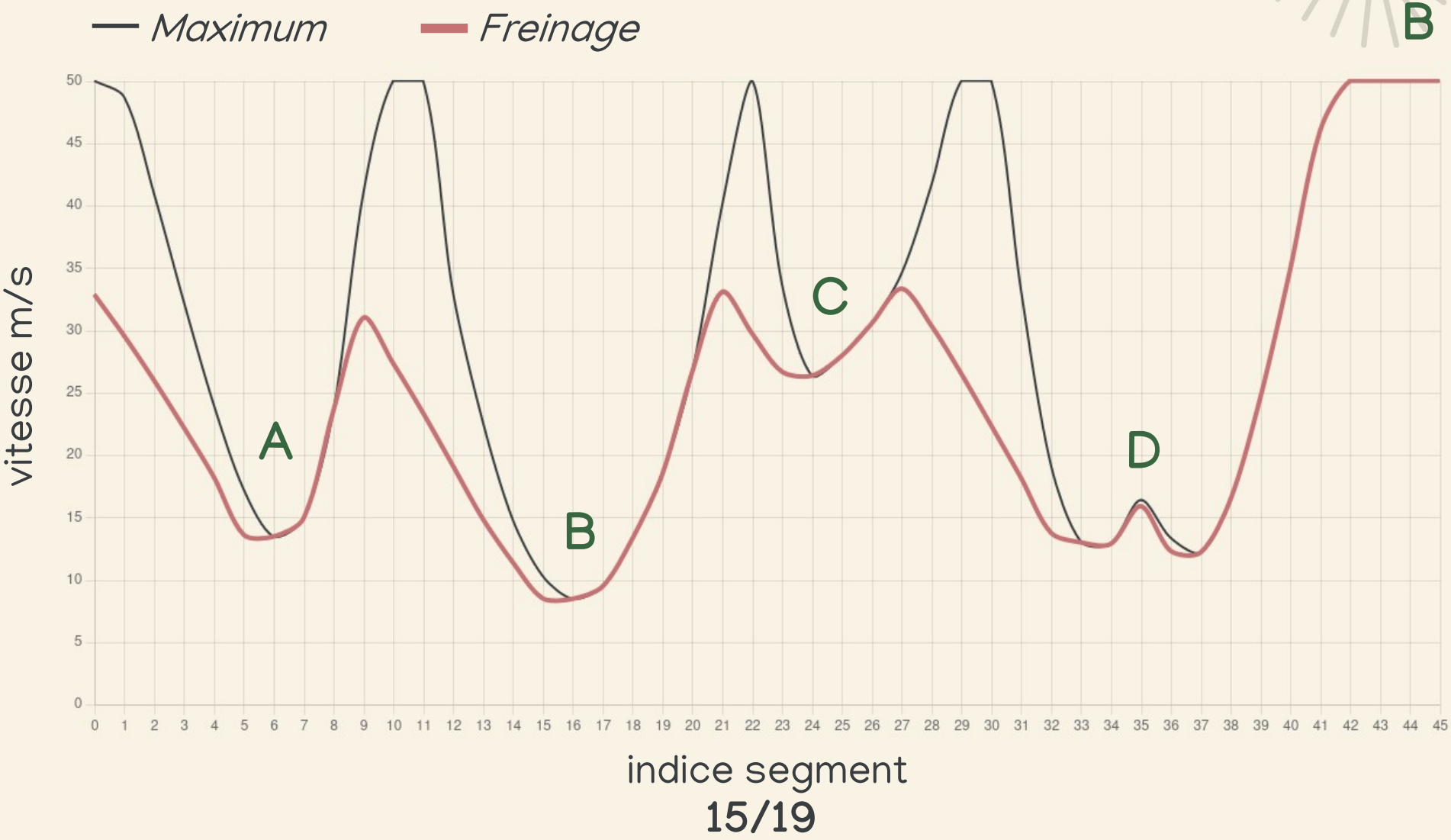
R *rayon de la roue*

C *couple développable par le moteur*

IV – Profil de vitesse

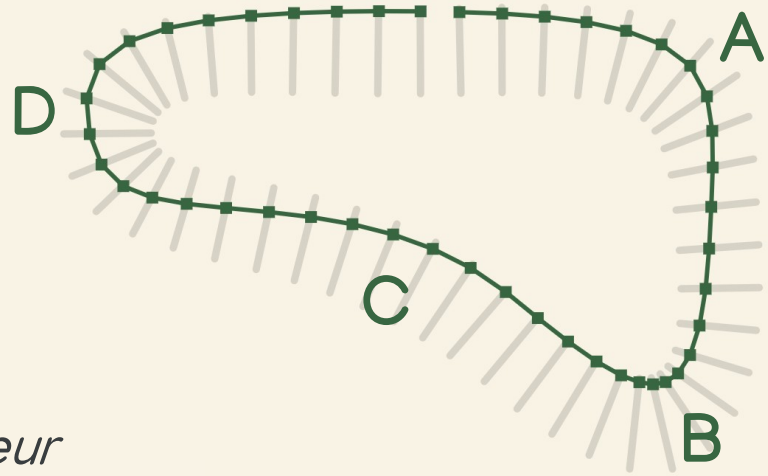
d) Freinage et accélération moteur

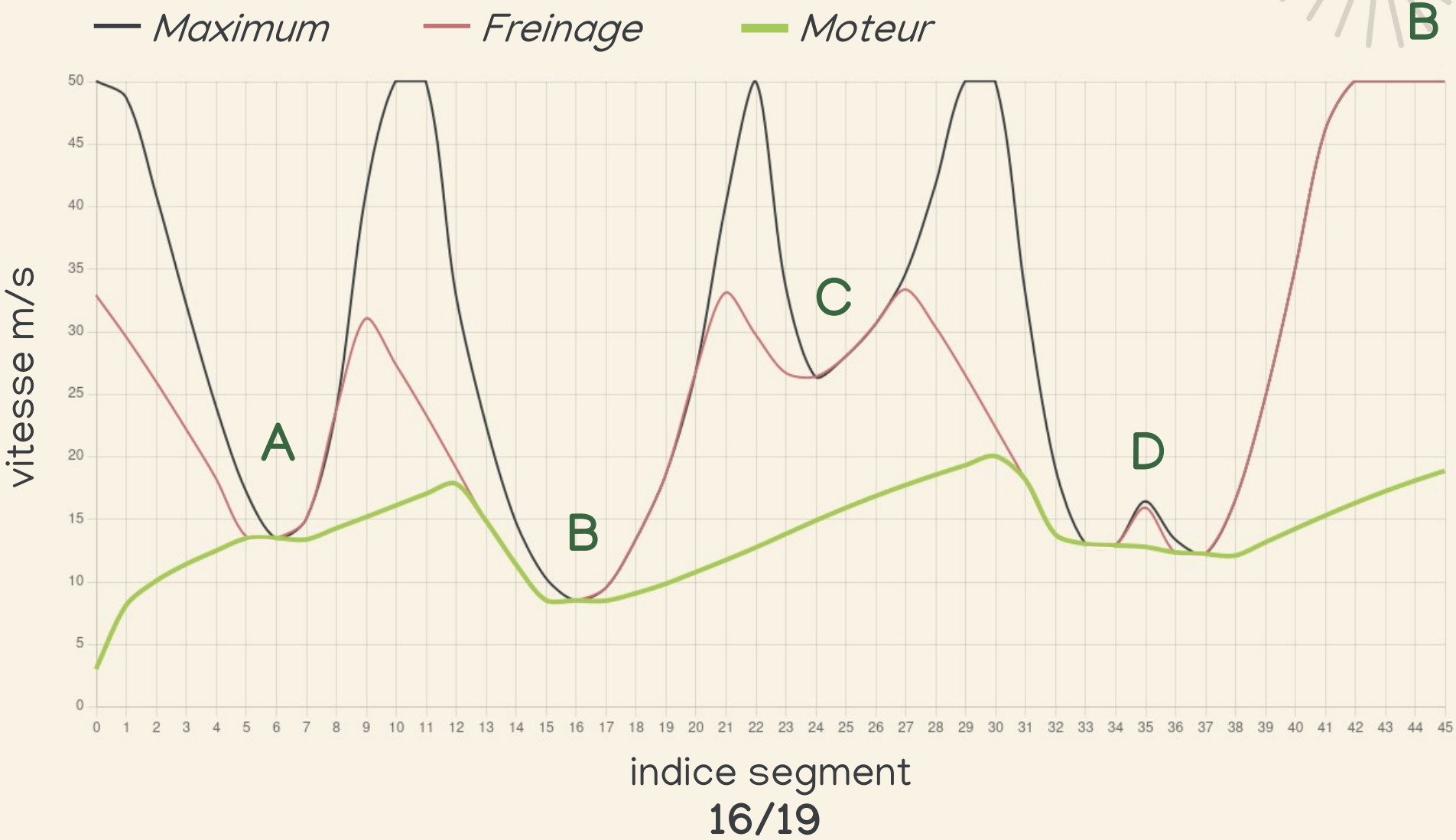




IV – Profil de vitesse

d) Freinage et accélération moteur

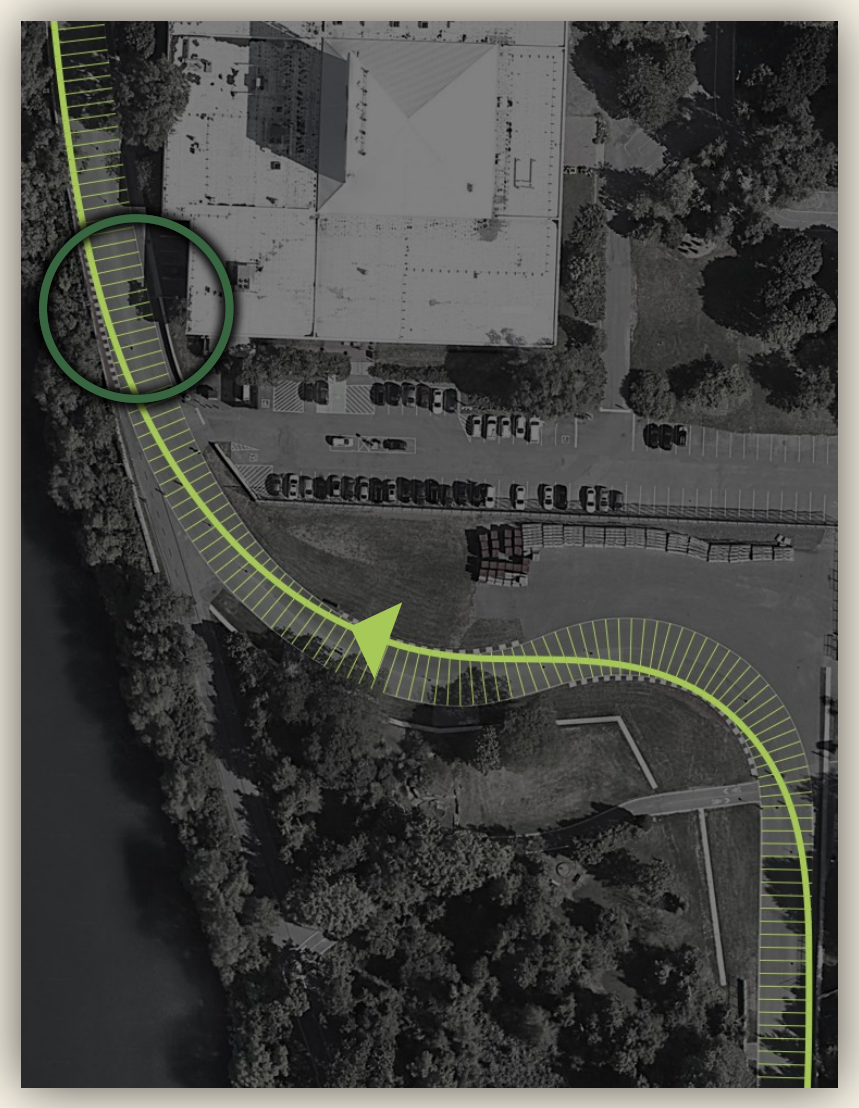




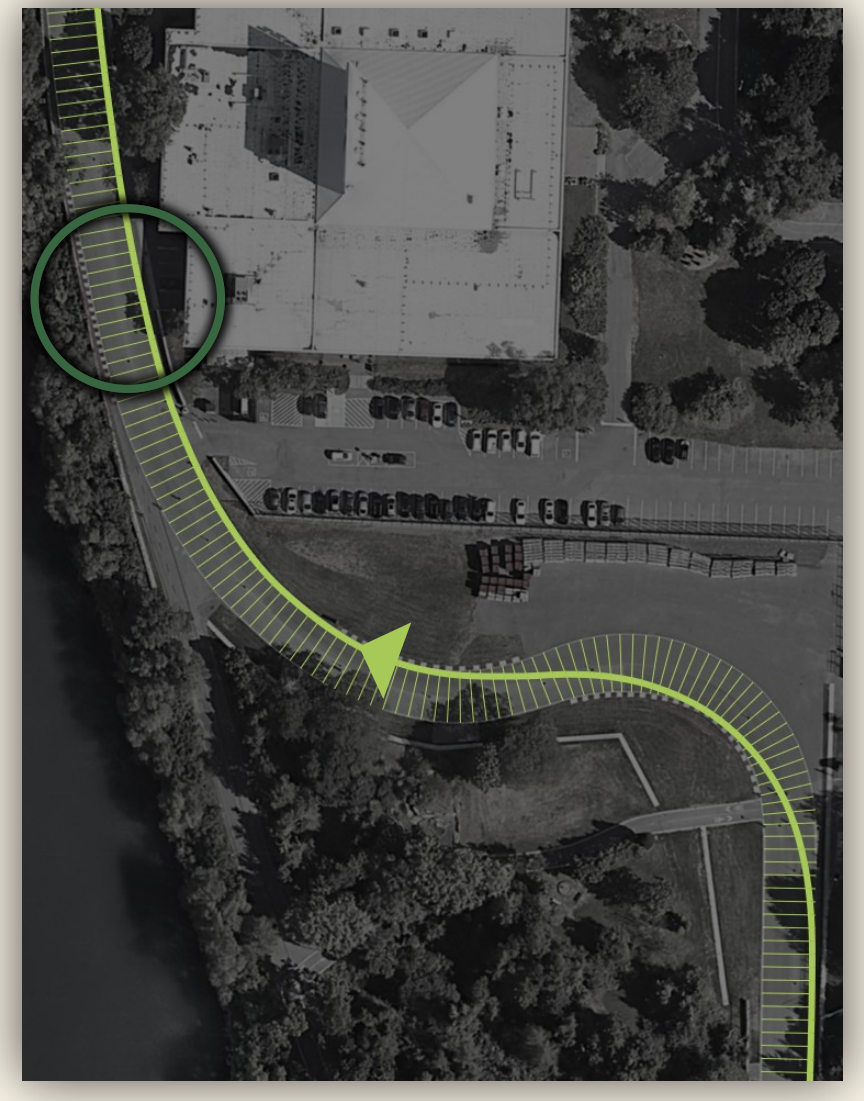
V – Conclusion

a) Comparaison en sortie de chicane

Minimisation de la courbure

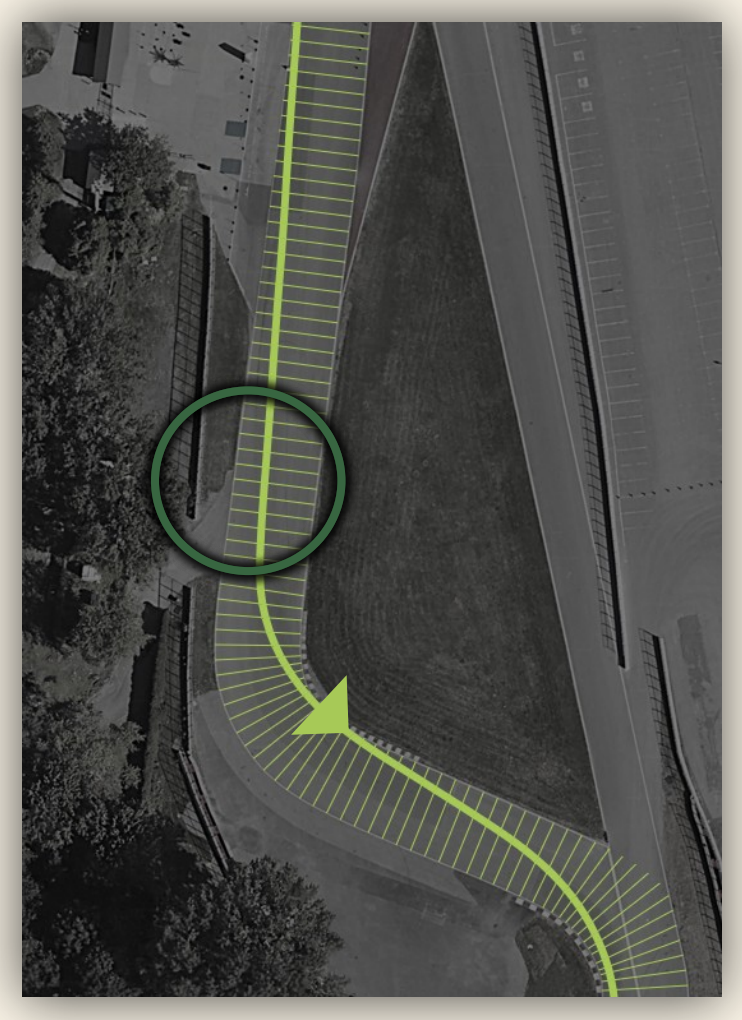


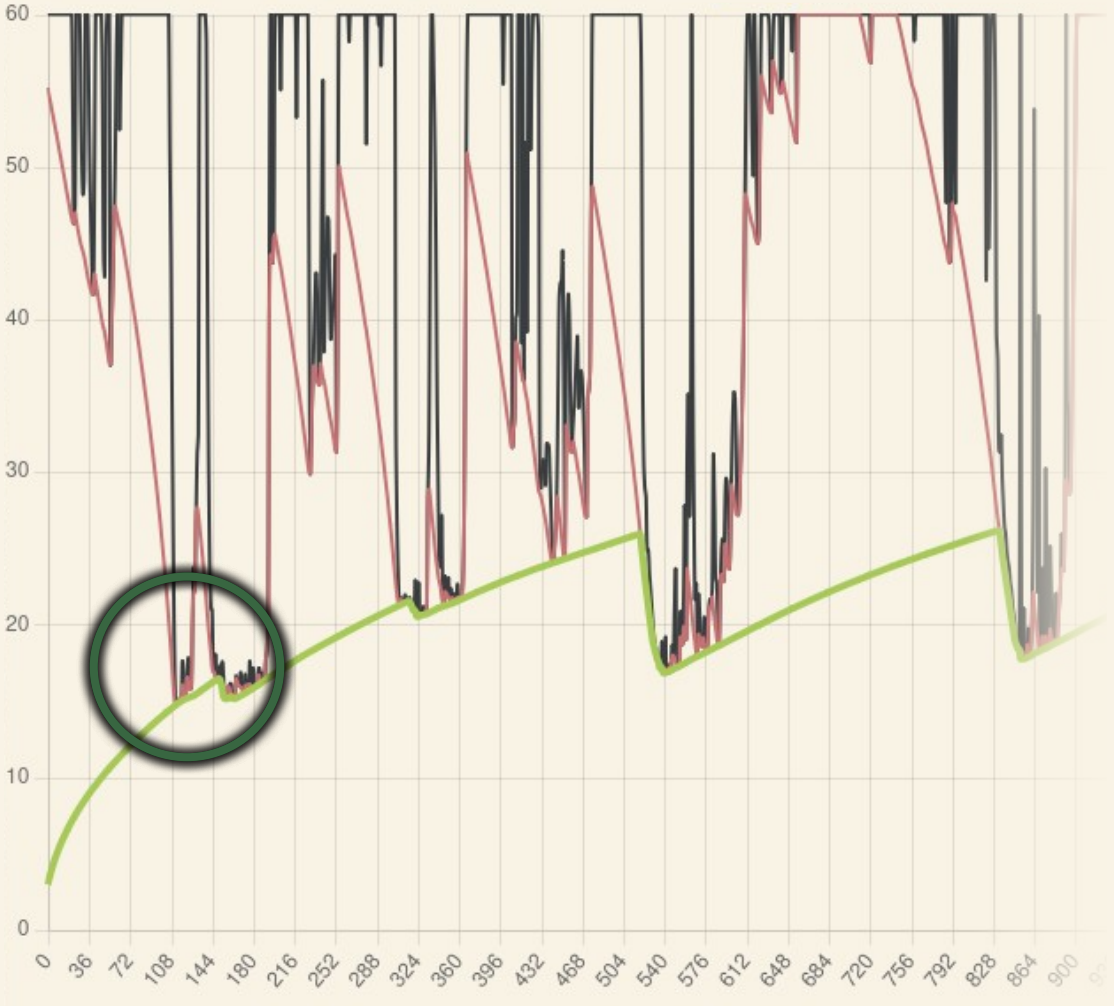
Minimisation du temps



V – Conclusion

b) Premier virage





V – Conclusion

c) Perspectives

– Générer le profil de vitesse sur 2 tours

– Faire un modèle de véhicule à 4 roues

– Modéliser différentes voitures

Annexes

GenerateBorderPoints

```
GenerateBorderPoints(){
    let quadDistConstraint = this.distBetweenLaterals*this.distBetweenLaterals;
    let precision = 1;
    //set first lateral at intBorder t=0;
    let tIntList = [0];
    this.intPoints = [this.intBorder.GetPointAtParam(0)];
    let tExtList = [this.intBorder.GetOtherBorderT(this.extBorder, 0)];
    this.extPoints = [this.extBorder.GetPointAtParam(tExtList[0])];

    while(true){//loop over number of points
        let lastIndex = tIntList.length - 1;
        let intTMin = tIntList[lastIndex];
        let intTMax = this.intBorder.GetParamAtDist(this.distBetweenLaterals, tIntList[lastIndex]);
        let intT; let extT; let intPoint; let extPoint;

        while(true){//dichotmy to create point with valid constraints
            intT = 0.5*(intTMin + intTMax);
            intPoint = this.intBorder.GetPointAtParam(intT);
            extT = this.intBorder.GetOtherBorderT(this.extBorder, intT);
            extPoint = this.extBorder.GetPointAtParam(extT);
            let quadDistInt = intPoint.QuadDistTo(this.intPoints[lastIndex]);
            let quadDistExt = extPoint.QuadDistTo(this.extPoints[lastIndex]);
            if(quadDistExt < quadDistConstraint && Math.abs(quadDistInt - quadDistConstraint) < precision){
                break;//constrained reached on intBorder with outBorder not too big
            }else if(Math.abs(quadDistExt - quadDistConstraint) < precision){
                break;//constrained reached on extBorder so valid on intBorder too
            }else if(quadDistExt > quadDistConstraint + precision){
                intTMax = intT;
            }else if(quadDistExt < quadDistConstraint - precision){
                intTMin = intT;
            }else{throw new Error("Fatal Error When Building Track Border Points");}
        }

        if(intT > this.intBorder.cubics.length){
            break;
        }
        tIntList.push(intT);
        tExtList.push(extT);
        this.intPoints.push(intPoint);
        this.extPoints.push(extPoint);
    }
}
```

Annexes

GeneticAlgorithm

```
//REPRODUCTION
while(children.length < genSize){
    //TOURNEMENT SELECTION
    let parents = [];
    let potentialParentIndexes = [];
    let nbPotentialParents = Math.floor(this.parentCount * this.selectionPressure);
    for(let i = 0; i < nbPotentialParents; i++){
        potentialParentIndexes.push(Math.floor(genSize * Math.random()))
    }
    while(parents.length < this.parentCount){
        let winnerIndex = potentialParentIndexes[0];
        let indexToRemove = 0;
        for(let i = 1; i < potentialParentIndexes.length; i++){
            if(this.trajs[winnerIndex].evaluation > this.trajs[potentialParentIndexes[i]].evaluation){
                winnerIndex = potentialParentIndexes[i];
                indexToRemove = i;
            }
        }
        if(potentialParentIndexes.length > 1){
            potentialParentIndexes.splice(indexToRemove, 1);
        }
        parents.push(this.trajs[winnerIndex]);
    }

    //CROSSOVER
    let newChild = Traj.Crossover(parents, this.crossoverSmoothZone);
    newChild.Evaluate(this.evaluationMode, this.track, this.car);
    children.push(newChild);
}

//MUTATION BUMP
for(let i = nbOfEliteChildren; i < genSize; i++){
    let cumul = this.mutationBumpProbability;
    while(cumul > Math.random()){
        let mutationPoint = Math.floor(Math.random() * this.track.n);
        let mutationZoneSemiLength = getRandomMutationZoneSemiLength(this.mutationMinSemiLength, this.mutationMedSemiLength, this.mutationMaxSemiLength);
        children[i].MutateBump(mutationPoint, mutationZoneSemiLength, this.mutationBumpForce);
        cumul -= 1;
    }
}
```

Annexes

Curvature

```
/**
 * @param {Point} a
 * @param {Point} b
 * @param {Point} c
 * @param {Number} pxToMetersRatio
 */
export function signedCurvatureBetween(a, b, c, pxToMetersRatio){
    //https://en.wikipedia.org/wiki/Menger_curvature#Definition
    //https://math.stackexchange.com/questions/2511452/how-do-i-calculate-the-signed-area-of-a-triangle-in-3d-space
    let ab = a.DistTo(b)*pxToMetersRatio;
    let bc = b.DistTo(c)*pxToMetersRatio;
    let ca = c.DistTo(a)*pxToMetersRatio;

    let signedTiangleArea = 0.5*((b.x*c.y-c.x*b.y) - (a.x*c.y-c.x*a.y) + (a.x*b.y-b.x*a.y))*pxToMetersRatio*pxToMetersRatio;
    return 4*signedTiangleArea/(ab*bc*ca);
}
```

```
/**
 * @param {Track} track
 */
CalcCurvature(track){
    this.absCurv = [Math.abs(signedCurvatureBetween(this.points[this.n-1], this.points[0], this.points[1], track.pxToMetersRatio))];
    for(let i = 1; i < this.n-1; i++){
        this.absCurv.push(Math.abs(signedCurvatureBetween(this.points[i-1], this.points[i], this.points[i+1], track.pxToMetersRatio)))
    }
    this.absCurv.push(Math.abs(signedCurvatureBetween(this.points[this.n-2], this.points[this.n-1], this.points[0], track.pxToMetersRatio)))
}
```

Annexes

3PassSpeed

```
/**
 * @param {Car} car
 */
CalcSpeeds3Pass(car){
    this.speed1 = [];
    for (let i = 0; i < this.n; i++) {
        if(this.absCurv[i] == 0){
            this.speed1.push(car.theoricalMaxSpeed);
            continue;
        }
        let maxSpeed = Math.sqrt(car.roadFrictionCoef * car.g / this.absCurv[i]);
        if(maxSpeed > car.theoricalMaxSpeed){
            this.speed1.push(car.theoricalMaxSpeed);
        }else{
            this.speed1.push(maxSpeed);
        }
    }

    this.speed2 = this.speed1.map(x => x);
    for (let i = this.n-1; i > 0; i--){
        if(this.speed2[i] <= 0){
            this.speed2[i - 1] = 0;
            continue;
        }

        var Rt;
        let maxDecel = Math.pow(car.roadFrictionCoef * car.mass * car.g, 2) - Math.pow(car.mass * this.absCurv[i] * this.speed2[i] * this.speed2[i], 2);
        if(maxDecel < 0){
            Rt = 0; // we can't decelerate or we will drift
        }else{
            Rt = -Math.sqrt(maxDecel);
        }
        this.speed2[i - 1] = -this.dists[i]*(Rt - (car.airDragCoef * this.speed2[i] * this.speed2[i])) / (this.speed2[i] * car.mass) + this.speed2[i]
        if(this.speed1[i - 1] < this.speed2[i - 1]){
            this.speed2[i - 1] = this.speed1[i - 1]
        }
    }

    this.speed3 = this.speed2.map(x => x);
    this.speed3[0] = 3;
```

```
        }
        this.speed2[i - 1] = -this.dists[i]*(Rt - (car.airDragCoef * this.speed2[i] * this.speed2[i])) / (this.speed2[i] * car.mass) + this.speed2[i]
        if(this.speed1[i - 1] < this.speed2[i - 1]){
            this.speed2[i - 1] = this.speed1[i - 1]
        }
    }

    this.speed3 = this.speed2.map(x => x);
    this.speed3[0] = 3;
    for (let i = 0; i < this.n - 1; i++){
        var Rt = 0;
        for(let gearId = 0; gearId < car.numOfGears; gearId++){
            let engineRotSpeed = car.GetEngineRotSpeed(this.speed3[i], gearId);
            let torque = car.GetTorque(engineRotSpeed);
            let RtAtGear = (torque * car.wheelRadius * car.mass) + (car.airDragCoef * this.speed3[i] * this.speed3[i] * car.angularMassJ);
            RtAtGear /= car.angularMassJ + (car.wheelRadius * car.wheelRadius * car.mass);
            if(RtAtGear > Rt){
                Rt = RtAtGear;
                this.gear[i] = gearId;
                this.rpm[i] = engineRotSpeed * 9.549296585513721 // convert rad/s to rpm
            }
        }
        let a = Math.pow(car.roadFrictionCoef * car.mass * car.g, 2) - Math.pow(car.mass * this.absCurv[i] * this.speed3[i] * this.speed3[i], 2);
        if(a < 0){
            Rt = 0; // we can't accelerate or we will drift
        }else{
            Rt = Math.min(Rt, Math.sqrt(a));
        }
        this.speed3[i + 1] = this.dists[i]*(Rt - (car.airDragCoef * this.speed3[i] * this.speed3[i])) / (this.speed3[i] * car.mass) + this.speed3[i]
        if(this.speed3[i + 1] > this.speed2[i + 1]){
            this.speed3[i + 1] = this.speed2[i + 1]
        }
    }
}
```

Annexes

Crossover

```
/**
 * @param {Traj[]} parentTrajs number of transitionPoints, must be even and shuffled
 * @param {Number} crossoverSmoothZone number of laterals to interpolate between parents at transitionPoints
 * @returns {Traj} the child crossover
 */
static Crossover(parentTrajs, crossoverSmoothZone){
    let trackN = parentTrajs[0].n;

    //GENERATE TRANSITION POINTS
    let transitionPoints = [];
    GenerateTransitionPoints :
    while(true){
        transitionPoints = [];
        for(let i = 0; i < parentTrajs.length; i++){
            let newTransitionPoint = Math.floor(Math.random() * trackN);
            for(let j = 0; j < i; j++){
                if(Math.abs(transitionPoints[j] - newTransitionPoint) <= 2*crossoverSmoothZone){
                    continue GenerateTransitionPoints;
                }
                if(Math.abs(transitionPoints[j] - newTransitionPoint + trackN) <= 2*crossoverSmoothZone){
                    continue GenerateTransitionPoints;
                }
                if(Math.abs(transitionPoints[j] - newTransitionPoint - trackN) <= 2*crossoverSmoothZone){
                    continue GenerateTransitionPoints;
                }
            }
            transitionPoints.push(newTransitionPoint);
        }
        transitionPoints.sort((a, b) => a - b);
        break;
    }

    //BUILD CHILD
    let newTraj = new Traj(trackN, false);
```

Annexes

Crossover

```
        transitionPoints.sort((a, b) => a - b);
        break;
    }

    //BUILD CHILD
    let newTraj = new Traj(trackN, false);
    let currentParent = 0;
    for(let i = transitionPoints[0]; i < transitionPoints[transitionPoints.length - 1]; i++){//between first and last transition points
        let distanceToNextTransitionPoint = transitionPoints[currentParent + 1] - i
        let alpha;
        if(distanceToNextTransitionPoint == 0){
            alpha = 0;
            currentParent++;
        }else if(distanceToNextTransitionPoint > crossoverSmoothZone){
            alpha = 0;
        }else{
            alpha = 1 - (distanceToNextTransitionPoint / crossoverSmoothZone);
            alpha = Math.sin(alpha*Math.PI/2) * Math.sin(alpha*Math.PI/2);
        }
        newTraj.laterals[i] = alpha * parentTrajs[currentParent + 1].laterals[i] + (1 - alpha) * parentTrajs[currentParent].laterals[i];
    }
    for(let i = transitionPoints[transitionPoints.length - 1]; i < transitionPoints[0] + trackN; i++){//last transition point loop back to first transition point
        let distanceToNextTransitionPoint = transitionPoints[0] + trackN - i
        let alpha;
        if(distanceToNextTransitionPoint > crossoverSmoothZone){
            alpha = 0;
        }else{
            alpha = 1 - (distanceToNextTransitionPoint / crossoverSmoothZone);
            alpha = Math.sin(alpha*Math.PI/2) * Math.sin(alpha*Math.PI/2);
        }
        newTraj.laterals[mod(i, trackN)] = alpha * parentTrajs[0].laterals[mod(i, trackN)] + (1 - alpha) * parentTrajs[transitionPoints.length - 1].laterals[mod(i, trackN)];
    }
    return newTraj;
}
```

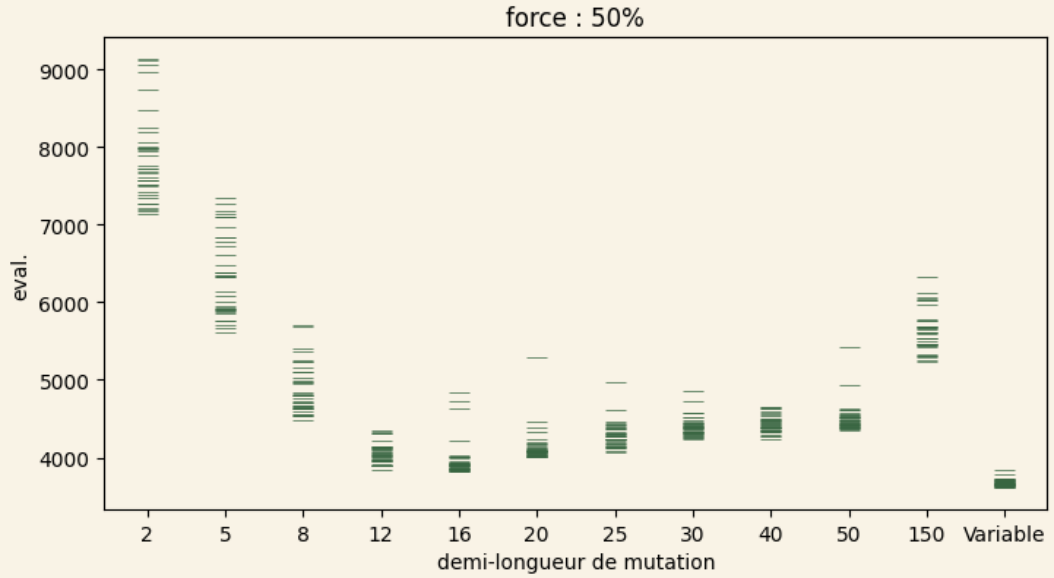
Annexes

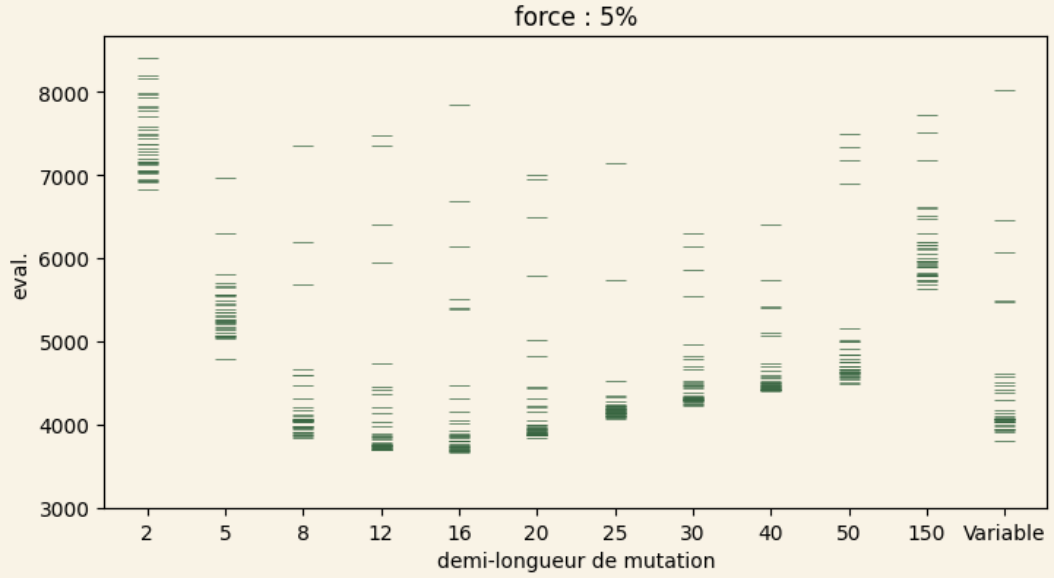
Mutate

```
/**
 * Makes bump mutation
 * @param {Number} center int of lateral where the mutation is centered
 * @param {Number} semiLength int defining half the number of laterals that will be modified
 * @param {Number} force in [0,1], 0 is no modifications, 1 is potentially harsh modifications
 */
MutateBump(center, semiLength, force) {
    this.evaluation = -1;
    let mutationValue = Math.random();
    for (let i = -semiLength + 1; i < semiLength; i++) {//skipping first and last indexes because they wouldn't actually be moved
        let blend = force * squaredSinJoin(1 - (Math.abs(i)/semiLength));
        let currentPoint = mod((center + i), this.n);
        this.laterals[currentPoint] = blend*mutationValue + (1-blend)*this.laterals[currentPoint];
        if(this.laterals[currentPoint] < 0){this.laterals[currentPoint] = 0;} // safety (float approx)
        if(this.laterals[currentPoint] > 1){this.laterals[currentPoint] = 1;}
    }
}
```

Annexes

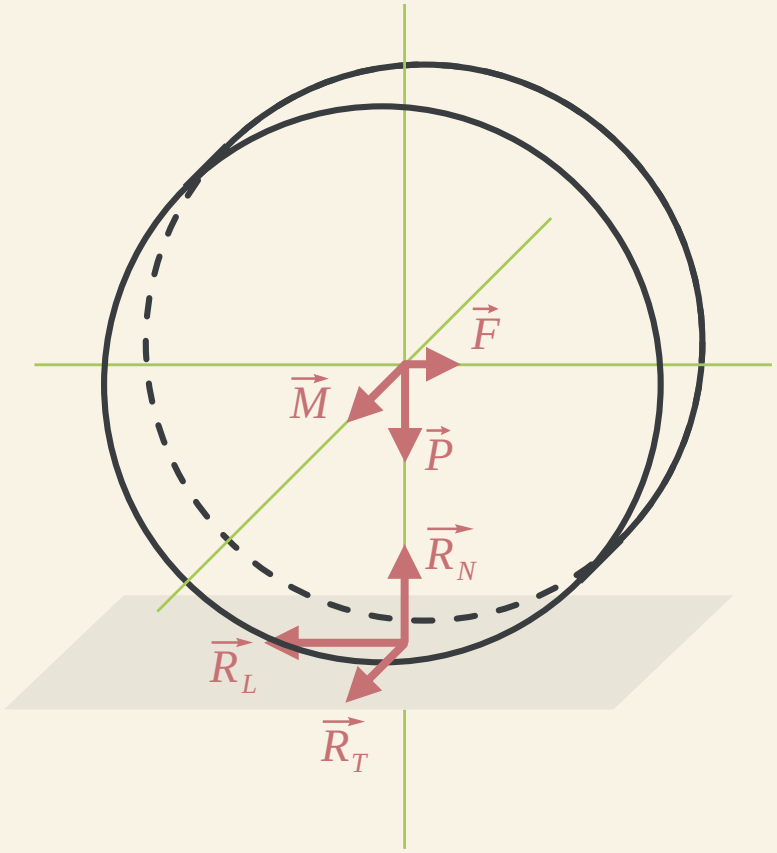
Experiences mutations





Annexes

Calculs partie physique



Dans le repère de Frenet :

$$\vec{a} = \ddot{s}\,\vec{u_L} + \dot{s}^2 \kappa(s)\,\vec{u_T}$$

D'après la 2ème loi de Newton :

$$m\vec{a} = \sum \vec{F_{ext}}$$

Ainsi en projetant sur les 3 axes :

$$\begin{cases} 0 = R_N - mg & (1) \\ m\dot{s}^2 \kappa(s) = R_T & (2) \\ m\ddot{s} = R_L - \alpha \dot{s}^2 & (3) \end{cases}$$

La loi de Coulomb avec $R_L = 0$ donne :

$$|R_T| \leq f_s R_N = f_s mg$$

Donc d'après (2)

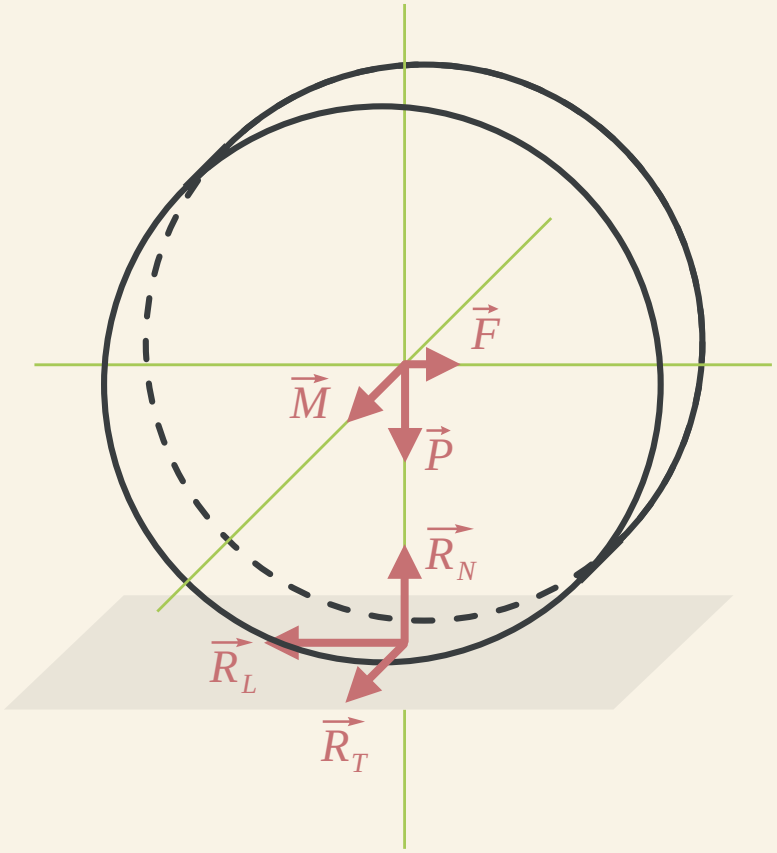
$$\dot{s}^2 \leqslant \frac{f_s mg}{m|\kappa(s)|}$$

Finalement :

$$v_i \leqslant \sqrt{\frac{f_s g}{|\kappa_i|}}$$

Annexes

Calculs partie physique



Quelques résultats utiles :

$$\begin{cases} 0 = R_N - mg & (1) \\ m\dot{s}^2\kappa(s) = R_T & (2) \\ m\ddot{s} = R_L - \alpha\dot{s}^2 & (3) \end{cases}$$

D'après (3) :

$$\frac{R_L - \alpha\dot{s}^2(t)}{m} = \ddot{s}(t) = \frac{\dot{s}(t) - \dot{s}(t-dt)}{dt} = \frac{\dot{s}(t) - \dot{s}(t-dt)}{ds} \times \dot{s}(t)$$

Ainsi on obtient l'équation :

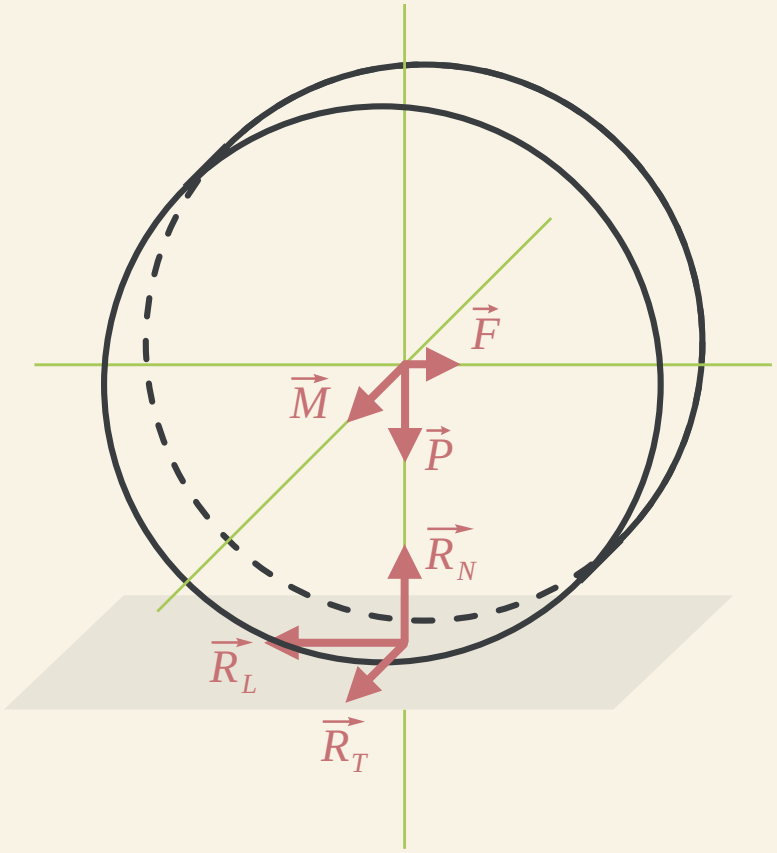
$$\dot{s}(t-dt) = \dot{s}(t) - \frac{ds \times (R_L - \alpha\dot{s}^2(t))}{m\dot{s}(t)} \quad (4)$$

De manière similaire on a aussi :

$$\dot{s}(t+dt) = \dot{s}(t) + \frac{ds \times (R_L - \alpha\dot{s}^2(t))}{m\dot{s}(t)} \quad (5)$$

Annexes

Calculs partie physique



Quelques résultats utiles :

$$\begin{cases} 0 = R_N - mg & (1) \\ m\dot{s}^2\kappa(s) = R_T & (2) \\ m\ddot{s} = R_L - \alpha\dot{s}^2 & (3) \end{cases}$$

D'après la loi de Coulomb :

$$R_L{}^2 \leq f_s{}^2 R_N{}^2 - R_T{}^2$$

Donc d'après (1) et (2) :

$$|R_L| \leq m\sqrt{(f_s g)^2 - (\kappa(s)\dot{s}^2)^2}$$

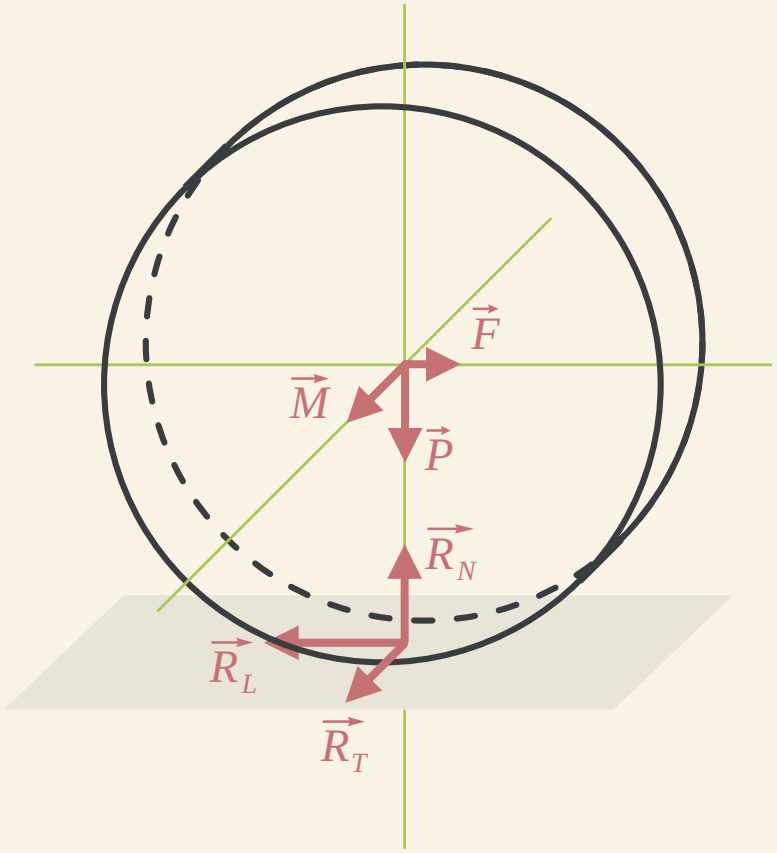
En réinjectant dans les équations (4) et (5) et en discrétisant on obtient :

$$v_{i-1} \leqslant v_i + \frac{\|P_{i-1} - P_i\|}{mv_i}\left(m\sqrt{f_s^2 g^2 - \kappa_i^2 v_i^4} + \alpha v_i^2\right)$$

$$v_{i+1} \leqslant v_i + \frac{\|P_{i+1} - P_i\|}{mv_i}\left(m\sqrt{f_s^2 g^2 - \kappa_i^2 v_i^4} - \alpha v_i^2\right)$$

Annexes

Calculs partie physique



Quelques résultats utiles :

$$\begin{cases} 0 = R_N - mg & (1) \\ m\dot{s}^2\kappa(s) = R_T & (2) \\ m\ddot{s} = R_L - \alpha\dot{s}^2 & (3) \end{cases}$$

D'après le Théorème du moment cinétique :

$$J\ddot{\theta} = C_{utilisé} + M_\Delta(\vec{R_L}) \leq C + M_\Delta(\vec{R_L})$$

D'où :

$$J\frac{\ddot{s}}{R} \leq C - R \times R_L$$

Donc d'après (3) :

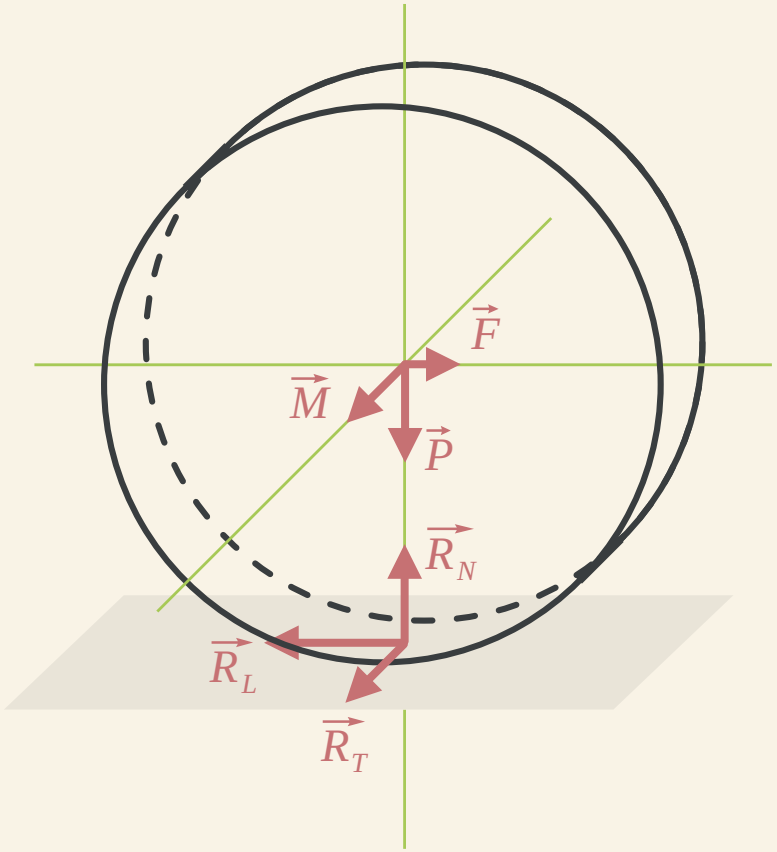
$$J\frac{R_L - \alpha\dot{s}^2}{mR} \leq C - R \times R_L$$

Et ainsi :

$$R_L \leq \frac{C + \frac{\alpha J \dot{s}^2}{mR}}{R + \frac{J}{mR}} = \frac{mRC + \alpha J \dot{s}^2}{mR^2 + J}$$

Annexes

Calculs partie physique



Quelques résultats utiles :

$$\begin{cases} 0 = R_N - mg & (1) \\ m\dot{s}^2\kappa(s) = R_T & (2) \\ m\ddot{s} = R_L - \alpha\dot{s}^2 & (3) \end{cases}$$

En réinjectant dans l'équation (5) et en discrétisant on obtient finalement :

$$v_{i+1} \leqslant v_i + \frac{\|P_{i+1} - P_i\|}{m v_i}\left(\frac{mRC + \alpha J v_i^2}{J + mR^2} - \alpha v_i^2\right)$$